MOL 2025

The 18th Meeting on the Mathematics of Language

Proceedings of the Conference

August 16–17, 2025
Stony Brook University
Stony Brook, NY, USA

Order copies of this and other ACL proceedings from:

Association for Computational Linguistics (ACL)
209 N. Eighth Street
Stroudsburg, PA 18360
USA
Tel: +1-570-476-8006
Fax: +1-570-476-0860
acl@aclweb.org

ISBN 978-1-950737-19-2

Introduction

These are the proceedings of the 18th Meeting on the Mathematics of Language (MOL 2025), held at Stony Brook University on August 16–17, 2025.

The volume contains twelve regular papers, which have been selected from a total of nineteen submissions, using the EasyChair conference management system. The volume also contains one paper contributed by an invited speaker.

The conference benefited from the financial support of Stony Brook's Department of Linguistics, the Institute for Advanced Computational Science (IACS), and the Stony Brook FAHSS fund. We gratefully acknowledge their support.

We would like to express our sincere gratitude to all the reviewers for MOL 2025 and to all the people who helped with the local organization.

Thomas Graf, Matthew Hayden, Jeffrey Heinz, Lawrence Moss, and Yang Wang (editors)
Stony Brook University
August 2025

Program Chairs:

Thomas Graf, Stony Brook University (USA)
Jeffrey Heinz, Stony Brook University (USA)
Lawrence Moss, Indiana University Bloomington (USA)
Yang Wang, University of Utah (USA)

Organizing Chairs:

Thomas Graf, Stony Brook University (USA)
Jeffrey Heinz, Stony Brook University (USA)

Technical Editor of Proceedings:

Matthew Hayden, Stony Brook University (USA)

Program Committee:

Siddharth Bhaskar, University of Southern Denmark (Denmark)
Alexander Clark, University of Gothenburg (Sweden)
Aniello De Santo, University of Utah (USA)
Frank Drewes, Umeå University (Sweden)
Remi Eyraud, CNRS & Jean Monnet University (France)
Robert Frank, Yale University (USA)
Philippe de Groote, INRIA Nancy – Grand Est (France)
Tim Hunter, University of California Los Angeles (USA)
Adam Jardine, Rutgers University (USA)
Greg Kobele, University of Leipzig (Germany)
Andras Kornai, Budapest Institute of Technology (Hungary)
Yusuke Kubota, National Institute for Japanese Language and Linguistics (Japan)
Giorgio Magri, CNRS (France)
Jens Michaelis, University of Bielefeld (Germany)
Gerald Penn, University of Toronto (Canada)
Owen Rambow, Stony Brook University (USA)
Mehrnoosh Sadrzadeh, University College London (UK)
Giorgio Satta, University of Padua (Italy)
Yoad Winter, Utrecht University (Netherlands)

Invited Speakers:

Sophie Hao, Boston University (USA)
Makoto Kanazawa, Hosei University (Japan)
Edward P. Stabler (Canada)
Lena Strobl, Yale University (USA)

Table of Contents

Realizational Morphology in a Modular Minimalist Grammar

Edward P. Stabler
stabler@ucla.edu

Abstract

A modular minimalist grammar with a realizational morphology is briefly motivated and defined, inspired by recent morphosyntax. A modular grammar identifies and isolates components that are relatively causally independent, making components and their interactions easier to understand, without imprecision or approximation. The modular grammar proposed here assumes a realizational morphology in the sense that the atoms of syntax are not pronounced words, and syntax plays a role in word building. These grammars capture generalizations that previous minimalist grammars, and many other generative grammars, miss. Preliminary support is provided for parsing and learnability results.

In Chomsky and Lasnik (1977) and Lasnik (1981), filters are added to a system of generative rules to block certain generated results. One filter blocks dependent affixes from appearing in structures that do not provide a suitable attachment site:

> . . . *Syntactic Structures* makes the claim that there could be another language just like English but where Affix Hopping is optional. The theory we're looking at now . . . makes the claim that there couldn't be any such language.
>
> Affix Hopping and DO-Support . . . describe but don't capture the . . . generalization: *A stranded affix is no good.* (Lasnik, 2000, p.123)

Various versions of this idea have persisted. Bresnan (2000) says, "To counter the fact that DO is ungrammatical elsewhere, there must be a constraint that penalizes its presence." Grimshaw (1997) and Sag et al. (2020) suggest that we need to account for the fact that DO is "necessary whenever it is possible." Stabler (2001) offers a DO-support transduction on morpheme sequences, to supplement a minimalist grammar (MG) with head movement and affix hopping.

One recent thread through this work is considered here, with particular attention to auxiliary verbs. Kayne (1993) says that while some heads may select their complements, "[t]here is no auxiliary selection rule." Bjorkman (2011) says "BE is not directly selected for, but is instead inserted to support inflectional material that was unable to combine with a main verb." Olivier (2025) argues, "HAVE and BE are allomorphs". And Kalin and Weisser (2025) say, "Combining all the evidence. . . the most adequate model. . . " is one that is non-lexicalist (syntactic word-building), post-syntactic (syntactic atoms have no phonology), and realizational (phonology 'realizes' features but not in lexical increments). Here, a very simple, preliminary kind of grammar with those three properties is defined, plausibly connecting them with parsing and learnability results.[1]

0 Some varieties of grammars

To situate this project, it is useful to briefly review some of the formal grammars that have been used for human languages. A context free grammar uses a finite set of rules like 'S $\rightarrow$ NP VP' to rewrite S and other categories repeatedly until there is a string of words. But context free grammars require large numbers of rules to define human languages even very approximately. For example, extracting a context free grammar from the annotations of the Penn Treebank (Marcus et al., 1994), even when done rather carefully, yields a grammar of well over 10,000 rules, not counting the lexical rules. With that many rules, exhaustive parsing of longer sentences from the *Wall Street Journal* becomes infeasible (Charniak et al., 1998). That set of rules is also a very poor grammar, over- and undergenerating badly. Statistics help these grammars to do better in a probabilistic sense, but they are provably unable to capture some patterns found in human languages: crossing dependencies, reduplication,

[1] For a transparent computer implementation and examples: https://github.com/epstabler/mol25

and non-semilinear patterns.[2]

More expressive grammars grammars have been explored, including tree adjoining grammars (Joshi and Schabes, 1997), (parallel) multiple context free grammars (Seki et al., 1991), combinatory categorial grammars (Steedman and Baldridge, 2011), and minimalist grammars (Stabler, 1997).[3] These can in principle be smaller and less ad hoc than context free grammars, since they can define general patterns found in human languages that context free grammars can only approximate by enumerating instances. But, in practice, these more expressive grammars are still large and complex. Take the minimalist grammars proposed by Stabler and Keenan (2003), for example. Those grammars use 5 rules (three cases of 'merge' and two cases of 'move'). But those grammars do not include head movement or affix hopping. To allow those, Stabler (2001) increases the set of rules to 13, and the rules are rather complex. Stanojević (2019) proposes an alternative, more efficient set of 31 rules. But those grammars still do not include rules for coordination, adjunction, agreement, case marking, DO-support, and other things that have been extensively studied in linguistic theory. Adding those things in a principled way is difficult, because each rule gets so complex, and because the set of rules gets larger.

Linguists have proposed, and this paper confirms: refactoring the problem can help. There are relatively independent regularities in the linear order of constituents, in agreement and case, and in patterns of head displacement. But each rule in generative grammars mentioned above defines all of those at once, the way context free grammars do. Those grammars are all 'monostratal' in the sense that each rule application in those grammars builds all aspects of one part of the linguistic structure. An alternative, modular strategy splits apart relatively independent aspects of language structure and defines each aspect separately. Then each piece of structure must satisfy a number of constraints, rather than being defined all at once by a single, complex rule.

Linguists have been exploring grammars one aspect at a time since the beginning. This kind of modularity was the hallmark of Chomsky (1981) and related work. The more recent minimalist tradition, Chomsky (1995) and following work has aimed to unify and consolidate some aspects of language previously regarded as distinct, and has separated others as 'post-syntactic'. But those post-syntactic components still cover significant aspects of language that remain of interest. So the minimalist perspective on language remains modular.

The minimalist grammars of Stabler and Keenan (2003) and related work are not modular. They fall squarely in the monostratal tradition mentioned earlier. Each rule application builds all aspects of a piece of structure. That facilitates relating the grammars to theories of parsing and learning, but it also makes wide coverage grammars complex.

Here, a modular grammar is proposed that generates structures very similar to the early monostratal minimalist grammars, but where the properties of different aspects of each piece of structure are separately defined. A trivially defined infinite set of binary trees is filtered in a sequence of steps, based on relatively distinct aspects of structure, and then transformed by making rather minor adjustments in agreement and morphological features. This architecture captures the monostratally defined minimalist structures and more.

Six modules are proposed. Each one is a bottom-up tree transduction, that is, each builds an output tree as it processes the input bottom-up, from the leaves to the root, node-by-node. The first 2 steps are partial identity transductions, filtering the initial set of trees that can be built from the atoms. The last 4 steps then make minor adjustments for agreement and morphology. The range of the composed grammar is very similar to the range of a rule-based, monostratal minimalist grammar from the 1990's. In fact, as explained below, I conjecture that the definable string languages are exactly those of the earlier minimalist grammars. But the modular grammars proposed here capture new structures and new generalizations. Instead of many rules for head movement, for example, there is one simpler head movement function with greater expressive power. It is simpler because it can be stated separately from everything else.[4]

[2] Shieber (1985); Culy (1985); Michaelis and Kracht (1997); Kobele (2006).

[3] I have left 'constraint based' grammars out of this list, since they rest on rather different formal foundations (Johnson, 1994; Pullum, 2013), but they do have some properties in common with the modular generative grammars that will be the focus here. Our modularization is distinctively Chomskian, and can be formalized either generatively in terms of tree transducers or in terms of constraints (Graf, 2013).

[4] Hornstein (2024, pp.7-8) proposes "All grammatical relations are merge-mediated" as the "Fundamental Principle of Grammar" (FPG). That FPG sounds like the approach formulated here, since after our merge function (mrg) builds/accepts the initial trees, the following modules either filter or make

1 A modular grammar

The grammars defined here will each be compositions of 6 functions:

$$\mathrm{g} = \mathrm{vi} \circ \mathrm{lin} \circ \mathrm{hm} \circ \mathrm{agr} \circ \mathrm{sel} \circ \mathrm{mrg},$$

where ∘ is standard function composition (often pronounced ‘after’) and where: mrg builds/accepts binary trees over a set of syntactic atoms; sel maps those trees to themselves if selection features match appropriately; agr matches and instantiates agreement features; hm applies head movement to adjust morphological contents at the leaves (with no change in hierarchical structure); lin maps each tree to its pronounced linear order (with no change in hierarchical structure); and vi, vocabulary insertion, replaces morphological indices with phonologically specified morphs (with no change in hierarchical structure). These 6 functions are defined in the following sections. The composed grammar g is a function from (unpronounceable) binary trees to (hierarchically unchanged but reordered, pronounceable) binary trees. Language variation is attributed to (i) the atoms over which mrg operates and (ii) the rules of vi.

2 mrg

Collins (2002), Chomsky (2007, p.8) and others propose an operation merge which forms binary sets over a lexicon. Here, for the definition of composed grammars, it is convenient to let mrg be the tree transducer which accepts (all and only) ordered binary trees with atoms at the leaves, mapping each such tree to itself. The linear ordering of the tree is inessential, but computationally useful, so we let the range of mrg be the ordered binary trees with lexical items at the leaves and no labels on internal nodes.[5] Lexical items are included as trees with just one node.

We assume that the lexicon is a set of triples, (root, sel-fs, agr-fs), where each root can be thought of as a morphological index that could underlie multiple pronunciations, sel-fs specifies selection features, and agr-fs specifies agreement features. The feature complexes are defined below.

3 sel

As noted by Collins (2002) and others, lexically specified selection requirements seem unavoidable.[6] The function sel accepts only those elements of the range of mrg that satisfy selection requirements, mapping those trees to themselves. Here, we use a notational variant of the selection checking from Kobele (2021), not too different from Stabler (1997). Selection requirements sel-fs are given by a formula fs⊸fs’ where fs and fs’ are (possibly empty) feature sequences (non-commutative conjunctions), where the features in the antecedent fs are *negative* and those in the consequent fs’ are *positive*.[7] If fs is empty, the antecedent is omitted and fs’ is simply written as a dot-separated sequence.

At each internal node, sel is defined only if a feature is ‘checked’. Selection features are checked bottom-up as follows. At each leaf, the features sel-fs are given by the syntactic atom. Then, moving up through the tree, each internal node has the features (some of which may be checked) at the leaf which is at the end of its left branch, its head. At any internal node with left subtree x and right subtree y, if x has a subtree x’ whose first unchecked feature is a positive f, then tree x’ must be identical to y (disregarding features checked), those features f in x and x’ are checked, and features of y are calculated from x’. This is called *internal merge* or *move*. If x has no subtree x’ whose first unchecked feature is a positive f, then the first unchecked feature of y must be positive f, and both of those features are checked, written $\not{f}$. This is *external merge*.[8] Feature checking is computed and passed upward in the ‘state’ of the transduction, without modifying the tree and its features in any way. Schematically:

non-syntactic modifications in them. But Hornstein intends something stricter, suggesting that, in his sense, there can be no modularity in the ‘faculty of language’, though there may of course be ‘interface’ operations.

[5] This function is definable by a bottom-up tree transducer (Baker, 1979). A head-first order is enforced in §3 and used at a number of points, but is just a convenience. §3 shows it could be replaced in a set-based computation, identifying the head as the unique child with unchecked ‘negative’ features. To keep all derivable syntactic objects in the domain of mrg, we allow, at the leaves, any of finitely many sequences of roots built by head movement, and any of finitely many vocabulary items that can replace the roots, as explained below.

[6] But is selection a single, uniform process? And what is selected, exactly? Many issues remain (Kalin and Rolle, 2024; Wurmbrand, 2014; Hirose, 2011; Pollock, 1989).

[7] The ⊸ is from linear logic (Girard, 1995, p.15).

[8] Some linguists have instead assumed a merge-over-move preference (Epstein et al., 2012; Chomsky, 2000, p.106), but there are empirical problems with those proposals (Shima, 2000; Castillo et al., 2009; Abels, 2012, §4.3.1). And it is sometimes proposed that the choice of merge and move is determined by a context-relative optimization (Heck and Müller, 2016). Any of these proposals could be deployed here.

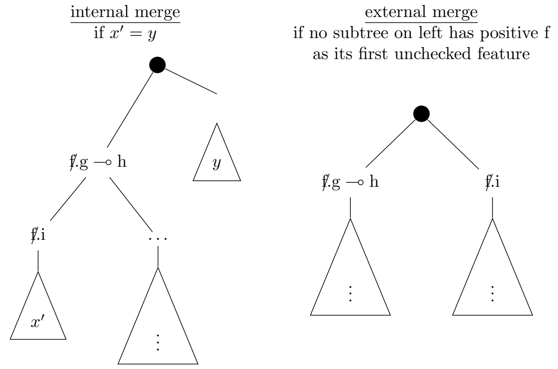


Figure 1 shows an example.

Stabler (1997, 2001) restricts feature checking with a *shortest move constraint*:

(smc) Neither internal nor external merge is possible if it creates a tree in which the first unchecked features of two distinct subtrees are positive and identical (where subtrees related by internal merge are not counted as relevantly distinct).

This blocks analyses in which a single clause has multiple wh phrases extracted by internal merge from a single clause, as in

(1) *What does who say [(who) read (what)]?

(2) *Who says what [(who) read (what)]?

The smc is simple and well-studied.[9] But evidence supports more complex locality conditions, conditions that are still poorly understood.[10]

4 agr

The function agr accepts only those elements of the range of sel that satisfy agreement requirements, mapping those trees to themselves. Following Béjar and Rezac (2009) and others, we assume this check on agreement is 'cyclic' in the sense that it is calculated bottom-up, at each node on paths from leaves to the root.

Adapting Hanson (2025), search is restricted to the first 'd-commanded' element on a specified tier, where a head *d-commands* the heads of its externally merged specifiers from highest to lowest, then the head of its complement, and then whatever the complement head d-commands – in that order.[11] A *tier* $t = \{s_1, \ldots, s_n\}$ is a set of sets of features. A d-commanded *head* h *is on tier* t if and only if some s_i is a subset of the features of h, where those include not only its agr features but also its positive selection features. Recall that atoms are triples (root,sel-fs,agr-fs). Each agr-fs is a set of (tier,feature) probes and feature goals. Probe f looks for a goal in the first d-commanded element on the specified tier, where f is feature with a type separated by :. An unspecified 'probe' ϕ:_ is instantiated by a 'goal' of the same type, a feature like ϕ:3s, in the tier-adjacent d-commanded element.[12]

Developing a suggestion from Chomsky (2000, 2001), Deal (2025b) also argues that agreement affects not only the probe by instantiating, 'valuing' it, but also the goal, 'flagging' it. We adopt that idea here, letting probe ϕ:_ valued by goal ϕ:3s result in ϕ:3s on the probe and ϕ:$\underline{3s}$ on the goal. With this mechanism, we can handle what looked like exceptions to a uniformly downward-looking mechanism.[13] Those exceptions become expected – as occasional morphological realizations of goal-flagging in vocabulary insertion. See Figure 3.[14]

[9] The smc allows feature checking to be computed by a finite state multi bottom up tree transduction. See Kobele et al. (2007); Graf (2023a); Hunter and Frank (2021); Kanazawa (2016, Ex. 5.8); Salvati (2011); Kobele et al. (2007); Michaelis (1998); Stabler (1997, 2011).

[10] Rizzi (2013, 2017); Hein (2024); Major and Torrence (2024); Schurr et al. (2024); Fernández-Serrano (2025); Branan and Erlewine (2025). Note that agr, defined in §4, has a locality condition built in, a kind of relativized minimality. Deal (2025a), a.o. argues that agr locality, when properly formulated, also controls internal merge. There are also arguments that internal merge can target heads rather than just their maximal projections. See Harizanov (2019) and references cited there. These matters are left for future work.

[11] The hypothesis that locality in agreement and selection derives from search has a long history (Chomsky, 2005; Epstein et al., 2020; Chow, 2022; Ke, 2024; Branan and Erlewine, 2025). Syntactic search on tiers is more recent (Graf, 2022, 2023b). In this first sketch of a modular MG, we leave a more careful consideration of search options to future work.

[12] When the lexicon is seen as associating words with structures, it may seem obvious that disjunctive and dependent types are needed, adding significantly to agreement complexity (Shieber, 1986; Johnson, 1988; Eisele and Dörre, 1988; Maxwell and Kaplan, 1989; Dörre and Eisele, 1990; Carpenter, 1992; Francez and Wintner, 2012). But that ambiguity is morphophonological, indicating syntactic composition only indirectly. In this syntax, record types suffice.

[13] These assumptions also make agr and sel feature checking strikingly similar, inviting a unification.

[14] Plausibly, Ermolaeva and Kobele's (2022) argument for the MG-definability of instantiated structures can be adapted to modular grammars with our agr, but further exploration of agr is beyond the scope of this paper. Deal (2025b, Appendix) provides a more elaborate agreement algorithm that allows properties of a probe to change in the course of a derivation (Deal, 2024; Keine, 2019, 2020). Sometimes case is also treated as agreement, but this is controversial. Deal (2025a) and Kidwai (2023) argue, for example, that case should be a head rather than a feature.

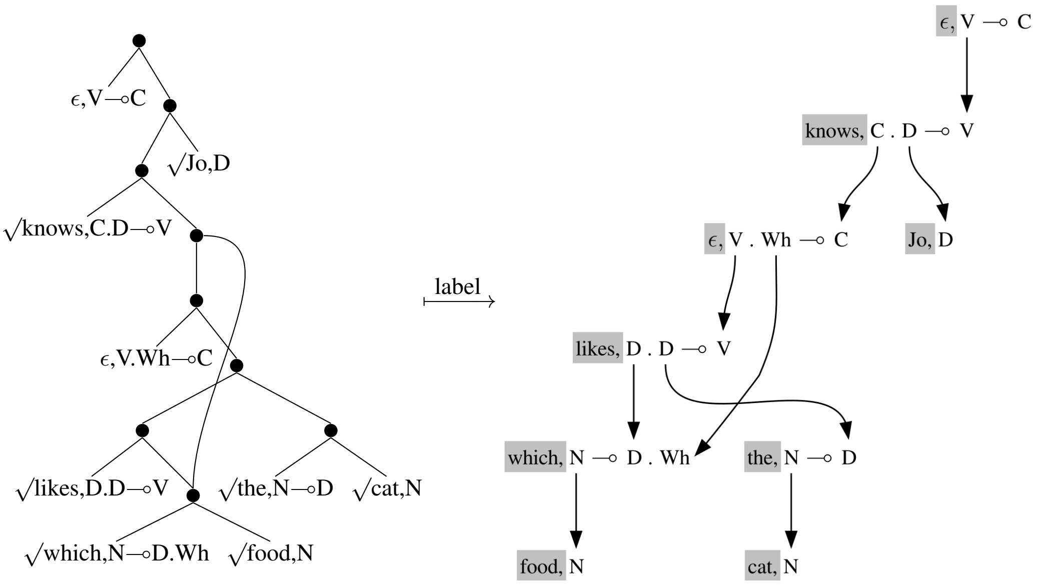


Figure 1: On the left, a tree for *Jo knows which food the cat likes* (showing roots and sel-fs). The function sel checks that a pair of features is checked at each internal node, as described in §3. To show where internal merge (movement) has applied in these graphs, the two identical subtrees for *which food* are shown as one node, but sel applies to the binary tree. At each internal node •, sel requires that the left child is the head, the node with unchecked negative features. The checking relations are made explicit in the corresponding checking graph on the right. Each checking arrow goes from a negative occurrence of a feature to a positive occurrence of that same feature, and each feature is involved in at most one such pairing. In this structure, one feature in the tree remains unchecked: an unchecked, positive C labels the root. Since one pair of features is checked at each node (and smc is respected), as required, sel maps the tree on the left to itself.

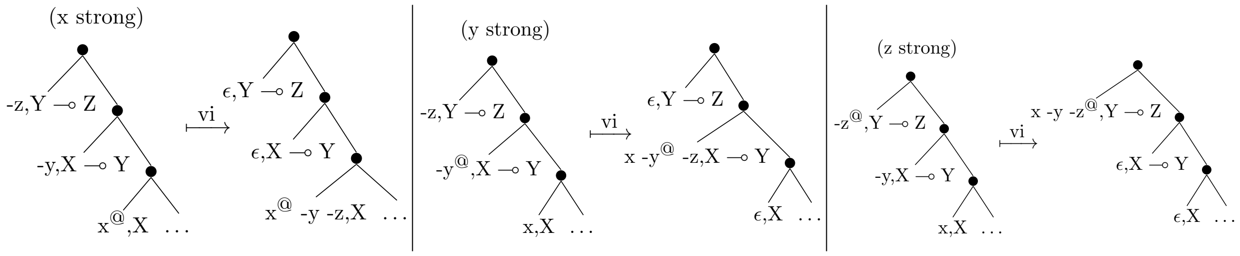


Figure 2: As described in §5, hm acts on three cases with span -z, -y, x. First, the morphs x and -y raise to left-concatenate to -z, in mirror order. Then the complex is placed in the highest strong @ position, leaving the other morph positions empty.

5 hm

Considering the contrast,

(3) Which cat (which cat) chase -s the rat

(4) Which rat (*do) -s the cat chase (which rat)

some have argued that DO is required in (4) because tense is blocked from uniting with the verb by the overt subject *the cat* between the heads T and C in the linearized tree..[15] But this is at best a weak argument for the relevance of linear adjacency in head movement. Since moved elements are featurally distinct, the position of the subject is featurally determined. So while we will agree with the proposal of Chomsky et al. (2023, p.66) and many others that head movement is post-syntactic, distinct from phrase construction, we reject the idea that this operation must apply after linearization, after lin.[16]

Adopting this hypothesis, we formulate a version of head movement inspired by Arregi and Pietraszko (2021). Let a *span* be a sequence of $h_1, \ldots, h_n$ $(n \geq 2)$ where (i) for $i < n$, h_i selects the projection of h_{i+1}, (ii) for $i < n$, h_i has a root marked as morphologically *dependent* with a dash, -$\sqrt{}$root, (iii) h_n is not marked dependent, and (iv) the sequence is maximal in the sense that h_1 is not the first merge of a higher dependent-marked head. We also require that there is no recursion in spans: no two heads in a span have the same first positive selection feature.

Any head in a span may also be marked *strong*, indicated here by following the root with @.[17] If no head in the span is marked strong, then h_1 is the strongest. Otherwise, the strongest head is the first in the span that is marked @. Bottom-up, the morph of each dependent head raises and left-adjoins a selected independent head (or head complex), and the complex is pronounced at the strongest position. Representing the left-adjunction of morphs by left-concatenation (of roots with their features), this defines the patterns of Figure 2, where the derived order x, y, z of morphs mirrors the syntactic projections Z, Y, X. In the figure, note that in the (x strong) case, both y and x are lowered. In (y strong), x is lowered and z is raised.[18]

Previous minimalist grammars (Stabler, 2001) formalize English subject-auxiliary inversion with a pattern like the (z strong) case of Figure 2, with the complex (verb$^@$ -tense -C$^@$) pronounced in strong question-forming -C$^@$. And English affix-hopping is similar to lowering -tense onto a verb$^@$ with a weak C, like the (x strong) case. See Figure 3.

6 lin

Adapting an idea from Kayne (1994, 2020), Chomsky (1995), Cinque (2023), linear order is adjusted:

(ord) at any node with daughters x, y where head x itself has daughters, reorder to y, x.

And setting the stage for vi, we strip features that are redundant at the interface:[19]

(del) Delete non-final moving elements.

This deletion replaces non-final moving elements, i.e. elements that arguments to an internal merge step, by an empty structure. Then we define: lin = ord ∘ del. See Figure 4.

With del, it is important to remember why internal merge produces structures with identical subconstituents in the first place. Why have those elements in the structure if they are not going to be pronounced? In the current formulation, that question stands out because sel uses an explicit check on whether two subconstituents of unbounded size are identical. And similarly, in tree transducer implementations, transducers with unbounded copying are required (Kobele et al., 2007). With del, why formulate internal merge that way?

Support for internal merge with copying comes from apparent evidence that copies are really there. There is a large literature, but one kind of supporting argument comes from evidence that, in some constructions, non-final copies are sometimes not completely deleted. Yuan (2025) notes, for example, that a number of analyses in the literature claim that deletion of nonfinal occurrences of a fronted

[15] Compare Pollock (1989, §5.5.4), Chomsky (1991, §2.3.1), Bobaljik (1995, §2.1), Sportiche (1998, §5.2.3.1), Embick and Noyer (2001), Bjorkman (2011), Arregi and Pietraszko (2021).

[16] In agreement with, e.g. Branigan (2023, §2.1).

[17] Brody (1997), Svenonius (2016), a.o. also use @. Arregi and Pietraszko (2021) use *.

[18] Cf. Brody (1997, 2000); Adger et al. (2009); Svenonius (2016); Harizanov and Gribanova (2019); Branigan (2023); Giannoula (2025). Kobele (2002) shows how the mirror theory grammars proposed by Brody (1997) are weakly equivalent to MGs. I conjecture that Kobele's result extends to our our similar composed grammars with hm.

[19] Cf. Chomsky et al. (2023, p.26): "An efficiency consideration at [the sensorimotor interface SM] is to pronounce just one (the highest) of a set of copies." But see the discussion of Yuan (2025) a.o. below.

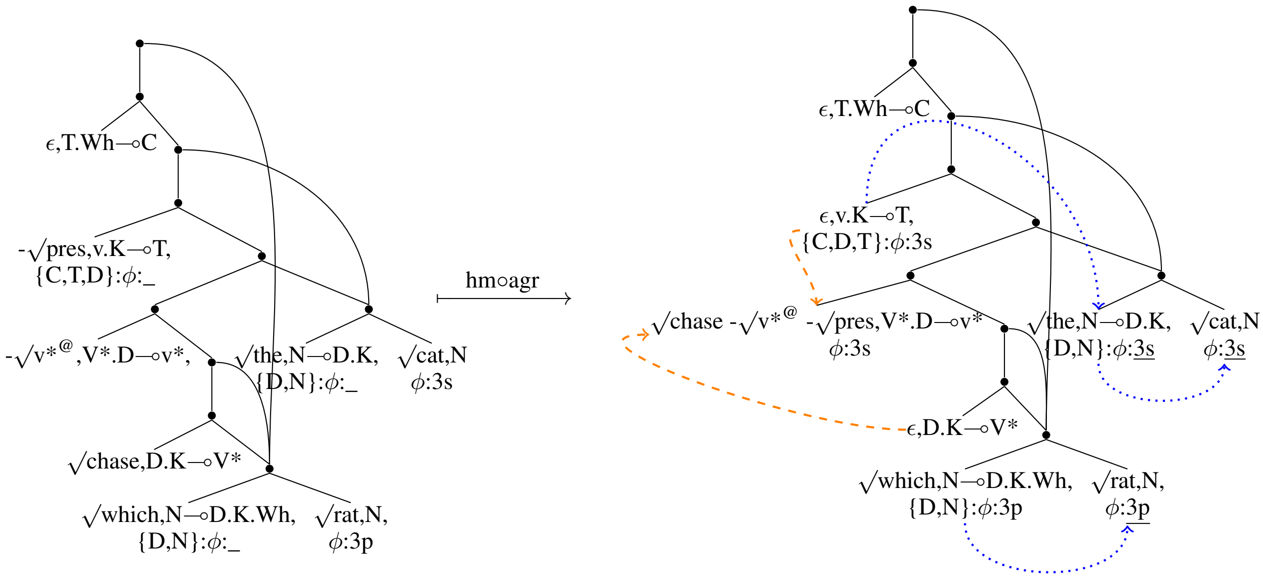


Figure 3: A structure for the embedded clause of *I know [which rat-s the cat chase-s]*. Note that clause structure is elaborated slightly beyond what Figure 1 depicts and atoms are shown with roots, sel-fs and agr-fs. As described in §4, agreement instantiates probe features bottom-up with adjacent elements in spinal d-command sequences on the indicated tier. The dotted arrows show how ϕ:_ probes from D to be valued by its d-commanded N; T probes to find its ϕ value in the d-commanded specifier D. Then, as described in §5, hm creates the head complex V* -$\sqrt{v}^{*@}$ -T as shown by the dashed arrows, putting that complex into the strong v* position. Trees like this that show phrasal and head movements with sel and agr features are complex, but each module is relatively simple.

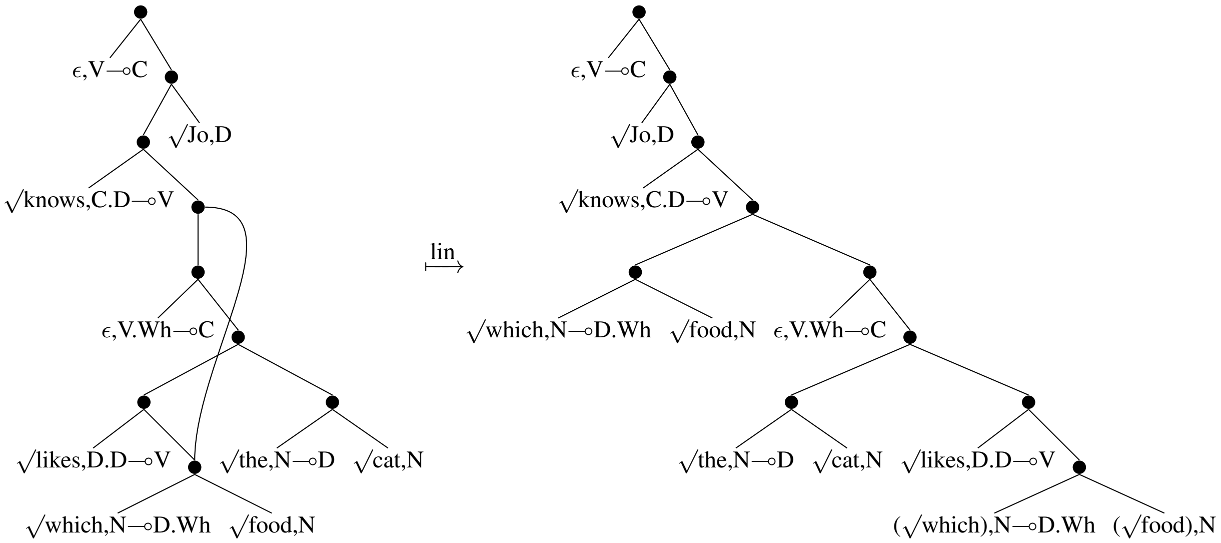


Figure 4: Left, repeated from Figure 1. Right, as described in §6, the result of applying lin. As is often done, instead of emptying non-final copies, those copies are shown with their morphs in parentheses.

verb is blocked when that would leave a tense affix without a host. She then observes a similar pattern in Inuktitut, where deletion of non-final occurrences is blocked when it would strand affixal verbs requiring a DP host. These patterns are inconsistent with the simple deletion process of del.[20] Allowing del to completely remove copies before pronunciation serves as a placeholder for future work.

7 vi

Function vi replaces roots with phonologically instantiated forms, based on syntactic context. For example, the rules

$$\begin{array}{rcl} \surd\text{cat} & \rightarrow & \text{cat} \\ \phi\text{:3s} & \rightarrow & \text{-s} \end{array}$$

add phonological content to roots in the leaves of the syntactic tree. Representing the phonologically instantiated roots are indicated with standard orthography, the root $\surd$cat in a leaf with ϕ:3s is replaced by phonologically instantiated forms:

$$(\surd\text{cat,D,}\phi\text{:3s}) \quad \Rightarrow \quad (\text{cat -s,D,}\phi\text{:3s}).$$

Irregular plurals are handled with special rules:

$$\surd\text{mouse } \phi\text{:3p} \rightarrow \text{mice.}$$

Special rules like this take precedence over the defaults, 'blocking' them, because they are 'more specific' in the sense that they specify more features (Halle and Marantz, 1993; Embick and Marantz, 2008). With this specificity order defined on our finite set of vi rules, we use the first applicable rule.

The domain of this kind of competition among vi rules was widely held to be a single atom, when atoms were morphemes. But as syntactic principles have entered into word building, features that used to be included among those associated with morphemes are now often split away, treated as syntactic heads. When heads are combined by head movement, these features may be reunited. For example, with the rules

$$\begin{array}{rcl} \surd\text{chase} & \rightarrow & \text{chase} \\ \surd\text{past} & \rightarrow & \text{-ed,} \end{array}$$

we derive a pronounced form for this complex head from Figure 3, mapping each element pointwise, but preferring rules with the largest numbers of features:

$$(\surd\text{chase } \surd\text{past } \epsilon) \Rightarrow \text{chase -ed.}$$

Adapting ideas from Svenonius (2016) and Haugen and Siddiqi (2016), vi rules can also target spans, where a span is a sequence of heads, each selecting the next, as in hm, but the heads need not be morphologically dependent (marked by -). We replace conditions (ii) and (iii) in the previous definition of *span* in §5 with: (ii') $h_1, \ldots, h_n$ are in the domain of a vi rule. As in hm, we let the highest head be the default strongest. So then, following Haugen and Siddiqi (2016, (7)) we allow this vi rule to derive the Spanish portmanteau *del* from a span that includes a preposition and determiner, indicating the boundaries of syntactic heads with parentheses:

$$(\surd\text{de})\ (\surd\text{el}) \quad \rightarrow \quad \text{del.}$$

This rule replaces ($\surd$de) with *del* in the preposition, and removes root ($\surd$el) from the determiner.

Much is still unsettled in morphology, but this sketch incorporates some of the represented ideas, to be fleshed out in future work. Note that mirror order is not predicted by these vi operations unless operating on a complex formed by hm, and vi also differs from sel in letting rule choice be subject to blocking.

8 Examples

8.1 Nominalization

In standard presentations of logic, each predicate has an arity, but in human languages, a single pronounced form of of a verb like *capture* allows various numbers of arities and can also be a noun with the same arguments, as in *the capture (of markets) (by oligarchs)*. The morph is associated with a kind of event, allowing its various arguments to be expressed or not in various contexts. The verb *destroy* on the other hand has a distinct a nominalized form *destruction*. Rather than treating such coincidences between nominal and verbal forms as accidental, Chomsky (1970) suggests a common underlying form may be mapped to different pronunciations. Here, vi plays that role using the rules,

$$\begin{array}{rcl} (\surd\text{capture}) & \rightarrow & \text{capture} \\ (\surd\text{destroy V}) & \rightarrow & \text{destroy} \\ (\surd\text{destroy N}) & \rightarrow & \text{destruction.} \end{array}$$

If -$\surd$-tion is a nominalizing head of some kind, then head movement forms a complex to which a very similar vi rule can apply.[21]

[20] Compare also Georgi (2017).

[21] See e.g. Alexiadou and Borer (2020) and references cited there for recent discussion of Chomsky (1970) and recent variants of that kind of proposal.

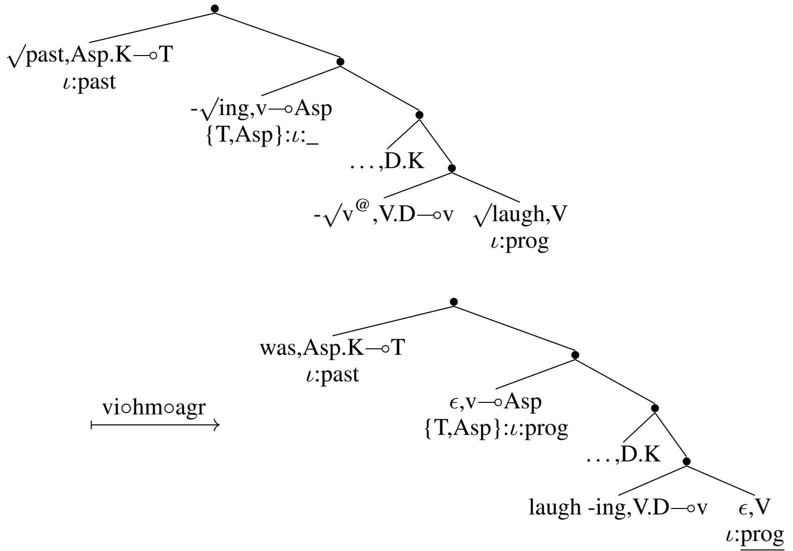


Figure 5: Insertion of English auxiliary BE by vi. An inflectional probe ι:_ on V can be instantiated by ι:past in simple clauses that lack Asp. But here, instantiating bottom-up, ι:prog on Asp values the feature on V, and so valuation by ι:past is not possible. The function vi rescues the structure by inserting √BE and mapping √BE+√past to *was*. The complex placed in the strong v* position by head movement, √laugh √v* √ing, is mapped by vi to *laugh-ing*. Compare e.g. Bjorkman (2011, §2.3.4.1).

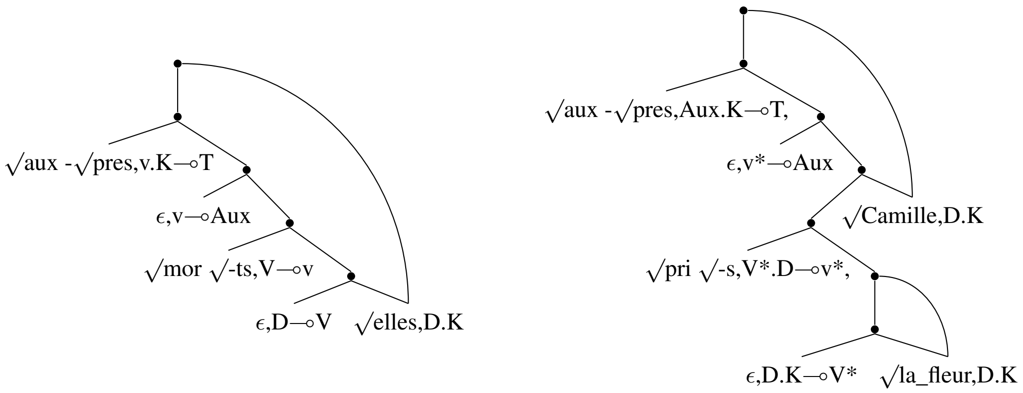


Figure 6: French auxiliary BE and HAVE as allomorphs. In unaccusative on the left, hm raises V √mor to v √-ts and raises Aux √aux to T √pres. With the simple vi rules in the text, Aux is realized as BE because it is not adjacent to v* on the {T,v*,v} tier, and so we get *Elles sont mor-ts* ('they died'). On the right, hm raises V √pri to v* √-s and Aux √aux to T √pres. Aux with v* adjacent on the tier {T,v*,v} is realized as HAVE, to yield *Camille a pri-s la fleur* ('Camille took the flower'). Compare Olivier (2025, (37),(38)), Bjorkman (2011, §2.4.4.1,(90),(91)).

8.2 English auxiliaries.

While the specification of various pronounced forms of $\sqrt{}$destroy could be entirely a lexical matter, agr, hm and vi get involved in determining how other words are built and pronounced. For example, pronouncing BE requires attention to features:

> Suppose the PF form of a lexical entry is completely unpredictable: the English copula, for example. In this case the lexical coding will provide whatever information the phonological rules need to assign a form to the structure [copula, {F}], where {F} is some set of formal features (tense, person, etc.). (Chomsky, 1995, §4.2.2)

This is exactly our strategy. But unlike Chomsky, we generalize it not just to certain auxiliaries but through the whole vocabulary.

In English, the auxiliary verbs HAVE and BE regularly combine with past and progressive participles, respectively. But in various languages, the appropriate auxiliary for a past participle depends on a number of factors that vary across dialects. Bjorkman (2011) argues that auxiliaries appear not when they are selected but when they are needed to rescue a kind of 'overflow' situation.[22] When a context does not provide a way to express tense on the verb, it can 'spill over' into another form, 'rescuing' the structure. She notes that in Arabic and in the Bantu language Kinande, verbs cannot have both a tense and an aspect marker, so when the main verb has aspect, tense gets expressed on an auxiliary. She argues that English falls into this pattern too, with an analysis that is easily modelled in our grammars, as shown in Figure 5.[23]

8.3 French auxiliaries

In Standard European French passé composé constructions, some verbs require a HAVE auxiliary, while others require BE. One rough approximation is that transitives and unergatives take HAVE, while unaccusatives take BE. Bjorkman (2011) proposes that the presence of an object between a perfective head and the verb blocks agreement between them, yielding HAVE instead BE. Assuming that transitive feature v* is distinct from v, in this simple case, the contrast can be decided by testing whether v* is adjacent on the {T,v*,v} tier:

(Aux) → HAVE if v* is {T, v*,v}-adjacent
(Aux) → BE

See Figure 6.

Olivier (2025) observes that this simple rule conflicts with the fact that reflexive verbs with direct objects require HAVE. He suggests instead that an Aux head intervenes between T and v_{Prt}, and that Aux is realized as HAVE if person features on T and Aux are not guaranteed to be identical, and otherwise as BE. This gives the basic unaccusative/transitive contrast if Aux always agrees with the internal argument, while T agrees with the closest element on the {C,T,D} tier. But we do not want the realization rules to have to test whether identities are guaranteed. The alternative is to test for the conditions of the guarantee. A fuller integration of Olivier (2025) and similar approaches is left for future work.

9 Conclusions

MG properties adjusted. Previous minimalist grammars have been monostratal, but simplicity and flexibility are enhanced in modular formulation, making understanding and experimentation with alternatives easier. Here, syntactic atoms with unpronounceable roots define the domain of mrg, while agreement, head-movement, linearization, and vocabulary insertion are 'post-syntactic' in the sense of applying after mrg.

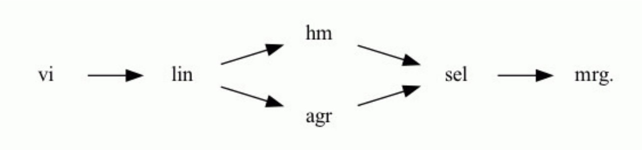
Function vi uses a finite set of (language-specific) rules, applied in an order of decreasing specificity. Defaults get listed as the last case. This kind of competition among alternatives amounts to providing defaults with a kind of negative condition – something that rule-based minimalist grammars do not offer. But since these rules are finite in number and scope, the expressive power of the formalism remains restricted.[24]

We can diagram dependencies among the six modules above with a graph of a sort used in software construction, where each module has those it depends on as its descendants:

[22] There are many rescue analyses for auxiliaries (Embick, 2000; Arregi and Klecha, 2015; Fenger, 2019; Cruschina and Calabrese, 2021). An empirical defense of these is beyond the scope of this paper, where we aim only to explore some computational properties. And see e.g. Pietraszko (2023) for a possible non-realizational account of overflow patterns.

[23] A naive DO-support could be implemented with a vi rule (T → DO). But Bjorkman (2011) argues that a broader consideration of *do*-like patterns disconfirms rescue analyses. So we leave this for future work.

[24] It is interesting to compare negation-as-failure and other extensions to Horn clauses in logic programming and type class definitions (Miller, 2022; Bottu et al., 2017). Kanazawa (2007, 2009, 2017) observes that because MCFGs can be elegantly expressed in definite Horn clauses, they are easily parsed in Datalog. Our modular grammars cannot be directly expressed in definite Horn clauses, but as noted just below, we conjecture that these grammars are weakly equivalent to MCFGs.



The function sel relies on the binary structure of mrg. Functions agr and hm rely on the head-complement and internal/external-merge distinctions enforced by sel. The function lin reorders heads and complements, so agr and hm can be slightly simpler if lin applies after them.[25] These consequential generalizations about dependencies among modules – i.e. among aspects of current morphosyntactic theory – are much more explicit in modular grammars than in monostratal theories.

Ongoing research is focused on whether we have properly understood the modules and dependencies. In particular, as research cited above shows, considerable efforts can be regarded as focused on uniting sel and agr in a more general labeling theory, and on uniting hm and vi in a general interface theory.

MG properties preserved: Two conjectures. I conjecture that, if we assume the smc of §3 and the del of §6, the string languages definable by our modular minimalist grammars are definable by rule-based, monostratal minimalist grammars and so, as shown by Michaelis (1998), also by multiple context free grammars (MCFGs). Some support for this conjecture is provided by the discussion and notes above.[26] We know how to parse MCFGs and hence also MGs (Seki et al., 1991; Kallmeyer, 2010; Harkema, 2001), and we have some first, definite ideas about how they can be learned (Clark and Yoshinaka, 2016; Yoshinaka and Clark, 2012).

But is this conjecture with its consequences interesting, when it depends on smc and del, which are clearly not right? I think the conjecture *is* interesting because, plausibly, well-understood computational approaches to MCFGs will extend easily to the better theories that improve on smc and del. This is my second, more speculative conjecture. One bit of evidence for it, relevant to del, comes from the easy extension of MCFGs to 'parallel' MCFGs with unbounded copying, with parsing (Seki et al., 1991) and learning results (Clark and Yoshinaka, 2014). The copying noted by Yuan (2025) and others is very restricted, perhaps shaped in large part by performance factors. Structures like that should not exact the same price in parsing and learning efficiency as parallel MCFGs impose.

Another piece of evidence for the second conjecture, evidence relevant to the smc, comes from the fact that the smc itself is not integrated into the foundations of the grammar. It is a separately stipulated condition. When we find better, empirically well-supported locality conditions, we may be able to impose those in a similar way, without significantly disrupting the basic structure our grammars and the empirical domains they cover. And on the small structures relevant in human language processing, again, less restrictive locality conditions may be compatible with feasible models of human parsing and learning.

New capture of prominent generalizations. With roots in syntactic atoms, we can model something like Chomsky's (1970) nominalization. With vocabulary insertion, we can model something like the Halle and Marantz (1993) competition-based allomorphy. And we can capture something similar to the 'overflow' analyses of auxiliaries proposed by Bjorkman (2011), Olivier (2025) and others. Like earlier DO-support analyses, these involve 'last-resort' rules which are, we conjecture, weakly equivalent to rule-based MGs but more succinct. Rule-based MGs miss those generalizations, but can enumerate the relevant cases.[27]

Acknowledgements

For discussions of earlier versions of this work, thanks to audiences at the Stony Brook University MoL and MG+ meetings, especially Meaghan Fowlie, Thomas Graf, John Hale, and Kristine Yu.

References

Klaus Abels. 2012. *Phases: An Essay on Cyclicity in Syntax*. Linguistische Arbeiten.

David Adger, Daniel Harbour, and Laurel Watkins. 2009. *Mirrors and Microparameters*. Cambridge University Press.

[25] But note that lin ordering arguments are weak, since our Kayne-ian linearization of 2 daughters is only 1 bit of information – so that's 1 bit of information per internal node – and it is efficiently computable.

[26] See fns. 5, 9, 14, 18. A compiler that takes modular grammars (i.e. finite sets of atoms and vi rules) to equivalent MCFGs is also in preparation, adding to the collection of similar compilers for previous mildly context sensitive grammars (Guillaumin, 2004).

[27] Note that we do not really capture the stray affix filter, as formulated by Lasnik in the introduction, but if stray affixes are morphs that cannot occur independently, it is not clear what there is to capture beyond the overflow analyses that rescue certain structures by providing those affixes with hosts.

Artemis Alexiadou and Hagit Borer. 2020. Introduction. In A. Alexiadou and H. Borer, editors, *Nominalization*, pages 1–23. Oxford University Press.

Karlos Arregi and Peter Klecha. 2015. The morphosemantics of english past tense. *Proceedings of North Eastern Linguistic Society*, 45:53–66.

Karlos Arregi and Asia Pietraszko. 2021. The ups and downs of head displacement. *Linguistic Inquiry*, 52:241–289.

Brenda Baker. 1979. Composition of top-down and bottom-up tree transductions. *Information and Control*, 41:186–213.

Susana Béjar and Milan Rezac. 2009. Cyclic agree. *Linguistic Inquiry*, 40:35–73.

Bronwyn Bjorkman. 2011. *BE-ing Default: The Morphosyntax of Auxiliaries*. Ph.D. thesis, MIT.

Jonathan David Bobaljik. 1995. *Morphosyntax: The syntax of verbal inflection*. Ph.D. thesis, MIT.

Gert-Jan Bottu, Georgios Karachalias, Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Quantified class constraints. *ACM SIGPLAN Notices*, pages 148–161.

Kenyon Branan and Michael Yoshitaka Erlewine. 2025. Locality and (minimal) search. In K.K. Grohmann and E. Leivada, editors, *Cambridge Handbook of the Minimalist Program*. Cambridge University Press.

Phil Branigan. 2023. *The Grammar of Multiple Head Movement*. Oxford University Press.

Joan Bresnan. 2000. Optimal syntax. In J. Dekkers, F. van der Leeuw, and J. van de Weijer, editors, *Optimality Theory: Phonology, Syntax, and Acquisition*, pages 335–385. Oxford University Press.

Michael Brody. 1997. Mirror theory. *UCL Working Papers in Linguistics*, 9.

Michael Brody. 2000. Mirror theory: Syntactic representation in perfect syntax. *Linguistic Inquiry*, 31:29–56.

Bob Carpenter. 1992. *The Logic of Typed Feature Structures*. Cambridge University Press.

Juan Carlos Castillo, John E. Drury, and Kleanthes Grohmann. 2009. Merge over move and the extended projection principle. *Iberia*, 1:53–114.

Eugene Charniak, Sharon Goldwater, and Mark Johnson. 1998. Edge-based best-first chart parsing. *Proceedings of Workshop on Very Large Corpora*, 6:127–133.

Noam Chomsky. 1970. Remarks on nominalization. In N. Chomsky, editor, *Studies on Semantics in Generative Grammar*, pages 1–61. De Gruyter.

Noam Chomsky. 1981. *Lectures on Government and Binding*. Foris, Dordrecht.

Noam Chomsky. 1991. Some notes on economy of derivation and representation. In R. Freidin, editor, *Principles and Parameters in Comparative Grammar*, page 417–454. MIT Press. Reprinted in Chomsky 1995.

Noam Chomsky. 1995. *The Minimalist Program*. MIT Press.

Noam Chomsky. 2000. Minimalist inquiries: The framework. In R. Martin, D. Michaels, and J. Uriagereka, editors, *Step by Step*, pages 89–155. MIT Press. https://isbnsearch.org/isbn/9780262133616.

Noam Chomsky. 2001. Derivation by phase. In M. Kenstowicz, editor, *Ken Hale: A Life in Language*, pages 1–52. MIT Press.

Noam Chomsky. 2005. Three factors in language design. *Linguistic Inquiry*, 36:1–22.

Noam Chomsky. 2007. Approaching UG from below. In U. Sauerland and H.-M. Gärtner, editors, *Interfaces + Recursion = Language?*, pages 1–30. de Gruyter.

Noam Chomsky and Howard Lasnik. 1977. Filters and control. *Linguistic Inquiry*, 8:425–504.

Noam Chomsky, T. Daniel Seely, Robert C. Berwick, Sandiway Fong, M.A.C. Huybregts, Hisatsugu Kitahara, Andrew McInnerney, and Yushi Sugimoto. 2023. *Merge and the Strong Minimalist Thesis*. Cambridge University Press.

Keng Ji Chow. 2022. A novel algorithm for minimal search. *Snippets*, 42:3–5.

Guglielmo Cinque. 2023. *On Linearization: Toward a Restrictive Theory*. MIT Press.

Alexander Clark and Ryo Yoshinaka. 2014. Distributional learning of parallel multiple context-free grammars. *Machine Learning*, 96:5–31.

Alexander Clark and Ryo Yoshinaka. 2016. Distributional learning of context-free and multiple context-free grammars. In J. Heinz and J.M. Sempere, editors, *Topics in Grammatical Inference*, pages 143–172. Springer.

Chris Collins. 2002. Eliminating labels. In S.D. Epstein and T.D. Seely, editors, *Derivation and Explanation*, pages 42–64. Blackwell.

Silvio Cruschina and Andrea Calabrese. 2021. Fifty shades of morphosyntactic microvariation. In M.-O. Hinzelin and E.-M. Remberger, editors, *Formal Approaches to Romance Morphosyntax*, pages 145–198. de Gruyter.

Christopher Culy. 1985. The complexity of the vocabulary of Bambara. *Linguistics and Philosophy*, 8:345–352.

Amy Rose Deal. 2024. Interaction, satisfaction, and the PCC. *Linguistic Inquiry*, 55:39–94.

Amy Rose Deal. 2025a. Case sensitivity reflects case structure: Agreement, extraction, and clitics. UC Berkeley.

Amy Rose Deal. 2025b. Current models of agree. *Berkeley Papers in Formal Linguistics*, 4. Forthcoming in J. Crippen, R.-M. Déchaine and H. Keupdjio, editors, *Move and Agree: Towards a formal typology*. John Benjamins.

Jochen Dörre and Andreas Eisele. 1990. Feature logic with disjunctive unification. *Proceedings of Computational linguistics*, 13:100–105.

Andreas Eisele and Jochen Dörre. 1988. Unification of disjunctive feature descriptions. *Proceedings of Association for Computational Linguistics*, 26.

David Embick. 2000. Features, syntax, and categories in the Latin perfect. *Linguistic Inquiry*, 31:185–230.

David Embick and Alec Marantz. 2008. Architecture and blocking. *Linguistic Inquiry*, 39:1–53.

David Embick and Rolf Noyer. 2001. Movement operations after syntax. *Linguistic Inquiry*, 32:555–595.

Samuel D. Epstein, Hisatsugu Kitahara, and T. Daniel Seely. 2012. Labeling by minimal search. *Linguistic Inquiry*, 45:463–481.

Samuel D. Epstein, Hisatsugu Kitahara, and T. Daniel Seely. 2020. Unifying labeling under minimal search in single- and multiple-specifier configurations. *Coyote Papers: Working Papers in Linguistics, Linguistic Theory at the University of Arizona*.

Marina Ermolaeva and Gregory M. Kobele. 2022. Agreement as information transmission. *Syntax*, 25:466–507.

Paula Fenger. 2019. Size matters: Auxiliary formation in the morpho-syntax and the morpho-phonology. *Proceedings of North East Linguistic Society*, 49.

Irene Fernández-Serrano. 2025. Phase theory: inception, developments and challenges. In K.K. Grohmann and E. Leivada, editors, *Cambridge Handbook of the Minimalist Program*. Cambridge University Press.

Nissim Francez and Shuly Wintner. 2012. *Unification Grammars*. Cambridge University Press.

Doreen Georgi. 2017. Patterns of movement reflexes as the result of the order of merge and agree. *Linguistic Inquiry*, 48:585–626.

Mina Giannoula. 2025. Deciphering mirror principle violations. *Morphology*, 35:305–338.

Jean-Yves Girard. 1995. Linear logic: Its syntax and semantics. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, pages 1–42. Cambridge University Press.

Thomas Graf. 2013. *Local and Transderivational Constraints in Syntax and Semantics*. Ph.D. thesis, UCLA.

Thomas Graf. 2022. Typological implications of tier-based strictly local movement. *Proceedings of Society for Computation in Linguistics*, 5:184–193.

Thomas Graf. 2023a. Minimalism and computational linguistics. *LingBuzz*, 005855.

Thomas Graf. 2023b. Subregular tree transductions, movement, copies, traces, and the ban on improper movement. *Proceedings of Society for Computation in Linguistics*, 6:289–299.

Jane Grimshaw. 1997. Projection, heads, and optimality. *Linguistic Inquiry*, 28:373–422.

Matthieu Guillaumin. 2004. Conversions between mildly sensitive grammars. UCLA and École Normale Supérieure, Paris.

Morris Halle and Alec Marantz. 1993. Distributed morphology and the pieces of inflection. In K. Hale and S.J. Keyser, editors, *The View from Building 20*, pages 111–176. MIT Press.

Kenneth Hanson. 2025. Tier-based strict locality and the typology of agreement. *Journal of Language Modeling*, 13:43–97.

Boris Harizanov. 2019. Head movement to specifier positions. *Glossa*, 4:140.

Boris Harizanov and Vera Gribanova. 2019. Whither head movement? *Natural Language and Linguistic Theory*, 37:461–522.

Henk Harkema. 2001. *Parsing Minimalist Languages*. Ph.D. thesis, University of California.

Jason D. Haugen and Daniel Siddiqi. 2016. Towards a restricted realization theory. In D. Siddiqi and H. Harley, editors, *Morphological Metatheory*. John Benjamins.

Fabian Heck and Gereon Müller. 2016. On accelerating and decelerating movement. In G. Legendre, M. Putnam, H. de Swart, and E. Zaroukian, editors, *Optimality-Theoretic Syntax, Semantics, and Pragmatics*, pages 78–110. Oxford University Press.

Johannes Hein. 2024. Category-sensitive escape from islands in Limbum and Asante Twi. *Languages*, 9:317.

Tomio Hirose. 2011. *Origins of Predicates: Evidence from Plains Cree*. Routledge.

Norbert Hornstein. 2024. *The Merge Hypothesis*. Cambridge University Press.

Tim Hunter and Robert Frank. 2021. Comparing methods of tree-construction across mildly context-sensitive formalisms. *Proceedings of Society for Computation in Linguistics*, 4:355–358.

Mark Johnson. 1988. *Attribute Value Logic and The Theory of Grammar*. CSLI. https://isbnsearch.org/isbn/9780937073360.

Mark Johnson. 1994. Two ways of formalizing grammars. *Linguistics and Philosophy*, 17:221–248.

Aravind K. Joshi and Yves Schabes. 1997. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Volume 3: Beyond Words*, pages 69–124. Springer.

Laura Kalin and Nicholas Rolle. 2024. Deconstructing subcategorization. *Linguistic Inquiry*, 55:197–218.

Laura Kalin and Philipp Weisser. 2025. Minimalism and morphology. In K.K. Grohmann and E. Leivada, editors, *Cambridge Handbook of the Minimalist Program*. Cambridge University Press.

Laura Kallmeyer. 2010. *Parsing Beyond Context-Free Grammars*. Springer.

Makoto Kanazawa. 2007. Parsing and generation as Datalog queries. *Proceedings of Association for Computational Linguistics*, 45:176–183. https://aclanthology.org/P07-1023/.

Makoto Kanazawa. 2009. The pumping lemma for well-nested multiple context-free languages. *Developments in Language Theory*, 13:312–325.

Makoto Kanazawa. 2016. Formal grammar: An introduction. Lecture 5: Mildly context-sensitive languages. Lecture notes, Hosei University.

Makoto Kanazawa. 2017. Parsing and generation as Datalog query evaluation. *Journal of Logics and their Applications*, 4. http://www.collegepublications.co.uk/downloads/ifcolog00013.pdf.

Richard S. Kayne. 1993. Toward a modular thoery of auxiliary selection. *Studia Linguistica*, 47:1–31.

Richard S. Kayne. 1994. *The Antisymmetry of Syntax*. MIT Press.

Richard S. Kayne. 2020. Antisymmetry and externalization. *LingBuzz*, 005554.

Alan Hezao Ke. 2024. Can agree and labeling be reduced to minimal search? *Linguistic Inquiry*, 55:849–870.

Stefan Keine. 2019. Selective opacity. *Linguistic Inquiry*, 50:13–61.

Stefan Keine. 2020. *Probes and their horizons*. MIT Press.

Sana Kidwai. 2023. *Voice, Case and the External Argument: The Perspective from Urdu*. Ph.D. thesis, University of Cambridge.

Gregory M. Kobele. 2002. Formalizing mirror theory. *Grammars*, 5:177–221.

Gregory M. Kobele. 2006. *Generating Copies*. Ph.D. thesis, UCLA.

Gregory M. Kobele. 2021. Minimalist grammars and decomposition. Leipzig University. https://home.uni-leipzig.de/gkobele/files/unpub/ Kobele21MGsAndDecomposition.pdf.

Gregory M. Kobele, Christian Retoré, and Sylvain Salvati. 2007. An automata-theoretic approach to minimalism. In *Model Theoretic Syntax at 10, ESSLLI'07*.

Howard Lasnik. 1981. Restricting the theory of transformations. In N. Hornstein and D. Lightfoot, editors, *Explanation in Linguistics*, pages 125–145. Longman.

Howard Lasnik. 2000. *Syntactic Structures Revisited*. MIT Press.

Travis Major and Harold Torrence. 2024. Escape from noun complement clauses in Avatime. *Languages*, 9:339.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The Penn Treebank: annotating predicate argument structure. *Proceedings of Workshop on Human Language Technology*, pages 114–119.

John T. Maxwell and Ronald M. Kaplan. 1989. An overview of disjunctive constraint satisfaction. *Proceedings of International Workshop on Parsing Technologies*, 1:18–27.

Jens Michaelis. 1998. Derivational minimalism is mildly context-sensitive. *Proceedings of Logical Aspects of Computational Linguistics*, 3:179–198.

Jens Michaelis and Marcus Kracht. 1997. Semilinearity as a syntactic invariant. *Proceedings of Logical Aspects of Computational Linguistics*, 1:329–345.

Dale Miller. 2022. A survey of the proof-theoretic foundations of logic programming. *Theory and Practice of Logic Programming*, 22:859–904.

Marc Olivier. 2025. A syntactic account of auxiliary selection in French. *Probus*, 31.

Asia Pietraszko. 2023. Cyclic selection. *Linguistic Inquiry*, 54:350–377.

Jean-Yves Pollock. 1989. Verb movement, universal grammar, and the structure of IP. *Linguistic Inquiry*, 20:365–424.

Geoffrey K. Pullum. 2013. The central question in comparative syntactic metatheory. *Mind and Language*, 28:492–521.

Luigi Rizzi. 2013. Locality. *Lingua*, 130:169–186.

Luigi Rizzi. 2017. Locality and the functional sequence in the left periphery. In E. Aboh, E. Haeberli, G. Puskás, and M. Schönenberger, editors, *Elements of Comparative Syntax*, pages 319–348. De Gruyter.

Ivan A. Sag, Rui P. Chaves, Anne Abeillé, Bruno Estigarribia, Dan Flickinger, Paul Kay, Laura A. Michaelis, Stefan Müller, Geoffrey K. Pullum, Frank Van Eynde, and Thomas Wasow. 2020. Lessons from the English auxiliary system. *Journal of Linguistics*, 56:87–155.

Sylvain Salvati. 2011. Minimalist grammars in the light of logic. In S. Pogodalla, M. Quatrini, and C. Retoré, editors, *Logic and Grammar*, pages 81–117. Springer.

Hagay Schurr, Jason Kandybowicz, Abdoulaye Laziz Nchare, Tysean Bucknor, Xiaomeng Ma, Magdalena Markowska, and Armando Tapia. 2024. Absence of clausal islands in Shupamem. *Languages*, 9:7.

Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88:191–229.

Stuart M. Shieber. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8:333–344.

Stuart M. Shieber. 1986. *An Introduction to Unification-Based Approaches to Grammar*. CSLI.

Esturo Shima. 2000. A preference for move over merge. *Linguistic Inquiry*, 32:375–385.

Dominique Sportiche. 1998. *Partitions and Atoms of Clause Structure*. Routledge.

Edward P. Stabler. 1997. Derivational minimalism. *Proceedings of Logical Aspects of Computational Linguistics*, 1:68–95.

Edward P. Stabler. 2001. Recognizing head movement. *Proceedings of Logical Aspects of Computational Linguistics*, 4:254–260.

Edward P. Stabler. 2011. Computational perspectives on minimalism. In C. Boeckx, editor, *Oxford Handbook of Linguistic Minimalism*, pages 617–641. Oxford University Press.

Edward P. Stabler and Edward L. Keenan. 2003. Structural similarity within and among languages. *Theoretical Computer Science*, 293:345–363.

Miloš Stanojević. 2019. On the computational complexity of head movement and affix hopping. *Proceedings of Formal Grammar*, 24:101–116. Springer LNCS 11668.

Mark Steedman and Jason Baldridge. 2011. Combinatory categorial grammar. In R. Borsley and K. Börjars, editors, *Non-Transformational Syntax*, pages 181–224. Wiley.

Peter Svenonius. 2016. Spans and words. In H. Harley and D. Siddiqi, editors, *Morphological Metatheory*, pages 199–220. John Benjamins, Amsterdam.

Susi Wurmbrand. 2014. The merge condition: A syntactic approach to selection. In P. Kosta, S.L. Franks, T. Radeva-Bork, and L. Schürcks, editors, *Minimalism and Beyond*, pages 130–166. John Benjamins.

Ryo Yoshinaka and Alexander Clark. 2012. Polynomial time learning of some multiple context-free languages with a minimally adequate teacher. In *Formal Grammar 2010, 2011*. Springer.

Michelle Yuan. 2025. Morphological conditions on movement chain resolution. *Natural Language and Linguistic Theory*, 43:509–562.

The Typology of TSL Case Assignment

Kenneth Hanson
Department of Linguistics
Stony Brook University
mail@kennethhanson.net

Abstract

Long-distance syntactic dependencies such as movement and agreement have been argued to be *tier-based strictly 2-local* (TSL-2) over Minimalist Grammar (MG) dependency trees, placing tight constraints on the range of predicted patterns (Graf 2022b; Hanson 2025b). Here, I argue that the same is true of the syntax of case. Building on Vu et al. (2019) and Hanson (2023), I show that small variations on a TSL-2 tree grammar correspond to well-known phenomena such as case spreading, variation in alignment, and several types of differential argument marking. At the same time, many simple yet unnatural patterns are ruled out. This study therefore strengthens the TSL-2 hypothesis while providing a computational explanation for several aspects of the typology of case.

1 Introduction

The *tier-based strictly 2-local* (TSL-2) formal languages over strings and trees have been found to be a good model of many long-distance phonological and syntactic dependencies, respectively (Heinz 2018; Graf 2022a). In addition to providing a tight upper bound on predicted structural patterns, the patterns that can be expressed correspond closely to attested cross-linguistic variation in long-distance phonotactics (McMullin and Hansson 2016), syntactic movement (Graf 2022b), and agreement (Hanson 2025b). These results thus show how formal considerations provide insight into issues of locality and typology which are of broad interest within linguistic theory.

The syntax of case has also been studied from a TSL perspective by Vu et al. (2019) and Hanson (2023), but no such cross-linguistic study has yet been conducted. In addition, prior work assumed that case may utilize structure-sensitive tier projection, which increases expressive power, and did not emphasize the typological significance of limiting the constraint window on the tier to two elements. This work addresses all of the above limitations. It argues that many well-attested case patterns are understood as follows: case assigners (licensors) determine the cases of one or more nominals in their domain (clause, VP, etc.) according to a TSL-2 tree language. Furthermore, the natural parameters of the model—the set of visible elements and the structure of the bigram constraints—correspond to key points of attested variation, while many unnatural patterns are ruled out. Case therefore turns out to be less mysterious than is sometimes thought: it occupies a small and well-structured corner of the space of formal patterns and is computationally similar to other long-distance dependencies.

The resulting model, which makes full use of both dimensions of a tree tier, is highly reminiscent of existing configurational theories of case (Yip et al. 1987; Baker 2015), but unlike those theories, it uses only computational mechanisms which are independently necessary. The analysis does raise several formal questions, in particular, why certain TSL-2 patterns do not seem to have any correlate in the realm of case. While some of these gaps are presumably best explained by extra-grammatical factors, I propose two additional formal generalizations which further narrow down the space of possible case patterns.

The remainder of this paper proceeds as follows. First, I introduce the TSL-2 model of case assignment and discuss some important characteristics of the model (2). Next, I show how the model accounts for many classic case phenomena while ruling out unattested patterns (3). Finally, I address the overgeneration question, and discuss how case fits within the larger picture (4).

2 Background and model

This section lays out the background on case and subregular syntax, and presents a TSL-2 model of case which forms the basis for the cross-linguistic analysis which follows. It begins with a brief overview of subregular complexity (2.1), the syntax of case (2.2), and previous subregular analyses of case (2.3). Next, I introduce the MG dependency tree framework (2.4) and how it can be used to model case dependencies (2.5), with examples from English. Finally, I discuss some formal and linguistic issues related to the proposed model (2.6).

2.1 Subregular complexity

The Chomsky Hierarchy (Chomsky 1959) measures the complexity of linguistic patterns in terms of string languages (string sets); similar hierarchies can be described for tree languages, string-to-string maps, and so on. Within these classifications, studies to date have found that linguistic patterns are overwhelming *subregular*, meaning that they do not require the full power of even the *regular*, or finite-state class (see Graf 2022a and references therein). Of the various subregular classes, TSL appears to be the upper bound for most individual dependencies, and M[ulti]-TSL for a grammar containing many such dependencies. This is true even for syntax if we use an appropriate tree-based representation. Limited exceptions also exist; so far these have been modeled with a set of extensions known as *structure-sensitive TSL*, or SS-TSL.[1]

2.2 Case in syntax

This work is solely concerned with the syntactic distribution of case, not the details of morphological realization, nor the distribution of overt/covert nominals ('Case' in the sense of Chomsky 1981). Given a syntactic structure, we seek only to correctly determine the cases assigned to each nominal. Within this scope, I focus almost exclusively on *structural cases* such as nominative, accusative, dative, ergative, and genitive, which have a non-trivial distribution. *Lexical case*, which is assigned to arguments of specific predicates, will be treated briefly, as it is interesting insofar as it interacts with structural case. *Semantic cases* such as locative/instrumental/etc. which occur mainly on adjuncts, are essentially self-licensed, and will therefore be ignored.[2] See Blake (2001), Butt (2006), and Malchukov and Spencer (2008) for additional general background on case.

Note that I assume to nominal phrases be DPs, but nothing crucial hinges on this; some or all nominals must instead be NPs, KPs, etc. Next, note that a given case may have multiple modes of assignment; for our purposes these will be considered distinct cases. Finally, although I speak of case being *assigned* to DPs, the present account is, mathematically speaking, a pure *licensing* model; see Section 2.6 for explanation.

2.3 Previous work

Vu et al. (2019) made the initial argument that the syntax of case is TSL. Technically, their model is SS-TSL rather than TSL proper; however, the need for SS-TSL comes solely from their use of traditional MG derivation trees, as is true of Graf (2018), on which their model is based. With the introduction of MG dependency trees (a different subtype of derivation tree) the need for SS-TSL appears to be eliminated for movement (Graf 2022b), agreement (Hanson 2025b), and case (this work), though not for binding (Graf and Shafiei 2019).

The present analysis therefore starts with the model in Hanson (2023), which uses dependency trees to analyze a variety of case patterns in Japanese. Although not emphasized in that paper, the analysis is entirely TSL-2 except for a brief use of SS-TSL for reasons of conciseness; Hanson (2025a, ch. 5) revises this to achieve a pure TSL-2 analysis. I also retain the approach of gathering all DPs in some case domain as the daughters of the domain node on a tier, with the case of each being determined by the tier mother and sisters together, as introduced there.

2.4 MG dependency trees

Here, I describe the MG dependency tree model as used in recent work in subregular syntax.[3]

MGs (Stabler 1997, 2011) are a formalization of ideas from Minimalist syntax. They can generate a variety of structures, but for present purposes the relevant structure is a *derivation tree*, which encodes the sequence of operations (Merge, Move, etc.) in a syntactic derivation. MG *dependency trees* are a subtype of derivation tree in which every node represents a lexical item; the operations themselves are indicated only in the form of *feature diacritics* on the nodes themselves. See Graf and Kostyszyn (2021) for a formal definition.

[1] See De Santo and Graf (2019) for a formal definition and Mayer and Major (2018), Graf and Mayer (2018), and Graf and Shafiei (2019) for empirical studies.

[2] Some theories distinguish an additional category of *inherent case*, which is assigned to arguments with a specific thematic role. For our purposes, it can be rolled into lexical case, since it is similarly local in character.

[3] The dependency trees in Kobele (2012) are essentially identical. In contrast, Boston et al. (2010) generate surface dependency trees using MGs.

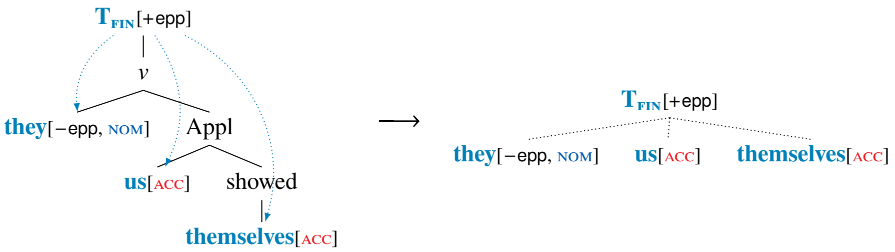


Figure 1: MG dependency tree and case tier for *They showed us themselves*. Left: full tree with case tier overlain. Right: case tier in isolation. Other dependencies such as EPP movement are controlled by a separate tier (not shown).

An example dependency tree for *They showed us themselves* is given in Figure 1, in which the subject moves to Spec-TP (call this EPP movement) and receives nominative case; both objects receive accusative case. Each daughter of a node is an argument, ordered from left to right in reverse order of first merge; this order also corresponds to asymmetric c-command among co-arguments. The features +epp and −epp indicate the landing site of EPP movement and the mover, respectively. Additionally, case features show which DPs receive nominative/accusative/genitive case. Following Hanson (2023), case assigners are not given a feature; instead, the label of the case assigner determines the string of cases of its dependents on a tree tier, as described in the next section. For simplicity, selectional features are omitted.[4]

As in traditional MG derivation trees, all elements appear in their base positions only. This works well for case, since scrambling and *wh*-movement, for example, preserve case. Even when movement is thought to alter case marking, the movement features themselves may be enough to determine the correct case; see Section 3.6 for an example. We diverge from standard MGs in one important way: the negative movement (and case) features are unordered, which is crucial if the tree language is to be TSL. For example, a DP bearing −epp and −wh simply moves to the closest landing site for each, and traces or copies are inserted as appropriate (cf. Graf 2022b, 2023).

2.5 TSL case assignment

I now explain the TSL model of case assignment as presented in Hanson (2023). I start with the definition of a TSL string language, which is then generalized to trees. I demonstrate the system with an analysis of some case patterns in English.

A string language is *tier-based strictly k-local* (TSL-k) iff there is a set of salient symbols such that, when all others are deleted, licit/illicit strings can be distinguished with a *strictly k-local* (SL-k) grammar (Heinz et al. 2011). The set of salient symbols is the *tier alphabet*, notated T, and the string obtained by deleting all non-salient symbols is a *tier projection*. The SL-k grammar may be either a *negative grammar* (G^-), which is a set of forbidden k-grams, or a *positive grammar* (G^+), which is a set of permitted k-grams; for present purposes the positive form is more convenient. Also, we include the *edge markers* ⋊/⋉ in the tier projection so that we can refer to the beginning/end of the string. For example, consider a natural language whose high vowels harmonize in backness, such that every word contains only [i] or [u] but never both. This pattern is TSL-2, with $T = \{i, u\}$ and $G^+ = \{\rtimes i, \rtimes u, ii, uu, i\ltimes, u\ltimes\}$. As a result, words like ‘kibili’ (⋊iii⋉) and ‘kubulu’ (⋊uuu⋉) are ruled in, but ‘kibulu’ (⋊iuu⋉) and ‘kubilu’ (⋊uiu⋉) are ruled out.

We generalize this idea to tree languages as follows. First, we ignore the non-salient elements and construct a *tree tier* which preserves dominance and precedence relations among those that remain. Continuing the preceding example, let us assume a single tier regulating case in English, containing all case assigners and assignees, which include at least T_{FIN} and all D heads (we consider other candidates below). The result is that all verbal arguments become daughters of T_{FIN}, as illustrated in Figure 1. Second, we associate each node on the tier with a TSL-k *daughter string language* (DSL), expressed in the form of a TSL-k *daughter string grammar* (DSG). In English finite clauses, the first (= highest) DP is nominative, and any others are accusative. Taking the position in the daughter string as a proxy for structural height (see Section 2.6), this pattern is described using the following grammar, which is

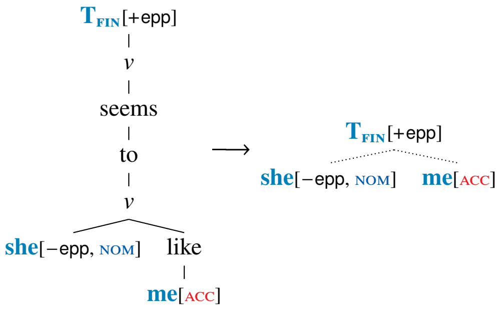
[4] Unlike in Minimalist syntax, MG features indicate operations that occur *in the present derivation* rather than the *potential* to undergo some operation.

SL-2, and therefore also TSL-2.[5] A dot is added between elements for readability, and movement features are omitted for simplicity.

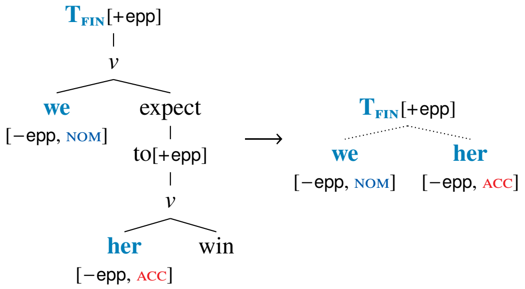
(1) Daughter string grammar for T_{FIN} (SL-2)
$G^+ = \{\rtimes \cdot \text{D[NOM]},\ \text{D[NOM]} \cdot \text{D[ACC]},\ \text{D[ACC]} \cdot \text{D[ACC]},\ \text{D[NOM]} \cdot \ltimes,\ \text{D[ACC]} \cdot \ltimes\}$

This grammar allows daughter strings such as D[NOM], D[NOM] · D[ACC], D[NOM] · D[ACC] · D[ACC], etc., while disallowing an accusative subject followed by a nominative object, among many other configurations. Assuming that infinitive *to* does not appear on the tier, the analysis immediately extends to raising infinitives as well as ECM infinitives. These are illustrated below.

(2) She seems to like me (raising)

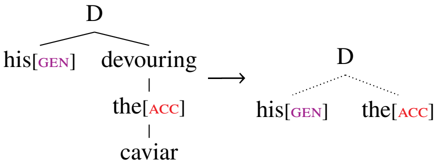


(3) We expect her to win (ECM)

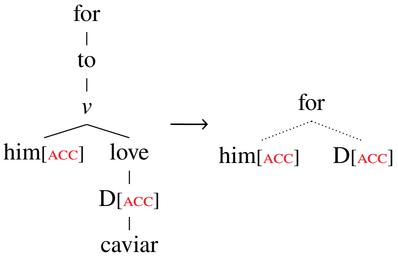


Several other aspects of English were previously treated by Vu et al. (2019) and could be translated into the current system with little difficultly. Since developing an in-depth analysis of English is not our goal, I will expand the grammar only slightly. (Movement features are henceforth omitted when not directly relevant.) I assume that DPs allow up to one genitive argument/possessor, possibly followed by one or more accusative arguments in the case of a *poss-ing* gerund (e.g. *his devouring the caviar*) and that other domains such as non-finite CPs (e.g. *for him to love caviar*) only allow accusative DPs. These structures are shown below. The case of the root is intentionally left unspecified, but would be accusative by default.

(4) his devouring the caviar (DP gerund)

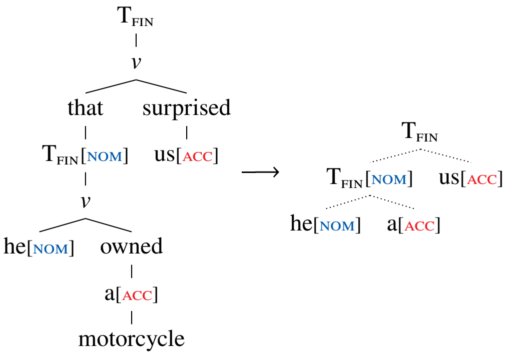


(5) *for him to love caviar* (non-finite CP)



We must now deal with the fact that T/C heads also occur in the DSL. Let us assume that they *absorb* the case that would otherwise be assigned in that position. That is, they are marked for case, but show no morphological realization, just as lexical DPs do not. The following example demonstrates absorption of nominative by a clausal subject. Later, we will consider examples where certain elements are invisible rather than filling a case slot.

(6) That he owned a motorcycle surprised us



Putting all of this together, we arrive at the following TSL grammar.[6]

(7) Case tier for English:

- Project: $\{\text{T}_{\text{FIN}}, \text{D}, \text{C}_{\text{INF}}\}$
- Daughter string grammars:
  T_{FIN}: $\{\rtimes \cdot \text{NOM},\ \text{NOM} \cdot \text{ACC},\ \text{ACC} \cdot \text{ACC},\ \text{NOM} \cdot \ltimes,\ \text{ACC} \cdot \ltimes\}$
  D: $\{\rtimes \cdot \text{GEN},\ \text{GEN} \cdot \text{ACC},\ \text{ACC} \cdot \text{ACC},\ \text{GEN} \cdot \ltimes,\ \text{ACC} \cdot \ltimes,\ \rtimes \cdot \ltimes\}$
  C_{INF}: $\{\rtimes \cdot \text{ACC},\ \text{ACC} \cdot \text{ACC},\ \text{ACC} \cdot \ltimes,\ \rtimes \cdot \ltimes\}$

This grammar incorporates several notational shortcuts. A lone category label stands in for any node of the given category *regardless of its MG*

[5] Although we do not make use of tier projection in the DSL at the moment, the definition of tree TSL allows it, and we do need it for case. See Sections 2.6 and 3.4.

[6] A more elegant analysis might use finite C for the domain of nominative case, mirroring C_{INF} for non-finite clauses.

features. Additionally, a label like NOM stands in for any nominative XP, regardless of its other features. The reader may confirm that each daughter string in the immediately preceding examples is well-formed according to the grammar.

To review, we have a TSL-2 model of case assignment which brings together all nominals in a case domain as the tier daughters of a case assigner, using the tier mother and sisters together in determining the case of each nominal. Compared to earlier work, the key innovation is the use of relative order among tier sisters. In existing TSL analyses of movement (Graf 2018, 2022b), all daughter string languages are closed under permutation, that is, the daughters can be shuffled with no change in wellformedness; here, this no longer holds. As shown in the next section, by making full use of both dimensions of the tree tier, we can account for a wide range of case phenomena in a straightforward manner.

2.6 Some consequences of the model

To close this section, I wish to briefly discuss a few formal and linguistic issues related to the model just presented.

Two chances for tier projection The definition of a TSL tree language allows for tier projection in the construction of the tree tier, and again in the daughter string languages. In the preceding example, there was no need for tier projection in the second step, but this is not always true. As discussed in Hanson (2023), adjuncts must be ignored for purposes of case assignment, but if they are ignored in the tree tier then any DPs inside them will be strewn among those in the containing clause. This problem can be avoided if we preserve adjuncts in the tree tier, then filter them out in the daughter string language. I make use of this technique in the treatment of lexical case in Section 3.4.

Configurational case assignment As mentioned in the introduction, the configurational perspective on case has ample precedent in the frameworks of *Case in Tiers* Yip et al. (1987) and *Dependent Case Theory* (Marantz 2000; Baker 2015), among others. While I do not conduct an in-depth comparison here, I do wish to highlight a few key points. Case in Tiers is a direct adaptation of autosegmental phonology (Goldsmith 1976), which itself inspired the TSL formal languages; unlike the current approach, it is framed in terms of linear surface order among DPs. The core idea of Dependent Case Theory is that case is assigned according to relative c-command relations among DPs in some domain; the present model is an implementation of the same idea.

As noted by Vu et al. (2019), tree tiers also subsume standard generative case theory, in which case is a pairwise relation between assigner and assignee. This is especially clear in the present model: such relations are just the special case in which the daughter string language is closed under permutation, as with movement. In this way, the model explains why case patterns which are described so naturally by configurational theories should exist—they are just examples of the more general capabilities of tree tiers.

Tier sisters and command Although we are using sister order as a proxy for structural height/prominence, it is possible for a node X to be the left sister of Y on the tree tier even though X does not c-command Y in the phrase structure tree. This could be useful in deriving certain c-command violations, such as the limited possibility for an NPI to be licensed by an element embedded in a left branch (e.g. [[*No one's*] directions] were *any* help). On the other hand, if c-command is in fact the relevant structural relation between DPs for purposes of case assignment, this could lead to incorrect predictions.

In practice, there is usually some independent reason to project the root of a complex left branch; this applies to adjuncts as just mentioned, but also to clausal subjects and PP subjects. That said, the issue points towards the need for additional constraints beyond those provided by tree tiers, as was already acknowledged by Graf (2022b). The other major approach to subregular syntax applies string constraints to paths which encode command relations (Graf and Shafiei 2019; Hanson 2025b). It may be possible to recast much of the present analysis in that system. However, given the naturalness of the tree tier analysis of case, it seems to me that the ideal solution for case would be a hybrid system in which tier sister order is guaranteed to respect c-command.[7]

Multi-tier patterns While a single case tier is sufficient for the above fragment of English, this might not be true for all languages. For example,

[7] While writing this paper, I considered redefining the tier sister relation in terms of the d[derivational]-command relation of Graf and Shafiei (2019), but this only created a new problem: a single node could end up with several discontinuous daughter strings. The complement spine model in Hanson (2025b), which filters out the unwanted command relations, may be a better starting point.

the analysis of Japanese in Hanson (2023) includes three case tiers, corresponding to case in the verbal domain (nominative, accusative, dative), case in the nominal domain (genitive), and lexical case; the entire case system is therefore MTSL-2.[8] In this paper, I focus on typological predictions of a single tier system, which by hypothesis is enough to handle the structural cases of a single domain, leaving a proper study of multi-tier patterns to future work. That said, I do include several examples of interactions across tiers in Section 3.

Assignment vs licensing Finally, a comment on the issue of case *assignment* versus *licensing*. A dependency tree is well-formed if it does not violate any constraints in an (M)TSL tree grammar. Conceptually, the presence of a case feature on a given node can be interpreted as saying that it will be assigned said case as part of the syntactic derivation. In a mathematical sense, however, the model is a pure licensing model; this applies not only to case, but to all syntactic dependencies. It may also be insightful to model case as a TSL transduction from the dependency tree to the corresponding phrase structure tree (see Graf 2023 regarding movement).

3 Cross-linguistic study

In this section, I demonstrate how the proposed model of case assignment accounts for many well-known and theoretically interesting patterns in the syntactic distribution of case. As in previous studies of this type, I emphasize how the natural parameters of the model—the tier projection and the constraints on the daughter strings—correspond closely to attested cross-linguistic variation (3.1). Specific phenomena of interest include case spreading (3.2), variation in alignment (accusative, ergative, etc.) (3.3), interactions between lexical and structural case (3.4), domain effects (3.5), and four types of differential argument marking (3.6). I also briefly discuss what we do *not* see, and why the model rules out such patterns (3.7).

3.1 Parameters of variation

I wish to highlight two aspects of the TSL-2 model of case which are the focus of the present study. First, the model provides two basic parameters for manipulation: i) the tier alphabet, which determines which items are visible, and ii) the bigrams of the daughter string grammars. Each of these is expected to vary across languages, and I show that the obvious ways that they may differ aligns closely with the variation that we see. For example, Graf (2022a) explains how the difference between single and multiple *wh*-movement boils down to presence or absence of a single bigram in the daughter string language. Similarly, many island effects arise from the presence of extra nodes on the tier which interrupt the relation between mother and daughter. Here, I show that the same sorts of variation can be found in the realm of case.

Second, we have considerable control over the relationship between a domain node and two adjacent DPs on the tier, both in the mother-daughter and left-sister relations. For example, we can refer to the properties of the the domain node to determine the case pattern for its daughters, and to the properties of the sisters in the derivation of differential argument marking. This also has precedent in previous work in TSL syntax (Graf 2022a; Hanson 2025b); as mentioned above, the main innovation here is the specific focus on tier sisters. In contrast, our ability to coordinate elements which do not stand in such a relationship is extremely limited. Accordingly, I show how many logically simple yet linguistically unnatural patterns are ruled out on the grounds that they are not TSL-2.

Of course, not every unattested pattern can be ruled out in this manner. Many other factors are involved, including substantive differences between different types of dependencies as well as constraints on diachronic development and language acquisition. Even so, we have a strong hypothesis for what constitutes a possible case pattern, which plausibly derives from some limitation of the underlying cognitive system (Graf 2022a). This in turn clarifies which gaps *must* be accounted for by other factors, computational or otherwise; I suggest some additional formal factors in Section 4.

3.2 Spreading

One of the major motivations of Yip et al. (1987) was to account for case spreading, that is, situations where a single case is assigned to multiple DPs in some domain. We saw spreading of accusative in the previous section, though as Yip et al. note, Swedish would be a better example, since accusative is plausibly the default case in English.

(8) Multiple accusative in Swedish (Yip et al. 1987)

Kungen	gav	honom	henne	(till maka).
the.king	gave	him.ACC	her.ACC	(for wife)

[8] An MTSL language is just the intersection of several TSL languages (De Santo and Graf 2019).

Similarly, Japanese allows multiple nominatives and genitives in several contexts.

(9) Multiple nominatives in Japanese

a. Stative object (Hiraiwa 2001)

Mary ga eigo ga yoku dekiru.
Mary NOM English NOM well can.do

'Mary can speak English well.'

b. Possessor raising (Kuno 1973)

Yama ga ki ga kirei desu.
mountain NOM trees NOM pretty are

'The mountains—their trees are pretty.'

(10) Multiple genitives in Japanese

a. Multiple possessors (Saito et al. 2008)

Taroo no Chomsky no hon
Taroo GEN Chomsky GEN book

'Taroo's book by Chomsky'

b. Nominal subject/object (Saito et al. 2008)

yabanzin no Rooma no hakai
barbarian GEN Rome GEN destruction

'the barbarians' destruction of Rome'

As discussed by Graf (2022b), multiple licensing is also a straightforward prediction of the TSL-2 model: all we need is a bigram in the relevant daughter string grammar which duplicates the same item twice. Thus, by including ACC · ACC, NOM · NOM, or GEN · GEN, etc., we enable spreading without the need to posit additional (covert) case assigners. We also predict that spreading will continue if additional DPs are present, since a daughter string with three or more repetitions of, say, accusative, has exactly the same bigrams: {⋊ · ACC, ACC · ACC, ACC · ⋉}. Put another way, an SL-2 grammar cannot distinguish "exactly two" from "two or more".

While the literature rarely discusses such constructions, this appears to be true at least of possessor raising in Japanese (Kuno 1973).[9] Furthermore, restricting spreading to a *maximum* of two DPs requires at least TSL-3. This kind of counting is highly unusual in natural language, and TSL-3 can generate many other strange patterns that TSL-2 cannot (see Section 3.7). In absence of contradictory evidence, the TSL-2 analysis of spreading is therefore preferred.

Another superficially similar phenomenon, not treated by Yip et al. (1987), is *case concord*, in which the case of the DP is realized in multiple places, including the head noun, the determiner, adjectival modifiers, and perhaps other elements. An extreme example of this type from Panjima (Pama-Nyungan), with five concordial elements, is shown below.

(11) Ngatha wiya-rna ngunha-yu maripa-yu
I.NOM see-PST that-ACC man-ACC
paka-lalha-ku nharniwalk-ku
come-PRF-ACC hither-ACC
warrungkamu-la-ku
morning-LOC-ACC

'I saw that man who came this way this morning.'
(Blake 2001, ch. 4)

There are several ways this might be implemented. The direct approach involves duplicating the bigrams of the spreading grammar to allow iteration of each case. The trouble with strategy is that the case shared among the concordial elements is not arbitrary, but rather determined by the case of the entire DP. One possible solution is to *split* the DSG for D on the case tier into several, one for each case.

(12) Schematic grammar for case concord

D[NOM]: {⋊ · NOM, NOM · NOM, NOM · ⋉, ⋊ · ⋉}
D[ACC]: {⋊ · ACC, ACC · ACC, ACC · ⋉, ⋊ · ⋉}
D[DAT]: {⋊ · DAT, DAT · DAT, DAT · ⋉, ⋊ · ⋉}
etc.

A better approach might be to treat case concord as a kind of agreement, since person/number/gender features are often shared as well. Furthermore, the class of items involved is somewhat different, including nouns and adjectives but excluding possessors and arguments (e.g. the possessor of an accusative DP does not become accusative). While the basic idea remains the same, the tier for case *assignment* would be completely separate from those for case *concord*; see Hanson (2024a) for a possible approach to the latter.

To briefly summarize, the TSL-2 model predicts the existence of iteration. The appears to be bourne out in several ways in the realm of case, mirroring its appearance in long-distance harmony, multiple *wh*-movement, and concord in the DP. However, we cannot reasonably expect every such pattern to occur in each of these domains. In particular, there are TSL-2 patterns which are highly characteristic of case but which are not nearly as robustly attested elsewhere. We consider some of these below.

3.3 Alignment

Case marking correlates imperfectly with both semantic role and grammatical function. In line with

[9] Numeral quantifiers and PP modifiers of nouns also take the *no* clitic, which would seem to imply an unlimited number of genitives. But *no* can also be an adnominal copula, and it is difficult to tell apart these two functions.

the configurational view adopted above, I show that many of the most common patterns correspond exactly to minor differences in the structure of the SL-2 daughter string language on a case tier. Note that many languages employ multiple alignments in the same domain, conditioned by various structural and lexical factors, which I abstract away from here; in Section 3.5 we consider case conditioned by aspect and finiteness.

Following common convention, I refer to the sole argument of an intransitive verb as S, the "more agent-like" argument of a transitive verb as A, and the "less agent-like" argument as O (cf. Bickel and Nichols 2008). The case of S is usually the (literally) unmarked case, which we call *nominative*. When A and S share a case and O receives a different case, we call the case of O *accusative*. This is known as accusative alignment, and is the dominant pattern in most Indo-European languages. It is also common for S and O share a case to the exclusion of A. We call the case of A *ergative*, and the overall pattern is known as ergative alignment. This is the dominant pattern in Shipibo (Panoan family).

(13) Shipibo (Baker 2015)

a. Maria-nin-ra ochiti noko-ke.
Maria-ERG-PRT dog find-PRF

'Maria found the dog.'

b. Maria-ra ka-ke.
Maria-PRT go-PRF

'Maria went.'

For the languages under discussion, I assume a consistent structural relationship such that A is higher than O. I further assume following Baker (2015) that ergative case in Shipibo is assigned to the higher of two DPs (the A argument) in the same clause.[10] The basic grammar for ergative alignment is therefore nearly identical to that for accusative. The key difference is that the 'new' case of the transitive structure is introduced to the left of nominative case.

(14) Grammar for ergative alignment
$G^{+} = \{\rtimes\cdot\text{NOM}, \rtimes\cdot\text{ERG}, \text{ERG}\cdot\text{NOM}, \text{NOM}\cdot\ltimes\}$

It is also possible for S, A, and O to each have a distinct case ('tripartite' alignment), or for them to all share the same case ('neutral' alignment). The neutral grammar is identical to what we called 'spreading' in 3.2, and the tripartite grammar is given below.

	Case of		
Alignment	S	A	O
Accusative	NOM	NOM	ACC
Ergative	NOM	ERG	NOM
Tripartite	NOM	ERG	ACC
Neutral	NOM	NOM	NOM

Table 1: Common case alignments

(15) Grammar for tripartite alignment
$G^{+} = \{\rtimes\cdot\text{NOM}, \rtimes\cdot\text{ERG}, \text{ERG}\cdot\text{ACC}, \text{NOM}\cdot\ltimes, \text{ACC}\cdot\ltimes\}$

We see that every logical possibility for the cases of the canonical transitive verb is realized: either A or O may have a special case, or both, or neither. This is summarized in Table 1. Furthermore, each variant differs in only a few bigrams compared to the others. Note that some authors, such as Bittner and Hale (1996), distinguish even more systems. For example, internal and external arguments of intransitive verbs may behave differently, which is a major factor in certain dialects of Basque. Additionally, alignment may change according to tense or aspect, as in Hindi. From the present perspective, such "alignments" usually involve multiple domain nodes, and will be addressed in Section 3.5.

Now, we ask what happens when there are more than two arguments. Following Bickel and Nichols (2008), I refer to the two objects of a ditransitive verb as G (for the more goal-like argument) and T (for the more theme-like argument); ditransitive subjects are rarely distinguished from transitive subjects, so they are still A. It is common for either G or T to receive the same case as O, while the other receives a new case. For example, G can be *dative*, and T aligned with O. This occurs both in languages with accusative alignment, such as German and Japanese, as well as languages with ergative alignment, such as Ingush (Nakh-Daghestanian).

(16) Japanese ditransitive verb
Jin ga Yumi ni hon o ageta.
Jin NOM Yumi DAT book ACC gave

'Jin give Yumi a book.' (Hanson 2023)

(17) Ingush ditransitive verb
Aaz cynna mashen jelar
1SG.ERG 3SG.DAT car.NOM gave

'I gave him/her a car' (Bickel and Nichols 2008)

Maintaining our earlier assumption that the default linear order indicate c-command among base

[10] It is likely that not every language with a case called 'ergative' works like this, which Baker acknowledges. See Butt (2006, ch. 6) for additional discussion. In Section 3.5, I examine another type of 'ergative' system.

argument positions, the grammar for such systems are given below.

(18) Accusative alignment with dative G
$G^+ = \{\rtimes\cdot\text{NOM},\ \text{NOM}\cdot\text{ACC},\ \text{NOM}\cdot\text{DAT},\ \text{DAT}\cdot\text{ACC},\ \text{NOM}\cdot\ltimes,\ \text{ACC}\cdot\ltimes\}$

(19) Ergative alignment with dative G
$G^+ = \{\rtimes\cdot\text{NOM},\ \rtimes\cdot\text{ERG},\ \text{ERG}\cdot\text{NOM},\ \text{ERG}\cdot\text{DAT},\ \text{DAT}\cdot\text{NOM},\ \text{NOM}\cdot\ltimes\}$

It is also possible for T to have a special case and G aligned with O, or for all three to be aligned, with various relationships to S/A. On top of this, some languages may employ low goals rather than high goal, or perhaps include both structures. While I cannot do justice to the full typology here, what is clear is that as with transitives, several well-attested systems correspond to simple manipulations of the TSL-2 grammar: is there a third basic case and does it occur before or after the case of O?

We could continue in this fashion by asking what happens when there are four arguments. Although simplex predicates with four DP arguments are rare (perhaps nonexistent), they can be constructed by valency increasing operations such as morphological causatives. Unfortunately, as with spreading, the literature on case rarely discusses such data. For instance, not a single example of a causative of a ditransitive could be found by searching all index entries for both “causative” and ”ditransitive” in the 900-page Oxford Handbook of Case (Malchukov and Spencer 2008). However, Hanson (2023) notes that such constructions are possible in Japanese, and the middle two DPs are both dative (both datives are plausibly structural, or at least ambiguous between structural and lexical dative).

(20) Causative of ditransitive
Ken ga Jin ni Yumi ni hon o agesaseta.
Ken NOM Jin DAT Yumi DAT book ACC gave.CAUS
‘Ken made/let Jin give Yumi a book.’

The grammar proposed there, shown below with extraneous details removed, involves spreading of dative.

(21) Verbal cases in Japanese
$G^+ = \{\rtimes\cdot\text{NOM},\ \text{NOM}\cdot\text{ACC},\ \text{NOM}\cdot\text{DAT},\ \text{DAT}\cdot\text{DAT},\ \text{DAT}\cdot\text{ACC},\ \text{NOM}\cdot\ltimes,\ \text{ACC}\cdot\ltimes\}$

While further investigation is clearly needed, our survey of alignment clearly supports the idea that the various structural cases available in a given domain can be assigned according to a TSL-2 grammar. There are, of course, alternatives that could fit the readily available data. For example, we could employ a traditional generative analysis, in which each case is associated with a unique functional head: T, Aspect, Voice, Appl, etc. But then we would lose generalizations like the dative rule in Japanese.[11] In contrast, such patterns are a straightforward prediction of the TSL-2 model with very coarse case domains.

3.4 Lexical case and visibility

In the preceding sections, we focused mostly on the structure of the constraints. Now, we begin to consider the contents of the tier alphabet as well.

In Section 2.5, I showed how a clausal subject fills the nominative slot in English, which I called case absorption. When a lexically case-marked DP occurs in a particular case position, it is possible for the corresponding structural case to instead be *displaced* to the next available DP. This occurs with lexical datives in Icelandic.

(22) Case displacement in Icelandic (Yip et al. 1987)

a. Siggi leyndi konuna sannleikanum.
S.NOM concealed the.woman.ACC the.truth.LDAT
‘Siggi concealed the truth from the woman.’

b. Siggi sagði barninu söguna.
S.NOM told the.child.LDAT the.story.ACC
‘Siggi told the child the story.’

c. Barninu batnaði veikin.
the.child.LDAT recovered.from the.disease.NOM
‘The child recovered from the disease.’

When LDAT occurs in the lowest (oblique) position, the subject and first object are unaffected. But when it marks in the first object position, accusative is displaced to the second object, and when it marks the subject, nominative is displaced to the object. Icelandic is a particularly good example of the latter phenomenon, as it is clear that the lexically-case marked subject is in fact in the usual subject position (Zaenen et al. 1985).

The core of the analysis is to make lexical datives as invisible on the tier controlling structural case in the clause. That is, the tier alphabet contains T and any D marked for structural case (nominative,

[11] The idea that dative is the case of the middle of three DPs originates in Kuno (1973). In Baker (2015), dative applies to the upper DP in the domain of VP, but only when *v*P is a phase. Kuno’s generalization therefore becomes an accident.

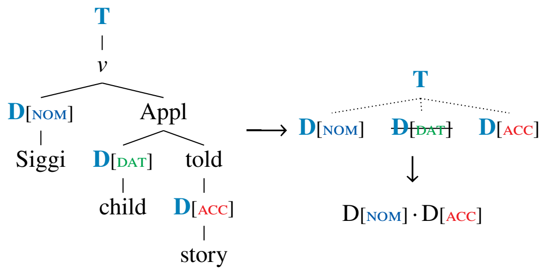


Figure 2: Derivation of case displacement. Lexical datives appear on the tree tier, but are ignored in the daughter string language of T. Nominative and accusative are adjacent as usual.

accusative, and perhaps structural dative) but not lexical dative. As a result, the tiers for the above examples look the same as they would for an ordinary transitive/intransitive clause.

As mentioned in Section 2.6, there is a complication, which is that any DPs inside the invisible lexical dative will be strewn among the arguments of the containing clause. The solution, borrowed from Hanson (2023), is to retain lexical datives in the tree tier, but ignore them in the daughter string language, which is now properly TSL-2. This is illustrated for (22b) in Figure 2.[12]

As a final remark, though I have not specified how lexical dative itself is assigned in Icelandic, lexical case is generally held to be assigned locally to arguments of specific predicates. This means that lexical case occupies the tier which contains everything; in other words, a simple SL grammar will do. But as we have seen, it is still formally interesting due to its interaction with cases assigned on non-trivial tiers.

3.5 Domain effects

Next, we consider effects arising from the the nature of the domain nodes which appear on the tier.

As mentioned in Section 2.6, it is normal to have several case domains, which might include TP, VP, DP, and perhaps others. Case may also correlate with clausal properties such as aspect, as in Hindi.

(23) Alignment split by aspect in Hindi (Mahajan 1990)

a. Raam roṭii khaataa thaa.
Raam.NOM bread.NOM eat.IPFV be.PST
'Raam ate bread (habitually).'

b. Raam-ne roṭii khaayii.
Raam-ERG bread.NOM eat.PFV
'Raam ate bread.'

Based on this data, imperfective clauses have neutral alignment while perfectives are ergative. (Hindi also has differential object marking, which is covered in 3.6.) Leaving aside the details, let us assume that the Asp(ect) head, realized by an auxiliary only in the imperfective, controls alignment according to the following grammar:

(24) Split alignment in Hindi

- Project: {Asp, D}
- Daughter string grammars:
  Ipfv: {⋊ · NOM, NOM · NOM, NOM · ⋉}
  Pfv: {⋊ · NOM, ⋊ · ERG, ERG · NOM, NOM · NOM, NOM · ⋉}

In reality, it is more likely that neutral and ergative alignment are associated with Asp and T or vice versa (Thomas McFadden, p.c.), but without delving into the details, it is difficult to distinguish between these alternatives. The key point is that major category alone need not be the distinguishing factor.

However, a clear example of this type can be found in Basque, in which finiteness is the distinguishing factor. For context, transitive clauses show ergative alignment, while the case of an intransitive subject varies. The data is complex, but in western/central dialects, the basic generalization is that a lone external argument receives ergative case, while a lone internal argument receives absolutive (=nominative) case, a pattern which is sometimes called 'Split S'.

(25) Split S in Basque (Berro and Etxepare 2017)

a. Jon-ek dantza-tu du.
Jon-ERG dance-PRF AUX
'Jon has danced.' (unergative)

b. Jon eror-i da.
Jon.NOM fall-PRF AUX
'Jon has fallen.' (unaccusative)

By testing a variety of constructions, it is possible to show that ergative case is only available in the domain of finite T (Rezac et al. 2014). For example, embedded finite clauses can have ergative subjects, whereas in gerunds, which are non-finite, ergative marking is lost.

(26) Ergative is lost in gerunds (Rezac et al. 2014)

a. [Katu-ek sagu-ak harrapa-tu
cat-PL.ERG mouse-PL.NOM caught
dituzte-la] ikusi dut.
AUX-that seen AUX

[12] Hanson (2023) uses this technique to factor out PP adjuncts—lexical datives do not require this treatment due to a quirk of the grammar of Japanese. We might also use it for PP arguments that induce case displacement, to the extent that such PPs can be distinguished from DPs with lexical case.

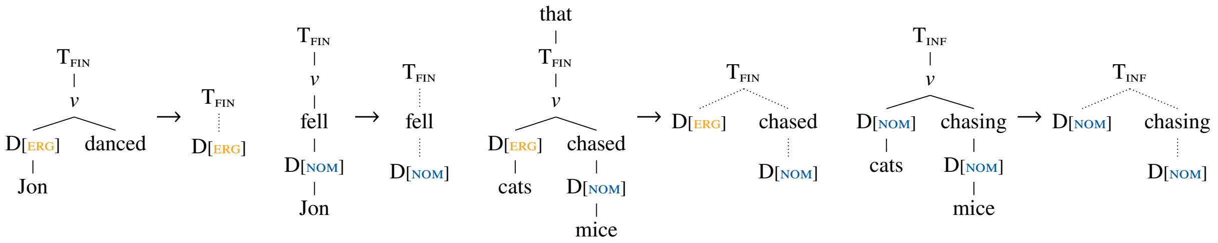


Figure 3: Ergative in Basque is available only in the domain of finite T. From left to right: unergative, unaccusative, transitive finite, transitive gerund.

'I saw that the cats caught the mice.'

b. [Katu-ak sagu-ak harrapa-tzen] ikusi ditut.
cat-PL.NOM mouse-PL.NOM catch-ing seen AUX

'I saw the cats catch the mice.'

We can derive this pattern if we assume that TP and VP are case domains and that only finite T can assign ergative; both V and non-finite T cannot. This is illustrated in Figure 3. I leave open whether the domain nodes should absorb case, or whether they should be ignored in the daughter string language as in Section 3.4.

It is also possible for a domain node that is usually present to be invisible in certain contexts. This is the basis of the analysis of long-distance (cross-clausal) case assignment in Hanson (2023), as exemplified by constructions such as the following.

(27) Finite ECM (adapted from Kishimoto 2018)

Ken ga [Eri {ga/o} kawaii to] omotteiru.
Ken NOM Eri {NOM/ACC} be.cute C think

'Ken thinks that Eri is cute.'

While I cannot go into detail here, the basic idea is that certain lexical predicates optionally select an embedded complementizer which is invisible on the relevant tier. Thus, all arguments become dependents of the upper case assigner, effectively merging two case domains, with the result that the embedded subject receives accusative case like a canonical object.[13] Though it is perhaps more common to analyze such phenomena as involving a truncated clause structure, the TSL approach allows us abstract away from the precise nature of the structural difference. This is helpful for Japanese since the particle *to* appears extremely high in the extended CP (Saito 2015), yet is still present under long-distance assignment of accusative.

To summarize what we have seen so far, we can derive many of the most common variations in case marking simply by manipulating the contents of the tier alphabet, the number of cases, and their positions in the bigrams of the daughter string grammars. In the next section, we consider the last obvious variable, which is to utilize features of DPs other than case in our grammars.

3.6 Differential argument marking

Differential argument marking (DAM) refers to phenomena in which some morphological marking of a DP, including but not limited to case, is only sometimes present; see Witzlack-Makarevich and Seržant (2018) for an overview. DAM subsumes both differential object marking (DOM) as well as differential subject marking (DSM), and tends to be linked to discourse and semantic factors, which themselves could involve the argument itself or a co-argument. Here, I show that the TSL-2 model derives this four-way typology.

We begin with a classic example of DOM, as illustrated with accusative case in Sakha (Turkic). In Sakha, accusative case marking is associated with a definitive/specific interpretation as well as movement out of the VP, as diagnosed by the position of manner adverbs.

(28) Sakha DOM (Baker and Vinokurova 2010)

a. Masha salamaat-*(y) türgennik sie-te.
Masha porridge-ACC quickly eat-PST.3SG

'Masha ate the porridge quickly.'

b. Masha türgennik salamaat-(#y) sie-te.
Masha quickly porridge-ACC eat-PST.3SG

'Masha ate porridge quickly.'
(ACC on 'porridge' only with contrastive focus)

Baker and Vinokurova (2010) posit that accusative is assigned to the lower of two DPs in the domain of T (as I have been assuming), and that

[13] In Japanese, the embedded subject need not vacate the clause for this to happen, though as discussed below, we can also handle case alternations triggered by movement in the present system.

	Locus of Controlling Property	
Subtype	Self	External
DSM	Kham, Folopa, Umpithamu	Eastern Ostyak, Ika, Kanuri
DOM	Hindi, Spanish, Turkish, Sakha	Ik

Table 2: Typology of self/externally driven DAM

movement of the object out of the VP is what causes it to receive accusative case. That movement is the trigger for DOM is not a foregone conclusion, as I discuss below, but let us adopt this view for now.

As discussed in Graf (2022b) and Hanson (2025b), we can handle certain interactions between syntactic dependencies by referencing the MG features for one dependency in the TSL grammar for another, as there is no *a priori* reason why such information should not be available. For example, case-sensitive agreement is modeled by projecting only D heads bearing an appropriate case on the tier regulating agreement. Here, we assign accusative only to DPs which bear the feature for object shift in addition to appearing after another another DP on the tier. This is illustrated in the grammar below.

(29) Accusative DOM triggered by object movement

- Project: $\mathrm{T}_{\textsc{fin}}$, all D
- DSG of $\mathrm{T}_{\textsc{fin}}$: $\{\rtimes \cdot \mathrm{D}[\textsc{nom}], \mathrm{D}[\textsc{nom}] \cdot \mathrm{D}[-\text{oshift}, \textsc{acc}], \mathrm{D}[\textsc{nom}] \cdot \mathrm{D}[\textsc{nom}], \mathrm{D}[\textsc{nom}] \cdot \ltimes, \mathrm{D}[-\text{oshift}, \textsc{acc}] \cdot \ltimes\}$

If such constraints are generally possible, we predict that movement of the object may also influence ergative case marking of the subject, a kind of DSM. Baker (2015, pp. 127–129) also gives several examples of this type. In addition, we predict that DSM could be triggered by an internal property of the subject, or conversely, DOM triggered externally by a property of the subject. As discussed by Daniel (2024), all four possibilities are realized. The set of examples she cites are shown in Table 2.

At least one cell in the typology is problematic for many approaches to case. For example, self-driven DSM is unexpected under Baker's assumptions, since subjects do not normally move for semantic/discourse reasons, while externally-driven DAM is generally unexpected under functionalist theories (Baker 2015, pp. 129–130). From the present perspective, all four cells are equally easy to derive. Furthermore, movement need not be the trigger; it could also be agreement, or the underlying semantic/discourse feature, if we make it available in the syntax. Indeed, Hindi also has differential object marking (not shown previously), but unlike in Sakha it is not thought to involve movement.

We can therefore provide a maximally simple explanation for the four-way typology: given that the subject and object appear in the same tier bigram, we can reference the properties of either in determining their cases. In future work it would be important to confirm whether global case splits (cf. Bárány 2017), in which the combined properties of *both* the subject and object influence case or agreement, can be handled in the same manner. In principle this should be possible, but the data is complex enough that a closer look is warranted.

As this marks the end of our final case study, the case phenomena which have been treated in this paper are summarized in Table 3. Just from this brief survey, we have identified clear correlates of visibility in both the tree tier and daughter string language as well as various manipulations of the mother-daughter and sister-sister relations on the tier. Conversely, many common and sometimes theoretically problematic case phenomena have a straightforward analysis in the TSL-2 model.

3.7 Non-existent case patterns

Having shown how many of the positive predictions of the TSL-2 model for case are bourne out, I close this section by examining some non-existent patterns which are ruled out purely on formal grounds. Graf (2022b) constructs a selection of logically simple yet unattested island constraints, and notes that none of them are TSL-2. Now, consider the following hypothetical case systems:

(30) **2x2 alignment:** the first two DPs are nominative; the next two DPs are accusative; repeat

(31) **Anti-local dependent case:** the subject is nominative if there are one or two arguments and ergative if there are three or more; objects are always accusative

(32) **Coordinated alignment:** when two clauses are coordinated, one must have accusative alignment and the other ergative, in either order

(33) **Rationed dative case:** up to two clauses may contain dative DPs; the rest use genitive instead

(34) **Rotating alignment:** the main clause is accusative; a singly embedded clause is ergative; a doubly embedded clause is accusative; etc.

All of these are straightforwardly ruled out due to requiring some non-TSL-2 mechanism, either in domain node identification or in the daughter string language:

Phenomenon	Section	One line summary
Spreading	3.2	Allow NOM · NOM, ACC · ACC, etc. in the DSL
Alignment	3.3	Does the marked case precede or follow the unmarked case?
Split alignment	3.5	PFV/IPFV, $\mathrm{T}_{\mathrm{FIN}}/\mathrm{T}_{\mathrm{INF}}$, etc., have different DSLs
Long-distance case	3.5	Certain C heads are invisible on the structural case tier
Case absorption	2.5	P/C/etc. are visible on the tree tier and in the DSL
Case displacement	3.4	Lexical case-marked D is invisible in the DSL
Diff. argument marking	3.6	DSL can reference non-case features of D

Table 3: Summary of case patterns and their TSL-2 analysis

- A window larger than two (30, 31)
- Global boolean logic (32)
- Threshold counting (33)
- Modulo counting (34)

These examples were selected in part because they are conceptually straightforward yet extremely unnatural from a linguistic perspective. They are also simple in computational terms, all being regular over dependency trees, and with the exception of (34), subregular. As such, we have good reason to believe it is TSL-2 specifically that characterizes the class of possible case patterns—it is powerful enough, but also not overpowered.

One possible objection to this line of reasoning is that all of the above systems are clearly dysfunctional as a means of mapping DPs to grammatical positions, and are therefore already ruled out for this reason, irrespective of computational complexity. But this mapping is tenuous at best even in real languages. Already, we have seen many examples of grammatical roles being conflated under the same case, and of the same role being split. The mapping is also subject to extensive but not unlimited variation. There remains ample room for formal factors to play a role in explaining the typology, and the fact that TSL-2 makes approximately the right cut is unlikely to be a coincidence. This is especially true when we consider the broader picture, since seemingly unrelated phenomena such as movement and long-distance phonotactics also seem to obey the same restriction.

4 The overgeneration question, or, what makes a possible case pattern?

In the preceding section, we saw how many robustly attested case patterns are TSL-2, and in contrast, how many simple yet unattested patterns are not. Now, we should also ask whether there are any instances of undergeneration and overgeneration. I am not aware of any attested case patterns which are clearly not TSL-2 (undergeneration), but we can construct certain unattested patterns which are TSL-2 (overgeneration). For instance, nominative and accusative could alternate according to the grammar below:

(35) $G^{+} = \{\rtimes \cdot \textsc{nom},\ \textsc{nom} \cdot \textsc{acc},\ \textsc{acc} \cdot \textsc{nom},\ \textsc{acc} \cdot \ltimes,\ \textsc{nom} \cdot \ltimes\}$

This pattern is only slightly different from (30) in that it utilizes a window of size 2 rather than 3, but it is just as unnatural.

However, as discussed in Section 3.1, we cannot expect every (T)SL-2 pattern to be realized in every type of linguistic dependency. Some, like iteration, are shared across many dependency types, while others show up only here and there. Indeed, although alternating patterns like the above are unnatural for case, they *can* be found in the realm of category selection—D selects N, which selects P, which selects D, and so on—as well as syllable structure (Hanson 2024b). Another set of patterns is characteristic of in feature-matching operations such as long-distance harmony and syntactic agreement (Hanson 2025b). Case exhibits yet another slice of the overall space of TSL-2 patterns.

Can we say anything more specific about the subclass of TSL-2 which is representative of case? I believe that we can. As mentioned at the outset, I have assumed that case is fundamentally a relation between one or more DPs which indicates their relative position in some domain. If this is on the right track, it is reasonable to posit that the mapping from position to case must be *monotonic*. This would rule out the example in (35), in which the first and third DPs are mapped to nominative, with the second mapped to accusative, much like the *ABA constraint in morphology. Furthermore, this would be just one of several monotonicity requirements affecting case, along with morphological syncretism

(Graf 2019), visibility (Bobaljik 2008), and locality (Poole 2022; also see Graf 2020).

Another possible formal restriction relates to iteration/spreading.[14] In previous examples, only a single case in each daughter string language could iterate. I propose that this is a hard constraint. Suppose that we had a language which is like English except that *both* nominative and accusative iterate.

(36) $G^{+} = \{\rtimes \cdot \textsc{nom}, \underline{\textsc{nom} \cdot \textsc{nom}}, \textsc{nom} \cdot \textsc{acc}, \textsc{acc} \cdot \textsc{acc}, \textsc{nom} \cdot \ltimes, \textsc{acc} \cdot \ltimes\}$

Now, whenever there are three DPs in a clause, the middle may be either nominative or accusative, with no corresponding difference in structure or meaning; if there are four, the number of possibilities only increases. This seems extremely unnatural.

Intuitively, the order of cases assigned in a given domain should be *deterministic* in the sense that only one sequence is permitted for a string of a given length, modulo category and other features. Restricting iteration to a single case slot achieves this for the present example. More generally, it could be derived if the string-to-string mapping was found to be a *input tier-based strictly local* (ITSL) function in the sense of Burness et al. (2021). This provides further motivation for revisiting case as a transduction, as suggested in Section 2.6.

In summary, there may be some unattested case patterns that are TSL-2, but there are far more that are not. Once embedded in a larger theory, the unwanted patterns can be further whittled down by the other factors—computational or otherwise—for which we have independent evidence.

5 Conclusion

In this paper, I have argued that the typology of case dependencies in syntax is explained well by the assumption that they are TSL-2. Major points of cross-linguistic variation such as presence of iteration, the structural order among cases, the visibility of DPs and domain nodes, and interactions between DPs in the same domain naturally fall out from the parameters of the model, while many unattested patterns are excluded. Case therefore joins agreement, movement, and long-distance phonotactics in exemplifying the close link between formal typology and computational complexity, further strengthening the TSL-2 hypothesis.

This does not at all mean that the investigation of the subregular complexity of case is complete. As mentioned at the outset, the typological predictions of a multi-tier system of case assignment are not nearly as clear as those for a single tier. The same can be said of movement, agreement, and the interactions among these phenomena. Working this out will require a close examination of the grammars of individual languages, similar to the study of Japanese case in Hanson (2023). We should also carefully investigate the parallels between case and agreement, particularly complex DAM effects, which have been argued by theorists as a reason for unification under the Agree operation (cf. Clem and Deal 2024). The present perspective also argues for unification, but of a very different sort: given that case and agreement are built upon the same computations and both can involve multiple DPs which are adjacent on a tier, the existence of such parallels is exactly what we expect.

In addition, the present study highlights several directions for further investigation of the formal model. These include treating case in the mapping from the dependency tree to the phrase structure tree, determining how best to reconcile the tree tier model with the c-command constraint, and exploring other formal constraints such as monotonicity which may complement subregular complexity. Now that we have a clearer understanding of the aspects of case which are explained well by the TSL-2 model, this should help with identifying good candidate factors to handle the rest.

Acknowledgments

This work was partly supported by the Institute for Advanced Computational Science at Stony Brook University. My thanks to three anonymous reviewers, whose comments led to many improvements in the presentation of this paper.

References

Baker, Mark C. (2015). *Case*. Cambridge Studies in Linguistics 146. Cambridge University Press. DOI: 10.1145/1463891.1464000.

Baker, Mark C. and Nadya Vinokurova (2010). Two modalities of case assignment: Case in Sakha. *Natural Language & Linguistic Theory* 28.3, pp. 593–642. DOI: 10.1007/s11049-010-9105-1.

[14] This proposal builds on a suggestion by a reviewer that the order of cases must not be uniformly reflexive or irreflexive. The former constraint is too strong (just one case with iteration is reflexive) and the latter is too weak, as described in the text.

Bárány, András (2017). *Person, Case, and Agreement*. Oxford University Press, Nov. doi: 10.1093/oso/9780198804185.001.0001.

Berro, Ane and Ricardo Etxepare (2017). Ergativity in Basque. *The Oxford Handbook of Ergativity*. Ed. by Jessica Coon, Diane Massam, and Lisa Demena Travis. Oxford University Press, Aug., pp. 782–806. doi: 10.1093/oxfordhb/9780198739371.013.32.

Bickel, Balthasar and Johanna Nichols (2008). Case Marking and Alignment. *The Oxford Handbook of Case*. Ed. by Andrej L. Malchukov and Andrew Spencer. Oxford University Press, pp. 304–321. doi: 10.1093/oxfordhb/9780199206476.013.0021.

Bittner, Maria and Ken Hale (1996). The Structural Determination of Case and Agreement. *Linguistic Inquiry* 27.1, pp. 1–68.

Blake, Barry J. (2001). *Case*. 2nd ed. Cambridge University Press.

Bobaljik, Jonathan (2008). Where's Phi? Agreement as a Postsyntactic Operation. *Phi Theory*. Ed. by Daniel Harbour, David Adger, and Susana Béjar. Oxford: Oxford University Press, May 1, pp. 295–328. doi: 10.1093/oso/9780199213764.003.0010.

Boston, Marisa Ferrara, John T. Hale, and Marco Kuhlmann (2010). Dependency structures derived from Minimalist Grammars. *The Mathematics of Language. 10th and 11th Biennial Conference, MOL 10, Los Angeles, CA, USA, July 28-30, 2007 and MOL 11, Bielefeld, Germany, August 20-21, 2009, Revised Selected Papers*. Ed. by Christian Ebert, Gerhard Jäger, and Jens Michaelis. Lecture Notes in Computer Science 6149. Berlin, Heidelberg: Springer, pp. 1–12. doi: 10.1007/978-3-642-14322-9_1.

Burness, Phillip Alexander et al. (2021). Long-distance phonological processes as tier-based strictly local functions. *Glossa: a journal of general linguistics* 6.1. doi: 10.16995/glossa.5780.

Butt, Miriam (2006). *Theories of case*. Cambridge University Press.

Chomsky, Noam (1959). On certain formal properties of grammars. *Information and control* 2.2, pp. 137–167. doi: 10.1016/S0019-9958(59)90362-6.

Chomsky, Noam (1981). *Lectures on government and binding: The Pisa lectures*. Berlin, New York: Walter de Gruyter.

Clem, Emily and Amy Rose Deal (2024). Dependent Case by Agree: Ergative in Shawi. *Linguistic Inquiry*, pp. 1–47. doi: 10.1162/ling_a_00529.

Daniel, Penelope (2024). Unifying Differential Argument Marking through Interpretable Features. *Proceedings of the 41st West Coast Conference on Formal Linguistics*. Somerville, MA: Cascadilla Press, pp. 105–112.

De Santo, Aniello and Thomas Graf (2019). Structure Sensitive Tier Projection: Applications and Formal Properties. *Formal Grammar: 24th International Conference*. FG 2019. Ed. by Raffaella Bernardi, Gregory Kobele, and Sylvain Pogodalla. Berlin, Heidelberg: Springer, pp. 35–50. doi: 10.1007/978-3-662-59648-7_3.

Goldsmith, John (1976). "Autosegmental phonology". PhD thesis. Cambridge, MA: Massachusetts Institute of Technology.

Graf, Thomas (2018). Why movement comes for free once you have adjunction. *Proceedings of CLS 53*. Chicago, IL: The Chicago Linguistic Society, pp. 117–136.

Graf, Thomas (2019). Monotonicity as an effective theory of morphosyntactic variation. *Journal of Language Modelling* 7.2, pp. 3–47. doi: 10.15398/jlm.v7i2.211.

Graf, Thomas (2020). Monotonicity in Syntax. *Monotonicity in Logic and Language*. Ed. by Dun Deng et al. Berlin, Heidelberg: Springer, pp. 35–53. doi: 10.1007/978-3-662-62843-0_3.

Graf, Thomas (2022a). Subregular linguistics: bridging theoretical linguistics and formal grammar. *Theoretical Linguistics* 48.3–4, pp. 145–184. doi: 10.1515/tl-2022-2037.

Graf, Thomas (2022b). Typological implications of tier-based strictly local movement. *Proceedings of the Society for Computation in Linguistics (SCiL) 2022*. Amherst, MA: University of Massachusetts Amherst, pp. 184–193. doi: 10.7275/gb65-ht31.

Graf, Thomas (2023). Subregular Tree Transductions, Movement, Copies, Traces, and the Ban on Improper Movement. *Proceedings of the Society for Computation in Linguistics (SCiL) 2023*. Ed. by Tim Hunter and Brandon Prickett. Amherst, MA: University of Massachusetts Amherst, pp. 289–299. doi: 10.7275/TK1N-Q855.

Graf, Thomas and Kalina Kostyszyn (2021). Multiple Wh-Movement is not Special: The Subregular

Complexity of Persistent Features in Minimalist Grammars. *Proceedings of the Society for Computation in Linguistics (SCiL) 2021*. Ed. by Allyson Ettinger, Ellie Pavlick, and Brandon Prickett. Amherst, MA: University of Massachusetts Amherst, pp. 275–285. DOI: 10.7275/2dfv-v219.

Graf, Thomas and Connor Mayer (2018). Sanskrit n-Retroflexion is Input-Output Tier-Based Strictly Local. *Proceedings of SIGMORPHON 2018*. Association for Computational Linguistics, pp. 151–160. DOI: 10.18653/v1/W18-5817.

Graf, Thomas and Nazila Shafiei (2019). C-command dependencies as TSL string constraints. *Proceedings of the Society for Computation in Linguistics (SCiL) 2019*. Ed. by Gaja Jarosz et al. Amherst, MA: University of Massachusetts Amherst, pp. 205–215. DOI: 10.7275/4rrx-x488.

Hanson, Kenneth (2023). A TSL Analysis of Japanese Case. *Proceedings of the Society for Computation in Linguistics (SCiL) 2023*. Ed. by Tim Hunter and Brandon Prickett. Amherst, MA: University of Massachusetts Amherst Libraries, pp. 15–24. DOI: 10.7275/xqhr-r404.

Hanson, Kenneth (2024a). A Tier-Based Model of Syntactic Agreement. *Proceedings of CLS 60*. Chicago, IL: The Chicago Linguistic Society. To appear.

Hanson, Kenneth (2024b). Strict Locality in Syntax. *Proceedings of CLS 59*. Ed. by Kutay Serova and M. K. Snigaroff. Chicago, IL: The Chicago Linguistic Society, pp. 131–145.

Hanson, Kenneth (2025a). “The Subregular Complexity of Case and Agreement”. PhD thesis. Stony Brook University.

Hanson, Kenneth (2025b). Tier-based strict locality and the typology of agreement. *Journal of Language Modelling* 13.1, pp. 43–97. DOI: 10.15398/jlm.v13i1.411.

Heinz, Jeffrey (2018). The computational nature of phonological generalizations. *Phonological Typology*. Ed. by Larry M. Hyman and Frans Plank. Phonetics and Phonology 23. De Gruyter Mouton, pp. 126–195. DOI: 10.1515/9783110451931-005.

Heinz, Jeffrey, Chetan Rawal, and Herbert G. Tanner (2011). Tier-based strictly local constraints for phonology. *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Ed. by Dekang Lin, Yuji Matsumoto, and Rada Mihalcea, pp. 58–64.

Hiraiwa, Ken (2001). Multiple Agree and the Defective Intervention Constraint in Japanese. *Proceedings of the MIT-Harvard Joint Conference (HUMIT 2000)*. Cambridge, MA: MITWPL, pp. 67–80.

Kishimoto, Hideki (2018). On exceptional case marking phenomena in Japanese. *Kobe Papers in Linguistics* 11, pp. 31–49.

Kobele, Gregory M. (2012). Eliding the derivation: A minimalist formalization of ellipsis. *Proceedings of the International Conference on Head-Driven Phrase Structure Grammar*. CSLI Publications, Oct. DOI: 10.21248/hpsg.2012.23.

Kuno, Susumu (1973). *The Structure of the Japanese Language*. Cambridge, MA: MIT Press.

Mahajan, Anoop Kumar (1990). “The A/A-bar distinction and movement theory”. PhD thesis. Cambridge, MA: Massachusetts Institute of Technology.

Malchukov, Andrej and Andrew Spencer, eds. (2008). *The handbook of case*. Oxford: Oxford University Press.

Marantz, Alec (2000). Case and licensing. *Arguments and Case: Explaining Burzio’s generalization*, pp. 11–30.

Mayer, Connor and Travis Major (2018). A challenge for tier-based strict locality from Uyghur backness harmony. *Formal Grammar 2018: 23rd International Conference*. FG 2018. Berlin, Heidelberg: Springer, pp. 62–83. DOI: 10.1007/978-3-662-57784-4_4.

McMullin, Kevin and Gunnar Ólafur Hansson (2016). Long-Distance Phonotactics as Tier-Based Strictly 2-Local Languages. *Proceedings of the Annual Meetings on Phonology*. Ed. by Adam Albright and Michelle A. Fullwood. Vol. 2. Washington, DC: Linguistic Society of America. DOI: 10.3765/amp.v2i0.3750.

Poole, Ethan (2022). Improper case. *Natural Language & Linguistic Theory* 41.1, pp. 347–397. DOI: 10.1007/s11049-022-09541-6.

Rezac, Milan, Pablo Albizu, and Ricardo Etxepare (2014). The structural ergative of Basque and the theory of Case. *Natural Language & Linguistic Theory* 32.4, pp. 1273–1330. DOI: 10.1007/s11049-014-9239-7.

Saito, Mamoru (2015). Cartography and Selection: Case Studies in Japanese. *Beyond Functional Sequence*. Oxford University Press, May, pp. 255–274. DOI: 10.1093/acprof:oso/9780190210588.003.0014.

Saito, Mamoru, T.-H. Jonah Lin, and Keiko Murasugi (2008). N'-Ellipsis and the Structure of Noun Phrases in Chinese and Japanese. *Journal of East Asian Linguistics* 17.3, pp. 247–271. DOI: 10.1007/s10831-008-9026-8.

Stabler, Edward P. (1997). Derivational minimalism. *Logical Aspects of Computational Linguistics. First International Conference, LACL '96, Nancy, France, September 23-25, 1996. Selected Papers*. Ed. by Christian Retoré. Lecture Notes in Computer Science 1328. Berlin, Heidelberg: Springer. DOI: 10.1007/BFb0052152.

Stabler, Edward P. (2011). Computational perspectives on Minimalism. *Oxford handbook of linguistic Minimalism*. Ed. by Cedric Boeckx. Oxford: Oxford University Press, pp. 617–643. DOI: 10.1093/oxfordhb/9780199549368.013.0027.

Vu, Mai Ha, Nazila Shafiei, and Thomas Graf (2019). Case assignment in TSL syntax: A case study. *Proceedings of the Society for Computation in Linguistics (SCiL) 2019*. Ed. by Gaja Jarosz et al. Amherst, MA: University of Massachusetts Amherst, pp. 267–276. DOI: 10.7275/sywz-xw23.

Witzlack-Makarevich, Alena and Ilja A. Seržant (2018). Differential argument marking: Patterns of variation. *Diachrony of differential argument marking*. Ed. by Ilja A. Seržant and Alena Witzlack-Makarevich. Studies in Diversity Linguistics 19. Berlin: Language Science Press. DOI: 10.5281/zenodo.1219168.

Yip, Moira, Joan Maling, and Ray Jackendoff (1987). Case in Tiers. *Language* 63.2, pp. 217–250.

Zaenen, A., J. Maling, and H. Thráinsson (1985). Case and Grammatical Functions: The Icelandic Passive. *Natural Language & Linguistic Theory* 3.4, pp. 441–483.

Learning Multi Tier-Based Strictly 2-Local Languages

Logan Swanson
Stony Brook University
logan.swanson@stonybrook.edu

Abstract

The class of *tier-based strictly 2-local* (TSL-2) languages has been shown to be useful in modeling patterns in both phonology (Heinz et al., 2011) and syntax (Graf, 2022a). This paper presents an algorithm for learning the intersection closure of the TSL-2 languages, the *multi*-TSL-2 (MTSL2) languages over arbitrary structures. The algorithm builds on prior work on learning a subclass of MTSL2 over trees (Swanson, 2024b), as well as insights about searching partially ordered spaces (Chandlee et al., 2019). I show that the algorithm correctly learns the MTSL2 class from a limited data sample and discuss tradeoffs with the existing approach offered by Swanson (2024b).

1 Introduction

Understanding the structure of linguistic knowledge and how humans acquire this knowledge from limited input is one of the key questions in linguistics. The field of *subregular linguistics* (Heinz, 2018, Graf, 2022a a.o.) approaches this question by seeking out formal language classes which are complex enough to represent the range of patterns found in human language, but structured enough to be efficiently learned. Creating learning algorithms for these classes not only demonstrates that they *can* be learned, but also sheds light on *how* different learning strategies behave, what kinds of time and data requirements they impose, and how they compare with human learners.

The subregular class of *tier-based strictly local* (TSL) languages has recently emerged as particularly relevant for linguists. Intuitively, TSL languages can capture dependencies which are immediately local, once a certain set of irrelevant elements is ignored (Lambert, 2023). This class is useful for analyzing many of the patterns found in human language, where long-distance dependencies are often restricted or relativized to a particular set of elements (Heinz, 2018). One such example is harmony patterns, like the sibilant harmony pattern found in Samala, a Chumash language from southern California. In Samala, sibilants in the same word must agree in anteriority, so words containing ‘s...s’ are permitted, but those with ‘*s...ʃ’ are forbidden (Hansson, 2010). The TSL grammar for this pattern consists of a tier for the sibilant sounds, $tier$ = {s, ʃ}, and constraints over that tier banning adjacent sibilants which disagree in anteriority: $constraints$ = *sʃ, *ʃs. So, a word like [sapitsolus] (‘he has a stroke of good luck’) is acceptable because its projection onto the sibilant tier, “sss”, does not contain any of these banned factors. Adding the past-tense suffix /-waʃ/, however, results in the form [ʃapitʃoluʃwaʃ]. This form is grammatical, with tier projection “ʃʃʃʃ”, but the fully faithful form [sapitsolus-waʃ] is ungrammatical, since its tier projection “sssʃ” contains the banned factor “sʃ”. Specifically, this pattern is TSL-**2**, since the constraints needed to enforce it contain at most two elements each.

TSL-2 can largely capture the typology of phonotactic patterns seen in natural language, including local dependencies, long-distance harmony, and blocking (McMullin and Hansson, 2014). In addition, a variety of syntactic patterns have been shown to belong to the parallel class of TSL-2 over tree structures, including verb agreement (Hanson, 2023b), case assignment (Hanson, 2023a), and movement patterns (Graf, 2022b). Moreover, the TSL-k languages are known to be efficiently learnable for any fixed value of k (Jardine and McMullin, 2017, Lambert, 2021). Human languages, however, typically involve many such TSL patterns operating at once, and these may interact with each other. To capture multiple TSL patterns which are active at once, the more complex class of *multi*-TSL (MTSL) is needed. An MTSL-k language is simply the intersection of one or more TSL-k languages, in other words “several TSL patterns applying at once”.

Existing work on learning MTSL is limited to an algorithm sketched by McMullin et al. (2019) and adapted by Swanson (2024b) to be provably efficient and generalizable across trees and strings. This algorithm, however, does not learn the exact class of MTSL, but rather a subclass with a few additional restrictions–restrictions which prove problematic in the realm of syntax, where they do not seem to hold on natural language data (see Section 3).

This paper introduces MTSL-BUFIA, an algorithm which learns the class of MTSL proper without these additional restrictions. I show that MTSL-BUFIA exactly identifies the class of MTSL in the limit, in the sense of Gold (1967) and has polynomial data complexity, in the sense of de la Higuera (1997). However, it has exponential time complexity in the worst case.

The remainder of this paper is organized as follows: Section 2 offers definitions for relevant terms used in this paper. Section 3 briefly recaps the existing MTSL learning work and summarizes the outstanding problems. Section 4 describes the novel MTSL learning algorithm MTSL-BUFIA, and Section 5 works through an example of it in action. Section 6 defines the representative sample for MTSL and proves that MTSL-BUFIA exactly identifies the class in the limit. Finally, Section 8 offers future directions and concludes.

2 Preliminaries

2.1 Representations and Relational Structure

An important insight from Swanson (2024b) is that MTSL-2 can be defined over any type of structure built from symbols and the relations between them. Rogers and Lambert (2019) describe this type of relational model and show some of their properties. I adopt adapted versions of three of their definitions here:

Definition 2.1 (Relational Structure)**.** A relational signature, $\mathbb{R}$, is a finite ranked alphabet of relation symbols. An $\mathbb{R}$-structure is a tuple $\mathcal{S} = \langle D, R_1^{\mathcal{S}}, R_2^{\mathcal{S}} ... \rangle$ where D is the domain and each $R_i^{\mathcal{S}}$ is an interpretation of some symbol from $\mathbb{R}$.

Definition 2.2 (Homomorphism)**.** Given $\mathbb{R}$-structures $\mathcal{S}$ and $\mathcal{S}'$ with domains D and D' respectively, a *homomorphism* from $\mathcal{S}$ to $\mathcal{S}'$ is a (total) function $h : D \rightarrow D'$ such that $\vec{a} \in R^{\mathcal{S}} \Longrightarrow h(\vec{a}) \in R^{\mathcal{S}'}$

Definition 2.3 (2-Factor)**.** Let $\mathcal{F}$ and $\mathcal{S}$ be relational structures with domains F and D respectively. $\mathcal{F}$ is a *2-factor* of $\mathcal{S}$ iff:

1. $|F| = 2$
2. $\forall x, y \in F[x \neq y \Longrightarrow \exists R \in \mathbb{R}, \vec{a} \in R^F[x, y$ both occur in $\vec{a}]]]$
3. $\exists h : F \rightarrow D$, a homomorphism

The set of all 2-factors of any structure $\mathcal{S}$ is denoted $2fac(\mathcal{S})$. Additionally, given some set of structures I, $2fac(I) := \{f \mid \exists \mathcal{S} \in I[f \in 2fac(\mathcal{S})]\}$.

2.2 Strings and Trees

These definitions are highly general, and can cover many types of structures. For the purposes of modeling language, however, the primary areas of interest are string models and tree models. I offer examples here to illustrate how these two kinds of models operate.

Example 2.1 (String Models)**.** *This example is adapted from Rogers and Lambert (2019). Let s be a string over the alphabet Σ. Let $|s|$ be the length of s. A string model for s is a structure:*

$$\mathcal{M}^{\triangleleft}(s) := \langle D^s, \triangleleft^s, P^s_{\sigma \in \Sigma} \rangle$$

Where:

D^s–is a set of natural numbers such that:

1. $0 \in D$

2. $\forall i, j \in D[i + 1 = j \Longleftrightarrow i \triangleleft j]$

$\triangleleft^s$–is the successor relation on s

P^s_σ–is the set of all positions in s at which the symbol σ occurs.

So the string "*pat*" would be modeled:

$$\mathcal{M}^{\triangleleft}(pat) := \langle D = \{0, 1, 2\}, \triangleleft = \{(0, 1), (1, 2)\}, \\ P_p = \{0\}, P_a = \{1\}, P_t = \{2\} \rangle$$

And the 2-factor corresponding to the "*pa*" substring would be modeled:

$$\mathcal{M}^{\triangleleft}(pa) := \langle D = \{0, 1\}, \triangleleft = \{(0, 1)\}, \\ P_p = \{0\}, P_a = \{1\} \rangle$$

These can trivially be related by a homomorphism $h(x) \rightarrow x$.

Example 2.2 (Tree Models)**.** *Let t be a tree over the alphabet Σ. Let $|t|$ be the number of nodes in t. A tree model for t is a structure:*

$$\mathcal{M}^{\lhd,\prec}(t) := \langle D^t, \lhd^t, \prec^t P^t_{\sigma\in\Sigma}\rangle$$

Where:

D^t–is a set of strings of natural numbers, including the empty string ε, such that:

1. $\varepsilon \in D$
2. $\forall u \in \mathbb{N}^*, j \in \mathbb{N}[uj \in D \Longrightarrow u \in D \wedge u \lhd uj]$
3. $\forall u, v \in \mathbb{N}^*[u \lhd v \Longrightarrow \exists i \in \mathbb{N}[v = ui]]$
4. $\forall u \in \mathbb{N}^*, j \in \mathbb{N}[uj \in D \Longrightarrow \forall i \in \mathbb{N}[i < j \Longrightarrow ui \in D]]$
5. $\forall u \in \mathbb{N}^*, j > 0 \in \mathbb{N}[uj \in D \Longrightarrow u(j-1) \prec uj]$
6. $\forall u, v \in \mathbb{N}^*[u \prec v \Longrightarrow \exists i, j \in \mathbb{N}, w \in \mathbb{N}^*[u = wi \wedge v = wj \wedge i + 1 = j]]$

$\lhd^t$–is the proper dominance relation on t

$\prec^t$–is the immediate left sibling relation on t

P^t_σ–is the set of all positions in t at which the symbol σ occurs.

These domain definitions essentially encode ordered addresses to each element, which is relevant for defining how tiers and 2-paths are constructed. For strings, this address encoding is a simple positional index. For trees, it is Gorn addresses, which encode the path through the tree required to reach that node from the root. Notably, these addresses are computationally easy to assign, requiring only a single traversal ($O(n)$ time) to do so.

In both of these structures, the unary relations of the form P_σ induce a *partition* on the domain elements, with each element being a member of exactly one of these sets. We can thus refer to the label of any given element $l(\alpha) = \sigma \Longleftrightarrow P_\sigma(\alpha)$.

Definition 2.4 (Tier Projection)**.** A *tier projection function* is a function $\tau()$ which takes as input a relational structure $\mathcal{S}$ with domain D and relations $\mathbb{R}$ and a set of unary relations $\mathbb{P} \subseteq \mathbb{R}$ and returns an output structure $\mathcal{S}'$ for which the following hold:

1. $\forall P \in \mathbb{P}[P = \varnothing]$
2. $\forall e \in D[\forall P \in \mathbb{P}[P(e) \Longrightarrow e \notin D']]$
3. $\forall e \in D[\nexists\, P \in \mathbb{P}[P(e)] \Longrightarrow e \in D']$
4. $\forall R \in \mathbb{R}[R \notin \mathbb{P} \Longrightarrow ((R^{\mathcal{S}}(e_1, e_2, ...e_n) \wedge \nexists\, P \in \mathbb{P}[P(e_{i<=n})]) \Longrightarrow R^{\mathcal{S}'}(e_1, e_2, ...e_n))]$

For a structure $\mathcal{S}$ over some alphabet Σ, with each element in the structure bearing exactly one symbol $\sigma \in \Sigma$ (like trees or strings) we can also write $\tau(\mathcal{S}, T) = \tau(\mathcal{S}, \mathbb{P})$, where $T = \Sigma - \{\sigma | P^{\mathcal{S}}_\sigma \notin \mathbb{P}\}$.

For strings, we use the tier projection function

$$\tau_s(\langle D, \lhd, P_\sigma\rangle_{\sigma\in\Sigma}, T) := \langle D_T, \lhd_T, P_\sigma\rangle_{\sigma\in T}$$

where:

$$D_T := \{e \in D | l(e) \in T\}$$

$$\begin{aligned}\lhd_T := \{\langle x, y\rangle | x \in D' \wedge y \in D' \wedge x < y \wedge \\ \nexists\, z \in D'[x < z < y]\}\end{aligned}$$

Intuitively, this is just removing the non-tier elements and re-stringing the elements back together in the successor relation in their original order. In other words, precedence is preserved.

For trees, we use the tier projection function

$$\tau_t(\langle D, \lhd, \prec, P_\sigma\rangle_{\sigma\in\Sigma}, T) := \langle D_T, \lhd_T, \prec_T, P_\sigma\rangle_{\sigma\in T}$$

where:

$$D_T := \{e \in D | l(e) \in T\} \cup \{\rtimes, \ltimes\}$$

$$\begin{gathered}\lhd_T := \{\langle x, y\rangle | x \in D' \wedge y \in D' \wedge \exists u \in \mathbb{N}^*[y = xu] \wedge \\ \nexists\, z \in D'[\exists u, v \in \mathbb{N}^*[z = xu \wedge y = zv]]\} \quad \cup \\ \{\langle \rtimes, x\rangle | \nexists\, u, v \in \mathbb{N}^*[u \in D' \wedge x = uv]\} \quad \cup \\ \{\langle x, \ltimes\rangle | \nexists\, u, v \in \mathbb{N}^*[u \in D' \wedge u = xv]\}\end{gathered}$$

$$\begin{gathered}\prec_T := \{\langle x, y\rangle | \\ \exists u, v, w \in \mathbb{N}^*, i < j \in \mathbb{N}[x = uiv \wedge y = ujw] \wedge \\ \exists u \in \mathbb{N}^*[u \lhd_T x \wedge u \lhd_T y \wedge \\ \nexists\, z \in D'[\exists v, w, q \in \mathbb{N}^*, i < j \in \mathbb{N}[x = viw \wedge z = vjq] \wedge \\ \exists v, w, q \in \mathbb{N}^*, i < j \in \mathbb{N}[z = viw \wedge y = vjq] \wedge \\ u \lhd_T z]]\}\end{gathered}$$

Intuitively, this can be thought of as ripping out all non-tier elements from the tree and attaching their children as additional daughters in the same slot that node was pulled out of. For notational convenience, we also stipulate designated head and foot markers, $\rtimes$ and $\ltimes$ which mark the upper and lower edges of the tree. The $\rtimes$ marker also ensures that the output is a well-formed tree by maintaining parent closure.

Definition 2.5 (2-Path)**.** The notion of *2-paths* was introduced by Jardine and Heinz (2016), and it is used to represent the idea of *intervention* in potential 2-factors. A 2-path (for a structure) is a tuple $\langle f, V \rangle$ where f is a 2-factor and V is a set of symbols which prevent that 2-factor from being present. This set of symbols which, if removed, would allow the 2-factor to be present is known as an *intervener set* for that 2-factor. The 2-path for factor f with intervener set V is denoted $\langle f, V \rangle$. For some set of structures D, we say that $paths(D)$ is all 2-paths present in all structures in D. For strings and trees, 2-paths can be computed in much the same way as tier projections, using the addresses of each element (Jardine and Heinz, 2016; Swanson, 2024b).

Table 1 demonstrates finding the 2-paths for an example tree. In this example, the tree is visualized with the labels of each domain element occupying each node, and with solid arrows indicating dominance and dashed arrows indicating siblinghood. Note that every relation which is present in this structure corresponds to a 2-path with an empty intervener set.

The remainder of this paper will not treat specific structures differently, but will instead deal exclusively with tiers, 2-factors, and 2-paths.

Definition 2.6 (MTSL)**.** An *MTSL language* is a set of structures over some alphabet Σ for which membership is defined by a grammar $G = \neg\langle f_1, T_1 \rangle \wedge \neg\langle f_2, T_2 \rangle \wedge ... \neg\langle f_i, T_i \rangle$ of 2-factor, tier pairs. A structure $\mathcal{S}$ is in the language (notated $L(G)$) iff $\nexists \langle f, T \in G \rangle [f \in 2fac(\tau(\mathcal{S}, T))]$.

3 Existing Approach: MT2SLIA

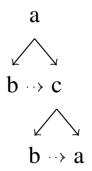
Swanson (2024b) introduces the *multi tier-based 2-strictly local inference algorithm* (MT2SLIA), a learning algorithm for a subclass of MTSL (over both trees and strings) which runs in polynomial time and data. This algorithm uses the idea that any 2-factor which is not present in the input sample must be banned on some tier. For each of these missing 2-factors, it finds all the intervener sets present in the data and uses these to construct the tier which forbids that 2-factor. To do this, elements from the smallest intervener sets are added to the hypothesized tier until each intervener set contains at least one tier element (this ensures that the 2-factor in question is in fact absent on that tier in the input data).

The MT2SLIA is provably efficient, operating in polynomial time and data with respect to the size of the target grammar. However, the concept class it learns imposes two additional requirements on the MTSL languages that it can induce. Firstly, a given structure can be banned on at most one tier. Secondly, all tier elements (for each tier) must be *independent* from all non-tier elements for that tier (meaning they can freely occur with or without each other).

This second restriction introduces issues in the realm of syntax, where many theories of syntactic structure predict that syntactic elements are *not* independent of each other in this way. In particular, theories of syntax rely on the *universal spine* (aka hierarchy of projections), which is the idea that certain functional elements always show up in a certain order in syntactic trees. This inherently introduces exactly the kind of element dependence which cannot be represented by the subclass of MTSL which the MT2SLIA learns. Indeed, Swanson (2024a) demonstrates how the universal spine disrupts the ability of the MT2SLIA to correctly learn the English that-trace effect pattern, which can be represented with a fairly simple TSL-2 analysis (Graf, 2022c). This poses a challenge to the idea that MTSL with these restrictions is a good model of possible human languages, and underscores the need for algorithms which can learn the class of MTSL proper.

4 MTSL-BUFIA

The algorithm introduced in this section combines key insights from the MT2SLIA and from the *Bottom-Up Factor Inference Algorithm* (BUFIA) introduced by Chandlee et al. (2019). Similar to the MT2SLIA, this algorithm leverages the idea that any 2-factor which is not present in the input data must be absent because it is banned on at least one tier. Additionally, 2-paths are used to inform how these forbidding tiers should be constructed: each set of interveners for a given 2-factor *must* contain at least one element from each tier on which that 2-factor is banned. Unlike the MT2SLIA, however, this algorithm also leverages the fact that the space of possible forbidding tiers is partially ordered, and can therefore be



2-paths:		$\langle b \prec a, \varnothing\rangle$
$\langle a \lhd b, \varnothing\rangle$	$\langle c \lhd b, \varnothing\rangle$	$\langle b \prec a, \{b,c\}\rangle$
$\langle a \lhd b, \{c\}\rangle$	$\langle c \lhd a, \varnothing\rangle$	$\langle b \prec b, \{c\}\rangle$
$\langle a \lhd c, \varnothing\rangle$	$\langle a \lhd a, \{c\}\rangle$	$\langle b \prec c, \varnothing\rangle$

Table 1: Example tree and 2-paths. Solid arrows indicate dominance relation, and dashed arrows indicate sibling relation.

traversed using a bottom-up breadth-first search, where the search space is pruned as constraints are located. This search strategy is inspired by Chandlee et al. (2019)'s Bottom-Up Factor Inference Algorithm (BUFIA). Using this approach, the most general tiers on which 2-factors do not appear can be exhaustively located.

4.1 Canonical Form

It is possible for multiple distinct MTSL grammars to be extensionally equivalent, i.e., generate the same language. For this reason, I provide a canonical form for MTSL grammars.

Definition 4.1 (MTSL Canonical Grammar). The Canonical Grammar for an MTSL language is a conjunction of 2-factor, tier pairs $G = \neg\langle f_0, T_0\rangle \wedge \neg\langle f_1, T_1\rangle \wedge ...$ for which the following hold:

1. $\forall \neg\langle f_i, T_i\rangle, \neg\langle f_j, T_j\rangle \in G[f_i = f_j \Longrightarrow T_i \notin T_j \wedge T_j \notin T_i]$
2. $\forall \langle f_i, T_i\rangle [L(G) \subseteq L(\neg\langle f_i, T_i\rangle) \Longrightarrow \exists T_i'[T_i' \subseteq T_i \wedge \neg\langle f_i, T_i'\rangle \in G]]$

This says that in order for an MTSL grammar to be canonical, 1) all forbidding tiers for the same factor must be incomparable, and 2) if the language generated by G obeys some constraint against a 2-factor f_i on some tier T_i, then there must be a constraint in G which bans f_i on T_i or some subset tier. The essential idea is that *all* surface-true constraints must be represented in the grammar in their most general form (i.e., on the smallest possible tier).

Lemma 1. *Any MTSL grammar is extensionally equivalent to a unique canonical MTSL grammar.*

Proof. Consider any MTSL grammar G. Suppose $L(G)$ obeys some constraint $\neg\langle f_i, T_i\rangle \notin G$. We can then construct a grammar $G' = G \wedge \neg\langle f_i, T_i\rangle$. Since $\neg\langle f_i, T_i\rangle$ is true over $L(G)$ and the two grammars differ only in the presence of this constraint, $L(G') = L(G)$. This process can be repeated to yield a G' for which $\nexists\ \neg\langle f_i, T_i\rangle[\neg\langle f_i, T_i\rangle \notin G' \wedge L(G') \subseteq L(\neg\langle f_i, T_i\rangle)]$ and $L(G') = L(G)$. G' then meets requirement 2) of a canonical grammar, since every surface-true constraint of $L(G)$ is present in G'.

Then, if $\exists \neg\langle f_i, T_i\rangle, \neg\langle f_i, T_j\rangle \in G'[T_i \subset T_j]$, we can construct $G'' = G' - \neg\langle f_i, T_j\rangle$. Since any form which contains the 2-factor f_i on the T_j tier will necessarily also contain f_i on the T_i tier, any langauge which obeys $\neg\langle f_i, T_i\rangle$ must also obey $\neg\langle f_i, T_j\rangle$. Therefore $L(G'') = L(G')$. Additionally, G'' still meets requirement 2) since the subset tier is always preserved. Once again, we can repeat this process until we reach a G'' for which $\nexists \neg\langle f_i, T_i\rangle, \neg\langle f_i, T_j\rangle \in G'[T_i \subset T_j]$ (i.e., requirement 1) holds) and $L(G'') = L(G') = L(G)$. G'' thus meets both requirements for a canonical MTSL grammar and is equivalent to G, meaning any MTSL grammar is equivalent to some canonical MTSL grammar.

Next, consider any canonical MTSL grammar G, and consider some other canonical MTSL grammar G' such that $L(G') = L(G)$. If $G' \neq G$, it must be the case that either G' contains some constraint which is not in G, or that G contains some constraint which is not in G'. Suppose G' contains a constraint $\neg\langle f_i, T_i\rangle$ which is not in G. Since $L(G) = L(G')$, $L(G)$ must obey this constraint. Since G is canonical, it must then (by requirement 2) contain some constraint $\neg\langle f_i, T_j\rangle$ such that $T_j \subset T_i$. But then, since G' is also canonical and must obey $\neg\langle f_i, T_j\rangle$, G' must contain some constraint $\neg\langle f_i, T_k\rangle$ such that $T_k \subset T_j$. But then by transitivity, $T_k \subset T_i$, meaning that G' violates requirement 1 and cannot be canonical. Thus, G' cannot contain any constraints which are not in G, and $G' \subseteq G$. Suppose G contains some constraint $\neg\langle f_i, T_i\rangle$ not in G'. By the same logic, G' must then contain some $\neg\langle f_i, T_j\rangle$, and G must contain some $\neg\langle f_i, T_k\rangle$ such that $T_k \subset T_j \subset T_i$. This means G violates requirement 1 and cannot be canonical. Thus, $G \subseteq G'$, and $G' = G$, meaning any canonical MTSL grammar is unique. □

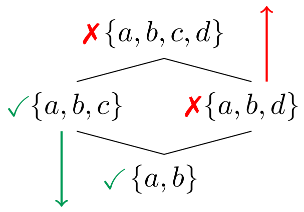


Figure 1: Hierarchy of possible forbidding tiers for ab over $\Sigma = \{a,b,c,d\}$. If ab is absent from a particular tier (denoted by ✗), it must be absent from all superset tiers. If it is present on a tier (denoted ✓) it is necesarily present on all subset tiers.

4.2 BUFIA

To find the forbidding tier(s) for each 2-factor, MTSL-BUFIA uses a bottom-up, breadth-first search of the partially ordered set of possible tiers. This approach is inspired by BUFIA (Chandlee et al., 2019; Rawski, 2021), an algorithm designed to find the most general constraints over this type of partially ordered search space.

It is easy enough to see that the set of possible tiers (i.e., the powerset of Σ) is partially ordered under the subset relation. Furthermore, the set of constraints introduced by these possible tiers (for the same 2-factor) obeys that same partial ordering. For example, consider a toy language with alphabet $\Sigma = \{a,b,c,d\}$ with just one constraint: $\langle ab, \{a,b,d\}\rangle$. So strings like $adbcca$ and $bcaccdb$ (with tier projections $adba$ and $badb$) are in the language, but $daccbda$ is out, since its projection ($dabda$) includes the substring ab. It is immediately clear that no string in this language can contain ab on the superset tier of $\{a,b,c,d\}(=\Sigma)$, since this would necessarily mean ab is present on the $\{a,b,d\}$ tier as well. More generally, a factor f_i is absent from a tier T_i if and only if every possible occurrence of f_i is precluded by the intervention of some element from T_i. If $T_i \subset T_j$, then any factor absent from T_i will also be absent from T_j, since the intervention of an element from T_i entails the intervention of an element from T_j (all elements of T_i are in T_j). Therefore, any language which obeys the constraint $\langle f_i, T_i\rangle$ obeys all other constraints $\langle f_i, T_j\rangle$ where $T_i \subset T_j$. Figure 1 visualizes this property on the set of possible forbidding tiers for this toy language rooted at the $\{a,b\}$ tier.

These entailment relationships between possible constraints are exactly what BUFIA uses for learning. Starting from the "bottom", or most general constraint, BUFIA proceeds upwards layer by layer, looking for constraints which are surface-true. When it finds them, it adds them to the grammar and prunes away the section of the search space above that new constraint. Chandlee et al. (2019) also outline several useful learning guarantees for this algorithm. Namely, BUFIA is guaranteed to find only and all incomparable constraints which are consistent with the input data, and it is guaranteed to find the *most general* such constraints. These are exactly the properties required to construct canonical MTSL grammars.

4.3 Algorithm

MTSL-BUFIA, defined in Algorithm 1, operates as follows: First, it finds all possible 2-factors which are *absent* from the input sample. This absence is an indicator that each of these 2-factors is forbidden on some tier(s). The algorithm then iterates through these missing 2-factors and computes the intervener sets for each. Once intervener sets are collected, the algorithm begins its bottom-up search for the smallest tier(s) on which that 2-factor is missing. It begins by looking at the tier consisting only of the symbols present in the 2-factor itself, and proceeds breadth-first to increasingly larger possible tiers. To determine whether a 2-factor is absent from a tier projection t, it must be the case that there is no intervener set i for that 2-factor such that $i \cap t$ is empty. In other words, each intervener set must contain *some* tier element, otherwise the 2-factor in question is present on that tier.

Any time the search encounters a tier where the 2-factor is not present in the data over that tier, the 2-factor, tier pair are added to the grammar, and *all* supersets of that tier are removed from the search space. In this way, the search space of possible tiers is pruned as the search proceeds.

Once this bottom-up search has been conducted for every 2-factor, the final grammar is returned.

5 Example

To demonstrate this algorithm in action, I will use a toy example language in which local and long-distance dependencies interact (this example is string-based, but recall that this can be smoothly generalized to tree structures as described in Section 2). The string language we will consider is defined by the following regular expression: $((ab^+c)|(eb^+d))^*$. This is all strings consisting of any number of sequences of one a followed by one or more bs followed by one c interspersed with any number of sequences of one e followed by one or

Algorithm 1 MTSL-BUFIA

Data: Positive sample D

Result: Grammar G, a conjunction of ⟨2-factor, forbidding tier⟩ pairs.

```
G := { }
B := $2fac(\Sigma^*) - 2fac(D)$
foreach $f := \rho_1 R \rho_2 \in B$:
  $S := \{I$ for $\langle x, I\rangle \in paths(D)$ where $x = f\}$
  Q := [{ $\rho_1, \rho_2$ }]
  while Q ≠ []:
    $T' =$ Q.$pop()$
    if $\exists \langle f, T\rangle \in G[T \subseteq T']$:
      continue
    if $\exists V \in S[T \cap V = \varnothing]$:
      Q.$append(NextSupersets(T'))$
    else:
      G = G$\cup\neg\langle f, T'\rangle$
return $\bigwedge_{c \in G} c$
```

more bs followed by one d. So $abbbcebd$, $ebbbbbd$, and $abcebdebdabbc$ are all valid strings in the language, but abd and $edac$ are not. This involves local restrictions, for example a and e can only be followed by b, but also non-local dependencies which interact: b can be followed by c only in the case where the first b in its sequence was preceded by a, and similarly with d and e. This violates the "tier-element independence" requirement of the MT2SLIA, which stipulates that each element on each tier must be independent from (i.e., not bound to always occur next to) each non-tier element. However, in this language it is critical to enforce constraints like "a cannot be followed by d unless there is a e in between them". In this case, a, d, and e must all be tier elements. However, b must be off the tier, since its presence as an intervener between a and d does **not** license the structure. But none of these tier elements are independent from b: each is required to either precede or follow a b. Therefore, this language would not be learnable by the MT2SLIA. As we will see, however, the BUFIA-MTSL learner has no problem.

Suppose we are given the data presented in Example 1.

$$\begin{aligned} D = abbbc, ebd, abcebdebdebd,& \\ ebdebdabcabcabbcebd,& \\ abcebdabcebdabcebd& \end{aligned} \tag{1}$$

In the first step, the algorithm will compute the 2-factors which are missing from this sample. These are given in the first column of Table 2.

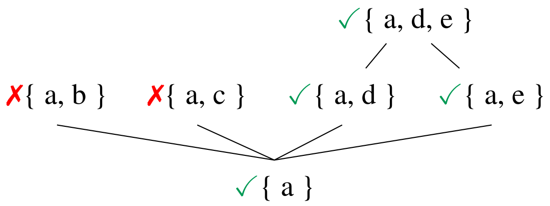


Figure 2: Caclulation of tiers for aa 2-factor.

Then, for each missing 2-factor, the intervener sets will be computed. This is column 2 of Table 2.

Finally, the algorithm will conduct a bottom-up search of the possible tiers. The search procedure is diagrammed in Figures 2 and 3, for the 2-factors aa and db respectively. If each intervener set contains at least one tier element, the factor is missing on that tier, and the tier is added as a constraint (notated by the ✗ symbol). For example, in Figure 2, the $\{a, b\}$ tier is added as a forbidding tier for the aa 2-factor because each intervener set contains a b. This is exactly as expected–aa cannot be present on the $\{a, b\}$ tier because every a *must* be followed by at least one b. Similarly, each a must be followed by a c before another a can be present, and so this 2-factor is also banned on the $\{a, c\}$ tier.

For the db 2-factor, meanwhile, there must be *either* an a or an e intervening. However, projecting just one of these to the tier would require that symbol to *always* be present. For example, if db were banned on the $\{a, b, d\}$ tier, the (licit) sequence $ebdebd$ would be banned, since its tier projection would be $bdbd$, which contains the db 2-factor.

If a tier is not a forbidding tier (indicated with ✓), then the search continues upwards. Notice, however, that supersets of previously added forbidding tiers are *not* searched, so the final set $\{a, b, c, d, e\}$ (i.e., Σ) is never reached in Figure 3, and the search in Figure 2 can get no higher than $\{a, d, e\}$, since any other tiers that could be searched would be supersets of one of the existing forbidding tiers, $\{a, b\}$ or $\{a, c\}$.

The final forbidding tier(s) for all 2-factors are given in column three of Table 2.

In some ways, the grammar found by this algorithm could be considered "inefficient", since it contains some constraints that are not strictly needed to enforce the target pattern. For example, there is no need for the constraint $\langle ad, \{a, b, d\}\rangle$, which enforces that a b must occur between a and

2-factor	interveners	tier(s)
aa	{ b, c }, { b, c, a }, { b, c, e, d }, { b, c, e, d, a }	{ a, b }, { a, c }
ac	{ b }, { a, b, c }, { b, c, e, d, a }	{ a, c, b }
ad	{ b, c, e }, { b, c, e, a }, { b, c, e, d }, { a, b, c, d, e }	{ a, d, b }, { a, d, c }, { a, d, e }
ae	{ b, c }, { b, c, a }, { b, c, a, e, d }	{ a, e, c }, { a, e, b }
ba	{ c }, { d }, { b, c }, { b, d }, { c, a, b }, { d, e, b }, { c, e, b, d }, { d, a, b, c }, { a, b, c, d, e }	{ a, b, c, d }
be	{ c }, { d }, { b, c }, { b, d }, { b, d, e }, { a, b, c }, { a, b, c, d }, { b, c, d, e }, { a, b, c, d, e }	{ b, c, d, e }
cb	{ a }, { e }, { a, b }, { b, e }, { a, b, c }, { e, b, d }, { a, b, c, d }, { e, b, d, a }, { a, b, c, d, e }	{ a, b, c, e }
cc	{ a, b }, { a, b, c }, { a, b, e, d }, { a, b, c, d, e }	{ c, a }, { c, b }
cd	{ e, b }, { e, b, d }, { a, b, c, e }, { a, b, c, d, e }	{ c, d, b }, { c, d, e }
db	{ a }, { e }, { a, b }, { b, e }, { a, b, c }, { e, b, d }, { a, b, c, e }, { a, b, d, e }, { a, b, c, d, e }	{ a, b, d, e }
dc	{ a, b }, { a, b, c }, { a, b, d, e }, { a, b, c, d, e }	{ d, c, a }, { d, c, b }
dd	{ e, b }, { e, b, d }, { e, b, a, c }, { a, b, c, d, e }	{ d, b }, { d, e }
ea	{ b, d }, { b, d, e }, { a, b, c, d, e }	{ a, b, e }, { a, d, e }
ec	{ b, d, a }, { b, d, a, e }, { b, d, a, c }, { a, b, c, d, e }	{ e, c, a }, { e, c, b }, { e, c, d }
ed	{ b }, { e, b, d }, { a, b, c, d, e }	{ e, d, b }
ee	{ b, d }, { b, d, e }, { b, d, a, c }, { a, b, c, d, e }	{ e, b }, { e, d }

Table 2: Missing 2-factors, intervener sets, and calculated tiers.

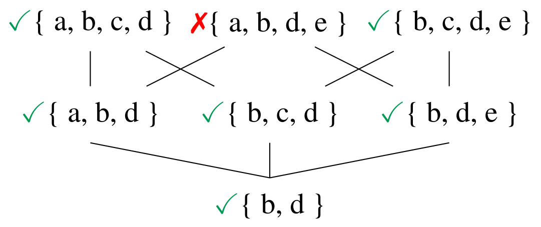


Figure 3: Calculation of tiers for *be* 2-factor.

d, since the other constraints already enforce that b is the only symbol which can follow a in the first place. This, however, is part of the definition of canonical grammar: if a constraint is true it *must* be represented in the grammar. This allows the grammar to be more universal by obviating the question of *which* constraint should be used for a given purpose.

6 Exact Identification in the Limit with Polynomial Data

The approach adopted here follows the tradition of grammatical inference, using the exact identification in the limit learning paradigm (Gold, 1967) with a polynomial bound on data (de la Higuera, 1997). Under this paradigm, the learner is presented with a *positive text* t of examples drawn from the target language L. The first n items in t are denoted t_n. The examples can appear in any order, and may repeat, but for any given structure s in L, it is guaranteed that there is some finite $n \in \mathbb{N}$ such that $s \in t_n$.

In this sense, it is assumed that the learner will eventually receive a *representative sample* which characterizes L with respect to MTSL-BUFIA. Once this sample has been seen, the learner must 1) converge on the correct grammar and 2) not deviate from this grammar when more positive data is presented.

Definition 6.1 (Grammar Size)**.** For any MTSL-k grammar G, its size is defined by:

$$|G| := \sum_{\neg\langle f,T\rangle \in G} k + |T|$$

Definition 6.2 (Representative Sample)**.** For a MTSL language L over alphabet Σ whose grammar is $G = \neg\langle f_0, T_0\rangle \wedge \neg\langle f_1, T_1\rangle \wedge \ldots \neg\langle f_i, T_i\rangle$, a set D of structures is a *representative sample* iff all of the following hold:

1. $\forall x \in 2fac(\Sigma)[x \notin \{f : \exists\neg\langle f,T\rangle \in G\} \Longrightarrow x \in 2fac(D)]$

2. $\forall\neg\langle f,T\rangle \in G[\forall\sigma \in T - symbols(f)[$

$\exists\langle f, V\rangle \in 2paths(D)[\sigma \in V \wedge \neg\exists\sigma' \in V[\sigma' \in T \wedge \sigma' \neq \sigma]]]$

3. $\forall x \in \{f : \exists\neg\langle f,T\rangle \in G\}[\neg\exists T'[(\langle x,V\rangle \in 2paths(D) \Longrightarrow V \cap T' \neq \varnothing) \wedge (\neg\exists\neg\langle x,T\rangle \in G[T \notin T'])]]$

This essentially states that 1) a representative sample must contain all 2-factors which are not banned on any tier, 2) for each banned 2-factor and each tier on which it is banned, every tier element that is not part of the 2-factor itself must be present in an intervener set (for that same 2-factor) which contains no other tier elements, and 3) for each banned 2-factor there can be no set of elements which is represented (i.e., at least one element of this set is present) in every intervener set (for that 2-factor) but which is not itself a superset of a tier on which that 2-factor is banned.

Lemma 2. *For a MTSL language L, the size of the representative sample D for L is polynomial in the size of G for any MTSL grammar G of L.*

Proof. The first condition requires that all 2-factors which are not banned on any tier are present in the sample. The number of possible 2-factors will vary with the type of structure being used, but it will be limited to $c \cdot |\Sigma|^2$ where c is the number of possible relations (for trees this is two, while for strings it is just one). Embedding these 2-factors in well-formed structures (to form a data sample) may require the addition of extra "connecting" symbols (such as in the case for the *sibling* relation over trees, where an additional parent node is needed). Assuming the number of these connecting symbols required is bounded by another constant k (which it is for both trees and strings), the total size will be $ck \cdot |\Sigma|^2$.

The second condition stipulates that each tier element of each restricting tier for each banned 2-factor must appear in some intervener set for that 2-factor without any other tier elements. Ensuring these intervener sets requires structures containing the two symbols present in the 2-factor, plus the tier element, plus any additional connecting symbols necessitated by the type of structure. These will contain some constant number of symbols (just 3 in the case of strings and 3 or 4 in the case of trees). Since one such structure is needed for each tier symbol, the amount data required to satisfy condition two is linear in the size of the grammar (since this is just the number of total tier symbols plus twice the number of 2-factors).

The third condition is about making sure that there are sufficiently small intervener sets represented in the sample (i.e., those uncluttered by many non-tier elements). Constructing the smallest possible sample which fufills conditions 1 and 2 ensures this condition without the need for additional data.

Therefore, the space complexity of the representative sample is $O(|\Sigma|^2 + |G|)$ (where $|G|$ is the number of total symbols present in G). Assuming all alphabet symbols are used in the grammar, this is polynomial in the size of the grammar. □

Lemma 3. *Given any superset I of a representative sample D for a MTSL language L with canonical grammar* $G = \neg\langle f_0, T_0\rangle \wedge \neg\langle f_1, T_1\rangle \wedge ... \wedge \neg\langle f_i, T_i\rangle$*, MTSL-BUFIA will return a grammar* $G' = G$*.*

Proof. Consider the set B of all the 2-factors which are banned on any tier in G. Since I contains only valid structures in L, these 2-factors must all be absent from I. By definition of a representative sample, all other possible 2-factors over Σ must be present in D, and therefore also in I. B is therefore equivalent to the set which will be iterated over in the outer loop. Each $f'_i \in B$ must therefore be associated to one or more pairs $\neg\langle f'_i, T_1\rangle, \neg\langle f'_i, T_2\rangle ... \neg\langle f'_i, T_k\rangle \in G$. First, we show that every constraint in G' is in G. Suppose G' contains a forbidding tier for some f'_i, $\neg\langle f'_i, T'_i\rangle$ which is *not* one of those in G. It must then be the case that each intervener set for f'_i, contains at least one element from T'_i, otherwise this pair would not get added to the grammar. Because $D \subset I$, the set of intervener sets for f'_i, S must contain all intervener sets for f'_i which are present in D. By requirement 3 for representative samples, it must then be the case that there is some constraint $\neg\langle f_j, T_j\rangle \in G$ such that $T_j \subset T'_i$. Since the queue (Q) grows breadth-first, the algorithm will consider T_j as a forbidding tier for f'_i before it considers T'_i. Since $\neg\langle f'_i, T_j\rangle$ holds on all the data, each intervener set in I for f_i must contain at least one element of T_j. Therefore, the algorithm will add $\neg\langle f'_i, T_j\rangle$ to the grammar, and when it considers T'_i this tier will be discounted since $T_j \subset T'_i$ is already present as a forbidding tier for f'_i. Therefore, G' cannot contain any forbidding tiers for any f'_i which are not contained in G.

Next, we show that every constraint in G is in G'. Suppose G contains a forbidding tier for f_i, $\neg\langle f_i, T_i\rangle$ which is not contained in G'. In order for T_i to *not* be added to G' in the inner loop, it must be the case that *either* a forbidding tier $\neg\langle f_i, T_j\rangle$ is already in G', where $T_j \subset T_i$, *or* there is some intervener set V which contains no elements of T_i. As established above, the only constraints which will be added to G' are those which are also present in G. Therefore, $\neg\langle f_i, T_j\rangle \in G' \Longrightarrow \neg\langle f_i, T_j\rangle \in G$. However, if G were to contain $\neg\langle f_i, T_j\rangle$ and $\neg\langle f_i, T_i\rangle$, where $T_j \subset T_i$ it would not be a canonical grammar by the definition in Section 4. Since $\neg\langle f_i, T_i\rangle \in G$, and I consists of only positive data generated by G, there can be no such intervener set V such that $V \cap T_i = \varnothing$, since this would mean f_i was in fact present on the tier T_i, thereby directly violating the constraint. Therefore, T_i will be added to G' during the inner loop, and G cannot contain any constraints not contained in G'.

Since $G \subset G'$ and $G' \subset G$, the two are equal: $G' = G$. □

Theorem 1. *For any MTSL language L, MTSL-BUFIA identifies the canonical grammar for L in the limit using polynomial data.*

Proof. From Lemmas 2 and 3. □

7 Time complexity

In the worst case, MTSL-BUFIA runs in exponential time. To see why, consider the time required for each step: The time complexity of computing 2-paths is $O(n^2)$ for strings (Jardine and Heinz, 2016) and $O(n^4)$ for trees (Swanson, 2024b). Then, for each missing 2-factor, the relevant tier(s) must be found. There are at most $|\Sigma|^2$ (or $2 \cdot |\Sigma|^2$ for trees) 2-factors to consider. Any missing 2-factor could be banned only on the segmental tier (i.e. $T = \Sigma$), and in this case the algorithm would have to traverse all possible tiers to discover this, with a time complexity of $2^{|\Sigma|}$ for each factor. This yields a time complexity of $O(|\Sigma|^2 \cdot 2^{|\Sigma|})$ The final step of traversing the powerset of the alphabet introduces exponential complexity to this algorithm, a disadvantage against the MT2SLIA which is guaranteed to run in polynomial time.

The utility of BUFIA, however, does not come from its worst-case performance but rather from its ability to turn sparsity in the input data to its advantage. As in the example given in Section 5, when the input data obeys highly general constraints, BUFIA is able to prune away large chunks of the search space along the way, enabling it to tractably search very large spaces under the right conditions. Indeed, BUFIA has been successfully implemented and used to analyze natural-language scale data (Swanson et al., 2025). So although MTSL-BUFIA to some degree trades a tighter concept class for a worse time complexity, its exponential worst-case bound may not indicate intractability on natural language data, which is highly marked by sparsity. In the case of subregular syntax, many of the phenomena which have been analyzed as TSL require tiers with three or fewer symbols (ie highly restrictive), which would play into BUFIA's ability to prune the search space.

8 Conclusion

This paper introduces MTSL-BUFIA, an algorithm which exactly identifies the class of MTSL in the limit from a polynomially-sized data sample. This algorithm avoids the shortcoming of previous MTSL learning algorithms which placed additional problematic restrictions on the concept class which could be learned. The greater expressiveness of patterns that MTSL-BUFIA can induce comes at the cost of a polynomial runtime guarantee, but this does not mean the algorithm is universally intractable. The BUFIA approach does best (in terms of time complexity) with sparse data which can be captured by more general constraints.

Future work in this area involves testing MTSL-BUFIA on natural-language data samples, which would not only allow exploration of its average-case time performance (on the types of data found in human language), but also give insight into whether natural language typically furnishes the requisite representative sample to learn the target patterns.

Additionally, MTSL-BUFIA is suggestive of several possible extensions which could be fruitful to explore. Lambert (2021) provides a method for online learning of TSL-k languages, and it might be possible to adapt MTSL-BUFIA along those same lines. While MTSL-BUFIA is limited to the MTSL-2 languages, the concept of intervener sets can be extended to larger k-factors, allowing for the possibility of learning MTSL-k. With larger k values, the set of possible 2-factor, tier pairs is itself partially ordered and can be traversed in the same fashion to yield a grammar with mixed sizes of k-factors. Another useful aspect of BUFIA is that it is well-suited to feature-based representations, so this approach could also be easily extended to operate over features rather than segments.

Finally, another area of future exploration is into additional concept classes between MTSL and the more restricted class learned by the MT2SLIA. For example, what would it mean to enforce *only* constraint-uniqueness or *only* tier-element independence? Are there different restrictions on MTSL that are more appropriate for natural language? The work presented in this paper offers a foundation on which to continue probing these open questions.

References

Jane Chandlee, Remi Eyraud, Jeffrey Heinz, Adam Jardine, and Jonathan Rawski. 2019. Learning with partially ordered representations. In *Proceedings of the 16th Meeting on the Mathematics of Language*, pages 91–101, Toronto, Canada. Association for Computational Linguistics.

Colin de la Higuera. 1997. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27:125–138.

E Mark Gold. 1967. Language identification in the limit. *Information and control*, 10(5):447–474.

Thomas Graf. 2022a. Subregular linguistics: bridging theoretical linguistics and formal grammar. *Theoretical Linguistics*, 48(3-4):145–184.

Thomas Graf. 2022b. Typological implications of tier-based strictly local movement. In *Proceedings of the Society for Computation in Linguistics 2022*, pages 184–193.

Thomas Graf. 2022c. Typological implications of tier-based strictly local movement. In *Proceedings of the Society for Computation in Linguistics 2022*, pages 184–193.

Kenneth Hanson. 2023a. A TSL Analysis of Japanese Case. *Proceedings of the Society for Computation in Linguistics*, 6(1):15–24.

Kenneth Hanson. 2023b. A computational perspective on the typology of agreement.

Gunnar Ólafur Hansson. 2010. *Consonant harmony: Long-distance interactions in phonology*, volume 145. Univ of California Press.

Jeffrey Heinz. 2018. The computational nature of phonological generalizations. *Phonological typology, Phonetics and Phonology*, pages 126–195.

Jeffrey Heinz, Chetan Rawal, and Herbert G. Tanner. 2011. Tier-based strictly local constraints for phonology. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 58–64, Portland, Oregon, USA. Association for Computational Linguistics.

Adam Jardine and Jeffrey Heinz. 2016. Learning tier-based strictly 2-local languages. *Transactions of the Association for Computational Linguistics*, 4:87–98.

Adam Jardine and Kevin McMullin. 2017. Efficient learning of tier-based strictly k-local languages. In *International conference on language and automata theory and applications*, pages 64–76. Springer.

Dakotah Lambert. 2021. Grammar interpretations and learning tsl online. In *International Conference on Grammatical Inference*, pages 81–91. PMLR.

Dakotah Lambert. 2023. Relativized adjacency. *Journal of Logic Language and Information*, 32:707–731.

Kevin McMullin, Alëna Aksënova, and Aniello De Santo. 2019. Learning phonotactic restrictions on multiple tiers. *Proceedings of the Society for Computation in Linguistics*, 2(1):377–378.

Kevin McMullin and Gunnar Ólafur Hansson. 2014. Long-distance phonotactics as tier-based strictly 2-local languages. In *Proceedings of the annual meetings on phonology*.

Jonathan Rawski. 2021. *Structure and Learning in Natural Language*. Ph.D. thesis, Stony Brook University.

James Rogers and Dakotah Lambert. 2019. Some classes of sets of structures definable without quantifiers. In *Proceedings of the 16th Meeting on the Mathematics of Language*, pages 63–77.

Logan Swanson. 2024a. Representing syntax for learning.

Logan Swanson. 2024b. Syntactic learning over tree tiers. In *Proceedings of the ESSLLI 2024 Student Session*.

Logan Swanson, Jeffrey Heinz, and Jonathan Rawski. 2025. Phonotactic learning with structure, not statistics. Submitted to Linguistic Inquiry remarks & replies.

A Framework for Learning Phonological Maps as Logical Transductions

Tatevik Yolyan
Department of Linguistics / Rutgers University / New Brunswick, NJ
tatevik.yolyan@rutgers.edu

Abstract

This paper presents an approach to learning a subclass of phonological maps that can be expressed with non-recursive quantifier-free logical transductions within the framework of Boolean Monadic Recursive Schemes (BMRS). Building on previous work on phonological learning with partially-ordered hypothesis spaces, this paper shows that similar structures can be used in learning phonological maps when these maps are represented by logical transducers over unconventional string models. This paper demonstrates how a model-theoretic framework can be used for computational learning of *phonological generalizations*.

1 Introduction

Input Strictly Local (ISL) functions characterize phonological maps in which sounds undergo change based on local information in the input (Chandlee, 2014; Chandlee and Heinz, 2018). In other words, every ISL function has a parameter k such that for every x in the input string, the information needed to determine the output of x is within some window of size k around x. Consider the postnasal voicing map below from Zoque, in which voiceless stops become voiced when they are immediately preceded by a nasal (Wonderly, 1951). The input-output pair of words in Figure 1 illustrates why this map is ISL-2.

pama	'clothing'	m**b**ama	'my clothing'
tatah	'father'	n**d**atah	'my father'
kama	'cornfield'	ŋ**g**ama	'my (corn)field'
hayah	'husband'	nhayah	'my husband'

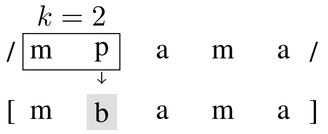


Figure 1: Postnasal voicing in Zoque is ISL-2.

The ISL class of functions is learnable from positive data; for every ISL function f, it is possible to learn a finite state transducer which computes f from a characteristic sample $S \subseteq f$ of input-output pairs of strings (Chandlee, 2014; Chandlee et al., 2014). The purpose of this paper is to revisit learning from a model-theoretic perspective, where the goal is to learn ISL functions as *logical* transductions from pairs of input-output string *models*. While finite state transducers traditionally operate over string representations, logical transducers operate over more enriched representations. This paper uses 'unconventional' string models (Strother-Garcia et al., 2016; Chandlee et al., 2019) which represent strings with phonological features.

One of the main ideas presented in this paper is that we can learn logical transductions from input-output pairs of string models by using partially-ordered hypothesis spaces. Similar spaces have been used for various learning problems within phonology (Rawski, 2021; Chandlee et al., 2019; Tesar, 2013; Heinz et al., 2012; Heinz, 2010). A common theme among them is that a partially-ordered hypothesis space encodes entailment relations that are relevant for learning. These entailments allow a large space to be pruned efficiently with a small number of data points. This paper shows that a similar approach can be applied to the problem of learning logical transducers. This paper culminates with a demonstration of how a logical transducer for postnasal voicing can be learned using a very small number of input-output pairs.

This work closely relates to previous work on model-theoretic learning of phonotactic constraints from Chandlee et al. (2019) and Rawski (2021). More specifically, this paper shows that the problem of learning the environment that triggers a particular feature to undergo change employs the same structure as learning banned k-factors in a grammar. In the case of postnasal voicing, for example, Zoque has a phonotactic constraint *[+nas][-voi,-

cont] which says a voiceless stop cannot be immediately preceded by a nasal. This constraint also expresses the *environment* where voicing takes place: if an underlying form violates *[+nas][-voi,-cont], the second sound undergoes voicing in order to repair the violation. The partially-ordered hypothesis space used to learn the surface constraint *[+nas][-voi,-cont] can therefore be adapted to learning the environment where a voiceless sound becomes voiced. As such, the procedure presented here is a means of *lifting* phonotactic learning to learning phonological maps via logical transductions.

A broader contribution of this paper is a demonstration of how this model-theoretic approach can be used to learn phonological generalizations from a sample of underlying and surface form pairs. To that end, the style of this paper is mainly expository, and the learning problem is abstract rather than formally-defined. This paper moreover focuses on length-preserving and order-preserving ISL functions because they are a natural starting point for motivating the larger research program of model-theoretic phonology. The method introduced here can be extended to more complex function classes, as well as more complex representations.

The structure of this paper is as follows. Section 2 provides the relevant background and definitions of string models, logical transductions, subfactors, and partially-ordered sets (posets). Section 3 discusses previous work on phonological learning with posets, with emphasis on the Bottom-Up Feature Inference Algorithm (BUFIA; Chandlee et al., 2019). Section 4 presents the learning goal and procedure, and Section 4.4 illustrates these ideas using postnasal voicing as a case study. Future extensions of this work are discussion in Section 5.

2 Preliminaries

Given an alphabet Σ, a string/word is a finite concatenation of symbols in Σ. The set of all such strings, along with the empty string λ, is denoted Σ^*. For each $w \in \Sigma^*$, we let w_i denote the i^{th} character in w, and $|w|$ denote the length of w. This section presents definitions of relational structures and string models which are used to represent words in Σ^*, and ultimately functions over Σ^*. These structures are then enhanced with phonological features over which phonological maps are defined.

2.1 String Models

Definition 1. *A **signature** is a collection of relation symbols $\{R_i\}_{i\leq m}$, and function symbols $\{f_i\}_{i\leq n}$.*[1]

Definition 2. *Given a signature $\mathcal{S}$, an $\mathcal{S}$-**structure***

$$\mathcal{M} = \langle D, \{R_i^{\mathcal{M}}\}_{i\leq m}, \{f_i^{\mathcal{M}}\}_{i\leq n}\rangle$$

consists of a domain D, a relation $R_i^{\mathcal{M}} \subseteq D^k$ for each k-ary relation symbol R_i, and a function $f^{\mathcal{M}} : D^k \rightarrow D$ for every k-ary function symbol f. $Struc(\mathcal{S})$ denotes the set of $\mathcal{S}$-structures.

Structures are built over a signature by specifying a domain and interpretations for the relation and functions symbols. Every word in Σ^* is associated with a particular kind of structure called a *string model* built over a signature consisting of unary predicates $\{\rtimes, \ltimes, P_\sigma\}_{\sigma\in\Sigma}$ and successor and predecessor functions. An alphabet and a signature over that alphabet are often designated by the same symbol; the string models built over an alphabet Σ are called Σ-structures.

This paper looks at a specific type of string model which we refer to here as *monadic* string models, following Chandlee and Lindell (forth.).

Definition 3. *For an alphabet Σ and word $w \in \Sigma^*$, a **monadic string model** for w is a Σ-structure*

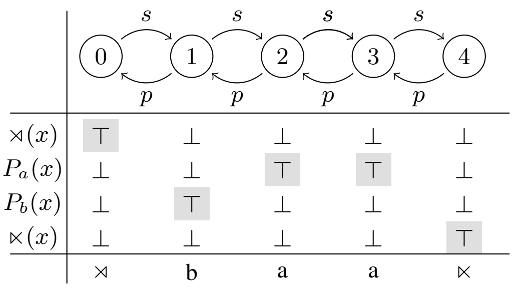
$$\mathcal{M}(w) := \langle D, \rtimes, \ltimes, P_\sigma, s, p\rangle_{\sigma\in\Sigma}$$

where $D = \{0, \ldots, |w| + 1\}$ is a set of indices; each $P_\sigma : D \rightarrow \{\top, \bot\}$ is a monadic predicate s.t. $P_\sigma(i) = \top$ iff $w_i = \sigma$; $\rtimes(i) = \top$ iff $i = 0$; $\ltimes(i) = \top$ iff $i = |w| + 1$; $s, p : D \rightarrow D$ are successor and predecessor functions over D; for each $i \in \{1, \ldots, |w|\}$, exactly one σ is s.t. $P_\sigma(i) = \top$.

The predicates P_σ indicate which symbol of the alphabet appears at an index, the symbols $\rtimes$ and $\ltimes$ indicate the left and right edges of the word, and p and s capture the linear ordering of the symbols. The monadic model for the string 'baa' over the alphabet $\{a, b\}$ is illustrated in Figure 2.

String models can be further enriched with other relations/functions to represent autosegmental structure (Lambert and Rogers, 2020; Chandlee and Jardine, 2019a; Jardine, 2017), syllable structure (Strother-Garcia, 2019, 2018; Strother-Garcia et al., 2016), prosodic structure (Dolatian, 2020), and syntactic structure (Rogers, 2003). String

[1] The concept of 'signature' is also commonly referred to in model theory literature as 'vocabulary' (e.g. Libkin, 2004) or 'language' (e.g. Marker, 2002).



	0	1	2	3	4
$\rtimes(x)$	$\top$	$\bot$	$\bot$	$\bot$	$\bot$
$P_a(x)$	$\bot$	$\bot$	$\top$	$\top$	$\bot$
$P_b(x)$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$
$\ltimes(x)$	$\bot$	$\bot$	$\bot$	$\bot$	$\top$
	$\rtimes$	b	a	a	$\ltimes$

Figure 2: $\mathcal{M}(baa)$: monadic model of the string ‘baa’.

input	$\rtimes$	b	a	a	$\ltimes$
	0	1	2	3	4
$\rtimes(x)$	$\top$	$\bot$	$\bot$	$\bot$	$\bot$
$P_a(x)$	$\bot$	$\bot$	$\top$	$\top$	$\bot$
$P_b(x)$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$
$\ltimes(x)$	$\bot$	$\bot$	$\bot$	$\bot$	$\top$
$P'_a(x)$	$\bot$	$\bot$	$\bot$	$\top$	$\bot$
$P'_b(x)$	$\bot$	$\top$	$\top$	$\bot$	$\bot$
output	$\rtimes$	b	b	a	$\ltimes$

Figure 3: Logical transducer in (1) over $\mathcal{M}(baa)$.

models have been used to characterize subregular classes of phonological grammars (Lambert and Rogers, 2020; Rogers and Lambert, 2019; Rogers and Pullum, 2011) and to develop inference algorithms for these grammars (Rawski, 2021; Chandlee et al., 2019; Strother-Garcia et al., 2016; Vu et al., 2018).

2.2 Logical Transductions

The focus of this paper is on phonological *maps*. We represent the underlying and surface forms of words with string models, and represent phonological maps as *relations* between string models. Consider a string function $f : \Sigma^* \to \Gamma^*$. For every input string $w \in \Sigma^*$ and corresponding Σ-structure $\mathcal{M}(w)$, there is an output string $f(w)$ and corresponding Γ-structure $\mathcal{M}(f(w))$. We call $\Sigma = \langle P_\sigma, \rtimes, \ltimes, p, s\rangle_{\sigma\in\Sigma}$ the *input signature* and $\Gamma = \langle P_\gamma, \rtimes, \ltimes, p, s\rangle_{\gamma\in\Gamma}$ the *output signature*, and capture the relationship between input and output string models with a map that transforms Σ-structures into Γ-structures by means of logical formulas. In particular, the predicates $\{P_\gamma\}_{\gamma\in\Gamma}$ in the output Γ-structure are associated with logical formulas over the predicates $\{P_\sigma\}_{\sigma\in\Sigma}$ in the input Σ-structure. These maps are called *logical transductions* (Courcelle, 1994; Engelfriet and Hoogeboom, 2001; Courcelle and Engelfriet, 2012). Logical transductions over monadic string models are quantifier-free (Chandlee and Jardine, 2019b, 2021; Chandlee and Lindell, forth.).

As an example, consider the simple function f expressed by the rewrite rule $a \to b/b\_\_$, applied over the alphabet $\Sigma = \{a, b\}$. The logical transducer which expresses f transforms a Σ-structure $\langle D, \rtimes, \ltimes, P_a, P_b, p, s\rangle$ into a Σ-structure $\langle D, \rtimes, \ltimes, P'_a, P'_b, p, s\rangle$, where P'_a and P'_b are expressed as logical formulas over the *signature of the input model*, as in (1). The predicates P_a and P_b express when an index in the input string model carries an ‘a’ or ‘b’ symbol, respectively. The predicates P'_a and P'_b express when an index in the *output* string model carries an ‘a’ or ‘b’. Since f is a length-preserving function, the input and output models have the same domain, linear ordering, and boundary predicates $\rtimes$ and $\ltimes$.

(1) *Logical transducer which expresses* $a \to b/b\_\_$

$$P'_a(x) \;= P_a(x) \wedge \neg P_b(p(x))$$
$$P'_b(x) \;= P_b(x) \vee \big(P_a(x) \wedge P_b(p(x))\big)$$

The equation for $P'_a(x)$ in (1) expresses the following: the index x will carry an ‘a’ in the output model iff it carries an ‘a’ in the input model, and the index preceding it does not carry a ‘b’ in the input model. Stated more simply, x will output as an ‘a’ iff it is an ‘a’ that is not preceded by a ‘b’ in the input. Similarly, $P'_b(x)$ says: x will carry a ‘b’ in the output iff it is either a ‘b’ in the input, or it is an ‘a’ that is preceded by a ‘b’. The string transformation *baa* $\mapsto$ *bba* is presented in Figure 3. In the input model $\mathcal{M}(baa)$, the logical formula for $P'_a(2)$ evaluates to $\bot$ and the logical formula for $P'_b(2)$ evaluates to $\top$. These facts together capture the following relationship between the input and output models: the ‘a’ at index 2 of the input model becomes a ‘b’ in the output model.

More recent work within model-theoretic phonology uses the framework of Boolean Monadic Recursive Schemes (BMRS) to represent string functions. Logical transductions over monadic string models within the BMRS framework have been used to represent phonological maps (Chandlee and Jardine, 2021; Jardine and Oakden, 2023), model process interaction (Oakden, 2021), and provide logical characterizations of the expressivity of phonological maps (Bhaskar et al., 2020, 2023; Yolyan, 2025; Chandlee and Lindell, forth.). This framework expresses the predicates in the output signature using an `if...then...else` syntax; logical transductions within the BMRS framework are referred to as *programs*. The logical transducer in (1) can equivalently be expressed as the BMRS program in (2).

(2) *Logical transducer with BMRS syntax*

$$\begin{aligned} P'_a(x) &= \text{if } P_b(p(x)) \text{ then } \bot \text{ else } P_a(x) \\ P'_b(x) &= \text{if } P_a(x) \text{ then } P_b(p(x)) \text{ else } P_b(x) \end{aligned}$$

The equation for $P'_b(x)$ in (2) expresses the following: if x is an 'a' in the input, then it will output as 'b' iff it is preceded by a 'b'; otherwise, it will output as an 'b' iff it is a 'b' in the input. Chandlee and Jardine (2021) discuss how the `if...then...else` syntax of BMRS can be used to represent meaningful phonological generalizations such as licensing/blocking structures and elsewhere conditions. Section 4.1 of this paper discusses how this syntax is also valuable for expressing the generalizations involved in learning phonological maps. Thus, while these logical transductions can be expressed using propositional logic operators, the syntax of BMRS programs is useful for refining the learning problem.

2.3 Feature Models

The string models discussed so far have the requirement that the predicates $\{\rtimes, \ltimes, P_\sigma\}_{\sigma \in \Sigma}$ partition the domain. Phonological maps, however, target particular features or bundles of features (Jakobson et al., 1952; Clements and Hume, 1995). In order to model phonological words and maps, we use *unconventional* string models (Strother-Garcia et al., 2016), in which more than one predicate can hold at each index.[2] In the case where the predicates represent phonological features, we will refer to these unconventional string models as 'feature models'. In this case, the monadic predicates range over an alphabet of phonological features $\mathbb{F}$, rather than an alphabet of characters. For simplicity, we use the same notation for each feature $F \in \mathbb{F}$ as the associated monadic predicate. That is, $F(x) = \top$ means the sound at index x of the model has feature [+F] and $F(x) = \bot$ means it has feature [-F].[3] Moreover, because the requirement that only one predicate hold at each index is dropped for feature models, the predicates INITIAL/FINAL or MIN/MAX can be used in place of the boundary symbols ⋊/⋉. For simplicity of demonstration, we consider here the limited collection of sounds /p,b,t,d,a,h/ over the four binary phonological features [sonorant], [coronal], [continuant], and [voice]. Each of these sounds corresponds with a unique combination of these four feature specifications, given in Figure 4. The sound /d/ for example, has the specification [−son, +cor, −cont, +voi]. Moreover, because postnasal voicing does not distinguish between nasal sounds, the feature [nasal] will be used to identify *any* nasal sound.

	p	b	t	d	a	h	N
[son]	−	−	−	−	+	−	+
[cor]	−	−	+	+	−	−	
[voi]	−	+	−	+	+	−	+
[cont]	−	−	−	−	+	+	−
[nas]	−	−	−	−	−	−	+

Figure 4: Limited inventory of sounds and features

A logical transducer over feature models specifies output predicates $F'(x)$ for every feature $F \in \mathbb{F}$. The logical transducer which expresses the postnasal voicing map in Zoque is given in (3). A sample transduction of /mpama/→[mbama] 'my clothing' is presented in Figure 5.

(3) *Logical transducer for Zoque postnasal voicing*

$$\begin{aligned} [voi]'(x) &= [voi](x) \lor \\ &\quad (\neg[cont](x) \land [nas](px)) \\ F'(x) &= F(x) \quad \text{for all other } F \in \mathbb{F} \end{aligned}$$

input	m	p	a	m	a
	0	1	2	3	4
`Initial`(x)	$\top$	$\bot$	$\bot$	$\bot$	$\bot$
[son](x)	$\top$	$\bot$	$\top$	$\top$	$\top$
[cor](x)	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
[voi](x)	$\top$	$\bot$	$\top$	$\top$	$\top$
[cont](x)	$\bot$	$\bot$	$\top$	$\bot$	$\top$
[nas](x)	$\top$	$\bot$	$\bot$	$\top$	$\bot$
`Final`(x)	$\bot$	$\bot$	$\bot$	$\bot$	$\top$
[voi]$'(x)$	$\top$	$\top$	$\top$	$\top$	$\bot$
output	m	b	a	m	a

Figure 5: Logical transducer in (3) over $\mathcal{M}(mpama)$

In order to represent feature models more compactly, we use feature matrices to encode all the relevant information in a feature model. The input feature model for /mpama/ in Figure 5, for example, can be represented with the shorthand in (4).

[2] String models in which exactly one predicate holds at each index are referred to as 'conventional models' in previous research because that is the convention used in computer science literature (e.g. Buchi, 1960).

[3] This convention is not the only way to model phonological maps within this framework. We may instead allow $\mathbb{F}$ to contain both $[+F]$ and $[-F]$ as predicates. Moreover, Chandlee and Jardine (2021, pg.42) show that feature models can be adapted for non-binary feature systems as well. Further discussion of feature systems from a model-theoretic perspective can also be found in Nelson (2022).

(4) *Shorthand notation for $\mathcal{M}(mpama)$*

$$\begin{bmatrix}+\text{son}\\-\text{cor}\\+\text{voi}\\-\text{cont}\\+\text{nas}\end{bmatrix}\begin{bmatrix}-\text{son}\\-\text{cor}\\-\text{voi}\\-\text{cont}\\-\text{nas}\end{bmatrix}\begin{bmatrix}+\text{son}\\-\text{cor}\\+\text{voi}\\+\text{cont}\\-\text{nas}\end{bmatrix}\begin{bmatrix}+\text{son}\\-\text{cor}\\+\text{voi}\\-\text{cont}\\+\text{nas}\end{bmatrix}\begin{bmatrix}+\text{son}\\-\text{cor}\\+\text{voi}\\+\text{cont}\\-\text{nas}\end{bmatrix}$$

2.4 Subfactors

The substructures which are relevant to this paper are *subfactors* of monadic feature models. This section does not delve into formal definitions of subfactors in general; discussions and definitions of subfactors can be found in Rogers and Lambert (2019); Rawski (2021); Chandlee et al. (2019); Strother-Garcia et al. (2016). The relevant result is that subfactors of monadic *string* models corresponds with substring models. The substrings of $w \in \Sigma^*$ are the strings $v \in \Sigma^*$ for which there exist $x, y \in \Sigma^*$ such that $w = xvy$. For example, the substrings of 'bac' are $\{\lambda, b, a, c, ba, ac, bac\}$. The *subfactors* of $\mathcal{M}(w)$ are the models $\mathcal{M}(v)$ such that v is a substring of w.

The subfactors of *unconventional* string models, however, are more complex than substring models because any number of predicates can be true at each index. Figure 6 presents the subfactors of a feature model with a domain that has cardinality 1. The feature [coronal] is left out of this figure in order to keep the space manageable. An example of a subfactor of [-nas,-son,-cont,-voi] is a model with no specification for [voice]: [-nas,-son,-cont]. These structures no longer represents a particular sound, but instead collections of sounds. The subfactor [-nas,-son,-cont] corresponds with the set of sounds {/p/, /b/, /t/, /d/}. Although the space of subfactors for feature models is very large, they allow for a formal representation of relevant linguistic generalizations because each subfactor in Figure 6 represents a natural class.

For any model $\mathcal{M}$, the set of subfactors is denoted $Subfact(\mathcal{M})$. A subfactor is called a k-factor if the domain has cardinality k. The set of k-factors of $\mathcal{M}$ is denoted $Subfact_k(\mathcal{M})$. The subfactor relation between models is denoted $\sqsubseteq$.

2.5 Partially-Ordered Sets

A structure $(X, \leq)$ is a partially-ordered set (poset) iff $\leq$ is reflexive, antisymmetric, and transitive. Figure 6 is an example of a poset. Chandlee et al. (2019) show that this is true in general; for a model $\mathcal{M}$, the structure $(Subfact(M), \sqsubseteq)$ is a poset.

Two types of posets which are particularly relevant for the discussion of learning in this paper are filters and ideals, defined in Definitions 4 and 5, respectively. For every $x \in X$ of a poset, the subset of objects above x forms a filter, and the set of objects below x forms an ideal. The remainder of this paper shows how filters and ideals are used to generate and prune a hypothesis space.

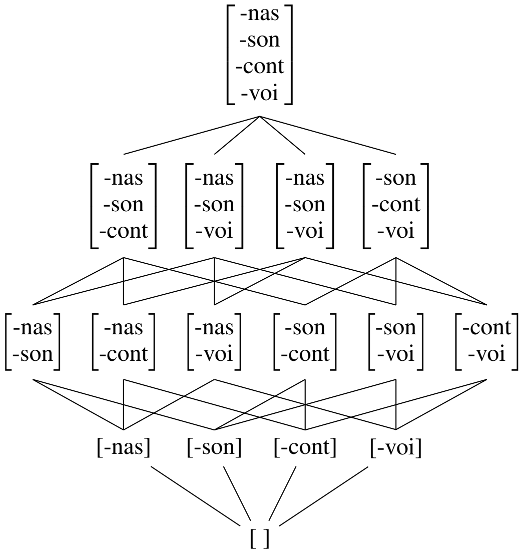


Figure 6: Poset of subfactors for feature models.

Definition 4 (Filters). *For a poset $(X, \leq)$, a filter is a non-empty set $F \subseteq X$ such that:*

(a) for every $x \in F$, if $x \leq y$ then $y \in F$

(b) for every $x, y \in F$, there is some $z \in F$ such that $z \leq x$ and $z \leq y$

*The **principal filter** generated by $x \in X$ is the set*

$$\uparrow x := \{y \in X | x \leq y\}$$

Definition 5 (Ideals). *For a poset $(X, \leq)$, an ideal is a non-empty set $I \subseteq X$ such that:*

(a) for every $x \in I$, if $y \leq x$ then $y \in I$

(b) for every $x, y \in I$, there is some $z \in I$ such that $x \leq z$ and $y \leq z$

*The **principal ideal** generated by $x \in X$ is the set*

$$\downarrow x := \{y \in X | y \leq x\}$$

Posets have been used as a hypothesis space for various learning problems within phonology (Rawski, 2021; Chandlee et al., 2019; Tesar, 2013; Heinz et al., 2012; Heinz, 2010). A partially-ordered hypothesis space encodes entailment relations that are relevant for learning, which allow a large space to be pruned efficiently with a small number of data points. The next section presents two such examples to highlight this point.

3 Phonological Learning with Partially-Ordered Structures

Tesar (2013) uses a stress-length system to illustrate a learning algorithm for underlying forms with a partially-ordered hypothesis space. In this system every vowel in a word can be stressed or unstressed, and long or short. Consider the surface form [paká:], where the second vowel is both stressed and long. The space of possible underlying forms of [paká] is a poset under the *relative similarity* relation, defined as follows: $x \leq y$ iff y is more similar to [paká:] than x. This poset encodes entailment relations relevant to learning. If the map /paká/→[paká:] leads to inconsistency, then /paká/ is not a possible underlying form. This information entails that anything *less similar to* [paká:] than /paká/ will lead to inconsistency (in an output-driven map). Thus, being inconsistent is *downward entailing*, as illustrated in Figure 7 with the candidates marked ✗.

Although the poset contains 16 possible candidates for the underlying form of [paká:] (Figure 8), downward entailment of inconsistency allows the space to be pruned efficiently. If the learner determines that /paká/ cannot be the underlying form, the entire substructure highlighted in gray (i.e. the principal ideal ↓/paká/) in Figure 8 can be removed from the hypothesis space. As Tesar (2013) explains, the benefit of the structured hypothesis space is computational efficiency:

> [...]the possible optimality of a node entails upward, while the definite non-optimality of a node entails downward. This makes it possible to draw conclusions about whole sublattices of candidates based on the evaluation of only a single candidate. [...] This converts exponential search into linear search: the number of possible underlying forms is exponential in the number of features, but the number of forms to actually be tested is linear in the number of features. (Tesar, 2013, pg.287-289)

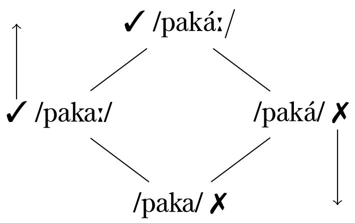


Figure 7: Entailments for learning underlying forms

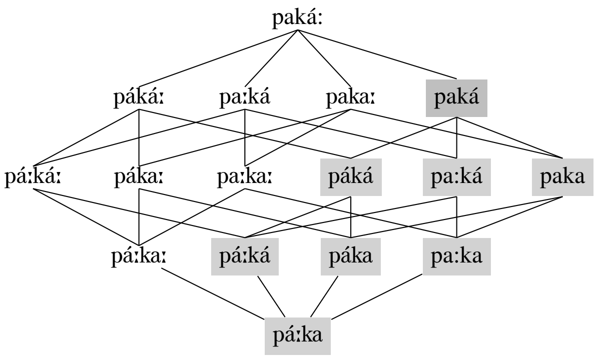


Figure 8: Removing ↓/paká/ from hypothesis space.

Similar observations are made by Chandlee et al. (2019) for the problem of learning phonotactic grammars. In this case, subfactors form a poset of potential banned structures in a grammar. Ungrammaticality is upward entailing; if a structure is ungrammatical, then all its superfactors must be ungrammatical. For example, if [+nas][-voi,-cont] is an ungramamtical surface form (as in the case of Zoque), then [+nas][-nas,-voi,-cont] must also be ungrammatical. On the other hand, grammaticality is downward entailing; if a structure is grammatical, then all its subfactors must also be grammatical. These entailments are illustrated in Figure 9, where ungrammatical structures are represented by ✗ and grammatical structures are represented by ✓. Similar to the previous example, these entailments are used to prune the hypothesis space. In this case, if a structure is ungrammatical, all the structures above it (i.e. the principal filter) will be removed from the hypothesis space. This process is discussed in the next section.

3.1 Bottom-Up Feature Inference Algorithm (BUFIA)

The learning algorithm introduced by Chandlee et al. (2019) is called the Bottom-Up Factor Inference Algorithm (BUFIA). The goal of BUFIA is

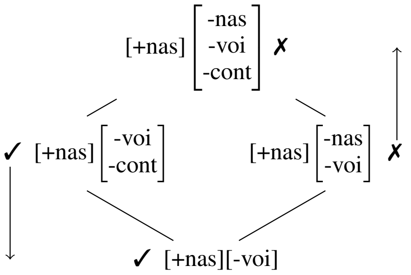


Figure 9: Entailments for learning grammars

to learn the *banned* k-factors of a language from a positive sample (i.e. collection of grammatical words from the language). BUFIA runs as follows. There is a finite sample S containing words from a language. The goal is to learn the *grammar* of the language, which is represented as a set of k-factors which are not permitted in the language. Because there are many ways to express the ungrammatical k-factors, a further goal is to find the most general description. For this reason, the algorithm searches through the hypothesis space *bottom-up*. If some k-factor x in the space is not found in any surface form in S, then it is added to the set of constraints. Because ungrammaticality is upward entailing, all of x's superfactors must also be ungrammatical. Thus, the entire filter $\uparrow x$ is removed from the hypothesis space since the grammaticality of these structures no longer needs to be considered. In this way, only the minimal (most general) ungrammatical k-factors are part of the resulting grammar.

BUFIA is illustrated here with the example of postnasal voicing in Zoque. For brevity, the features [sonorant] and [coronal] are left out of the discussion because they are ultimately not necessary to identify the minimal ungrammatical 2-factor in Zoque.[4] The surface forms from Section 1 provide evidence that all the subfactors of [+nas][-voi,-cont] are grammatical. These subfactors are presented in Figure 10 with N, V, and C for the features [nasal], [voi], and [cont], respectively. The form [tatah] 'father' provides evidence that [][-voi,-cont] is a grammatical structure. The form [nhayah] 'my husband' provides evidence that [+nas][-voi] is grammatical. The form [ndatah] 'my father' provides evidence that [+nas][-cont] is grammatical. Moreover, all the subfactors of these three 2-factors are grammatical. The 2-factor [+nas][-voi,-cont] is the *minimal* structure in the poset that is not found in any surface form of Zoque; all the structures below it are grammatical, while all the structures above it ungrammatical.

These data points are revisited in the next section to show how the same structures and principles can be used to learn the postnasal voicing map. Further discussion of BUFIA and abduction principles used in learning can be found in Rawski (2021). Recent applications of BUFIA can also be found in Li (2025) and Payne (2024).

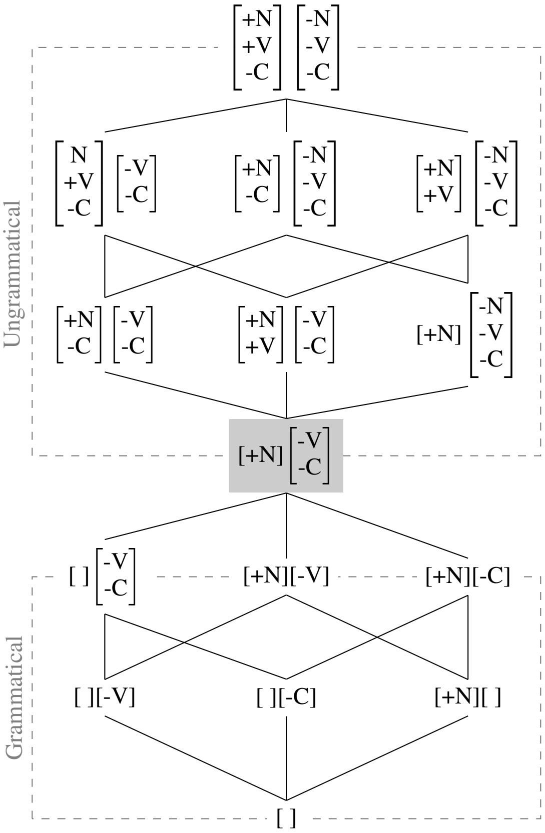


Figure 10: Minimal ungrammatical structure in Zoque.

As a final note, the learning problem for BUFIA (Def. 9 of Chandlee et al., 2019) is not stated in terms of finding the *correct* grammar in the limit, as in Gold (1967), but instead finding the most *general* grammar that is consistent with the sample of surface forms. The goal of learning phonological maps is similar: generate a logical transducer that expresses the most general phonological map that is consistent with the sample of input-output pairs. A formal statement of the learning problem in this paper is left for future work.

4 Learning Logical Transducers

Chandlee and Lindell (forth.) show that non-recursive quantifier-free logical transductions over monadic models are the logical characterization of finite-to-one ISL functions. A string function $f : \Sigma^* \to \Gamma^*$ is finite-to-one iff for every $w \in \Gamma^*$, the set $\{u \in \Sigma^* | f(u) = w\}$ is finite. In other

[4] The [-voi,-son] /h/ in [nhayah] 'my husband' does not have a voiced counterpart in Zoque, and therefore does not become voiced after a nasal (Wonderly, 1951). Thus, [nhayah] provides evidence that [+nas][-voi,-son] (and every subfactor of it) is grammatical. The feature [cont] is therefore necessary to distinguish the sounds which can be preceded by a nasal from the sounds which cannot. Moreover, since the ungrammaticality of [+nas][-voi,-cont] entails the ungrammaticality of both [+nas][-voi,-son,-cont] and [+nas][-voi,-cor,-cont], [son] and [cor] are not necessary to represent the ungrammatical structures in Zoque because a more general subfactor suffices.

words, every surface form has finitely many underlying forms that map to it under the application of f. The output of the learning procedure introduced here is a quantifier-free BMRS program. This means we assume that we are always learning finite-to-one phonological maps. We moreover restrict the maps we consider to those that order-preserving and length-preserving. That is, epenthesis, deletion, and metathesis are not considered here. Future extensions to non-length-preserving maps are discussed in Section 5.

The examples from Tesar (2013) and Chandlee et al. (2019) illustrate how two very different learning goals can be pursued with partially-ordered hypothesis spaces, where entailment relations encoded in the structure make it possible for the learner to remove large substructures with a single data point. The remainder of this paper shows that the same observation holds of the problem of learning environments where features undergo change, and ultimately logical transducers which express a phonological map. The main idea expressed in this section is that learning a phonological map amounts to learning several phonotactic grammars simultaneously. In that sense, the learning procedure here is a way of *lifting* the ideas in Chandlee et al. (2019) to maps. This idea is formalized with the normal form for logical transducers proposed in Definition 6 and the main result in Theorem 7.

4.1 Normal Form for Logical Transductions

Consider a signature $\mathcal{S} = \{P_1, \ldots P_n, s, p\}$, where each P_i is a unary predicate. A *term* over this signature is recursively defined as follows: a variable is a term; for every term T, $p(T)$ and $s(T)$ are terms. The *atoms* over this signature are defined as

$$atoms(\mathcal{S}) := \{P_i(T) \,|\, T \text{ is a term}\}$$

Definition 6. *Let $\{P'_\sigma\}_{\sigma\in\Sigma}$ be a collection of formulas defined over an input signature Σ. Each P'_σ is in **normal form** if it is expressed as*

$$P'_\sigma(x) = \textit{if } P_\sigma(x) \textit{ then } \neg\phi_\sigma(x) \textit{ else } \psi_\sigma(x)$$

where $\phi_\sigma(x)$ and $\psi_\sigma(x)$ are either $\top$, $\bot$, or disjunctive normal form expressions over $Atoms(\Sigma)$.[5]

Although normal form programs use the `if...then...else` syntax, the 'then' and 'else' parts of the equations are expressed with propositional operators.[6] This brings into question why we would bother using the `if...then...else` syntax at all. The reason for this representations is that the syntax compartmentalizes each equation in a program into two components: the 'then' expression and the 'else' expression. We consider first two examples which illustrate the information the 'then' and 'else' parts of these equations encode.

The normal form of the program in (2) is presented in (5). Recall that this program expresses the same function as the rewrite rule $a \rightarrow b/b\_\_$. The four pieces of information contained within this normal form program are presented in (6). The expression $\phi_a(x)$ is a logical description of the environment where an 'a' input becomes a non-'a' output. $\psi_b(x)$ is a logical description of the environment where a non-'b' input becomes a 'b' output. Similarly, $\psi_a(x) = \bot$ and $\phi_b(x) = \bot$ encode the facts that an non-'a' input never becomes an 'a', and a 'b' input never becomes a non-'b', respectively. The normal form in (5) rearranges the atoms of (2) into a syntactic form that explicitly references environments where change takes place.

(5) *Logical transducer from (2) in normal form*

$$\begin{aligned} P^*_a(x) &= \text{if } P_a(x) \text{ then } \neg\, P_b(px) \text{ else } \bot \\ P^*_b(x) &= \text{if } P_b(x) \text{ then } \neg\bot \\ &\qquad \text{else } (P_a(x) \wedge P_b(px)) \end{aligned}$$

(6) *Information encoded in the normal form in (5)*

$$\begin{array}{ll} \phi_a(x) = P_b(px) & \psi_a(x) = \bot \\ \phi_b(x) = \bot & \psi_b(x) = P_a(x) \wedge P_b(px) \end{array}$$

A similar translation can be done for the postnasal voicing transducer in (3), presented in (7). The highlighted expression $[nas](px) \wedge \neg[cont](x)$ is a logical description of the environment in which a [-voi] sound in the input model becomes [+voi] in the output (i.e. $\psi_{[voi]}(x)$).

(7) *Logical transducer from (3) in normal form*

$$\begin{aligned} [voi]^*(x) &= \text{if } [voi](x) \text{ then } \neg\bot \\ &\qquad \text{else } ([nas](px) \wedge \neg[cont](x)) \\ F^*(x) &= \text{if } F(x) \text{ then } \neg\bot \text{ else } \bot \end{aligned}$$

The intuition for the normal form in Definition 6 is that every string function can be uniquely-identified with $\{\phi_\sigma, \psi_\sigma\}_{\sigma\in\Sigma}$, which encode information about the changes the function induces, and the environments those changes take place. With

[5] We assume that the input and output alphabets are the same. This means that we assume the underlying and surface forms have the same feature inventory. This assumption is not necessary, and is only used for simplicity.

[6] Every propositional operator can be expressed using `if...then...else`. For example, $p \vee q =$ if p then $\top$ else q.

respect to learning, the significance of the normal form is that it refines the learning problem in the following way: in order to learn a program $\{P'_\sigma\}_{\sigma\in\Sigma}$, it is sufficient to learn $\{\phi_\sigma, \psi_\sigma\}_{\sigma\in\Sigma}$. This fact is the immediate result of Theorem 7.

Theorem 7. *Every program $\{P'_\sigma\}_{\sigma\in\Sigma}$ over an input signature $\Sigma = \langle P_\sigma, s, p\rangle_{\sigma\in\Sigma}$ is logically equivalent to a program $\{P^*_\sigma\}_{\sigma\in\Sigma}$ where each P^*_σ is in normal form.*

The proof of Theorem 7 is presented in the appendix. The main idea is that for any output predicate P'_σ, there is a systematic way to construct an equivalent normal form predicate P^*_σ by generating a truth table of all the atoms in the definition of P'_σ, and determining ϕ_σ and ψ_σ from that truth table.

For phonological maps, learning a program over feature models amounts to learning $\{\phi_F, \psi_F\}_{F\in\mathbb{F}}$, where ϕ_F (ψ_F) expresses the environment where a [+F] ([-F]) sound in the underlying form becomes [-F] ([+F]) in the surface form. In this way, learning programs in this normal form amount to learning *phonological generalizations* regarding environments where features undergo change. The observation that connects this learning goal to the examples presented in Section 3 is that the space of possible expressions for ϕ_F and ψ_F can also be modeled as a poset, with relevant entailment relations that allow the space to be pruned efficiently.

Consider the case of postnasal voicing and the goal of learning the environment where a [-voice] input becomes [+voice] ($\psi_{[voi]}$). A similar diagram to Figures 7 and 9 is presented in Figure 11 for the goal of learning the environment where voicing takes place. Environments where voicing takes place are upward entailing, while environments where voicing does *not* take place are downward entailing. For example, if being a [-cont] that is immediately preceded by a [+nas] is sufficient to trigger voicing, then being a [-cont,-son] that is immediately preceded by a [+nas] is also sufficient. This is illustrated in Figure 11 with the logical expressions marked ✓. This means that in order to find the most general description of the environment that triggers voicing, we must traverse the space *bottom-up*. On the other hand, if an input-output pair indicates that some environment does *not* trigger voicing, then none of its subfactors are going to trigger voicing and therefore, its entire principal ideal can be removed from the hypothesis space, similar to the example of learning underlying forms in Figure 8.

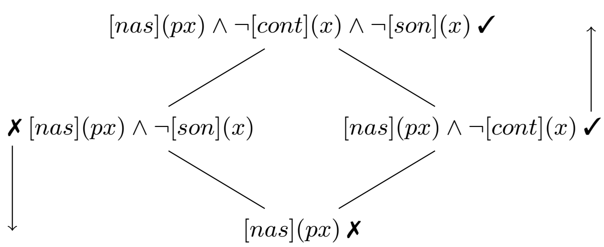
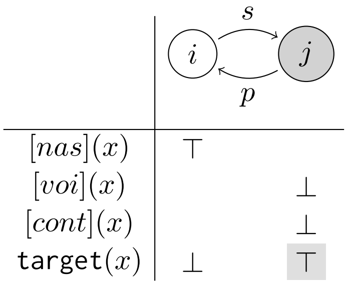

$[nas](px) \wedge \neg[cont](x) \wedge \neg[son](x)$ ✓
✗ $[nas](px) \wedge \neg[son](x)$
$[nas](px) \wedge \neg[cont](x)$ ✓
$[nas](px)$ ✗

Figure 11: Entailments for learning environments

Each logical expression in Figure 11 corresponds with a description of a 2-*factor*. The expression $[nas](px)\wedge\neg[cont](x)$, for example, describes the 2-factor [+nas][-cont]. The subfactors which are used to learn grammars in BUFIA can therefore be adapted to learning environments. The distinction is that when we are learning *maps* rather than surface constraints, some index of the subfactor is the target of the map. Every k-factor we consider will therefore have an extra predicate `target` which identifies the unique index of the structure that is the target of the feature change in question. Figure 12 presents the 2-factor [+nas][-voi,-cont] with an additional predicate `target` which picks out the index that the voicing process targets. As a shorthand, we underline the subfactor that is the target of the map, as (8).



	i	j
$[nas](x)$	$\top$	
$[voi](x)$		$\bot$
$[cont](x)$		$\bot$
`target`(x)	$\bot$	$\top$

Figure 12: 2-factor targeted by postnasal voicing

(8) *Shorthand notation for 2-factor in Figure 12*

$$[+nas]\underline{\begin{bmatrix}-cont\\-son\end{bmatrix}}$$

Note that the subfactor partial order extends to a natural ordering on environments. For example, [+nas]<u>[-cont]</u> is a subfactor of [+nas]<u>[-cont,-son]</u>, but <u>[+nas]</u>[-cont] is not. Thus, the resulting structures with the `target` predicate form a partially ordered hypothesis space consisting of *environments*.

4.2 Variables and Initialization

In the case of learning grammars, there is a single hypothesis space consisting of all the possible subfactors. In the case of learning maps, there are several hypothesis spaces: one for each of the learning targets $\{\phi_F, \psi_F\}_{F \in \mathbb{F}}$. Thus, learning a program can be viewed as learning several grammars simultaneously. This section shows how principal ideals are used to construct each of the hypothesis spaces.

For every $X \in \{\phi_F, \psi_F\}_{F \in \mathbb{F}}$ we need two sets of k-factors: $V(X)$ and $\widehat{V}(X)$. $V(X)$ contains the relevant k-factors in which the targeted change takes place. $V(\phi_F)$, for example, contains all the k-factors in which an index x is such that $F(x) = \top$ in the input model but $F'(x) = \bot$ in the output model. $\widehat{V}(\phi_F)$ contains all the k-factors in which an index x is such that $F(x) = \top$ in the input model and $F'(x) = \top$ in the output model. In other words, $V(\phi_F)$ contains all the k-factors in which a [+F] sound becomes [-F], and $\widehat{V}(\phi_F)$ contains all the k-factors in which [+F] sounds do not undergo change. In this way, the collections V and $\widehat{V}$ encode positive and negative evidence for the environments where features undergo change. The hypothesis spaces are then defined in Definition 8.

Definition 8. *For every $X \in \{\phi_F, \psi_F\}_{F \in \mathbb{F}}$, the corresponding **hypothesis space** is the set*

$$\mathcal{H}(X) := \bigcup_{x \in V(X)} \downarrow x - \bigcup_{x \in \widehat{V}(X)} \downarrow x$$

subject to the following two restrictions:

(i) Every $\mathcal{M} \in \mathcal{H}(X)$ has exactly one index i in the domain of $\mathcal{M}$ such that `target`$(i) = \top$.

(ii) If X is ϕ_F for some $F \in \mathbb{F}$, then for every $\mathcal{M} \in \mathcal{H}(X)$, $F(i) = \top$ must hold for the unique i such that `target`$(i) = \top$. *If X is ψ_F, then $F(i) = \bot$ must hold.*

The posets defined in Definitions 8 takes all the k-factors in which the relevant change was observed in some pair of input-output models, and removes the subfactors in which the relevant change was *not* observed. The restrictions in (i) and (ii) then ensure that only the subfactors which represent relevant environments are included in the space. The restriction in (i) ensures that every structure in the hypothesis space represents an *environment*. The restriction in (ii) ensures that the hypothesis space includes only the *relevant* environments. If the goal is to learn the environment where a [-voi] sound becomes [+voi], then the hypothesis space should only include environments where the target of the change is [-voi]. This means that $\mathcal{H}(\psi_{[voi]})$ will only include k-factors where the unique index that satisfies the predicate `target` is such that $[voi](x) = \bot$.

The learning procedure starts with an initial hypothesis, and generates a new hypothesis with each input-output pair of models. Before any data have been observed, $V(X)$ and $\widehat{V}(X)$ are empty for each $X \in \{\phi_F, \psi_F\}$. This means that the initial hypothesis space for each X will also be empty by definition. Consequently the set of minimal elements, will be empty. In this case, the corresponding logical formula for X must be $\bot$. The initial variables, hypothesis, and corresponding BMRS program are summarized in (9). The consequence of V and $\widehat{V}$ being empty is that the initial (null) hypothesis corresponds with a BMRS program that represents the identity map.

(9) *Initial Variables and Hypothesis*
Variables; for all $X \in \{\phi_F, \psi_F\}_{F \in \mathbb{F}}$

$$\begin{aligned} V(X) &= \emptyset \\ \widehat{V}(X) &= \emptyset \\ min(\mathcal{H}(X)) &= \emptyset \end{aligned}$$

Individual learning goals

$$\{\phi_F(x) = \bot;\ \psi_F(x) = \bot\}_{F \in \mathbb{F}}$$

Initial Hypothesis as a BMRS program

$$\begin{aligned} \{F'(x) &= \text{if } F(x) \text{ then } \neg\bot \text{ else } \bot\}_{F \in \mathbb{F}} \\ &\equiv \{F'(x) = F(x)\}_{F \in \mathbb{F}} \end{aligned}$$

4.3 Updating the Hypothesis

When an input-output pair provides evidence for a feature change, the corresponding hypothesis space is updated. For example, if a pair of models $(\mathcal{M}, \mathcal{M}')$ is such that some index x is [-F] in the input model but [+F] in the output model, the set $V(\psi_F)$ will be updated with the k-factor(s) of $\mathcal{M}$ which contain x as the target. If, on the other hand, some index x is [-F] in the input and remains [-F] in the output, the set $\widehat{V}(\psi_F)$ will be updated with the relevant k-factors. Each time V or $\widehat{V}$ is updated, the set of minimal elements must also be updated. Thus, there are two possible types of updates.

Consider first the case in which a k-factor x is added to $V(X)$ for some X. In this case, the hypothesis space $\mathcal{H}(X)$ grows, and potentially the set of minimal elements grow. This means that when a new k-factor x is added to $V(X)$, the update function maintain all previous minimal elements and only determines whether to add new ones. The procedure for determining which elements to add to $M(X)$ is as follows: we traverse the structure $\downarrow x$ bottom-up in order to find the minimal subfactors of x which are neither subfactors of any current minimal elements, nor subfactors of any k-factor in $\widehat{V}(X)$. Because both k and the number of features we consider are fixed, the structure $\downarrow x$ is always of fixed size. Moreover, we only need to know the set of factors in $M(X)$ and $\widehat{V}(X)$ in order to determine the new set of minimal elements.

In the case where a new k-factor x is added to $\widehat{V}(X)$, the update function must remove any element $m \in M(X)$ such that $m \sqsubseteq x$, and potentially add new minimal elements. In this case, there are two possibilities: if $x \in V(X)$, then m is removed from the set of minimal elements; otherwise m is replaced by new minimal elements that are immediately above m in the poset. In this case, we only need information about $V(X)$ and $M(X)$ in order to update the set of minimal elements.

Both types of updates therefore amount to traversing a restricted space in order to find new minimal elements. Although the hypothesis space can get very large, for each $X \in \{\phi_F, \psi_F\}$, only the sets $M(X)$, $V(X)$, and $\widehat{V}(X)$ need to be maintained in memory in order to update the hypothesis. The next section illustrates the update process with the example of postnasal voicing.

4.4 Case Study: Postnasal Voicing in Zoque

Consider first the input-output pair of models in Figure 5 representing /mpama/ → [mbama]. The input-output models for this transformation were presented in Figure 5. The sound at index 1 is [-voi] in the input model and [+voi] in the output model. This means that the set $V(\psi_{[voi]})$ must be updated with the 2-factors that contain the sound at index 1 as a target. There are two possible 2-factors in the input model which contain the sound at index 1. These are presented in Figure 13. The updated hypothesis space is presented in Figure 14. The corresponding updates are summarized in (10).

	m	p
[nas](x)	⊤	⊥
[voi](x)	⊤	⊥
[cont](x)	⊥	⊥
target(x)	⊥	⊤

	p	a
[nas](x)	⊥	⊥
[voi](x)	⊥	⊤
[cont](x)	⊥	⊤
target(x)	⊤	⊥

Figure 13: 2-factors added to $V(\psi_{[voi]})$

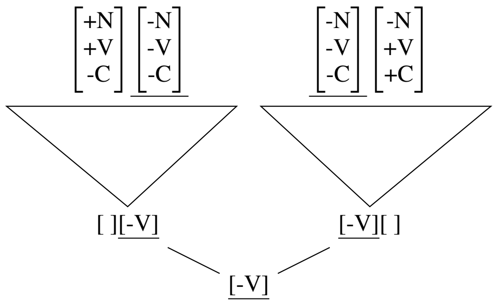


Figure 14: $\mathcal{H}(\psi_{[voi]})$ given (/mpama/, [mbama]).

Since being [-voi] is presumed to be true when learning $\psi_{[voi]}$ (by restriction (ii) of Definition 8), the minimal element is $\underline{\text{[-V]}}$ rather than $\emptyset$. This element corresponds with $\psi_{[voi]}(x) = \top$. We could also set $\psi_{[voi]} = \neg[voi](x)$; this would give a logically equivalent program and corresponding map, but with unnecessary redundancy. The new hypothesis therefore says that a [-voi] sound *always* becomes [+voi]. This hypothesis corresponds with the *most general* hypothesis that accounts for the input-output sample observed.

(10) *Hypothesis update given /mpama/→[mbama]*

$$min(\mathcal{H}(\psi_{[voi]})) = \{\underline{[-V]}\}$$
$$\psi_{[voi]}(x) = \top$$

as a BMRS program

$$[voi]^*(x) = \text{if } [voi](x) \text{ then } \neg\bot \text{ else } \top$$

as a phonological map

$$[-voi] \rightarrow [+voi]$$

We consider next the case where the underlying and surface forms are the same. Consider the surface form [tatah] 'father'. The hypothesis in (10) predicts that we should not see this surface form; we would instead expect to see [dadah]. This surface form therefore provides evidence that the the previous hypothesis is incorrect. In this case, the pair of 2-factors in Figure 15 are added to $\widehat{V}(\psi_{[voi]})$. The updated hypothesis space is presented in Figure 16. The corresponding updates are summarized in (11)

	a	t
[nas](x)	⊥	⊥
[voi](x)	⊤	⊥
[cont](x)	⊤	⊥
target(x)	⊥	⊤

	t	a
[nas](x)	⊥	⊥
[voi](x)	⊥	⊤
[cont](x)	⊥	⊤
target(x)	⊤	⊥

Figure 15: 2-factors added to $\widehat{V}(\psi_{[voi]})$

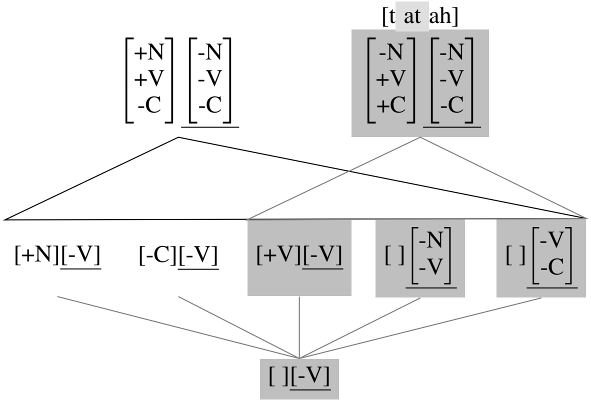


Figure 16: Updated $\mathcal{H}(\psi_{[voi]})$ given (/tatah/, [tatah]).

The hypothesis space has two minimal elements: [+N][-V] and $[-C]\underline{[-V]}$. The resulting hypothesis is therefore a *disjunction*. The new hypothesis says that a [-voi] sound becomes [+voi] if it is either preceded by a [+nas] *or* by a [-cont] sound. The remaining updates with /nhayah/→[nhayah] and /ku?tpa/→[ku?tpa] are presented in Figure 17.

(11) *Hypothesis update given /tatah/→[tatah]*

$$min(\mathcal{H}(\psi_{[voi]})) \quad = \{[+N]\underline{[-V]}, [-C]\underline{[-V]}\}$$
$$\psi_{[voi]}(x) \quad = [nas](px) \vee \neg[cont](px)$$

as a BMRS program

$$[voi]^*(x) \quad = \text{if } [voi](x) \text{ then } \neg\bot \quad \text{else } [nas](px) \vee \neg[cont](px)$$

as a phonological map

$$[-voi] \rightarrow [+voi]/ \begin{Bmatrix} [+nas] \\ [-cont] \end{Bmatrix} \_$$

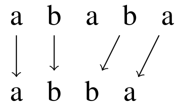
The remaining subfactors in the hypothesis space in Figure 17 are exactly the ungrammatical 2-factors as Figure 10. While [+N][-V,-C] is the most general description of ungrammatical 2-factors in Zoque, [+N][-V,-C] is the most general description of the environment where a sound becomes voiced. Since ungrammaticality is upward entailing (Figure 9), it makes sense that being an environment where voicing takes place is also upward entailing (Figure 11); a [-V,-C] segment becomes voiced when it is preceded by a nasal as a means of *repairing* the fact that [+N][-V,-C] is an ungrammatical surface form. The learning approach presented here therefore makes the relationship between surface constraints and phonological maps concrete.

5 Discussion and Conclusion

This paper starts with length-preserving and order-preserving ISL functions in order to show how partially-ordered hypothesis spaces over model-theoretic representations can be used to learn phonological maps as logical transducers. Chandlee and Jardine (2021, pg.3) state the appeal of the BMRS formalism for phonology as follows: “this formalism has a well-understood complexity bound that corresponds to previous results in the study of computational phonology, but it also provides a way to implement phonological substance”. The approach presented here is extends this advantage of BMRS to learning; because BMRS can be used to represent phonological generalizations, we can adapt previous work on model-theoretic learning to show that BMRS can also be used *learn* phonological generalization. This latter point is ultimately the broader contribution of this work.

This paper assumed that there is a trivial one-to-one correspondence between input and output elements, where the segment at index x in the output model is the surface form of the segment at index x in the input model. This assumptions rules out deletion, epenthesis, and metathesis maps which are also ISL (Chandlee, 2014). An interesting extension of this work would be to develop a learning procedure for input-output index correspondences. Consider for example a simple deletion map $a \rightarrow \lambda/b\_b$, where an ‘a’ is deleted whenever it is between two ‘b’s. (12) presents the input-output correspondences for the string transformation $ababa \mapsto abba$.

(12) *Deletion input-output correspondences*



Recent work from Li (2025) uses BUFIA to learn tonotactic patterns, where the structures in the hypothesis space encode autosegmental representations. An interesting question to pursue further is whether a similar hypothesis space can be used to represent input-output correspondences. Doing so would amount to learning the environments where features undergo change and the input-output correspondence between segments in the underlying and surface forms simultaneously.

A further extension of this work is to extend hypothesis space to contain *pairs* of subfactors, corresponding with input and output string models. In this case, the corresponding BMRS programs would be recursive, allowing for representation of output strictly local maps (Chandlee et al., 2015).

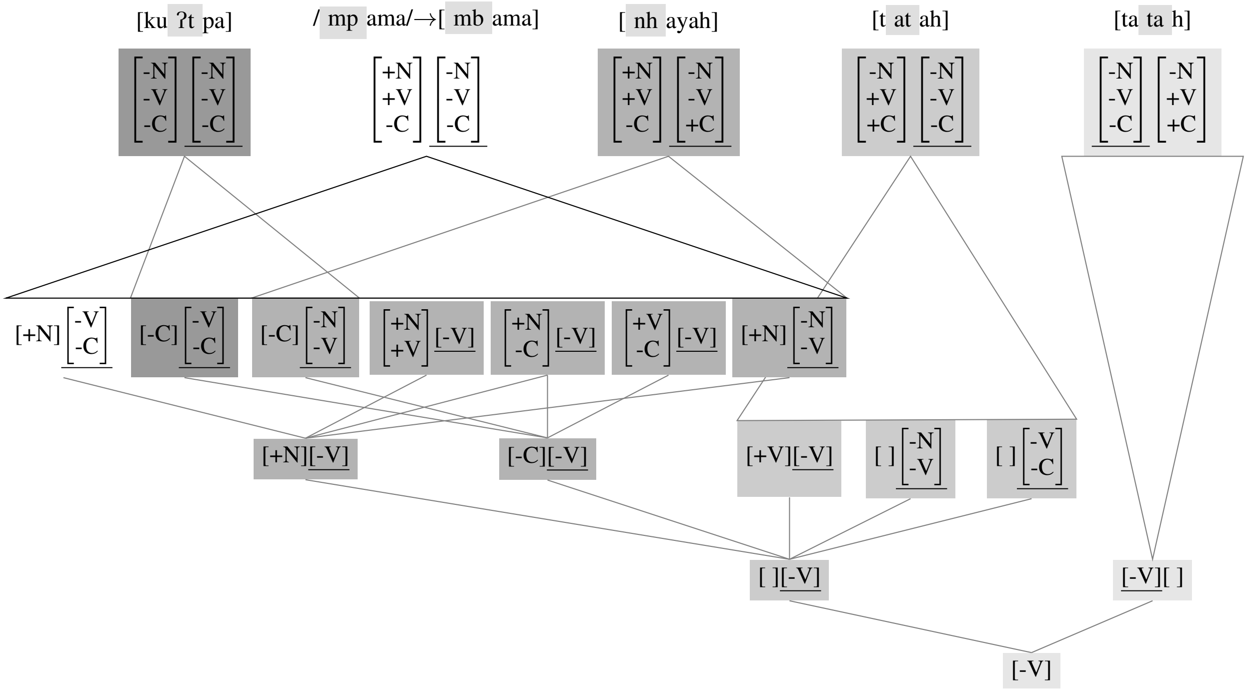


Figure 17: Updated $\mathcal{H}(\psi_{[voi]})$ given /nhayah/→[nhayah] and /kuʔtpa/→[kuʔtpa].

Theorem 7 stated that every non-recursive BMRS programs can be expressed as an equivalent normal form program. However, it seems intuitive that this result should extend to programs in general because *every* map can be uniquely identified in terms of environments where features undergo change; this is afterall how SPE-style rewrite rules express a function. However, such a result requires reasoning about recursion in BMRS, which uses a fixed point semantics (Bhaskar et al., 2020), and is outside the scope of this paper.

Ultimately, the purpose of this paper was to present a starting point for a model-theoretic approach to learning phonological maps, and the merits of such an approach. The ideas and procedure presented here can be extended to different representations and classes of functions. In order to extend this work to more complex representations, the next step for this research is an implementation of the learning procedure presented here. Doing so would also make it possible to obtain *empirical* results.

References

Siddharth Bhaskar, Jane Chandlee, and Adam Jardine. 2023. Rational functions via recursive schemes. *Preprint*, arXiv:2302.03074.

Siddharth Bhaskar, Jane Chandlee, Adam Jardine, and Christopher Oakden. 2020. Boolean monadic recursive schemes as a logical characterization of the subsequential functions. In *Language and Automata Theory and Applications (LATA)*, Lecture Notes in Computer Science, pages 157–169. Springer.

Richard Buchi. 1960. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6:66–92.

Jane Chandlee. 2014. *Strictly Local Phonological Processes*. Ph.D. thesis, University of Delaware.

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. 2014. Learning strictly local subsequential functions. *Transactions of the Association for Computational Linguistics*, 2:491–504.

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. 2015. Output strictly local functions. In *Proceedings of the 14th Meeting on the Mathematics of Language*, page 112–125. Association for Computational Linguistics.

Jane Chandlee, Remi Eyraud, Jeffrey Heinz, Adam Jardine, and Jonathan Rawski. 2019. Learning with partially ordered representations. In *Proceedings of the 16th Meeting on the Mathematics of Language*, pages 91–101, Toronto, Canada. Association for Computational Linguistics.

Jane Chandlee and Jeffrey Heinz. 2018. Strict locality and phonological maps. *Linguistic Inquiry*, 49(1):23–60.

Jane Chandlee and Adam Jardine. 2019a. Autosegmental input strictly local functions. In *Transactions of the Association for Computational Linguistics*, volume 7.

Jane Chandlee and Adam Jardine. 2019b. Quantifier-free least fixed point functions for phonology. In *Proceedings of the 16th Meeting on the Mathematics of Language (MOL)*, pages 50–62. Association for Computational Linguistics.

Jane Chandlee and Adam Jardine. 2021. Computational universals in linguistic theory: Using recursive programs for phonological analysis. *Language*, 97.

Jane Chandlee and Steven Lindell. forth. Logical perspectives on strictly local transformations. In *Doing Computational Phonology*. Oxford University Press.

Nick Clements and Elizabeth Hume. 1995. The internal organisation of speech sounds. In John Goldsmith, editor, *The Handbook of Phonological Theory*, chapter 7, pages 245–307. Blackwell.

Bruno Courcelle. 1994. Monadic second-order definable graph transductions: a survey. *Theoretical Computer Science*, 126(1):53–75.

Bruno Courcelle and Joost Engelfriet. 2012. Monadic second-order transductions. In *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*, chapter 7, pages 505–577. Cambridge University Press.

Hossep Dolatian. 2020. *Computational locality of cyclic phonology in Armenian*. Ph.D. thesis, Stony Brook University.

Joost Engelfriet and Hendrik Jan Hoogeboom. 2001. Mso definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic (TOCL)*, 2(2):216–254.

Mark Gold. 1967. Language identification in the limit. *Information and Control*, 10:447–474.

Jeffrey Heinz. 2010. String extension learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Jeffrey Heinz, Anna Kasprzik, and Timo Kötzing. 2012. Learning in the limit with lattice-structured hypothesis spaces. *Theoretical Computer Science*, 457.

Roman Jakobson, Gunnar Fant, and Morris Halle. 1952. *Preliminaries to Speech Analysis*. MIT Press.

Adam Jardine. 2017. The local nature of tone-association patterns. *Phonology*, 34:385–405.

Adam Jardine and Chris Oakden. 2023. Computing process-specific constraints. *Linguistic Inquiry*.

Dakotah Lambert and James Rogers. 2020. Tier-based strictly local stringsets: Perspectives from model and automata theory. In *Proceedings of the Society for Computation in Linguistics 2020*, pages 159–166. Association for Computational Linguistics.

Han Li. 2025. Learning tonotactic patterns over autosegmental. *Proceedings of the 2023 and 2024 Annual Meetings on Phonology*.

Leonid Libkin. 2004. *Elements of Finite Model Theory*.

David Marker. 2002. *Model Theory: An Introduction*.

Scott Nelson. 2022. A model theoretic perspective on phonological feature systems. In *Proceedings of the Society for Computation in Linguistics*, volume 5.

Chris Oakden. 2021. *Modeling phonological interactions using recursive schemes*. Ph.D. thesis, Rutgers University.

Sarah Payne. 2024. A generalized algorithm for learning positive and negative grammars with unconventional string models. In *Proceedings of the Society for Computation in Linguistics (SCiL)*, pages 75–85.

Jonathan Rawski. 2021. *Structure and Learning in Natural Language*. Ph.D. thesis, Stony Brook University.

James Rogers. 2003. Syntactic structures as multi-dimensional trees. *Research on Language and Computation*, 1:265–305.

James Rogers and Dakotah Lambert. 2019. Some classes of sets of structures definable without quantifiers. In *Proceedings of the 16th Meeting on the Mathematics of Language*, pages 63–77. Association for Computational Linguistics.

James Rogers and Geoffrey Pullum. 2011. Aural pattern recognition experiments and the subregular hierarchy. *Journal of Logic, Language and Information*, 20:329–342.

Kristina Strother-Garcia. 2018. Imdlawn Tashlhiyt Berber syllabification is quantifier-free. In *Proceedings of the Society for Computation in Linguistics*.

Kristina Strother-Garcia. 2019. *Using model theory in phonology: a novel characterization of syllable structure and syllabification*. Ph.D. thesis, University of Delaware.

Kristina Strother-Garcia, Jeffrey Heinz, and Hyun Jin Hwangbo. 2016. Using model theory for grammatical inference: a case study from phonology. In *Proceedings of The 13th International Conference on Grammatical Inference*, pages 66–78. PMLR.

Bruce Tesar. 2013. *Output-Driven Phonology: Theory and Learning*. Cambridge University Press.

Mai Vu, Ashkan Zehfroosh, Kristina Strother-Garcia, Michael Sebok, Jeffrey Heinz, and Herbert G. Tanner. 2018. Statistical relational learning with unconventional string models. *Frontier in Robotics and AI*.

William L. Wonderly. 1951. Zoque II: Phonemes and Morphophonemics. *International Journal of American Linguistics*, 17:105–123.

Tatevik Yolyan. 2025. A logical characterization of weak determinism as simultaneous application. *Journal of Logic, Language and Information*, 34(1):89–153.

A Appendix

Proof of Theorem 7. Let $\{P'_\sigma\}_{\sigma\in\Sigma}$ be a non-recursive BMRS program over the input signature $\Sigma = \{P_\sigma, p, s\}_{\sigma\in\Sigma}$. Fix $\sigma \in \Sigma$, and let $Atoms(P'_\sigma(x))$ be the set of Boolean expressions in $Atoms(\Sigma)$ which appear in the equation $P'_\sigma(x)$. We can then construct a truth table with columns $\mathbb{C} = atoms(P'_\sigma(x)) \cup \{P'_\sigma(x)\}$. For every $\alpha \in \mathbb{C}$ and row r, we say $r \models \alpha$ if and only if α evaluates to $\top$ in row r of the table. For every row r, we define a conjunction consisting of all the information contained in row r as follows

$$\Psi_r(x) := \bigwedge_{\alpha\in Atoms(P'_\sigma(x))} \begin{cases} \alpha & \text{if } r \models \alpha \\ \neg\alpha & \text{otherwise} \end{cases}$$

We show that there is a normal form expression P^*_σ such that P'_σ is logically equivalent to P^*_σ.

Case 1. Consider first the case where $P_\sigma(x) \notin \mathbb{C}$. In other words, the equation $P'_\sigma(x)$ is not defined in terms of the input predicate $P_\sigma(x)$. Set $\psi_\sigma(x)$ and $\phi_\sigma(x)$ as follows.

$$\phi_\sigma(x) := \bigvee_{\{r: r\not\models P'_\sigma(x)\}} \Psi_r(x)$$

$$\psi_\sigma(x) := \bigvee_{\{r: r\models P'_\sigma(x)\}} \Psi_r(x)$$

Let $P^*_\sigma(x) =$ if $P_\sigma(x)$ then $\neg\phi_\sigma(x)$ else $\psi_\sigma(x)$. By construction, $\phi_\sigma(x) = \neg\psi_\sigma(x)$, and therefore $P^*_\sigma(x) \equiv \psi_\sigma(x)$. Because $\psi_\sigma(x)$ exhaustively contains all the possible situations in which $P'_\sigma(x)$ evaluates to $\top$, $\psi_\sigma(x) \equiv P'_\sigma(x)$. Thus, P^*_σ is equivalent to P'_σ.

Case 2. $P_\sigma(x) \in atoms(P'_\sigma(x))$. Define $\phi_\sigma(x)$ and $\psi_\sigma(x)$ as follows.

$$\phi_\sigma(x) := \bigvee_{\{r: r\models P_\sigma(x),\, r\not\models P'_\sigma(x)\}} \Psi_r(x) \qquad (1)$$

$$\psi_\sigma(x) := \bigvee_{\{r: r\not\models P_\sigma(x),\, r\models P'_\sigma(x)\}} \Psi_r(x) \qquad (2)$$

Set $P^*(x) =$ if $P_\sigma(x)$ then $\neg\phi_\sigma(x)$ else $\psi_\sigma(x)$. We show that $P^*_\sigma(x)$ evaluates to $\top$ iff $P'_\sigma(x)$ evaluates to $\top$. Consider first the case where $P_\sigma(x)$ evaluates to $\top$. Then $P^*_\sigma(x)$ evaluates to $\top$ iff $\phi_\sigma(x)$ evaluates to $\bot$. By the definition in (1), this means Ψ_r evaluates to $\bot$ for every row r such that $r \not\models P'_\sigma(x)$. In other words, none of the conditions under which $P'_\sigma(x)$ can evaluate to $\bot$ are satisfied. Thus, $\phi_\sigma(x)$ evaluates to $\bot$ iff $P'_\sigma(x)$ evaluates to $\top$. Similarly in the case where $P_\sigma(x)$ evaluates to $\bot$, $P^*_\sigma(x)$ evaluates to $\top$ iff $\psi_\sigma(x)$ evaluates to $\top$. By the definition in (2), $\psi_\sigma(x)$ evaluates to $\top$ iff $\Psi_r(x)$ evaluates to $\top$ for *some* row r. In other words, one of the conditions under which $P'_\sigma(x)$ can evaluate to $\top$ is satisfied. Thus, $\psi_\sigma(x)$ evaluates to $\top$ iff $P'_\sigma(x)$ evaluates to $\top$. □

On the Structure of Sets Testable in the Strict Sense

Dakotah Lambert
Department of Mathematics and Computer Science
Lake Forest College
dakotahlambert@acm.org

Abstract

This paper presents purely algebraic characterizations of the languages locally testable in the strict sense (strictly local) and those piecewise testable in the strict sense (strictly piecewise). These characterizations provide polynomial time algorithms to effectively decide whether a given regular language, presented only as its algebraic structure, belongs to the class: linear time for strictly piecewise and quintic time for strictly local. The techniques described herein are broadly applicable across the subregular hierarchy, admitting a universal approach to deciding membership.

1 Introduction

A longstanding hypothesis in computational phonology is that all phonological surface constraints are regular, and indeed that they require significantly less computational power to recognize or to learn than arbitrary regular languages (Rogers et al., 2013; Heinz, 2018). Each of the many *subregular* language classes provides another perspective on this space; a language within a given class has some additional structure that can facilitate perception and learning (Heinz et al., 2012; Heinz and Rogers, 2013; Rogers et al., 2013; Lambert, 2021; Lambert et al., 2021; Johnson and De Santo, 2024), while languages not in the class do not have this additional structure. A common question regards which single class has enough structure to be effectively learnable but weak enough structure to contain every attested pattern, and much work has gone into the quest to find such a class or to find patterns witnessing that a given class is insufficient (Graf, 2017; Heinz, 2018; Mayer and Major, 2018; De Santo and Graf, 2019; Jardine, 2020). Recently, researchers have investigated the performance of neural networks on various subregular classes, showing a bias toward simpler classes (Bhattamishra et al., 2020; Torres and Futrell, 2023; van der Poel et al., 2024), lending support to the simplicity bias proposed by Lambert et al. (2021).

As abstract algebra is the study of structure, algebraic techniques have long been used by computer scientists to study subregular classes of languages (Straubing, 1985; Tilson, 1987; Almeida, 1995; Pin, 1997; Straubing and Weil, 2021), and these techniques have recently been applied to linguistic study by Lambert (2023) and by Lambert and Heinz (2023, 2024). A semigroup is a set alongside an associative binary operation under which it is closed; this kind of structure naturally represents a formal language when the set is the set of strings over some alphabet Σ and the operation is concatenation. By collapsing strings into equivalence classes that respect the language, each regular and subregular language is associated to a finite semigroup. Many subregular classes are varieties of languages in the sense of Eilenberg (1976), meaning they are completely characterized by equations that hold for every instantiation of variables for all and only the languages in the class (Reiterman, 1982). The Language Toolkit software of Lambert (2024) uses these equations to classify regular languages with respect to a large portion of the subregular hierarchy.

However, to be a variety, a class of languages must be closed under the Boolean operations of union, intersection and complement. Many classes important to the study of natural language patterns are not closed under all of these operations. In this work, we primarily consider the strictly local languages (which operate by forbidding particular substrings, blocks of adjacent symbols) and the strictly piecewise languages (which operate by forbidding particular subsequences, which may have gaps), neither of which is closed under complementation. In order to capture these less-well-behaved classes, one has traditionally turned to external information outside of the semigroup structure. For instance, De Luca and Restivo (1980) first query whether the language accepts words of the form $\Sigma^* w \Sigma^*$ before applying algebraic techniques in characterizing the strictly local languages. Fu et al. (2011) similarly make membership queries as part of their characterization of the strictly piecewise languages.

We show that the strictly piecewise languages can be captured by imposing an order over the semigroup elements that is compatible with multiplication and captures some amount of membership information while not explicitly querying acceptability. The limited information added by this ordering is not, however, sufficient to capture the strictly local languages. For that, we add additional structure, looking not at individual words under concatenation, but instead at sets of words with both concatenation and union: a *join-semigroup* structure. This is a semigroup augmented with an extra operation that forms a semilattice; the resulting structure is akin to a *semiring* with extra requirements on its addition, except that it need not necessarily have neutral elements. Our primary results are that the strictly piecewise languages are characterized by an inequality over their associated monoid structure:

$$1 \leqslant x$$

where 1 is the neutral element of the monoid, and the strictly local languages are characterized by an inequality over an associated join-semigroup structure:

$$ax^{\omega}d \leqslant ax^{\omega}b \vee cx^{\omega}d.$$

In each of these inequalities, all variables (a, b, c, d, and x) are universally quantified over the elements of the algebraic structure in question. The notation x^{ω} refers to the unique element that both is a power of x and squares to itself; this is formally defined in §3. Note that $\vee$ here refers not to a logical disjunction but to the least upper bound in the ordering imposed by the semilattice in the join-semigroup.

2 Preliminaries

Given a finite alphabet Σ, the set Σ^* is the set of all finite strings over Σ. A **formal language** L is a subset of Σ^*. The unique empty string is denoted by λ and the set of nonempty strings is $\Sigma^+ = \Sigma^* - \{\lambda\}$. Henceforth, the word "language" is used unambiguously to mean "formal language". A **semigroup** is a set S alongside a binary operation, usually indicated by adjacency, under which the set is closed and associative: $s(tu) = (st)u$ for all s, t and u in S. A **monoid** is a semigroup with a neutral element 1, where $1s = s = s1$ for all elements s. The **free semigroup** (**free monoid**) over Σ is Σ^+ (resp. Σ^*) under concatenation. The neutral element of such a monoid is $1 = \lambda$.

Define the **Myhill relation** for a subset L of a semigroup S such that $x \sim_L y$ means that for every context $u\_\_v$ it holds that $uxv \in L$ if and only if $uyv \in L$, where u, v, x and y are elements in S. If there is any context that distinguishes x and y, then they are not related. The Myhill relation is an equivalence relation: it is reflexive ($x \sim_L x$), symmetric ($x \sim_L y$ implies $y \sim_L x$), and transitive ($x \sim_L y$ and $y \sim_L z$ together imply $x \sim_L z$).

A relation R on a semigroup S is **stable** if and only if for all elements x, y and z of S it holds that whenever $x \mathrel{R} y$ it is also the case that $zx \mathrel{R} zy$ and $xz \mathrel{R} yz$. That is, a stable relation is a relation that is compatible with the operation of the semigroup.

The Myhill relation is a stable equivalence relation. Denote by $[x]_L$ the equivalence class of x under $\sim_L$. One can form a new semigroup $\Sigma^+/\sim_L$ (or monoid, $\Sigma^*/\sim_L$) whose elements are the $[x]_L$ and whose operation is $[x]_L[y]_L = [xy]_L$. This is the **syntactic semigroup**, $S(L)$ (resp. **syntactic monoid**, $M(L)$) of the language L. The syntactic monoid **recognizes** L, in the sense that L is a union of equivalence classes.

A deterministic finite-state automaton is a five-tuple $\langle Q, \Sigma, \delta, q_0, F\rangle$ where Q is a finite set of states, Σ is a finite alphabet, $\delta \colon \Sigma \to Q \to Q$ is a transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of accepting states. The automaton is in canonical form if for all states q_1 and q_2 there is some string w such that the path labeled w from q_1 leads to a state in F while that from q_2 does not, or vice versa. It is trimmed if the transition function is made partial by removing any states from which no path leads to a state in F.

3 Varieties of Sets and Structures

In this section we present the standard definitions of varieties of sets and pseudovarieties of algebraic structures, which we will refer to simply as varieties, and their importance in the study of formal languages. We briefly demonstrate that these notions are not equipped to handle the classes of languages under discussion in this work.

A class $\mathcal{C}$ of languages associates with each finite alphabet Σ the set $\Sigma\mathcal{C}$ of languages $L \subseteq \Sigma^*$

that are in the class. A $*$**-variety** of languages is a class $\mathcal{V}$ such that each $\Sigma\mathcal{V}$ is a Boolean algebra (*i.e.* it is closed under finite unions, finite intersections, and complementation) and is closed under both left and right quotients in the sense of Brzozowski (1964). Further, if $\varphi : \Gamma^* \rightarrow \Sigma^*$ is a homomorphism (a function such that $\varphi(xy) = \varphi(x)\varphi(y)$ for all x and y and where $\varphi(1) = 1$ where applicable) and L is a language in $\Sigma\mathcal{V}$, then $\varphi^{-1}(L)$ must be in $\Gamma\mathcal{V}$; the class is closed under inverse images of homomorphisms. A $+$**-variety** is exactly the same, except that φ is a nonerasing homomorphism from Γ^+ to Σ^+. One often uses $+$-varieties when dependencies are local, as, if φ is allowed to erase symbols, then φ^{-1} can freely insert those same symbols, destroying any semblance of locality.

A **pseudovariety of monoids** is a set $\mathbf{V}$ of finite monoids that is closed under finitary direct products, quotients, and inverse images of homomorphisms. A **pseudovariety of semigroups** is defined analogously. The "pseudo-" in pseudovariety regards the fact that we are considering only finite structures. Henceforth, the prefix shall be omitted and we will say simply "variety". The direct product of S and T, written $S \times T$ is a semigroup whose elements are of the form $\langle s, t\rangle$ for $s \in S$ and $t \in T$. Multiplication is pointwise, so $\langle s_1, t_1\rangle\langle s_2, t_2\rangle = \langle s_1s_2, t_1t_2\rangle$ If both S and T are monoids, then so too is $S \times T$. A semigroup $\langle T, \cdot\rangle$ is a **subsemigroup** of a semigroup $\langle S, \cdot\rangle$ with the same operation iff $T \subseteq S$ and T is closed under the semigroup operation. Finally, a semigroup T divides a semigroup S if there is some equivalence relation $\sim$ such that for some subsemigroup U of S it holds that $T = U/\sim$.

Eilenberg's theorem (1976, Theorems 3.2–3.4) states that there is a 1–1 correspondence between varieties of monoids and $*$-varieties of languages, and that there is a 1–1 correspondence between varieties of semigroups and $+$-varieties of languages. Reiterman's theorem (1982, Theorem 3.1) states that varieties of semigroups and monoids can be characterized by systems of universally-satisfied equations over profinite words $u = v$. This means that, given the syntactic semigroup or monoid of a language, one can determine whether it belongs to a given variety by merely checking that the equations are satisfied for all (finitely many) instantiations of its variables. If a system includes k variables, then this method decides membership in time $O(n^k)$ where n is the number of elements in the structure.

The variables in the equations can usually be treated as words, but in reality they represent a completion of this space. This work will not discuss topology in depth, but we must introduce one small notion: each element in the equations is the limit of a Cauchy sequence of words. A sequence is **Cauchy** if for any given positive distance, all but finitely many elements of the sequence are within that distance from one another. Following Almeida (1995), we consider the distance between two distinct words to be $2^{-r(w_1,w_2)}$, where $r(w_1, w_2)$ is the number of elements in the smallest monoid that separates w_1 and w_2, and the distance between a word and itself to be zero. This means that words that are only distinct in large monoids are "closer" than words that can be distinguished by small monoids. In the context of regular languages, whose syntactic monoids M are finite, this further means that Cauchy sequences of words map to sequences of Myhill classes (semigroup elements) that are eventually constant, as words with distance below $2^{-|M|}$ cannot be distinguished by M.

The simplest Cauchy sequence of words is x representing the limit of the constant sequence $a_n = x$, where all words are at distance 0 from each other. The sequence $a_n = x^{n!}$ is also Cauchy;[1] its limit is often denoted by x^ω. In any finite semigroup, x^ω is the unique element that is both a positive power of x and **idempotent** (it squares to itself). Oftentimes, the notion of "for all sufficiently long words" is useful; in these instances one turns to elements of the form x^ω.

For example, per Almeida (1989), the piecewise testable languages are all and only those whose syntactic monoids universally satisfy the equation $(xy)^\omega x = (xy)^\omega = y(xy)^\omega$ (or, alternatively both $x^\omega x = x^\omega$ and $(xy)^\omega = (yx)^\omega$). And the locally testable languages are all and only those whose syntactic semigroups universally satisfy both $x^\omega y x^\omega z x^\omega = x^\omega z x^\omega y x^\omega$ and $x^\omega y x^\omega = x^\omega y x^\omega y x^\omega$ (Straubing, 1985). The variety induced by a set of equations is denoted by the equations separated by semicolon (;) within double-brackets $[\![\ldots]\!]$. That is, we say that the variety corresponding to piecewise testable is

$$\mathbf{J} = [\![(xy)^\omega = (yx)^\omega; x^\omega x = x^\omega]\!]_{(*)}$$

[1] Note that $a_n = x^n$ is not Cauchy, as any infinite subsequence will contain words whose length modulo $2|x|$ is 0 and words whose length modulo $2|x|$ is $|x|$, with a distance of at least $2^{-2|x|}$.

and the variety corresponding to locally testable is

$$\begin{aligned}\mathbf{J_1} * \mathbf{D} = [\![x^\omega y x^\omega z x^\omega = x^\omega z x^\omega y x^\omega;\\ x^\omega y x^\omega = x^\omega y x^\omega y x^\omega]\!]_{(+)}\end{aligned}$$

See Straubing (1985) for more information on varieties of the form $\mathbf{V} * \mathbf{D}$; this $*$ is a product of varieties, the details of which exceed the scope of the present work. The "$(*)$" and "$(+)$" subscripts are not standard notation but are used here to distinguish between monoid varieties and semigroup varieties.

However, none of these results can immediately apply in our situation. A language L and its complement $\overline{L}$ share the same syntactic semigroup (or monoid), which means that if $S(L) \in \mathbf{V}$ it must hold that $S(\overline{L}) \in \mathbf{V}$. But the language classes considered in this work are not closed under complementation, so no variety can possibly characterize them. In later sections, we discuss how imposing additional structure allows us to capture classes that do not have all of the closure properties required to be a variety.

4 Strictly Piecewise Languages

In order to reason about long-distance dependencies that arise in the study of natural language constraints, Rogers et al. (2010) examined the languages **piecewise testable in the strict sense**, also called **strictly piecewise**. Per Heinz (2007, 2010a), these languages capture aspects of long-distance harmony patterns that are attested in natural language, such as the sibilant harmony of Tsuut'ina. The following examples taken from Cook (1978) and Li (1930) demonstrate that, in Tsuut'ina, a [+anterior] sibilant like [s] cannot precede [−anterior] sibilants like [ʃ] at any distance, although the reverse is not true.

1.	/sì-tʃɒ́gɒ̀/	ʃìtʃɒ́gɒ̀	'my flank'
	/sì-tsáɣà/	sìtsáɣá	'my hair'
	/gi-si-tˀʃast/	giʃitˀʃast	'they are piled up'

The strictly piecewise languages are defined in terms of a grammar. Let Σ be a fixed finite alphabet. A (negative) **precedence grammar** G is a finite set of finite words over Σ. A word $x = x_1 x_2 \ldots x_n$ is a **subsequence** of another word w, written $x \sqsubseteq w$, if and only if $w = u_0 x_1 u_1 \ldots x_n u_n$ for strings u_i. The language of the grammar G is the set of words $w \in \Sigma^*$ such that $x \sqsubseteq w \Rightarrow x \notin G$. A language is **piecewise testable in the strict sense** (**strictly piecewise**) if and only if it is the language of a precedence grammar. If the longest word in the precedence grammar is at most length k, we say that its language is piecewise k-testable in the strict sense, or alternatively, strictly k-piecewise. Rogers et al. (2010) provide several other characterizations of the class, but the most useful to us here is that the strictly piecewise languages are all and only those that are closed under taking of subsequences.

4.1 Known Decision Procedures

Several algorithms have been proposed to determine whether a given regular language is strictly piecewise, and, perhaps, to obtain its grammar if it is.

Rogers et al. (2010) provide a simple grammar-based decision procedure. The input is presented in the form of a deterministic finite-state automaton in canonical form. First, they show that if a language is properly strictly k-piecewise, then it must contain at least k states. Then, by enumerating all possible grammars whose longest word is at most length k, each can be converted to a finite-state automaton and compared against the original language. If one is equal, then that grammar witnesses the fact that the language is strictly piecewise. Otherwise, the language is not strictly piecewise. This algorithm runs in time $\Theta(|\Sigma|^{|Q|})$, where Σ is the alphabet and Q is the set of states in the input automaton.

Rogers and Lambert (2019a) construct the smallest strictly piecewise superset of the given language by allowing edges in the automaton to be skipped, and test whether the resulting language is equal to the original. This too runs in exponential time, as the construction invokes nondeterminism and the result must be determinized before comparison.

Fu et al. (2011) use some tools from algebra and state that a language is strictly piecewise if and only if it is "wholly nonzero" and "right annihilating". A **zero** in a semigroup is an element 0 such that $0x = 0 = x0$ for all x. Let L be a language and let $\eta : \Sigma^* \to M(L)$ be the **syntactic morphism** mapping $w \mapsto [w]_L$. Fu et al. (2011, Definition 5) define that a language is "wholly nonzero" if and only if $\eta(\overline{L}) = 0$. However, the syntactic monoid for the strictly piecewise language Σ^* has $\eta(L) = 0$ while $\eta(\overline{L})$ does not exist.[2] Per-

[2] Fu et al. (2011, Corollary 2) also claim that strictly local languages are wholly nonzero. We describe strictly local languages in a later section, but at this point it is important to note that the language of words that begin with "a" is strictly

haps a better phrasing would be that L is wholly nonzero if and only if $w \in \overline{L} \Rightarrow \eta(w) = 0$. Fu et al. (2011, Definition 7) define that a language is "right annihilating" if and only if its monoid satisfies $xa = 0 \Rightarrow xya = 0$. Modulo the trivial languages Σ^* and $\varnothing$, this provides a mechanism by which one can decide in $O(n^3)$ time whether a given monoid represents a strictly piecewise language, assuming that the elements are marked for whether they contain accepted words or rejected words.

In the following sections, we present a different algebraic characterization that admits a linear-time decision procedure and more closely resembles how varieties of languages are defined.

4.2 Closure Properties

In this section we explore the closure properties of the class of strictly piecewise languages and see where they fail to be a variety. From there, we present a piece of additional structure that allows algebraic techniques to make the distinctions necessary in order to characterize the class. Finally, we present a characterization that admits a decision procedure that operates in linear time when given the appropriate kind of input. Recall that to be a variety, a class $\mathcal{C}$ must be such that each $\Sigma\mathcal{C}$ is a Boolean algebra closed under left- and right-quotients, and the class must be closed under taking inverse-images of homomorphisms.

First, the strictly piecewise languages over Σ are not a Boolean algebra. They are not closed under complementation. If Σ is nonempty, the language containing only the empty string $L = \{\lambda\}$ is strictly piecewise, but its complement is not, as $\lambda \notin \overline{L}$ despite the fact that $\lambda \sqsubseteq w$ for all w. They are, however, closed under both union and intersection. Consider the union. Let L_1 and L_2 be strictly piecewise languages and consider their union $L_1 \cup L_2$. Let $w \in L_1 \cup L_2$ and let $x \sqsubseteq w$. Then either $w \in L_1$ or $w \in L_2$. In the case that $w \in L_1$, because L_1 is strictly piecewise, it follows that $x \in L_1$ and therefore $x \in L_1 \cup L_2$. Alternatively, $w \in L_2$ and, because L_2 is strictly piecewise, it follows that $x \in L_2$ and therefore $x \in L_1 \cup L_2$. In either case, the union is closed under subsequence and is therefore strictly piecewise. Next consider the intersection $L_1 \cap L_2$. Let $w \in L_1 \cap L_2$ and let $x \sqsubseteq w$. Then $w \in L_1$ and $w \in L_2$, and as each is strictly piecewise, it follows that $x \in L_1$ and $x \in L_2$, so $x \in L_1 \cap L_2$. The intersection is closed under subsequence and is therefore strictly piecewise. As they are closed under both union and intersection, the strictly piecewise languages over Σ are a **positive Boolean algebra**.[3]

The strictly piecewise languages over Σ are closed under left- and right-quotients in the sense of Brzozowski (1964). The left quotient of a language L over Σ by a symbol $a \in \Sigma$ is

$$a^{-1}L = \{w \in \Sigma^* : aw \in L\}$$

and the right quotient is

$$La^{-1} = \{w \in \Sigma^* : wa \in L\}.$$

Let $a \in \Sigma$, let w be in $a^{-1}L_1$, and let $x \sqsubseteq w$. Then $aw \in L_1$ and because $x \sqsubseteq w$, and therefore $ax \sqsubseteq aw$, it follows that $ax \in L_1$ and $x \in a^{-1}L_1$. Similarly, if $w \in L_1a^{-1}$ then $x \in L_1a^{-1}$. Both quotients are closed under subsequence and thus strictly piecewise.

Finally, the strictly piecewise languages are closed under inverse images of homomorphisms. Let $\varphi \colon \Gamma^* \to \Sigma^*$ be a **homomorphism**, a function where $\varphi(xy) = \varphi(x)\varphi(y)$. Also let $L \subseteq \Gamma^*$ be the language

$$L = \varphi^{-1}(L_1) = \{w : \varphi(w) \in L\}.$$

Let $w \in L$ and let $x \sqsubseteq w$. This means that $x = x_1 \ldots x_n$ for some n and there exist $u_0 \ldots u_n$ such that $w = u_0x_1u_1 \ldots x_nu_n$. We have that $w \in \varphi^{-1}(L_1)$ and so it follows that $\varphi(w) = \varphi(u_0)\varphi(x_1)\varphi(u_1) \ldots \varphi(x_n)\varphi(u_n)$ is in L_1. As L_1 is closed under subsequence, the subsequence $\varphi(x_1)\varphi(x_2) \ldots \varphi(x_n) = \varphi(x)$ of $\varphi(w)$ is also in L_1, so $x \in \varphi^{-1}(L_1)$. The inverse image of φ is closed under subsequence and therefore it is strictly piecewise.

Definition 1 (Pin, 1997)**.** A **positive $*$-variety of languages** is a class that is closed under taking inverse images of homomorphisms and assigns to each alphabet Σ a set of languages that is a positive Boolean algebra closed under the left- and right-quotient operations.

Lemma 1. *The strictly piecewise languages form a positive $*$-variety.*

local but is decidedly *not* wholly nonzero whenever $\{a\} \subsetneq \Sigma$, as its syntactic monoid does not even have a zero.

[3] The term "positive Boolean algebra" is common in French treatments of formal language theory. Another name for a Boolean algebra without complementation (a structure that is isomorphic to a class of sets that is closed under finitary union and intersection) is a "distributive lattice".

4.3 Ordered Semigroups and Monoids

A **preorder** is a relation $\leqslant$ that is both reflexive and transitive. Let S be a semigroup. An **ordered semigroup** is a semigroup equipped with a stable preorder. Every semigroup may be treated as an ordered semigroup under the trivial order where $x \leqslant y$ means $x = y$.

The **syntactic order** of a language L is the ordering defined such that $[s] \leqslant [t]$ if and only if $utv \in L \Rightarrow usv \in L$ for all strings u and v in Σ^* (Pin, 1997).[4] This is a stable preorder. The **syntactic ordered semigroup** of L is its syntactic semigroup endowed with the syntactic order.

Varieties of ordered semigroups and monoids are defined analogously to their unordered counterparts. Pin (1995, Theorems 5.7 and 5.8) citing to Bloom (1976) extends Eilenberg's theorem to provide a 1–1 correspondence between varieties of ordered monoids (ordered semigroups) and positive $*$-varieties (resp. positive $+$-varieties) of languages. Pin and Weil (1996, Theorem 3.3) extend Reiterman's theorem to show that varieties of ordered monoids or semigroups are characterized by universally satisfied sets of inequalities. In either case, a structure belongs to the variety if and only if the equations or inequalities are satisfied for every instantiation of the variables involved.

As we have shown that the strictly piecewise languages are a positive $*$-variety of languages, this means that there is a corresponding variety of ordered monoids.

Theorem 1. *Let L be a language. The following are equivalent.*

- *L is strictly piecewise*
- *$M(L)$ is a finite ordered monoid in the variety $[\![1 \leqslant x]\!]_{(*)}$*

Proof. First, we show that the syntactic ordered monoid of any strictly piecewise language is finite and satisfies the inequality $1 \leqslant x$ for all x. Finiteness is trivial, as strictly piecewise languages are regular and the defining characteristic of a regular language is that it has a finite syntactic monoid (Rabin and Scott, 1959). Let L be a strictly piecewise language over Σ whose syntactic ordered monoid is $M(L)$. Further, let $uxv \in L$ for strings u and v in Σ^*. Then, as $uv \sqsubseteq uxv$ we have that $uv \in L$ as well. By the definition of the syntactic order, we have that $1 \leqslant x$.

Next, we show that any language whose syntactic ordered monoid is finite and satisfies the inequality $1 \leqslant x$ for all x must be strictly piecewise. Let L be a language whose syntactic monoid $M(L)$ is finite and universally satisfies the inequality and let $\eta : \Sigma^* \to M(L)$ be the syntactic morphism. Let $x = x_1 \ldots x_n$ and $w = u_0 x_1 u_1 \ldots x_n u_n$ be strings such that $x \sqsubseteq w$. Because $\leqslant$ is stable we have the following.

$$\begin{aligned}\eta(x) &= \eta(x_1)\eta(x_2)\ldots\eta(x_n)\\ &\leqslant \eta(u_0)\eta(x_1)\eta(u_1)\ldots\eta(x_n)\eta(u_n)\\ &= \eta(w)\end{aligned}$$

By the definition of the syntactic order, then, we have that whenever $uwv \in L$ it is also the case that $uxv \in L$. By instantiating $u = v = \lambda$ we have $w \in L \to x \in L$. Thus, L is closed under subsequence and is strictly piecewise. □

Recall that two strings x and y map to the same equivalence class if and only if in all contexts u_v it holds that both $uxv \in L$ and $uyv \in L$, or neither is. This means that L and its complement, $\overline{L}$, share the same syntactic monoid. However, their syntactic *ordered* monoids are duals: whenever $[x] \leqslant [y]$ in $M(L)$, we have that for all u and v it holds that $uyv \in L \to uxv \in L$. By contrapositive, we have $uxv \notin L \to uyv \notin L$, so $[y] \leqslant [x]$ in $M(\overline{L})$.

Corollary. *A language L is the complement of a strictly piecewise language if and only if its ordered syntactic monoid is in the variety $[\![x \leqslant 1]\!]_{(*)}$.*

Corollary. *If a language and its complement are both strictly piecewise, then for all x, it holds that $1 \leqslant x \leqslant 1$, i.e. $x = 1$. The only such languages are the empty set and its complement.*

As a result of this characterization, there is an algorithm that decides in linear time whether a given regular language is strictly piecewise (or the complement of a strictly piecewise language), when the language is presented as its ordered syntactic monoid, or even as its syntactic order alone. For each element x, compare x with 1. If for all comparisons, $1 \leqslant x$, then the language is strictly piecewise. If for all comparisons, $x \leqslant 1$, then the complement of the language is strictly piecewise.

[4] Pin (2017) states that this definition was "regrettable" and that the inverse order should be used instead. However, the choice of orientation does not affect the results, and using $\leqslant$ in this sense connects more strongly to the additional structure imposed in the next section for strictly local languages. All this means is that one must be careful to consult the definitions used in any given work and to swap the roles of $\leqslant$ and $\geqslant$ where appropriate.

This class has been studied before under many names. The strictly piecewise languages that we know and love are known by Pin (1997) as "the complements of languages at the one-half level of the Straubing–Thérien hierarchy" and the order-theoretic characterization of this class is presented in that work. And before that, Haines (1969) also studied the languages closed under subsequence (and their complements: those closed under super-sequence).

It is also worth noting that every variety of languages is a positive variety of languages, and every variety of semigroups (monoids) is a variety of ordered semigroups (resp. monoids) where identities of the form $x = y$ are replaced by the pair of inequalities $x \leqslant y$ and $y \leqslant x$. Therefore this structure provides sufficient information to capture varieties such as the piecewise testable and locally testable languages as well.

5 Strictly Local Languages

Just as the strictly piecewise languages are a restriction of the piecewise testable languages, the strictly local languages are a restriction of the locally testable languages. Introduced by McNaughton and Papert (1971) the **languages locally k-testable in the strict sense** (**strictly k-local**) are defined by a grammar $G \subseteq (\Sigma \cup \{\rtimes, \ltimes\})^k$: a word w is accepted if and only if every sliding window of size k of $\rtimes w {\ltimes}^{k-1}$ is in G (Rogers and Pullum, 2011). A language is **strictly local** if and only if it is strictly k-local for some k. Per Edlefsen et al. (2008), approximately 75% of the more than one hundred distinct patterns governing stress assignment in natural language catalogued by Goedemans et al. (2015) are strictly local; Rogers and Lambert (2019a) found that 92.5% of these patterns were covered by a conjunction of a strictly piecewise constraint, a strictly local constraint, and a constraint whose complement is strictly local. The class handles restrictions on prefixes, on suffixes, and on what symbol clusters are permissible.

As reported by Edlefsen et al. (2008) and by Rogers and Pullum (2011), another characterization of the strictly local languages, adapted from McNaughton (1974), is **suffix substitution closure**: L is a strictly k-local language if and only if whenever $axb \in L$ and $cxd \in L$ are strings such that $|x| \geqslant k - 1$, it also holds that $axd \in L$.

5.1 Known Decision Procedures

The strictly local languages are all and only those in which all sufficiently long words w are **constant** in the canonical finite-state automaton: there exists a state q' such that for all q, w maps into the set $\{q', 0\}$, where 0 is the unique rejecting sink state, if any (De Luca and Restivo, 1980).

This means that, if the sink state is removed, then there will be no cycles involving states that correspond to sets of size greater than one in the automaton that results from applying the powerset construction used for determinization with an initial state corresponding to the set of all original states. This construction was used by Rogers and Lambert (2019a) to facilitate both the decision procedure and to recover the grammar, if one exists. This procedure runs in exponential time if presented with the trimmed canonical automaton, or in linear time if given the powerset graph.

Edlefsen et al. (2008) present a modification of this algorithm from Caron (2000) that runs in quadratic time when given the trimmed canonical automaton: consider only the graph generated by states that correspond to pairs in the original automaton. As it is deterministic, no larger sets will ever be generated.

De Luca and Restivo (1980) use the constancy property to show that a language that does not contain two-sided ideals $\Sigma^* w \Sigma^*$ is strictly local if and only if for any idempotent x^ω of its syntactic semigroup, $x^\omega y x^\omega \subseteq \{x^\omega, 0\}$. (If there is no 0, this becomes $x^\omega y x^\omega = x^\omega$.) As a two-sided ideal contains every possible length-k substring, the only strictly local language to contain such ideals is Σ^*, so this also results in an effective characterization. This subset property can be checked in quadratic time when presented with a syntactic semigroup.

All of these techniques employ information not contained in the algebraic structure of the language. In the next section, we demonstrate why even ordered semigroups cannot characterize the strictly local languages, then we present a related structure that does contain sufficient information to capture the class in a purely algebraic way. The resulting characterization strongly resembles the statement of the suffix substitution closure property.

5.2 Closure Properties

The strictly local languages over a fixed alphabet Σ are not a Boolean algebra nor even a positive Boolean algebra, as they are not closed under complementation nor are they closed under union. To witness lack of closure under complementation, consider the set $L_1 = \Sigma^* - \Sigma^* ab \Sigma^*$, the strictly 2-local language in which the ab substring does not occur. Its complement then, is $\Sigma^* ab \Sigma^*$, but as discussed in the preceding section, the only strictly local language to contain a two-sided ideal is Σ^* itself. Every possible substring appears in $\overline{L_1}$, so every word must be accepted for this to be strictly local, but bb is not. The reversal of L_1 is $L_2 = \Sigma^* - \Sigma^* ba \Sigma^*$, another strictly 2-local language, in which the ba substring does not occur. To witness lack of closure under union, we consider $L_1 \cup L_2$. This is the language in which either there is no ab or there is no ba; accepted words are all and only those that do not contain both ab and ba. This does not satisfy suffix substitution closure, as $a(b^k)b \in L$ and $b(b^k)a \in L$ but $a(b^k)a \notin L$ for all $k \geqslant 1$.

This means that the strictly local languages cannot be captured by a variety of (ordered) semigroups. A variety of semigroups is insufficient because a language and its complement share the same semigroup. A variety of ordered semigroups is insufficient because, while a language and its complement can be distinguished, the inequalities used entail closure under union. Recall that $x \leqslant y$ means that $uyv \in L \Rightarrow uxv \in L$ for all $u, v \in \Sigma^*$. Let L_1 and L_2 be languages universally satisfying the inequality and let u, v, and y be strings such that $uyv \in L_1 \cup L_2$. Then either $uyv \in L_1$ and by the inequality we have $uxv \in L_1$, or $uyv \in L_2$ and by the inequality we have $uxv \in L_2$. In either case, $uxv \in L_1 \cup L_2$, and the inequality is satisfied.

However, the strictly local languages over Σ are closed under intersection. Let L_1 and L_2 be any strictly k-local languages over Σ, and let axb and cxd be words in $L_1 \cap L_2$ such that $|x| \geqslant k-1$. Then, because each of axb and cxd are in each of L_1 and L_2, it follows by suffix substitution closure that axd is in both L_1 and L_2. That is, $axd \in L_1 \cap L_2$ and suffix substitution closure holds. The result is strictly k-local.

The class is also closed under left and right quotients. Let L be a strictly k-local language over Σ and let axb and cxd be words in $s^{-1}L$ for some $s \in \Sigma$ where $|x| \geqslant k-1$. Then $saxb$ and $scxd$ are in L, which by suffix substitution closure means that $saxd \in L$ and $axd \in s^{-1}L$. Suffix substitution closure holds, so $s^{-1}L$ is strictly k-local. Similarly, Ls^{-1} is strictly k-local.

The class is not closed under inverse images of arbitrary homomorphisms; indeed the tier-based strictly local languages are defined as the inverse image of a strictly local language under a particular kind of erasing homomorphism (Heinz et al., 2011; Lambert, 2023). But the class is closed under inverse images of nonerasing homomorphisms. Let $\varphi\colon \Gamma^+ \to \Sigma^+$ be a (nonerasing) homomorphism Also let L be a strictly k-local language over Σ^*, and let axb and cxd be words in $\varphi^{-1}(L)$ such that $|x| \geqslant k-1$. Then $\varphi(axb) = \varphi(a)\varphi(x)\varphi(b) \in L$ and $\varphi(cxd) = \varphi(c)\varphi(x)\varphi(d) \in L$, and because φ is nonerasing, $|\varphi(x)| \geqslant |x| \geqslant k-1$. By suffix substitution closure, this means that $\varphi(axd) = \varphi(a)\varphi(x)\varphi(d) \in L$ and therefore $axd \in \varphi^{-1}(L)$. This demonstrates closure under inverse images of nonerasing homomorphisms.

Indeed, the strictly local languages satisfy a stronger condition. The following is adapted from Polák (2001). For a set S define $S^{\square}$ to be the set of all nonempty finite subsets of S^*, and $S^{\boxplus}$ to be the set of all nonempty finite subsets of S^+. Let $\psi\colon \Gamma^{\boxplus} \to \Sigma^{\boxplus}$ be a (nonerasing) homomorphism where the operation is concatenation lifted to sets, and define for $L \subseteq \Sigma^*$ an inverse:

$$\psi^{[-1]}(L) = \{w \in \Gamma^+ : \psi(\{w\}) \subseteq L\}$$

Let $L \subseteq \Sigma^*$ be strictly k-local, and let axb and cxd be words in $\psi^{-1}(L)$ such that $|x| \geqslant k-1$. Observe the following (omitting $\psi(\{w\})$ if $w = \lambda$):

$$\psi(\{axb\}) = \psi(\{a\})\psi(\{x\})\psi(\{b\}) \subseteq L$$
$$\psi(\{cxd\}) = \psi(\{c\})\psi(\{x\})\psi(\{d\}) \subseteq L$$

As every word in $\psi(\{x\})$ is at least as long as x, it holds that by suffix substitution $\psi(\{a\})\psi(\{x\})\psi(\{d\}) \subseteq L$ and therefore it holds that $axd \in \psi^{[-1]}(L)$. We have that $\psi^{[-1]}$ is strictly k-local.

Definition 2 (Polák, 2001)**.** A **conjunctive $+$-variety of languages** is a class that is closed under $\psi^{[-1]}$ for homomorphisms of the form $\psi\colon \Gamma^{\boxplus} \to \Sigma^{\boxplus}$ and that assigns to each alphabet Σ a set of languages that is closed under intersection and the left- and right-quotient operations.

Lemma 2. *The strictly local languages form a conjunctive $+$-variety.*

A **conjunctive $*$-variety** is defined analogously, using $^{\square}$ in place of $^{\boxplus}$.

5.3 Join-Semigroups

The following concepts are adapted from Polák (2001) and Klíma and Polák (2019). A **join-**

semigroup is a structure $\langle S, \cdot, \vee \rangle$ such that $\langle S, \cdot \rangle$ is a semigroup and $\langle S, \vee \rangle$ is a join-semilattice (a commutative semigroup where all elements are idempotent) alongside a distributive property: for all x, y, and z in S it holds that $x(y \vee z) = xy \vee xz$ and $(x \vee y)z = xz \vee yz$.[5] A join-monoid is a join-semigroup where $\langle S, \cdot \rangle$ is a monoid.

The usual ordering relation for a join-semilattice induces an order on the structure: $x \leqslant y$ means $x \vee y = y$. This is stable under multiplication. Let $x \leqslant y$ and let a be an element. Then $x \vee y = y$ and so $a(x \vee y) = ay$. By the distributive property, we have $ax \vee ay = ay$ and thus $ax \leqslant ay$. Similarly, $xa \leqslant ya$. The order is clearly also stable under join, as $a \vee (x \vee y) = a \vee y$ and $(x \vee y) \vee a = y \vee a$.

The **syntactic join-semigroup** of a language $L \subseteq \Sigma^*$, is the join-semigroup $\langle \Sigma^{\boxplus}, \cdot, \cup \rangle / {\approx_L}$, where $A \approx_L B$ means that for all sets U and V in $\Sigma^{\square}$ it holds that $UAV \subseteq L$ if and only if $UBV \subseteq L$. This is a set-based analogy to the Myhill relation. As before, $\Sigma^{\square}$ and $\Sigma^{\boxplus}$ are defined to be the set of nonempty finite subsets of Σ^* and Σ^+, respectively. The **syntactic join-monoid** is defined analogously. Note that for a regular language, this structure is finite, as one can consider the elements $\eta(w)$ of the syntactic semigroup or monoid rather than the strings w themselves.

Polák (2001) extends Eilenberg's and Reiterman's theorems to provide a 1–1 correspondence between varieties of join-monoids (join-semigroups) and conjunctive $*$-varieties (resp. conjunctive $+$-varieties) of languages and to provide a means for characterization by a system of inequalities. The inequalities are of the form $x \leqslant y_1 \vee y_2 \vee \cdots \vee y_n$ for $n \geqslant 1$, and can be interpreted as requiring that whenever it holds for all strings u and v that $uy_iv \in L$ for all $1 \leqslant i \leqslant n$, it also holds that $uxv \in L$.

As we have shown that the strictly local languages are an conjunctive $+$-variety of languages, this means that there is a corresponding variety of join-semigroups. To get there, we first must show that all sufficiently long words can be written in terms of an idempotent.

Lemma 3. *Let S be a finite semigroup and let x be a sequence of elements of S of length $k > |S|$. Then $x = ae^{\omega}b$ for some a, b and c in S.*

Proof. If any individual x_i is idempotent, then $x = (x_1 \ldots x_{i-1})(x_i)(x_{i+1} \ldots x_k)$ and we are done. Otherwise, consider the prefixes $x_{[i]} = x_1 \ldots x_i$. By the pigeonhole principle, there exist $i < j$ such that $x_{[i]} = x_{[j]} = x_{[i]}(x_{i+1} \ldots x_j)$. Let $a = x_{[i]}$ and $e = x_{i+1} \ldots x_j$. By iteratively expanding, we have $a = ae = aee = \cdots = ae^{\omega}$. Finally, let $b = x_{j+1} \ldots x_k$ if $j < k$ else e^{ω}. We then have $x = ae^{\omega}b$. □

Theorem 2. *Let L be a language. The following are equivalent.*

- *L is strictly local*
- *The syntactic join-semigroup of L is in the variety $[\![ax^{\omega}d \leqslant ax^{\omega}b \vee cx^{\omega}d]\!]_{(+)}$*

Proof. First, we show that the syntactic join-semigroup of any strictly local language is finite and satisfies the inequality $ax^{\omega}d \leqslant ax^{\omega}b \vee cx^{\omega}d$. Finiteness is trivial, as strictly local languages are regular. Let L be a strictly local language over Σ. Then there is some k for which L is strictly k-local. Further, let $uaebv$ and $ucedv$ belong to L for strings a, b, e, u, and v where $\eta(e) = \eta(e)\eta(e)$. That is, $\eta(e)^{\omega} = \eta(e)$. Then $ua(e^k)bv$ and $uc(e^k)dv$ are in L, as they are Myhill-equivalent to $uaebv$ and $ucedv$, respectively. By suffix-substitution, it follows that $ua(e^k)dv \in L$, that $uaedv \in L$. This demonstrates satisfaction of the inequality.

Next, we show that any language whose syntactic join-semigroup, Z, is finite and satisfies the inequality must be strictly local. Specifically, we show that it is strictly k-local for $k = |Z| + 2$. Let L be a regular language with finite syntactic join-semigroup Z satisfying the inequality, and let axb and cxd be words in L such that $|x| \geqslant k - 1$. In the case that $\eta(x) = \eta(x)^{\omega}$, the inequality directly guarantees that $axd \in L$. It is not necessarily the case that all long words are themselves idempotent, but, by Lemma 3, we can rewrite x as x_1ex_2 where $\eta(e) = \eta(e)^{\omega}$ to find that $ax_1ex_2d \in L$. This means that $axd \in L$, that suffix substitution closure is satisfied, and hence L is strictly k-local. □

A consequence of this is an effective algorithm for deciding whether a language is strictly local: for all instantiations of the five variables a, b, c, d and x^{ω}, where x^{ω} is restricted to idempotents, verify that $ax^{\omega}d \leqslant ax^{\omega}b \vee cx^{\omega}d$. This is not a particularly efficient algorithm; naïvely it runs in quintic

[5] Polák (2001) used the term *idempotent semiring*, with $\vee$ being $+$, but this is an abuse of terminology, as, properly, the definition of a semiring requires that both operations impose a monoid structure, with the neutral element of addition being 0 such that $0x = 0 = x0$ for all x. However, there appears to be no standard terminology for two interacting *semigroups* rather than two interacting *monoids*.

time in the size of the syntactic join-semigroup, as there are five variables. However, it does provide a purely algebraic characterization of the strictly local languages, unifying their analysis with that of other classes in the piecewise-local subregular hierarchy.

It also allows for a new proof of the obvious fact that strictly local languages are locally testable.

Proposition 1. *Let L be a language whose syntactic join-semigroup Z belongs to*

$$[\![ax^\omega d \leqslant ax^\omega b \vee cx^\omega d]\!]_{(+)},$$

i.e. let L be strictly local. Then L is locally testable, i.e. Z belongs to

$$\begin{aligned}\mathbf{J_1} * \mathbf{D} = [\![x^\omega y x^\omega z x^\omega = x^\omega z x^\omega y x^\omega;\\ x^\omega y x^\omega = x^\omega y x^\omega y x^\omega]\!]_{(+)}.\end{aligned}$$

Proof. Let L be a strictly local language and let Z be its syntactic join-semigroup. In the following analyses, for clarity, we will parenthesize the shared idempotent that enables substitution of suffixes. Consider the element $x^\omega y x^\omega$ for all x and y. We have by strict locality and by the idempotence of both x^ω and $\vee$ that

$$\begin{aligned}x^\omega y x^\omega y x^\omega &\leqslant x^\omega y(x^\omega)x^\omega \vee x^\omega(x^\omega)yx^\omega\\ &= x^\omega y x^\omega.\end{aligned}$$

And we also have that

$$\begin{aligned}x^\omega y x^\omega &\leqslant x^\omega y(x^\omega)yx^\omega \vee x^\omega y x^\omega y(x^\omega)x^\omega\\ &= x^\omega y x^\omega y x^\omega.\end{aligned}$$

As each is less than or equal to the other,

$$x^\omega y x^\omega = x^\omega y x^\omega y x^\omega$$

Next, consider an element of the form $x^\omega y x^\omega z x^\omega$. Let $e = x^\omega$. We have by strict locality and by the idempotence of both $e = x^\omega$ and $\vee$ that

$$\begin{aligned}eyeze &\leqslant e(e)zeyeze \vee ez(e)yeze\\ &= ezeyeze\\ &\leqslant ezeyez(e)ye \vee ezeyezey(e)e\\ &= ezeyezeye.\end{aligned}$$

By the preceding argument this final form reduces:

$$x^\omega(zx^\omega y)x^\omega(zx^\omega y)x^\omega = x^\omega z x^\omega y x^\omega$$

So, $x^\omega y x^\omega z x^\omega \leqslant x^\omega z x^\omega y x^\omega$ by transitivity. By a similar argument, the reverse holds as well and the two are equal. Both equations that characterize local testability are universally satisfied. □

Without using any external knowledge about how the classes are defined or how they interact, we showed purely by symbolic manipulation that every language locally testable in the strict sense is locally testable.

5.4 Positive Varieties as Conjunctive Varieties

Recall that a conjunctive variety is like a positive variety except that closure under union is not required but additionally the class must be closed under inverse images homomorphisms of the form $\psi\colon A^{\square} \to B^{\square}$ (or $\psi\colon A^{\boxplus} \to B^{\boxplus}$). It turns out that every positive variety is a conjunctive variety of the same type.

Theorem 3. *Let $\mathcal{V}$ be a positive $*$-variety ($+$-variety). Then $\mathcal{V}$ is also a conjunctive $*$-variety (resp. $+$-variety). Moreover, the characteristic inequalities are identical.*

Proof. Let $\mathcal{V}$ be a positive variety and let ψ be a homomorphism from $A^{\square}$ or $A^{\boxplus}$ to $B^{\square}$ or $B^{\boxplus}$, as appropriate. Also let $L \subseteq B\mathcal{V}$. For all words $w \in A^*$, the output $\psi(\{w\})$ is completely characterized by the output $\psi(\{a\})$ for $a \in A$. Because $B^{\square}$ and $B^{\boxplus}$ contain only *finite* sets, it is the case that $\psi(\{a\})$ is finite for all $a \in A$. Construct $\widehat{A}$ by creating a symbol a_i for each $a \in A$ and each $1 \leqslant i \leqslant |\psi(\{a\})|$.

As there are finitely many elements of A and each maps to finitely many objects, there are finitely many functions f that map each pair $\langle a, i\rangle$ (with $a \in \widehat{A}$ and $1 \leqslant i \leqslant |\psi(\{a\})|$) to an element of $\psi(\{a\})$. Let F be the collection of these functions. Construct $\widehat{\psi}_f\colon \widehat{A}^* \to B^*$ such that $\widehat{\psi}_f(a_i) = f(a, i)$. As $\mathcal{V}$ is a positive variety, we have that each $\widehat{\psi}_f^{-1}(L) \in \widehat{A}\mathcal{V}$. The intersection of this family, $X = \bigcap_{f \in F} \widehat{\psi}_f^{-1}(L)$, is also in $\widehat{A}\mathcal{V}$, as positive varieties are closed under finitary intersections. The intersection accounts for the restriction that $\psi(\{w\}) \subseteq L$, rather than capturing only nondisjointness.

Moreover, as every permutation of the indices is handled by some $f \in F$, it is the case that $a_i \sim_X a_j$ for all appropriate i and j. Let $\#$ be the stable subrelation of the Myhill relation where $a_i \# a_j$ for all i and j but no other elements are related. Let S be the syntactic ordered semigroup (ordered monoid) of X; then $S/\#$ is a quotient of S and therefore the corresponding language is in $A\mathcal{V}$.

Finally, that the characteristic inequalities are identical is from the definition. An inequality for an ordered variety of the form $x \leqslant y$ means that $uyv \in L \Rightarrow uxv \in L$, which is exactly what $x \leqslant y$ means under the semilattice ordering. □

Corollary. *The strictly piecewise languages are the conjunctive $*$-variety that corresponds to the variety of join-monoids* $\mathbf{J}^- = [\![1 \leqslant x]\!]_{(*)}$.

It is for this reason that we chose to use the syntactic order as presented by Pin (1997) rather than the inverse order as used by Pin (2017).

This means that the syntactic join-monoid or join-semigroup provides sufficient information to capture varieties such as the piecewise testable and locally testable languages as well as positive varieties such as the strictly piecewise languages.

6 Constructing Join-Semigroups

For completeness, this section briefly describes the construction of the syntactic join-semigroup or join-monoid detailed by Polák (2001). Let L be a language over Σ recognized by a deterministic finite-state automaton $\mathcal{A}$ in canonical form, whose state set is Q and whose transition function is $\delta\colon \Sigma \to Q \to Q$. Define $\mathcal{L}(q)$ to be the language of the state q, the set of permissible continuations from that state. Fix for each equivalence class $[w]$ in Σ^+ (Σ^*) under $\sim_L$ a shortest representative w such that $\eta(w) = [w]$ and let W be this set of representatives. Construct a modified automaton with state set

$$\widehat{Q} = \Big\{\bigcap_{q\in S} \mathcal{L}(q) : S \subseteq Q\Big\}$$

and transition function

$$\widehat{\delta}\colon \mathcal{P}(W) - \{\varnothing\} \to \widehat{Q} \to \widehat{Q}$$

where $\widehat{\delta}(\{a\})(\bigcap_{q\in S}\mathcal{L}(q)) = \bigcap_{q\in S}\mathcal{L}(\delta(a)(q))$ for $S \subseteq Q$. This transition function is implicitly extended to (singleton sets of) finite words in the typical way, and is then extended again to arbitrary sets by $\widehat{\delta}^*(T)(q) = \bigcap_{w\in T}\widehat{\delta}(\{w\})(q)$. The syntactic join-semigroup (join-monoid) of the language is the transition semigroup of this automaton (Polák, 2001), *i.e* the structure

$$\langle\{\widehat{\delta}^*(T) : T \in \mathcal{P}(W) - \{\varnothing\}\}, \cdot, +\rangle$$

where $\cdot$ is $\widehat{\delta}^*(T_1)\widehat{\delta}^*(T_2) = \widehat{\delta}^*(T_2) \circ \widehat{\delta}^*(T_1)$ and $+$ is $\widehat{\delta}^*(T_1) + \widehat{\delta}^*(T_2) = \widehat{\delta}^*(T_1 \cup T_2)$. Whether it is the join-semigroup or join-monoid depends on whether $[\lambda]$ is included in W.

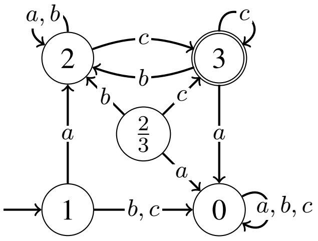


Figure 1: A strictly local language L, states extended.

6.1 Example Computation

Consider for example the strictly 2-local language L over $\Sigma = \{a, b, c\}$ consisting of all and only those words which begin with 'a', end with 'c', and do not contain "ca" as a substring. Its canonical finite-state automaton is shown in Figure 1 restricted to states $\{0, 1, 2, 3\}$. State 1 corresponds to the language itself. In state 2, the initial a has been accounted for, so its language is simply the set of continuations which end with 'c' and do not contain "ca" as a substring. In state 3, the initial 'a' has been accounted for and a 'c' has just been read: its language is the set of continuations which end with neither 'a' nor 'b' and do not contain "ca" as a substring (much like state 2, modulo λ), with the added restriction that the continuation not start with 'a', as, if it does, this creates a "ca" substring in the broader word. Finally state 0 is the rejecting sink.

The only nonempty subset of states whose languages intersect to yield a language not already accounted for is $\{2, 3\}$, whose intersection is the language of state 3 minus the empty string. The state $\frac{2}{3}$ represents this state in $\overline{Q}$. The empty intersection is Σ^* and is unaffected by transformations and therefore omitted.

By calculating the transition semigroup in the usual way, one finds that there are five elements in the syntactic semigroup of L, with representatives "a", "b", "c", "ac", and "ca". These five words represent every possible function over the original state set $\{0, 1, 2, 3\}$ that arise from any word in Σ^+. These elements multiply as shown in Table 1.

For each state q in $\overline{Q}$ and for each nonempty subset of semigroup elements, follow the paths labeled by each semigroup element in the subset, and construct the intersection of the resulting states. For example, $\{a, c\}$ acts on state 2 by having 'a' go to state 2 while 'c' goes to state 3; to simulate having done both of these at once, the resulting state is $\{2, 3\}$ (represented in Figure 1 as $\frac{2}{3}$). In

Table 1: Multiplication of semigroup elements of L, idempotents highlighted.

	a	b	c	ac	ca
a	a	a	ac	ac	ca
b	b	b	c	c	ca
c	ca	b	c	ca	ca
ac	ca	a	ac	ca	ca
ca	ca	ca	ca	ca	ca

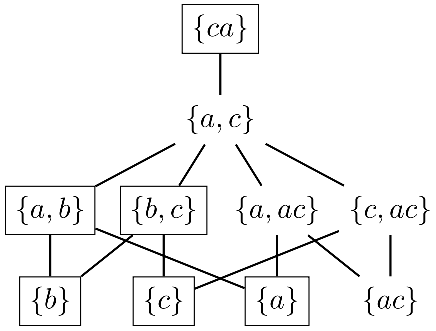


Figure 2: The join-semilattice structure that orders L. Multiplicative idempotents boxed.

the end, there are ten distinct actions arising from nonempty sets of semigroup elements, arranged into the semilattice shown in Figure 2. The join operation $x \vee y$ finds the least upper bound of x and y; *e.g.* $\{ac\} \vee \{b\} = \{a, c\}$.

Finally, one can verify that for any for sets A, B, C, and D and for any idempotent set X^ω, it holds that

$$AX^\omega D \vee AX^\omega B \vee CX^\omega D = AX^\omega B \vee CX^\omega D.$$

For instance, consider $A = \{ac\}$, $B = \{c\}$, $C = D = \{b\}$, and $X^\omega = \{b, c\}$:

$$\begin{aligned} AX^\omega B &= \{ac\}\{b,c\}\{c\} = \{acbc, accc\} = \{ac\} \\ CX^\omega D &= \{b\}\{b,c\}\{b\} = \{bbb, bcb\} = \{b\} \\ AX^\omega D &= \{ac\}\{b,c\}\{b\} = \{acbb, accb\} = \{a\} \end{aligned}$$

We verify that $\{ac\} \vee \{b\} = \{a, c\}$ and that $\{a\} \vee (\{ac\} \vee \{b\}) = \{a\} \vee \{a, c\} = \{a, c\}$. In other words, $AX^\omega D \leqslant AX^\omega B \vee CX^\omega D$ as desired. The same holds when considering any other valid instantiation of the variables, so the language is strictly local.

7 Conclusions and Prospects

We presented augmentations of the syntactic semigroup (monoid) structure to accommodate the additional information that is needed in order to capture two fundamental classes in the subregular hierarchy that are not closed under the Boolean operations. The syntactic join-semigroup (join-monoid) has enough information to capture *conjunctive* varieties of languages, which may not be closed under union or complementation. As computational linguistics research emphasizes intersection-closed classes without regard to whether they are also closed under the other Boolean operations, these techniques may prove more valuable in this space. Specifically, we provided purely algebraic characterizations for the strictly piecewise languages:

$$[\![1 \leqslant x]\!]_{(*)}$$

and the strictly local languages:

$$[\![ax^\omega d \leqslant ax^\omega b \vee cx^\omega d]\!]_{(+)}.$$

Recall that inequalities in the join-semigroup or join-monoid structure are notational shorthand for equalities, so one could also say that the strictly piecewise languages are characterized by

$$[\![1 \vee x = x]\!]_{(*)}$$

and the strictly local languages by

$$[\![ax^\omega d \vee ax^\omega b \vee cx^\omega d = ax^\omega b \vee cx^\omega d]\!]_{(+)}.$$

The latter perspective is preferred from the perspective of universal algebra, but the former offers a simpler statement of the characterizations.

Following Lambert (2023), the characterization of the strictly local languages extends to a characterization for tier-based strictly local languages of Heinz et al. (2011), by omitting $[\lambda]$ where possible in the construction of the syntactic join-semigroup, even when that class contains some *neutral letters*.

This also paves the way toward characterization and analysis of several other classes introduced in the computational linguistics literature. Using the techniques introduced herein alongside those of Lambert (in press), one might capture the multiple-tier-based strictly local languages of De Santo and Graf (2019). Join-semigroups may also help to capture the interval-based strictly piecewise languages of Graf (2017) or the melody-local languages of Jardine (2020). The decision procedures derived from the algebraic characterization could then be used to efficiently catalogue attested language patterns. For now, we leave open the question of whether these classes are conjunctive varieties, and, if so, what their characterizing inequalities are.

Known classes that are too restrictive can be extended to a more general class by removing one or more equations, perhaps after inserting redundant ones. For instance, piecewise testable languages satisfy both $(xy)^\omega x = (xy)^\omega$ *and* $y(xy)^\omega = (xy)^\omega$, while the broader class of $\mathcal{R}$-trivial languages (see Brzozowski and Fich, 1984) requires only the former. One can also extend a class, *e.g.* $\mathbf{J}^-$, by forming a product variety, *e.g.* $\mathbf{J}^- * \mathbf{D}$, which characterizes the "strictly piecewise-local" languages of Rogers and Lambert (2019b), also known as "generalized subsequence languages" (Heinz, 2010b) or "languages of dot-depth at most one-half". The resulting structures can instantiate learning algorithms using ideas from García and Ruiz (2006) and Heinz et al. (2012).

Another area of future work concerns the extension of these techniques to transducers. Lambert and Heinz (2023, 2024) explored the algebraic structure of phonological maps and some of the classes used to classify them. For each subregular $*$-variety ($+$-variety), one can say that a finite-state transducer belongs to an analogous class of functions if and only if its syntactic monoid (resp. semigroup) belongs to the corresponding variety of algebraic structures. Lambert and Heinz (2023) demonstrated that the "input strictly local" functions of Chandlee et al. (2014) have definite structure, a proper subclass of strictly local. Understanding how to derive a join-semigroup structure from a transducer will allow us to understand the class of functions that has the richer structure of the strictly local languages. It will also show whether the function class that Burness and McMullin (2020) refer to as strictly piecewise captures all and only those processes that are structurally strictly piecewise, or if there is another related class that does so.

Acknowledgments

The author thanks three anonymous reviewers for their contributions to the clarity of this document.

References

Jorge Almeida. 1989. Equations for pseudovarieties. In *Formal Properties of Finite Automata and Applications: Proceedings of the LITP Spring School on Theoretical Computer Science*, volume 386 of *Lecture Notes in Computer Science*, pages 148–164.

Jorge Almeida. 1995. *Finite Semigroups and Universal Algebra*, volume 3 of *Series in Algebra*. World Scientific, Singapore.

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. 2020. On the ability and limitations of transformers to recognize formal languages. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Proceessing (EMNLP)*, pages 7096–7116. Association for Computational Linguistics.

Stephen L. Bloom. 1976. Varieties of ordered algebras. *Journal of Computer and System Sciences*, 13(2):200–212.

Janusz Antoni Brzozowski. 1964. Derivatives of regular expressions. *Journal of the ACM*, 11(4):481–494.

Janusz Antoni Brzozowski and Faith Ellen Fich. 1984. On generalized locally testable languages. *Discrete Mathematics*, 50:153–169.

Phillip Burness and Kevin McMullin. 2020. Modelling non-local maps as strictly piecewise functions. In *Proceedings of the Society for Computation in Linguistics*, volume 3, pages 493–495, New Orleans, Louisiana.

Pascal Caron. 2000. Families of locally testable languages. *Theoretical Computer Science*, 242(1–2):361–376.

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. 2014. Learning strictly local subsequential functions. *Transactions of the Association for Computational Linguistics*, 2:491–503.

Eung-Do Cook. 1978. The synchronic and diachronic status of Sarcee γ^y. *International Journal of American Linguistics*, 44(3):192–196.

Aldo De Luca and Antonio Restivo. 1980. A characterization of strictly locally testable languages and its application to subsemigroups of a free semigroup. *Information and Control*, 44(3):300–319.

Aniello De Santo and Thomas Graf. 2019. Structure sensitive tier projection: Applications and formal properties. In Raffaella Bernardi, Greg Kobele, and Sylvain Pogodalla, editors, *Formal Grammar 2019*, volume 11668 of *Lecture Notes in Computer Science*, pages 35–50. Springer Verlag.

Matt Edlefsen, Dylan Leeman, Nathan Myers, Nathaniel Smith, Molly Visscher, and David Wellcome. 2008. Deciding strictly local (SL) languages. In *Proceedings of the 2008 Midstates Conference for Undergraduate Research in Computer Science and Mathematics*, pages 66–73.

Samuel Eilenberg. 1976. *Automata, Languages, and Machines*, volume B. Academic Press, New York, New York.

Jie Fu, Jeffrey Heinz, and Herbert G. Tanner. 2011. An algebraic characterization of strictly piecewise languages. In Mitsunori Ogihara and Jun Tarui, editors, *Theory and Applications of Models of Computation*, volume 6648 of *Lecture Notes in Computer Science*, pages 252–263. Springer Berlin / Heidelberg.

Pedro García and José Ruiz. 2006. Learning in varieties of the form $V * LI$ from positive data. *Theoretical Computer Science*, 362(1–3):100–114.

R. W. N. Goedemans, Jeffrey Heinz, and Harry van der Hulst. 2015. StressTyp2.

Thomas Graf. 2017. The power of locality domains in phonology. *Phonology*, 34(2):385–405.

Leonard H. Haines. 1969. On free monoids partially ordered by embedding. *Journal of Combinatorial Theory*, 6(1):94–98.

Jeffrey Heinz. 2007. *Inductive Learning of Phonotactic Patterns*. Ph.D. thesis, University of California, Los Angeles.

Jeffrey Heinz. 2010a. Learning long-distance phonotactics. *Linguistic Inquiry*, 41(4):623–661.

Jeffrey Heinz. 2010b. String extension learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 897–906, Uppsala, Sweden. Association for Computational Linguistics.

Jeffrey Heinz. 2018. The computational nature of phonological generalizations. In Larry Hyman and Frank Plank, editors, *Phonological Typology*, volume 23 of *Phonetics and Phonology*, chapter 5, pages 126–195. Mouton de Gruyter.

Jeffrey Heinz, Anna Kasprzik, and Timo Kötzing. 2012. Learning in the limit with lattice-structured hypothesis spaces. *Theoretical Computer Science*, 457:111–127.

Jeffrey Heinz, Chetan Rawal, and Herbert G. Tanner. 2011. Tier-based strictly local constraints for phonology. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Short Papers*, volume 2, pages 58–64, Portland, Oregon. Association for Computational Linguistics.

Jeffrey Heinz and James Rogers. 2013. Learning subregular classes of languages with factored deterministic automata. In *Proceedings of the 13th Meeting on the Mathematics of Language*, pages 64–71, Sofia, Bulgaria. Association for Computational Linguistics.

Adam Jardine. 2020. Melody learning and long-distance phonotactics in tone. *Natural Language & Linguistic Theory*, 38:1145–1195.

Jacob K. Johnson and Aniello De Santo. 2024. Online learning of ITSL grammars. In *Proceedings of the Society for Computation in Linguistics*, volume 7, pages 257–267, Irvine, California.

Ondřej Klíma and Libor Polák. 2019. Syntactic structures of regular languages. *Theoretical Computer Science*, 800:125–141.

Dakotah Lambert. 2021. Grammar interpretations and learning TSL online. In *Proceedings of the Fifteenth International Conference on Grammatical Inference*, volume 153 of *Proceedings of Machine Learning Research*, pages 81–91.

Dakotah Lambert. 2023. Relativized adjacency. *Journal of Logic, Language and Information*, 32(4):707–731.

Dakotah Lambert. 2024. System description: A theorem-prover for subregular systems: The Language Toolkit and its interpreter, plebby. In *Functional and Logic Programming: 17th Annual Symposium, FLOPS 2024*, volume 14659 of *Lecture Notes in Computer Science*, pages 311–328, Kumamoto, Japan. Springer, Singapore.

Dakotah Lambert. in press. Multitier phonotactics with logic and algebra. *Phonology*.

Dakotah Lambert and Jeffrey Heinz. 2023. An algebraic characterization of total input strictly local functions. In *Proceedings of the Society for Computation in Linguistics*, volume 6, pages 25–34, Amherst, Massachusetts.

Dakotah Lambert and Jeffrey Heinz. 2024. Algebraic reanalysis of phonological processes described as output-oriented. In *Proceedings of the Society for Computation in Linguistics*, volume 7, pages 129–138, Irvine, California.

Dakotah Lambert, Jonathan Rawski, and Jeffrey Heinz. 2021. Typology emerges from simplicity in representations and learning. *Journal of Language Modelling*, 9(1):151–194.

Fang-Kuei Li. 1930. A study of Sarcee verb-stems. *International Journal of American Linguistics*, 6(1):3–27.

Connor Mayer and Travis Major. 2018. A challenge for tier-based strict locality from Uyghur backness harmony. In Annie Foret, Greg Kobele, and Sylvain Pogodalla, editors, *Formal Grammar 2018*, volume 10950 of *Lecture Notes in Computer Science*, pages 62–83. Springer.

Robert McNaughton. 1974. Algebraic decision procedures for local testability. *Mathematical Systems Theory*, 8(1):60–76.

Robert McNaughton and Seymour Aubrey Papert. 1971. *Counter-Free Automata*. MIT Press.

Jean-Éric Pin. 1995. A variety theorem without complementation. *Russian Mathematics (Izvestija vuzov. Matematika)*, 39:80–90.

Jean-Éric Pin. 1997. Syntactic semigroups. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages: Volume 1 Word, Language, Grammar*, pages 679–746. Springer-Verlag, Berlin.

Jean-Éric Pin. 2017. The dot-depth hierarchy, 45 years later. In Stavros Konstantinidis, Nelma Moreira, Rogério Reis, and Jeffrey Shallit, editors, *The Role of Theory in Computer Science*, pages 177–201. World Scientific.

Jean-Éric Pin and Pascal Weil. 1996. A Reiterman theorem for pseudovarieties of finite first-order structures. *Algebra Universalis*, 35:577–595.

Libor Polák. 2001. Syntactic semiring of a language. In *Mathematical Foundations of Computer Science 2001*, volume 2136 of *Lecture Notes in Computer Science*, pages 611–620.

Michael Oser Rabin and Dana Scott. 1959. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125.

Jan Reiterman. 1982. The Birkhoff theorem for finite algebras. *Algebra Universalis*, 14:1–10.

James Rogers, Jeffrey Heinz, Gil Bailey, Matt Edlefsen, Molly Visscher, David Wellcome, and Sean Wibel. 2010. On languages piecewise testable in the strict sense. In Christian Ebert, Gerhard Jäger, and Jens Michaelis, editors, *The Mathematics of Language: Revised Selected Papers from the 10th and 11th Biennial Conference on the Mathematics of Language*, volume 6149 of *LNCS/LNAI*, pages 255–265. FoLLI/Springer.

James Rogers, Jeffrey Heinz, Margaret Fero, Jeremy Hurst, Dakotah Lambert, and Sean Wibel. 2013. Cognitive and sub-regular complexity. In Glyn Morrill and Mark-Jan Nederhof, editors, *Formal Grammar: 17th and 18th International Conferences FG 2012 Opole, Poland, August 2012, Revised Selected Papers and FG 2013 Düsseldorf, Germany, August 2013, Proceedings*, volume 8036 of *Lecture Notes in Computer Science*, pages 90–108. Springer-Verlag.

James Rogers and Dakotah Lambert. 2019a. Extracting Subregular constraints from Regular stringsets. *Journal of Language Modelling*, 7(2):143–176.

James Rogers and Dakotah Lambert. 2019b. Some classes of sets of structures definable without quantifiers. In *Proceedings of the 16th Meeting on the Mathematics of Language*, pages 63–77, Toronto, Canada. Association for Computational Linguistics.

James Rogers and Geoffrey K. Pullum. 2011. Aural pattern recognition experiments and the subregular hierarchy. *Journal of Logic, Language and Information*, 20(3):329–342.

Howard Straubing. 1985. Finite semigroup varieties of the form $V * D$. *Journal of Pure and Applied Algebra*, 36:53–94.

Howard Straubing and Pascal Weil. 2021. Varieties. In Jean-Éric Pin, editor, *Handbook of Automata Theory*, volume 1, chapter 16, pages 569–614. EMS Press, Berlin, Germany.

Bret Tilson. 1987. Categories as algebra: An essential ingredient in the theory of monoids. *Journal of Pure and Applied Algebra*, 48(1–2):83–198.

Charles Torres and Richard Futrell. 2023. L_0-regularization induces subregular biases in LSTMs. In *Proceedings of the Society for Computation in Linguistics*, volume 6, pages 394–396, Amherst, Massachusetts.

Sam van der Poel, Dakotah Lambert, Kalina Kostyszyn, Tiantian Gao, Rahul Verma, Derek Andersen, Joanne Chau, Emily Peterson, Cody St. Clair, Paul Fodor, Chihiro Shibata, and Jeffrey Heinz. 2024. MLRegTest: A benchmark for the machine learning of regular languages. *Journal of Machine Learning Research*, 25(283):1–45.

A Lie-Algebraic Perspective on Tree-Adjoining Grammars

Isabella Senturia
Yale University

Elizabeth Xiao
California Institute of Technology

Matilde Marcolli
California Institute of Technology

Abstract

We provide a novel mathematical implementation of tree-adjoining grammars using two combinatorial definitions of graphs. With this lens, we demonstrate that the adjoining operation defines a pre-Lie operation and subsequently forms a Lie algebra. We demonstrate the utility of this perspective by showing how one of our mathematical formulations of TAG captures properties of the TAG system without needing to posit them as additional components of the system, such as null-adjoining constraints and feature TAG.

1 Introduction

Formal languages have long been one of the most prominent tools used by mathematical linguistics in attempts to develop formal systems generating natural language (Harrison, 1978; Sipser, 1996; Hopcroft et al., 2001). Different grammars enable descriptions of different phenomena in natural language to varying degrees. While the goal is often conceptualized as generating a given language as string of words, the inherent hierarchical nature of language leads to another aim: to generate a language via the appropriate tree structures modeling those hierarchical relationships within natural language. Tree-Adjoining Grammars (TAGs) (Joshi et al., 1975; Joshi, 1987; Joshi and Schabes, 1997; Kroch and Joshi, 1985) are one approach to such a goal. Structure building via Merge in Minimalism is another approach.

At the same time, mathematicians have worked to reframe various linguistic structures and formulations as mathematical systems. With respect to formal languages, and in particular tree languages, Giraudo (2019) demonstrates that the algebraic structures known as colored operads can successfully define various string languages (such as context-free languages) and tree languages (including those generated by tree grammars, regular tree grammars, and synchronous grammars). Most recently, Marcolli et al. (2025a) have algebraically formalized Merge in the framework of the Strong Minimalist Thesis, modeling syntactic derivations in Hopf-algebraic terms. Also relevant to the Lie algebra formalism considered here, Marcolli and Port (2015) use different combinatorial definitions of graphs to show specific context-free and context-sensitive graph grammars have associated Lie algebras.

With respect to linguistic structure, operations to build larger trees from smaller trees are fundamental in capturing the compositional nature of syntax. In Minimalism, two trees are combined together by Merge by creating a new node and two edges, each of which joins to the two respective roots of the other trees. A complimentary perspective to this external composition, where the combining operation does not affect the internal structure of either of the two independent components, would be internal composition, where one tree is inserted *into* another. This operation is exactly the way that TAG operates, and pre-Lie operations' ability to capture this underpins the relevance of Lie algebras to syntactic structure-building. Moreover, there are two complimentary types of insertion of one tree into another, i.e. at a node (as in TAGs) or at an edge.[1] This paper explores the pre-Lie operation derived from node-insertion.

One mathematical motivation for studying Lie algebras is their close and richly-studied relationship with Hopf algebras. Every Lie algebra has a Hopf algebra associated to it through its universal enveloping algebra; conversely, the Milnor-Moore theorem provides conditions for a Hopf algebra to occur as the universal enveloping algebra of some Lie algebra. Pre-Lie algebras are also closely tied

[1] Edge-insertion provides another example of a pre-Lie operation which can be compared and contrasted to the TAG pre-Lie operation. Due to issues of space, we are unable to expand upon this in this work.

to rooted trees in that the free pre-Lie algebra generated by a single element is isomorphic to the vector space of nonplanar rooted trees equipped with an insertion pre-Lie operator, which also has an inherent operadic description (Chapoton and Livernet, 2001). In fact, the Connes-Kreimer Hopf algebra of rooted trees is dual to the universal enveloping algebra of the Lie algebra from a free pre-Lie operator (Connes and Kreimer, 1998).

In this work, we explore the formal properties of tree-adjoining through the mathematical lens of Lie algebras. Although we initially define adjunction on trees in a more general setting, we can constrain our analysis to a particular TAG by restricting the set of generators to the initial and auxiliary trees which occur in that TAG. By recasting TAGs as an algebraic system, we demonstrate that the most appropriate mathematical representation of TAG that yields a Lie algebra requires TAG to be defined using the combinatorial definition of graphs in terms of corollas and half-edges (typically used in theoretical physics but not in computer science or formal language theory, so we refer to it as the "physics definition"). By allowing a single edge to be described as comprising of two half-edges, this provides a more elegant description of the elementary and auxiliary trees and their compositions. With this physics definition of trees, components of the TAG system such as null-adjoining constraints and specification of insertion positions naturally follow and do not need to be postulated ad-hoc (as they do with standard formulations of TAG). Not only this, but elaborations of TAG such as feature-TAG are easily implementable.

From the algebraic side, we show that various alternatives to the physics definition (undirected binary tree graphs and an elaboration on these, which we coin *double vertex* trees) cause problems with the Lie algebra structure (in particular, the pre-Lie operation) and also converge to the physics definition as the most natural algebraic description. We also show that colored operads can model TAG insertion via the composition of two insertion operations at the leaf.

A formulation of TAGs in terms of Lie algebra and operad insertions was suggested in Section 2.5.1 of Marcolli et al. (2025a), for the purpose of comparison with Merge, and is implemented here. While in formal languages generative models are usually compared in terms of their weak and strong generative capacity, a more refined comparison is possible, in the sense of being able to more explicitly define and compare the underlying algebraic structures.

2 Background

In this section, we introduce the linguistic and mathematical concepts and tools necessary for the paper. We begin with TAGs, before defining graphs and Lie algebras.

2.1 Tree-Adjoining Grammars

We follow the presentation of tree-adjoining grammars as given in Joshi and Schabes (1997). We define the grammar below, before defining the two types of operations that allow building of larger trees out of insertion/composition of smaller trees.

Definition 2.1. *A tree-adjoining grammar (TAG) is a 5-tuple (Σ, N, I, A, S), where Σ and N are finite sets of terminal and nonterminal symbols, respectively ($\Sigma \cap N = \emptyset$); $S \in N$ is a distinguished nonterminal symbol known as the* start symbol*; I and A are finite sets of (finite) trees called* initial *and* auxiliary *trees, respectively, whose interior nodes are labeled by nonterminal symbols, leaves of initial trees are labeled by either terminal or nonterminal symbols and those labeled by nonterminal symbols are marked for substitution. The auxiliary trees' leaves are all marked for substitution except one, denoted by an asterisk and required to be labeled with the same label as the root.*

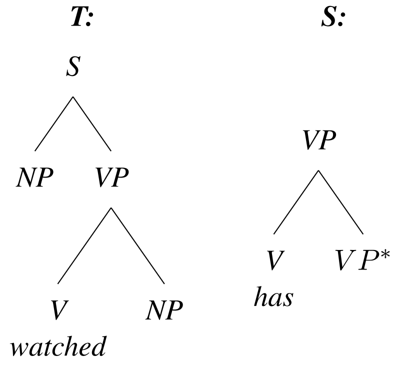
Lexicalized TAG requires at least one terminal symbol to appear at a leaf of every initial or auxiliary tree. *Elementary trees* are those contained in $I \cup A$, and *derived trees* are those built by composition of two or more trees. The two composition operations are defined as follows:

Definition 2.2. *The* adjoining *operation builds a new tree from an auxiliary tree S and an initial, auxiliary or derived tree T. This is done by inserting S into T at a node α when the root of S and a leaf of S are both labeled with α. In other words, the node labeled α in T is replaced by the tree S. We denote this operation, somewhat nontraditionally, as*

$$T \lhd_\alpha S,$$

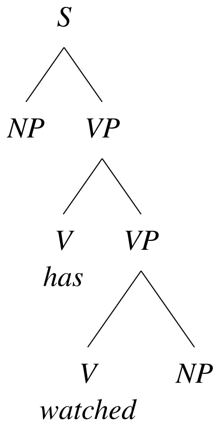
in order to introduce the insertion operation which is relevant for the Lie algebra.

Example 2.3. *Let S and T be the following trees:*[2]



Then inserting S into T at the VP node of T yields

$T \lhd_{VP} S =$



The second operation available in TAGs, *substitution*, was not included in the original definition of TAGs and in fact does not provide any additional expressive power. As such, our Lie-algebraic formalization of TAGs explicitly implementing only the adjoining operation (as the pre-Lie operation) is sufficient to express the entirety of the TAG formalism.

Typically, the arity of TAG trees is unary-binary—that is, nodes can have 0, 1 or 2 children. One can restrict to (full) binary trees (all non-leaves have two children and leaves have zero children) to simplify the set of objects or for comparison to other analyses (Lie-algebras of binary trees and linguistic analyses which consider Merge to be binary, such as Marcolli et al., 2025a).

TAG trees have labeled insertion spots at interior nodes. In our formulation, one can either have uniform insertions at every interior node, or labeled insertion: both cases are realized as a sum over all possible insertions. Similarly, we allow re-attaching of the remainder (part beneath the insertion spot) of the tree being inserted into to be reattached at any leaf, and denote this as a sum over all leaves. We discuss variations of this labeling later in the paper when we demonstrate the ability of the physics graph formulation of TAG to seamlessly incorporate null-adjoining constraints and feature-TAG.

2.2 Math

We now present the mathematical concepts utilized in this paper: graphs, Lie algebras, and operads.

2.2.1 Graphs

The usual combinatorial definition of graphs is in terms of vertices and edges, namely we define a graph $G = (V(G), E(G))$, where $V(G) = \{v_1, v_2, \ldots, v_n\}$ is a set of n *vertices*, $E(G) = \{e_1 = (v_a, v_b), \ldots, e_m = (v_p, v_q)\}$ is a set of m *edges*. If the edges are undirected, the edge pair (v_i, v_j) is unordered, whereas if the edge is directed, the edge pair is ordered $(start, end)$.

The *degree* d_v of a vertex v is the number of edges connected to that node. A *leaf* is a node of degree 1. Two *adjacent* vertices are connected by an edge. A *path* from some vertex v_i to another v_j is the sequence of edges connecting adjacent nodes between v_i and v_j. A graph is *connected* if there is a path from every node to every other node.

The class of *trees* is the class of connected *acyclic graphs* $T = (V, E)$ defined by the existence of exactly one path connecting any two distinct vertices $v_1, v_2 \in V$—that is, they have no loops. A *directed tree* is a tree with directed edges. A *rooted* tree is a tree for which a specific node has been designated as the root, and is graphed with this root at the top or bottom. We refer to the set of nodes adjacent to, and below, another node v as the children of v. Any rooted tree can be viewed as a directed tree, where all edges are either uniformly directed towards the root, or away from it. A tree is *planar* (or, planarly embedded) if for all v, the children of v have a linear order.

2.2.2 Lie algebras

We follow the definition of Lie and pre-Lie algebras given in Procesi (2007) and Cartier and Patras (2021). All vector spaces are assumed to be over the real numbers $\mathbb{R}$, although our exposition will work for any field of characteristic 0.

A *Lie algebra* is a vector space V equipped with a bilinear product $[a, b]$ satisfying the Lie axioms of *antisymmetry*,

$$[a, b] = -[b, a],$$

[2] Adapted from Joshi and Schabes (1997).

and the *Jacobi identity*,

$$[a,[b,c]]+[b,[c,a]]+[c,[a,b]]=0.$$

This product is called the *Lie bracket*. Due to antisymmetry, the Lie bracket is not commutative unless it is uniformly zero.

A vector space V is *graded* if it decomposes into a direct sum indexed by the non-negative integers: $V=\bigoplus_{n=0}^{\infty} V_n$, where each V_n is a subspace of V and $V_i \cap V_j = \{0\}$ whenever $i \neq j$. We say a graded vector space V is *connected* if V_0 is one-dimensional.[3] If $a \in V_n$, then a is called a *homogeneous* (or *pure*) *element of degree* n. If each V_n is finite-dimensional, then V is *locally finite*.

A Lie algebra L is graded if the Lie bracket respects the grading: if $a \in L_m$ and $b \in L_n$, then $[a,b] \in L_{n+m}$.

Example 2.4. *Any associative algebra A has a Lie bracket defined by $[a,b] := ab - ba$, which is called the* commutator *of a and b. For instance, let $A = M_2(\mathbb{R})$, the space of 2×2 matrices. Then*

$$[\left(\begin{smallmatrix}1&0\\0&0\end{smallmatrix}\right),\left(\begin{smallmatrix}2&1\\1&0\end{smallmatrix}\right)]=\left(\begin{smallmatrix}2&1\\0&0\end{smallmatrix}\right)-\left(\begin{smallmatrix}2&0\\1&0\end{smallmatrix}\right)=\left(\begin{smallmatrix}0&1\\-1&0\end{smallmatrix}\right).$$

We can hence say the following about a commutative algebra:

Example 2.5. *If A is a commutative algebra, then $[a,b]=0$ for all $a,b \in A$ and the commutator Lie bracket is trivial.*

More generally, the Lie bracket is the commutator of a binary operation (such as a product or a pre-Lie operation), meaning it can be considered to be a measure of the degree to which the operation is not symmetric.

Let L, M be two Lie algebras. A linear map $\phi : L \to M$ is called a *Lie algebra homomorphism* if it commutes with the Lie brackets; that is, for $a, b \in L$,

$$\phi([a,b]_L) = [\phi(a),\phi(b)]_M.$$

2.2.3 Pre-Lie algebras

Let V be a vector space. A bilinear product $\lhd$: $V \times V \to V$ is called a *(right) pre-Lie operator* if it satisfies the *(right) Vinberg identity*

$$\begin{aligned}(a \lhd b) \lhd c - a \lhd (b \lhd c)\\ = (a \lhd c) \lhd b - a \lhd (c \lhd b),\end{aligned}$$

making $(V, \lhd)$ a *(right) pre-Lie algebra*. The expression $(a \lhd b) \lhd c - a \lhd (b \lhd c)$ is referred to as the *associator* $A(a,b,c)$ with respect to $\lhd$; if $\lhd$ is associative, then $A(a,b,c)=0$ uniformly. The Vinberg identity asserts that the associator is unchanged when the second and third arguments are swapped; that is, for all $a,b,c \in V$,

$$A(a,b,c) = A(a,c,b).$$

A pre-Lie operator induces a Lie bracket defined by $[a,b] := a \lhd b - b \lhd a$, which automatically satisfies antisymmetry and the Jacobi identity, making A a Lie algebra. We do note that not all Lie algebras arise from a pre-Lie operator.

2.2.4 Tree-insertion as a free pre-Lie operator

Let $V = \operatorname{span}(B)$ be a vector space, say of countable dimension, with basis given by $B = \{a,b,c,\dots\}$. We use $\operatorname{span}(B)$ and $\langle B\rangle$ interchangeably to refer to the span. The *free associative algebra over V* has, as its basis, words over the alphabet B, such as $aabc$, $acccbba$ or the empty word $\varnothing$; the product is defined over basis elements by concatenation of words with no other relations. This construction is often denoted by $T(V)$ and called the *tensor algebra* over V; if V is d-dimensional, we often call $T(V)$ the *free associative algebra on d generators* (or letters), which is unique up to isomorphism. Any tensor algebra has a natural grading over $\mathbb{N}$, where the n-degree component of $T(V)$ is generated by words of length n. There is an injective copy of V embedded in $T(V)$, and any extension of V to an associative algebra is a quotient of $T(V)$.

We can similarly define $L(V)$, the *free pre-Lie algebra over V*, with pre-Lie operator $\lhd$ satisfying the Vinberg identity on the associators and no other relations. If $V = \operatorname{span}(B)$ is d-dimensional, then we will refer to $L(V)$ as a *free pre-Lie algebra on d generators* (or letters), the elements of B. However, words over B are no longer sufficient to form a basis for $L(V)$, since $\lhd$ is not associative; instead, we require strings to be parenthesized such that each pair of parentheses corresponds to one application of the $\lhd$ operator. One interpretation would be as planar full binary trees with leaves labeled by elements of B, since such trees with n leaves are in correspondence with complete parenthesizations of a string of length n. This basis works for any non-associative algebra over V with no further relations, and it is reminiscent of the Merge operation since it combines left and right arguments into a single

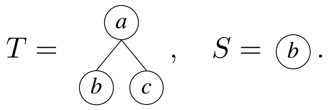
[3] This terminology originates from topology, where the number of generators of the zeroth homology group of a topological space counts its connected components.

tree by grafting them together as the left and right child, respectively, of a new root node. However, this basis does not provide us with a convenient geometric or combinatorial interpretation of the associator, since they are defined over a difference of basis elements.

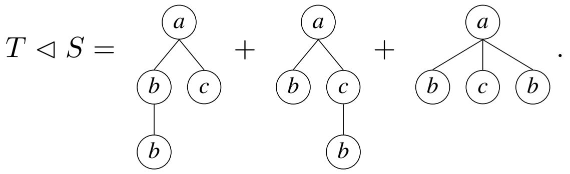
Another interpretation of the free pre-Lie algebra over V is due to Chapoton and Livernet (2001), where basis elements are nonplanar rooted trees (not necessarily binary), with vertices labelled by basis elements of V. The pre-Lie operator can now be interpreted geometrically as insertion: if T and S are two trees, then $T \lhd S$ is the sum of all possible trees that result from grafting S to T by joining a node of T to the root of S with a new edge.

Since a new edge is produced in every nontrivial insertion, trees are graded by numbers of vertices and $\lhd$ respects the natural grading. The degree zero component of $L(V)$ is generated by the empty tree $\varnothing$; for any tree T, we have $T \lhd \varnothing = T$ and $\varnothing \lhd T = \varnothing$, which extends to the entire vector space by linearity. In the following example, note that the trees are non-planar and so we are currently ignoring linear ordering of the leaves.

Example 2.6. *Let*



Then



This model provides a geometric interpretation for the associator. If T_1, T_2, T_3 are three trees, then $(T_1 \lhd T_2) \lhd T_3$ is a sum of all trees obtained by first inserting T_2 into T_1, and then inserting T_3 into the resulting tree. The associator consists exactly of the trees produced by inserting T_2 and T_3 into distinct vertices of T_1.

Fix a label $\alpha \in B$ from the basis. We can define a restricted insertion operator $\lhd_\alpha$ where grafting only occurs at nodes in the first argument labelled α; this is still a pre-Lie operator. The general pre-Lie operator allowing insertion at every node is the sum of the restricted operator over all basis elements: $\lhd = \sum_{\alpha \in B} \lhd_\alpha$.

The insertion operators here will serve as a framework for our definition of TAG as a pre-Lie operator. Tree-adjoining will be viewed as a kind of "insertion" where, instead of grafting one tree to a node of the other, the inserted tree becomes embedded inside another one.

2.2.5 Tree-adjoining as a pre-Lie operator

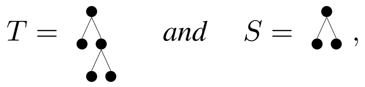
Let X be a collection of trees. For now, we will assume the trees in X are binary, planar and unlabelled. Let $\mathcal{T} = \mathcal{T}_{bin,pl}$ be the free vector space with X as its basis. We define $\lhd$ on basis elements $T, S \in X$ by adjoining S at every possible position in T over every leaf of S; then, $T \lhd S$ is the sum of all trees produced by this process.

Note that we permit adjunction of S into T at leaves of T. This procedure superficially resembles the process of substitution, where a leaf is replaced by a larger tree. However, adjunction at leaves is different because it implicitly pairs the root of the inserted tree with one of its leaves, whereas substitution only considers the root of the inserted tree. Hence the coefficient of a tree obtained in such a way in $T \lhd S$ will be at least the number of leaves in S.

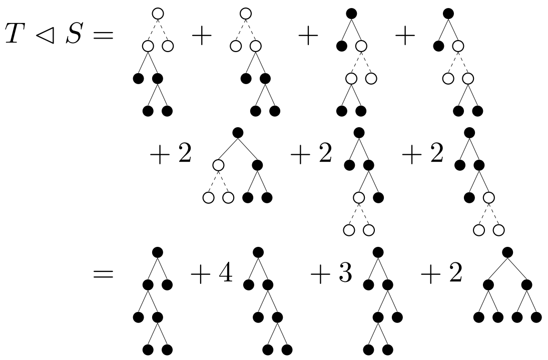
In general, a tree having a coefficient larger than 1 in $T \lhd S$ means it is obtainable from adjunction in multiple different ways.

If T has n vertices and S has ℓ leaves, then the sum of the coefficients of $T \lhd S$ will be $n\ell$. The operation $\lhd$ is then extended to the whole vector space $\mathcal{T}$ by linearity in both arguments.

Example 2.7. *Let*



which respectively have 5 nodes and 2 leaves. We show the result of adjoining S to T. The copy of S in each term is drawn with white nodes and dashed edges purely for illustrative purposes; the actual trees are unlabelled.



In the first expression for $T \triangleleft S$, the last three terms each have a coefficient of 2 because adjunction occurs at a leaf of T. There is no subtree to be attached to any leaf of S; however, a leaf of S still needs to be chosen to ensure consistency in the associator, so the two copies correspond to the choices of leaf in S.

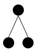
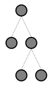
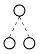
We reiterate that each tree has a fixed planar embedding; indeed, if the trees were nonplanar, then the first three terms in the last line would represent the same tree.

The insertion operation above has the following property:

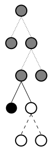
Theorem 2.8. *$\triangleleft$ is a pre-Lie operator.*

See proof on page 13.

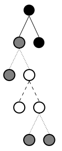
Example 2.9. *Let*

$T_1 =$  , $T_2 =$  , $T_3 =$  .

Then



appears as a term of $(T_1 \triangleleft T_2) \triangleleft T_3$ only, since adjunctions occur at different vertices of T_1, while



appears in both $(T_1 \triangleleft T_2) \triangleleft T_3$ and $T_1 \triangleleft (T_2 \triangleleft T_3)$, hence will be cancelled out in the associator.

An immediate question is whether or not tree-adjoining, in any formulation (planar or non-planar, labelled or unlabelled), is isomorphic as a pre-Lie algebra to a free one. This question is relevant to an algebraic comparison between TAGs and the insertion Lie algebra dual to the Hopf algebra of workspaces in Minimalism. Before we can answer this question, we will need to fix some more conditions on the basis elements of $\mathcal{T}$ and how exactly the tree-adjunction operation is defined. Several different possibilities will be considered in §3 on the mathematical formulation of TAG, but we will use the binary planar unlabelled case as our first example.

We note the behaviour of the single-node tree in insertions:

Example 2.10. *Let T be a binary planar unlabelled tree, and let $\bullet$ be the tree consisting of a single node. Then*

$$T \triangleleft \bullet = |T|T$$
$$\bullet \triangleleft T = \ell(T)T$$

where $|T|$ is the number of nodes in T, and $\ell(T)$ is the number of leaves. Hence the Lie bracket of T and $\bullet$ is

$$[T, \bullet] = (|T| - \ell(T))T.$$

Theorem 2.11. *$\mathcal{T}_{bin,pl}$ is not isomorphic as a pre-Lie algebra to L_{free}, the free pre-Lie algebra on one generator.*

See proof on page 14.

To prevent overgeneration, we can introduce labels to $\mathcal{T}_{bin,pl}$ and modify the adjunction operation so that adjunction only occurs when an insertion spot of T has the same label as the root of S, and moreover that S has at least one leaf with the same label, to which the subtree of T will be attached. If S has a unique such leaf, then it represents the distinguished foot node of an auxiliary tree in TAG. See 3.1 for an example.

2.2.6 Operads

As mentioned in the introduction, operads have recently been connected to mathematical linguistics (Giraudo, 2019; Marcolli et al., 2025a). Colored operads have most recently been used in formal language theory to obtain a new proof of the well-known Chomsky-Schützenberger representation theorem (Melliès and Zeilberger, 2025). Colored operads are also used in Marcolli and Larson (2025) and Marcolli et al. (2025b) to model theta roles and phases in Minimalism, as outlined in Marcolli et al. (2025a). Indeed, in this work we also utilize colored operads as operations on the vector space containing TAG trees that can mimic the TAG pre-Lie insertion operation as a combination of two colored-operad compositions. We define the notion of a colored operad below, before returning to it in section 3.3.2 to demonstrate how it appropriately captures the Lie-algebraic TAG system.

We provide here a preliminary introduction to the theory of colored operads, and we refer the reader to Giraudo (2019) for a more detailed formulation in a formal languages context.

An *operad* organizes the compositional structure of operations that take multiple inputs and produce a single output. It is a collection $\mathfrak{D}$ of sets $\mathfrak{D}(n)$,

$\mathfrak{D} = \{\mathfrak{D}(n)\}$ that consist of operations $T \in \mathfrak{D}(n)$ with n inputs and one output. The operad has an algebraic structure defined by composition operations

$$\gamma : \mathfrak{D}(n) \times \mathfrak{D}(k_1) \times ... \times \mathfrak{D}(k_n) \to \mathfrak{D}(k_1 + ... + k_n)$$

These composition operations take the single output out of an operation in $\mathfrak{D}(k_i)$ and plug it into the i-th input of an operation in $\mathfrak{D}(n)$. This results in a new operation in $\mathfrak{D}(k_1 + ... + k_n)$ whose set of inputs is the union of all the inputs of the operations $\mathfrak{D}(k_i)$, and a single output, which is the output of the operation in $\mathfrak{D}(n)$ they are composed with. The compositions γ are associative. Operads can be *unital*, *symmetric*, etc., but we omit these definitions here as they are not relevant for the current work.

An *algebra A over an operad $\mathfrak{D}$* is a set that can serve as inputs and outputs for the operations in $\mathfrak{D}$, namely for which there are maps

$$\gamma_A : \mathfrak{D}(n) \times A^n \to A$$

compatible with the γ compositions of $\mathfrak{D}$.

A *colored operad $\mathcal{O} = \{\mathcal{O}(c, c_1, ..., c_n)\}$* is an operad where inputs and outputs are labelled (by a set of colors $c, c_i \in \Omega$) and composition can only occur when the output color c of one operation matches the color c_i of the i-th input it feeds to.

In fact, the concept of a pre-Lie operator is inherently operadic. Chapoton and Livernet (2001) explain that pre-Lie algebras naturally arise as algebras over a binary quadratic operad.

In the next section, we mathematically define TAGs in three different ways, and show that only the third mathematical formulation of TAG (using the physics definition of graphs) suffices when we would like to require the TAG to be a Lie algebra.

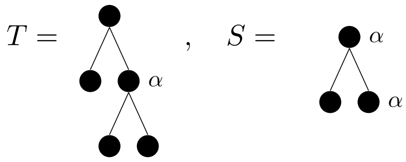
3 Mathematical Formulation of TAG

To begin formulating TAG as a mathematical system, we use the combinatorial graph definition of trees as given in section 2.2.1.

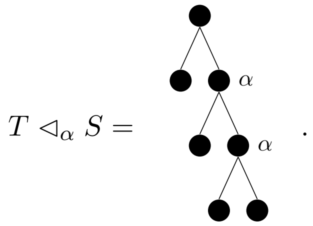
3.1 Normal single vertex method

In what follows, we will demonstrate insertion at a specific (labeled) location in the tree, instead of giving the sum over all possible insertions. We label the spot we insert at in T, as well as the two locations (root and specified leaf) that the adjoining occurs at in S, with α. Note also that usually we would be summing over all leaves for the lower part of T to re-attach to, but again for simplicity we omit this.

Example 3.1. *We formulate the trees from the initial TAG example (2.3) as graphs. Let*



Then



The problem with this comes when we think about the grading of the vector space. We are motivated to introduce a grading to our vector space because trees are fundamentally combinatorial objects with a notion of size. Furthermore, provided that the set of nodes labels is finite, then the vector space will be locally finite. Ideally we would also want the degree-0 part of the vector space to be generated by a single element, such as the empty tree, so that the vector space is connected, much as in the case of the free pre-Lie algebra. Indeed, if V is connected as a graded vector space, then the universal enveloping algebra of $L(V)$ is a graded connected Hopf algebra, which carry many desirable properties. The pre-Lie operator and Lie bracket with respect to the degree-0 generator should be simple enough that their outputs can be described in closed-form, potentially allowing us to determine whether or not the (pre-)Lie algebra is isomorphic as (pre-)Lie algebras to another one.

If we grade over vertices, we see that the insertion operation does not preserve gradation: inserting an n-noded tree S into an m-noded tree T yields an $n + m - 1$-noded tree, because S is replacing a vertex in T and hence one vertex overall is lost.

A possible repair to this, to preserve the gradation of the vector space, would be to grade over number of edges rather than number of vertices. Then inserting an n-edged tree S into an m-edged tree T yields an $n + m$-edged tree, which fixes that problem, but there is now a new issue: there are two unique trees with zero edges. These are the empty tree, and the tree comprising of a single node.

The situation is further complicated when trees are labeled, since there is now a distinct single-noded tree for every label. As stated in 2.10, the insertion of any of these single-noded trees will return T multiplied by a factor of $|T|_\alpha$, the number of times α occurs in T, meaning it is not exactly the identity insertion operation and hence even more problematic.

Mathematically, we can solve this ambiguity by equating all single-noded trees with the generator of the degree-0 component, which is achieved by quotienting out the ideal generated by elements $1 - \bullet_\alpha$ for all labels α, where $\bullet_\alpha$ is the single-noded tree where the only node is labelled α. However, this trivializes the Lie bracket with respect to $\bullet_\alpha$, and insertions of the form $T \lhd \bullet_\alpha$ are no longer eligible to be used as a way of counting occurrences of α in T. Thus, we are motivated to reformalize the TAG tree graphs in a different way. This is done with the double vertex method in the following section.

3.2 Double vertex trees

Rambow et al. (2001) suggest that locations within a tree where adjunction was possible, which are normally represented as a node in the tree, are actually two nodes, an upper copy and a lower copy, connected by a dashed line (see figure 1).

Inspired by this and in order to fix the grading issue raised by defining trees as regular graphs, we can introduce a modification on the previous trees which we call *double-vertex trees*. These trees are also graphical trees, with the caveat that any node α that is able to be inserted into is actually a double node, i.e. is counted as two nodes. The maximum count of any vertex is two, meaning we cannot increase the count of a vertex to any $n \in \mathbb{N}$. In order to prevent this unbounded increase, and because we are working with unlabeled trees, i.e. where adjoining can occur freely at any location, this means that in any non-auxiliary tree every node will be a double vertex. In every auxiliary tree, there will be exactly two single vertices: the root, and one leaf. These are the two nodes that will each get one additional count from the double vertex α they are inserting into, meaning that the root and that distinguished leaf will become double-vertex interior vertices in the larger post-adjunction tree.

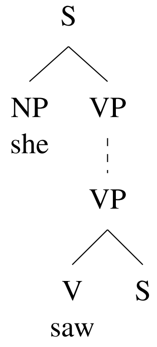
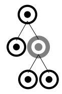
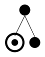


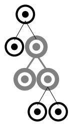
Figure 1: A tree adapted from Rambow et al. (2001) which can be inserted into at the VP node(s).

We can see this in the following example, which recasts example (3.1) in terms of this new counting system.

Example 3.2. *Let S, T be as above, i.e.*

$T =$  , $S =$ 

Note that every node in T is doubled and otherwise unlabelled. Inserting S into T at the marked node (which is gray, with a thicker border) yields

$T \lhd S =$  $+ \cdots$

However, this model also has its drawbacks. In particular, we are now able to grade by vertices, because we see that the insertion operation does preserve gradation: inserting an n-noded tree S (with $(n-2)/2$ double-nodes and 2 single nodes) into an m-noded tree T yields an $n+m$-noded tree (as the double node in T where they are both inserted contributes one node to each of the two single nodes, where adjunction takes place, turning those two single nodes into two double nodes). That being said, there is still a strange effect that this has on the overall gradation of the vector space. All odd-degree components are ignored when restricting to the trees we use in TAG: since all binary trees have an odd number of nodes, odd-degree indicates that there are an odd number of single nodes.

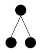
The following is the set of all gradations of the space up to five nodes.

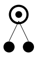
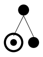
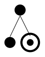
Example 3.3.

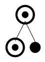
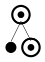
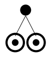
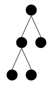
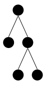
$\mathcal{T}_0 = \langle \varnothing \rangle$

$\mathcal{T}_1 = \langle$ • $\rangle$

$\mathcal{T}_2 = \langle$ ⊙ $\rangle$

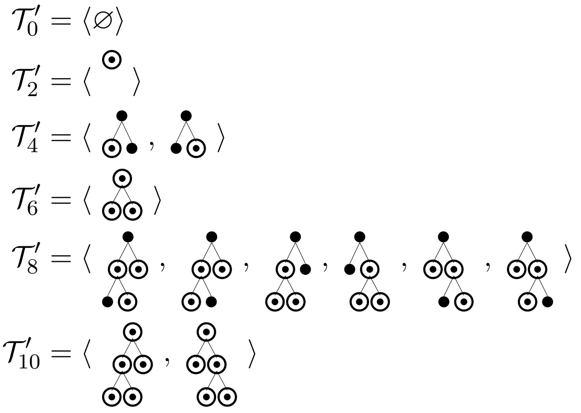
$\mathcal{T}_3 = \langle$  $\rangle$

$\mathcal{T}_4 = \langle$  ,  ,  $\rangle$

$\mathcal{T}_5 = \langle$  ,  ,  ,  ,  $\rangle$

We notice that gradation by nodes results in two trees being able to have the same dimension (number of nodes) but they can have different numbers of edges, e.g. in $\mathcal{T}_5$ there are trees with either two or four edges. This vector space contains many trees that will never be used in tree-adjunction, so we will consider the subspace $\mathcal{T}' \subseteq \mathcal{T}$ restricted to the trees that appear in TAG, which is closed under $\triangleleft$. Under the grading inherited from $\mathcal{T}$, the even-degree components of $\mathcal{T}'$ have the following generators; the odd-degree, as mentioned above, are trivial.

Example 3.4.


$\mathcal{T}'_0 = \langle\varnothing\rangle$
$\mathcal{T}'_2 = \langle \; \rangle$
$\mathcal{T}'_4 = \langle \; , \; \rangle$
$\mathcal{T}'_6 = \langle \; \rangle$
$\mathcal{T}'_8 = \langle \; , \; , \; , \; , \; , \; \rangle$
$\mathcal{T}'_{10} = \langle \; , \; \rangle$

There is now an alternation in the degrees; the ones which are 0 mod 4 contain the auxiliary trees, whereas the ones which are 2 mod 4 contain the non-auxiliary trees where all nodes are doubled. The difference between the two fundamental types of trees is encoded in the grading. Note that $\mathcal{T}'$ is still not a free pre-Lie algebra since certain insertions of basis elements produce zero: namely, any attempt to adjoin a non-auxiliary tree into another tree is zero since it is an invalid insertion.

This method's particular utility in being able to encode the different types of trees (auxiliary vs. elementary) in the grading scheme is one reason that these tree graph encodings should continue to be studied in future work. However, the amount of hoops necessary to jump through in order to establish this class of graphs as useful as a mathematical formulation of TAG motivates us to look at a third graphical encoding of TAG trees, which is even simpler than this version and has particularly interesting linguistic implications.

3.3 TAG as physics graphs

We present an alternative combinatorial definition of graphs that is primarily used in theoretical physics (Marcolli and Port, 2015), which is convenient for defining the colored operads model of tree operations.

3.3.1 Physics definition

A graph $G = (C(G), \mathcal{F}(G), \mathcal{I})$ consists of a set $C(G) = \{v_1, \ldots, v_n\}$ of n *corollas* (nodes distinguished by having half-edges attached to them), a set $\mathcal{F}(G) = \{f_1, \ldots, f_m\}$ of m *flags* (or *half-edges*), and an involution $\mathcal{I} : \mathcal{F}(G) \to \mathcal{F}(G)$ on the flags. The *external edges* $E_{ext}(G)$ of G are the flags $f \in \mathcal{F}(G)$ fixed by the involution, i.e. $\mathcal{I}(f) = f$, while the *internal edges* $E_{int}(G)$ correspond to the two-element subsets $\{f, f'\} \in \binom{\mathcal{F}(G)}{2}$ such that $\mathcal{I}(f) = f'$ (and correspondingly $\mathcal{I}(f') = f$). The *valence* of a corolla $v \in C(G)$ is the number of half-edges attached to it.

Operadic insertion is now represented by joining external edges; adjunction involves splitting an internal edge by dividing it into its component flags and joining two external edges of a new graph, one to each flag.

This model also provides a concrete distinction between non-auxiliary trees and auxiliary trees. Auxiliary trees have exactly two half edges; one corresponding to the root, and another corresponding to the leaf that has the same label as the root in Joshi's original formulation of TAG.[4]

In fact, there is some nuance here. Because we are representing the pre-Lie insertion operation as a summation over all possible insertions, this will be generalized as *all* leaves of auxiliary trees being half edges—the insertion operation of $T \triangleleft S$ then varies across the half edge beneath the insertion point in T being re-attached in turn to each half edge leaf of S. This naturally motivates the colored-operad lens on the TAG Lie algebra—every single leaf is a location where the colored operad will, in turn, insert a terminal node with a half edge attached to it. We return to this idea later on in this section, but note that we will not be writing the trees with terminal symbols and half-edges for all leaves, for sake of convenience.

The concept of half-edges is reminiscent of the usage of half-arcs to represent features in Minimalist grammars, where matching features are joined together. We explore this interpretation further in Section 4.1.

[4] See, for instance, tree S in example (3.6).

3.3.2 Operadic definition of TAG Lie algebra

The following describes how TAG insertion can be implemented via the composition of two unary operad insertions.

Definition 3.5. *Consider the insertion $T \lhd S$, where insertion location is labeled with α. T was originally derived from composing two trees at α via an operad composition:*

$$T = \mathcal{O}(T_1, T_2) = T_1 \circ_\alpha T_2. \qquad (1)$$

This α composition location is exactly where T will be broken apart in order to insert S between the two:

$$T \lhd_\alpha S = \mathcal{O}(\mathcal{O}(T_1, S), T_2) = (T_1 \circ_{\alpha_u} S) \circ_{\alpha_l} T_2$$

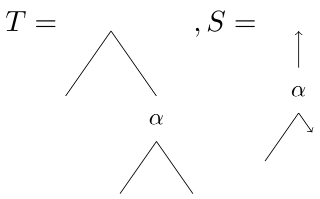
u and l stand for "upper" and "lower" copies of α, respectively.[5] *Note that α_u actually comes from S, and α_l comes from T, because the edge above α in T is what is split into two half edges in order to insert S.*

Because one of the α nodes is coming from the auxiliary tree S, and the other is coming from the tree being adjoined into, T, this results in the grading by vertices is preserved with this formalization of graphs, maintaining the connectedness of the space and hence solving those issues as they pertained to the previous two graph formalisms.

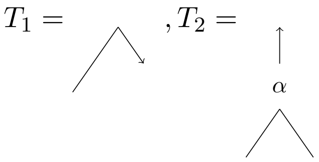
In the following example, we demonstrate the physics graph definition as it is applied to the TAG trees, using our running example, and show how the adjoining operation can be performed via the composition of two operad operations.

Example 3.6. *An example of the insertion operation (outside of the operad). Supposed the grading is by nodes.*

Let S, T as in the previous examples:

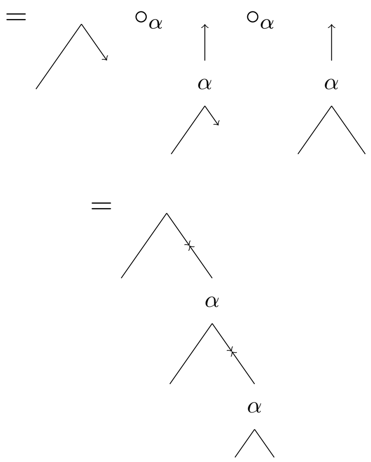


To insert S into T at α, forming $T \lhd_\alpha S$, we break T into T_1 and T_2 such that $T = T_1 \circ_\alpha T_2$, and specifically the edge above α in T is broken into two half edges. Hence, T_2 retains α:



Then inserting S into T at α is simply the composition

$$T \lhd_\alpha S = T_1 \circ_\alpha S \circ_\alpha T_2$$



There is a distinction that must be made regarding the arity of the colored operads we will use in the context of the TAG Lie algebra. In the Lie algebra formalism we sum over insertions at all possible locations, and readjoining at any possible leaf. Then, in order to complete the TAG tree, after readjoining at one of the leaves, the remaining $l-1$ leaves must be "capped off" with operadic insertion of $l-1$ half-edges whose single nodes are labeled with terminals. When we use operadic compositions to describe the insertion operations, we can decompose the composition γ (that fills all inputs at once) into repeated compositions $\circ_i : \mathfrak{D}(n) \times \mathfrak{D}(m) \to \mathfrak{D}(n+m-1)$ that perform a single match of output to input. When the color-matching is also taken into account, these give (1). Thus, we can assume that only one leaf is a half-edge and all the other leaves are filled with terminals. In operadic terms, filling with terminals is part of the algebra over an operad structure, see Giraudo (2019).

4 Implications of Mathematical Formalizations of TAG

The physics definition of a graph makes it easier to recast TAG in terms of the graph grammars in

[5] One may view conceptualizing a tree containing an insertion spot at an interior node as being comprised of two trees who each contain that node on their periphery (the upper copy being a leaf in T_1 and the lower copy being the root of T_2) as linguistically-alternative, but any tree can be viewed as the composition of a subgraph and the quotient graph over the subgraph.

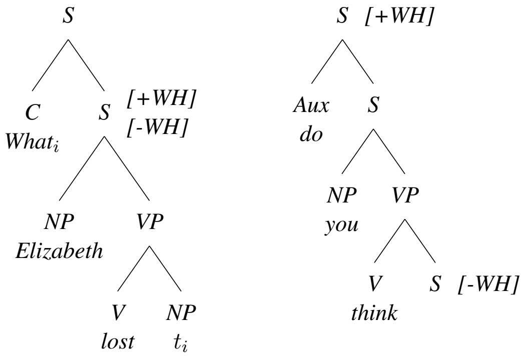
Marcolli and Port (2015). While it was possible to describe the production rules of TAG in terms of the combinatorial definition of graphs, the fact that tree-adjoining must be ultimately viewed as an insertion-elimination operation introduces an unwelcome degree of context-dependence to the formulation. For every label at which adjunction may occur, there needs to be a distinct production rule for every possible degree and every possible combination of labels that the neighbours of this node may have, in order to account for all possible insertions. When using the physics definition, the production rules may be defined in terms of the corollas; in this case, we still need a distinct production rule for each possible valence but the labels of the neighbouring vertices no longer need to be considered.[6]

4.1 Linguistic implications

A striking benefit of this model is that null-adjoining constraints are *automatically* granted based on whether or not an edge is allowed to be split into two half edges. If it is restricted from being two half-edges, then adjoining is not possible. If it is required to split, then adjoining is required. This leads into another immediate linguistic payoff: feature-TAG.

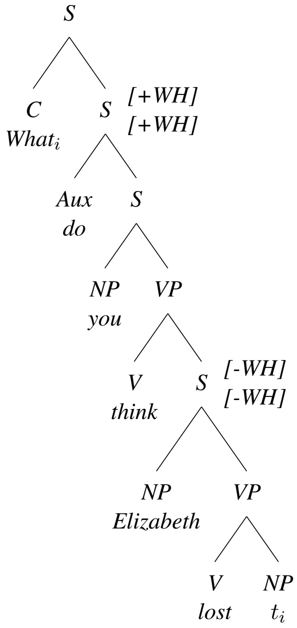
In feature-TAG (Vijay-Shanker and Joshi, 1988), the argument is made that tree adjoining is required when there is a featural mismatch in an elementary tree at a given node, e.g. it has [+WH] from above but [-WH] from below. Then in order to resolve this, the insertion of an auxiliary tree which has [+WH] in the upper node and [-WH] in the lower node would fix this, because the two [+WH] would match as well as the two [-WH], resolving the fact that originally the elementary tree had [+WH] and [-WH] in the same node.

Example 4.1. *To form the question "What do you think Elizabeth lost?", we can use the following single elementary tree T (left) and auxiliary tree S (right):*



In the elementary tree, there is a featural mismatch in the lower S node, because above this S node "What" dictates it is [+WH], but below this S node there is no such lexical time/syntactic feature, meaning the lower feature is [-WH]. This featural mismatch prompts the insertion of S into T, yielding the following well-formed tree with no featural mismatches:

$T \lhd S =$



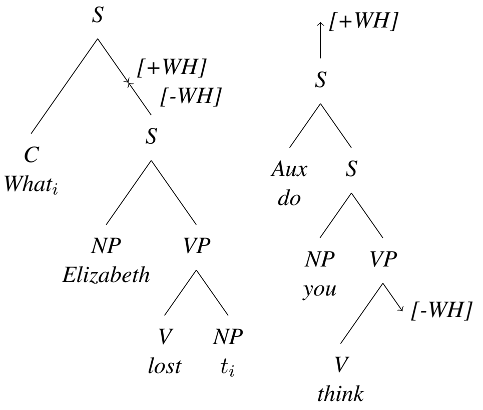
Implementation of feature TAG follows easily from the half-edge model of TAG.[7] We can think about the half edges as carrying the feature labels, so an original α node's parent edge is split in half exactly when the upper half edge has a [+F] and the lower half edge is labeled by [-F] (or vice-versa). Then the auxiliary tree's upper half edge coming out of the root must carry a [+F] and the lower half edge, representing a leaf location, must carry a [-F]. After adjoining the two sets of two half edges will

[6] For an example of this, see §3 of Marcolli and Port (2015).

[7] Note that this implementation is based upon labeling half-edges—this is an inherent component of the physics definition, so no additional functionality is being added to the system.

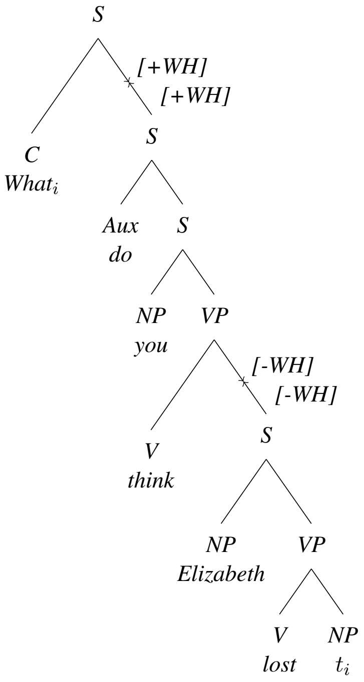
each have featural match. Below, we implement example (4.1) with the physics-based TAG graphs.

Example 4.2. *The trees T and S are, respectively,*



The insertion of S into T follows from the feature mismatch of the two half-edges—and this is resolved with the insertion:

$T \lhd S =$

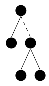
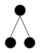


4.2 Mathematical implications

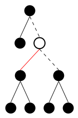
The Lie-algebraic structure provides many further avenues of exploration. The fact that we have managed to define graded and connected Lie algebra structures using TAG is convenient for future investigation of the universal enveloping algebra, since the universal enveloping algebra is graded, connected and furthermore commutative as a Hopf algebra. A graded connected commutative Hopf algebra is known to be free as an algebra (in other words, it is isomorphic to a polynomial algebra), thanks to a theorem of Hopf-Leray, which means that it can easily be implemented in computational algebra programs. If the Hopf algebra arises from a pre-Lie structure, then the coproduct can be defined in terms of the pre-Lie operator, which in our case has an intuitive combinatorial description in terms of trees. Indeed, any monomial over tree generators can be interpreted as a forest, so the Hopf algebra has a basis indexed by forests. All of these properties facilitate explicit calculations in the Hopf algebra.

As mentioned in the introduction, TAG insertion via nodes is only one type of graphical insertion. The other insertion type involves inserting into existing edges via splitting an edge into two with a node in the middle, and then connecting that node to the tree being inserted via another edge, as depicted in the following example.

Example 4.3. *Let S, T be as previously, i.e.*

$T =$  , $S =$  .

Inserting S into T at the dashed edge yields

$T \lhd S =$  ,

where the single dashed edge has been split into two dashed edges via the creation of the white node and a new (red) edge has been created connecting the root of S to the new node.

This creation of two new edges allows the insertion operation to preserve the binary nature of the trees. It is worth recasting TAG insertion as an edge-insertion operation (e.g. above the labeled vertex, as the half-edges constructed in the physics formulation were) and exploring the algebraic differences and implications of these two different types of TAG insertion.

Another interesting perspective is due to a universal property of connected commutative Hopf algebras, which states that any such Hopf algebra is characterized by a unique nontrivial 1-cocycle. For the Connes-Kreimer Hopf algebra of rooted nonplanar trees, which arises from the free pre-Lie algebra on one generator, this cocycle acts on forests by grafting the component trees to a common root. When the forest consists of exactly two trees, this provides a description of the fundamental Merge operation in Marcolli et al. (2025a).

It would be interesting to examine the corresponding cocycles for the Hopf algebras obtained from the TAG Lie algebras. We expect them to be different from the grafting procedure since the pre-Lie operation based on TAG is not free.

5 Conclusion

In this paper, we formalized tree-adjoining grammars mathematically as Lie algebras where the pre-Lie operation was defined via the adjoining operation of TAG. In order to work with TAGs as mathematical objects, we demonstrated that representing the trees as graphs is only feasible in the Lie-algebra setting when we use the physics definition of graphs, allowing us to express single edges as two half-edges. This mathematical motivation also had striking linguistic payoffs, allowing various components of TAGs usually posited as additional constraints to be inherent via the nature of the graphs.

Acknowledgements

We thank Robert Frank for his useful comments and suggestions. The third author is supported by NSF grants DMS-2104330 and DMS-2506176.

References

Pierre Cartier and Frédéric Patras. 2021. *Classical Hopf algebras and their applications*, volume 29 of *Algebra and Applications*. Springer, Cham.

Frédéric Chapoton and Muriel Livernet. 2001. Pre-Lie algebras and the rooted trees operad. *Internat. Math. Res. Notices*, (8):395–408.

Alain Connes and Dirk Kreimer. 1998. Hopf algebras, renormalization and noncommutative geometry. *Comm. Math. Phys.*, 199(1):203–242.

Samuele Giraudo. 2019. Colored operads, series on colored operads, and combinatorial generating systems. *Discrete Math.*, 342(6):1624–1657.

Michael A. Harrison. 1978. *Introduction to Formal Language Theory*. Addison-Wesley Publishing Co., Reading, MA.

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2001. Introduction to automata theory, languages, and computation, 2nd edition. *SIGACT News*, 32(1):60–65.

Aravind K. Joshi. 1987. An introduction to tree adjoining grammars. In *Mathematics of language*, pages 87–114. Benjamins, Amsterdam.

Aravind K Joshi, Leon S Levy, and Masako Takahashi. 1975. Tree adjunct grammars. *Journal of computer and system sciences*, 10(1):136–163.

Aravind K. Joshi and Yves Schabes. 1997. Tree-adjoining grammars. In *Handbook of Formal Languages: Volume 3 Beyond Words*, pages 69–123. Springer Berlin Heidelberg, Berlin, Heidelberg.

Anthony S Kroch and Aravind K Joshi. 1985. The linguistic relevance of tree adjoining grammar.

Matilde Marcolli, Noam Chomsky, and Robert C Berwick. 2025a. *Mathematical Structure of Syntactic Merge: An Algebraic Model for Generative Linguistics*. MIT Press.

Matilde Marcolli, Riny Huijbregts, and Richard K. Larson. 2025b. Hypermagmas and colored operads: Heads, phases, and theta roles. *Preprint*, arXiv:2507.06393.

Matilde Marcolli and Richard K. Larson. 2025. Theta theory: operads and coloring. *Preprint*, arXiv:2503.06091.

Matilde Marcolli and Alexander Port. 2015. Graph grammars, insertion Lie algebras, and quantum field theory. *Math. Comput. Sci.*, 9(4):391–408.

Paul-André Melliès and Noam Zeilberger. 2025. The categorical contours of the Chomsky-Schützenberger representation theorem. *Log. Methods Comput. Sci.*, 21(2):Paper No. 12.

Claudio Procesi. 2007. *Lie groups*. Universitext. Springer, New York. An approach through invariants and representations.

Owen Rambow, K. Vijay-Shanker, and David Weir. 2001. D-tree substitution grammars. *Computational Linguistics*, 27(1):87–121.

Michael Sipser. 1996. *Introduction to the Theory of Computation*, 1st edition. International Thomson Publishing.

K. Vijay-Shanker and A. K. Joshi. 1988. Feature structures based tree adjoining grammars. In *Proceedings of the 12th Conference on Computational Linguistics - Volume 2*, COLING '88, page 714–719, USA. Association for Computational Linguistics.

A Appendix: Proofs of Theorems

Theorem 2.8. *$\triangleleft$ is a pre-Lie operator.*

Proof of Theorem 2.8. It is enough to show that the right Vinberg identity holds for basis elements. Let T_1, T_2, T_3 be trees in X. In the associator $A(T_1, T_2, T_3) = (T_1 \triangleleft T_2) \triangleleft T_3 - T_1 \triangleleft (T_2 \triangleleft T_3)$, the only basis elements with nonzero coefficient are the ones where T_2 and T_3 are adjoined at distinct vertices of T_1, meaning that T_2 and T_3 are disjoint subgraphs of the resulting tree. These terms will appear in $(T_1 \triangleleft T_2) \triangleleft T_3$, but not $T_1 \triangleleft (T_2 \triangleleft T_3)$. Any term where T_3 is adjoined to T_2, or a copy of T_2 in a term of $T_1 \lhd T_2$, necessarily appears in both terms of the associator, hence are cancelled out. This cancellation ensures that $A(T_1, T_2, T_3) = A(T_1, T_3, T_2)$. □

Theorem 2.11. *$\mathcal{T}_{bin,pl}$ is not isomorphic as a pre-Lie algebra to L_{free}, the free pre-Lie algebra on one generator.*

Proof of Theorem 2.11. Suppose, for the sake of contradiction, that $\mathcal{T}_{bin,pl}$ is isomorphic to L_{free}; then there exists an isomorphism $\phi : L_{free} \to \mathcal{T}_{bin,pl}$ such that $\phi(x \lhd y) = \phi(x) \lhd \phi(y)$ for all $x, y \in L_{free}$. (We use the same symbol $\lhd$ to denote the pre-Lie operation on both algebras, since context prevents any ambiguity.)

The contradiction is defined by considering the inverse element $\phi^{-1}(\bullet) \in L_{free}$, and showing that it cannot be well-defined.

We will use the basis on L_{free} given by unlabelled nonplanar trees of arbitrary arity, where trees are graded by its number of nodes.

Since L_{free} is graded, $\mathcal{T}_{bin,pl}$ inherits a grading from the isomorphism, although it may not be an obvious one based on any combinatorial properties of trees. In fact, a basis element of $\mathcal{T}_{bin,pl}$ may not even be a homogeneous element under this grading.

Let T be a tree in $\mathcal{T}_{bin,pl}$; then $T \lhd \bullet = |T|T$, so

$$\phi^{-1}(T) \lhd \phi^{-1}(\bullet) = |T|\phi^{-1}(T).$$

Let $c \in \mathbb{R}$ be the coefficient of $\varnothing$ in $\phi^{-1}(\bullet)$, so that

$$\phi^{-1}(\bullet) = c\varnothing + (\text{higher degree terms}).$$

Let y be a term in $\phi^{-1}(T)$ of minimal degree. Since $\varnothing$ is the unique degree-0 generator in L_{free}, it is the only element that does not raise the degree of y when inserted into it. Since $|T|y$ occurs in the expression $\phi^{-1}(T) \lhd \phi^{-1}(\bullet)$, it can only be the result of performing $\phi^{-1}(T) \lhd c\varnothing$, hence $c = |T|$. But $\phi^{-1}(\bullet)$ cannot depend on any specific T. □

Derivational and Interpretational Equivalence of LIG and HG

Kalen Chang
Department of Linguistics
University of California, Los Angeles
Los Angeles, CA, USA
kalen@ucla.edu

Abstract

Linear Indexed Grammar (LIG) and Head Grammar (HG) are derivationally equivalent: not only do they generate the same string languages, each pair of equivalent grammars generates each string in the same number of ways. Moreover, any compositional interpretation for a grammar in one formalism can be transformed into a compositional interpretation of the equivalent grammar in the other formalism such that the same strings are assigned the same meanings. Although this paper focuses on these two formalisms, it serves as an example for how to compare formalisms beyond the string languages they generate.

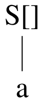
1 Introduction

When considering different ways to compare grammar formalisms, weak generative capacity, or the string languages generated, is one common point of comparison. For example, Vijay-Shanker and Weir (1994) proved that four mildly context-sensitive grammar formalisms generate the same set of string languages. Of those four, this paper will examine two in detail: Linear Indexed Grammars (LIG) and Head Grammars (HG).

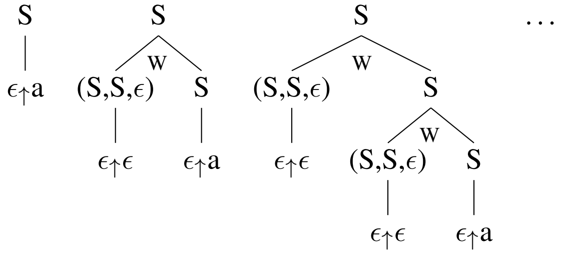
Though generating the correct set of strings is important, linguists use grammars to provide analyses of languages. They may assign internal structure to strings to explain how the subcomponents of a string relate to the whole – compositionality – or how a string may be associated with more than one meaning – ambiguity.

Thus, it is no surprise that linguists use notions beyond weak generative capacity to compare formalisms, such as comparing tree structures associated with grammars (e.g. Bresnan et al., 1982; Frank and Hunter, 2021). However, comparison across tree structures is not infallible. While some formalisms like Tree Adjoining Grammars directly derive tree structures, others, such as HG, only derive strings (or pairs thereof). The only notion of structure present in HG and string-generating formalisms are derivation structures, which record which rules were used to construct the string and how those rules interact. Though grammar formalisms vary wildly in how derivations are structured, there are still ways to compare derivations across formalisms.

For example, we might count the number of derivations assigned to a each string, so that a two-way ambiguous string in a grammar is assigned two different derivations. The translation given in Vijay-Shanker and Weir (1994) does not preserve the number of derivations per string. To see this, consider an extremely simple LIG, which has a single rule S[] → a, and a single derivation.



Following their translation, the corresponding HG has infinitely many derivations:[1]



This problem arises more generally for any pair of weakly equivalent grammars given by their translation: despite being weakly equivalent, the HG will have infinitely many derivations. In fact, no matter how many derivations generate a string in the LIG, there will be infinitely many derivations that generate that string in the HG.

I propose two ways to handle this problem. One involves changing the grammar translation so that the resulting grammar is derivationally equivalent: the LIG and the HG generate the same number of derivations per string. The other way involves

[1] The W under a node indicates wrapping two components of the left daughter around those of the right daughter.

defining a method of interpreting HG derivations given an LIG interpretation function such that the resulting interpreted HG generates the same set of string-meaning pairs, or vice versa.

In this paper, I will show both of these are possible with LIG and HG: they are both derivationally and interpretationally equivalent. After preliminary definitions in Section 2, Section 3 lays out a function to construct, given an arbitrary LIG, a derivationally equivalent HG, i.e. one which has the same number of derivations per string, in a sense comparing the multiset of strings generated. Section 4 presents the reverse translation, from HG to LIG.

In Section 5, I will prove that for any LIG equipped with a homomorphic interpretation function on derivations, there is an HG with a homomorphic interpretation function such that the two interpreted grammars generate the same set of string-meaning pairs (and vice versa). The notion of interpretational equivalence is intended to encompass any compositional property relating to meaning or relations between parts of the derivation that linguists wish to capture in a grammar as an analysis of a language.

Section 6 discusses the relation between the approach taken in this paper and a few other papers which have looked at notions of equivalence across formalisms, and Section 7 concludes the paper.

2 Definitions

2.1 Linear Indexed Grammars

Linear Indexed Grammars (Gazdar, 1988) extend Context-Free Grammars by adding stacks to nonterminals and allowing rules where exactly one daughter inherits the stack from the mother, modulo whatever changes to the stack the rule specifies. In effect, there are now an infinite number of categories represented by nonterminals with stacks, in the same way a pushdown automaton can be viewed as having an infinite number of states.

Formally, an LIG is a 5-tuple of finite sets $\mathcal{G} = (V_\mathrm{N}, V_\mathrm{T}, V_\mathrm{I}, S, P)$, consisting of nonterminal symbols V_N, terminal symbols V_T, stack symbols (indices) V_I, start symbols $S \subseteq V_\mathrm{N}$, and production rules P. Each production rule has one of two forms:

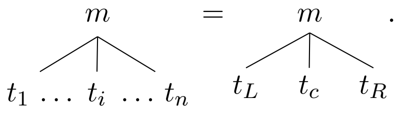
- Leaf: $A[] \to w$, where $A \in V_\mathrm{N}, w \in V_\mathrm{T}^*$
- Branch: $A[\zeta] \to B_1 \dots B_i[\eta] \dots B_n$, where $n \geq 1$; $\zeta, \eta \in V_\mathrm{I} \cup \{\epsilon\}$; $\forall j.\, B_j \in V_\mathrm{N}$

Let n denote the rank of a Branch rule, and Leaf rules have rank 0. I assume each Branch rule is either Push ($\zeta = \epsilon \neq \eta$), Pop ($\eta = \epsilon \neq \zeta$), or No Change ($\zeta = \eta = \epsilon$).[2]

Additionally, I will abbreviate such a rule as $A[\zeta] \to B_L\, B_c[\eta]\, B_R$, where $B_c = B_i$ is the designated/central daughter of A which receives the remainder of the stack, $B_L = B_1 \dots B_{i-1}$ represents the daughters to the left of the center, and similarly $B_R = B_{i+1} \dots B_n$ the daughters right of the center.

Treat LIG production rules as a ranked alphabet, where the rank of each alphabet symbol is the rank of the rule. Let the set of trees over such rules be $\mathcal{T}_\mathrm{LIG}$ (i.e. where the number of daughters of a node matches the rank of the rule of that node). LIG derivations (or derivation trees) are elements of $\mathcal{T}_\mathrm{LIG}$. Define $\mathcal{T}(\mathcal{G})$ as the set of trees over the rules of $\mathcal{G}$.

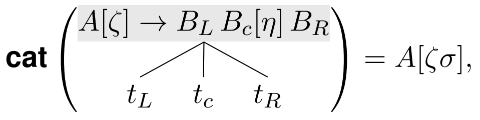
I will use the following abbreviation for trees, similar to rules: if $m = A[\zeta] \to B_1 \dots B_i[\eta] \dots B_n$, $t_L = t_1, \dots, t_{i-1}$, $t_c = t_i$, and $t_R = t_{i+1}, \dots, t_n$ where each $t_j \in \mathcal{T}(\mathcal{G})$, then



Let **cat** denote the category ($V_\mathrm{N} \times V_\mathrm{I}^*$) of a derivation. For trees consisting of a single node:

$$\mathbf{cat}(A[] \to w) = A[]$$

For larger trees: if $\mathbf{cat}^*(t_L) = B_L[]$, $\mathbf{cat}(t_c) = B_c[\eta\sigma]$ ($\sigma \in V_\mathrm{I}^*$), and $\mathbf{cat}^*(t_R) = B_R[]$, then[3]



else it is undefined.

An LIG derivation d is well-formed with respect to an LIG $\mathcal{G}$, i.e. $\mathbf{wf}_\mathcal{G}(d)$, iff $\mathbf{cat}(d) = A[s]$ for $A \in V_\mathrm{N}$, $s \in V_\mathrm{I}^*$, and for every node r in d, $r \in P$. Let $\mathcal{D}(\mathcal{G})$ be the set of LIG derivations well-formed with respect to $\mathcal{G}$. A well-formed LIG derivation d is complete with respect to an LIG $\mathcal{G}$, i.e. $\mathbf{cwf}_\mathcal{G}(d)$, iff $\mathbf{cat}(d) = A[]$ for some $A \in S$. Let $\mathcal{D}_\mathrm{LIG}$ be the set of all well-formed LIG derivations, i.e. the set of derivations which are well-formed with respect to any LIG.

[2] This assumption does not affect any equivalence results; simply replace any rule which performs more than one stack actions with a set of rules which only perform one.

[3] f^* denotes mapping f over a set or list of arguments.

A spine is a node which is not a designated daughter, along with the maximal chain of designated daughters it dominates. Define the sequence $\textbf{flagseq}(d) = [\textbf{flag}(r) \mid r \in s]$, where s is the spine of the root of d, and

$$\textbf{flag}(A[\zeta] \to A_L\, A_c[\eta]\, A_R) = \begin{cases} (_\eta & \text{if } \eta \neq \epsilon, \\ )_\zeta & \text{if } \zeta \neq \epsilon, \\ \epsilon & \text{otherwise.} \end{cases}$$

Lemma 1. For a well-formed derivation d, if $\textbf{cat}(d) = A[], A \in V_{\mathrm{N}}$, then $\textbf{flagseq}(d)$ is a Dyck word, i.e. one generated by $S \to \epsilon \mid (_\eta\ S\ )_\eta\ S$, for each $\eta \in V_{\mathrm{I}}$.

The yield $\textbf{y} : \mathcal{D}(\mathcal{G}) \to V_{\mathrm{T}}^*$ is the result of reading the terminal symbols at the leaves of the tree.

$$\textbf{y}(A[] \to w) = w$$

$$\textbf{y}\left(\begin{array}{ccc} & m & \\ t_L & t_c & t_R \end{array}\right) = \textbf{y}^*(t_L)\textbf{y}(t_c)\textbf{y}^*(t_R)$$

The language of a grammar $\mathcal{L}(\mathcal{G})$ is given as the set of yields of the complete well-formed derivations of $\mathcal{G}$: $\mathcal{L}(\mathcal{G}) = \{\textbf{y}(d) \mid \textbf{cwf}_{\mathcal{G}}(d)\}$

2.2 LIG Tree contexts

In this subsection, I will define LIG tree contexts, which are closely related to LIG derivations, to serve as a useful intermediate structure in the translation from LIG to HG derivations. Intuitively, LIG tree contexts are LIG derivation trees where the rule at the bottom of the root spine has been removed. Accordingly, a context paired with a Leaf rule is isomorphic to a derivation.

LIG tree contexts $\mathcal{K}_{\mathrm{LIG}}$ is the smallest set s.t.

- $\Box \in \mathcal{K}_{\mathrm{LIG}}$
- If $m \in P$ has rank $n \neq 0$; $\gamma_1, \ldots, \gamma_n \in \mathcal{K}_{\mathrm{LIG}}$; and $\beta_1, \ldots, \beta_{i-1}, \beta_{i+1}, \ldots, \beta_n$ are Leaf rules; then $\begin{array}{ccccc} & & m & & \\ (\gamma_1, \beta_1) & \ldots & \gamma_i & \ldots & (\gamma_n, \beta_n) \end{array} \in \mathcal{K}_{\mathrm{LIG}}$

and similarly for $\mathcal{K}(\mathcal{G})$, LIG tree contexts over a grammar $\mathcal{G}$.

It will be handy to have a function **pt** to extract the node at the bottom of the spine of a derivation (the tail of the spine), which will be a Leaf (rank 0) rule (P_L). Define $\textbf{pt} : \mathcal{T}(\mathcal{G}) \to \mathcal{K}(\mathcal{G}) \times P_L(\mathcal{G})$ as:

- For $\textbf{rank}(\beta) = 0$:
  $\textbf{pt}(\beta) = (\Box, \beta)$
- For $\textbf{rank}(m) > 0$:
  $$\textbf{pt}\left(\begin{array}{ccc} & m & \\ t_L & t_c & t_R \end{array}\right) = \left(\begin{array}{ccc} & m & \\ \textbf{pt}^* t_L & \Gamma & \textbf{pt}^* t_R \end{array}, \beta\right),$$
  where $(\Gamma, \beta) = \textbf{pt}(t_c)$.

Let $\Gamma\{\Delta_1 \mapsto \Delta_2\}$ denote locating the subcontext or subtree Δ_1 in Γ and replacing it with the context or tree Δ_2. Formally, $\Gamma\{\Delta_1 \mapsto \Delta_2\} =$

$$\begin{cases} \Delta_2 & \text{if } \Gamma = \Delta_1 \\ \begin{array}{ccc} & m & \\ t_L & \gamma_c\{\Delta_1 \mapsto \Delta_2\} & t_R \end{array} & \text{if } \Gamma = \begin{array}{ccc} & m & \\ t_L & \gamma_c & t_R \end{array} \end{cases}$$

and let $\Gamma\{\Delta\} = \Gamma\{\Box \mapsto \Delta\}$.

It is straightforward to show that if $\textbf{pt}(d) = (\Gamma, \beta)$, then $\Gamma\{\beta\} = d$, and so **pt** is invertible. From now on, I will treat elements of $\mathcal{T}_{\mathrm{LIG}}$ and $\mathcal{K}_{\mathrm{LIG}} \times P_L$ as interchangeable.

Let $\mathcal{C}(\mathcal{G})$ denote the set of well-formed LIG tree contexts over $\mathcal{G}$ — those which come from well-formed LIG derivations.

$$\mathcal{C}(\mathcal{G}) = \{\Gamma \mid d \in \mathcal{D}(\mathcal{G}), \textbf{pt}(d) = (\Gamma, \beta)\}$$

I define the notion of category for an LIG context: $\textbf{catc} : \mathcal{C}(\mathcal{G}) \times V_{\mathrm{N}} \to V_{\mathrm{N}} \times V_{\mathrm{I}}^*$, which returns the category of what the context would be if it was filled by a lexical rule of the given category.

$$\textbf{catc}(\Gamma, B) = \textbf{cat}(\Gamma\{B[] \to w\})$$

Thus, $\textbf{catc}(\Box, B) = B[]$.

2.3 Head Grammars

Head Grammars (Pollard, 1984; Roach, 1987) increase the expressive power of CFGs by manipulating a slightly more complicated structure: a pair of strings, separated by an arrow $_\uparrow$. Derivations manipulate pairs of strings, and a new kind of rule is added: one which wraps one pair of strings around another.

Formally, an HG is a 4-tuple $G = (V_{\mathrm{N}}, V_{\mathrm{T}}, S, P)$, consisting of nonterminal symbols V_{N}, terminal symbols V_{T}, start symbols $S \subseteq V_{\mathrm{N}}$, and production rules P. Each production rule has one of three forms:

- Leaves: $A \to u_\uparrow v$, where $A \in V_{\mathrm{N}}; u, v \in V_{\mathrm{T}}^*$
- Wrap: $A \xrightarrow{W} B_l\, B_r$, where $A, B_l, B_r \in V_{\mathrm{N}}$
- Concat$_i$: $A \xrightarrow{C_i} B_1 \ldots B_n$, where $i \leq n$ and $A, B_1, \ldots, B_n \in V_{\mathrm{N}}$

The rank of Leaves rules is 0, Wrap rules 2, and Concat rules n. An abbreviation similar to that for LIG rules will be applied to Concat rules: $A \xrightarrow{C_i} B_1 \dots B_n = A \xrightarrow{C_c} B_L\, B_c\, B_R$.

Treating HG production rules as a ranked alphabet, where the rank of each alphabet symbol is the rank of the rule, let the set of trees over such rules be $\mathcal{T}_{\text{HG}}$, and an HG derivation is a tree of that set. Define $\mathcal{T}(\mathcal{G})$ as the set of trees over the rules of $\mathcal{G}$.

Let **cat** denote the category (V_{N}) of a derivation.

$$\mathbf{cat}(A \to u_{\uparrow}v) = A$$

If $\mathbf{cat}(t_l) = B_l$ and $\mathbf{cat}(t_r) = B_r$, then

$$\mathbf{cat}\left(\begin{array}{c} A \xrightarrow{W} B_l\, B_r \\ \diagup \quad \diagdown \\ t_l \quad\; t_r \end{array}\right) = A.$$

If $\mathbf{cat}^*(t_L) = B_L$, $\mathbf{cat}(t_c) = B_c$, and $\mathbf{cat}^*(t_R) = B_R$, then

$$\mathbf{cat}\left(\begin{array}{c} A \xrightarrow{C_c} B_L\, B_c\, B_R \\ \diagup \quad | \quad \diagdown \\ t_L \quad t_c \quad t_R \end{array}\right) = A,$$

else it is undefined.

An HG derivation d is well-formed with respect to an HG $\mathcal{G}$, i.e. $\mathbf{wf}_{\mathcal{G}}(d)$, iff $\mathbf{cat}(d) \in V_{\text{N}}$, and for every node r in d, $r \in P$. Let $\mathcal{D}(\mathcal{G})$ be the set of HG derivations well-formed with respect to $\mathcal{G}$. A well-formed derivation d is complete with respect to an HG $\mathcal{G}$, i.e. $\mathbf{cwf}_{\mathcal{G}}(d)$, iff $\mathbf{cat}(d) \in S$. Let $\mathcal{D}_{\text{HG}}$ be the set of all well-formed HG derivations, i.e. derivations well-formed with respect to any HG.

The yield function $\mathbf{y} : \mathcal{D}(\text{HG}) \to V_{\text{T}}^* \times V_{\text{T}}^*$ is defined as:

$$\mathbf{y}(A \to u_{\uparrow}v) = u_{\uparrow}v$$

If $\mathbf{y}(t_l) = x_{l\uparrow}y_l$, and $\mathbf{y}(t_r) = x_{r\uparrow}y_r$:

$$\mathbf{y}\left(\begin{array}{c} A \xrightarrow{W} B_l\, B_r \\ \diagup \quad \diagdown \\ t_l \quad\; t_r \end{array}\right) = x_l x_{r\uparrow} y_r y_l$$

If $\mathbf{y}(t_j) = x_{j\uparrow}y_j$ for all $1 \leq j \leq n$:

$$\mathbf{y}\left(\begin{array}{c} A \xrightarrow{C_c} B_L\, B_c\, B_R \\ \diagup \quad | \quad \diagdown \\ t_L \quad t_c \quad t_R \end{array}\right) = x_1 y_1 \dots x_{i\uparrow} y_i \dots x_n y_n$$

The language of a grammar $\mathcal{L}(\mathcal{G})$ is defined as:

$$\mathcal{L}(\mathcal{G}) = \{uv \mid \mathbf{cwf}_{\mathcal{G}}(d), \mathbf{y}(d) = u_{\uparrow}v\}$$

3 Derivational equivalence: LIG to HG

Two grammars are *derivationally equivalent* iff they generate the same number of derivations for each string. Two formalisms $\mathcal{F}_1$ and $\mathcal{F}_2$ are *derivationally equivalent* iff for each grammar $\mathcal{G}_1 \in \mathcal{F}_1$, there is some $\mathcal{G}_2 \in \mathcal{F}_2$ that is derivationally equivalent to $\mathcal{G}_1$, and vice versa. In this section, I will prove that for every LIG, there is a derivationally equivalent HG.

First, given the source LIG $\mathcal{G}$, I will define the resulting HG $\mathcal{G}' = \text{LH}(\mathcal{G})$. Next, I define a function $\mathbf{lh} : \mathcal{D}_{\text{LIG}} \to \mathcal{D}_{\text{HG}}$ which transforms a well-formed LIG derivation into a well-formed HG derivation while preserving its yield and category.

To show that $\mathcal{G}$ and $\mathcal{G}'$ are derivationally equivalent, I show that the set of well-formed derivations of $\mathcal{G}$ are bijective to the derivations of $\mathcal{G}'$ via **lh**, and since **lh** preserves yields, so too are the sets of well-formed derivations for a given string. The diagram [4] below commutes, that is, $\mathbf{lh}^*(\mathcal{D}(\mathcal{G})) = \mathcal{D}(\text{LH}(\mathcal{G})) = \mathcal{D}(\mathcal{G}')$.

$$\begin{tikzcd} \text{LIG} \arrow[r, "\text{LH}"] \arrow[d, "\mathcal{D}"] & \text{HG} \arrow[d, "\mathcal{D}"] \\ \mathcal{P}(\mathcal{D}_{\text{LIG}}) \arrow[r, "\mathbf{lh}^*"] & \mathcal{P}(\mathcal{D}_{\text{HG}}) \end{tikzcd}$$

The proof proceeds by showing $\mathbf{lh}^*(\mathcal{D}(\mathcal{G})) \subseteq \mathcal{D}(\text{LH}(\mathcal{G}))$, and then the reverse, relying on the fact that **lh** has an inverse $\mathbf{lh}^{-1}$.

3.1 Grammar translation

Theorem 2. For any LIG $\mathcal{G}$, we can construct an HG $\text{LH}(\mathcal{G}) = \mathcal{G}'$ s.t. $\mathcal{G}$ and $\mathcal{G}'$ are derivationally equivalent, i.e. $\mathcal{D}(\mathcal{G})$ and $\mathcal{D}(\mathcal{G}')$ contain the same number of derivations per string.

This theorem will be proved over the next few subsections. First, I present the construction of $\text{LH}(\mathcal{G})$. Given an LIG $\mathcal{G} = (V_{\text{N}}, V_{\text{T}}, V_{\text{I}}, S, P)$, construct HG $\mathcal{G}' = \text{LH}(\mathcal{G}) = (V'_{\text{N}}, V_{\text{T}}, S, P')$ such that

- $V'_{\text{N}} = V_{\text{N}} \cup \underline{V_{\text{N}}} \cup \{\frac{A\eta}{B} \mid A \in V_{\text{N}}, B \in V_{\text{N}} \cup \underline{V_{\text{N}}}, \eta \in V_{\text{I}} \cup \{\epsilon\}\}$, where $\underline{V_{\text{N}}} = \{\underline{A} \mid A \in V_{\text{N}}\}$

- For each rule in P: $A[\zeta] \to A_L\, A_c[\eta]\, A_R$; add to P': $\frac{A\zeta}{\underset{\sim}{B}} \xrightarrow{C_c} A_L\, \frac{A_c\eta}{B}\, A_R$, for all $B \in V_{\text{N}}$, where $\underset{\sim}{B}$ represents two separate instances of the rule, one with B and one with $\underline{B}$.

[4] $\mathcal{P}$ denotes the powerset.

- For each rule in P: $A[] \to w$; add to P': $\underset{\sim}{A} \to \epsilon_\uparrow w$.
- Add these HG rules to P', for all $A, B, D \in V_\mathrm{N}$ and $\eta \in V_\mathrm{I}$:
  - $\frac{A\eta}{B} \xrightarrow{W} \frac{A\epsilon}{\underline{D}}\;\frac{D\eta}{\underline{B}}$ COMP(OSE)
  - $A \xrightarrow{W} \frac{A\epsilon}{\underline{B}}\;\underline{B}$ FILL
  - $\frac{A\epsilon}{A} \to \epsilon_\uparrow\epsilon$ EMPTY

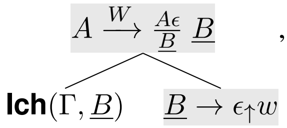
The construction of the HG $\mathcal{G}'$ is similar to that given in Vijay-Shanker and Weir (1994), with several modifications to ensure derivational equivalence. The additional nonterminals $\underset{\sim}{V_\mathrm{N}}$ indicate a subtree whose root is a rule originating in the LIG (either a Leaf or Branch rule). This will prevent spurious applications of the HG structural rules (COMP, FILL, and EMPTY).

Additionally, there are additional nonterminals written as fractions: $\frac{A\eta}{B}$, presented in Vijay-Shanker and Weir (1994) as (A, B, η). This HG nonterminal corresponds to an LIG tree context that would have category $A[\eta\sigma]$, if the gap was filled in with a tree with category $B[\sigma]$. Combining these fraction nonterminals with Wrap rules allows the HG to emulate the stack actions of the LIG.

3.2 The lh and lch translation functions

Let $\mathcal{G}$ be an arbitrary LIG, and $\mathcal{G}' = \textsc{lh}(\mathcal{G})$. Then **lh** : $\mathcal{D}(\mathcal{G}) \to \mathcal{D}(\mathcal{G}')$ converts well-formed LIG derivations to well-formed HG derivations. Since **pt** can produce pairs of LIG contexts and Leaf rules from trees, I define **lh** on some derivations with the tail of the spine already separated for convenience. After defining **lh**, I prove that the HG derivation produced matches the category of the source LIG derivation.

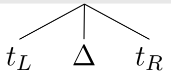
- **lh**$(B[] \to w) = B \to \epsilon_\uparrow w$
- **lh**$(\Gamma\{B[] \to w\}) =$


$A \xrightarrow{W} \frac{A\epsilon}{\underline{B}}\;\underline{B}$
lch$(\Gamma, \underline{B})$
$\underline{B} \to \epsilon_\uparrow w$

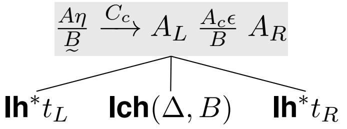
,
where $A \to \ldots$ is at the root of Γ.

A separate function to translate LIG tree contexts combined with an LIG category to HG derivations is given as **lch** : $\mathcal{C}(\mathcal{G}) \times \underset{\sim}{V_\mathrm{N}} \to \mathcal{D}(\mathcal{G}')$.

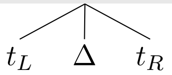
- **lch**$(\square, B) = \frac{B\epsilon}{B} \to \epsilon_\uparrow\epsilon$
- If $\Gamma = A[\eta] \to A_L\, A_c[]\, A_R$ and $\eta \in V_\mathrm{I} \cup \{\epsilon\}$,



then **lch**$(\Gamma, \underset{\sim}{B}) =$

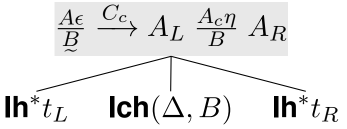

$\frac{A\eta}{\underset{\sim}{B}} \xrightarrow{C_c} A_L\;\frac{A_c\epsilon}{B}\;A_R$
lh$^*t_L$
lch(Δ, B)
lh$^*t_R$

- If $\Gamma = A[] \to A_L\, A_c[\eta]\, A_R$ and $\eta \in V_\mathrm{I}$, then

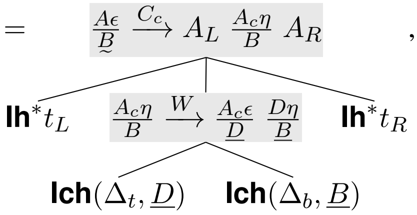


there are two subcases. Decompose **flagseq**(Γ) into $(_\eta u)_\eta v$, where u, v are Dyck words. Let Δ_b be the context at the node corresponding to $)_\eta$ in the decomposition, and let $\Delta_t = \Delta\{\Delta_b \mapsto \square\}$.

(a) If $\Delta_t = \square$ (i.e. the root of Δ is a Pop η rule), then **lch**$(\Gamma, \underset{\sim}{B}) =$


$\frac{A\epsilon}{\underset{\sim}{B}} \xrightarrow{C_c} A_L\;\frac{A_c\eta}{B}\;A_R$
lh$^*t_L$
lch(Δ, B)
lh$^*t_R$

(b) Otherwise,
lch$(\Gamma, \underset{\sim}{B}) =$


$\frac{A\epsilon}{\underset{\sim}{B}} \xrightarrow{C_c} A_L\;\frac{A_c\eta}{B}\;A_R$
lh$^*t_L$
$\frac{A_c\eta}{B} \xrightarrow{W} \frac{A_c\epsilon}{\underline{D}}\;\frac{D\eta}{\underline{B}}$
lh$^*t_R$
lch$(\Delta_t, \underline{D})$
lch$(\Delta_b, \underline{B})$

,
where $D[\eta] \to \ldots$ is the top node of Δ_b.

Note that the **flagseq** of Δ_t is exactly u, a Dyck word, and that of Δ_b is exactly $)_\eta v$, with v a Dyck word. Since the root of Δ_b has $)_\eta$ as its flag, the root node of Δ_b is the Pop η rule that corresponds to the Push η rule at the root of Γ.

Theorem 3. **lh** preserves categories
cat$(\textbf{lh}(d))[] =$ **cat**(d), and thus the image of **lh** contains only well-formed HG derivations.

Proof. The proof will proceed by induction on these two statements:

(i) for $d \in \mathcal{D}(\text{LIG})$, if **cat**$(d) = A[]$, then **cat**$(\textbf{lh}(d)) = A$

(ii) for $\Gamma \in \mathcal{C}(\text{LIG})$, if **catc**$(\Gamma, B) = A[\eta]$ $(\eta \in V_I \cup \{\epsilon\})$ and **lch**(Γ, B) is defined, then **cat**$(\textbf{lch}(\Gamma, \underset{\sim}{B})) = \frac{A\eta}{\underset{\sim}{B}}$

Base cases:

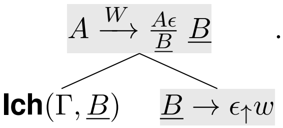
(i) **cat**$(B[] \to w) = B[]$
cat$(\textbf{lh}(B[] \to w)) =$ **cat**$(B \to \epsilon_\uparrow w) = B$

(ii) **catc**$(\square, B) = B[]$
cat$(\textbf{lch}(\square, B)) =$ **cat**$(\frac{B\epsilon}{B} \to \epsilon_\uparrow\epsilon) = \frac{B\epsilon}{B}$

Inductive cases:

(i) Let $d = \Gamma\{B[] \to w\}$, and $\mathbf{cat}(d) = A[] = \mathbf{catc}(\Gamma, B)$.

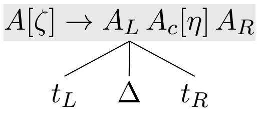
Let $d' = \mathbf{lh}(d) =$


$A \xrightarrow{W} \frac{A\epsilon}{\underline{B}}\ \underline{B}$
$\mathbf{lch}(\Gamma, \underline{B})$
$\underline{B} \to \epsilon_{\uparrow} w$

.

By the IH (ii), since $\mathbf{catc}(\Gamma, B) = A[]$, $\mathbf{cat}(\mathbf{lch}(\Gamma, \underline{B})) = \frac{A\epsilon}{\underline{B}}$, so $\mathbf{cat}(d') = A$.

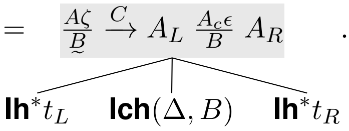
(ii) Let $\Gamma =$



with $\mathbf{catc}(\Gamma, B) = A[\zeta]$, $\mathbf{catc}(\Delta, B) = A_c[\eta]$.

Since $\mathbf{cat}^*(t_L) = A_L[]$, by the IH(i), $\mathbf{cat}^*(\mathbf{lh}^*(t_L)) = A_L$, and similarly for t_R. Let $d' = \mathbf{lch}(\Gamma, \underset{\sim}{B})$. It follows that (ii) is true at Γ by considering two possibilities for Γ:

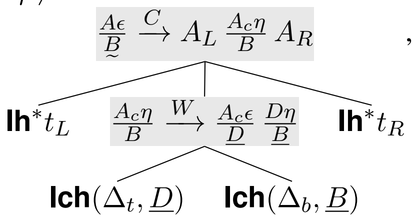
(a) If $\eta = \epsilon$: $d' =$


$\frac{A\zeta}{\underset{\sim}{B}} \xrightarrow{C} A_L\, \frac{A_c\epsilon}{\underline{B}}\, A_R$
$\mathbf{lh}^* t_L$
$\mathbf{lch}(\Delta, B)$
$\mathbf{lh}^* t_R$

.

Since $\mathbf{catc}(\Delta, B) = A_c[]$, by the IH $\mathbf{cat}(\mathbf{lch}(\Delta, B)) = \frac{A_c\epsilon}{\underline{B}}$, so $\mathbf{cat}(d') = \frac{A\zeta}{\underset{\sim}{B}}$.

(b) If $\eta \neq \epsilon$:
$d' =$


$\frac{A\epsilon}{\underset{\sim}{B}} \xrightarrow{C} A_L\, \frac{A_c\eta}{B}\, A_R$
$\mathbf{lh}^* t_L$
$\frac{A_c\eta}{B} \xrightarrow{W} \frac{A_c\epsilon}{\underline{D}}\ \frac{D\eta}{\underline{B}}$
$\mathbf{lh}^* t_R$
$\mathbf{lch}(\Delta_t, \underline{D})$
$\mathbf{lch}(\Delta_b, \underline{B})$

,

where Δ_t and Δ_b are as defined above in $\mathbf{lch}$.

As noted above, the root of Δ_b is a Pop rule $D[\eta] \to \ldots$, and the designated daughter of that node is a context whose $\mathbf{flagseq}$ is a Dyck word. Thus, by the IH (i), $\mathbf{catc}(\Delta_b, B) = D[\eta]$ and by the IH (ii) $\mathbf{cat}(\mathbf{lch}(\Delta_b, \underline{B})) = \frac{D\eta}{\underline{B}}$.

Also, as noted above, $\mathbf{flagseq}(\Delta_t)$ is a Dyck word. By the IH (ii), $\mathbf{catc}(\Delta, B) = A_c[]$ and we know $\Delta_t = \Delta\{\Delta_b \mapsto \square\}$, so $\mathbf{catc}(\Delta_t, D) = A_c[]$. By the IH (ii), $\mathbf{cat}(\mathbf{lch}(\Delta_t, \underline{D})) = \frac{A_c\epsilon}{\underline{D}}$

Thus, $\mathbf{cat}(d') = \frac{A\epsilon}{B}$. □

Corollary 4. If d is a complete well-formed derivation of LIG $\mathcal{G}$, then $\mathbf{lh}(d)$ is a complete well-formed derivation of HG $\mathcal{G}' = \textsc{lh}(\mathcal{G})$. This shows $\mathbf{lh}^*(\mathcal{D}(\mathcal{G})) \subseteq \mathcal{D}(\textsc{lh}(\mathcal{G}))$.

Proof. This follows fairly straightforwardly from the previous theorem. Given any complete well-formed derivation $d \in \mathcal{D}(\mathcal{G})$, we can easily inspect that each rule used in $\mathbf{lh}(d)$ is a rule of $P_{\mathcal{G}'}$. Furthermore, $\mathcal{G}$ and $\mathcal{G}'$ share the same start symbols S, so $\mathbf{lh}(d)$ must be complete and well-formed with respect to $\mathcal{G}'$. □

The proof that $\mathbf{lh}$ preserves yields is very similar to that of Theorem 3 and will not be provided here. Moreover, it follows as a corollary of Theorem 10.

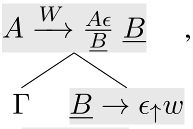
Example 1. Here is an example of an LIG $\mathcal{G}_1$ which generates $\{ww \,|\, w \in \{a,b\}^*\}$. Figure 1 shows an LIG derivation $d_1 \in \mathcal{D}(\mathcal{G}_1)$ for $abbabb$, and the HG derivation $\mathbf{lh}(d_1)$, which is in $\mathcal{D}(\textsc{lh}(\mathcal{G}_1))$.
$\mathcal{G}_1 = (\{\text{S,T,A,B}\},\{\text{a,b}\},\{1,2\},\{\text{S}\},P_1)$ where $P_1 = \{\text{S}[] \to \text{A S}[1],\ \text{S}[] \to \text{B S}[2],\ \text{S}[] \to \text{T}[],\ \text{T}[1] \to \text{T}[]\text{ A},\ \text{T}[2] \to \text{T}[]\text{ B},\ \text{T}[] \to \epsilon,\ \text{A}[] \to \text{a},\ \text{B}[] \to \text{b}\}$

3.3 Derivational equivalence

In this subsection, I complete the proof of Theorem 2 by proving that $\mathcal{D}(\textsc{lh}(\mathcal{G})) \subseteq \mathbf{lh}^*(\mathcal{D}(\mathcal{G}))$. To do that, I will show that $\mathbf{lh}$ and $\mathbf{lch}$ are invertible, with their inverses identified below. $\mathbf{lh}^{-1} : \mathcal{D}(\mathcal{G}') \to \mathcal{C}(\mathcal{G}) \times P_{\text{L}}(\mathcal{G})$ is defined over HG derivations with a simple category A, and $\mathbf{lch}^{-1} : \mathcal{D}(\mathcal{G}') \to \mathcal{C}(\mathcal{G}) \times \underset{\sim}{V_{\text{N}}}$ over those with a fractional category $\frac{A\eta}{\underset{\sim}{B}}$.

(i) $\mathbf{lh}^{-1}(B \to \epsilon_{\uparrow} w) = (\square, B[] \to w)$

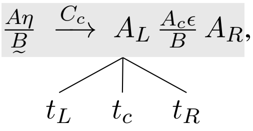
(ii) if $d' =$


$A \xrightarrow{W} \frac{A\epsilon}{\underline{B}}\ \underline{B}$
Γ
$\underline{B} \to \epsilon_{\uparrow} w$

, then $\mathbf{lh}^{-1}(d') =$
$(\mathbf{lch}^{-1}(\Gamma), B[] \to w)$

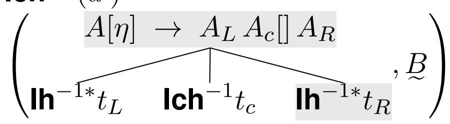
(iii) $\mathbf{lch}^{-1}(\frac{B\epsilon}{B} \to \epsilon_{\uparrow}\epsilon) = (\square, B)$

(iv) if $d' =$

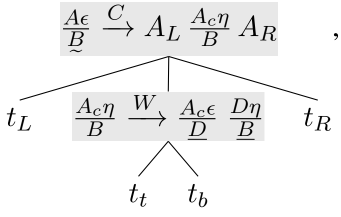

$\frac{A\eta}{\underset{\sim}{B}} \xrightarrow{C_c} A_L\, \frac{A_c\epsilon}{B}\, A_R$,
t_L
t_c
t_R

then

$\mathbf{lch}^{-1}(d') =$



(v) if $d' =$


$\frac{A\epsilon}{\underset{\sim}{B}} \xrightarrow{C} A_L\, \frac{A_c\eta}{B}\, A_R$
t_L
$\frac{A_c\eta}{B} \xrightarrow{W} \frac{A_c\epsilon}{\underline{D}}\ \frac{D\eta}{\underline{B}}$
t_R
t_t
t_b

, then

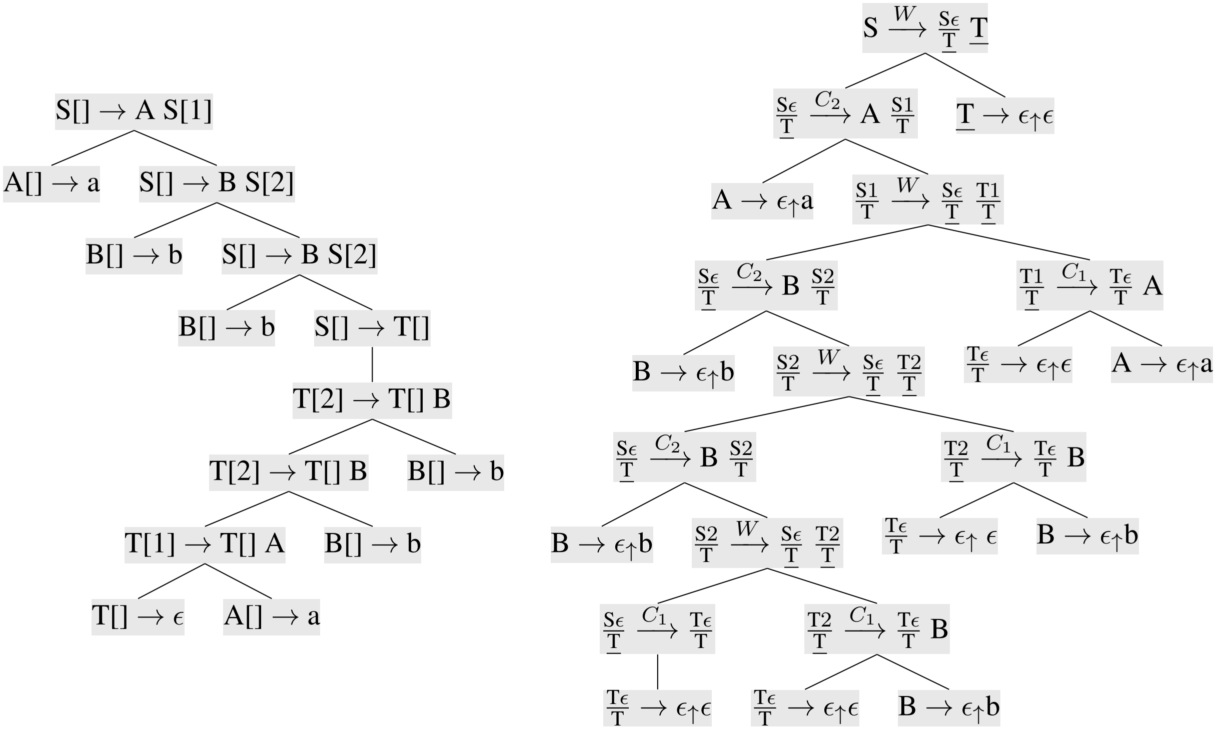

S[] → A S[1]
A[] → a
S[] → B S[2]
B[] → b
S[] → B S[2]
B[] → b
S[] → T[]
T[2] → T[] B
T[2] → T[] B
B[] → b
T[1] → T[] A
B[] → b
T[] → ϵ
A[] → a
$\text{S} \xrightarrow{W} \frac{\text{S}\epsilon}{\underline{\text{T}}}\ \underline{\text{T}}$
$\frac{\text{S}\epsilon}{\underline{\text{T}}} \xrightarrow{C_2} \text{A}\ \frac{\text{S1}}{\text{T}}$
$\underline{\text{T}} \to \epsilon_\uparrow\epsilon$
$\text{A} \to \epsilon_\uparrow\text{a}$
$\frac{\text{S1}}{\text{T}} \xrightarrow{W} \frac{\text{S}\epsilon}{\underline{\text{T}}}\ \frac{\text{T1}}{\underline{\text{T}}}$
$\frac{\text{S}\epsilon}{\underline{\text{T}}} \xrightarrow{C_2} \text{B}\ \frac{\text{S2}}{\text{T}}$
$\frac{\text{T1}}{\underline{\text{T}}} \xrightarrow{C_1} \frac{\text{T}\epsilon}{\text{T}}\ \text{A}$
$\text{B} \to \epsilon_\uparrow\text{b}$
$\frac{\text{S2}}{\text{T}} \xrightarrow{W} \frac{\text{S}\epsilon}{\underline{\text{T}}}\ \frac{\text{T2}}{\underline{\text{T}}}$
$\frac{\text{T}\epsilon}{\text{T}} \to \epsilon_\uparrow\epsilon$
$\text{A} \to \epsilon_\uparrow\text{a}$
$\frac{\text{S}\epsilon}{\underline{\text{T}}} \xrightarrow{C_2} \text{B}\ \frac{\text{S2}}{\text{T}}$
$\frac{\text{T2}}{\underline{\text{T}}} \xrightarrow{C_1} \frac{\text{T}\epsilon}{\text{T}}\ \text{B}$
$\text{B} \to \epsilon_\uparrow\text{b}$
$\frac{\text{S2}}{\text{T}} \xrightarrow{W} \frac{\text{S}\epsilon}{\underline{\text{T}}}\ \frac{\text{T2}}{\underline{\text{T}}}$
$\frac{\text{T}\epsilon}{\text{T}} \to \epsilon_\uparrow\ \epsilon$
$\text{B} \to \epsilon_\uparrow\text{b}$
$\frac{\text{S}\epsilon}{\underline{\text{T}}} \xrightarrow{C_1} \frac{\text{T}\epsilon}{\text{T}}$
$\frac{\text{T2}}{\underline{\text{T}}} \xrightarrow{C_1} \frac{\text{T}\epsilon}{\text{T}}\ \text{B}$
$\frac{\text{T}\epsilon}{\text{T}} \to \epsilon_\uparrow\epsilon$
$\frac{\text{T}\epsilon}{\text{T}} \to \epsilon_\uparrow\epsilon$
$\text{B} \to \epsilon_\uparrow\text{b}$

Figure 1: Derivation of *abbabb* in LIG $\mathcal{G}_1$ (left) and HG $\textsc{lh}(\mathcal{G}_1)$ (right)

$\mathsf{lch}^{-1}(d') =$

$$\left(\begin{array}{ccc} & A[] \to A_L\, A_c[\eta]\, A_R & \\ \mathsf{lh}^{-1*}t_L & (\mathsf{lch}_0^{-1}t_t)\{\mathsf{lch}_0^{-1}t_b\} & \mathsf{lh}^{-1*}t_R \end{array}, \underset{\sim}{B}\right),$$

where lch_0^{-1} denotes the first element returned by lch^{-1}.

It is not hard to verify that these functions composed with lh and lch, respectively, return the identity. I show this is the case for (v):

$\mathsf{lch}(\mathsf{lch}^{-1}(d')) =$

$$\begin{array}{ccc} & \frac{A\epsilon}{\underset{\sim}{B}} \xrightarrow{C} A_L\, \frac{A_c\eta}{B}\, A_R & \\ \mathsf{lh}^*(\mathsf{lh}^{-1*}t_L) & \begin{array}{cc} \multicolumn{2}{c}{\frac{A_c\eta}{B} \xrightarrow{W} \frac{A_c\epsilon}{\underline{D}}\, \frac{D\eta}{\underline{B}}} \\ \mathsf{lch}(\mathsf{lch}^{-1}t_t) & \mathsf{lch}(\mathsf{lch}^{-1}t_b) \end{array} & \mathsf{lh}^*(\mathsf{lh}^{-1*}t_R) \end{array}$$

By induction, $\mathsf{lh}^*(\mathsf{lh}^{-1*}t_{L/R}) = t_{L/R}$ and $\mathsf{lch}(\mathsf{lch}^{-1}t_{t/b}) = t_{t/b}$, and we know that $(\mathsf{lch}^{-1}t_t)\{\mathsf{lch}^{-1}t_b\}$ will be split by lch into $\mathsf{lch}^{-1}t_t$ and $\mathsf{lch}^{-1}t_b$. This is because $\mathsf{cat}(t_t) = \frac{A_c\epsilon}{\underline{D}}$, so $\mathsf{cat}(\mathsf{lch}^{-1}t_t, \underline{D}) = A_c[]$; thus $\mathsf{lch}^{-1}t_t$ is a context whose $\mathsf{flagseq}$ will be a Dyck word.

If $\Gamma = A[\zeta] \to A_L\, A_c[\eta]\, A_R$, then

$$\begin{array}{ccc} t_L & \Delta & t_R \end{array}$$

$\mathsf{lch}^{-1}(\mathsf{lch}(\Gamma, \underset{\sim}{B})) =$

$$\left(\begin{array}{ccc} & A[] \to A_L\, A_c[\eta]\, A_R & \\ \mathsf{lh}^{-1}(\mathsf{lh}^*t_L) & \Delta_x & \mathsf{lh}^{-1}(\mathsf{lh}^*t_R) \end{array}, \underset{\sim}{B}\right), \text{ where}$$

$\Delta_x = (\mathsf{lch}_0^{-1}(\mathsf{lch}(\Delta_t, \underline{D})))\{\mathsf{lch}_0^{-1}(\mathsf{lch}(\Delta_b, \underline{B}))\}$. By induction, we know $\Delta_x = \Delta_t\{\Delta_b\} = \Delta$, and also that $\mathsf{lh}^{-1*}(\mathsf{lh}^*t_{L/R}) = t_{L/R}$; therefore, $\mathsf{lch}^{-1}(\mathsf{lch}(\Gamma, \underset{\sim}{B})) = (\Gamma, \underset{\sim}{B})$.

It is important to note that for every derivation in $\mathcal{D}(\mathcal{G}')$, either lh^{-1} or lch^{-1} is defined. To see why, consider all the kinds of rules that could be at the root of an arbitrary derivation d' in $\mathcal{D}(\mathcal{G}')$. If the root of d' is a Leaves or Fill rule, $\mathsf{lh}^{-1}(d)$ is defined in cases (i) and (ii). If the root of d' is an Empty rule or a Concat rule which did not come from an LIG Push rule, $\mathsf{lch}^{-1}(d)$ is defined in cases (iii) and (iv). The only time a Compose rule appears in an HG derivation is when it is dominated by a Concat-from-Push rule, which is defined in (v).

Thus, every $d' \in \mathcal{D}(\textsc{lh}(\mathcal{G}))$ can be mapped to a derivation in $\mathcal{D}(\mathcal{G})$. Since $\mathsf{lh}^{-1*}(\mathcal{D}(\textsc{lh}(\mathcal{G}))) \subseteq \mathcal{D}(\mathcal{G})$, this proves that $\mathcal{D}(\textsc{lh}(\mathcal{G})) \subseteq \mathsf{lh}^*(\mathcal{D}(\mathcal{G}))$. And since $\mathcal{D}(\mathcal{G})$ and $\mathcal{D}(\textsc{lh}(\mathcal{G}))$ are bijective and the bijection lh preserves yields, each derivation set contains the same number of derivations per string generated. This completes the proof of Theorem 2, that $\textsc{lh}$ constructs derivationally equivalent HGs from LIGs.

4 Derivational equivalence: HG to LIG

In this section, I will prove the reverse of the previous section: that for every HG, there is a derivationally equivalent LIG. The proof will proceed with the same structure.

First, given a source HG $\mathcal{G}$, I will define the resulting LIG $\textsc{hl}(\mathcal{G}) = \mathcal{G}'$. Next, I define a function $\textbf{\textsf{hl}}_\alpha : \mathcal{D}_{\text{HG}} \to \mathcal{D}_{\text{LIG}}$ which transforms a well-formed HG derivation into a well-formed LIG derivation while preserving its yield and category.

To show that $\mathcal{G}$ and $\mathcal{G}'$ are derivationally equivalent, I show that the set of well-formed derivations of $\mathcal{G}$ stand in correspondence to the derivations of $\mathcal{G}'$ via $\textbf{\textsf{hl}}$. I will show that the diagram below commutes, that is, $\textbf{\textsf{hl}}^*_\alpha(\mathcal{D}(\mathcal{G})) = \mathcal{D}(\textsc{hl}(\mathcal{G})) = \mathcal{D}(\mathcal{G}')$.

$$\begin{tikzcd}
\text{HG} \arrow[r, "\textsc{hl}"] \arrow[d, "\mathcal{D}"] & \text{LIG} \arrow[d, "\mathcal{D}"] \\
\mathcal{P}(\mathcal{D}_{\text{HG}}) \arrow[r, "\textbf{\textsf{hl}}^*_\alpha"] & \mathcal{P}(\mathcal{D}_{\text{LIG}})
\end{tikzcd}$$

The proof proceeds by showing $\textbf{\textsf{hl}}^*_\alpha(\mathcal{D}(\mathcal{G})) \subseteq \mathcal{D}(\textsc{hl}(\mathcal{G}))$, and then the reverse. I show that $\textbf{\textsf{hl}}_\alpha$ has an inverse $\textbf{\textsf{hl}}^{-1}_{-\alpha}$ and is thus bijective, so there is a one-to-one correspondence between HG and LIG derivations.

4.1 Grammar translation

Theorem 5. For any HG $\mathcal{G}$, there is an LIG $\mathcal{G}' = \textsc{hl}(\mathcal{G})$ s.t. $\mathcal{G}$ and $\mathcal{G}'$ are derivationally equivalent.

Again, this theorem will be proven through this section. I first present the construction of the equivalent LIG. Given an HG $\mathcal{G} = (V_{\text{N}}, V_{\text{T}}, S, P)$, construct LIG $\mathcal{G}' = \textsc{hl}(\mathcal{G}) = (V'_{\text{N}}, V_{\text{T}}, V_{\text{N}}, S, P')$ such that:

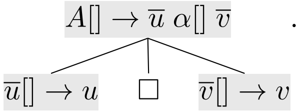
- $V'_{\text{N}} = V_{\text{N}} \sqcup \{\alpha\} \sqcup \overline{V_{\text{T}}}$, where $\overline{V_{\text{T}}} = \{\overline{u} \,|\, u \in V_{\text{T}}\}$
- For each rule in P of the form $A \xrightarrow{C_c} A_L\, A_c\, A_R$, add to P' the LIG rule $A[] \to A_L\, A_c[]\, A_R$.
- For each rule in P: $A \xrightarrow{W} B\, D$, add to P': $A[] \to B[D]$ and $\alpha[D] \to D[]$.
- For each rule in P: $A \to u_\uparrow v$, add to P': $A[] \to \overline{u}\; \alpha[]\; \overline{v}$, $\overline{u}[] \to u$, and $\overline{v}[] \to v$.
- Add to P': $\alpha[] \to \epsilon$. ALPHA

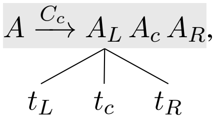
The constructed LIG is similar to the original HG, but there are several new nonterminals: $\overline{V_{\text{T}}}$, which are solely to introduce the corresponding terminals; and α, which serves as a gap in the LIG tree structure, allowing the LIG to emulate the bipartite nature of HG strings.

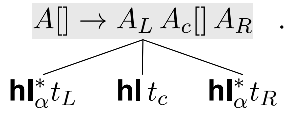
The indices of the LIG are just the nonterminals of the HG. Intuitively, if an HG rule wraps B around D, the corresponding LIG rule will push D on the stack, to construct a D subtree inside the B subtree at the bottom of its spine. It is in this way that the constructed LIG can use the stack to emulate the HG Wrap rules.

4.2 The hl translation function

$\textbf{\textsf{hl}} : \mathcal{D}_{\text{HG}} \to \mathcal{C}_{\text{LIG}}$ converts HG trees into LIG contexts. $\textbf{\textsf{hl}}_\alpha$ fills the context resulting from $\textbf{\textsf{hl}}$ with the rule $\alpha[] \to \epsilon$ to produce a well-formed LIG derivation which is equivalent to the HG derivation.

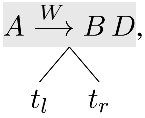
- $\textbf{\textsf{hl}}(A \to u_\uparrow v) =$ .

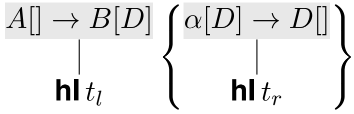
- If $d =$ , then

  $\textbf{\textsf{hl}}(d) =$ 

$A[] \to A_L\, A_c[]\, A_R$
$\textbf{\textsf{hl}}^*_\alpha t_L$
$\textbf{\textsf{hl}}\, t_c$
$\textbf{\textsf{hl}}^*_\alpha t_R$

 .

- If $d =$ , then

  $\textbf{\textsf{hl}}(d) =$ .

- $\textbf{\textsf{hl}}_\alpha(d) = \textbf{\textsf{hl}}(d)\{\alpha[] \to \epsilon\}$

The $\textbf{\textsf{hl}}$ translation function preserves categories and well-formedness. First, I introduce a lemma about context composition.

Lemma 6. Composition lemma for categories Let $\Gamma, \Delta \in \mathcal{C}(\text{LIG})$. Let $\sigma_1, \sigma_2 \in V_I^*$. If $\textbf{\textsf{catc}}(\Gamma, B) = A[\sigma_1]$ and $\textbf{\textsf{catc}}(\Delta, D) = B[\sigma_2]$, then $\textbf{\textsf{catc}}(\Gamma\{\Delta\}, D) = A[\sigma_1\sigma_2]$.

Proof. Induction over the structure of Γ. □

Theorem 7. $\textbf{\textsf{hl}}$ preserves categories
If $d \in \mathcal{D}_{\text{HG}}$, then $\textbf{\textsf{hl}}_\alpha(d)$ is a well-formed LIG derivation. In addition, $\textbf{\textsf{hl}}$ preserves categories to the extent that $\textbf{\textsf{catc}}(\textbf{\textsf{hl}}(d), \alpha) = \textbf{\textsf{cat}}(d)[]$.

Proof. Induction will proceed on the statement $\mathbf{catc}(\mathbf{hl}(d), \alpha) = \mathbf{cat}(d)[]$.
Base case:
$d = A \to u_{\uparrow}v$. $\mathbf{cat}(d) = A$.
$\mathbf{cat}(\mathbf{hl}(d), \alpha) = A[]$.
Inductive cases:
(i) If $d = \begin{array}{c} A \xrightarrow{C_c} A_L\,A_c\,A_R \\ t_L \quad t_c \quad t_R \end{array}$ and $\mathbf{cat}(d) = A$, then $\mathbf{cat}(t_{L/c/R}) = A_{L/c/R}$. By induction, $\mathbf{catc}(\mathbf{hl}(t_{L/c/R}), \alpha) = \mathbf{cat}(t_{L/c/R})[] = A_{L/c/R}[]$. So, $\mathbf{catc}(\mathbf{hl}(d), \alpha) = A[]$.
(ii) If $d = \begin{array}{c} A \xrightarrow{W} B\,D \\ t_l \quad t_r \end{array}$, and $\mathbf{cat}(d) = A$, then $\mathbf{cat}(t_l) = B$ and $\mathbf{cat}(t_r) = D$. By induction, $\mathbf{catc}(\mathbf{hl}(t_l), \alpha) = \mathbf{cat}(t_l)[] = B[]$, and $\mathbf{catc}(\mathbf{hl}(t_r), \alpha) = \mathbf{cat}(t_r)[] = D[]$.

Let $\Gamma = \mathbf{hl}(t_l)$ and $\Delta = \begin{array}{c} \alpha[D] \to D[] \\ | \\ \mathbf{hl}\,t_r \end{array}$, so $\mathbf{catc}(\Delta, \alpha) = \alpha[D]$ and $\mathbf{hl}(d) = \begin{array}{c} A[] \to B[D] \\ | \\ \Gamma\{\Delta\} \end{array}$.

Applying Lemma 6 where $\mathbf{catc}(\Gamma, \alpha) = B[]$ and $\mathbf{cat}(\Delta, \alpha[]) = \alpha[D]$ yields $\mathbf{catc}(\Gamma\{\Delta\}, \alpha) = B[D]$. Thus, $\mathbf{catc}(\mathbf{hl}(d), \alpha[]) = A[]$. □

Corollary 8. If d is a complete well-formed derivation of HG $\mathcal{G}$, then $\mathbf{hl}_{\alpha}(d)$ is a complete well-formed derivation of LIG $\mathcal{G}' = \textsc{hl}(\mathcal{G})$. This shows $\mathbf{hl}^{*}_{\alpha}(\mathcal{D}(\mathcal{G})) \subseteq \mathcal{D}(\textsc{hl}(\mathcal{G}))$.

Proof. This follows fairly straightforwardly from the previous theorem. Given any complete well-formed derivation $d \in \mathcal{D}(\mathcal{G})$, we can easily inspect that each rule used in $\mathbf{hl}_{\alpha}(d)$ is a rule of $P_{\mathcal{G}'}$. Furthermore, $\mathcal{G}$ and $\mathcal{G}'$ share the same start symbols S, so $\mathbf{hl}_{\alpha}(d)$ must be complete and well-formed with respect to $\mathcal{G}'$. □

Again, the proof of yield preservation is not difficult, and would follow the same structure as Theorem 7. It also follows as a corollary of Theorem 12.

Example 2. Figure 2 shows an example of an HG $\mathcal{G}_2$ which generates $\{ww \mid w \in \{a, b\}^{*}\}$, and the equivalent LIG $\mathcal{G}'_2$ resulting from HL. The figure also shows a $\mathcal{G}_2$ derivation for *abbabb*, and the $\mathcal{G}'_2$ derivation resulting from **hl**.
$\mathcal{G}_2 =$ ({S,T,U,A,B}, {a,b}, {S},P_2), where $P_2 =$ {S $\xrightarrow{C_2}$ A T, T $\xrightarrow{W}$ S A, S $\xrightarrow{C_2}$ B U, U $\xrightarrow{W}$ S B, A $\to \epsilon_{\uparrow}$a, B $\to \epsilon_{\uparrow}$b}

4.3 Inverting hl

In this subsection, I complete the proof of Theorem 5 by proving that $\mathcal{D}(\textsc{hl}(\mathcal{G})) \subseteq \mathbf{hl}^{*}_{\alpha}(\mathcal{D}(\mathcal{G}))$. To do that, I will show that $\mathbf{hl}_{\alpha}$ has an inverse, $\mathbf{hl}^{-1}_{-\alpha}$, and use it to show that $\mathbf{hl}^{-1*}_{-\alpha}(\mathcal{D}(\textsc{hl}(\mathcal{G}))) \subseteq \mathcal{D}(\mathcal{G})$.

Since the last step of $\mathbf{hl}_{\alpha}$ is to add $\alpha[] \to \epsilon$, the first step of $\mathbf{hl}^{-1}_{-\alpha}$ is to remove it: $\mathbf{hl}^{-1}_{-\alpha}(d') = \mathbf{hl}^{-1}(d'\{\alpha[] \to \epsilon \mapsto \square\})$. Then, $\mathbf{hl}^{-1}$ is the inverse of **hl**. $\mathbf{hl}^{-1}$ can be spelled out as:

(i) If $d' = \begin{array}{c} A[] \to \overline{u}\,\alpha[]\,\overline{v} \\ \overline{u}[] \to u \quad \square \quad \overline{v}[] \to v \end{array}$, then $\mathbf{hl}^{-1}(d') = A \to u_{\uparrow}v$.

(ii) If $d' = \begin{array}{c} A[] \to A_L\,A_c[]\,A_R \\ (\gamma_L, \alpha[] \to \epsilon) \quad \gamma_c \quad (\gamma_R, \alpha[] \to \epsilon) \end{array}$, then $\mathbf{hl}^{-1}(d') = \begin{array}{c} A \xrightarrow{C_c} A_L\,A_c\,A_R \\ \mathbf{hl}^{-1*}\gamma_L \quad \mathbf{hl}^{-1}\gamma_c \quad \mathbf{hl}^{-1*}\gamma_R \end{array}$.

(iii) If $d' = \begin{array}{c} A[] \to B[D] \\ | \\ \gamma_l \end{array} \left\{ \begin{array}{c} \alpha[D] \to D[] \\ | \\ \gamma_r \end{array} \right\}$, then $\mathbf{hl}^{-1}(d') = \begin{array}{c} A \xrightarrow{W} B\,D \\ \mathbf{hl}^{-1}\gamma_l \quad \mathbf{hl}^{-1}\gamma_r \end{array}$. That is, d' is a LIG context whose root node is a Push D rule; this means somewhere along the spine must be the corresponding Pop D rule because $\mathbf{flagseq}(d')$ is a Dyck word. The corresponding Pop rule is found in the same way as in the **lh** translation function.

It is not difficult to verify that $\mathbf{hl} \circ \mathbf{hl}^{-1} = \mathbf{id}$ and $\mathbf{hl}^{-1} \circ \mathbf{hl} = \mathbf{id}$; I show case (iii) below:

$$\mathbf{hl}(\mathbf{hl}^{-1}(d')) = \begin{array}{c} A[] \to B[D] \\ | \\ \mathbf{hl}(\mathbf{hl}^{-1}\gamma_l) \end{array} \left\{ \begin{array}{c} \alpha[D] \to D[] \\ | \\ \mathbf{hl}(\mathbf{hl}^{-1}\gamma_r) \end{array} \right\}.$$

The **flagseq** of γ_l and γ_r must be Dyck words as well, and γ_l and γ_r have an α-shaped gap, meaning $\mathbf{hl}^{-1}$ is defined over them, and by induction, $\mathbf{hl}(\mathbf{hl}^{-1}\gamma_{l/r}) = \gamma_{l/r}$

For $\mathbf{hl}^{-1} \circ \mathbf{hl}$: If $d = \begin{array}{c} A \xrightarrow{W} B\,D \\ t_l \quad t_r \end{array}$, then $\mathbf{hl}(d) = \begin{array}{c} A[] \to B[D] \\ | \\ \mathbf{hl}\,t_l \end{array} \left\{ \begin{array}{c} \alpha[D] \to D[] \\ | \\ \mathbf{hl}\,t_r \end{array} \right\}$. The intuition here is that since t_l is a well-formed HG derivation, the

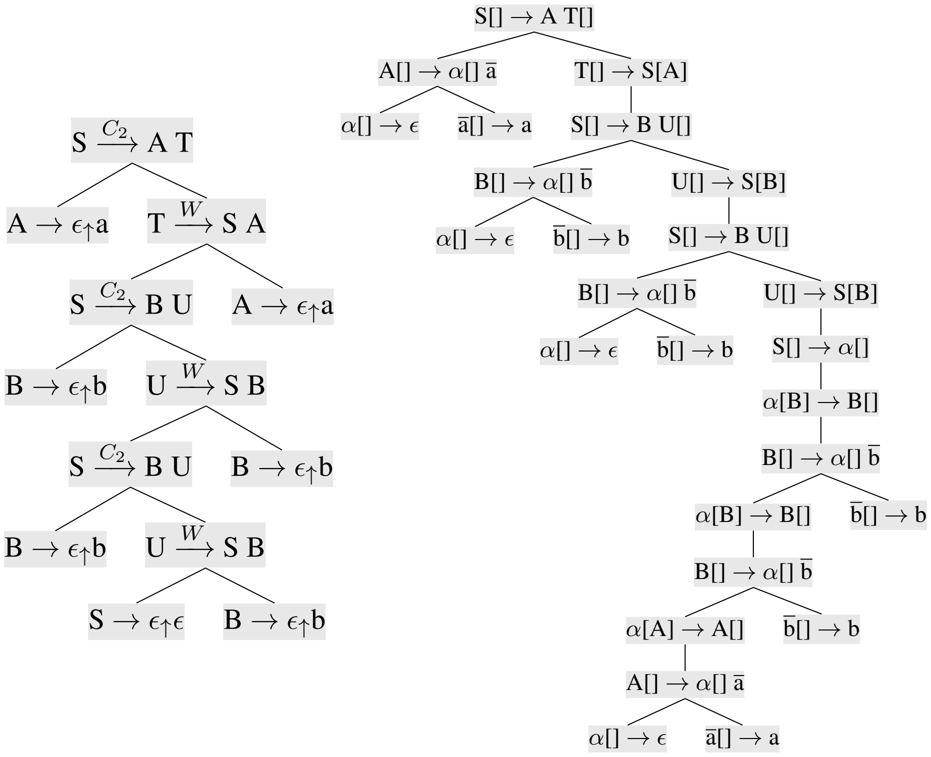

$S \xrightarrow{C_2} A\ T$
$A \to \epsilon_\uparrow a$
$T \xrightarrow{W} S\ A$
$S \xrightarrow{C_2} B\ U$
$A \to \epsilon_\uparrow a$
$B \to \epsilon_\uparrow b$
$U \xrightarrow{W} S\ B$
$S \xrightarrow{C_2} B\ U$
$B \to \epsilon_\uparrow b$
$B \to \epsilon_\uparrow b$
$U \xrightarrow{W} S\ B$
$S \to \epsilon_\uparrow \epsilon$
$B \to \epsilon_\uparrow b$
$S[] \to A\ T[]$
$A[] \to \alpha[]\ \bar{a}$
$T[] \to S[A]$
$\alpha[] \to \epsilon$
$\bar{a}[] \to a$
$S[] \to B\ U[]$
$B[] \to \alpha[]\ \bar{b}$
$U[] \to S[B]$
$\alpha[] \to \epsilon$
$\bar{b}[] \to b$
$S[] \to B\ U[]$
$B[] \to \alpha[]\ \bar{b}$
$U[] \to S[B]$
$\alpha[] \to \epsilon$
$\bar{b}[] \to b$
$S[] \to \alpha[]$
$\alpha[B] \to B[]$
$B[] \to \alpha[]\ \bar{b}$
$\alpha[B] \to B[]$
$\bar{b}[] \to b$
$B[] \to \alpha[]\ \bar{b}$
$\alpha[A] \to A[]$
$\bar{b}[] \to b$
$A[] \to \alpha[]\ \bar{a}$
$\alpha[] \to \epsilon$
$\bar{a}[] \to a$

Figure 2: Derivation of *abbabb* in HG $\mathcal{G}_2$ (left) and LIG $\textsc{hl}(\mathcal{G}_2)$ (right)

flagseq of $\mathbf{hl}(t_l)$ will be a Dyck word, so $\mathbf{hl}^{-1}$ will split $\mathbf{hl}(d)$ at the Pop D corresponding to the Push D at the root of $\mathbf{hl}(d)$. Thus, $\mathbf{hl}^{-1}(\mathbf{hl}\,d) =$

$$\begin{array}{c} A \xrightarrow{W} B\,D \\ \mathbf{hl}^{-1}\,\mathbf{hl}(t_l) \qquad \mathbf{hl}^{-1}(\mathbf{hl}\,t_r) \end{array} \qquad = d.$$

Combining the fact that $\mathbf{hl}$ and $\mathbf{hl}^{-1}$ are inverses with Theorem 7 results in the category preservation of $\mathbf{hl}^{-1}_{-\alpha}$: $\mathbf{cat}(d') = \mathbf{cat}(\mathbf{hl}^{-1}_{-\alpha} d')[]$. Given any (complete) well-formed derivation $d' \in \mathcal{D}(\textsc{hl}(\mathcal{G}))$, $\mathbf{hl}^{-1*}_{-\alpha}(d')$ will be a (complete) well-formed derivation in $\mathcal{D}(\mathcal{G})$. Therefore, $\mathcal{D}(\mathcal{G})$ and $\mathcal{D}(\textsc{hl}(\mathcal{G}))$ are bijective. Additionally, since $\mathbf{hl}_\alpha$ preserves yields, the number of derivations per string is the same in $\mathcal{G}$ and $\textsc{hl}(\mathcal{G})$. This completes the proof of Theorem 5, that $\textsc{hl}$ constructs derivationally equivalent LIGs from HGs.

5 Interpretational equivalence

Let us now consider the ways grammars can match semantics to syntactic structures, following the approach taken by Miller (1999). An interpreted grammar consists of a grammar paired with a rule interpretation function: $(\mathcal{G}, \mu)$, where μ is a function from rules of $\mathcal{G}$ to functions over the interpretation domain M. The interpretation domain M can be a set of any kind of elements: lambda terms, strings, and even trees. Two interpreted grammars are *interpretationally equivalent* iff they generate the same set of string-meaning pairs using their respective interpretation functions. And two formalisms $\mathcal{F}_1$ and $\mathcal{F}_2$ are *interpretationally equivalent* iff for each interpreted grammar $\mathcal{G}_1 \in \mathcal{F}_1$ with interpretation function μ_1, there is some $(\mathcal{G}_2, \mu_2)$ (with $\mathcal{G}_2 \in \mathcal{F}_2$) that is interpretationally equivalent to $(\mathcal{G}_1, \mu_1)$, and vice versa.

Formally, for a grammar $\mathcal{G}$, let μ be a function which assigns interpretations to rules of $\mathcal{G}$: it maps each rank 0 rule to an element of M, and each rank n rule to an n-ary function $M^n \to M$. Then, let $[\![\cdot]\!]^\mu$ be the homomorphic extension of the rule interpretation function μ, defined over $\mathcal{D}(\mathcal{G})$: for β rank 0, $[\![\beta]\!]^\mu = \mu(\beta)$, and for m rank n,

$$\left[\!\!\left[\begin{array}{c} m \\ t_1 \quad \dots \quad t_n \end{array}\right]\!\!\right]^\mu = \mu(m)[\![t_1]\!]^\mu \dots [\![t_n]\!]^\mu.$$

Additionally, define the interpretation function $[\![\cdot]\!]^\mu$ over tree contexts in such a way that if $d = \Gamma\{\beta\}$, then $[\![d]\!]^\mu = [\![\Gamma]\!]^\mu(\mu(\beta))$.

- $[\![\square]\!]^\mu = \lambda x.\, x$
- If $\Gamma = \begin{array}{c} m \\ t_L \quad \gamma_c \quad t_R \end{array}$, then $[\![\Gamma]\!]^\mu = \lambda x.\, \mu(m)[\![t_L]\!]^\mu([\![\gamma_c]\!]^\mu(x))[\![t_R]\!]^\mu$.

Lemma 9. Composition lemma for interpretations

The interpretation of the composition of two tree contexts Γ, Δ is function composition: $[\![\Gamma\{\Delta\}]\!]^{\mu} = \lambda x.\, [\![\Gamma]\!]^{\mu}([\![\Delta]\!]^{\mu} x)$

Proof. Induction over the structure of Γ. □

The proofs to show interpretational equivalence between LIG and HG will proceed as follows: for any LIG $\mathcal{G}$ equipped with an arbitrary rule interpretation function μ, I will show how to construct a rule interpretation function μ' for the HG $\textsc{lh}(\mathcal{G})$, such that for all $d \in \mathcal{D}(\mathcal{G})$, $[\![d]\!]^{\mu} = [\![\mathbf{lh}(d)]\!]^{\mu'}$. In essence, in addition to translating grammars to grammars with LH, and derivations to derivations with **lh**, we can also translate interpretation functions to interpretation functions. I will also show the same for the reverse translation: how to construct an function to interpret derivations resulting from **hl**, in grammars resulting from HL.

5.1 LIG to HG

Theorem 10. lh preserves interpretations

Given an LIG $\mathcal{G}$ with interpretation function μ, we can construct an HG interpretation function μ' for $\mathcal{G}' = \textsc{lh}(\mathcal{G})$ such that for every derivation $d \in \mathcal{D}(\mathcal{G})$, $[\![d]\!]^{\mu} = [\![\mathbf{lh}(d)]\!]^{\mu'}$.

- $\mu'(B \to \epsilon_{\uparrow} w) = \mu(B[] \to w)$
- If $m = A[\zeta] \to A_L\, A_c[\eta]\, A_R$ and $m' = \frac{A\zeta}{\underset{\sim}{\underline{B}}} \xrightarrow{C_i} A_L\, \frac{A_c\eta}{B}\, A_R$, then $\mu'(m') = \lambda t_L t_c t_R x.\, \mu m t_L (t_c x) t_R$
- $\mu'(\frac{A\eta}{B} \xrightarrow{W} \frac{A\epsilon}{\underline{D}}\, \frac{D\eta}{\underline{B}}) = \lambda xyz.\, x(yz)$ COMP
- $\mu'(A \xrightarrow{W} \frac{A\epsilon}{\underline{B}}\, \underline{B}) = \lambda xy.\, xy$ FILL
- $\mu'(\frac{A\epsilon}{A} \to \epsilon_{\uparrow}\epsilon) = \lambda x.\, x$ EMPTY

Proof. We want to show that both **lh** and **lch** preserve interpretations. We prove this by induction on both of these statements:

- $\forall d \in \mathcal{D}(\mathcal{G}). [\![d]\!]^{\mu} = [\![\mathbf{lh}(d)]\!]^{\mu'}$
- $\forall \Gamma \in \mathcal{C}(\mathcal{G}), B \in \underset{\sim}{V_N}. [\![\Gamma]\!]^{\mu} = [\![\mathbf{lch}(\Gamma, B)]\!]^{\mu'}$ whenever $\mathbf{lch}(\Gamma, B)$ is defined

Base cases:

(i) If $d = B[] \to u$, then $[\![d]\!]^{\mu} = \mu(B[] \to u)$. $[\![\mathbf{lh}(d)]\!]^{\mu'} = [\![B \to \epsilon_{\uparrow} u]\!]^{\mu'} = \mu(B[] \to u)$.

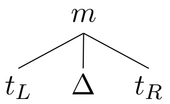
(ii) If $\Gamma = \square$, then $[\![\Gamma]\!]^{\mu} = \lambda x.\, x$. $[\![\mathbf{lch}(\Gamma, B)]\!]^{\mu'} = [\![\frac{B\epsilon}{B} \to \epsilon_{\uparrow}\epsilon]\!]^{\mu'} = \lambda x.\, x$

Induction:

(i) If $d = \Gamma\{\beta\}$, then $[\![d]\!]^{\mu} = [\![\Gamma]\!]^{\mu}(\mu(\beta))$. By the inductive hypothesis, for any B, $[\![\Gamma]\!]^{\mu} = [\![\mathbf{lch}(\Gamma, B)]\!]^{\mu'}$, where defined.

$$\begin{aligned}
&[\![\mathbf{lh}(d)]\!]^{\mu'} \\
&= \mu'(\textsc{fill})([\![\mathbf{lch}(\Gamma, B)]\!]^{\mu'})([\![\mathbf{lh}(\beta)]\!]^{\mu'}) \\
&= [\![\mathbf{lch}(\Gamma, B)]\!]^{\mu'}([\![\mathbf{lh}(\beta)]\!]^{\mu'}) \\
&= [\![\Gamma]\!]^{\mu}(\mu(\beta))
\end{aligned}$$

(ii) If $\Gamma =$



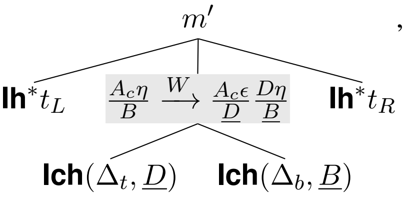
and $m = A[\zeta] \to A_L A_c[\eta] A_R$, then $[\![\Gamma]\!]^{\mu} = \lambda x.\, \mu(m)[\![t_L]\!]^{\mu}([\![\Delta]\!]^{\mu}(x))[\![t_R]\!]^{\mu}$.

By induction, $[\![t_{L/R}]\!]^{\mu} = [\![\mathbf{lh}(t_{L/R})]\!]^{\mu'}$ and $[\![\Delta]\!]^{\mu} = [\![\mathbf{lch}(\Delta, B)]\!]^{\mu'}$ for any B s.t $\mathbf{lch}(\Delta, B)$ is defined. There are two kinds of Γ, depending on the root node:

(a) If m is not a Push rule ($\eta = \epsilon$), then

$$\begin{aligned}
&[\![\mathbf{lch}(\Gamma, B)]\!]^{\mu'} \\
&= \mu'(\mathbf{lh}(m))[\![\mathbf{lh}(t_L)]\!]^{\mu'}[\![\mathbf{lch}(\Delta, B)]\!]^{\mu'}[\![\mathbf{lh}(t_R)]\!]^{\mu'} \\
&= (\lambda t_L t_c t_R x.\, \mu(m) t_L (t_c x) t_R)[\![t_L]\!]^{\mu}[\![\Delta]\!]^{\mu}[\![t_R]\!]^{\mu} \\
&= \lambda x.\, \mu(m)[\![t_L]\!]^{\mu}([\![\Delta]\!]^{\mu} x)[\![t_R]\!]^{\mu}
\end{aligned}$$

(b) If m is a Push rule ($\eta \neq \epsilon$), then let $m' = \frac{A\epsilon}{\underset{\sim}{\underline{B}}} \xrightarrow{C_i} A_L\, \frac{A_c\eta}{B}\, A_R$. Thus, $\mathbf{lch}(\Gamma, \underset{\sim}{\underline{B}}) =$


m'
$\mathbf{lh}^{*} t_L$
$\frac{A_c\eta}{B} \xrightarrow{W} \frac{A_c\epsilon}{\underline{D}}\, \frac{D\eta}{\underline{B}}$
$\mathbf{lh}^{*} t_R$
$\mathbf{lch}(\Delta_t, \underline{D})$
$\mathbf{lch}(\Delta_b, \underline{B})$

, where $\Delta_t\{\Delta_b\} = \Delta$.

$[\![\Delta]\!]^{\mu} = [\![\Delta_t\{\Delta_b\}]\!]^{\mu} = \lambda x.\, [\![\Delta_t]\!]^{\mu}([\![\Delta_b]\!]^{\mu} x)$ by Lemma 9.

By the IH, $[\![\Delta_t]\!]^{\mu} = [\![\mathbf{lch}(\Delta_t, \underline{D})]\!]^{\mu'}$ and $[\![\Delta_b]\!]^{\mu} = [\![\mathbf{lch}(\Delta_b, \underline{B})]\!]^{\mu'}$.

$$
\begin{aligned}
&\llbracket \mathbf{lch}(\Gamma, \underline{B})\rrbracket^{\mu'}\\
&= \mu'(m')\llbracket \mathbf{lh}(t_L)\rrbracket^{\mu'}(\mu'(\textsc{comp})\\
&\qquad \llbracket \mathbf{lch}(\Delta_t, \underline{D})\rrbracket^{\mu'}\llbracket \mathbf{lch}(\Delta_b, \underline{B})\rrbracket^{\mu'})\llbracket \mathbf{lh}(t_R)\rrbracket^{\mu'}\\
&= \mu'(m')\llbracket t_L\rrbracket^{\mu}(\mu'(\textsc{comp})\llbracket \Delta_t\rrbracket^{\mu}\llbracket \Delta_b\rrbracket^{\mu})\llbracket t_R\rrbracket^{\mu}\\
&= \mu'(m')\llbracket t_L\rrbracket^{\mu}[\lambda z.\, \llbracket \Delta_t\rrbracket^{\mu}(\llbracket \Delta_b\rrbracket^{\mu}(z))]\llbracket t_R\rrbracket^{\mu}\\
&= \mu'(m')\llbracket t_L\rrbracket^{\mu}\llbracket \Delta\rrbracket^{\mu}\llbracket t_R\rrbracket^{\mu}\\
&= \lambda x.\, \mu(m)\llbracket t_L\rrbracket^{\mu}(\llbracket \Delta\rrbracket^{\mu}x)\llbracket t_R\rrbracket^{\mu}
\end{aligned}
$$

So **lh** and **lch** both preserve the results of any arbitrary interpretation function μ via μ'. □

Corollary 11. Weak equivalence and preservation of yields by **lh** follows from interpretational equivalence and preservation of interpretations. Let $\mu = \mathbf{y}$, the natural yield function for LIGs, and $\mu' = \mathbf{y}$, the natural yield function for HGs, with a pair of strings being equivalent to a function from strings to strings: $u_{\uparrow}v = \lambda x.\, uxv$.

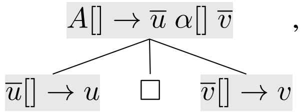
5.2 HG to LIG

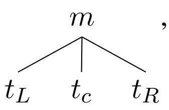
This subsection shows the reverse: how to preserve interpretations in the translation from HG to LIG. LIG interpretations in this section will involve lists, so here are a few preliminary definitions regarding lists: $[x]^{\downarrow} = x$ and $x{:}[y, \ldots] = [x, y, \ldots]$.

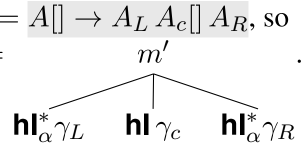
Theorem 12. **hl** preserves interpretations
Given an HG $\mathcal{G}$ with μ, we can construct an interpretation function μ' for $\textsc{hl}(\mathcal{G})$, such that $\forall d \in \mathcal{D}(\mathcal{G}).\, [\llbracket d\rrbracket^{\mu}] = \llbracket \mathbf{hl}(d)\rrbracket^{\mu'}[\,]$, i.e. $\llbracket d\rrbracket^{\mu} = (\llbracket \mathbf{hl}(d)\rrbracket^{\mu'}[\,])^{\downarrow}$. That is, the interpretation of $\mathbf{hl}(d)$ applied to the empty list is equal to the interpretation of d contained in a list. To define μ':

- $\mu'(\overline{u} \to u) = [\,]$
- $\mu'(A[] \to \overline{u}\ \alpha[]\ \overline{v}) = \lambda usv.(\mu(A \to u_{\uparrow}v)){:}s$
- If $m = A \xrightarrow{C_c} A_L\, A_c\, A_R$ and $m' = A[] \to A_L\, A_c[]\, A_R$, then $\mu'(m') = \lambda x(y{:}s)z.\, (\mu(m)(x^{\downarrow})y(z^{\downarrow})){:}s$.
- If $m = A \xrightarrow{W} B\, D$, then $\mu'(A[] \to B[D]) = \lambda(x{:}y{:}s).\, (\mu mxy){:}s$ and $\mu'(\alpha[D] \to D[]) = \lambda x.\, x$.
- $\mu'(\alpha[] \to \epsilon) = [\,]$ ALPHA

Proof. Note that since the image of **hl** is $\mathcal{C}$(LIG), the interpretation of contexts applies.
Base case:
$d = A \to u_{\uparrow}v$, so $\llbracket d\rrbracket^{\mu} = \mu(A \to u_{\uparrow}v)$.

$\mathbf{hl}(d) =$  , so

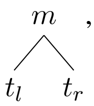
$\llbracket \mathbf{hl}(d)\rrbracket^{\mu'}[\,] = \mu'(A[] \to \overline{u}\ \alpha[]\ \overline{v})[\,] = (\lambda s.\, \mu(A \to u_{\uparrow}v){:}s)[\,] = [\mu(A \to u_{\uparrow}v)] = [\llbracket d\rrbracket^{\mu}]$.
Inductive cases:
(i) If $m = A \xrightarrow{C_c} A_L\, A_c\, A_R$ and $d =$  , then $\llbracket d\rrbracket^{\mu} = \mu(m)\llbracket t_L\rrbracket^{\mu}\llbracket t_c\rrbracket^{\mu}\llbracket t_R\rrbracket^{\mu}$.

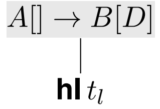
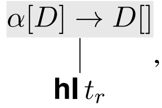
Let $m' = A[] \to A_L\, A_c[]\, A_R$, so
$\mathbf{hl}(d) =$  .

m'
$\mathbf{hl}^{*}_{\alpha}\gamma_L$
$\mathbf{hl}\,\gamma_c$
$\mathbf{hl}^{*}_{\alpha}\gamma_R$

And by the IH, $[\llbracket t_{L/c/R}\rrbracket^{\mu}] = \llbracket \mathbf{hl}\,\gamma_{L/c/R}\rrbracket^{\mu'}[\,]$.

$$
\begin{aligned}
&\llbracket li(d)\rrbracket^{\mu'}([\,])\\
&= (\lambda x.\, \mu'(m')\llbracket(\mathbf{hl}^{*}\gamma_L, \textsc{alpha})\rrbracket^{\mu'}\\
&\qquad (\llbracket \mathbf{hl}\,\gamma_c\rrbracket^{\mu'}x)\llbracket(\mathbf{hl}^{*}\gamma_R, \textsc{alpha})\rrbracket^{\mu'})[\,]\\
&= \mu'(m')(\llbracket \mathbf{hl}^{*}\gamma_L\rrbracket^{\mu'}[\,])(\llbracket \mathbf{hl}\,\gamma_c\rrbracket^{\mu'}[\,])(\llbracket \mathbf{hl}^{*}\gamma_R\rrbracket^{\mu'}[\,])\\
&= \mu'(m')[\llbracket t_L\rrbracket^{\mu}][\llbracket t_c\rrbracket^{\mu}][\llbracket t_R\rrbracket^{\mu}]\\
&= (\mu(m)([\llbracket t_L\rrbracket^{\mu}]^{\downarrow})(\llbracket t_c\rrbracket^{\mu})([\llbracket t_R\rrbracket^{\mu}]^{\downarrow})){:}[\,]\\
&= [\mu(m)\llbracket t_L\rrbracket^{\mu}\llbracket t_c\rrbracket^{\mu}\llbracket t_R\rrbracket^{\mu}]
\end{aligned}
$$

(ii) If $m = A \xrightarrow{W} B\, D$ and $d =$  , then

$\llbracket d\rrbracket^{\mu} = \mu(m)\llbracket t_l\rrbracket^{\mu}\llbracket t_r\rrbracket^{\mu}$.
Let $\Gamma =$  and $\Delta =$  ,

so $\mathbf{hl}(d) = \Gamma\{\Delta\}$. By induction, $[\llbracket t_{l/r}\rrbracket^{\mu}] = \llbracket \mathbf{hl}\,t_{l/r}\rrbracket^{\mu'}[\,]$. This entails $\llbracket \mathbf{hl}\,t_l\rrbracket^{\mu'} = \lambda x.\, \llbracket t_l\rrbracket^{\mu}{:}x$. $\llbracket \Delta\rrbracket^{\mu'} = \lambda y.\, (\lambda x.\, x)(\llbracket \mathbf{hl}\,t_r\rrbracket^{\mu'}y) = \llbracket \mathbf{hl}\,t_r\rrbracket^{\mu'}$. $\llbracket \Gamma\rrbracket^{\mu'} = \lambda z.\, (\lambda(x{:}y{:}s).\, (\mu mxy){:}s)(\llbracket \mathbf{hl}\,t_l\rrbracket^{\mu'}z)$.

$$
\begin{aligned}
&\llbracket \mathbf{hl}(d)\rrbracket^{\mu'}[\,] = \llbracket \Gamma\{\Delta\}\rrbracket^{\mu'}[]\\
&= \llbracket \Gamma\rrbracket^{\mu'}(\llbracket \Delta\rrbracket^{\mu'}[\,]) \text{ (by Lemma 9)}\\
&= (\lambda(x{:}y{:}s).\, (\mu mxy){:}s)(\llbracket \mathbf{hl}\,t_l\rrbracket^{\mu'}(\llbracket \mathbf{hl}\,t_r\rrbracket^{\mu'}[\,]))\\
&= (\lambda(x{:}y{:}s).\, (\mu mxy){:}s)((\lambda x.\, \llbracket t_l\rrbracket^{\mu}{:}x)[\llbracket t_r\rrbracket^{\mu}])\\
&= [\mu(m)\llbracket t_l\rrbracket^{\mu}\llbracket t_r\rrbracket^{\mu}]
\end{aligned}
$$

Finally, since $\mathbf{hl}_{\alpha}(d) = \mathbf{hl}(d)\{\alpha[] \to \epsilon\}$, $\llbracket \mathbf{hl}_{\alpha}(d)\rrbracket^{\mu'} = [\llbracket d\rrbracket^{\mu}]$. □

Again, weak equivalence and preservation of yields by $\mathbf{hl}_{\alpha}$ follows as an instance of μ. In addition, if we let μ be the identity function on $\mathcal{D}(\mathcal{G})$, μ' will emulate $\mathbf{hl}^{-1}_{-\alpha}$.

6 Discussion

The translation from LIG to HG presented in Vijay-Shanker and Weir (1994) formed the basis for this paper; because they only considered generated string languages, their translations did not preserve the number of derivations per string. Thus, to study equivalences deeper than of sets of strings, Section 3.1 presented a modified version of their grammar translation which is indeed derivation-count preserving.

However, it is worth noting that when paired with the semantic translation defined in Theorem 10, the original translation algorithm in Vijay-Shanker and Weir (1994) is also string-meaning preserving, despite not producing derivationally equivalent grammars. The HGs produced from their translation contain infinite families of spurious ambiguities, but the semantic translation in this paper is defined in such a way to overlook these instances of spurious ambiguity, treating them as semantically vacuous. This shows that derivational equivalence is not a prerequisite to interpretational equivalence. In general, derivational equivalence may become a less useful notion when working with infinitely ambiguous grammars, but interpretational equivalence does not.

Other researchers have proposed ways to compare generative capacity at a level beyond strings, such as Koller and Kuhlmann (2009) and Schiffer and Maletti (2021). Most notably, Kanazawa (2014) shows that the set of derivation trees of arboreal indexed grammars (with categories stripped of indices at each node) is equal to the set of trees derived by simple context-free tree grammars. In the base case, this equivalence roughly reduces down to the deindexed derivation trees of LIG being equal to the trees derived by Tree-Adjoining Grammars (TAG), the latter of which has essentially been proven strongly equivalent to HG in Fujiyoshi and Kasai (2000).

The goal of establishing notions of equivalences beyond string languages is to consider what kinds of properties each formalism is capable of ascribing to each string. If these properties, such as semantics, are computed compositionally from the derivations, then in comparing interpretations, we are implicitly comparing derivations which have that interpretation. In theory, it is not enough to establish that the derived trees of a tree grammar are in correspondence with the deindexed derivation trees of an index grammar, since there could be multiple derivations per derived tree in the former case, or multiple derivations which are identical modulo indices in the latter case. However, it is easy to see that given the semantic translation developed here, Kanazawa's result can be strengthened to interpretational equivalence.

7 Conclusion

In this paper, I explored two kinds of equivalence beyond weak equivalence and proved they both apply to LIG and HG. Two grammars are derivationally equivalent if they generate the same number of derivations per string, and two grammars are interpretationally equivalent if they generate the same set of string-interpretation pairs.

These proofs were facilitated by the functions lh and hl_α, which translated derivations from a grammar in one formalism to an equivalent grammar in the other formalism. It is important to note that these functions did not depend on the grammars themselves in any way. This means that they could be defined over $\mathcal{D}_{\text{LIG}}$ and $\mathcal{D}_{\text{HG}}$, showing that all LIG and HG derivations stand in a bijection.

This paper can serve as a model for how to rigorously compare recursive mechanisms across formalisms. The notions of derivational and interpretational equivalences shed light on how the generative structures of different grammar formalisms relate. In this case, the equivalence between LIG and HG reveals how working with a linear stack-based category system is equivalent on some fundamental level to the ability to wrap pairs of strings.

Acknowledgments

I would like to thank Dylan Bumford, Tim Hunter, Laurel Perkins, Robert Frank, and audiences at UCLA SynSem and MoL for invaluable feedback and discussion. This paper has also benefited from the comments of three anonymous reviewers.

References

Joan Bresnan, Ronald M. Kaplan, Stanley Peters, and Annie Zaenen. 1982. Cross-Serial Dependencies in Dutch. Linguistic Inquiry, 13(4):613–635. Publisher: MIT Press.

Robert Frank and Tim Hunter. 2021. Variation in mild context-sensitivity: Derivational state and structural monotonicity. Evolutionary Linguistic Theory, 3(2):181–214.

A. Fujiyoshi and T. Kasai. 2000. Spinal-Formed Context-Free Tree Grammars. Theory of Computing Systems, 33(1):59–83.

Gerald Gazdar. 1988. Applicability of indexed grammars to natural languages. In U. Reyle and C. Rohrer, editors, Natural Language Parsing and Linguistic Theories, pages 69–94. Springer Netherlands, Dordrecht.

Makoto Kanazawa. 2014. A Generalization of Linear Indexed Grammars Equivalent to Simple Context-Free Tree Grammars. In Formal Grammar, volume 8612, pages 86–103, Berlin, Heidelberg. Springer Berlin Heidelberg.

Alexander Koller and Marco Kuhlmann. 2009. Dependency trees and the strong generative capacity of CCG. In Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics, pages 460–468, Athens, Greece. Association for Computational Linguistics.

Philip H. Miller. 1999. Strong Generative Capacity: The Semantics of Linguistic Formalism. Number no. 103 in CSLI Lecture Notes. CSLI Publications, Stanford, CA.

Carl Jesse Pollard. 1984. Generalized phrase structure grammars, head grammars, and natural language. Ph.D. thesis, Stanford University, Stanford, CA.

Kelly Roach. 1987. Formal Properties of Head Grammars. In Alexis Manaster-Ramer, editor, Mathematics of Language: Proceedings of a conference held at the University of Michigan, Ann Arbor, October 1984, pages 293–348. John Benjamins Publishing Company.

Lena Katharina Schiffer and Andreas Maletti. 2021. Strong Equivalence of TAG and CCG. Transactions of the Association for Computational Linguistics, 9:707–720.

K. Vijay-Shanker and D. J. Weir. 1994. The equivalence of four extensions of context-free grammars. Mathematical Systems Theory, 27(6):511–546.

On the Logic of LGB Type Structures. Part II: Reentrancy Structures

Marcus Kracht
Hungarian Resarch Centre for Linguistics
Benzúr u. 33,
1068 Budapest VI., Hungary
marcus.kracht@uni-bielefeld.de

Abstract

It is shown that the dynamic logics corresponding to theories of generative grammar are decidable. The theorems establish fairly general decidability results for structures that use a restricted form of reentrancy. Our methods give effective algorithms and upper bounds for the complexity of the decision problem. Although the bounds are fairly high (sometimes in the order 2EXPTIME) it is hoped that the complexity can be reduced in most cases. This opens the way to create effective tools for testing grammars and theories, and for checking satisfiability of web queries for reentrant structures.

In Memoriam dr. András Domokos

1 Introduction

This paper is a continuation of (Kracht, 2008). Although the proofs are independent of those of the previous paper, we omit here the motivations that lead to the definitions of the structures whose logics we shall look at here.

A grammar can be seen as a theory of a class $\mathcal{K}$ of structures. A framework admits classes of grammars, and so classes of classes of structures. As I have shown in (Kracht, 2001), the structures used in generative grammar since the GB framework—thus including the theory of Principles and Parameters as well as the Minimalist Program—are more complex than trees. Their geometrical part consists of a relational structure of the form $\langle M, \prec\rangle$, where M is finite, $\prec^+$ is cycle free, and for all x, $\{y : x \prec y\}$ is linearly ordered by $\prec^+$. These are called **multidominance structures**. Everything else, starting from categories, adjunction and subcategorisation can be dealt with by adding suitable propositional constants or by considering lexical entries to be structurally complex (see (Kracht, 2008)). Adjunction complicates the matter insofar as it requires a more liberal notion of reentrancy. We shall deal at the end of the paper with adjunction by proving a very general theorem that admits structures that have any kind of relations added to trees. Thus, particular grammars determine classes of MDSs. Theories of grammar determine sets of grammars and thus—indirectly—sets of logics. Consider now a class of structures $\mathcal{K}$. We may now ask: what is the logic of these structures? In the literature one typically settles either on monadic second order logic (see (Rogers, 1994) and (Kolb et al., 2003)) or on modal logic. The disadvantage of MSO is that multidominance prohibits a straight application of Rabin's Theorem. So far it is—to my knowledge—not known, how Rabin's Theorem can be applied to MDSs, see (Kepser, 2010). But it is not necessary to use such strong logics, whose complexity anyhow is far beyond the practical needs. It turns out that everything that needs to be said at all can be said in a relatively weak logic, using five modal operators. The language allows to distinguish two different grammars and is therefore expressive enough.

We shall show that virtually all theories formulated within GB and MP are decidable. This is a big step forward. It technically means that one can decide in principle whether a principle follows from a conjunction of other principles, whether a grammar admits certain types of structures, whether a given string is grammatical given a particular grammar, and so on. It means that these questions can be settled once and for all in a formal way, and not simply by looking at examples. For all those who do not wish to perform such arguments themselves it may be said that since all methods are constructive they can be executed by a machine as well and so it is also possible to design work benches for syntacticians that allows to define a grammar or even a constraint on grammars and see how it interacts with other constraints, or whether it is satisfied in a grammar, and so on. It also allows to check queries on linguistic structures for consistency with a given theory, eliminating costly searches in the internet in case of unsatisfiability. [1]

2 Reentrancy Structures and Multidominance Structures

In the previous papers we studied the logic of MDSs. It turns out that MDSs can be coded in reentrancy structures. From a formal viewpoint, reentrancy structures are somewhat easier to deal with. A reentrancy structure uses two kinds of relations: the 'white' relations $\triangleright_i$, $i < m$, and the 'black' relations $\blacktriangleright_j$, $j < n$. (The use of $\triangleright$ in place of the more usual $\triangleleft$ is mnemonic: we remind ourselves that we are inside a tree and we are going down, whence following $\triangleright$ rather than $\triangleleft$.) On the basis of these relations we set

$$>:= \left(\bigcup_{i<m} \triangleright_i\right)^+ \tag{1}$$

Definition 1 *A **reentrancy structure** is a structure*

$$\mathfrak{M} = \langle M, \{\triangleright_i : i < m\}, \{\blacktriangleright_j : j < n\}\rangle \tag{2}$$

*where $\langle M, \{\triangleright_i : i < m\}\rangle$ is an m-branching tree with successor relations $\triangleleft_i$, $i < m$, and for every $j < n$, $\blacktriangleright_j$ is a partial function such that $\blacktriangleright_j \subseteq >$. $\mathfrak{M}$ is **narrow** if for all $j < n$, $\blacktriangleleft_j := \blacktriangleright_j^{\smile}$ also is a partial function.*

$$\triangleright := \bigcup_{i<m} \triangleright_i \tag{3}$$

We use $\triangledown_i$ and $\triangledown$ for the operators that use these relations. And we use $\blacktriangledown_j$ for the operator of the relation $\blacktriangleright_j$.

Before we dive into technicalities let us indicate how we code MDSs by means of narrow reentrancy structures. Recall from (Kracht, 2008) the following.

Definition 2 *A **multidominance structure** is a structure*

$$\langle M, \succ_{00}, \succ_{01}, \succ_{10}, \succ_{11}\rangle \tag{4}$$

such that

① *$\langle M, \succ_{00}, \succ_{01}\rangle$ is a tree with dominance relation $(\succ_{00} \cup \succ_{10})^+$.*

② *If there is a $y \prec_{1i} x$ for some $i \in \{0,1\}$ then there is a $y' \prec_{0j} x$ for some $j \in \{0,1\}$.*

③ *For $i \in \{0,1\}$: If $x \succ_{i1} y$ then x $(\succ_{00} \cup \succ_{10}$ $)^+$ y.*

④ *For $i \in \{0,1\}$: If $x \succ_{i0} y$ and $x \succ_{i1} y'$ then $y = y'$.*

⑤ *For $i \in \{0,1\}$: $\succ_{i0} \cap \succ_{i1} = \varnothing$.*

⑥ *For $i \in \{0,1\}$: $\succ_{0i} \cap \succ_{1i} = \varnothing$.*

*Let $M(x) := \{y : y \succ_{01} x$ or $y \succ_{11} x\}$. $\mathfrak{M}$ is called **narrow** if $|M(x)| \leq 1$ for every $x \in M$.*

It follows from the definitions that

$$(\succ_{00} \cup \succ_{01} \cup \succ_{10} \cup \succ_{11})^+ = (\succ_{00} \cup \succ_{01})^+ \tag{5}$$

Let $\mathfrak{M}$ be given. For all $i \in \{0,1\}$ we put

$$\triangleleft_i :=\succ_{0i} \tag{6}$$

By definition. $M(x)$ is linearly ordered by the tree order $<$. So, $M(x) = \{y_i : i < p\}$, where $y_k < y_{k+1}$ for all $k < p - 1$. Then put

$$\begin{aligned} y_i \blacktriangleleft_{0j} y_{i+1} &:\Leftrightarrow x \prec_{1j} y_i \text{ and } x \prec_{01} y_{i+1} \\ y_i \blacktriangleleft_{1j} y_{i+1} &:\Leftrightarrow x \prec_{1j} y_{i+1} \text{ and } x \prec_{11} y_{i+1} \end{aligned} \tag{7}$$

First of all, let us note the following.

Lemma 3 *$\blacktriangleright_{ij}$ and $\blacktriangleleft_{ij}$ are partial functions.*

Proof. Assume $x \blacktriangleleft_{00} y$ and $x \blacktriangleleft_{00} y'$. Then $x, y \in M(z)$ for some z, and $x, y' \in M(z')$ for some z'. Now, $z = z'$, otherwise $M(z) \cap M(z') = \varnothing$. To see this, observe that $x \in M(z) \cap M(z')$, which is to say that $x > z$ as well as $x > z'$. Moreover, by definition of $\blacktriangleleft_{00}$, $x \succ_{01} z$ and $x \succ_{01} z'$. This means $z = z'$. So, $y = y'$, since y and y' both are the next node up from x in the tree order. Similarly for the other relations $\blacktriangleleft_{ij}$. Now assume $x \blacktriangleright_{00} y$ and $x \blacktriangleright_{00} y'$. This again means $x, y \in M(z)$ for some z, $x, y' \in M(z')$ for some z', from which $z = z'$. Since $M(z)$ is linearly ordered by the tree order, $y = y'$, by definition of $\blacktriangleright_{00}$. Similarly for the other relations $\blacktriangleright_{ij}$. □

Put

$$\nu(\mathfrak{M}) := \langle M, \{\triangleright_i : i < 2\}, \{\blacktriangleright_{ij} : i, j < 2\}\rangle \tag{8}$$

It is not hard to see that $\blacktriangleright_{ij} \subseteq \triangleright^+$ for all $i, j < 2$. Hence we have

Lemma 4 *$\nu(\mathfrak{M})$ is a narrow reentrancy structure.*

[1] I thank Hans-Martin Gärtner for the help he has given to me. Thank to Ed Keenan, Greg Kobele, Ed Stabler and András Kornai.

We have to see how to recover $\mathfrak{M}$ from $\nu(\mathfrak{M})$. Put

$$\succ_{00} := \lhd_0 \qquad \succ_{01} := \lhd_1 \tag{9}$$

This defines the same tree order underlying the MDS. Put

$$\blacktriangleleft_i := \blacktriangleleft_{i0} \cup \blacktriangleleft_{i1} \tag{10}$$

We define now: $x \succ_{10} y$ iff there is a $j < 2$ and a sequence $x = z_0 \lhd_j z_1 \blacktriangleleft_{1j} z_2 \blacktriangleleft_{1j} z_3 ... \blacktriangleleft_{j0} z_n = y$, $n > 1$. $x \succ_{11} y$ iff there is a $j < 2$ and a sequence $x = z_0 \lhd_{1j} z_1 \blacktriangleleft_{1j} z_2 \blacktriangleleft_{1j} z_3 ... \blacktriangleleft_{j1} z_n = y$, $n > 1$. Given a reentrancy structure $\mathfrak{N}$ put

$$\mu(\mathfrak{N}) := \langle M, \succ_{01}, \succ_{10}, \succ_{10}, \succ_{11} \rangle \tag{11}$$

This defines an MDS, a fact that follows immediately from the following fact.

Lemma 5 *For every MDS:* $\mu(\nu(\mathfrak{M})) = \mathfrak{M}$.

Proof. The tree order is identical, so we need to care only about $\succ_{10}$ and $\succ_{11}$. Suppose that in $\mathfrak{M}$ $x \succ_{10} y$. Then $y \in M(x)$ and so we can enumerate $M(x)$ as z_i, $i < p + 1$, such that $x \succ_{10} z_1 \succ_{10}^{+} z_2 \succ_{10}^{+} z_3 ... \succ_{10}^{+} z_p$. Suppose that $x \succ_{00} z_0$. Then $x \lhd_0 z_0$. By definition of $\blacktriangleleft_{ij}$, we have $z_i \blacktriangleleft_{0i} z_{i+1}$ for all $i < p$. Moreover, we have $z_{p-1} \blacktriangleleft_{00} z_p$. This is exactly the definition of (the reconstructed version of) $\succ_{10}$ given above. Similarly for the other cases. □

For the next theorem we only need to remark that we can use the full power of dynamic logic.

Corollary 6 *The modal logic of reentrancy structures has the finite model property and is decidable.*

3 Axiomatisation

We work with polymodal logic. We shall now rehearse the details, and refer instead to standard sources. Let $Var := \{p_i : i \in \mathbb{N}\}$ be the set of variables. The set of constants is C. The boolean connectives are $\top$, $\neg$ and $\wedge$. For every white relation $\rhd_i$ we assume a modal operator $[\triangledown_i]$, and for every black relation $\blacktriangleright_j$ a modal operator $[\blacktriangledown_j]$. (In fact, $\triangledown_i$ and $\blacktriangledown_j$ are the programs in the sense of dynamic logic, and modalities are formed from them by using the brackets $[-]$ or $\langle - \rangle$.) Finally, there is a master modality $\Box$. We also write $\boxdot\varphi := \varphi \wedge \Box\varphi$. The first set of postulates regulates that the structure $\langle M, \{\rhd_i : i < m\}\rangle$ is acyclic.

(R1) $\Box(\Box p \to p) \to \Box p$.

(R2) For all $i < m$: $\langle \triangledown_i \rangle p \to \Diamond p$.

(R3) For all $i < m$: $\langle \triangledown_i \rangle p \to [\triangledown_i] p$.

The proof is not repeated here. Using unravelling one can show that the structures all derive from finite trees by collapsing certain subtrees.

The logic with the axioms (R1) – (R3) is the logic of acyclic structures. We call it $\mathbf{AC_m}$. If we expand the language into dynamic logic we get the logic $\mathbf{DPDL_m.f}$, also called the logic of finite deterministic computations on m programs. This logic was shown in (Kracht, 1999) to be complete with respect to finite m-branching trees. Although we are not dealing with a dynamic logic, it was proved in (Kracht, 2008), that the star can be added with impunity, since all programs are terminating in finite structures. Technically, therefore, although the language defined above contains no programs in the sense of PDL, we can add them in the form of abbreviations in the following way.

$$\begin{aligned} \langle \alpha \cup \beta \rangle \chi &:= \langle \alpha \rangle \chi \vee \langle \beta \rangle \chi \\ \langle \alpha ; \beta \rangle \chi &:= \langle \alpha \rangle \langle \beta \rangle \chi \\ \langle \delta ? \rangle \chi &:= \delta \wedge \chi \\ \langle \alpha^* \rangle \chi &:= \boxdot(q \leftrightarrow \chi \vee \langle \alpha \rangle \chi) \to q \end{aligned} \tag{12}$$

where in the last line q is a variable not occurring in χ or α and α is cycle free. The last reduction works in general for every program, since $R(\alpha)$ can be shown in **PDL.f** to be always of the form $R(\delta?) \cup R(\beta)$ for a cycle free β. To be a bit more precise, let us be given a formula φ in the language of **DPDL.f**. Then for every formula χ in the Fisher-Ladner closure we introduce a variable q_χ and replace φ by

$$\boxdot D(\varphi) \to q_\varphi \tag{13}$$

where $D(\varphi)$ is the conjunction of the following formulae:

$$\begin{aligned} & q_{\langle \alpha ; \beta \rangle \delta} \leftrightarrow q_{\langle \alpha \rangle \langle \beta \rangle \delta} \\ & q_{\langle \alpha \cup \beta \rangle \delta} \leftrightarrow q_{\langle \alpha \rangle \delta} \vee q_{\langle \beta \rangle \delta} \\ & q_{\langle \eta ? \rangle \delta} \leftrightarrow q_\eta \wedge q_\delta \\ & q_{\langle \alpha^* \rangle \delta} \leftrightarrow q_\delta \vee q_{\langle \alpha ; \alpha^* \rangle \delta} \\ & q_{\langle \triangledown_i \rangle \delta} \leftrightarrow \langle \triangledown_i \rangle q_\delta \\ & q_{\Diamond \delta} \leftrightarrow \Diamond q_\delta \end{aligned} \tag{14}$$

Moreover, as we have explained in (Kracht, 2008), it is possible to add the converse $\alpha^\smile$ with impunity provided that $R(\alpha^\smile)$ is a partial function and that we have a formula c equivalent to $[\alpha^\smile]\bot$. In order to replace $\alpha^\smile$, suppose the following holds at the root of the frame:

$$\boxdot(\chi \leftrightarrow (\chi \wedge c) \vee \langle \alpha \rangle q) \tag{15}$$

Then q is true at a node u iff $[\alpha^{\smile}]\chi$. Then $\langle\alpha^{\smile}\rangle\chi \leftrightarrow q \wedge \neg c$, by functionality of $\alpha^{\smile}$ and the definition of c. Alternatively, add to $D(\varphi)$ the following:

$$q_\delta \leftrightarrow q_\delta \wedge q_c \vee q_{\langle\alpha;\alpha^{\smile}\rangle\delta} \tag{16}$$

(This will require adding $q_{\langle\alpha;\alpha^{\smile}\rangle\delta}$ to the set of variables, and some more variables for the subformulae, whenever $\langle\alpha^{\smile}\rangle\delta$ is in the Fisher-Ladner closure.)

What we need, however, is the constant c. If, for example, we are interested in inverting the dominance relation, $\rhd$, we can do the following. If φ is consistent, so is $\varphi \wedge \Box\neg\varphi$. Thus, we can always assume that our model satisfies φ only at the root. In that case, the desired c is φ itself. It is likewise possible to invert $\succ_i$ for all $i < m$.

Thus, adding the converse does not increase expressibility as long as one can define c and the converse is a partial function. It follows that for every regular expression formed from programs which have this property, is also definable. However, one may not to use $*$ in programs that contain basic programs as well as their converses, since the resulting program may contain cycles.

The next batch of postulates concerns the axiomatisation of reentrancy structures.

(R4) For all $j < n$: $\langle\blacktriangledown_j\rangle p \to \Diamond p$.

(R5) For all $j < n$: $\langle\blacktriangledown_j\rangle p \to [\blacktriangledown_j]p$.

(R6) For all $j < n$: $\langle\blacktriangledown_j\rangle p \to \langle\triangledown^+\rangle p$.

(R7) For all $i < j < m$: $\langle\triangledown_i^{\smile}\rangle\top \to \neg\langle\triangledown_j^{\smile}\rangle\top$.

The axioms (R4) – (R5) are unproblematic, see the discussion above. They in fact give us the logic of finite computations for $m + n$ programs. (R6) adds the requirement that $\blacktriangleright_i$ is contained in the transitive closure of $\rhd$. (R7) finally makes the models trees. For it says that any node can have only one mother via $\lhd$ so that reentrancy is eliminated for the white relations. Notice that we have used the converse here; so although this postulate seems to use no variables, it abbreviates a formula that does.

Finally, we present an axiom for narrow reentrancy structures. Before we begin we draw attention to the fact that in trees one can effectively use nominals; these are variables which are true at exactly one point (see (Blackburn, 1993)). Set

$$\begin{aligned} n(p) := \neg \boxdot \neg p \wedge \boxdot(p \to [\triangledown^+]\neg p) \\ \wedge \boxdot \neg \textstyle\bigwedge_{i<j<m}(\langle\triangledown_i;\triangledown^*\rangle p \wedge \langle\triangledown_j;\triangledown^*\rangle p) \end{aligned} \tag{17}$$

Lemma 7 *Suppose $\mathfrak{M} = \langle M, \{\rhd_i : i < m\}\rangle$ is a tree with root w. Then $\langle\mathfrak{M}, \beta, w\rangle \vDash n(p)$ iff $\beta(p) = \{x\}$ for some $x \in M$.*

Proof. ($\Rightarrow$). First, $w \vDash \neg \boxdot \neg p$ guarantees that the set $\beta(p)$ is nonempty. Next, since $w \vDash \boxdot(p \to [\triangledown^+]\neg p)$ and w is the root, for every $x \in M$: $x \vDash p \to [\triangledown^+]\neg p$. This means that the set $\beta(p)$ is an antichain with respect to $\rhd^+$. For if $x \rhd^+ y$ and $x \vDash p$ then $y \vDash p$ cannot hold. Finally, this antichain has only one point. For if it contains two different points y and y' then there is a z and u, u' such that $y \lhd^* u \lhd_i z$ and $y' \lhd^* u' \lhd_j z$ for $i < j$. Thus $z \vDash \langle\triangledown_i;\triangledown^*\rangle p; \langle\triangledown_j;\triangledown^*\rangle p$, which is also excluded. ($\Leftarrow$). Basically similar. □

(R8) For all $j < n$: $n(p) \to \boxdot(\langle\blacktriangledown_j\rangle p \to [\triangledown^+;\blacktriangledown_j]\neg p)$

The logic **RS** the logic axiomatised over K_{m+n+1} by (R1) – (R7), while **NRS** is the logic obtained by adding to K_{m+m+1} the axioms (R1) – (R8).

Lemma 8 *A reentrancy structure is narrow iff it satisfies the postulate (R8).*

Proof. ($\Leftarrow$). Suppose $\mathfrak{M}$ is not narrow; say, $x \blacktriangleleft_{00} y, y'$ with $y \rhd^+ y'$. Define $\beta(p) := \{x\}$ and the above axiom is violated at y (and, as is not hard to see, also at the root). From Lemma 7 follows that at the root w:

$$\langle\mathfrak{M}, \beta, w\rangle \vDash n(p) \tag{18}$$

Nevertheless,

$$w \nvDash \boxdot(\langle\blacktriangledown_j\rangle p \to [\triangledown^+;\blacktriangledown_j]\neg p) \tag{19}$$

For we have $y \vDash \langle\blacktriangledown_j\rangle p$. However, $y \nvDash [\triangledown^+;\blacktriangledown_j]\neg p$ since $y \rhd^+ y'$ and $x \blacktriangleleft_j y'$ whence $y' \vDash \langle\blacktriangledown_j\rangle p$. ($\Rightarrow$). Now, conversely, assume that $\mathfrak{M}$ is narrow. Assume that β is such that

$$\langle\mathfrak{M}, \beta, w\rangle \vDash n(p) \tag{20}$$

Then by Lemma 7, $\beta(p) = \{x\}$ for some x. Let y be such that

$$y \vDash \langle\blacktriangledown_j\rangle p \tag{21}$$

Then $y \rhd^* x$, since p is only true at x. Let $y' \lhd^+ y$ be such that $y' \vDash \langle\blacktriangledown_j\rangle p$. Then $y' \rhd x$. However, also $y \rhd x \vDash p$. Since the structure is a narrow RS, $y = y'$. Contradiction. Hence, $y' \vDash \neg\langle\blacktriangledown_j\rangle p$, and since y' was arbitrary,

$$y \vDash [\triangledown^+;\blacktriangledown_j]\neg p \tag{22}$$

as promised. □

We are ultimately interested in the logic of the class of structures of the form $\nu(\mathfrak{M})$. It is obtained by adding to **NRS** a few postulates. Namely, notice first that the relations $\triangleright_0$, $\blacktriangleright_{00}$ and $\blacktriangleright_{10}$ are mutually exclusive. A node can only have one left daughter. Similarly for right hand daughters. And finally, if a node has a right hand daughter it also has a left hand daughter:

(R9) For all $i < 2$: $(\langle \triangledown_i \rangle \top \to \neg \langle \blacktriangledown_{0i} \top \wedge \neg \langle \blacktriangledown_{1i} \rangle \top) \wedge \langle \blacktriangledown_{0i} \rangle \top \to \neg \langle \triangledown_i \top \wedge \neg \langle \blacktriangledown_{1i} \rangle \top) \wedge \langle \blacktriangledown_{1i} \rangle \top \to \neg \langle \triangledown_i \top \wedge \neg \langle \blacktriangledown_{0i} \rangle \top)$

(R10) $(\langle \blacktriangledown_{01} \rangle \top \vee \langle \triangledown_1 \top \vee \langle \blacktriangledown_{11} \rangle \top) \to (\langle \blacktriangledown_{00} \rangle \top \vee \langle \triangledown_0 \top \vee \langle \blacktriangledown_{10} \rangle \top)$

It is clear that these axioms characterise the described properties; also, they are constant and therefore will be left out of consideration in the sequel, since constant axioms preserve decidability and also complexity.

In what follows we give a proof that both **RS** and **NRS** have the finite model property, and so characterise exactly the class of finite (narrow) RSs.

4 Some Basic Results on Complexity

Call a finite structure **linearisable** if there exists a linear irreflexive order $\ll$ computable in linear time such that (1) for every modal operator $\blacksquare$, $R(\blacksquare) \subseteq \ll$, and for every modal operator $\blacksquare$ either (2a) there is a number k such that $\blacksquare$ no point has more than k successors via $\blacksquare$, or (2b) there is a number k such that for every point x, $|\{y : x \ll y, \neg(x \; R(\blacksquare) \; y)\}| \leq k$.

Lemma 9 *Truth in a linearisable model is computable in linear time.*

Proof. Suppose that we have a well-order $\ll$ on the domain of the model such that if $x \; R(\blacksquare) \; y$ then $x \ll y$ for all basic modalities $\blacksquare$. Let $h(w) := |\{z : w \ll z\}|$. Assume that for all subformulae δ of φ and all $z \gg w$, truth at z is computed. Truth at w of a subformula δ is a matter of checking a bounded boolean combination of formulae. □

This can be applied to our logics as follows. Each node is given an address in the following way. The root has address ε. If $x \lhd_j y$ and y has address $\vec{c}$, then x has address $\vec{c} \cdot j$. By assumption, every node has a unique address. As order we take $\vec{c} \ll \vec{e}$ iff $\vec{c}$ is a prefix of $\vec{c}$ or there are $\vec{p}$, $\vec{q}$ and $\vec{r}$ such that $\vec{c} = \vec{p} \cdot 0 \cdot \vec{q}$ and $\vec{e} = \vec{p} \cdot 1 \cdot \vec{r}$. This takes care of the basic modalities. $\Box$ still is tricky. Notice, however, that for a node $\vec{c}$, $\vec{c} \vDash \Box\varphi$ iff $\vec{c} \cdot 0 \vDash \varphi; \Box\varphi$, $\vec{c} \cdot 1 \vDash \varphi; \Box\varphi$, so that truth at $\vec{c}$ is a bounded boolean combination of truth at some nodes later in the order.

The following is from (Vardi and Wolper, 1986).

Theorem 10 (Volper & Vardi) *Satisfiability in* **DPDL.f** *is globally (and locally) EXPTIME-complete.*

5 Preliminaries

Let $\mathfrak{M}$, β, φ be fixed. We assume that φ contains only basic modalities; everything else is just an abbreviation as defined above. Let $SF(\varphi)$ denote the set of subformulae of φ. For $A \subseteq SF(\varphi)$ put

$$a_{\mathfrak{M},\beta}(H) := \bigwedge_{\chi \in A} \chi \wedge \bigwedge_{\chi \in SF(\varphi)-A} \neg\chi \tag{23}$$

If $\mathfrak{M}$ and β are clear from the context, we drop them and write $a(H)$. Such formulae are called φ-**atoms**. Let $At(\varphi)$ denote the set of all φ-atoms. For a set Δ of formulae, the notation $SF(\Delta)$ and $At(\Delta)$ are used in the obvious way. For $w \in M$, let $a(w)$ denote the atom which is true at w.

Here is a general result on how to make new models from old ones. Fix a set Δ closed under taking subformulae. Let $\langle \mathfrak{M}, \beta \rangle$ be a model and $x, y \in M$. Write $x \sim_\Delta y$ if for all $\delta \in \Delta$: $\langle \mathfrak{M}, \beta, x \rangle \vDash \delta$ iff $\langle \mathfrak{M}, \beta, y \rangle \vDash \delta$.

Lemma 11 *Let $\langle M, \{\lhd_i : i < m\} \rangle$ be a frame, β a valuation. Now let $N \subseteq M$ and $\hat{\lhd}_i$ be relations such that the following holds for all i $< m$, $x \in M$ and $z \in N$:*

① *If $x \lhd_i y$ then there exists a $y' \sim_\Delta y$ such that $y' \in N$ and $x \hat{\lhd}_i y'$.*

② *If $z \hat{\lhd}_i y$ then there exists a $y' \sim_\Delta y$ such that $x \lhd_i y'$.*

Then for all $y \in N$ and $\delta \in \Delta$:

$$\begin{aligned} &\langle \langle M, \{\lhd_i : i < n\} \rangle, \beta, y \rangle \vDash \delta \\ &\quad \Leftrightarrow \quad \langle \langle N, \{\hat{\lhd}_i : i < n\} \rangle, \beta, y \rangle \vDash \delta \end{aligned} \tag{24}$$

Proof. By induction on the complexity of δ. If δ is a variable, the claim is trivial. The induction steps for $\neg$ and $\wedge$ are immediate. Now let $\delta = \Diamond_i \vartheta$. Then $\vartheta \in \Delta$, by assumption. ($\Rightarrow$). Assume that $\langle \mathfrak{M}, \beta, y \rangle \vDash \Diamond_i \vartheta$. Then there exists a $z \in M$ such that $y \lhd_i z$ and $\langle \mathfrak{M}, \beta, z \rangle \vDash \vartheta$. By assumption there is a $z' \sim_\Delta z$ such that $z' \in N$ and

$y\hat{\lhd}_i z'$. Hence $\langle \mathfrak{M}, \beta, z'\rangle \vDash \vartheta$. By inductive hypothesis, $\langle \mathfrak{N}, \beta, z'\rangle \vDash \vartheta$ and so $\langle \mathfrak{N}, \beta, y\rangle \vDash \Diamond_i \vartheta$. ($\Leftarrow$). Assume now that $\langle \mathfrak{N}, \beta, y\rangle \vDash \Diamond_i \vartheta$. Then there is a $z \in N$ such that $y\hat{\lhd}_i z$ and $\langle \mathfrak{N}, \beta, z\rangle \vDash \vartheta$. By assumption, there is a $z' \sim_\Delta z$ such that $y \lhd_i z'$. By inductive hypothesis, $\langle \mathfrak{M}, \beta, z\rangle \vDash \vartheta$ and so $\langle \mathfrak{M}, \beta, z'\rangle \vDash \vartheta$. From this we get $\langle \mathfrak{M}, \beta, y\rangle \vDash \Diamond_i \vartheta$. □

Call w **minimal** if $w \vDash a(w) \wedge [\nabla^+]\neg a(w)$. No two minimal points with same atom are comparable via $<$. Equivalently, if $a(w) = a(v)$ and v and w are both minimal, and $v \leq w$ then $w = v$. We shall show how to build a model on some set of minimal points. First, observe that we may choose the root of the model to be minimal. Let the **depth** of w be the defined by

$$dp(w) := |\{a(x) : x < w\}| \qquad (25)$$

Call w **egregious** iff either (a) $dp(w) = 0$ and w is the root and minimal, or (b) $dp(w) = n+1 > 0$ and for the unique x such that $w < x$ and $dp(x) = n$, and y such that $w \leq y \lhd x$ w is the leftmost minimal member of the set

$$\{z : z \leq x, a(z) = a(y)\} \qquad (26)$$

(Here leftmost is defined as follows: if $x \lhd_i y$ and $x' \lhd_j y$ then x is left of x' iff $i < j$. In general, x is left of x' iff there are u, u' and z such that $x \leq u \lhd_i z$ and $x' \leq u' \lhd_j u$ and $i < j$.)

The definition makes sure that for every egregious x which has a daughter, there is a unique egregious w with depth $dp(x)+1$ below that daughter. We denote by x^ε the unique egregious point u such that $u \leq x$ and $a(u) = a(x)$.

We shall prove a rather general theorem on the logic of finite trees based on m primitive relations and one master $\Box$.

Theorem 12 *For every* **RS***-model* $\langle \mathfrak{M}, \beta, x\rangle \vDash \varphi$ *where* $<$ *is a tree order, there is an* **RS***-model for* φ *such that* $<$ *is a tree order, and which has at most* 2^{2^n} *points.*

Proof. Assume that $\langle \mathfrak{M}, \beta, w\rangle \vDash \varphi$. Let $E := \{x^\varepsilon : x \in M\}$ be the set of egregious points of $\langle \mathfrak{M}, \beta\rangle$. For a relation R put $\hat{R} := \{\langle x, y^\varepsilon\rangle : \langle x, y\rangle \in R\}$. Put

$$\mathfrak{E} := \langle E, \{\hat{\rhd}_i \cap E^2 : i < m\}, \{\hat{\blacktriangleright}_j \cap E^2 : j < n\}\rangle \qquad (27)$$

It is not hard to see that the order $\hat{>}$ is a tree order. First we notice that for egregious points y and y': $y\hat{<}y'$ iff $y < y'$. This is shown by induction on the number of egregious points between y and y'. Suppose this number is zero. ($\Rightarrow$). We have $y\hat{\lhd}y'$. Hence $y = x^\varepsilon$ for some $x \lhd y'$. From this follows $y < y'$. ($\Leftarrow$). $y < y'$ and there is x such that $y = x^\varepsilon \lhd y'$ from which $y\hat{\lhd}y'$. Now suppose that this number is not zero. ($\Rightarrow$). $y\hat{\lhd}^+y'$ implies $y < y'$ from the previous and the fact that $<$ is transitive. ($\Leftarrow$). There is a chain of egregious points $y = y_0 < y_1 < y_2 < ... < y_n = y'$ such that there is no egregious point between y_i and y_{i+1}, $i < n-1$. By the previous, $y_i\hat{<}y_{i+1}$, and so $y\hat{<}y'$.

For assume that $y, y'\hat{>}x$. Then also $y, y' > x$, by the fact that $x^\varepsilon \leq x$ so that in general $y\hat{\rhd}_i z$ implies $y > z$. It follows that $y \leq y'$ or $y' \leq y$. From this we get $y\hat{\leq}y'$ or $y'\hat{\leq}y$.

The valuation is $\gamma(p) := \beta(p) \cap E$. From Lemma 11 we get for every $\chi \in SF(\varphi)$ that

$$\langle \mathfrak{E}, \gamma, x\rangle \vDash \chi \quad \Leftrightarrow \quad \langle \mathfrak{M}, \beta, x\rangle \vDash \chi \qquad (28)$$

It follows that

$$\langle \mathfrak{E}, \gamma, w^\varepsilon\rangle \vDash \varphi \qquad (29)$$

Thus the model is based on a tree. Given a formula of length n, there are n subformulae, whence 2^n atoms. The depth of a point is therefore bounded by 2^n. This is equal to the depth in $\mathfrak{E}$. It follows that since the models are based on binary branching trees of height at most 2^n there are at most 2^{2^n} points in E. □

This is the basis of all the proofs that we give in the sequel. The only addition they make is that there shall be additional relations to take care of.

6 The Main Theorem

We now formulate our main theorem.

Theorem 13 **NRS** *has the finite model property, and the size of models is bounded from above by* 2^{2^n}*, where* n *is the length of the formula.*

Proof. Let φ be given. We write $a(v)$ for the φ-atom of v. For each $\alpha \in At(\varphi)$ let r_α be a new variable and put $MVar := \{r_\alpha : \alpha \in At(\varphi)\}$. Next let Π:

$$\begin{array}{ll} & \bigwedge\langle \neg r_\alpha : \alpha \in At(\varphi)\rangle \\ \wedge & \bigwedge\langle \boxdot[\nabla^+]\neg r_\alpha : \alpha \in At(\varphi)\rangle \\ \wedge & \bigwedge\langle \boxdot[\blacktriangledown_j](\alpha \rightarrow r_\alpha) : \alpha \in At(\varphi), \\ & \quad j < n\rangle \\ \wedge & \bigwedge\langle \boxdot(r_\alpha \rightarrow \neg r_\beta) : \alpha \neq \beta \in At(\varphi)\rangle \end{array} \qquad (30)$$

First we verify that if $\langle \mathfrak{M}, \beta, w\rangle \vDash \varphi$, where $\beta : Var \rightarrow \wp(M)$ is a valuation and $<$ a tree order on $\mathfrak{M}$ then there is a unique valuation β^+ on $Var \cup$

$MVar$ extending β such that $\langle\mathfrak{M},\beta^+,w\rangle \vDash \varphi;\Pi$. The formulae say the following. (1) r_α is not true at the root. (2) r_α is not true at a node x where there is a y such that $x < y$. Hence, r_α can only be true if there is a y such that $x \blacktriangleleft_j y$. (3) says that in that case r_α is true iff α is true at x. Since $\mathfrak{M}$ is a tree, y is unique, and so the valuation is uniquely defined. (4) says that not both r_α and r_β can hold if $\alpha \neq \beta$. (It is actually a consequence of (1), (2) and (3).) So,

$$\beta(r_\alpha) = \{u : a(u) = \alpha \text{ and for no } v\text{: } u \lhd v\} \tag{31}$$

Suppose that φ is **DPDL.f**-consistent and φ contains no variable from $MVar$. Then $\varphi;\Pi$ is **DPDL.f**-consistent as well.

Define $X(\varphi)$

$$\begin{aligned}&\textstyle\bigwedge\langle n(r_\alpha) \to \boxdot(\bigwedge\langle\blacktriangledown_j\rangle r_\alpha \to [\nabla^+;\blacktriangledown_j]\neg r_\alpha)\\ &\qquad : \alpha \in At(\varphi), j < n\rangle\\ &\textstyle\wedge \bigwedge\langle\langle\blacktriangledown_j\rangle\pi \to \langle\nabla^+\rangle\pi : \pi \in At(\varphi), j < n\rangle\end{aligned} \tag{32}$$

Notice that $X(\varphi)$ is a set of instances of **NRS**-axioms.

Now suppose that φ is **NRS**-consistent. Hence $\varphi;X(\varphi)$ is **NRS**-consistent and a fortiori **DPDL.f**-consistent. Thus, $\varphi;\Pi;X(\varphi)$ is **DPDL.f**-consistent. Therefore there is a structure $\mathfrak{T} := \langle T,\{\rhd_i : i < m\},\{\blacktriangleright_j : j < n\}\rangle$ which is an $m+n$-branching tree, a valuation β and a world w such that

$$\langle\mathfrak{T},\beta,w\rangle \vDash \varphi;\Pi;X(\varphi) \tag{33}$$

and

❶ all $\lhd_i$, $i < m$, and all $\blacktriangleleft_j$, $j < n$, are partial functions;

❷ all $\rhd_i$, $i < m$, and all $\blacktriangleright_j$, $j < n$, are partial functions, and

❸ $\lhd$ is cycle free.

This is because **DPDL.f** has the finite model property and the fact that we can apply unravelling. We shall now produce a narrow reentrancy structure based on the egregious points.

Let H be the closure of w under the relation $>$. Define the function $u \mapsto u^\varepsilon$ as in the proof of the previous theorem. Then let $E := \{x^\varepsilon : x \in H\}$ be the set of egregious points and define $\hat{\rhd}_i$, $i < n$, as before. Now suppose that $x \blacktriangleright_j u$, $j < n$. Then $\langle\mathfrak{T},\beta,x\rangle \vDash \langle\blacktriangledown_j\rangle a(u)$. So, by choice of $X(\varphi)$, we have $\langle\mathfrak{T},\beta,x\rangle \vDash \langle\nabla^+\rangle a(u)$. So we choose an egregious $u^\heartsuit$ such that $u^\heartsuit < x$ and $a(u^\heartsuit) = a(u)$. Then $u^\heartsuit \in E$. We keep the partial function $u \mapsto u^\heartsuit$ fixed now. (In fact, a single such function suffices.)

$$\mathfrak{E} := \langle E,\{\hat{\rhd}_i : i < m\},\{\hat{\blacktriangleright}_j : j < n\}\rangle \tag{34}$$

The valuation is $\gamma(p) := \beta^+(p) \cap E$. For $\chi \in SF(\varphi)$ we get by Lemma 11 that for every egregious x:

$$\langle\mathfrak{E},\gamma,x\rangle \vDash \chi \quad\Leftrightarrow\quad \langle\mathfrak{T},\beta^+,x\rangle \vDash \chi \tag{35}$$

Thus we have a model $\langle\mathfrak{E},\gamma,w^\varepsilon\rangle \vDash \varphi$. Finally, we need to see that $\mathfrak{E}$ is a narrow reentrancy structure. Recall the function $u \mapsto u^\heartsuit$. By definition, $u^\heartsuit < x$ if $u \lhd x$, and from $u^\heartsuit < x$ it follows that $u^\heartsuit \hat{<} x$. Thus, $\mathfrak{E}$ is a reentrancy structure. To show that it is narrow we need to show that the map $u \mapsto u^\heartsuit$ is injective. Let therefore u and v be distinct egregious points such that $u^\heartsuit = v^\heartsuit$. Let $u \blacktriangleleft_j x$ and $v \blacktriangleleft_j y$. Then, since $u^\heartsuit < x$ and $v^\heartsuit < y$ we have $v^\heartsuit < x, y$ and so $y \leq x$ or $x \leq y$. Without loss of generality, assume the first. Then either $x = y$, and we are done; or $y < x$. We shall derive a contradiction. Notice that $a(u) = a(v)$, whence u and v both satisfy the same r_α in $\mathfrak{T}$. Since $\langle\mathfrak{T},\beta,w\rangle \vDash n(r_\alpha)$ and $\langle\mathfrak{T},\beta^+,x\rangle \vDash \langle\blacktriangledown_j\rangle r_\alpha$, we find that $\langle\mathfrak{T},\beta^+,x\rangle \vDash [\nabla^+;\blacktriangledown_j]\neg r_\alpha$, in particular $\langle\mathfrak{T},\beta^+,y\rangle \vDash \langle\blacktriangledown_j\rangle\neg r_\alpha$. This is the desired contradiction. □

Theorem 14 *Satisfiability in* **NRS** *is decidable in 2EXPTIME.*

Proof. Let n be the length of the formula. $SF(\varphi)$ is linear in n, and so there are 2^n many atoms. Observe next that the auxiliary formulae are of combined length $O(2^n)$. This is because there are 2^n atoms, and there are 2^{n^2} formulae of the form $r_\beta \to \neg r_\alpha$ and $c2^n$ formulae of the remaining kinds. Notice, though, that the formulae of the first kind are redundant. So, we really only need $c2^n$ many formulae. Each formula has length at most dn for some d. This, combined with the fact that the logic of trees is EXPTIME, gives the result. □

7 Further Results: Distance Principles

In (Kracht, 2008) I have considered variants of the definition of MDSs where lengths of movement steps are restricted. We shall look at such principle here again. Since we have changed the format of encoding, the form of the distance principles also changes slightly. Notice that in the reentrancy format it is not the links that get encoded directly but rather the movement paths. So, if $x \blacktriangleright_{ij} y$ this means that x and y are members of a chain and that there has been movement from y to x. This has advantages in the codification of movement. For we can set down distance principles in a very direct way as follows. While before we had

$$\langle \blacktriangledown_j \rangle p \to \langle \nabla^+ \rangle p \qquad (36)$$

we now consider postulates of the form

$$\langle \blacktriangledown_j \rangle p \to \langle \alpha_j \rangle p \qquad (37)$$

There are now two cases to consider. If we consider Freeze Movement then the distance covered in a single movement step is measured in terms of underived links (see (Kracht, 2003)), that is, white relations. We can capture this by requiring that α_j is a program not using any of the $\blacktriangledown_j$. If we are interested in Shortest Move then matters are different. Here the movement path will involve also derived links, that is to say, black relations. In this case the principle is stated as follows:

$$\langle \blacktriangledown_j \rangle p \to \langle \alpha_j \rangle p \qquad (38)$$

with the condition that in the execution chain (to be defined below) δ_0 does not contain $\blacktriangledown_j$. This condition ensures that we measure the movement path of the link against the different alternatives. We shall defer the treatment of Shortest Steps and concentrate here on Freeze derivations.

On certain conditions on α_j the present construction can be repeated almost verbatim. What one must ensure is that if $y \xrightarrow{\alpha_j} x$ holds in $\mathfrak{T}$, it also holds in $\mathfrak{E}$. This is not the case for all α_j. However, under certain conditions this is the case. One case that interests us here is the case where α_j defines a command relation in terms of the white relations (see (Kracht, 1999)).

We shall give a proof below. Command relations have the property that they continue to hold even if points are removed in a tree. To define that notion, let us say the following. An **execution chain** of α is a series $\gamma = \pi_0;\delta_0?;\pi_1;\delta_1?;...;\pi_{n-1};\delta_{n-1}?$ such that all π_i, $i < n$, are basic programs and $R([\gamma]) \subseteq R([\alpha])$. γ' is called a **subchain** if it is obtained from γ by removing some occurrences of the δ_i or π_i (but keeping their order).

Definition 15 *Let α be a program. α has the* ***subchain property*** *if for every execution chain γ of α every subchain of γ is an execution chain of α.*

Let us compare the chains of programs in $\mathfrak{E}$ and $\mathfrak{M}$, in particular those of the tree relations. If $x \rhd_i y$ in $\mathfrak{E}$ then $x >_i y'$ for some $y' \geq y$ with $a(y') = a(y)$. Thus, all we can say is the following. Every chain $\gamma = \nabla_j;\delta?$ in $\mathfrak{E}$ where $\delta? \in SF(\varphi)$ corresponds to an execution chain γ' of the program

$$\nabla_j;\delta? \cup \nabla_j;\delta?;(\nabla;\top?)^*;\nabla;\delta? \qquad (39)$$

It is not hard to see that γ is a subchain of γ'.

Definition 16 *Let α be a program that has the subchain property. A reentrancy structure is called (α, j)-**distance restricted** if the logic satisfies*

$$\langle \blacktriangledown_j \rangle p \to \langle \alpha \rangle p \qquad (40)$$

*For $\Delta = \{(\alpha_0, 0), (\alpha_1, 1), ..., (\alpha_{n-1}, n-1)\}$ we say that $\mathfrak{M}$ is Δ-**distance restricted** if it is (α_i, i)-distance restricted for every $i < n$.*

Now recall again the proof of Theorem 13. The proof goes through as before. What we must ensure however is that $\mathfrak{E}$ also is a structure for the logic. To this end it suffices to note the following: every chain of α in $\mathfrak{E}$ is a subchain of a chain of α in $\mathfrak{M}$. By construction, if $x \blacktriangleleft_j y$ in $\mathfrak{M}$ there is a z such that $z\ R([\alpha])\ y$ and $x = z$ or $x \blacktriangleleft_j z$. We have seen to it that there is also an egregious z such that either $z = x$ or $x \blacktriangleleft_j z$. What remains to be seen is that $z\ R([\alpha])\ y$ in $\mathfrak{E}$. This follows from the fact that there is an α-chain γ in $\mathfrak{M}$ from y to z, and it has a correlate γ $\mathfrak{E}$, which is a subchain of γ. Hence it is an α-chain from y to z, as promised.

Theorem 17 *For every Δ where all programs have the subchain property the logic of Δ-distance restricted reentrancy structures has the finite model property and is decidable in 2EXPTIME.*

8 Movement

We shall point a particular application. A **command relation** is a relation R that is characterised by the following property: there is a finite set S of sequences $\vec{\eta} = \langle \eta_0;\eta_1;...;\eta_{n-1}\rangle$ of constant formulae such that $x\ R\ y$ iff for the least z that strictly dominates x and nonstrictly dominates y: the sequence of points that are strictly between x and z does not contain a subsequence that is contained in S. Somewhat more exactly: S contains sequences of properties of points, and a sequence $\langle x_i : i < n\rangle$ of points satisfies such a sequence $\vec{\eta}$ iff $x_i \vDash \eta_i$ for all $i < n$. Let us denote the relation by $C(S)$.

For example, **idc-command** is defined be the set $\{\langle \top \rangle\}$. Thus, x c-commands y iff for the least $z > x$ and $z \geq y$: the set $U = \{u : x < u < z\}$ does not contain a subsequence satisfying $\langle \top \rangle$. This is the case iff there is no nonempty subsequence iff $U = \varnothing$ iff z is immediately above x. Next, **0-subjacency** is defined by $\{\langle \mathrm{cp}, \mathrm{ip}\rangle\}$. Thus, x 0-subjacency commands y iff for the least z such that $z > x$, $z \geq y$: the set $V := \{u : x < u < z\}$ does not contain a subsequence $\langle x_0, x_1\rangle$ of points such that $x_0 \vDash \mathrm{ip}$ and $x_1 \vDash \mathrm{cp}$ (see (Kracht, 1998)). We are interested in such relations $D(S)$ of the form

$$x\ D(S)\ y :\Leftrightarrow y\ C(S)\ x \text{ and } y < x \tag{41}$$

These are the nearness relations defined in the Koster-matrix (see (Koster, 1986) and (Kracht, 1993)). They can be described by programs, which we denote by $\delta(S)$. These programs have the subchain property. Suppose that γ is an execution chain of $\delta(S)$ and that γ' is a subchain. By definition, no subchain of γ is an execution chain of $D(S)$, and this holds a fortiori of γ'.

Corollary 18 *Let $\Delta = \{(\delta(S_j), j) : j < n\}$ and let $\mathcal{K}$ be the class of Δ-distance restricted reentrancy structures. Then $L(\mathcal{K})$ has the finite model property and satisfiability is in 2EXPTIME.*

It follows that the α defined through $D(S)$—and these are the ones that are of linguistic interest—are preserved by passing from $\mathfrak{M}$ to the model $\mathfrak{E}$ of egregious points. Yet, I should point out that the restriction to white relations in the definitions practically means that the distance principle define distance with respect to D-structure. Or, equivalently, if we are looking for a derivational account, they encode true movement paths only for Freeze-movement. This means that we still have to find analogous results for Shortest Steps (which is the most common type of movement).

Now what if α is not a command relation or of the form $D(S)$? Then so far anything is possible. However let us mention a particular case, namely when we have conditions of the form

$$\langle \blacktriangledown_j\rangle p \to \langle \alpha_j\rangle p \tag{42}$$

where α_j contains only white relations (even in tests).

Corollary 19 *Suppose that $\mathcal{K}$ is the class of Δ-distance restricted reentrancy structures defined by distance programs of the form (42). Then $L(\mathcal{K})$ has the finite model property and is decidable in 2EXPTIME.*

9 Shortest Steps

Now let us consider the distance principles related to Shortest Steps Movement. They are of the form

$$\langle \blacktriangledown_j\rangle p \to \langle \alpha_j\rangle p \tag{43}$$

where the first program of the computation trace of α_j does not contain $\blacktriangledown_j$.

Definition 20 *Call α* ***initially white*** *if there are β_i, $i < n$, such that*

$$\alpha \subseteq \bigcup_{i<m} \triangledown_i; \beta_i \tag{44}$$

We start with fact that **NRS** has the finite model property. Let $X(\varphi)$ be the following set

$$X(\varphi) := \{\langle \blacktriangledown_j\rangle p \to \langle \alpha_j\rangle p : p \in At(\varphi), j < n\} \tag{45}$$

Furthermore, let $At^+(\varphi)$ the set of atoms based on $\varphi; X(\varphi)$. By the previous results there is a finite **NRS**-model

$$\langle \mathfrak{M}, \beta, x\rangle \vDash \varphi; \boxdot X(\varphi) \tag{46}$$

We may assume the frame is generated from x via the white relations. By induction we define a sequence $\hat{\blacktriangleright}^p_j$ of relations and a sequence

$$\mathfrak{M}_p := \langle M, \{\triangleright_i : i < m\}, \{\blacktriangleright_j : j < n\}\rangle \tag{47}$$

Moreover, the inductive claim is that for every $\delta \in FL(\varphi; X(\varphi))$:

$$\langle \mathfrak{M}_{p+1}, \beta, x\rangle \vDash \delta \quad \Leftrightarrow \quad \langle \mathfrak{M}_p, \beta, x\rangle \vDash \delta \tag{48}$$

Denote by $a_p(x)$ the φ-atom of x in $\langle \mathfrak{M}_p, \beta\rangle$. Then (48) is true if for all x: ❶: $a_{p+1}(x) = a_p(x)$ and ❷: for all $\delta = \langle \beta\rangle \nu$ a subformula of $\langle \alpha_j\rangle \nu$ or $\delta = \langle \blacktriangledown_j\rangle \nu$ (48) holds. This is the way the results is going to be proved.

From this we get that $a_{p+1}(x) = a_p(x)$, and so by induction $a_p(x) = a_0(x)$. From (48) we get that for all $x \in M$:

$$\langle \mathfrak{M}_p, \beta, x\rangle \vDash X(\varphi) \tag{49}$$

For all points x of height $\neq p$ we set $x \hat{\blacktriangleright}^{p+1}_j y$ iff $x \hat{\blacktriangleright}^p_j y$. For x of height p we do the following. Suppose that $x \hat{\blacktriangleright}^p_j y$. Two cases arise. Either $x \xrightarrow{\alpha_j}_{\mathfrak{M}_p} y$ or not. In the first case we put $x \hat{\blacktriangleright}^{p+1}_j y$. In the second case we choose a y' such that $x \xrightarrow{\alpha_j}_{\mathfrak{M}_p} y'$

and $a_p(y') = a_p(y)$ and then put $x\hat{\blacktriangleright}^p_j y'$ (and eliminate the old arc). That y' exists is seen as follows. First, we have $\langle\mathfrak{M}_p,\beta,x\rangle \vDash \langle\blacktriangledown_j\rangle a_p(y)$, where $a_p(y)$ is the atom of y in $\langle\mathfrak{M}_p,\beta\rangle$. By (49) we have $\langle\mathfrak{M}_p,\beta,x\rangle \vDash \langle\alpha_j\rangle a_p(y)$. And so there is a y' with $x \overset{\alpha_j}{\rightarrow}_{\mathfrak{M}_p} y'$ and $a_p(y') = a_p(y)$, as desired. Now using Lemma 11 we get ❶, which is (48) for all $\delta \in FL(\varphi)$. Finally, we need to establish ❷, which is (48) for formulae of the form (A) $\langle\blacktriangledown_j\rangle\nu$ or (B) $\langle\beta\rangle\nu$. Case (❷A) is immediate from the definition. Case (❷B) is done by induction on the complexity of β. $\beta = \beta' \cup \beta''$ is immediate. $\beta = \chi?$ is immediate. $\beta = \beta'^*$. Then $\beta = \top? \cup \beta';\beta'^*$ and so is reduced to the cases $\top?$ and $\beta';\beta'^*$. There remains the case $\beta = \beta';\beta''$. Now, either β' is simple or we can reduce it analogously. Using the associativity of ; and distributivity over $\cup$ we can reduce everything to the case that β' is basic and the formula has the form $\langle\beta';\beta''\rangle\nu$, which is equivalent to $\langle\beta'\rangle\langle\beta''\rangle\nu$. Now, $\langle\beta''\rangle\nu \in FL(X(\varphi))$. Assume that $\beta \neq \blacktriangledown_j$. $x \overset{\beta_j}{\rightarrow}_{\mathfrak{M}_{p+1}} y$ iff $x \overset{\beta_j}{\rightarrow}_{\mathfrak{M}_p} y$ and this gives the claim together with (48) for y. If $\beta = \blacktriangledown_j$ then let y and y' be such that $x \overset{\blacktriangledown_j}{\rightarrow}_{\mathfrak{M}_p} y$ and $x \overset{\blacktriangledown_j}{\rightarrow}_{\mathfrak{M}_{p+1}} y'$. We apply the inductive hypothesis (48) for y.

The inductive construction is such that if x is of height n then $x \overset{\blacktriangledown_j}{\rightarrow}_{\mathfrak{M}_p} y$ implies $x \overset{\alpha_j}{\rightarrow}_{\mathfrak{M}_p} y$ for all $p \geq n$. So if q is the height of the entire tree, the model we need is $\langle\mathfrak{M}_q,\beta\rangle$.

Theorem 21 *Suppose that $\mathcal{K}$ is the class of Δ-distance restricted reentrancy structures defined by initially white distance programs. Then $L(\mathcal{K})$ has the finite model property and is decidable in 2EXPTIME.*

Proof. The finite model property and decidability follow from the previous. Now, the complexity is more subtle. Given φ we are building a model for $\varphi; X(\varphi)$, which contains 2^n formulae of length linear in $n := |\varphi|$. Given the 2EXPTIME bound for **NRS** this gives a bound of 3EXPTIME. However, rather than cascading the proof one can work out a direct proof of an analogue of Theorem 13. □

It follows that the theory of any class of structures of generative grammar constrained by Shortest Steps movement and distance regulated by command relations is decidable.

10 Naming The Egregious Points

Given that we can bound the size of a model we can now also introduce nominals that will cover the entire frame. This is done as follows. An **address** is a sequence $\mathfrak{v} = \alpha_0; b_0; \alpha_1; b_1; ...; b_{n-1}; \alpha_n$, where the α_i are atoms and pairwise distinct, and $b_i < i$. Call $\sigma(\varphi)$ the set of addresses. For each address $\mathfrak{v}$ we introduce a new variable $p_{\mathfrak{v}}$. These variables are contained in the set EVar.

$$\xi := \bigvee\langle p_{\mathfrak{v}} : \mathfrak{v} \in \sigma(\varphi)\rangle \tag{50}$$

Consider now the following formula A. Call it Θ:

$$\begin{aligned}
& \textstyle\bigwedge\langle p_{\mathfrak{v};b;\alpha} \rightarrow [\triangledown](\alpha \rightarrow p_{\mathfrak{v};b;\alpha}) \\
& \quad : \mathfrak{v}; b; \alpha \in \sigma(\varphi)\rangle \\
\wedge \quad & \textstyle\bigwedge\langle p_{\mathfrak{v};b;\alpha} \wedge [\triangledown_i]\neg\alpha \rightarrow \bigwedge_{i<j<m}[\triangledown_j](\alpha \\
& \rightarrow \neg\xi) : \mathfrak{v}; b; \alpha \in \sigma(\varphi), i < m\rangle \\
\wedge \quad & \textstyle\bigwedge\langle p_{\mathfrak{v};b;\alpha} \rightarrow [\triangledown_i](\beta \rightarrow \neg\xi) \\
& \quad : \beta \in \mathfrak{v}, \beta \neq \alpha\rangle \\
\wedge \quad & \textstyle\bigwedge\langle p_{\mathfrak{v}} \rightarrow [\triangledown_i](\alpha \rightarrow p_{\mathfrak{v};j;\alpha}) \\
& : \alpha \notin \mathfrak{v}, j < i < m\rangle \\
\wedge \quad & \textstyle\bigwedge\langle p_{\mathfrak{v}} \rightarrow \neg p_{\mathfrak{w}} : \mathfrak{v} \neq \mathfrak{w}\rangle
\end{aligned} \tag{51}$$

Further,

$$\Xi := [\triangledown^+]\Theta \wedge \bigwedge_{\alpha\in\sigma(\varphi)} \alpha \leftrightarrow p_\alpha \tag{52}$$

Proof similar of Theorem 13:

Lemma 22 *For every valuation β on the set Var such that $\langle\mathfrak{M},\beta,x\rangle \vDash \varphi$ there is a unique extension γ defined on $Var \cup EVar$ such that $\langle\mathfrak{M},\gamma,x\rangle \vDash \varphi; \Xi$.*

Lemma 23 *Let $\langle\mathfrak{M},\beta,x\rangle \vDash \Xi$. Then $w \vDash p_{\mathfrak{v}} \wedge [\triangledown_{10}]\neg p_{\mathfrak{v}}$ iff w is egregious of address $\mathfrak{v}$.*

Thus put

$$E(\varphi) := \bigvee\langle p_{\mathfrak{v}} \wedge [\triangledown^+]\neg p_{\mathfrak{v}} : \mathfrak{v} \in \sigma(\varphi)\rangle \tag{53}$$

Then $E(\varphi)$ is true exactly at the egregious points. For an egregious point w we have a formula q_w which is true exactly at w. Let us recall how the new model was defined on the egregious points. We have $w \lhd_i v$ iff $w \leq u \prec_i v$ and $a(w) = a(u)$. Thus the fact that $w \lhd_i v$ can be expressed as

$$[\triangledown^*](p_v \rightarrow \langle\triangledown_i; \triangledown^*\rangle p_w) \tag{54}$$

11 Broadening The Scope

Now we shall generalise the theorems even further. This will allow to derive decidability even in presence of adjunction. We start again with a structure $\langle M, \{\rhd_i : i < m\}, \{\blacktriangleright_j : j < n\}\rangle$ such that $\langle M, \{\rhd_i : i < m\}\rangle$ is a finite tree. The additional relations must satisfy a few conditions. First, we assume that $\blacktriangleright_j$ as well as its converse is a partial function. This is axiomatised as follows. Put

$$\begin{aligned} n(p) := \neg \boxdot \neg p \wedge \boxdot(p \to [\triangledown^+]\neg p)\wedge \\ \boxdot \textstyle\bigwedge_{i<j<m} \neg(\langle \triangledown_i; \triangledown^* \rangle p \wedge \langle \triangledown_j; \triangledown^* \rangle p) \end{aligned} \tag{55}$$

The desired axiom is

$$\begin{aligned} d(p,q) := n(p) \wedge \neg \boxdot \neg(q \wedge \langle \blacktriangledown_j \rangle p) \\ \to \boxdot([\triangledown_j]p \to q) \end{aligned} \tag{56}$$

It says that if p is true at a single point, then the set of points seeing p through $R([\blacktriangledown_j])$ is a singleton as well. Second, we shall require that if $x\ R([\blacktriangledown_j])\ y$ then y is within a certain distance of x. This notion of distance is what we now turn to. For the purpose of the next definition notice that if α has the subchain property, so does $\alpha^{\smile}$.

Definition 24 *An **oval** is a program of the form $\alpha^{\smile};\beta$, where both α and β have the subchain property and neither contains any of the $\blacktriangleright_j$, $j < n$.*

So the desired axiom is

$$o(p) := \langle \blacktriangledown_j \rangle p \to \langle \alpha^{\smile};\beta \rangle p \tag{57}$$

Definition 25 *Let $\mathcal{K}$ be a class of structures $\langle M, \{\rhd_i : i < m\}, \{\blacktriangleright_j : j < n\}\rangle$ such that*

① *$\langle M, \{\rhd_i : i < m\}\rangle$ is an m-branching tree, all relations being partial functions.*

② *All $\blacktriangleright_j$ are partial functions.*

③ *There are ovals $\alpha^{\smile};\beta$ such that if $x \blacktriangleright_j y$ then $x \xrightarrow{\alpha^{\smile};\beta} y$.*

*Then $\mathcal{K}$ is a class of **oval-expanded trees**.*

As usual, our logic will be a $i + j + 1$-modal logic. The added modalities are all definable and used for the eye only. It is crucial to understand that since we did not require of the $\blacktriangleright_j$ that they are cycle-free we cannot use the Kleene star on programs containing any black relations. It is used only on white relations.

We now proceed to a proof that the logic of a class of oval-expanded trees is decidable. As usual it will turn out that we can construct a model from the egregious points, from which a complexity bound can be derived. We start with the logic $\mathbf{Tree}_m$, which comprises axioms for $\triangledown_i$ and $>$. This logic has the finite model property and is EXPTIME complete. For $\blacktriangledown_j$ we choose the logic $\mathbf{Alt}_1$. Thus we start with

$$L := \mathbf{Tree}_m \otimes \bigotimes_{j<n} \mathbf{Alt}_1 \tag{58}$$

This logic has the finite model property and is complete with respect to finite trees. Put

$$\begin{aligned} Y(\varphi) := \{d(p_{\mathfrak{v}}, p_{\mathfrak{w}}) : \mathfrak{v}, \mathfrak{w} \in \sigma(\varphi)\}; \\ \{o(p) : p \in At(\varphi)\} \end{aligned} \tag{59}$$

Assume that φ is satisfiable in the logic of $\mathcal{K}$. Hence $Y(\varphi)$ is $\mathcal{K}$-satisfiable, since it adds instances of the axioms. A fortiori, it is L-satisfiable in a tree. Therefore, $Y(\varphi); \Xi$ is satisfiable as well. So, assume $\mathfrak{M}$ is a tree and that

$$\langle \mathfrak{M}, \beta, w \rangle \vDash \Xi; Y(\varphi) \tag{60}$$

We let $E(\varphi)$ be the set of egregious points, and define $\rhd_i$ as before: $u \rhd_i v$ iff v is the unique egregious point such that $v \leq v' \rhd_i u$ and $a(v) = a(v')$. Further, for $j < n$, define the following function $(-)^{\spadesuit_j}$. It is defined on the egregious points u with a $\blacktriangleright_j$-successor. Assume u has a successor via $\blacktriangleright_j$ in $\mathfrak{M}$. Then by assumption there is a v such that $u \xrightarrow{\alpha^{\smile};\beta} v$. Now let $v^{\spadesuit}$ be an egregious point below v with the same atom. Put $u \gtrdot_j v^{\spadesuit}$ in the new structure. By the subchain property, it will also hold that $v^{\spadesuit}$ is in the oval of u in the new structure. $\gtrdot_j$ is a partial function since $v \mapsto v^{\spadesuit}$ is. It follows that egregious points have the same atom in the new structure as they do in the old structure (by induction on the formulae). The new structure is now

$$\mathfrak{E} := \langle E(\varphi), \{\rhd_i : i < m\}, \{\gtrdot_j : j < n\}\rangle \tag{61}$$

The valuation is $\gamma(p) := \beta(p) \cap E(\varphi)$. As before, it is shown by induction on $\chi \in SF(\varphi)$ that for every egregious x:

$$\langle \mathfrak{E}, \gamma, x \rangle \vDash \chi \quad \Leftrightarrow \quad \langle \mathfrak{M}, \beta, x \rangle \vDash \chi \tag{62}$$

This is by now routine. All that needs to be shown is that the converse of $\gtrdot_j$ is a partial function as well. Pick $v^{\spadesuit}$. We have $\langle \mathfrak{M}, \beta, v^{\spadesuit} \rangle \vDash p_{\mathfrak{v}}$ for a certain $\mathfrak{v} \in \sigma(\varphi)$. It follows that also $\langle \mathfrak{M}, \beta, v \rangle \vDash p_{\mathfrak{v}}$. By construction, $u \gtrdot_j v^{\spadesuit}$ means that u is egregious, and so $u \vDash p_{\mathfrak{w}}$ for some $\mathfrak{w}$. Since the root satisfies $d(p_{\mathfrak{v}}, p_{\mathfrak{w}})$, we have that every point u' such that u' sees a point satisfying $p_{\mathfrak{v}}$ must satisfy $p_{\mathfrak{w}}$. Thus, $u' \geq u$, and either $u' = u$ or u' is not egregious. This shows the claim.

Theorem 26 *The modal logic of a class of oval-expanded trees has the finite model property and is decidable. A model for φ has at most 2^{2^n} points, where n is the number of subformulae of φ.*

12 Conclusion

As outlined ((Kracht, 1989), (Kracht, 1993), (Kracht, 1995b), (Kracht, 1998), (Kracht, 2001), (Kracht, 2003) and (Kracht, 2008)), the theories of generative grammar, starting with GB can be modelled entirely using modal logic over five basic relations. This does not mean that there is a single logic that describes them all (see (Kracht, 1995a) for an extensive discussion). It means however that the theories describe a set of grammars, and thus a set of possible logics for grammars. Here I have shown that—excluding economy principles and copies e.g. (Kobele, 2006)—the entire logic is decidable. Thus, one can effectively decide for any pair of GB/MP theories whether they generate the same structures (not strings, as this is even undecidable in the context free case).

References

Patrick Blackburn. 1993. Nominal tense logic. *Notre Dame Journal of Formal Logic*, 39:56 – 83.

Stephan Kepser. 2010. On monadic second-order theories of multidomanince structures. In *The Mathematics of Language*, volume 6149, pages 117–128. Springer.

Greg Kobele. 2006. *Generating Copies: An investion into structural identity in language and grammar*. PhD thesis, University of California, Los Angeles.

Hans-Peter Kolb, Jens Michaelis, Uwe Mönnich, and Frank Morawietz. 2003. An operational and denotational approach to non-context-freeness. *Theoretical Computer Science*, 293(2):261 – 289.

Jan Koster. 1986. *Domains and Dynasties: The Radical Autonomy of Syntax*. Foris, Dordrecht.

Marcus Kracht. 1989. On the logic of category definitions. *Computional Linguistics*, 15:111 – 113.

Marcus Kracht. 1993. Mathematical aspects of command relations. In *Proceedings of the EACL 93*, pages 241 – 250.

Marcus Kracht. 1995a. Is there a genuine modal perspective on feature structures? *Linguistics and Philosophy*, 18:401 – 458.

Marcus Kracht. 1995b. Syntactic Codes and Grammar Refinement. *Journal of Logic. Language and Information*, 4:41 – 60.

Marcus Kracht. 1998. Adjunction Structures and Syntactic Domains. In *The Mathematics of Sentence Structure. Trees and Their Logics*, 44, pages 259 – 299, Berlin. Mouton–de Gruyter.

Marcus Kracht. 1999. *Tools and Techniques in Modal Logic*. 142. Elsevier, Amsterdam.

Marcus Kracht. 2001. Syntax in Chains. *Linguistics and Philosophy*, 24:467 – 529.

Marcus Kracht. 2003. Constraints on derivations. *Grammars*, 6:89 – 113.

Marcus Kracht. 2008. On The Decidability of The Logic of LGB–Type Structures. pages 105 – 142.

James Rogers. 1994. *Studies in the Logic of Trees with Applications to Grammar Formalisms*. PhD thesis, University of Delaware, Department of Computer & Information Sciences.

Moshe Vardi and P. Wolper. 1986. Automata theoretic techniques for modal logics of programs. *Journal of Computer and Systems Sciences*, 32:183 – 221.

Conditions for Global Asymptotic Dominance in Multidimensional Grammar Competition

Henri Kauhanen
Department of Linguistics, University of Konstanz
henri.kauhanen@uni-konstanz.de

Abstract

In variational learning of two grammars, the grammar with the greater advantage is globally asymptotically dominant, meaning that this grammar wins out over inter-generational time, leading to the extinction of its competitor. Here, we prove necessary and sufficient conditions for global asymptotic dominance in competition between an arbitrary number $n \geq 2$ of grammars, working in the deterministic limit of iterated inter-generational linear reward–penalty learning.

1 Introduction

Grammar competition (Kroch, 1989), particularly when combined with variational learning[1] (Yang, 2002), is a much-employed framework for explaining language change (see e.g. Yang, 2000; Heycock and Wallenberg, 2013; Simonenko et al., 2019). In its usual application, the procedure is to define a simple model of language acquisition, and then to bootstrap a model of population-level linguistic change from it. This is typically done by setting up a discrete sequence of non-overlapping generations of learners, assuming that the learning environment for learners in each generation is constituted by the average linguistic production of learners of the immediately preceding generation, abstracting away from stochastic fluctuations (cf. Andersen, 1973).

The dynamics of such a system are fully understood in the case of two competing grammars; in this special case, the population-level equation is a replicator dynamic with a flat fitness landscape. As a consequence, these systems are capable of two types of behaviour only: either every population state is a stable (albeit not asymptotically stable) equilibrium, or a single asymptotically stable equilibrium exists, attracting all non-equilibrium initial states. Typically, the latter situation is the linguistically interesting one, implying that, over repeated inter-generational interactions, one grammar ousts the other. Which grammar wins depends on the competing grammars' relative advantages; these can be estimated from corpora, enabling one to empirically test the predictions made by the mathematical model.

The linear reward–penalty learning scheme (Bush and Mosteller, 1955) underlying the variational learner has a natural extension to n actions—in our case, to competition between an arbitrary number n of grammars. This extension has rarely been employed, however, and its mathematics also remain—apart from results concerning systems embodying certain kinds of strong symmetries (Kauhanen, 2019)—unexplored.

Here, we study the general n-grammar model in detail. We demonstrate that the population-level equation is again a replicator dynamic, although generically (whenever $n > 2$) fitnesses are frequency-dependent and nonlinear. Despite this, certain crucial aspects of the dynamics remain characterizable. In particular, we prove necessary and sufficient conditions for the *global asymptotic dominance* of a single grammar under n-way competition, meaning that the population state corresponding to total use of this grammar is an attractor for any non-equilibrium initial state in which the grammar has some non-zero abundance. In other words, we provide an answer to the question: once a particular grammatical change has been actuated (cf. Weinreich et al., 1968), under what conditions will it proceed to completion? These conditions only reference pairwise advantages between grammars and as such they generalize the well-known "fundamental theorem of language change" for two grammars (Yang, 2000). Our results turn on the notion of the *resilience* of a grammar, defined as the reciprocal of the advantage that its competitor(s) hold(s) over it.

[1] Not to be confused with the collection of methods known as variational inference in Bayesian learning theory (MacKay, 2003).

The necessary and sufficient conditions for global asymptotic dominance make available further results about the dynamics of grammar competition between more than two grammars. As an illustration, we demonstrate how the sufficient condition can alternatively be characterized using the notion of grammatical *flux*, a measure of the tendency for a grammar to lose abundance to its competitors in a speech community.

2 Definitions

In the general case, we assume that the language learner has access to n grammars $G_1, \ldots, G_n$ and attaches a *weight* W_i to each grammar G_i (cf. Yang, 2002). The weights are assumed to form a categorical probability distribution over the grammars; in other words, they must satisfy the following conditions:[2]

1. $W_i \geq 0$ for all i
2. $\sum_i W_i = 1$

Each weight is a random variable; together they form the random probability vector $\mathbf{W} = [W_1 \ldots W_n]^T = (W_1, \ldots, W_n)$. We note that $\mathbf{W}$ is an element of Δ^{n-1}, the simplex

$$\Delta^{n-1} = \left\{ \mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}_+^n : \sum_i x_i = 1 \right\}.$$

Whenever no confusion can arise, Δ^{n-1} will be written as Δ for short. This set is partitioned into its *boundary*

$$\partial\Delta = \{\mathbf{x} \in \Delta : x_i = 0 \text{ for some } i\}$$

and *interior*

$$\Delta^\circ = \{\mathbf{x} \in \Delta : x_i > 0 \text{ for all } i\}.$$

The boundary $\partial\Delta$ contains, in particular, the n *vertices*, which are the standard basis vectors of $\mathbb{R}^n$; we denote these by $\mathbf{e}_1 = (1, 0, \ldots, 0)$ through to $\mathbf{e}_n = (0, \ldots, 0, 1)$.

Additionally, for every non-empty subset of grammar indices $H \subseteq \{1, \ldots, n\}$, we define Δ_H as the set of points $\mathbf{x} \in \Delta$ in which G_i has zero weight for each $i \notin H$, in other words

$$\Delta_H = \{\mathbf{x} \in \Delta : x_i = 0 \text{ for all } i \notin H\}.$$

Equivalently, $\Delta_H = \Delta \cap \operatorname{span}\{\mathbf{e}_i : i \in H\}$. Intuitively, Δ_H is the set of possible weight vectors in which grammars outside the index set H never occur; it is itself an $(|H| - 1)$-dimensional simplex.

Each grammar is assumed to accept a language, understood as a set of grammatical expressions out of all expressions it is possible to form using a common alphabet Σ. In other words, to each G_i we attach a set L_i satisfying $L_i \subseteq \Sigma^*$ where Σ^* is the Kleene star (the infinite set of all finite sequences of symbols from Σ). If $s \in L_i$ (i.e. if G_i accepts the string s), we also express this as $G_i \vdash s$. The complement of L_i with respect to Σ^* will be written $\complement L_i$; for all $s \in \complement L_i$, we have $G_i \nvdash s$.

To each G_i, we attach a probability measure μ_i on Σ^* with support on L_i, referred to as the grammar's associated *production measure*. For any $K \subseteq \Sigma^*$, $\mu_i(K)$ gives the probability of G_i producing an expression belonging to K. Whereas acceptance ($\vdash$) is a purely formal, grammatical property, production (μ_i) may depend on both grammatical and extra-grammatical factors, the latter including factors such as discourse constraints and performance limitations.[3]

Expanding on Yang (2000), and following Kauhanen (2019), we adopt the following definition.

Definition 1. The quantity

$$a_{ij} = \mu_j(\complement L_i)$$

is called the *(pairwise) advantage* of grammar G_j over grammar G_i.

In words, a_{ij} is the probability of G_j producing a string that G_i does not accept. It is useful to collect pairwise advantages in an $n \times n$ matrix $A = [a_{ij}]$; we refer to this as an *advantage matrix*. Note that such matrices have zero diagonals: $a_{ii} = 0$ since, by assumption, each production measure μ_i has no support outside L_i.

A learner operates in a learning environment, which we take to be a probability measure on Σ^* according to which sequences of expressions are presented to the learner.

Definition 2. A *learning environment* is any probability measure μ on Σ^*.

[2] Throughout this paper, missing bounds such as on a summation ($\sum_i$) are taken to imply that the index ranges from 1 to n, i.e. over all grammars. Further conditions on indices will be explicitly indicated whenever necessary.

[3] Construction of production measures is non-trivial. Under traditional conceptions of grammar, the language L_i generated by a grammar G_i is typically a (countably) infinite set, containing, among other things, expressions whose production probabilities are practically indistinguishable from zero (e.g. centre-embeddings to arbitrary depth). In practice, it normally suffices to assume that μ_i has finite support. In any case, these complexities need not concern us here, as long as *some* well-posed probability measure μ_i exists for each G_i.

Definition 3. The quantity

$$c_i^\mu = \mu(\complement L_i)$$

is referred to as the *penalty probability* of grammar G_i in the environment μ.

Whenever no confusion can arise, we drop the measure μ from the notation and denote the penalty probability of G_i simply as c_i.

Note that, on this definition, a learning environment is necessarily stationary, as it is associated with a single, unchanging measure on Σ^*. In practice, this means we assume that each learner (in a given generation of learners) is exposed to a constant set of penalty probabilities for the competing grammars over the duration of learning.

An important distinction exists between learning environments that punish each grammar at least some of the time, and environments which never punish some grammar(s). In order to have terminology for these, we define:

Definition 4. A learning environment is *omnipunitive* if $c_i > 0$ for all i. If this is not the case (i.e. if $c_i = 0$ for at least one i), the learning environment is *parapunitive*.

Now assume the learner receives an infinite random sequence of expressions $s_1, s_2, s_3, \ldots \in \Sigma^*$ according to such a μ. Upon reception of s_m, the learner samples a grammar to employ according to the current value of $\mathbf{W}$ (i.e. G_i is chosen with probability W_i). Suppose this grammar is G_k. The learner then checks whether $G_k \vdash s_m$ and, based on this result, applies one or another operator to $\mathbf{W}$ to transform it to its new value, $\mathbf{W}'$.[4] This amounts to assuming the existence of n *reward operators* $u_i^+ : \Delta \longrightarrow \Delta$ and n *punishment operators* $u_i^- : \Delta \longrightarrow \Delta$ and setting

$$\mathbf{W}' = \begin{cases} u_k^+(\mathbf{W}) & \text{if } G_k \vdash s_m \\ u_k^-(\mathbf{W}) & \text{if } G_k \nvdash s_m \end{cases} \qquad (1)$$

The learner then receives the next expression in the sequence (s_{m+1}), and the cycle continues.

The central learning-theoretic challenge concerns whether $\mathbf{W}$ converges in some relevant sense as learning continues, based on particular choices for the reward (u_i^+) and punishment (u_i^-) operators, subject to a given learning environment μ. Convergence results exist for a number of choices of these operators (see Bush and Mosteller, 1955; Norman, 1972; Narendra and Thathachar, 1989); in what follows, we will make use of these results in order to study a deterministic approximation, or *mean dynamic* (Sandholm, 2010), of the population-level evolution of a sequence of generations of such learners.

To obtain the population-level mean dynamic, we first need to obtain a mean dynamic that expresses how the expected value of the learner's weight vector, $\overline{\mathbf{W}} = \mathrm{E}[\mathbf{W}]$, evolves. In general, from (1) we have that

$$\overline{W}'_i = \sum_j W_j \big(c_j u_j^-(\mathbf{W})_i + (1 - c_j) u_j^+(\mathbf{W})_i \big).$$

Taking expectations on both sides, this becomes

$$\overline{W}'_i = \sum_j \overline{W}_j \big(c_j u_j^-(\overline{\mathbf{W}})_i + (1 - c_j) u_j^+(\overline{\mathbf{W}})_i \big). \qquad (2)$$

In the following sections, the operators $\{u_i^+, u_i^-\}$ will be given particular forms and the resulting mean dynamic studied in detail.

Passage to the population-level mean dynamic then follows via one (or both) of two routes. One can either assume that learning is so slow (but continued for long enough) that stochastic fluctuations inherent in the learning process affect the learner's eventual weight vector $\mathbf{W}$ only to an insignificant extent: the actual value of $\mathbf{W}$ stays close to its expected value, $\overline{\mathbf{W}}$ (see Norman, 1972; Narendra and Thathachar, 1989). Alternatively (or additionally), one can assume that the learners in each generation randomly sample input from an infinite population of independent, identically distributed learners in the previous generation; adherence to the population-level mean dynamic then follows by the Law of Large Numbers (cf. Niyogi, 2006). Taking either of these limits (learning rate to zero, or population size to infinity) recovers the same mean dynamic that relates the competing grammars' abundances from one generation to the next.

3 Linear Reward–Penalty Learning of Two Grammars

In the classical two-action learning problem (i.e. when $n = 2$), it has been customary ever since Bush and Mosteller (1955) to employ the linear map described by the matrix

$$U_1^+ = \begin{bmatrix} 1 & \gamma \\ 0 & 1 - \gamma \end{bmatrix}$$

[4] Throughout this paper, we use the notation x' for the successor of x.

for the reward operator u_1^+, meaning that

$$u_1^+(\mathbf{W}) = U_1^+\mathbf{W} = \begin{bmatrix} W_1 + \gamma W_2 \\ W_2 - \gamma W_2 \end{bmatrix}.$$

To punish G_1, the matrix

$$U_1^- = \begin{bmatrix} 1-\gamma & 0 \\ \gamma & 1 \end{bmatrix}$$

is used, so that

$$u_1^-(\mathbf{W}) = \begin{bmatrix} 1-\gamma & 0 \\ \gamma & 1 \end{bmatrix} \begin{bmatrix} W_1 \\ W_2 \end{bmatrix} = \begin{bmatrix} W_1 - \gamma W_1 \\ W_2 + \gamma W_1 \end{bmatrix}.$$

The corresponding matrices for rewarding and punishing G_2 are obtained as row permutations of U_1^+ and U_1^-:

$$U_2^+ = \begin{bmatrix} 0 & 1-\gamma \\ 1 & \gamma \end{bmatrix} \quad \text{and} \quad U_2^- = \begin{bmatrix} \gamma & 1 \\ 1-\gamma & 0 \end{bmatrix}.$$

In each of these formulae, $0 < \gamma < 1$ is a parameter that sets the *learning rate*; note we assume that the same learning rate applies to each operator.

Suppose that at least one of the penalty probabilities c_1 and c_2 is strictly positive. Then it is well known (see Bush and Mosteller, 1955; Narendra and Thathachar, 1989) that the expected value $\overline{\mathbf{W}}$ tends to

$$\overline{\mathbf{W}}^* = \left(\frac{c_2}{c_1 + c_2}, \frac{c_1}{c_1 + c_2} \right) \qquad (3)$$

with increasing learning iteration. In other words, $\overline{\mathbf{W}}^*$ is the (unique) fixed point of the learner's mean dynamic in this case.

Suppose we have a generation of learners, all exposed to the same learning environment, and sample a learner at random from this population. What is the probability that this learner employs G_1? Since all learners are assumed to operate in the same learning environment, this probability is identical to the probability of a single learner employing G_1 which, in turn, is just the expected value $\overline{\mathbf{W}}^*$. To avoid a proliferation of complex notation, and also to consistently distinguish between the individual and population levels, in what follows we will write x_i for the probability of encountering an individual employing grammar G_i, and call this the *abundance* of G_i. The abundances can be collected in a vector, $\mathbf{x} = (x_1, \ldots, x_n)$; of course, $\mathbf{x} \in \Delta$.

At the population level, the probability measure μ which defines the learning environment can be decomposed into the individual production measures of the competing grammars. Specifically, for any $K \subseteq \Sigma^*$,

$$\mu(K) = x_1\mu_1(K) + x_2\mu_2(K).$$

In other words, the probability of encountering a string $s \in K$ equals the probability of meeting a speaker employing G_1 and this speaker producing $s \in K$, plus the probability of meeting a speaker employing G_2 and this speaker producing $s \in K$. By the definition of penalty probability, we then have

$$\begin{cases} c_1 = \mu(\complement L_1) = x_2\mu_2(\complement L_1) = a_{12}x_2, \\ c_2 = \mu(\complement L_2) = x_1\mu_1(\complement L_2) = a_{21}x_1. \end{cases}$$

By (3), a learner in such an environment will tend to converge to

$$(\overline{W}_1^*, \overline{W}_2^*) = \left(\frac{a_{21}x_1}{a_{12}x_2 + a_{21}x_1}, \frac{a_{12}x_2}{a_{12}x_2 + a_{21}x_1} \right).$$

Assuming a discrete sequence of generations of this kind, we thus have the following deterministic difference equation for the abundances of the two grammars in the population:

$$\begin{cases} x_1' = \dfrac{a_{21}x_1}{a_{12}x_2 + a_{21}x_1} \\ x_2' = \dfrac{a_{12}x_2}{a_{12}x_2 + a_{21}x_1} \end{cases}$$

Since $x_1 + x_2 = 1$, it of course suffices to track the evolution of $x = x_1$ only, whereby we have the simple expression

$$x' = \frac{a_{21}}{\alpha(x)}x, \qquad (4)$$

where

$$\alpha(x) = a_{12}(1-x) + a_{21}x$$

denotes average advantage.

Equation (4) is a discrete-time constant-fitness Maynard Smith replicator dynamic (cf. Sandholm, 2010). As such, its behaviour is extremely simple. Outside of the case of a symmetric advantage matrix ($a_{12} = a_{21}$), in which case every $x \in [0,1]$ is a fixed point, the system has a single asymptotically stable equilibrium that attracts from any initial state $0 < x < 1$. If $a_{21} > a_{12}$, this equilibrium is $x = 1$; if $a_{21} < a_{12}$, it is $x = 0$.[5] This is summarized in:

Theorem 1 (Yang, 2000)**.** *When $n = 2$, the grammar with the greater advantage wins.*

[5] We do not concern ourselves with the pathological case $a_{12} = a_{21} = 0$ in which neither grammar ever produces output that its competitor cannot parse, i.e. in which the grammars are extensionally equivalent.

4 Linear Reward–Penalty Learning of n Grammars

We now turn to the general case of n competing grammars, where Bush and Mosteller's (1955) linear scheme takes the following form. Suppose G_k is the grammar selected by the learner for input sequence s_m. Then, the grammar weights are updated as follows:[6]

$$\begin{aligned}&\text{if } G_k \vdash s_m\text{:} &&\begin{cases} W_k' = (1-\gamma)W_k + \gamma \\ W_i' = (1-\gamma)W_i & (i \neq k)\end{cases}\\ &\text{if } G_k \nvdash s_m\text{:} &&\begin{cases} W_k' = (1-\gamma)W_k \\ W_i' = (1-\gamma)W_i + \frac{\gamma}{n-1} & (i \neq k)\end{cases}\end{aligned} \tag{5}$$

To characterize a learner's behaviour under this learning algorithm, we focus on the expected weight vector $\overline{\mathbf{W}} = \mathrm{E}[\mathbf{W}]$. With the operators (5), it can be checked that the learner's mean dynamic (2) simplifies to

$$\overline{W}_i' = (1-\gamma c_i)\overline{W}_i + \sum_{j\neq i} \frac{\gamma}{n-1} c_j \overline{W}_j.$$

In other words, we may write

$$\overline{\mathbf{W}}' = B\overline{\mathbf{W}}, \tag{6}$$

where $B = [b_{ij}]$ is the $n \times n$ square matrix with

$$b_{ij} = \begin{cases} 1-\gamma c_i & \text{if } i = j, \\ \frac{\gamma}{n-1} c_j & \text{if } i \neq j. \end{cases}$$

Generically, $\overline{\mathbf{W}}$ evolves to a limit whose value can be obtained by solving the system of equations $B\overline{\mathbf{W}} = \overline{\mathbf{W}}$. In an omnipunitive environment (i.e. $c_i > 0$ for all i), this system of equations has the unique solution $\overline{\mathbf{W}} = (c_1^{-1}/C, \ldots, c_n^{-1}/C)$ with $C = \sum_i c_i^{-1}$ (Bush and Mosteller, 1955; Narendra and Thathachar, 1989).[7] The following result generalizes this to also cover parapunitive environments.

Theorem 2. *If the learning environment is omnipunitive, then*

$$\overline{\mathbf{W}}^* = \left(\frac{c_1^{-1}}{\sum_i c_i^{-1}}, \ldots, \frac{c_n^{-1}}{\sum_i c_i^{-1}}\right)$$

is the only fixed point of the learner's mean dynamic (6).

If it is parapunitive so that $c_i = 0$ for $i \in H_0$, then all $\overline{\mathbf{W}} \in \Delta_{H_0}$ are fixed points.

Proof. First suppose the environment is omnipunitive. Then $c_i > 0$ for all i, and so the matrix B is both positive and column-stochastic. By the Perron–Frobenius Theorem, there is then a unique stationary $\overline{\mathbf{W}}$ that satisfies the equilibrium condition $\overline{\mathbf{W}} = B\overline{\mathbf{W}}$. It is quick to check that the $\overline{\mathbf{W}}^*$ stated in the theorem satisfies it.

Then suppose the environment is parapunitive. We begin by showing that a $\overline{\mathbf{W}}$ that satisfies the conditions stated in the theorem is a fixed point of the mean dynamic. We have

$$\overline{W}_i' - \overline{W}_i = -\gamma c_i \overline{W}_i + \sum_{\substack{j\neq i\\ j\in H_0}} \frac{\gamma c_j}{n-1}\overline{W}_j + \sum_{\substack{j\neq i\\ j\notin H_0}} \frac{\gamma c_j}{n-1}\overline{W}_j.$$

On the right hand side, the second term is zero since all the c_j are zero (since $j \in H_0$), and the third term is zero because all the $\overline{W}_j$ are zero (since $j \notin H_0$). Hence

$$\overline{W}_i' - \overline{W}_i = -\gamma c_i \overline{W}_i.$$

Now, if $i \in H_0$, then $c_i = 0$ and so $\overline{W}_i' - \overline{W}_i = 0$, and we are done. On the other hand, if $i \notin H_0$, then $\overline{W}_i = 0$ and again $\overline{W}_i' - \overline{W}_i = 0$.

To see that the assumed form of $\overline{\mathbf{W}}$ is necessary, suppose $i \in H_0$. Then $c_i = 0$, and so

$$\overline{W}_i' - \overline{W}_i = \sum_{\substack{j\neq i\\ j\notin H_0}} \frac{\gamma}{n-1} c_j \overline{W}_j.$$

In the summation, we have $c_j > 0$ since $j \notin H_0$. Hence, if we had $\overline{W}_j > 0$ for even one j, we would obtain that $\overline{W}_i' - \overline{W}_i \neq 0$. Hence, we must have $\overline{W}_j = 0$ for all $j \notin H_0$, and so $\overline{\mathbf{W}} \in \Delta_{H_0}$. □

[6] Just as in the two-grammar case, this definition gives rise to a set of linear operators. To see this, notice that the first equation, for instance, can be rewritten as

$$W_k' = W_k + \gamma(1-W_k) = W_k + \gamma \textstyle\sum_{i\neq k} W_i$$

so that W_k' is found to be a linear combination of $W_1, \ldots, W_n$. In fact, if I denotes the $n \times n$ identity matrix (i.e. I has 1 in each diagonal cell and 0 in each non-diagonal cell), J denotes the $n \times n$ matrix of ones (i.e. J has 1 in each cell), and E_{kk} denotes the matrix unit (i.e. E_{kk} has a 1 in cell (k,k) and is 0 elsewhere), then the learning operators can be written concisely as the following matrices:

$$\begin{cases} U_k^+ = (1-\gamma)I + \gamma E_{kk}J \\ U_k^- = (1-\gamma)I + \dfrac{\gamma}{n-1}(J - E_{kk}J) \end{cases}$$

[7] Note that (3) is a special case: we have

$$\overline{W}_1 = \frac{c_1^{-1}}{c_1^{-1}+c_2^{-1}} = \frac{c_1c_2c_1^{-1}}{c_1c_2(c_1^{-1}+c_2^{-1})} = \frac{c_2}{c_2+c_1}$$

and similarly for $\overline{W}_2$.

Corollary 1. *If the learning environment is parapunitive with $c_k = 0$ for a unique k, then $\overline{\mathbf{W}}$ tends to the vertex $\mathbf{e}_k$.*

Proof. This follows because $\Delta_{\{k\}}$ is a singleton: $\Delta_{\{k\}} = \Delta \cap \operatorname{span}\{\mathbf{e}_k\} = \{\mathbf{e}_k\}$. □

See Figure 1 for illustration.

Moving to the population level, we note that the learning environment μ again decomposes: for any $K \subseteq \Sigma^*$, we have

$$\mu(K) = \sum_j x_j \mu_j(K).$$

Hence

$$c_i = \mu(\complement L_i) = \sum_j x_j \mu_j(\complement L_i) = \sum_j a_{ij} x_j,$$

and so the penalty probabilities are linear combinations of the grammar abundances. Assuming as before a discrete sequence of infinite generations, we have

$$x_i' = \frac{c_i^{-1}}{\sum_j c_j^{-1}}$$

if $c_i > 0$ for all i, i.e. if the learning environment is omnipunitive at the population state $\mathbf{x}$. As will be explained in the following section, this is usually the case.

In particular, if omnipunitivity holds, then in the interior Δ° (i.e. when $x_i > 0$ for all i), the above equation may be expressed in more familiar terms: multiplying the numerator by $x_i^{-1}x_i$ and each summand in the denominator by $x_j^{-1}x_j$, we obtain

$$x_i' = \frac{x_i^{-1}c_i^{-1}}{\sum_j x_j x_j^{-1} c_j^{-1}} x_i = \frac{f_i(\mathbf{x})}{\sum_j x_j f_j(\mathbf{x})} x_i,$$

i.e.

$$x_i' = \frac{f_i(\mathbf{x})}{\varphi(\mathbf{x})} x_i,$$

where

$$f_i(\mathbf{x}) = x_i^{-1} c_i^{-1} = \frac{1}{\sum_j a_{ij} x_i x_j}$$

supplies the *fitness* of grammar G_i in the population state $\mathbf{x}$, and

$$\varphi(\mathbf{x}) = \sum_j x_j f_j(\mathbf{x})$$

is average fitness. This shows that the system is again characterized by a replicator dynamic, albeit

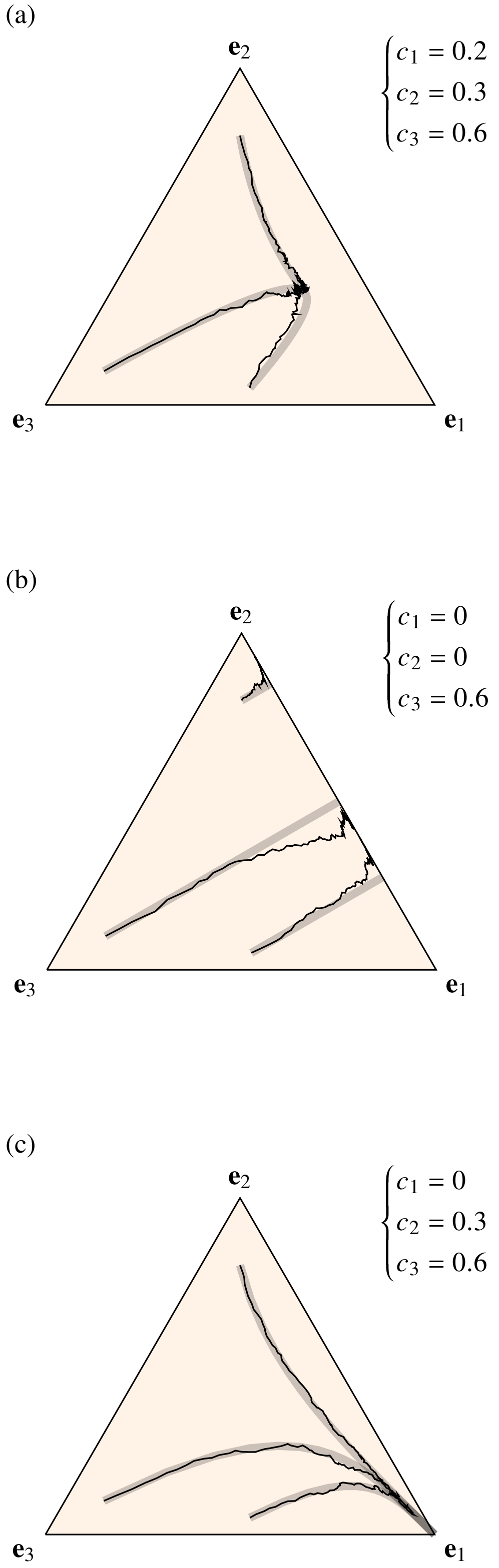


Figure 1: Learning in an omnipunitive (a) and two parapunitive (b, c) environments. Three learning trajectories are simulated in each case, starting from different initial states $\mathbf{W}$. In each case, the learning rate was set at $\gamma = 0.001$. The light thick lines show the corresponding mean dynamics.

in this more general case, fitness is frequency-dependent rather than constant. Note that the product $a_{ij}x_ix_j$ may be interpreted as the amount of motion away from grammar G_i in the direction of grammar G_j, explaining the appearance of these terms in the *denominator* of $f_i(\mathbf{x})$.

5 Global Asymptotic Dominance in *n*-grammar Competition

In the two-grammar case, Theorem 1 supplies a necessary and sufficient condition for one grammar to be an attractor. We now ask if a corresponding condition can be found in the more general case of *n*-grammar competition. The abundance vector $\mathbf{x}$, describing the abundances of the n grammars in a given generation of learners, belongs to the simplex $\Delta = \Delta^{n-1}$. At each vertex $\mathbf{e}_k \in \Delta$ of the simplex, one grammar claims all the abundance, and we define:

Definition 5. Grammar G_k is *dominant* if $\mathbf{e}_k$ is an equilibrium. It is *asymptotically dominant* if $\mathbf{e}_k$ is asymptotically stable. It is *globally asymptotically dominant*—abbreviated *g.a.d.*—if $\mathbf{e}_k$ is an attractor for any non-equilibrium initial state $\mathbf{x} \in \Delta$ with $x_k > 0$.

Clearly, Theorem 1 implies that, in the $n = 2$ case, G_k is g.a.d if and only if $a_{ik} > a_{ki}$. The restriction to initial states that satisfy $x_k > 0$ is a technicality which will be required in some of the proofs below; note that this assumption is innocuous since, empirically, convergence to a grammar can only occur after an innovation in the direction of this grammar has happened.

We moreover define:

Definition 6. Grammar G_i is *vulnerable* to grammar G_j if $a_{ij} > 0$. Grammar G_j is *non-submissive* if every other grammar G_i is vulnerable to it.

We then have the following result:

Theorem 3. *G_k is dominant only if it is non-submissive.*

Proof. Suppose G_k is not non-submissive. Then some G_i exists such that $a_{ik} = 0$. At the vertex $\mathbf{e}_k$, the penalty for this grammar G_i is

$$c_i = a_{ik}x_k = a_{ik} = 0.$$

Hence, G_k is not the only zero-penalty grammar, and so, by Theorem 2, $\overline{\mathbf{W}}$ does not generically converge to the vertex $\mathbf{e}_k$. In other words, in an environment in which only G_k is employed, a learner does not acquire full use of G_k. This implies that $\mathbf{e}_k$ is not an equilibrium and hence that G_k is not dominant. □

As an immediate consequence, we obtain that a non-submissive grammar cannot be globally asymptotically dominant:

Corollary 2. *G_k is g.a.d. only if it is non-submissive.*

In particular, the above results imply that no grammar G_i that stands in a subset relation to another grammar G_j, in the sense that $L_i \subseteq L_j$, can be (globally asymptotically) dominant.

Corollary 3. *Suppose $L_i \subseteq L_j$. Then G_i is not (globally asymptotically) dominant.*

Proof. $L_i \subseteq L_j$ implies that $a_{ji} = 0$. Hence G_j is not vulnerable to G_i and so G_i is not non-submissive. □

On the other hand, suppose a grammar G_k is non-submissive and that $a_{ki} = 0$ for all $i \neq k$. In this case, G_k is vulnerable to none of its competitors, while each of the latter is vulnerable to G_k. Let us call such a grammar *apical* (it is located at the "apex" of the competition situation).

Definition 7. Grammar G_k is *apical* if it is not vulnerable to any of its competitors but all its competitors are vulnerable to it.

An apical grammar is necessarily g.a.d.:

Theorem 4. *If G_k is apical, it is g.a.d.*

Proof. Let $\mathbf{x} \in \Delta$ such that $x_k > 0$. For all grammars G_i with $i \neq k$, the penalty is

$$c_i = \sum_j a_{ij}x_j \geq a_{ik}x_k > 0.$$

For G_k, the penalty is

$$c_k = \sum_j a_{kj}x_j = 0$$

on the assumption of apicality. Hence, by Theorem 2, G_k is g.a.d. (In fact, in this case, the vertex $\mathbf{e}_k$ is attained in one generation of learning.) □

We now continue to look for a general criterion for global asymptotic dominance. With the above results in mind, we can assume without loss of generality that a putative g.a.d. grammar G_k is both non-submissive (since if it were not non-submissive, it could not be g.a.d.) and non-apical (since if it were apical, it would necessarily be g.a.d.). On the assumption that G_k is non-submissive, we have $a_{ik} > 0$ for all $i \neq k$. Hence, whenever $x_k > 0$, we have

$$c_i = \sum_j a_{ij} x_j \geq a_{ik} x_k > 0.$$

On the other hand, since G_k is not apical, we have $a_{ki} > 0$ for at least one $i \neq k$, whereby

$$c_k = \sum_j a_{kj} x_j \geq a_{ki} x_i > 0$$

whenever $x_i > 0$. Hence, at least in the interior Δ°, every grammar has non-zero penalty, and so the learning environment is omnipunitive in all of the interior. Therefore, we have the well-posed population mean dynamic

$$x_i' = \frac{f_i(\mathbf{x})}{\varphi(\mathbf{x})} x_i$$

with

$$f_i(\mathbf{x}) = \frac{1}{\sum_j a_{ij} x_i x_j}$$

and $\varphi(\mathbf{x}) = \sum_j x_j f_j(\mathbf{x})$. By the following result, we can also assume without loss of any generality that the initial state $\mathbf{x}$ lies in the interior:

Lemma 1. *Suppose $\mathbf{x} \in \partial\Delta$ with $x_k > 0$ and that G_k is non-submissive. Then either $\overline{\mathbf{W}}$ converges to some point in the interior Δ° or it converges to the vertex $\mathbf{e}_k$.*

Proof. For any $i \neq k$, we have

$$c_i = \sum_j a_{ij} x_j \geq a_{ik} x_k > 0$$

on the assumption that G_k is non-submissive. Now, either $c_k > 0$ or $c_k = 0$. In the former case, every penalty probability is non-zero and so the learning environment is omnipunitive. By Theorem 2, $\overline{\mathbf{W}}$ converges to $\overline{\mathbf{W}}^* = c_i^{-1} / \sum_j c_j^{-1} \in \Delta^\circ$.

Then suppose $c_k = 0$. In this case, Corollary 1 implies that $\overline{\mathbf{W}}$ converges to the vertex $\mathbf{e}_k$. □

We now ask under what conditions trajectories starting at $\mathbf{x} \in \Delta^\circ$ converge to $\mathbf{e}_k$, so that G_k is g.a.d. (assuming, as above, that G_k is non-submissive and not apical). We define:

Definition 8. The *resilience* of grammar G_i against grammar G_j is $r_{ji} = 1/a_{ij}$, the inverse of the advantage that G_j holds over G_i. The *total resilience* of grammar G_i (against all its competitors) is defined as

$$R_i = \frac{1}{\sum_j a_{ij}},$$

the inverse of the cumulative advantage against G_i.

Of course, the resilience r_{ji} is only defined if $a_{ij} > 0$, i.e. only if G_i is vulnerable to G_j. In consequence, the total resilience R_i is only defined if G_i is vulnerable to at least one other G_j. Since a putative g.a.d. grammar G_k is assumed to be non-submissive and non-apical, the total resilience R_i is defined for all i, and the pairwise resiliences r_{ki} also exist for all i.

We point out that (total) resilience measures how robust a grammar G_i is against its competitor(s), i.e. it only references the advantage(s) of the latter over G_i, and not the advantage that G_i itself holds over its competitor(s).

We can now state our two main results. The first gives a sufficient condition for a grammar's global asymptotic dominance; the second, a necessary one. The proofs, which are tedious, are relegated to the Appendix.

Theorem 5. *Grammar G_k is g.a.d. if*

$$R_k > \sum_{i \neq k} r_{ki}$$

i.e. if its total resilience is greater than the sum of pairwise resiliences against it.

Theorem 6. *Grammar G_k is g.a.d. only if*

$$R_k > \frac{1}{n-1} \sum_{i \neq k} R_i$$

i.e. only if its total resilience is greater than the average total resilience across its competitors.

6 Grammar Flux

Theorem 5 enables an alternative characterization of sufficient conditions for a grammar's being g.a.d. This turns on the notion of a *chain* of grammars and its associated *flux*.

Definition 9. A *chain* of grammars is any triple (G_i, G_k, G_j) such that $i \neq k$ and $j \neq k$ (but possibly $i = j$). We denote this by $C_{ikj} = (G_i, G_k, G_j)$. A chain *traverses* grammar G_k if G_k is in the middle position of the chain. The *set of chains through* G_k is the set of all chains that traverse G_k.

Definition 10. The *flux* through a chain of grammars C_{ikj} is defined as the ratio

$$\varphi(C_{ikj}) = \frac{a_{kj}}{a_{ik}}$$

whenever $a_{ik} > 0$.

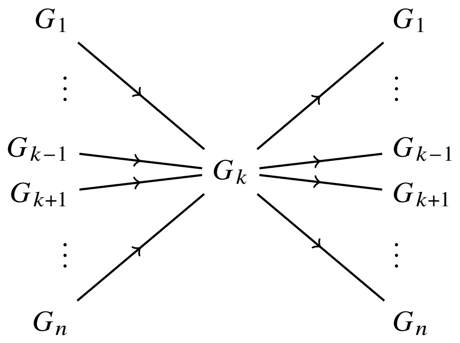


Figure 2: The total flux through grammar G_k, $\Phi(G_k)$, is defined by adding together the fluxes over all chains that traverse G_k.

If we can imagine abundance as "flowing" from one grammar to another, then $\varphi(C_{ikj})$ measures how much abundance, obtained from G_i, grammar G_k lets flow through to G_j. The better G_k is at resisting losing advantage to G_j under "pressure" from (advantage flowing from) G_i, the lower the flux.

We further define (cf. Figure 2):

Definition 11. The *total flux through grammar* G_k, denoted $\Phi(G_k)$, is the sum of fluxes over all chains through G_k, i.e. the quantity

$$\Phi(G_k) = \sum_{i \neq k} \sum_{j \neq k} \varphi(C_{ikj}).$$

A sufficient condition for g.a.d. is then the following: the total flux through a grammar is bounded from above by one.

Theorem 7. *Grammar G_k is g.a.d. if $\Phi(G_k) < 1$.*

Proof. By Theorem 5, G_k is g.a.d. if

$$\frac{1}{\sum_j a_{kj}} > \sum_{i \neq k} \frac{1}{a_{ik}}.$$

Since $a_{kk} = 0$, this is equivalent to

$$\frac{1}{\sum_{j \neq k} a_{kj}} > \sum_{i \neq k} \frac{1}{a_{ik}}.$$

Multiplying both sides by $\sum_{j \neq k} a_{kj}$, we obtain

$$1 > \sum_{j \neq k} a_{kj} \sum_{i \neq k} \frac{1}{a_{ik}} = \sum_{j \neq k} \sum_{i \neq k} \frac{a_{kj}}{a_{ik}} = \Phi(G_k)$$

as wished. □

A loose analogy exists between the notion of grammar flux and elementary properties of electronic circuits that may assist in the former's interpretation. By Ohm's Law, the current passed by a component, I, equals the voltage applied to the component, V, divided by its resistance, R. From this, one obtains $R^{-1} = I/V$, where R^{-1}, the inverse of resistance, measures the component's conductance, i.e. how many units of current it passes per unit of voltage applied. The flux through a grammar—even though a dimensionless number—similarly measures how much abundance a grammar "leaks" under the pressure of abundance flowing into it; Theorem 7 shows that, in order for a grammar to be globally asymptotically dominant, it is enough for this grammar to be a sufficiently poor conductor of abundance.

7 Discussion

We have studied the inter-generational dynamics of the general n-grammar variational learning model, asking the following question: under what conditions is a single grammar globally asymptotically dominant (g.a.d.), meaning that this grammar attracts from any non-equilibrium initial state once the grammar has been innovated? We have proved two main results, each turning on the notion of a grammar's resilience. The first result gives a sufficient condition, stating that grammar G_k is g.a.d. if its total resilience is greater than the cumulative pairwise resiliences of its competitors,

$$R_k > \sum_{i \neq k} r_{ki}.$$

The second result gives a necessary condition, and states that if G_k is g.a.d., then necessarily

$$R_k > \frac{1}{n-1} \sum_{i \neq k} R_i,$$

i.e. the total resilience of G_k is greater than the average total resilience computed over all its competitors. An alternative characterization by way of the notion of the flux through a grammar found that the sufficient condition is equivalent to $\Phi(G_k) < 1$, which states that the total flux through G_k is bounded from above by 1.

These results may now be applied to empirical data to evaluate the merits of this particular model of language acquisition and language change. The estimation of advantage parameters from corpora is by now commonplace in the two-grammar special case (e.g. Yang, 2000; Heycock and Wallenberg, 2013; Simonenko et al., 2019). In principle, estimation of such parameters in multidimensional competition is no different, although it is more

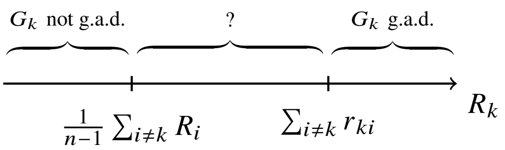


Figure 3: Necessary and sufficient bounds on R_k for grammar G_k to be globally asymptotically dominant.

laborious due to the greater number of pairwise comparisons required. The sufficient condition for global asymptotic dominance may then be used to predict whether a given set of estimates for the a_{ij} parameters (a particular advantage matrix A) leads to eventual dominance by one grammar. Conversely, the necessary condition may be used to argue that a particular choice of A could never lead to a grammar's being g.a.d. In practice, application of the sufficient condition for purposes of arguing for a grammar's g.a.d. status may be more useful than using the necessary condition to argue against a grammar's g.a.d. status—a particular advantage matrix may fail the necessary condition, and hence imply that a grammar is not formally g.a.d., yet the system's stable equilibrium may still be arbitrarily close to the vertex representing full use of this grammar.

The sufficient and necessary conditions of global asymptotic dominance reference the competing grammars' pairwise and total resiliences; in particular, both give a lower bound for the total resilience R_k of a candidate grammar. It is important to point out that the bounds for R_k established here do not coincide (except in the special case $n = 2$). In other words, a region exists in which mere examination of the magnitude of R_k does not (yet) tell us whether G_k is g.a.d. or not (see Figure 3). This is because the proofs of Theorems 5–6 rely on simple bounding arguments; in particular, the proof of the sufficient condition in fact establishes the stronger claim that convergence to dominance is strictly monotonic, in the sense that the abundance of the g.a.d. grammar always increases. If a trajectory converging on a vertex is non-monotonic, this is not captured by Theorem 5. Future work should thus look for a more complete characterization of the conditions under which convergence to the vertices occurs.

These results were obtained for a particular model of language change obtained from a particular model of language learning through a particular set of assumptions. We have assumed a discrete sequence of non-overlapping generations, each of which consists of infinitely many well-mixing, identically learning speakers. In other words, we have only studied the deterministic limit in which the system's mean dynamic is a faithful description of the system's evolution. Any of these simplifying assumptions could in principle be lifted, resulting in a stochastic process also at the inter-generational population level.

On the other hand, ample possibilities exist for future work even in the deterministic limit. We have here assumed, with tradition, that the pairwise advantages a_{ij} remain constant for the duration of any evolutionary process we may be interested in observing and modelling. This is not necessarily so in the real world. Interesting extensions of the model occur when the a_{ij} are allowed to be frequency-dependent, i.e. to depend on the current abundance vector $\mathbf{x}$ obtaining in the population. These await formal study.

Finally, we point out that the operators in (5) are but one out of many possible ways of generalizing the two-grammar learning algorithm for $n > 2$ grammars. In particular, this choice of operators implies that, whenever a grammar is punished, then *all* remaining $n - 1$ grammars are rewarded. Yet this is clearly not a necessary feature of a model of language learning, and may not be particularly realistic. An alternative model might equip the learner with a short-term memory, which would be used to reward only grammars employed by the learner in the very near past, for example. The effects such modifications have on the population-level evolution remain to be studied.

Acknowledgements

The work reported here has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement n° 851423). I wish to thank Fernanda Barrientos, Caroline Heycock, Mara Maza-Cuello, Gemma McCarley, Raquel Montero Estebaranz, Molly Rolf, George Walkden, Joel Wallenberg and Charles Yang for discussions on the topic of grammar competition and variational learning; as well as the three reviewers for their helpful comments. I am particularly grateful to Mara Maza-Cuello for discovering and helping to rectify an embarrassing mistake in an earlier version of the proof of Theorem 6. Any remaining errors, of course, are my sole responsibility.

References

Henning Andersen. 1973. Abductive and deductive change. *Language*, 49(4):765–793.

Robert R. Bush and Frederick Mosteller. 1955. *Stochastic models for learning*. Wiley, New York, NY.

Caroline Heycock and Joel Wallenberg. 2013. How variational acquisition drives syntactic change: the loss of verb movement in Scandinavian. *Journal of Comparative Germanic Linguistics*, 16:127–157.

Henri Kauhanen. 2019. Stable variation in multidimensional competition. In Anne Breitbarth, Miriam Bouzouita, Lieven Danckaert, and Melissa Farasyn, editors, *The Determinants of Diachronic Stability*, pages 263–290. Benjamins, Amsterdam.

Anthony S. Kroch. 1989. Reflexes of grammar in patterns of language change. *Language Variation and Change*, 1(3):199–244.

David J. C. MacKay. 2003. *Information theory, inference, and learning algorithms*. Cambridge University Press, Cambridge.

Kumpati S. Narendra and Mandayam A. L. Thathachar. 1989. *Learning automata: an introduction*. Prentice-Hall, Englewood Hills, NJ.

Partha Niyogi. 2006. *The computational nature of language learning and evolution*. MIT Press, Cambridge, MA.

M. Frank Norman. 1972. *Markov processes and learning models*. Academic Press, New York, NY.

William H. Sandholm. 2010. *Population games and evolutionary dynamics*. MIT Press, Cambridge, MA.

Alexandra Simonenko, Benoit Crabbé, and Sophie Prévost. 2019. Agreement syncretization and the loss of null subjects: quantificational models for Medieval French. *Language Variation and Change*, 31(3):275–301.

Uriel Weinreich, William Labov, and Marvin I. Herzog. 1968. Empirical foundations for a theory of language change. In Winfred P. Lehmann and Yakov Malkiel, editors, *Directions for historical linguistics: a symposium*, pages 95–195. University of Texas Press, Austin, TX.

Charles D. Yang. 2000. Internal and external forces in language change. *Language Variation and Change*, 12:231–250.

Charles D. Yang. 2002. *Knowledge and learning in natural language*. Oxford University Press, Oxford.

A Proofs

Proof of Theorem 5. Let $\mathbf{x} \in \Delta^\circ$. We look for a condition under which $x'_k > x_k$. If this holds for arbitrary $\mathbf{x} \in \Delta^\circ$, then x_k increases in the entire interior of the simplex, which suffices to show that G_k is g.a.d.

Now, $x'_k > x_k$ if

$$f_k(\mathbf{x}) > \sum_i x_i f_i(\mathbf{x}).$$

Write $x_k = 1 - \delta$ for $\delta > 0$. Then the above inequality becomes

$$f_k(\mathbf{x}) > (1-\delta) f_k(\mathbf{x}) + \sum_{i \neq k} x_i f_i(\mathbf{x}),$$

or

$$\delta f_k(\mathbf{x}) > \sum_{i \neq k} x_i f_i(\mathbf{x}).$$

Expanding the fitnesses, we have

$$\frac{\delta}{\sum_i a_{ki}(1-\delta)x_i} > \sum_{i \neq k} \frac{x_i}{\sum_j a_{ij} x_i x_j},$$

i.e.

$$\frac{\delta}{\sum_i a_{ki} x_i} > (1-\delta) \sum_{i \neq k} \frac{1}{\sum_j a_{ij} x_j}. \tag{7}$$

Since $a_{kk} = 0$, this is equivalent to

$$\frac{\delta}{\sum_{i \neq k} a_{ki} x_i} > (1-\delta) \sum_{i \neq k} \frac{1}{\sum_j a_{ij} x_j}. \tag{8}$$

Since $x_k = 1 - \delta$, it follows that $x_i \leq \delta$ for $i \neq k$. Thus, the left-hand side is bounded from below; in fact, it is bounded from below by R_k:

$$\frac{\delta}{\sum_{i \neq k} a_{ki} x_i} \geq \frac{\delta}{\sum_{i \neq k} a_{ki} \delta} = \frac{1}{\sum_{i \neq k} a_{ki}} = R_k.$$

Hence, to establish (8), it suffices to show that

$$R_k > (1-\delta) \sum_{i \neq k} \frac{1}{\sum_j a_{ij} x_j}.$$

Working on the right-hand side, we find

$$\begin{aligned} \sum_{i \neq k} \frac{1-\delta}{\sum_j a_{ij} x_j} &= \sum_{i \neq k} \frac{1-\delta}{a_{ik}(1-\delta) + \sum_{j \neq k} a_{ij} x_j} \\ &= \sum_{i \neq k} \frac{1}{a_{ik} + \frac{1}{1-\delta} \sum_{j \neq k} a_{ij} x_j}, \end{aligned}$$

which is obviously bounded from above:

$$\sum_{i \neq k} \frac{1}{a_{ik} + \frac{1}{1-\delta} \sum_{j \neq k} a_{ij} x_j} < \sum_{i \neq k} \frac{1}{a_{ik}} = \sum_{i \neq k} r_{ki}.$$

Hence, the condition

$$R_k > \sum_{i \neq k} r_{ki}$$

implies that $x'_k > x_k$ everywhere in the interior of Δ, which implies that G_k is g.a.d. □

Proof of Theorem 6. We in fact show the following stronger result: if $R_k < \sum_{i \neq k} R_i/(n-1)$, the vertex $\mathbf{e}_k$ is not asymptotically stable. From this it immediately follows that G_k is not g.a.d.

To prove that $\mathbf{e}_k$ is not asymptotically stable, we show that, in (possibly infinitesimally small) local environments of the vertex, $x'_k < x_k$, so that $\mathbf{x}$ moves away from $\mathbf{e}_k$. Let $\mathbf{x} \in \Delta^\circ$ such that $x_k = 1 - \delta$ for $\delta > 0$. The condition $x'_k < x_k$ is equivalent to

$$\frac{\delta}{1-\delta} \cdot \frac{1}{\sum_i a_{ki} x_i} < \sum_{j \neq k} \frac{1}{\sum_i a_{ji} x_i}. \tag{9}$$

(cf. the derivation of (7) in the proof of Theorem 5). Let $m = \min_{i \neq k}\{x_i\}$. Then $x_i \geq m$ for all $i \neq k$, and so

$$\frac{1}{\sum_i a_{ki} x_i} \leq \frac{1}{m \sum_i a_{ki}}.$$

Hence, to show (9), it suffices to show that

$$\frac{\delta}{1-\delta} \cdot \frac{1}{m} \cdot \frac{1}{\sum_i a_{ki}} < \sum_{j \neq k} \frac{1}{\sum_i a_{ji} x_i}. \tag{10}$$

Suppose δ is small ($\mathbf{x}$ is close to $\mathbf{e}_k$). Then $x_i < 1-\delta$ for all $i \neq k$. We have

$$\sum_i a_{ji} x_i = a_{jk}(1-\delta) + \sum_{i \neq k} a_{ji} x_i,$$

and so (10) is equivalent to

$$\frac{\delta}{m} \cdot \frac{1}{\sum_i a_{ki}} < \sum_{j \neq k} \frac{1}{a_{jk} + \frac{1}{1-\delta} \sum_{i \neq k} a_{ji} x_i}. \tag{11}$$

Since $x_i < 1 - \delta$ (for $i \neq k$), the right-hand side is bounded from below by

$$\sum_{j \neq k} \frac{1}{a_{jk} + \frac{1}{1-\delta} \sum_{i \neq k} a_{ji}(1-\delta)} = \sum_{j \neq k} \frac{1}{\sum_i a_{ji}}.$$

Hence, to show (11), it suffices to show that

$$\frac{\delta}{m} \cdot \frac{1}{\sum_i a_{ki}} < \sum_{j \neq k} \frac{1}{\sum_i a_{ji}}$$

i.e. that

$$\frac{1}{\sum_i a_{ki}} < \frac{m}{\delta} \sum_{j \neq k} \frac{1}{\sum_i a_{ji}},$$

i.e.

$$R_k < \frac{m}{\delta} \sum_{j \neq k} R_j.$$

Let us rewrite this last inequality:

$$1 - \delta > 1 - m\rho_k,$$

where $\rho_k = \sum_{j \neq k} R_j / R_k$. Since $x_k = 1 - \delta$, we thus have

$$x_k > 1 - m\rho_k.$$

Now, since $\mathbf{x} \in \Delta$ and since $m = \min_{i \neq k}\{x_i\}$, we have

$$\begin{aligned} 1 &= x_k + \sum_{i \neq k} x_i \\ &\geq x_k + (n-1)m \\ &> 1 - m\rho_k + (n-1)m, \end{aligned}$$

which is equivalent to

$$\rho_k > n - 1.$$

In other words,

$$\sum_{j \neq k} \frac{R_j}{R_k} > n - 1,$$

or

$$R_k < \frac{1}{n-1} \sum_{j \neq k} R_j. \tag{12}$$

To recap: if condition (12) holds, then some neighbourhood of $\mathbf{e}_k$ exists in Δ° in which the value of x_k increases. This implies that $\mathbf{e}_k$ is not asymptotically stable; *a fortiori*, G_k is not g.a.d. □

A Full Numerical Rank Basis Selection Algorithm for Spectral Learning

Yibin Zhao
University of Toronto
ybzhao@cs.toronto.edu

Gerald Penn
University of Toronto
gpenn@cs.toronto.edu

Abstract

This paper re-visits the basis selection methods for Hankel matrices as in spectral learning methods. The known algorithms have many limitations, one being that no such algorithm could find a full numerical rank sub-block. In this paper, we present an efficient, deterministic divide-and-conquer algorithm that finds a full numerical rank basis. Our solution is especially appealing due to its stable behaviour and its versatility in training Hidden Markov Models. We also provide two parameters for controlling the trade-off between computational complexity and model performance.

1 Introduction

Spectral learning methods have attracted the attention of many researchers in computational linguistics (CL) and the natural language processing (NLP) community due to the strong theoretical properties of the algorithms and the advantages of Hidden Markov Models in processing sequences (Hsu et al., 2012; Balle et al., 2011; Cohen et al., 2012).

It has been challenging to obtain language-modelling results with spectral methods that are competitive with neural methods mainly due to two issues, as pointed out by Quattoni and Carreras (2019): the scalability of handling long-range dependencies, and the loss function for the spectral learning scheme does not align with that of the language modelling task. For the first issue, a natural solution is to increase the length of sequences for learning the distribution. As a consequence, the size of the Hankel matrix increases so drastically that it is not practical to perform factorizations (i.e. singular value decomposition) on it. Hence, people seek to find informative yet much smaller sub-blocks of the Hankel matrices, induced by a set of prefixes and a set of suffixes, to perform the matrix factorizations. We call this *basis selection* for Hankel matrices.

Over the past two decades, many basis selection methods have been proposed. One basic approach is to choose all the prefixes and suffixes observed in the sample (Bailly et al., 2009). Although this method has many good theoretical properties (Denis et al., 2016) for modelling a probability and is always optimal in terms of loss, it is practically infeasible for large corpora. Previously, the most popular approach was to choose prefixes and suffixes of length $\leq T$ (Hsu et al., 2012; Siddiqi et al., 2010). Bases chosen by this method tend to lose too much information, making it hard to model long-term dependencies. Balle et al. (2012) provided a randomized algorithm by cutting a string at each time step over n iterations, resulting in an $n \times n$ sub-block. When n is sufficiently large, there is a high probability the algorithm returns a full-rank sub-block of the empirical Hankel matrix. But n must be $\mathrm{polylog}\ n$ times the rank of a complete Hankel matrix in order to obtain a sub-block with nearly full rank.

The work of Quattoni et al. (2017) proposed an algorithm that performs maximum matching on the underlying bipartite graphs realized by Hankel matrices to get a full structural rank sub-block. The bases generated by this method are very compact. This method was the first to exploit the structural properties of Hankel matrices in this task. The authors claimed that bases generated by bipartite matching have approximately the full numerical rank of a complete empirical Hankel matrix. But there is no theoretical analysis, and the matching properties that they used in claiming the rank observation do not hold in most cases.

In this paper, we present an efficient algorithm for finding a compact, full-rank basis of an empirical Hankel matrix in a divide-and-conquer approach. Our method recursively computes the basis for sets of sub-strings with a fixed first symbol. The algorithm then combines such computed bases for each first symbol in the alphabet. We also present a trade-off between the run-time of the algorithm and the size of the resulting basis through the choice of parameters called α and γ. Due to the nice theoretical properties of a full numerical rank sub-block (Denis et al., 2016) in learning a weighted automaton and the limitation of each established method mentioned above, we believe that our basis selection method is very competitive and can be an excellent complement to the method of Quattoni et al. (2017) for improving the accuracy of language models without overly increasing the run time for both training and testing.

2 Preliminaries

2.1 Notation

We start by introducing the notation we use throughout this paper. Let Σ be a finite alphabet with $|\Sigma|$ symbols and let ϵ denote the null sequence. We use Σ^* to denote the set of all strings over Σ. The concatenation of two strings $x, x' \in \Sigma^*$ is denoted by $x \cdot x'$. $\mathcal{P}$ is a set of unique prefixes and $\mathcal{S}$ is a set of unique suffixes in sample W. We always assume $\epsilon \in \mathcal{P}$ and $\epsilon \in \mathcal{S}$ unless stated otherwise. For some strings q and t, we define $\mathcal{P}(q) = \{p \in \mathcal{P} | \exists v \in \Sigma,^* p = q \cdot v\}$ and $\mathcal{S}(t) = \{s \in \mathcal{S} | \exists u \in \Sigma, *s = u \cdot a\}$. For a suffix t, the set of all prefixes of strings with suffix t is denoted by $\mathcal{P}^*(t) = \{p \in \mathcal{P} | \exists s \in \mathcal{S}(t), p \cdot s \in W\}$. We similarly define $\mathcal{S}^*(q) = \{s \in \mathcal{S} | \exists p \in \mathcal{P}(q), p \cdot s \in W\}$. Let a be some symbol in Σ, we define $\mathcal{P}_a = \{v \in \Sigma^* | a \cdot v \in \mathcal{P}\}$ and $\mathcal{S}_a = \{u \in \Sigma^* | u \cdot a \in \mathcal{S}\}$. Note that both $\mathcal{P}_a$ and $\mathcal{S}_a$ contain ϵ. Also, $\mathcal{S}^*_a = \{s \in \mathcal{S} | \exists p \in \mathcal{P}_a, a \cdot p \cdot s \in W\}$ and $\mathcal{P}^*_a = \{p \in \mathcal{P} | \exists s \in \mathcal{S}_a, p \cdot s \cdot a \in W\}$.

For a sequence $x = x_1 \cdots x_n \in \Sigma,^*$ we let $x^\top = x_n \cdots x_1$ denote the reversed sequence of x. If we define $W^\top = \{w^\top | w \in W\}$ as the multi-set of reverse sequences, then $\mathbf{H}(W^\top) = \mathbf{H}^\top(W)$. In addition, we let $\mathcal{P}^\top = \{p^\top | p \in \mathcal{P}\}$ and $\mathcal{S}^\top = \{s^\top | s \in \mathcal{S}\}$. $\mathcal{S}^\top$ and $\mathcal{P}^\top$ are the respective prefix and suffix set of sample $W^\top$.

For a matrix $\mathbf{M}$, we denote the ith row of $\mathbf{M}$ by $\mathbf{M}(i,:)$, the jth column by $\mathbf{M}(:,j)$. For $i \geq 1$, we use $\boldsymbol{e}_i$ to denote the standard basis vector with the i-th entry as 1 and 0 otherwise.

2.2 Non Deterministic Weighted Finite State Automata

Here, we define a class of Non-Deterministic Weighted Automaton (WA) over strings. Let $x = x_1 \cdots x_n$ be a sequence of length n over finite alphabet Σ. A WA with k states is defined as a tuple: $A = (\boldsymbol{\alpha}_0, \boldsymbol{\alpha}_\infty, \{\mathbf{A}_\sigma\}_{\sigma \in \Sigma})$ where $\boldsymbol{\alpha}_0, \boldsymbol{\alpha}_\infty \in \mathbb{R}^k$ are the initial and final weight vectors and $\mathbf{A}_\sigma \in \mathbb{R}^{k \times k}$ are the transition matrices associated with each letter $\sigma \in \Sigma$. The function $f_A : \Sigma^* \to [0,1]$ realized by a WA A is defined as:

$$f_A(x) = \boldsymbol{\alpha}_0^\top \mathbf{A}_{x_1} \cdots \mathbf{A}_{x_n} \boldsymbol{\alpha}_\infty^\top. \qquad (1)$$

The equation above is an algebraic representation of the computation performed by a WA on a sequence x. For an explanation of this observation, we refer readers to (Hsu et al., 2012; Balle et al., 2012; Quattoni et al., 2017).

A stochastic language $\mathcal{L} : \Sigma^* \to \mathbb{R}$ is a probability distribution over $\Sigma.^*$ If for every $x \in \Sigma,^* \mathcal{L}(x) > 0$, we say that $\mathcal{L}$ has full-support. We say $\mathcal{L}$ is rational when there exists a WA A such that $f_A(x) = \mathcal{L}(x)$ for any sequence x. For our purpose, we assume the stochastic language that we sample from is rational and has full-support. We refer readers to (Denis et al., 2016; Balle et al., 2012) for a rigorous discussion of stochastic languages in relation to WA and spectral learning.

2.3 Hankel Matrices and Underlying Graphs

We now introduce the concept of a Hankel matrix in WA. This paper focuses on Hankel matrices in the context of learning a stochastic language $\mathcal{L}$ from sample W.

The Hankel matrix of $\mathcal{L}$ is a bi-infinite matrix $\mathbf{H}_\mathcal{L} : \Sigma^* \times \Sigma^* \to \mathbb{R}$ with entries indexed by prefixes and suffixes. The rank of $\mathcal{L}$ is defined as $\mathrm{rank}(\mathcal{L}) = \mathrm{rank}(\mathbf{H}_\mathcal{L})$. Given $\mathcal{P}, \mathcal{S} \in \Sigma,^*$ a Hankel sub-block is denoted by $\mathbf{H}_\mathcal{L}^{(\mathcal{P},\mathcal{S})} \in \mathbb{R}^{|\mathcal{P}| \times |\mathcal{S}|}$. The rows and columns of $\mathbf{H}_\mathcal{L}^{(\mathcal{P},\mathcal{S})}$ are indexed by $\mathcal{P}$ and $\mathcal{S}$ with $\mathbf{H}_\mathcal{L}^{(\mathcal{P},\mathcal{S})}(p,s) = \mathcal{L}(p \cdot s)$. We say that $(\mathcal{P}, \mathcal{S})$ is a basis for $\mathcal{L}$ if $\mathrm{rank}(\mathcal{L}) = \mathrm{rank}(\mathbf{H}_\mathcal{L})$. For any symbol $\sigma \in \Sigma$, we define $\mathbf{H}_\sigma \in \mathbb{R}^{|\mathcal{P}| \times |\mathcal{S}|}$ as $\mathbf{H}_\sigma(p,s) = \mathcal{L}(p \cdot \sigma \cdot s)$.

In the context of learning, the *empirical* Hankel matrix of a finite multi-set W sampled $i.i.d.$ from $\mathcal{L}$ is denoted by $\mathbf{H}(W)$, or simply, $\mathbf{H}$. For any string $x \in \Sigma,^*$ we define $\mathbf{H}_x$ by $\mathbf{H}_x(p,s) = \mathbf{1}_{p \cdot s = x}$ and $\mathbf{H} = \frac{1}{|W|} \sum_{w \in W} \mathbf{H}_w$. Since W is finite, we can always find a minimal $\mathcal{P}, \mathcal{S} \in \Sigma^*$ such that $\mathbf{H}^{(\mathcal{P},\mathcal{S})}$ contains all the non-trivial entries of $\mathbf{H}(W)$. We say this matrix is the complete (empirical) Hankel matrix. In addition, we define $\boldsymbol{h}_x(x') = \mathbf{1}_{x'=x}$, and $\boldsymbol{h}_\mathcal{P} \in \mathbb{R}^{|\mathcal{P}|}, \boldsymbol{h}_\mathcal{S} \in \mathbb{R}^{|\mathcal{S}|}$ by $\boldsymbol{h}_\mathcal{P}(p) = \frac{1}{|W|} \sum_{w \in W} \boldsymbol{h}_w(p)$ and $\boldsymbol{h}_\mathcal{S}(s) = \frac{1}{|W|} \sum_{w \in W} \boldsymbol{h}_w(s)$. For any $\sigma \in \Sigma$, we define

$\mathbf{H}_\sigma \in \mathbb{R}^{|\mathcal{P}|\times|\mathcal{S}|}$ by $\mathbf{H}_\sigma(p, s) = \mathbf{H}(p, \sigma \cdot s)$. Notice that $\mathbf{H}(p, \sigma\cdot s) = \mathbf{H}(p\cdot\sigma, s)$ as $p\cdot(\sigma\cdot s) = (p\cdot\sigma)\cdot s$. If the sub-block Hankel matrix $\mathbf{H}^{(\mathcal{P}^*,\mathcal{S}^*)}$ induced by $\mathcal{P}^* \subseteq \mathcal{P}$ and $\mathcal{S}^* \subseteq \mathcal{S}$ satisfies $\operatorname{rank}(\mathbf{H}^{(\mathcal{P}^*,\mathcal{S}^*)}) = \operatorname{rank}(\mathbf{H}^{(\mathcal{P},\mathcal{S})})$, we say that $(\mathcal{P}^*, \mathcal{S}^*)$ is a *basis* for $\mathbf{H}(W)$. We denote the row of $\mathbf{H}$ induced by a prefix p by $\mathbf{H}(p, :)$, and the column induced by a suffix s by $\mathbf{H}(:, s)$.

Let $G = (\mathcal{P}, \mathcal{S}, E, \mu)$ denote the underlying bipartite graph represented by $\mathbf{H}^{(\mathcal{P},\mathcal{S})}$, where $p\cdot s \in E$ if and only if $\mathbf{H}^{(\mathcal{P},\mathcal{S})}(p, s) \neq 0$ and $\mu : E \to \mathbb{N}$ is the weight function such that $\mu(ps) = \mathbf{H}^{(\mathcal{P},\mathcal{S})}(p, s)$. Since all information in the 4-tuple can be directly derived from W, we also use $G(W)$ to denote such a graph. $d_G(v)$ is the degree of a vertex $v \in V(G)$. Let $N_G(v)$ denote the neighbours of v in G, $|N_G(v)| = d_G(v)$. We also use $N_G(V)$ to denote the union of neighbours of vertices in V. Without specifying, we assume G is the graph by default and use $d(v), N(v), N(V)$ as short forms instead. Supposing V is a set with $v \in V$, we use $V - v$ as a short form to denote $V \setminus \{v\}$.

In the context of an underlying graph, $\mathcal{S}^*(q)$ and $\mathcal{P}^*(t)$ correspond to $N(\mathcal{P}(q))$ and $N(\mathcal{S}(t))$, respectively. Note that, while $\mathcal{S}^*(q) = N(\mathcal{P}(q))$ in graph G, $\mathcal{P}(q)$ is likely not to be equal to $N(\mathcal{S}^*(q))$. Alternatively, if we let $W^a_p = \{w^a_p \in \Sigma^* | a\cdot w^a_p \in W\}$ and $W^a_s = \{w^a_s \in \Sigma^* | w^a_s \cdot a \in W\}$ be two multisets of samples induced from the original sample W and let $H = G(W^a_p)$ and $I = G(W^a_s)$ be the bipartite graphs built from these samples, we then have $\mathcal{S}^*_a = N_H(\mathcal{P}_a)$ and $\mathcal{P}^*_a = N_I(\mathcal{S}_a)$. Moreover, $\mathcal{S}^*_a$ is essentially the same as $\mathcal{S}^*(a)$.

2.4 The spectral learning method

We now give a description of the spectral learning algorithm. Given a training set of samples W and a parameter $k \in \mathbb{N}^+$, the spectral algorithm computes a WA A with k states. We refer readers to (Hsu et al., 2012) for the theory behind this algorithm:

1. Select a sub-block $(\mathcal{P}, \mathcal{S})$ of the complete Hankel matrix, denoted by $\mathbf{H}^{(\mathcal{P},\mathcal{S})}$ or $\mathbf{H}$.

2. Compute Hankel matrices for the sub-block selection $(\mathcal{P}, \mathcal{S})$, including $\mathbf{H}$, $\boldsymbol{h}_\mathcal{P}$, $\boldsymbol{h}_\mathcal{S}$ and $\mathbf{H}_\sigma$ for all $\sigma \in \Sigma$.

3. Compute the k-rank factorization of $\mathbf{H}$ through SVD (i.e. $\mathbf{H} \approx \mathbf{U\Sigma V}^\top$). Define forward matrix $\mathbf{F} = \mathbf{U\Sigma} \in \mathbb{R}^{|\mathcal{P}|\times k}$ and backward matrix $\mathbf{B} = \mathbf{V} \in \mathbb{R},^{|\mathcal{S}|\times k}$ then $\mathbf{H} \approx \mathbf{FB}^\top$ is a k-rank factorization.

4. Build the WA A of k states. The model parameters are computed by:
   (a) Initial vector $\boldsymbol{\alpha}_0^\top = \boldsymbol{h}_\mathcal{S}^\top \mathbf{B}$,
   (b) Final vector $\boldsymbol{\alpha}_\infty^\top = \mathbf{F}^+ \boldsymbol{h}_\mathcal{P}$,
   (c) Transition matrices $\mathbf{A}_\sigma = \mathbf{F}^+ \mathbf{H}_\sigma \mathbf{B}$.

Note that we use $\mathbf{M}^+$ to denote the Moore-Penrose pseudo-inverse of a matrix $\mathbf{M}$. The computation is dominated by SVD, which has a run-time complexity of $O(\min(|\mathcal{P}|, |\mathcal{S}|)^3)$. In Section 4.4.2, we will present a trade off between run-time and the resulting sub-block size in our basis selection algorithm. This, in turn, affects the run-time of the SVD step.

3 Properties of Hankel matrices

We will show some general properties regarding a Hankel matrix with arbitrarily large but finite sets of alphabet and samples. For the following propositions, we denote the prefix and suffix of a complete Hankel matrix over samples W, $\mathbf{H}$, by $\mathcal{P}$ and $\mathcal{S}$ respectively.

Proposition 1. [1] *Let $P^* \subseteq \mathcal{P}$ and $S^* \subseteq \mathcal{S}$ be spanning sets of row and column spaces, respectively. The rank of the sub-block Hankel matrix $\mathbf{H}^{(P^*,S^*)}$ is equal to the rank of $\mathbf{H}^{(\mathcal{P},\mathcal{S})}$.*

As we present in later sections, our algorithm finds a row spanning set for the complete Hankel matrix. To get a column spanning set, simply perform the same algorithm on its transposition. Combining these two sets thus results in a basis of the complete Hankel matrix.

Proof. Suppose $\operatorname{rank}(\mathbf{H}^{(P,^*S^*)}) < \operatorname{rank}(\mathbf{H}^{(\mathcal{P},\mathcal{S})})$. Since P^* spans the row space, $\operatorname{rank}(\mathbf{H}^{(P,^*\mathcal{S})}) = \operatorname{rank}(\mathbf{H}^{(\mathcal{P},\mathcal{S})})$. There must exist a column induced by a suffix $s \notin S,^*$ $\mathbf{H}^{(P,^*\mathcal{S})}(:, s)$, that is not a linear combination of columns of $\mathbf{H}^{(P,^*S^*)}$. Since S^* spans the column space, $\mathbf{H}^{(\mathcal{P},\mathcal{S})}(: s)$ is a linear combination of columns of $\mathbf{H}.^{(\mathcal{P},S^*)}$ This means that $\mathbf{H}^{(P,^*\mathcal{S})}(:, s)$ must also be a linear combination of columns of $\mathbf{H},^{(P,^*S^*)}$ which creates a contradiction. □

The following two propositions point out a crucial insight into reducing the size of the sub-block basis without losing rank.

[1] A stronger statement of this proposition is true for any matrix.

Proposition 2. *Let $U \subseteq \mathcal{P}$ and $V \subseteq \mathcal{S}$ be the sets of vertices in G such that $\forall u \in U, d(u) = 1$ and $(u, \epsilon) \in E$ and $\forall v \in V, d(v) = 1$ and $(\epsilon, v) \in E$. Let u, v be any vertex in U, V, respectively. Then,*

$$\mathrm{rank}(\mathbf{H}^{(\mathcal{P}\setminus(U-u),\mathcal{S}\setminus(V-v))}) = \mathrm{rank}(\mathbf{H}).$$

Proof. We prove this by showing that $\mathcal{P} \setminus (U - u)$ is a row spanning set and $\mathcal{S} \setminus (V - v)$ is a column spanning set. When $|U| = 1$, $U - u = \emptyset$ and $\mathcal{P} \setminus (U - u) = \mathcal{P}$. Consider $|U| > 1$, and let $u' \in U$ be any vertex other than u. By the assumption in the proposition, $\mathbf{H}(u, :) = \mathbf{H}(u', :)$ as $\mathbf{H}(u, \epsilon) = \mathbf{H}(u', \epsilon) = 1$ and $\mathbf{H}(u, w) = \mathbf{H}(u', w) = 0$ for any $w \in \mathcal{S}, w \neq \epsilon$. Thus $\mathcal{P} \setminus (U - u)$ is a row spanning set. A similar argument can be used to prove $\mathcal{S} \setminus (V - v)$ is a column spanning set, which we omit.

By Proposition 1, we can infer that $\mathrm{rank}(\mathbf{H}^{(\mathcal{P}\setminus(U-u),\mathcal{S}\setminus(V-v))}) = \mathrm{rank}(\mathbf{H})$. □

We denote matrix $\mathbf{H}^{(\mathcal{P}\setminus(U-u),\mathcal{S}\setminus(V-v))}$ for any u and v by $\mathbf{H}'$ and let $\mathcal{P}'$ and $\mathcal{S}'$ denote the prefix and suffix set for $\mathbf{H}'$. We do not distinguish such matrices by the choice of u, v since deductions in later sections do not depend on their specific properties.

Proposition 3. *The matrix obtained by removing the row and column induced by ϵ satisfies the following equations:*

$$\mathrm{rank}(\mathbf{H}^{(\mathcal{P},\mathcal{S}-\epsilon)}) + 1 = \mathrm{rank}(\mathbf{H})$$
$$\mathrm{rank}(\mathbf{H}^{(\mathcal{P}-\epsilon,\mathcal{S})}) + 1 = \mathrm{rank}(\mathbf{H}).$$

Also, the following equalities hold:

$$\mathrm{rank}(\mathbf{H}^{(\mathcal{P}'-\epsilon,\mathcal{S}')}) + 1 = \mathrm{rank}(\mathbf{H})$$
$$\mathrm{rank}(\mathbf{H}^{(\mathcal{P}',\mathcal{S}'-\epsilon)}) + 1 = \mathrm{rank}(\mathbf{H}).$$

Proof. By symmetry, we can safely omit the proof for $\mathbf{H}^{(\mathcal{P},\mathcal{S}-\epsilon)}$ and $\mathbf{H}.^{(\mathcal{P}',\mathcal{S}'-\epsilon)}$

We can always find a sequence $w \in W$ that is neither a prefix nor a suffix of any other sequence. Otherwise, W is non-finite, which contradicts our assumption. By Hankel's construction, $w \in \mathcal{P}$ and $w \in \mathcal{S}$ as $w \cdot \epsilon = \epsilon \cdot w = w \in W$. We claim that such a sequence w satisfies the condition for both U and V in Proposition 2. Thus, there is such a non-empty V.

Let $v \in V$ be arbitrary and let $\mathcal{S}' = \mathcal{S} \setminus (V - v)$. By the proof of Proposition 2, we know that $\mathbf{H}(:, v) = \boldsymbol{e}_1$ and $\mathbf{H}'(:, v) = \boldsymbol{e}_1$ (but these two $\boldsymbol{e}_1$'s are distinct). This directly implies that $\mathbf{H}(\epsilon, :)$ is not a linear combination of rows in $\mathbf{H}.^{(\mathcal{P}-\epsilon,\mathcal{S})}$ Hence, removing the row induced by ϵ reduces the rank of H by 1.

Similarly, $\mathbf{H}'(\epsilon, :)$ is not a linear combination of rows in $\mathbf{H}.^{(\mathcal{P}'-\epsilon,\mathcal{S}')}$ We then have $\mathrm{rank}(\mathbf{H}^{(\mathcal{P}'-\epsilon,\mathcal{S}')}) + 1 = \mathrm{rank}(\mathbf{H}') = \mathrm{rank}(\mathbf{H})$ by Proposition 2. □

4 A Divide-And-Conquer Algorithm For Basis Selection

4.1 A High Level Description of the Algorithm

We present a high-level description of the algorithm as well as its intuitions. In the next two subsections, we discuss correctness and limitations in the binary-alphabet case and then extend the results to any finite alphabet.

The algorithm returns a row spanning set for the complete Hankel matrix $\mathbf{H}$ in the form of $P^* \subseteq \mathcal{P}$. By applying the same algorithm to $\mathbf{H},^\top$ one can derive a column spanning set, $S^* \subseteq \mathcal{S}$. Recall Proposition 1: such $\mathcal{P}^*$ and $\mathcal{S}^*$ form a basis for the Hankel matrix.

Let us start by defining a subroutine Row-Reduce that takes an input matrix and two additional parameters α and γ, returning a full-rank sub-block of the matrix.

Algorithm 1 Row-Reduce(W, P, S, α, γ)

```
if $|P| \in [\alpha, \gamma)$ then
    $P' \leftarrow$ Row-Basis($\mathbf{H}^{(P,S)}(W)$)
    $S' \leftarrow N_{G(W)}(P')$
else
    $P' \leftarrow P, S' \leftarrow S$
return $(P', S')$
```

Row-Basis computes the input matrix's row basis, such as by Gaussian Elimination (GE) or QR decomposition on $M,^\top$ which has a run-time complexity of $O(n^2 m)$ or $O(nm \cdot \mathrm{rank}(M))$ for M of size $n \times m$. Faster randomized algorithms (Cheung et al., 2013) are known to run in[2] $\tilde{O}(\mathrm{nnz}(M) + k^\omega)$ to compute a linearly independent row set of size $\min\{\mathrm{rank}(M), k\}$, where $\omega < 2.38$ is the matrix multiplication exponent. One can compute a row set of size exactly $\mathrm{rank}(M)$ by iteratively doubling the size k from $k = 1$. Since this only requires at most $O(\log \mathrm{rank}(M))$ iterations, our Row-Reduce algorithm runs in at most

[2] $\tilde{O}(\cdot)$ hides polylogarithmic factors.

$\tilde{O}(\mathrm{nnz}(M) + \mathrm{rank}(M)^{\omega})$. Practically speaking, when γ is sufficiently small, there is no real advantage to using a randomized algorithm for Row-Basis.

Our parameter α serves as a lower threshold for Row-Reduce. When a set P is small, performing Row-Basis cannot reduce the overall size by much while still costing run-time. We can choose to bypass such "redundant" computations using α.

We now present the algorithm, Row-Span , for computing a row spanning set for the complete Hankel matrix. The algorithm first divides the current set of samples W by the first symbol of each string and recursively computes spanning prefix sets for each symbol that appears. Since the spanning prefix sets, when combined, might be much larger than the rank of the Hankel matrix of the current sample, our algorithm attempts to reduce the size of the combined set by performing Row-Reduce on it.

When the combined set is large, it becomes computationally expensive to perform Row-Reduce. Our parameter γ then allows us to choose to bypass such computations at the expense of having larger sub-blocks. Moreover, if each recursively computed prefix subset is large, then this is intuitively a good indication that there is limited promise for the algorithm to further reduce the size of the combined set. As such, we additionally have the lower-bound condition in line 7. A formal analysis of the runtime is presented in subsection 4.4.1.

As mentioned above, we apply this algorithm to both H and $H^{\top}$ to get the row and column spanning sets, $P^* \subseteq \mathcal{P}$ and $S^* \subseteq \mathcal{S}$, which then form a basis for Hankel matrix $\mathbf{H}(W)$. To compute the column spanning set, one can simply call Row-Span$(\Sigma, W^{\top}, \mathcal{S}^{\top}, \mathcal{P}^{\top}, \alpha, \gamma)$. Reversing the sequences in the first set of the returned tuple, one can obtain the column spanning set S.* Row-Span is recursive. We choose to present it in a top-down, divide-and-conquer fashion to facilitate its inductive analysis and our later discussion. In fact, a slightly faster, recursion-free, bottom-up approach exists.

The parameters α and γ are related to the estimation of the rank of the complete Hankel matrix $\mathbf{H}(W)$. When γ is sufficiently large, say $\gamma > |\Sigma| \cdot \mathrm{rank}(W)$ and $\alpha \leq \mathrm{rank}(W)$, the resulting basis of Row-Span is a minimal basis where both the row spanning set and column spanning set are linearly independent.

Algorithm 2 Row-Span$(\Sigma, W, \mathcal{P}, \mathcal{S}, \alpha, \gamma)$

```
 1: $P \leftarrow \emptyset, S \leftarrow \emptyset, U \leftarrow \emptyset\ L \leftarrow 0$
 2: for $\sigma \in \Sigma$ with $\mathcal{P}_\sigma \neq \emptyset$ do
 3:     $(P^*_\sigma, S^*_\sigma, u, L_\sigma) \leftarrow$ Row-Span$(\Sigma,\ W^\sigma_p,\ \mathcal{P}_\sigma, \mathcal{S}^*_\sigma, \alpha, \gamma)$
 4:     $P \leftarrow P \cup \{p | \exists p_\sigma \in P^*_\sigma, p = \sigma \cdot p_\sigma\} - (\sigma \cdot u)$
 5:     $S \leftarrow S \cup S^*_\sigma, U \leftarrow U \cup \{\sigma \cdot u\}$
 6:     $L \leftarrow L_\sigma$ if $L < L_\sigma$
 7: if $\gamma > |P| \geq 2L$ then
 8:     $(P, S) \leftarrow$ Row-Reduce$(W,\ P,\ S,\ \alpha, \gamma)$
 9:     $L \leftarrow \max(|P|, 2L)$
10: if $|U| = 0$ then
11:     $u \leftarrow \epsilon$
12: else
13:     Pick an arbitrary $u \in U$
14:     $P \leftarrow P \cup \{u\}, S \leftarrow S \cup \{\epsilon\}$
15: $P^* \leftarrow P \cup \{\epsilon\}, S^* \leftarrow S \cup \mathcal{S}^*(\epsilon)$
16: return $(P^*, S^*, u, L)$
```

4.2 Simple Case with an Alphabet of 2 symbols

To motivate the analysis of the algorithm, it will be illustrative to consider a simple alphabet with 2 symbols, $\Sigma = \{a, b\}$, before generalizing the results to arbitrary finite alphabets. We hold off the proof of correctness and the generalized results until subsection 4.3.

We start with the division scheme. Recall from the definition of $\mathcal{P}(q)$ in subsection 2.1, the intersection $\mathcal{P}(a) \cap \mathcal{P}(b) = \emptyset$. Notice that $\mathcal{P}_\sigma$ is simply $\mathcal{P}(\sigma)$ with the first symbol trimmed off of all sequences in the set. Recall also that $\epsilon \in \mathcal{P}_a$ and $\epsilon \in \mathcal{P}_b$ by definition. We then have $\mathcal{P}_a \cap \mathcal{P}_b = \epsilon$. However, the ϵ's in $\mathcal{P}_a$ and $\mathcal{P}_b$ are distinct in that the ϵ in $\mathcal{P}_a$ corresponds to the prefix a in $\mathcal{P}$ while the ϵ in $\mathcal{P}_b$ instead corresponds to b. Hence, the algorithm does not repeatedly consider the same prefix.

Before we move on to the combining steps of the algorithm, there are a couple of conditions we need to establish to outline our inductive proof of correctness.

1. After line 6, $P \cap U = \emptyset$. This is straightforward to see, as we intentionally remove the elements in U from P at line 4.

2. At line 10, $|U| = 0$ iff all $\mathcal{P}_\sigma$ are empty. Within each iteration of the for-loop, we add a unique element to U. As such, we can only

$$\begin{array}{r} \mathcal{P}(a)\left\{ \vphantom{\mathbf{H}(W_p^a)} \right. \\ \mathcal{P}(b)\left\{ \vphantom{\mathbf{H}(W_p^b)} \right. \end{array} \left[\begin{array}{c|c} \overbrace{\mathbf{H}(W_p^a)}^{\mathcal{S}^*(a)} & \mathbf{0} \\ \hdashline \mathbf{0} & \underbrace{\mathbf{H}(W_p^b)}_{\mathcal{S}^*(b)} \end{array}\right]$$

Figure 1: A visual representation of $\mathbf{H}^{(\mathcal{P}-\epsilon,\mathcal{S}-\epsilon)}$ with alphabet of 2 symbols.

end up with $|U| = 0$ when the if-condition is not satisfied for all iterations of the loop.

3. After line 9, the pair (P, S) satisfies $\text{rank}(\mathbf{H}^{(P,S)}) + 1 \leq \text{rank}(\mathbf{H}^{(P^*,S^*)})$, where P^* and S^* are the returned prefix and suffix set. Given fixed u, the matrix $\mathbf{H}^{(P-\epsilon,S)}$ is a sub-block of the matrix $\mathbf{H},^{(P^*,S^*)}$ which means that the rank of the latter must be at least as large as the former. We can then obtain the desired inequality by Proposition 3.

We now consider the combining steps (the algorithm except line 3). In the case of a 2-symbol alphabet, consider first the set $\mathcal{P}(a)$. Recall that $\mathcal{S}^*(a) = N_G(\mathcal{P}(a))$. Then each row and each column in the sub-block matrix $\mathbf{H}^{(\mathcal{P}(a),\mathcal{S}^*(a))}$ is non-empty (i.e., there is at least one non-zero entry). In contrast, the sub-block matrix $\mathbf{H}^{(\mathcal{P}(a),\mathcal{S}\setminus\mathcal{S}^*(a))} = \mathbf{0}$. To see this, consider the underlying bipartite graph G. Any $v \in \mathcal{S}$ that is not in $N(\mathcal{P}(a))$ does not have an edge connecting to the vertex set $\mathcal{P}(a)$. Similarly, $\mathbf{H}^{(\mathcal{P}(b),\mathcal{S}\setminus\mathcal{S}^*(b))} = \mathbf{0}$. When $\mathcal{S}^*(a)$ and $\mathcal{S}^*(b)$ have no intersection, then simply combining a basis for $\mathbf{H}(W_p^a)$ and one for $\mathbf{H}(W_p^b)$ with the ϵ row and column of $\mathbf{H}(W)$ yields a full numerical rank sub-block. In fact, the resulting basis is an optimal solution if the condition holds for all recursive calls. However, this is not always the case. When $\mathcal{S}^*(a) \cap \mathcal{S}^*(b) \neq \emptyset$, there is no guarantee that combining ϵ and the two column spanning sets of $\mathbf{H}(W_p^a)$ and $\mathbf{H}(W_p^b)$ would yield a column spanning set for $\mathbf{H}(W)$. This is the reason for keeping at least all the non-trivial columns. In contrast, combining the rows still results in a row spanning set, since $\mathcal{P}(a) \cap \mathcal{P}(b)$ is always an empty set. We give a visual representation of the matrix in Figure 1.

4.3 General Case

In the general case, we consider a finite alphabet with $|\Sigma| = n \in \mathbb{N}^+$ symbols. Many results in subsection 4.2 can be generalized to an arbitrarily larger finite alphabet. In this subsection, we present an inductive proof of the correctness of this algorithm.

Our goal is to prove that the algorithm returns a row spanning set P.* We shall consider a stronger statement that implies this:

Claim 1. Row-Span returns a tuple (P^*, S^*, u) s.t. P^* is a row spanning set of the input Hankel matrix $\mathbf{H}(W)$, for any $s \in N_G(P^*)$, $s \in S^*$ and where $u \in P^*$ is the only prefix that satisfies the condition in Proposition 2.

We perform our induction on the maximum length of a sequence T in the sample W. When $T = 0$, Condition 2 always holds.[3]Hence, by line 15, we explicitly added the ϵ row and column back in and the resulting P^* and S^* are exactly $\{\epsilon\}$ and $u = \epsilon$. As for the inductive case, assume Claim 1 holds for $T \leq t$. We want to prove it for $T = t + 1$. For any $\sigma \in \Sigma$, the longest sequence in W_p^σ must be at most t. Hence, in the recursive step, the returned tuple $(P_\sigma^*, S_\sigma^*, u_\sigma)$ satisfies the correctness condition by the inductive hypothesis. As a result, after line 6, the set P is a row spanning set for $\mathbf{H},^{(\mathcal{P}-\epsilon,\mathcal{S}-\epsilon)}$ U satisfies the condition in Proposition 2 and no prefix P has the same property, by Condition 1. Moreover, S has all neighbours of both P and U as in graph $G(W)$. By Condition 2, we know that U must be non-empty and that the else-block in lines 12–14 must be executed. This then leads us to Condition 3, which is $\text{rank}(\mathbf{H}^{(P,S)}) + 1 \leq \text{rank}(\mathbf{H}^{(P^*,S^*)})$. By adding the ϵ row back in, as in line 15, we have $\text{rank}(\mathbf{H}^{(\mathcal{P},\mathcal{S})}) = \text{rank}(\mathbf{H}^{(P^*,S^*)})$, which also implies that P^* is a spanning set. The condition $N_G(P^*) \subseteq S^*$ naturally holds, since each assignment to S' and S^* guarantees it. Notice also we added u to P' at line 14, which consequently proves the condition between u and P^* in the claim. This concludes the proof.

We remark that, unlike the random-cut algorithm (Balle et al., 2012) or the bipartite matching algorithm (Quattoni et al., 2017), our proof suggests that this algorithm does not require the length of sequences (length of support) in the sample W to be uniform, which provides more flexibility in choosing training data.

[3] We assume this exists and $\mathbf{H}(\epsilon, \epsilon) \neq 0$ in this case.

4.4 Run-time Complexity

In this subsection, we consider the run-time complexity of the algorithm with a fixed moment size of T. That is, the length of each sequence is at most T symbols long. We set $n = |\mathcal{P}|$ and $m = |\mathcal{S}|$. Then, the complete Hankel matrix $\mathbf{H}$ is an $n \times m$ matrix. We also let $w = |W|$ denote the total number of words in the multi-set of samples W.

Our theoretical runtime complexity is presented in Theorem 1. We remark that our algorithm is more efficient when the stochastic language follows Heaps' Law. By Heaps' Law, the size of the vocabulary (i.e., the number of unique words) $V = Kw^{\beta} = \Theta(w^{\beta})$, where K is typically between 10 and 100 and β, between 0.4 and 0.6 for English corpora. It is easy to see that both n and m are bounded above by $(T+1)V$ since each unique word can spawn at most T unique prefixes and T unique suffixes, with one addition of the null prefix and suffix ϵ. Moreover, by the same argument, the number of non-zero entries $\text{nnz}(\mathbf{H}) \le (T+1)V$.

4.4.1 Run-time Complexity of Basis Selection algorithm

Given a fixed γ parameter, the total running time of the algorithm is maximal when $\alpha = 1$. For the rest of this subsection, we only evaluate the running time of the algorithm with $\alpha = 1$ and we represent the complexity in terms of T, w, γ.

Theorem 1. *The run-time complexity of the algorithm* Row-Span *is* $\tilde{O}(TV + n\gamma^{(\omega-1)}) = \tilde{O}(Tw^{\beta} + n\gamma^{(\omega-1)})$*, or* $O(nm\gamma\log\gamma)$

We separate the running time of the algorithm into two parts. The first part is the total cost of all recursive calls other than Row-Reduce. The second part is the total cost of the Row-Reduce algorithm.

The running time of all operations other than Row-Reduce is dominated by the set-union operations. As the complete prefix and suffix sets $\mathcal{P}$ and $\mathcal{S}$ are known and fixed, we can safely assume a table representation. In that case, the set union operation is linear in the size of the smaller set to be unioned. That is, in each recursion the additional cost to calling Row-Reduce is $O(|\mathcal{S}^*(\epsilon)|)$. Consider the corresponding cells of $\mathcal{S}^*(\epsilon)$ in the complete Hankel matrix $\mathbf{H}$. By the recursion, there is no overlap between the corresponding cells of two different calls. Then, $\sum |\mathcal{S}^*(\epsilon)| \le nm$ or $\sum |\mathcal{S}^*(\epsilon)| \le TV$. Therefore, the running time of the first part is $O(nm)$ or $O(TV)$.

Now, consider the total cost of Row-Reduce.

Lemma 1. *Suppose given valid input,* Row-Span *returns a prefix subset* P^* *and a value* L*. Then* $L < \gamma$ *and the total cost of* Row-Reduce *throughout its entire operation is at most* $\tilde{O}(TV + nL^{\omega-1}) \le \tilde{O}(TV + n\gamma^{\omega-1})$ *or* $O(nmL\log L) \le O(nm\gamma\log\gamma)$.

Proof. By our recursive algorithm, we can define a unique tree, where each node, except the root, can be denoted by a unique prefix $p \in \mathcal{P}$. For a node with prefix p, there is a call of Row-Span with input $W_p = \{w \in \Sigma^* | p \cdot w \in W\}$ where the condition in line 7 is satisfied and $|P| \le \gamma$, i.e., it performs a non-trivial case of Row-Reduce. For two nodes with prefixes p and $p,'$ p' is a child of p, p is a prefix of p' and no other node p'' is both a prefix of p' and suffix of p. We assign three attributes to each node p: the total number of non-zero entries $U_p = \text{nnz}(H^{(P,S)}(W_p))$ and $|P_p|$, the size $|P|$ after line 6, and the final value L_p. The one-time cost of Row-Reduce for node p is then at most $\tilde{O}(U_p + L_p^{\omega})$, or $O(|P|mL_p)$ if using GE or QR. For the root, we denote it by the trivial string ϵ and let size $U_p = \text{nnz}(\mathbf{H}) = O(TV)$, $|P_\epsilon| = |\mathcal{P}| = n$ and $L_\epsilon = L$ be the final returned L.

Let N_h be all nodes of height h. The root has height 0. For any distinct pair of nodes $x, y \in P_h$, neither can be a prefix of the other. Then, we observe:

$$\sum_{p \in N_h} U_p \le \text{nnz}(\mathbf{H}) \le TV, \tag{2}$$

$$\sum_{p \in N_h} |P_p| \le n, \tag{3}$$

for all valid heights h. Notice that $L = \max_{p \in N_1} L_p$. By line 9, we obtain a third observation:

$$L_p \le \frac{1}{2^{h-1}} L \quad \forall p \in N_h. \tag{4}$$

$L_p < \gamma$, and the total height $H \le \log_2 L + 2$. By the convexity of the function x^{ω} on $[0, \infty)$ for any $\omega \ge 2$, the condition in line 7 and (3) further yield:

$$\begin{aligned}\sum_{p \in N_h} L_p^{\omega} &\le \frac{2^{h-1} n}{L} \cdot \frac{1}{2^{\omega(h-1)}} L^{\omega} \\ &\le \frac{n}{2^{(\omega-1)(h-1)}} L.^{\omega-1}\end{aligned}$$

For a randomized algorithm, by summing over the entire tree using (2), (4) and above, we reckon a total cost of $\tilde{O}(TV + nL^{\omega-1})$ where we recall that $\tilde{O}(\cdot)$ hides polylogarithmic factors. If we use GE or QR instead, by (3) and (4), we get a total cost of $O(nmL \log L) \leq O(nm\gamma \log \gamma)$. □

Theorem 1 follows by combining our run-time analysis for both parts. We remark that the run-time guarantees using randomized algorithms from (Cheung et al., 2013) are significantly better asymptotically than the ones using Gaussian Elimination. Note that $TV \leq O(nm)$ and we should always choose $\gamma \leq \min(n, m)$. This run-time difference is most apparent when the complete Hankel matrix $\mathbf{H}(W)$ is sparse, i.e., $TV = \tilde{O}(n + m)$.

4.4.2 Run-time Complexity in respect of SVD

There is a trade off between run time and the resulting sub-block size in our basis selection algorithm, controlled by the parameters α and γ. As mentioned before, the computational complexity is dominated by SVD, which has a run-time complexity of $O(\min(|P|, |S|)^3)$.

Since this is a divide-and-conquer algorithm, there are many recursive calls where the subsample is small, leading to small row spanning sets. Even if we do not perform Row-Basis on rows, the resulting row spanning set size is still reasonably small. This is the fundamental reason for introducing α. Generally speaking, α should be around $\gamma/|\Sigma|$ to trigger the Row-Basis subroutine. Consider when α is larger than $\gamma/|\Sigma| + 1$, and suppose the returned sub-block matrices in the recursive call all have a prefix set of size $\alpha - 1$. In the recursive call, Row-Basis is not performed, as $\alpha - 1 < \alpha$. Moreover, in the current call, Row-Basis is not performed either since the total size of prefixes is greater than γ.

According to our need, we can choose a larger γ, which leads to a more compact basis but with longer run-time, and *vice versa*. This trade-off is not linear, as we demonstrate in our experimental evaluations in section 5.

5 Experimental Evaluation

In this section, we evaluate our basis selection algorithm Row-Span , refereed to as RS. The inner algorithm Row-Reduce that combines rows and performs matrix rank algorithm Row-Basis is referred to as RR.

Implementation Details Our algorithm is implemented in Python.We consider two choices of matrix rank algorithm, GE and QR, for the Row-Basis algorithm in RR.

For GE, we implemented an algorithm that returns a reduced row echelon form (RREF) of a matrix with partial pivoting. To reduce redundancy in linear algebraic operations, our implementation additionally returns the RREF of the Hankel subblock $\mathbf{H}.^{(P^*,S^*)}$ Note that the RREF of a matrix $\mathbf{M}$ has the same row span as $\mathbf{M}$.

Our QR-decomposition-based RR algorithm uses a sparse QR implementation from SuiteSparse.[4] Since a matrix $\mathbf{M}$ passed to RR typically has a much larger number of columns than rows, our algorithm performs an additional QR decomposition first on $\mathbf{M}$ to obtain a column spanning set C. Note that the worst-case runtime of QR on $\mathbf{M}$ depends quadratically on the number of rows, but only linearly on the columns. Our algorithm then proceeds to perform a QR on $(\mathbf{M}^{(:,C)})^\top$ to retrieve a row spanning set.

Note that we discarded the lowerbound check of $|P| \geq 2L$ in line 7 of RS from our implementation. This lowerbound check is only included for proof-theoretic reasons and has little practical benefit.

Setup and Datasets All experiments are performed on a Linux machine using an Intel® Core™ i7-9700KF CPU @ 3.60GHz and 12G RAM and 32G SWAP. All computations are performed on a CPU. We measured the processing time using the Python built-in function `time.process_time()`.

For our evaluations, we considered two datasets: the English Penn Treebank dataset (Marcus et al., 1993) with universal part-of-speech tags (12 symbols), and the War and Peace dataset (Karpathy et al., 2016) for a character-based model (104 symbols and a total of $\sim$ 2.58m non-space characters). Datasets were cleaned and prepared using tools from SPiCe.[5] Hankel matrices are computed using the `scikit-splearn` package.[6]

Evaluations For our evaluations, we draw comparisons between our proposed algorithm RS using QR or GE and a maximal-matching-based algorithm, referred to as MM, from (Quattoni et al., 2017). Since no implementation of the MM algorithm was available, all experiments are performed using our own re-implementation of the algorithm

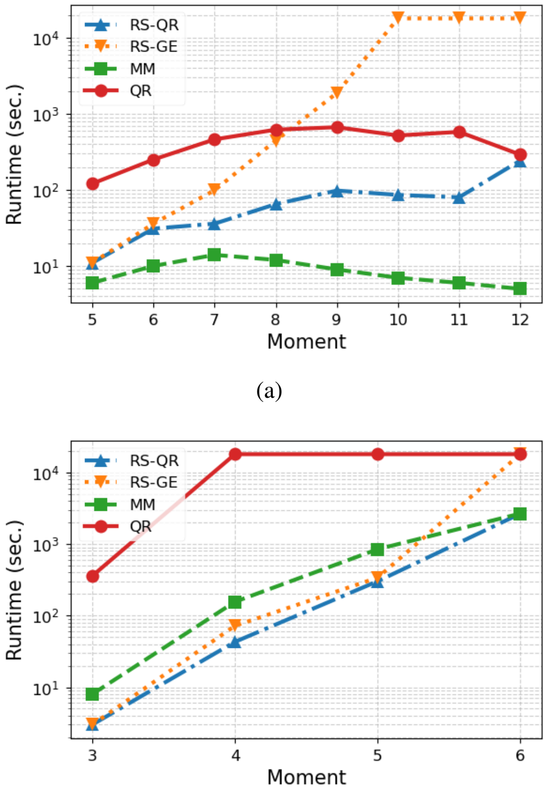
[4] SuiteSparse https://people.engr.tamu.edu/davis/suitesparse.html. Python wrapper for sparse QR from https://github.com/yig/PySPQR.

[5] https://spice.lis-lab.fr/index.php

[6] https://pypi.org/project/scikit-splearn/

in Python. A plain sparse QR decomposition algorithm, referred to simply as QR, is used as a baseline. Note that QR always returns optimal-sized bases (prefix and suffix sets of size $\mathrm{rank}(\mathbf{H})$). A timeout limit of 5 hours (180k seconds) is set for all experiments.

Runtime comparison to MM. We compared the runtime of our basis-selection algorithm RS against MM. We used the Penn Treebank and War and Peace datasets with various choices of moments. For these experiments, the parameters are fixed[7] to ensure that RR always returns a full numerical rank basis of optimal size.



(a)

(b)

Figure 2: Runtime comparison for varying moments. (a) Penn Treebank, (b) War and Peace.

On the Penn Treebank corpus, RS is comparable to MM for smaller moments, but significantly less efficient for larger moments ($T \geq 8$), see Figure 2 (a). The runtime of both the QR- and GE-based RS algorithms increases as the moment grows larger. This is as expected, since the dimension of the complete empirical Hankel matrix increases ($\sim 13k \times 13k$ for $T = 5$, $\sim 55k \times 55k$ for $T = 10$). We observe that the increase in runtime scales sub-linearly to the size of $\mathbf{H}$. From $T = 5$ to 10, the runtime increases by a factor of less than 7, from 11s to 73s, while the size of $\mathbf{H}$ increases by a factor of over 17. This suggests that RS is able to exploit the structure and sparsity of Hankel matrices.

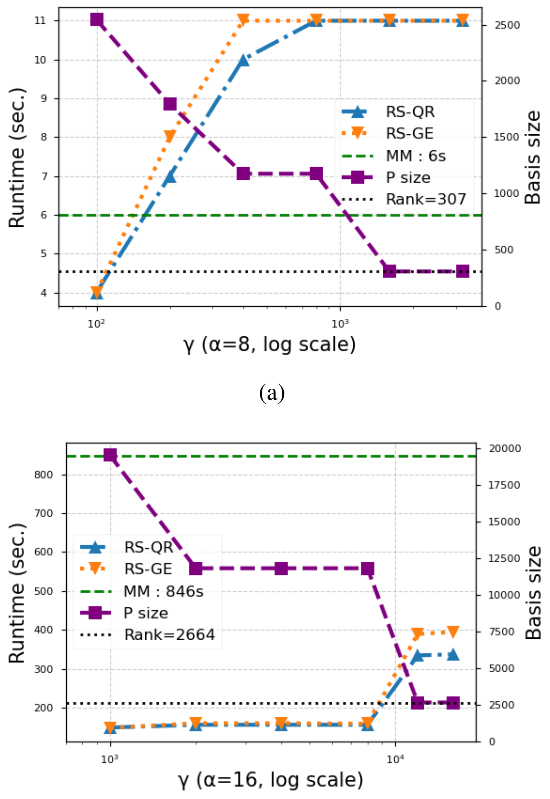
The runtime of MM is less than 16s for all tested moments. We remark that the number of nonzero entries (nnz) of the Hankel matrix remains roughly the same from $T = 8$ to 10, at between $99k$ and $110k$. The observed runtime of MM is also as expected, since the theoretical runtime of the MM algorithm scales linearly to the nnz of $\mathbf{H}$.

Note that the timeouts that accrued to RS-GE for moments $T = 10$ to 12 are due to memory limitations. The runtime of RS-GE is also consistently worse than RS-QR. We believe such inefficiency is due to our sub-optimal implementation using dense matrix operations for GE. While it is possible to utilize sparse matrices in Python, no sparse matrix implementations are particularly suitable to GE due to excessive row and column slicing and modifications to the sparsity structure. We are optimistic that a tailor-made sparse matrix implementation can significantly improve our runtime: the returned RREF of sub-block matrices is always sparse, with less than 4 nonzero entries per column on average when computing row-spanning sets. This implementation is beyond the scope of this paper.

Contrary to the above, for the War and Peace corpus, both RS-QR and RS-GE consistently outperformed MM, see Figure 2 (b). We stopped at $T = 6$ since both RS-QR and RS-GE timeout at $T = 7$ due to memory limitations. Hankel matrices for this set of experiments are considerably larger: $\mathbf{H}$ is $\sim 58k \times 58k$ in size with $\sim 225k$ nnz even at $T = 4$, and the largest matrix is $\sim 304k \times 304k$ with over $1.1m$ nnz. These matrices have ranks that are considerably smaller relative to their size, with a rank-to-rows ratio below 0.035, compared to 0.09–0.37 for the Penn Treebank dataset when moment $T \geq 8$. A relatively large rank-to-rows ratio suggests that the number of samples in the Penn Treebank dataset may be insufficient. We expect that, for sufficiently large datasets, our method would achieve better runtime performance, albeit at the cost of increased memory usage.

Finally, we remark that our algorithm RS is readily parallelizable. Our divide-and-conquer scheme is well-suited to parallel computing. The linear-algebraic operations are efficient to perform in parallel as well. The sparse QR implementation from SuiteSparse supports multi-thread computing. The elapsed times of RS-QR are typically less than 40%

[7] It suffices to choose $\gamma = |\mathcal{P}|$ and $\alpha = 1$.



(a)

(b)

Figure 3: Runtime and Basis size trade-off in γ (a) Penn Treebank ($T = 5$), (b) War and Peace ($T = 5$).

of the reported processing time.

Impact of α and γ. We study the impact of the parameters α and γ to the runtime of RS and the sizes of the bases produced. For fixed γ, we experimented with varying the value of α ($\alpha < 0.01\gamma$ for both corpora) in a range that does not affect the resulting basis size. Under such conditions, α only has a marginal effect on the runtime — less than 10% of the optimal runtime.

The parameter γ, on the other hand, has a significant impact and controls the trade-off between runtime and basis size, demonstrated by Figure 3. Increasing γ generally increases the runtime of RS while reducing the size of the basis. When γ is close to but smaller than the threshold that produces an optimal-sized basis, the runtime of RS is significantly shorter (50% to 0.1%) compared to the case where γ is above the threshold. The size of the spanning prefix/suffix set produced in this case is often within a factor of 3 times the optimal size (i.e., $\mathrm{rank}(\mathbf{H})$). For a large Hankel matrix with relatively small rank, one could prefer a smaller γ to leverage runtime reduction.

We note that the impact of γ to both runtime and basis size is not gradual, especially for larger alphabets. This is expected from our algorithmic structure: the combined prefix set passed to RR can be as large as $|\Sigma|$ times the largest prefix set returned from RS recursion. One can smoothe the effect of γ by partitioning the alphabet into a few groups and performing multiple RR steps for each group before executing RR on the entire alphabet.

6 Conclusions

We presented a simple divide-and-conquer algorithm for choosing a full numerical rank basis for Hankel matrices with good theoretical running time guarantees and a practical trade-off between running time and the resulting sub-block size.

References

Raphaël Bailly, François Denis, and Liva Ralaivola. 2009. Grammatical inference as a principal component analysis problem. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 33–40, New York, NY, USA. ACM.

Borja Balle, Ariadna Quattoni, and Xavier Carreras. 2011. A spectral learning algorithm for finite state transducers. In *Machine Learning and Knowledge Discovery in Databases*, pages 156–171, Berlin, Heidelberg. Springer Berlin Heidelberg.

Borja Balle, Ariadna Quattoni, and Xavier Carreras. 2012. Local loss optimization in operator models: A new insight into spectral learning. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, pages 1819–1826, USA. Omnipress.

Ho Yee Cheung, Tsz Chiu Kwok, and Lap Chi Lau. 2013. Fast matrix rank algorithms and applications. *J. ACM*, 60(5):31:1–31:25.

Shay B. Cohen, Karl Stratos, Michael Collins, Dean P. Foster, and Lyle Ungar. 2012. Spectral learning of latent-variable PCFGs. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 223–231, Jeju Island, Korea. Association for Computational Linguistics.

François Denis, Mattias Gybels, and Amaury Habrard. 2016. Dimension-free concentration bounds on hankel matrices for spectral learning. *J. Mach. Learn. Res.*, 17(1):1071–1102.

Daniel Hsu, Sham M. Kakade, and Tong Zhang. 2012. A spectral algorithm for learning hidden Markov models. *Journal of Computer and System Sciences*, 78(5):1460 – 1480. JCSS Special Issue: Cloud Computing 2011.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. 2016. Visualizing and understanding recurrent networks. *In ICLR Workshop Track*.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of English: the Penn treebank. *Computational Linguistics*, 19(2):313–330.

Ariadna Quattoni and Xavier Carreras. 2019. Interpolated spectral NGram language models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5926–5930, Florence, Italy. Association for Computational Linguistics.

Ariadna Quattoni, Xavier Carreras, and Matthias Gallé. 2017. A maximum matching algorithm for basis selection in spectral learning. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1477–1485. PMLR.

Sajid Siddiqi, Byron Boots, and Geoffrey Gordon. 2010. Reduced-rank hidden Markov models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 741–748, Chia Laguna Resort, Sardinia, Italy. PMLR.

Existence as Positive Measurement: Mass Nouns and Intensionality

Junri Shimada
Tokyo Keizai University
Keio University
Meiji Gakuin University
junrishimada@gmail.com

Abstract

Focusing particularly on mass nouns, we develop an ontology for natural language semantics based on measure theory. The new framework makes for rigorous discussion on changes of states and on events of continuous nature.

1 Introduction

In languages like English, nouns come in two kinds: count and mass. Count nouns, such as *child* and *statue*, exhibit a singular/plural morphological distinction, and typically denote things or individuated objects. In contrast, mass nouns, such as *milk* and *clay*, have no morphological number distinction, and typically denote stuffs or materials.

Two properties standardly associated with mass nouns are **cumulativity** (Quine, 1960) and **distributivity** (Cheng, 1973). Cumulativity refers to the fact that if x is milk and y is milk, then the sum of x and y is also milk. Distributivity refers to the fact that any part of milk is also milk. Theoretical analyses of mass nouns that take these properties into account are naturally led to assume some kind of Boolean structure that need not be atomic (Bunt, 1979; Link, 1983; Roeper, 1983; Lønning, 1987). Let's follow suit and assume that mass entities forms a nonatomic Boolean algebra $\langle D, \vee, \wedge, \neg, 0_D, 1_D\rangle$, where $\vee$ is join (or supremum), $\wedge$ is meet (or infimum), $\neg$ is complement, and 0_D and 1_D are the bottom and top elements. It is reasonable to take this algebra to be **σ-complete**, meaning that every countable subset has a join and a meet. The part-of relation $\leq$ on D is the partial order defined by

$$x \leq y \quad \text{iff} \quad x \vee y = y \quad (\text{iff} \quad x \wedge y = x).$$

For more on Boolean algebras, the reader is referred to Givant and Halmos (2009).

Assuming that the denotation of *milk* is identical to that of the predicate *is milk*, it seems appropriate to have

(1) $[\![\text{milk}]\!] = \{\, x \in D \mid x \leq m \,\}$,

where m stands for the sum of all milk in the world. To use a technical term, $[\![\text{milk}]\!]$ is an ideal.

Definition. A subset I of a Boolean algebra $\mathscr{B}$ is an **ideal** iff it satisfies the following conditions:

(C1) $0_{\mathscr{B}} \in I$.

(C2) If $x \in I$ and $y \in I$, then $x \vee y \in I$.

(C3) If $x \in I$ and $y \leq x$, then $y \in I$.

One can see that (C2) and (C3) correspond to cumulativity and distributivity, respectively. (C1) is to ensure that I is nonempty, but in the present context, it implies that $0_D \in [\![\text{milk}]\!]$. This might appear nonsensical, but I assume it to be a mere technicality with no real semantic consequences. An ideal I is said to be **σ-complete** if (C2) is strengthened to the following:

(C2′) If $S \subseteq I$ is countable, then $\bigvee S \in I$.

The elements that are less than or equal to some element $a \in \mathscr{B}$ form an ideal, known as the **principal ideal** generated by a, and written $\downarrow a$. With this notation, (1) can be expressed as $[\![\text{milk}]\!] = \downarrow m$. It is easy to see that a principal ideal in a σ-complete algebra is necessarily a σ-complete ideal.

This kind of classic model, which is static in nature, might be good enough, if all that there is is what exists at this moment in actuality. However, the existing entities might not exist in the past or future, and can be hypothesized to be absent in a state of affairs—or possible world—other than actuality. Conversely, something that does not exist now may exist at a different time or in a different world. As language enables such intensional talk, a simple analysis like (1) is bound to be deficient.

This paper develops a theory to repair this inadequacy. Section 2 explains our ontological standpoint that equates existence with positive measurement. Section 3 defines the life of an entity, the time period throughout which the entity exists. This leads us to review our ontological assumptions about times, which shall now be understood as (equivalence classes of) sets of time points of positive measure. Section 4 looks into the denotation of a mass noun in our new ontological setting. Section 5 examines sentences of continuous production or consumption and concludes that they call for mathematical integration, which shall be carried out measure-theoretically. Finally, Section 6 notes connections between telicity and integration.

2 To Be Is To Measure Positively

Quantities of entities denoted by count nouns are numerically expressed with cardinal numbers, as in *9 children*. In contrast, entities denoted by mass nouns are generally not countable, and their quantities are numerically expressed with measuring expressions such as *9.8 liters of*. To analyze them under the assumption that mass entities form a σ-complete Boolean algebra, functions known as measures come in handy (Halmos, 1950; Cartwright, 1975; Krifka, 1989; Higginbotham, 1994).

Definition. A **measure** μ on a σ-complete Boolean algebra $\mathscr{B}$ is a function from $\mathscr{B}$ into $\mathbb{R} \cup \{\infty\}$ that satisfies the following three conditions:

- Nonnegativity: $\mu(x) \geq 0$ for all x.
- $\mu(0_{\mathscr{B}}) = 0$.
- Countable additivity: if $\{x_n\}_{n\in\mathbb{N}}$ is a sequence of pairwise disjoint elements (i.e. $i \neq j$ implies $x_i \wedge x_j = 0_{\mathscr{B}}$), then $\mu\big(\bigvee_{n\in\mathbb{N}} x_n\big) = \sum_{n\in\mathbb{N}} \mu(x_n)$.

μ is called a **positive measure** if it further satisfies

- Positivity: $\mu(x) > 0$ for all $x > 0_{\mathscr{B}}$.

A Boolean algebra equipped with a positive measure is called a **measure algebra**.

Lemma 1. *If $x \leq y$, then $\mu(x) \leq \mu(y)$.*

Proof. If $x \leq y$, then $x \wedge y = x$, so $y = 1_D \wedge y = (x \vee \neg x) \wedge y = (x \wedge y) \vee (\neg x \wedge y) = x \vee (\neg x \wedge y)$, so $\mu(y) = \mu(x \vee (\neg x \wedge y)) = \mu(x) + \mu(\neg x \wedge y) \geq \mu(x)$ by countable additivity and nonnegativity. □

Using measures, we may expect sentences asserting the existence of mass entities as in (2) to translate as conditions that those entities have a positive measurement as in (3):

(2) a. There is some milk in the tank.
b. There is 9.8 liters of milk in the tank.

(3) a. $\mu(\bigvee [\![\text{milk in the tank}]\!]) > 0$.
b. $\mu_{\text{liter}}(\bigvee [\![\text{milk in the tank}]\!]) \geq 9.8$.[1]

However, such translations give us a conundrum. Unless μ is a positive measure, it is possible that $a = \bigvee([\![\text{milk in the tank}]\!]) \neq 0_D$ and yet $\mu(a) = 0$. If this happens, then (3-a) predicts (2-a) to be false even though the entity a *is* milk in the tank.

To avoid such puzzling situations, one might stipulate that μ be a positive measure. This is not what we want, however. In the previous section, we have noted that things change, and that something that exists now might be no more at other times or in other worlds, and vice versa. I propose that we capture this intuition by employing a family $\{\mu^{w,p}\}_{w\in W,\, p\in T}$ of measures, where W is the set of possible worlds and T the set of time points. Our slogan here is "existence as positive measurement," which means something like the following:

(4) An entity $x \in D$ exists at p in w iff $\mu^{w,p}(x) > 0$.

This will allow us to describe how entities come into or go out of existence. For instance, if x was born at noon, then $\mu^{w,p}(x) = 0$ if p is a time point before the noon, and $\mu^{w,p}(x) > 0$ if p is a time point after the noon. On this view, entities fundamentally never come into or go out of existence. They are always there as elements of D and may or may not have a positive measurement, depending on the world and the time. On this approach, it is essential that $\mu^{w,p}$ be not a positive measure.

3 The Life of an Entity

Let's discuss the life of a mass entity x, i.e., the set of time points at which x exists.[2] In what follows,

[1] We have "$\geq$" rather than "=" here because (2-b) is consistent with there being more than 9.8 liters of milk in the tank. The impression that (2-b) says that there is exactly 9.8 liters of milk in the tank can be attributed to scalar implicature.

[2] By "a mass entity x," I mean an entity that happens to be predicated of by a mass noun. I am not sure whether the count/mass distinction is inherent in the entities in the model. It seems possible for an entity to be predicated of by a count noun and by a mass noun at the same time, as in *This is furniture—a chair to be precise.*

we will not vary the world parameter and focus on a single, fixed world. Accordingly, we will drop the world parameter and simply write μ^p. Under the slogan "existence as positive measurement," that x exists at p will mean that x measures positively at p. However, if some nonzero part of x (i.e., part of x that is not 0_D) is of measure zero at p, we do not want to think that x exists at p. In order to be able to assert that x exists at p, every bit of x must measure positively. We therefore put forward the following definition.

Definition. The **life of** x **with respect to** a family $\{\mu^p\}_{p\in T}$ of measures, written $\heartsuit_\mu(x)$, is

$$\heartsuit_\mu(x) := \{\, p \in T \mid \\ \forall y(0_D < y \leq x \to \mu^p(y) > 0)\,\}.$$

Lemma 2. (i) *$\heartsuit_\mu(\bigvee X) = \bigcap_{x\in X} \heartsuit_\mu(x)$ for all countable $X \subseteq D$.*

(ii) *If $x \leq y$, then $\heartsuit_\mu(y) \subseteq \heartsuit_\mu(x)$ for all $x, y \in D$.*

Proof. (i) For every $x \in X$, since $x \leq \bigvee X$, we have

$$\begin{aligned} & p \in \heartsuit_\mu(\textstyle\bigvee X) \\ \iff & \forall y(0_D < y \leq \textstyle\bigvee X \to \mu^p(y) > 0) \\ \implies & \forall y(0_D < y \leq x \to \mu^p(y) > 0) \\ \iff & p \in \heartsuit_\mu(x), \end{aligned}$$

so $\heartsuit_\mu(\bigvee X) \subseteq \heartsuit_\mu(x)$. Hence $\heartsuit_\mu(\bigvee X) \subseteq \bigcap_{x\in X} \heartsuit_\mu(x)$.

Conversely, suppose that $p \in \bigcap_{x\in X} \heartsuit_\mu(x)$. Suppose that $0_D < y \leq \bigvee X$. Then $\bigvee_{x\in X}(y \wedge x) = y \wedge (\bigvee X) = y > 0_D$, so there is some $x_0 \in X$ with $y \wedge x_0 \neq 0_D$. Since $p \in \heartsuit_\mu(x_0)$ and $y \wedge x_0 \leq x_0$, we have $\mu^p(y \wedge x_0) > 0$ by the definition of $\heartsuit_\mu$. Since $\mu^p(y) \geq \mu^p(y \wedge x_0)$ by Lemma 1, it follows that $\mu^p(y) > 0$. This shows that $p \in \heartsuit_\mu(\bigvee X)$.

(ii) If $x \leq y$, then $x \vee y = y$, so by (i), $\heartsuit_\mu(y) = \heartsuit_\mu(x \vee y) = \heartsuit_\mu(x) \cap \heartsuit_\mu(y) \subseteq \heartsuit_\mu(x)$. □

It might seem that the assertion that an entity x exists sometime will be translated as $\heartsuit_\mu(x) \neq \varnothing$. However, what if $\heartsuit_\mu(x)$ consists of a single time point p_0, i.e., $\heartsuit_\mu(x) = \{p_0\}$? This means that x had some physical presence at p_0, but none either before or after. Do we still want to say that x existed? Even if such a situation obtained in reality, there would be no way of ascertaining it, and I believe most of us intuitively think that things do not work that way. Then, we might as well ignore the possibility of such literally instantaneous existence. For something to exist, it must be there for some positive length of time. It is time we extended the slogan "existence as positive measurement" to the realm of times.

As in physics, let's identify T with the set $\mathbb{R}$ of real numbers. Then, the **Lebesgue measure** μ_{L} can be used to measure the length of a subset of T. For instance, if an interval $X \subseteq T$ begins at p and ends at q, then $\mu_{\mathrm{L}}(X) = q - p$ (regardless of whether X is open, closed, or half-open). Let $\mathscr{L}(T)$ be the set of Lebesgue-measurable subsets of T, i.e., subsets X of T for which $\mu_{\mathrm{L}}(X)$ is defined. It is known that $\langle \mathscr{L}(T), \cup, \cap, {}^{\mathsf{c}}, \varnothing, T\rangle$ is a σ-complete Boolean algebra, and μ_{L} gives a measure on it. Since $\mu_{\mathrm{L}}(\{p_0\}) = 0$, through the lens of the Lebesgue measure, $\{p_0\}$ is equivalent to $\varnothing$. In other words, being there only at p_0 will be identified with being there at no time at all. The mathematical way to achieve this perspective is to go to the **quotient algebra** of $\mathscr{L}(T)$ modulo the set I of sets of measure zero, i.e.,

$$I = \{\, X \in \mathscr{L}(T) \mid \mu_{\mathrm{L}}(X) = 0\,\}.$$

I is a σ-complete ideal in $\mathscr{L}(T)$, and one can define an equivalence relation $\sim$ on $\mathscr{L}(T)$ by

$$X \sim Y \quad \text{iff} \quad (X \cap Y^{\mathsf{c}}) \cup (Y \cap X^{\mathsf{c}}) \in I.$$

Let $[X] = \{\, Y \in \mathscr{L}(T) \mid X \sim Y\,\}$ denote the equivalence class of X. Then

$$\mathcal{T} := \{\, [X] \mid X \in \mathscr{L}(T)\,\}$$

gives rise to the quotient algebra $\langle \mathcal{T}, \sqcup, \sqcap, {}^{\mathsf{c}}, 0_{\mathcal{T}}, 1_{\mathcal{T}}\rangle$. This is a σ-complete Boolean algebra. For $X, Y \in \mathscr{L}(T)$, we have $[X] \sqcup [Y] = [X \cup Y]$, $[X] \sqcap [Y] = [X \cap Y]$, $[X]^{\mathsf{c}} = [X^{\mathsf{c}}]$, $0_{\mathcal{T}} = [\varnothing]$ and $1_{\mathcal{T}} = [T]$. For $s, t \in \mathcal{T}$, we define $s \sqsubseteq t$ iff $s \sqcup t = t$. If $X \subseteq Y$, then $[X] \sqsubseteq [Y]$. Whenever $X, Y \in t \in \mathcal{T}$, we have $\mu_{\mathrm{L}}(X) = \mu_{\mathrm{L}}(Y)$, so we can let $\mu_{\mathrm{L}}(t)$ be this unique value that μ_{L} assumes at any element of t. This way, μ_{L} can be extended to $\mathcal{T}$, and it is known that μ_{L} gives a positive measure on $\mathcal{T}$. Thus $\mathcal{T}$ is a measure algebra.

We can now forsake (4) and put forward the following definition:

Definition. For all $x \in D$:

- x **exists at** $t \in \mathcal{T}$ **with respect to** $\{\mu^p\}_{p\in T}$ iff $[\heartsuit_\mu(x)] \sqsupseteq t \neq 0_{\mathcal{T}}$.

- x **never exists with respect to** $\{\mu^p\}_{p\in T}$ iff $[\heartsuit_\mu(x)] = 0_{\mathcal{T}}$.

According to this definition, if x exists at $t \neq 0_{\mathcal{T}}$, then $\mu_{\mathrm{L}}([\heartsuit_\mu(x)]) \geq \mu_{\mathrm{L}}(t) > 0$ by Lemma 1 and the positivity of μ_{L} on $\mathcal{T}$, so x's life has some positive length. On the other hand, if $\heartsuit_\mu(x) = \{p_0\}$, then $[\heartsuit_\mu(x)] = 0_{\mathcal{T}}$, so x never exists.[3]

Some immediate worries present themselves. In this new setting, a "time" is a member of $\mathcal{T}$, i.e., an equivalence class of Lebesgue-measurable subsets of T.[4] This means that if $X \subseteq T$ is not Lebesgue-measurable, we cannot even discuss whether an entity exists at X. Also, if $\heartsuit_\mu(x)$ is not Lebesgue-measurable, we cannot discuss whether x exists at any time. Indeed, using the axiom of choice, one can construct subsets of $\mathbb{R}$ that are not Lebesgue-measurable (Vitali, 1905). On the other hand, Zermelo–Fraenkel set theory without the axiom of choice has a model where every subset of $\mathbb{R}$ is Lebesgue-measurable (Solovay, 1970). What I take this to imply is that we probably need not worry about the possible existence of sets that are not Lebesgue-measurable. Even if they exist, since we obviously cannot mentally execute an infinite-step process involved in the use of the axiom of choice, I believe it is safe to assume that such sets do not figure in our mental model for natural language semantics.

Another worry concerns equating subsets of T whose symmetric differences are of measure zero. For instance, suppose that $\heartsuit_\mu(x) = (0,1)\cup(1,2)$.[5] Since $[\heartsuit_\mu(x)] = [(0,2)]$, it follows that x exists at $[(0,2)]$ by our definition. Such a statement, however, would naturally lead one to expect every nonzero part of x to measure positively throughout $(0,2)$. Nevertheless, since $1 \notin \heartsuit_\mu(x)$, there is some nonzero part y of x such that $\mu^1(y) = 0$. This is quite counterintuitive. How can something only momentarily vanish and then come back? Makoto Kanazawa (personal communication) suggests that this should not be a problem from a measure theoretic point of view, since the probability of picking such a point is zero. More precisely, if $X \subseteq T$ and $X \sim \heartsuit_\mu(x)$, then $\mu_{\mathrm{L}}(X \cap \heartsuit_\mu(x)^{\mathrm{c}}) = 0$, so assuming that $\mu_{\mathrm{L}}(X) = \mu_{\mathrm{L}}(\heartsuit_\mu(x)) > 0$, the probability of picking a point p from X that is not in $\heartsuit_\mu(x)$ is

$$\frac{\mu_{\mathrm{L}}(X \cap \heartsuit_\mu(x)^{\mathrm{c}})}{\mu_{\mathrm{L}}(X)} = 0.$$

In measure theory, a property is said to hold **almost everywhere** if it does except on a set of measure zero. Analogously, in probability theory, an event is said to happen **almost surely** if its probability is 1. Let's shift to this way of thinking and speak as follows:

(5) $p \in X \sim \heartsuit_\mu(x)$ **almost surely entails** that $\mu^p(x) > 0$.

Thus, $[\heartsuit_\mu(x)] = [(0,2)]$ almost surely entails that $\mu^1(x) > 0$.

Lastly, note that since $\heartsuit_\mu(0_D) = T$, it follows that 0_D exists at any $t \neq 0_{\mathcal{T}}$. I take this to be a mere technicality of no real semantic import.

4 Mass Noun Denotations

Let M be a mass noun. To describe situations where an entity becomes M or ceases to be M, the meaning of M needs to be sensitive to the time. So let's take M to denote a binary relation between times and entities. Now that our slogan "existence as positive measurement" motivates the view that the times that matter in human minds are members of $\mathcal{T}$, this means that M denotes a relation between the two Boolean algebras $\mathcal{T}$ and D, i.e., $[\![\mathrm{M}]\!] \subseteq \mathcal{T} \times D$. Let $[\![\mathrm{M}]\!]\,(t)$ denote the set of entities that are M at a given time t:

$$[\![\mathrm{M}]\!]\,(t) := \{\, x \in D \mid \langle t, x\rangle \in [\![\mathrm{M}]\!] \,\}.$$

As mentioned in Section 1, $[\![\mathrm{M}]\!]\,(t)$ ought to be a principal ideal. This means that there is a family $\{m_t\}_{t\in\mathcal{T}}$ of elements of D such that

$$[\![\mathrm{M}]\!]\,(t) = {\downarrow} m_t.$$

Now, let's consider the set of times at which a given entity x is M, for which we write:

$$[\![\mathrm{M}]\!]^{-1}(x) := \{\, t \in \mathcal{T} \mid \langle t, x\rangle \in [\![\mathrm{M}]\!] \,\}.$$

Intuitively, if x is M both at s and at t, then x ought to be M at $s \sqcup t$. Also, if x is M at t and $s \sqsubseteq t$, then x must be M at s as well. This means that $[\![\mathrm{M}]\!]^{-1}(x)$ should be an ideal in $\mathcal{T}$. Regarding this matter, the following holds.

[3] If $t = 0_{\mathcal{T}}$, then $[\heartsuit_\mu(x)] \sqsupseteq t$ trivially holds, so this case has been excluded from the definition in order to avoid such a clashing statement as "x exists at t and x never exists."

[4] In natural language semantics, "times" often mean time intervals. Aside from dealing with equivalence classes, our "times" are different from time intervals in that they can be discontinuous.

[5] (a,b) denotes an open interval. Thus $(0,1) = \{\, p \in T \mid 0 < p < 1 \,\}$.

Proposition 3. *Suppose that each* $[\![\mathrm{M}]\!]\,(t)$ *is a principal ideal* $\downarrow m_t$*. Then, the following are equivalent:*

(a) $[\![\mathrm{M}]\!]^{-1}(x)$ *is a* σ*-complete ideal in* $\mathcal{T}$ *for all* $x \in D$.

(b) $\bigwedge_{t\in\mathcal{C}} m_t = m_{\bigsqcup\mathcal{C}}$ *for all countable* $\mathcal{C} \subseteq \mathcal{T}$.

Proof of this proposition follows shortly. Before that, note that what Condition (b) says is twofold. First, if x is M at s and y is M at t, then $x \wedge y$ is M at $s \sqcup t$. This is expected of M's denotation; since $x \wedge y$ is part both of x and of y, it ought to be M both at s and at t, and hence throughout $s \sqcup t$. Second, if something is M at $s \sqcup t$, then it must be M both at s and at t. Again, this is only expected.

Lemma 4. *Suppose that Condition (b) of Proposition 3 holds. Then, the following hold:*

(i) $m_{0_\mathcal{T}} = 1_D$.

(ii) *If* $s \sqsubseteq t$*, then* $m_t \leq m_s$.

Proof. (i) $m_{0_\mathcal{T}} = m_{\bigsqcup\varnothing} = \bigwedge_{t\in\varnothing} m_t = \bigwedge\varnothing = 1_D$.

(ii) If $s \sqsubseteq t$, then $t = s \sqcup t$, so $m_t = m_{s\sqcup t} = m_s \wedge m_t \leq m_s$. □

Proof of Proposition 3. (a) ⇒ (b). Let $\mathcal{C} \subseteq \mathcal{T}$ be countable.

Suppose that $x \leq m_t$ for all $t \in \mathcal{C}$. Then for all $t \in \mathcal{C}$, we have $x \in \downarrow m_t = [\![\mathrm{M}]\!]\,(t)$, so $\langle t, x\rangle \in [\![\mathrm{M}]\!]$ and hence $t \in [\![\mathrm{M}]\!]^{-1}(x)$. Since $[\![\mathrm{M}]\!]^{-1}(x)$ is a σ-complete ideal by assumption, it follows that $\bigsqcup\mathcal{C} \in [\![\mathrm{M}]\!]^{-1}(x)$, so $x \in [\![\mathrm{M}]\!]\,(\bigsqcup\mathcal{C}) = \downarrow m_{\bigsqcup\mathcal{C}}$ and hence $x \leq m_{\bigsqcup\mathcal{C}}$. Taking $\bigwedge_{t\in\mathcal{C}} m_t$ for x in particular, we obtain $\bigwedge_{t\in\mathcal{C}} m_t \leq m_{\bigsqcup\mathcal{C}}$.

Next, suppose that $x \leq m_{\bigsqcup\mathcal{C}}$. Then $x \in \downarrow m_{\bigsqcup\mathcal{C}} = [\![\mathrm{M}]\!]\,(\bigsqcup\mathcal{C})$, so $\bigsqcup\mathcal{C} \in [\![\mathrm{M}]\!]^{-1}(x)$. For every $t \in \mathcal{C}$, we have $t \leq \bigsqcup\mathcal{C}$, and since $[\![\mathrm{M}]\!]^{-1}(x)$ is a σ-complete ideal by assumption, it follows that $t \in [\![\mathrm{M}]\!]^{-1}(x)$, and hence $x \in [\![\mathrm{M}]\!]\,(t) = \downarrow m_t$, so $x \leq m_t$. It follows that $x \leq \bigwedge_{t\in\mathcal{C}} m_t$. Taking $m_{\bigsqcup\mathcal{C}}$ for x in particular, we obtain $m_{\bigsqcup\mathcal{C}} \leq \bigwedge_{t\in\mathcal{C}} m_t$.

This completes the proof that $\bigwedge_{t\in\mathcal{C}} m_t = m_{\bigsqcup\mathcal{C}}$.

(b) ⇒ (a). We verity that $[\![\mathrm{M}]\!]^{-1}(x)$ satisfies the three conditions for being a σ-complete ideal.

(C1) $x \leq 1_D = m_{0_\mathcal{T}}$ by Lemma 4(i), so $x \in \downarrow m_{0_\mathcal{T}} = [\![\mathrm{M}]\!]\,(0_\mathcal{T})$. Thus, $\langle 0_\mathcal{T}, x\rangle \in [\![\mathrm{M}]\!]$ and $0_\mathcal{T} \in [\![\mathrm{M}]\!]^{-1}(x)$.

(C2′) Suppose that $\mathcal{C} \subseteq \mathcal{T}$ is countable, and $t \in [\![\mathrm{M}]\!]^{-1}(x)$ for all $t \in \mathcal{C}$. Then for all $t \in \mathcal{C}$, we have $x \in [\![\mathrm{M}]\!]\,(t) = \downarrow m_t$ and hence $x \leq m_t$, so $x \leq \bigwedge_{t\in\mathcal{C}} m_t = m_{\bigsqcup\mathcal{C}}$ by the assumed condition. Thus, $x \in \downarrow m_{\bigsqcup\mathcal{C}} = [\![\mathrm{M}]\!]\,(\bigsqcup\mathcal{C})$ and hence $\bigsqcup\mathcal{C} \in [\![\mathrm{M}]\!]^{-1}(x)$.

(C3) Suppose that $t \in [\![\mathrm{M}]\!]^{-1}(x)$ and $s \sqsubseteq t$. Since $x \in [\![\mathrm{M}]\!]\,(t) = \downarrow m_t$, we have $x \leq m_t \leq m_s$ by Lemma 4(ii), so $x \in \downarrow m_s = [\![\mathrm{M}]\!]\,(s)$. Hence $s \in [\![\mathrm{M}]\!]^{-1}(x)$. □

I think it is reasonable to assume that $[\![\mathrm{M}]\!]^{-1}(x)$ is indeed a σ-complete ideal in $\mathcal{T}$, and a principal ideal at that, just as $[\![\mathrm{M}]\!]\,(t)$ is a principal ideal in D. This means that there is a family $\{\ell_x\}_{x\in D}$ of elements of $\mathcal{T}$ such that

$$[\![\mathrm{M}]\!]^{-1}(x) = \downarrow \ell_x.$$[6]

As times and entities play symmetric roles in M's denotation, the following are immediate as parallels of Proposition 3 and Lemma 4.

Proposition 5. *Suppose that each* $[\![\mathrm{M}]\!]^{-1}(x)$ *is a principal ideal* $\downarrow \ell_t$*. Then, the following are equivalent:*

(a) $[\![\mathrm{M}]\!]\,(t)$ *is a* σ*-complete ideal in* D *for all* $t \in \mathcal{T}$.

(b) $\bigsqcap_{x\in C} \ell_x = \ell_{\bigvee C}$ *for all countable* $C \subseteq D$.

Lemma 6. *Suppose that Condition (b) of Proposition 5 holds. Then, the following hold:*

(i) $\ell_{0_D} = 1_\mathcal{T}$.

(ii) *If* $x \leq y$*, then* $\ell_y \sqsubseteq \ell_x$.

What is the meaning of Condition (b) of Proposition 5? First, it says that if x is M at s and y is M at t, then $x \vee y$ is M at $s \sqcap t$. Second, it says that if $x \vee y$ is M at some time, then both x and y are M at that time.

Let $\mathcal{M} = \{\, m_t \mid t \in \mathcal{T} \,\}$ and $\mathcal{L} = \{\, \ell_x \mid x \in D \,\}$. Through domain restriction, ℓ can be regarded as a function from $\mathcal{M}$ into $\mathcal{L}$. Likewise, m can be viewed as a function from $\mathcal{L}$ into $\mathcal{M}$. Then the pair of ℓ and m forms what is known as a **Galois connection** between $\langle\mathcal{M}, \leq\rangle$ and $\langle\mathcal{L}, \sqsubseteq\rangle$, satisfying the condition in (i) below (*vide* Mac Lane, 1998).

Lemma 7. *Suppose that* $x \in D$ *and* $t \in \mathcal{T}$.

(i) $t \sqsubseteq \ell_x$ *if and only if* $x \leq m_t$.

(ii) $x \leq m_{\ell_x}$ *and* $t \sqsubseteq \ell_{m_t}$.

[6] Speaking intuitively, ℓ_x is x's lifespan as M.

(iii) $m_{\ell_{m_t}} = m_t$ *and* $\ell_{m_{\ell_x}} = \ell_x$.

Proof. (i) $t \sqsubseteq \ell_x$ iff $t \in {\downarrow}\ell_x$ iff $t \in \llbracket \mathrm{M} \rrbracket^{-1}(x)$ iff $\langle t, x\rangle \in \llbracket \mathrm{M} \rrbracket$ iff $x \in \llbracket \mathrm{M} \rrbracket(t)$ iff $x \in {\downarrow}m_t$ iff $x \leq m_t$.

(ii) Since $\ell_x \in {\downarrow}\ell_x = \llbracket \mathrm{M} \rrbracket^{-1}(x)$, we have $\langle \ell_x, x\rangle \in \llbracket \mathrm{M} \rrbracket$, so $x \in \llbracket \mathrm{M} \rrbracket(\ell_x) = {\downarrow}m_{\ell_x}$ and hence $x \leq m_{\ell_x}$. Dually for $t \sqsubseteq \ell_{m_t}$.

(iii) By (ii), $m_t \leq m_{\ell_{m_t}}$. Also, since $t \sqsubseteq \ell_{m_t}$ by (ii), we have $m_{\ell_{m_t}} \leq m_t$ by Lemma 4(ii). Hence $m_{\ell_{m_t}} = m_t$. Dually for $\ell_{m_{\ell_x}} = \ell_x$. □

Our denotation of a mass noun M is a binary relation between equivalence classes of Lebesgue-measurable sets of time points and entities. However, we should also like to be able to talk about whether or not an entity is M at a specific time point. I say that a natural idea would be that an entity is M at a time point p if and only if it is M at some open set (in topological sense) of time points that contains p. Let's define the derivative denotation of M for a time point $p \in T$ by

Definition. $\llbracket \mathrm{M} \rrbracket(p) := \bigcup_{G \text{ open},\, p \in G} \llbracket \mathrm{M} \rrbracket([G])$.

Note that every open set G is Lebesgue-measurable, so $[G]$ makes sense. Here is a reason for considering only open sets. Imagine that for an entity x, we have $\ell_x = [(0,1)] = [[0,1]]$.[7] This means that x begins to be M at time point 0 and ceases to be M at time point 1. Now, do we want to say that x is M at time point 0? How about at time point 1? Some might say yes, but it seems fair to say that it is uncertain whether x is M at those points. If we allowed G to be the non-open set $[0,1]$ in the definition of $\llbracket \mathrm{M} \rrbracket(0)$, we would be forced to say that x is M at 0. By dealing exclusively with open sets, we can avoid such an undesirable conclusion. On our definition, in contrast, if $x \in \llbracket \mathrm{M} \rrbracket(p)$, then there is an open set G such that $p \in G$ and $x \in \llbracket \mathrm{M} \rrbracket([G])$. As G is open, there is an open neighborhood of p in G, or more specifically, an open interval B such that $p \in B \subseteq G$. Since $[G] \in \llbracket \mathrm{M} \rrbracket^{-1}(x)$ and $\llbracket \mathrm{M} \rrbracket^{-1}(x)$ is an ideal, it follows that $[B] \in \llbracket \mathrm{M} \rrbracket^{-1}(x)$. Thus, we conclude that x is M at some time interval that surrounds p instead of having p on its edge. Assuming that $\llbracket \mathrm{M} \rrbracket(t)$ and $\llbracket \mathrm{M} \rrbracket^{-1}(x)$ are principal ideals, our definition leads to the following nice consequence.

Proposition 8. $\llbracket \mathrm{M} \rrbracket(p)$ *is an ideal in* D.

Proof. We check the three conditions for being an ideal.

(C1) Since $0_D \leq m_{1_\mathcal{T}}$, we have $0_D \in {\downarrow}m_{1_\mathcal{T}} = \llbracket \mathrm{M} \rrbracket(1_\mathcal{T}) = \llbracket \mathrm{M} \rrbracket([T])$. Since the whole space T is open and $p \in T$, this shows that $0_D \in \llbracket \mathrm{M} \rrbracket(p)$.

(C2) Suppose that $x, y \in \llbracket \mathrm{M} \rrbracket(p)$. Then, there exist an open set G_1 such that $p \in G_1$ and $x \in \llbracket \mathrm{M} \rrbracket([G_1])$, and an open set G_2 such that $p \in G_2$ and $y \in \llbracket \mathrm{M} \rrbracket([G_2])$. Then $[G_1] \in \llbracket \mathrm{M} \rrbracket^{-1}(x) = {\downarrow}\ell_x$, so $[G_1] \sqsubseteq \ell_x$. Similarly, $[G_2] \sqsubseteq \ell_y$. Then $[G_1 \cap G_2] = [G_1] \sqcap [G_2] \sqsubseteq \ell_x \sqcap \ell_y = \ell_{x \vee y}$ by Proposition 5(b). So $[G_1 \cap G_2] \in {\downarrow}\ell_{x\vee y} = \llbracket \mathrm{M} \rrbracket^{-1}(x \vee y)$ and hence $x \vee y \in \llbracket \mathrm{M} \rrbracket([G_1 \cap G_2])$. As $G_1 \cap G_2$ is an open set and $p \in G_1 \cap G_2$, this shows that $x \vee y \in \llbracket \mathrm{M} \rrbracket(p)$.

(C3) Suppose that $x \in \llbracket \mathrm{M} \rrbracket(p)$ and $y \leq x$. Then, there is an open set G such that $p \in G$ and $x \in \llbracket \mathrm{M} \rrbracket([G]) = {\downarrow}m_{[G]}$. Then $y \leq x \leq m_{[G]}$, so $y \in {\downarrow}m_{[G]} = \llbracket \mathrm{M} \rrbracket([G])$. Hence $y \in \llbracket \mathrm{M} \rrbracket(p)$. □

Note that $\llbracket \mathrm{M} \rrbracket(p)$ is not necessarily a σ-complete ideal. This is due to the fact that countable intersection of open sets need not be an open set. For instance, $G_n = \left(-\frac{1}{n}, \frac{1}{n}\right)$ is open for every positive integer n, but $\bigcap_{n\geq 1} G_n = \{0\}$ is not.

We have derived $\llbracket \mathrm{M} \rrbracket(p)$'s from $\llbracket \mathrm{M} \rrbracket([G])$'s, but we can also get back to $\llbracket \mathrm{M} \rrbracket([G])$'s from $\llbracket \mathrm{M} \rrbracket(p)$'s.

Proposition 9. *Let* $G \subseteq T$ *be an open set. Then* $\llbracket \mathrm{M} \rrbracket([G]) = \bigcap_{p \in G} \llbracket \mathrm{M} \rrbracket(p)$.

Proof. By the definition of $\llbracket \mathrm{M} \rrbracket(p)$, for every $p \in G$, we have $\llbracket \mathrm{M} \rrbracket([G]) \subseteq \llbracket \mathrm{M} \rrbracket(p)$, so $\llbracket \mathrm{M} \rrbracket([G]) \subseteq \bigcap_{p\in G} \llbracket \mathrm{M} \rrbracket(p)$.

For the reverse inclusion, suppose that $x \in \bigcap_{p\in G} \llbracket \mathrm{M} \rrbracket(p)$. For each $p \in G$, we have $x \in \llbracket \mathrm{M} \rrbracket(p)$, so there is some open set G_p such that $p \in G_p$ and $x \in \llbracket \mathrm{M} \rrbracket([G_p]) = {\downarrow}m_{[G_p]}$ and hence $x \leq m_{[G_p]}$. Since $G \subseteq \bigcup_{p\in G} G_p$, $\{G_p\}_{p\in G}$ is an open cover of G. Since $T = \mathbb{R}$ has a countable base for its topology, there exists a countable subcover $\{G_{p_n}\}_{n\in\mathbb{N}}$, so $G \subseteq \bigcup_{n\in\mathbb{N}} G_{p_n}$. Then $[G] \sqsubseteq \left[\bigcup_{n\in\mathbb{N}} G_{p_n}\right]$, and

$$\begin{aligned} x &\leq \bigwedge_{n\in\mathbb{N}} m_{[G_{p_n}]} && \text{(since } x \leq m_{[G_{p_n}]} \text{ for all } n) \\ &= m_{\bigsqcup_{n\in\mathbb{N}}[G_{p_n}]} && \text{(by Proposition 3(b))} \\ &= m_{[\bigcup_{n\in\mathbb{N}} G_{p_n}]} && \\ &\leq m_{[G]}, && \text{(by Lemma 4(ii))} \end{aligned}$$

so $x \in {\downarrow}m_{[G]} = \llbracket \mathrm{M} \rrbracket([G])$. This shows that $\bigcap_{p\in G} \llbracket \mathrm{M} \rrbracket(p) \subseteq \llbracket \mathrm{M} \rrbracket([G])$. □

[7] $[0,1]$ denotes the closed interval $\{\, p \in T \mid 0 \leq p \leq 1 \,\}$, and $[[0,1]]$ its equivalence class.

It is time we revisited mass entities' lives. If something is M at some time, then that thing had better exist at that time in the sense defined in the previous section. I would now like to propose the following:

(6) For every mass noun M, there is a relevant family $\{\mu^p\}_{p \in T}$ of measures such that $t \sqsubseteq [\heartsuit_\mu(x)]$ for all $t \in \mathcal{T}$ and $x \in D$ such that $\langle t, x \rangle \in \llbracket \mathrm{M} \rrbracket$.

This does not mean that there is a unique relevant measure for a given mass noun. For instance, milk can be measured in terms of volume (*liters of*) or in terms of mass (*grams of*). (6) ensures that even if sentences do not explicitly use expressions like *liters of*, some implicit measure is involved in modeling their meaning. In fact, the sentences in (7) contain no specific measuring unit, and yet demonstrate that the amount of milk is somehow measured; otherwise, one could not state if the milk in the tank is a lot or a little. As (8) shows, the same point holds for nouns like *love* as well, for which no appropriate word of a measuring unit seems to exist.

(7) a. There is a lot of milk in the tank.
b. There is a little milk in the tank.

(8) a. There is a lot of love between Christa and Ymir.
b. There is a little love between Christa and Ymir.

Why shouldn't we rather say the following stronger statement instead of (6)?

(6′) For every mass noun M, there is a relevant family $\{\mu^p\}_{p \in T}$ of measures such that $\ell_x = [\heartsuit_\mu(x)]$ for all $x \in D$.

This says that the maximum time at which x is, say, milk coincides with x's life, i.e., the time throughout which x has some physical presence. However, we do not want to stipulate that x be allowed to exist only as milk. Imagine that x starts its life as milk at time point 0, turns into cheese at time point 1, and eventually gets eaten by Chris at time point 2 to completely vanish from the world. In this case, $\ell_x = [(0,1)]$ and $[\heartsuit_\mu(x)] = [(0,2)]$, so $\ell_x \neq [\heartsuit_\mu(x)]$. The following proposition shows that (6) can be equivalently expressed in different ways.

Proposition 10. *The following are equivalent:*

(i) $t \sqsubseteq [\heartsuit_\mu(x)]$ *for all* $t \in \mathcal{T}$ *and* $x \in D$ *such that* $\langle t, x \rangle \in \llbracket \mathrm{M} \rrbracket$.

(ii) $\ell_x \sqsubseteq [\heartsuit_\mu(x)]$ *for all* $x \in D$.

(iii) $t \sqsubseteq [\heartsuit_\mu(m_t)]$ *for all* $t \in \mathcal{T}$.

Proof. (i) ⇒ (ii). Let $x \in D$. Since $\ell_x \in \downarrow\ell_x = \llbracket \mathrm{M} \rrbracket^{-1}(x)$, we have $\langle \ell_x, x \rangle \in \llbracket \mathrm{M} \rrbracket$. The assumed condition (i) then implies that $\ell_x \sqsubseteq [\heartsuit_\mu(x)]$.

(ii) ⇒ (iii). Let $t \in \mathcal{T}$. By Lemma 7(ii) and the assumed condition (ii), $t \sqsubseteq \ell_{m_t} \sqsubseteq [\heartsuit_\mu(m_t)]$.

(iii) ⇒ (i). Suppose that $\langle t, x \rangle \in \llbracket \mathrm{M} \rrbracket$. Since $t \in \llbracket \mathrm{M} \rrbracket^{-1}(x) = \downarrow\ell_x$, we have $t \sqsubseteq \ell_x$. By the assumed condition (iii), $\ell_x \sqsubseteq [\heartsuit_\mu(m_{\ell_x})]$. Since $x \leq m_{\ell_x}$ by Lemma 7(ii), $\heartsuit_\mu(m_{\ell_x}) \subseteq \heartsuit_\mu(x)$ by Lemma 2(ii). Hence $t \sqsubseteq \ell_x \sqsubseteq [\heartsuit_\mu(m_{\ell_x})] \sqsubseteq [\heartsuit_\mu(x)]$. □

Finally, let's get back to the sentences in (2). They can now be analyzed as follows, where p^* denotes the utterance time point:

(9) a. There is some x such that $x \in \llbracket \text{milk in the tank} \rrbracket(p^*)$.
b. There is some x such that $x \in \llbracket \text{milk in the tank} \rrbracket(p^*)$ and $\mu^{p^*}_{\mathsf{liter}}(x) \geq 9.8$.

Note that a number of assumptions are at work under cover of the deceptively simple appearance of (9-a). By the definition of the derivative denotation for time points, (9-a) means that there exists an open $G \subseteq T$ such that $p^* \in G$ and $x \in \llbracket \text{milk in the tank} \rrbracket([G])$. By (6), there is a relevant family $\{\mu^p\}_{p \in T}$ of measures and $[G] \sqsubseteq [\heartsuit_\mu(x)]$. Adopting the way of talking in (5), this almost surely entails that $\mu^{p^*}(x) > 0$.

5 Continuous Production/Consumption

Let's move on to sentences describing continuous production or consumption of mass entities like the following:

(10) a. The cow produced 9.8 liters of milk yesterday.
b. The calf consumed 9.8 liters of milk yesterday.

When the subject is fixed to some particular individual like *the cow* or *the calf*, transitive verbs may be regarded as denoting a binary relation between times and entities.

While not necessary for analyzing (10), it might be illuminating to compare the denotations of these verbs with mass noun denotations.

An important property of these verbs is what might be called double cumulativity,[8] in the sense illustrated below:

(11) the cow produced x at s and y at t
$\Longrightarrow$ the cow produced $x \vee y$ at $s \sqcup t$

This property implies that the verb denotation is cumulative separately in the entity domain and in the temporal domain:[9]

(12) a. the cow produced x at t and y at t
$\Longrightarrow$ the cow produced $x \vee y$ at t
b. the cow produced x at s and x at t
$\Longrightarrow$ the cow produced x and $s \sqcup t$

However, these separate cumulative properties in the two domains do not entail double cumulativity. Indeed, the denotation of a mass noun is cumulative in the two separate domains as detailed in the previous section, but is not doubly cumulative:

(13) x is milk at s and y is milk at t
$\not\Longrightarrow$ $x \vee y$ is milk at $s \sqcup t$

Also, I say that verbs V of production or consumption (with a fixed singular subject) have double distributivity[10] in the following sense:

(14) If $\langle t, x\rangle \in [\![\mathrm{V}]\!]$, then for all nonzero $s \sqsubseteq t$, there is some nonzero $y \leq x$ such that $\langle s, y\rangle \in [\![\mathrm{V}]\!]$, and for all nonzero $y \leq x$, there is some nonzero $s \sqsubseteq t$ such that $\langle s, y\rangle \in [\![\mathrm{V}]\!]$.

However, this does not imply distributivity in each separate domain:

(15) a. the cow produced x at t and $y \leq x$
$\not\Longrightarrow$ the cow produced y at t
b. the cow produced x at t and $s \sqsubseteq t$
$\not\Longrightarrow$ the cow produced x at s

To understand this, imagine that the cow spent the time period $[(0, 1)]$ to produce y, a total of 5.5 liters of milk, and spent the time period $t = [(0, 2)]$ to produce x, a total of 9.8 liters of milk. Let $s = [(1, 2)]$. Then $y \leq x$ and $s \sqsubseteq t$, but no nonzero part of y was produced at any nonzero part of s. The assumed double distributivity then entails that the cow did not produce y at t nor did it produce x at s. This concludes that unlike in the case of mass nouns, neither $[\![\mathrm{V}]\!]\,(t)$ nor $[\![\mathrm{V}]\!]^{-1}(x)$ is an ideal. I would like to note, however, that the weaker form of distributivity in (16-a) should hold, as evidenced by the valid inference in (16-b).

(16) a. the cow produced $x \neq 0_D$ at t
$\Longrightarrow$ the cow produced some nonzero proper part of x at t
b. the cow produced 9.8 liters of milk at t
$\Longrightarrow$ the cow produced 5.5 liters of milk at t

Now, let me hope that I can come back to study lexical denotations of verbs of production/consumption in more detail in the future, and let me return to (10). A straightforward analysis of (10-a) will look something like the following:

(17) There is some x such that $x \in [\![\text{milk}]\!]\,(p^*)$, $\mu^{p^*}_{\mathsf{liter}}(x) \geq 9.8$, and $\langle \mathsf{ystd}, x\rangle \in [\![\text{the cow produced}]\!]$,

where p^* denotes the utterance time point and $\mathsf{ystd} \in \mathcal{T}$ is the equivalence class of the whole interval of yesterday. (17) can be equivalently expressed as follows:

$$\mu^{p^*}_{\mathsf{liter}}\Big(\bigvee\big(\,[\![\text{milk}]\!]\,(p^*) \cap [\![\text{the cow produced}]\!]\,(\mathsf{ystd})\big)\Big) \geq 9.8.$$

Imagine, however, that the calf consumed all the milk produced by the cow within yesterday. Then no portion of the milk in question remains today, so (17) will come out false. Can we perhaps fix this problem by changing p^* to some appropriate time point in yesterday? No. Imagine that the calf drank the milk directly from the breast of the dam as she produced it. If the calf's swallowing milk is to be understood as disappearance of the milk from the world, the milk in question that existed at any given time point p is the milk that was in the mouth of the calf at p, and it is quite possible that its volume was always below 9.8 liters.[11] We then might want to split yesterday into small subintervals, and sum up the amounts of milk that the cow produced at those

[8] Krifka (1989) calls this property *Summativity* for relations between events and objects.

[9] Although the kind of situation described in (10-b) is unrealistic, I see nothing wrong with the inference *per se*; so long as one can imagine one and the same entity can be produced twice, one sees that the inference goes through.

[10] This corresponds to the properties Krifka (1989) calls *Mapping to Objects* and *Mapping to Events*.

[11] Szabó (2006) discusses a similar problem involving count nouns such as *Helen had three husbands*, which can be true even if there was no past time when Helen had three husbands simultaneously.

intervals. I argue, however, that that appraoch is still insufficient for two reasons.

To illustrate my first point, let's say that within yesterday, the cow started milk production at time point p_0 and ended it at p_n. Then, we need only look at (the equivalence class of) the interval $[(p_0, p_n)]$. Let's say that we split this into n subintervals $t_1 = [(p_0, p_1)]$, $t_2 = [(p_1, p_2)]$, …, $t_n = [(p_{n-1}, p_n)]$ and want to see whether something like the following holds:

(18)
$$\sum_{k=1}^{n} \mu^{p_k}_{\text{liter}}\Big(\bigvee(\llbracket \text{milk} \rrbracket\ (p_k) \cap \llbracket \text{the cow produced} \rrbracket\ (t_k))\Big) \geq 9.8.$$

Unfortunately, exactly the same problem persists for each subinterval in principle: there is no guarantee that the portion of the milk produced during t_k remained wholly at p_k. However, by making the splitting finer and finer, we can expect to obtain a more and more precise measurement of the total milk production.

The second reason has to do with the fact that even in a situation where an entity keeps existing, its measurement can change throughout its life. (10-a) can be naturally used when the hearer has no idea that such milk existed. In that case, *9.8 liters of milk* is **nonpresuppositional**, and we will focus on this reading in this paper. Musan (1995) observes that the temporal interpretation of a nonpresuppositional noun phrase is obligatorily temporally dependent on the main predicate. In the German example in (19), unless *Einige* is stressed, which would indicate presuppositionality, the temporal interpretation of *Einige Professoren* must coincide with that of *glücklich*, so the sentence is only understood as talking about individuals who were professors in the sixties.

(19) Einige Professoren waren in den sechziger
some professors were in the sixties
Jahren glücklich.
happy
(Musan, 1995, p. 79)

As for (10-a), we see that it talks about entities that were milk at the time of production, as Musan's generalization predicts. What about the temporal interpretation of the measuring phrase *9.8 liters*? Intuitively, what matters here is the volume of milk as measured at the time of production. As a matter of fact, milk shrinks in volume when cooled. So if the produced milk was refrigerated in the tank after the milking, then its volume must have become smaller than at the time of production. However, none of such concerns seem to matter in judging whether (10-a) is true.[12] I therefore suppose that Musan's generalization extends to measuring phrases (Shimada, 2009). That is to say, the temporal interpretation of the measuring phrase of a nonpresuppositional noun phrase is obligatorily dependent on the main predicate. But then, we see a problem in the summands in (18), which are:

$$\mu^{p_k}_{\text{liter}}\big(\bigvee(\llbracket \text{milk} \rrbracket\ (p_k) \cap \llbracket \text{the cow produced} \rrbracket\ (t_k))\big).$$

Here, what is measured is the milk produced during t_k, but it is measure at p_k, the endpoint of t_k. This means that what is produced, say, in the first half of t_k is not measured at the time of its production. However, if we assume that volume change can only be so gradual, then making the splitting finer and finer will lead to a more and more precise measurement.

At this point, it must be evident that we need to look at infinitesimally small subintervals. In other words, we need the full power of integration. Indeed, Krifka (1989) suggests using a semantically well-motivated "calculus" (p. 91) to analyze such expressions as *drink wine*, and my idea is to carry this out literally.[13] Now, the set of entities that the cow produced during $[(p, p+h)]$, where $h > 0$, is given by

$$\llbracket \text{the cow produced} \rrbracket\ ([(p, p+h)]).$$

Suppose that x belongs to this set. Is x milk at $[(p, p+h)]$? Definitely not. Halfway into this time period, say at $p + h/2$, some part of x has yet to be produced, so not only was that part not milk, but it had measure zero. Therefore, in order to correctly predicate the mass noun *milk* of x, it has to be done at $p + h$, when all of x has been produced (ignoring the possible partial loss of x due to the calf consuming it by that point). Then, the total milk that the cow produced during $[(p, p+h)]$, for

[12] Granted that in reality, the volume of milk does not change so wildly as its temperature changes, but if one imagines that it does, this intuition will be clearer.

[13] In Shimada (2009), I treated mass nouns and verbs of production/consumption as essentially denoting relations between time points and entities. I think that that was the wrong idea, and now I am a proponent of the view that "times" that nouns and verbs are predicated of have positive length. From a measure-theoretic perspective, points are as good as nothingness. Simply put, size matters!

which we shall write $\alpha_{\mathsf{p}}([(p, p+h)])$, is

$$\alpha_{\mathsf{p}}([(p, p+h)]) = \bigvee(\llbracket\text{milk}\rrbracket\,(p+h) \cap \\ \llbracket\text{the cow produced}\rrbracket\,([(p, p+h)])),$$

and its size can be meaningfully measured, again, only at $p+h$, which is

$$\mu^{p+h}_{\mathsf{liter}}(\alpha_{\mathsf{p}}([(p, p+h)])).$$

Since the average rate of milk production during $[(p, p+h)]$ in terms of measurement at $p+h$ is given by dividing the above by h, the rate $\varrho_{\mathsf{p}}(p)$ of milk production at p will be obtained by taking its limit:

$$\varrho_{\mathsf{p}}(p) = \lim_{h \to 0^+} \frac{\mu^{p+h}_{\mathsf{liter}}(\alpha_{\mathsf{p}}([(p, p+h)]))}{h}.$$

The total amount of milk production is obtained by integrating it over yesterday, so the truth conditions of (10-a) will be expressed as

$$\int_{\mathsf{ystd}} \varrho_{\mathsf{p}}\, d\mu_{\mathrm{L}} \geq 9.8.$$

This is a **Lebesgue integral** (*vide* Halmos, 1950), and μ_{L} is the Lebesgue measure. Similarly, the rate $\varrho_{\mathsf{c}}(p)$ of milk consumption at p can be calculated by

$$\varrho_{\mathsf{c}}(p) = \lim_{h \to 0^+} \frac{\mu^{p-h}_{\mathsf{liter}}(\alpha_{\mathsf{c}}([(p-h, p)]))}{h},$$

where

$$\alpha_{\mathsf{c}}([(p-h, p)]) = \bigvee(\llbracket\text{milk}\rrbracket\,(p-h) \cap \\ \llbracket\text{the calf consumed}\rrbracket\,([(p-h, p)])).$$

To calculate $\varrho_{\mathsf{c}}(p)$, we now have to look at intervals ending at p since the milk consumed by the calf no longer exists after p. (10-b) can then be analyzed as

(20) $$\int_{\mathsf{ystd}} \varrho_{\mathsf{c}}\, d\mu_{\mathrm{L}} \geq 9.8.$$

6 Telicity and Integration

Since Verkuyl (1972), it has been known that the type of direct object can affect the aspectual interpretation of the whole VP. If the direct object is a bare noun as in (21), the VP becomes **atelic** (i.e. having no endpoint) and can take a *for* temporal PP.

(21) The calf consumed milk for 90 seconds yesterday.

In contrast, if the direct object is quantified as in (22) and (23), the VP becomes **telic** (i.e. having an endpoint) and incompatible with a *for* temporal PP.

(22) a. The calf consumed 9.8 liters of milk yesterday. (= (10-b))
b. *The calf consumed 9.8 liters of milk for 90 seconds yesterday.

(23) a. The calf consumed some milk yesterday.
b. *The calf consumed some milk for 90 seconds yesterday.

In this final section, I will sketch how our approach might deal with this phenomenon, leaving a fuller theoretical development and comparisons with existing theories (Krifka, 1989, 1992, 1998; Zucchi and White, 2001; Rothstein, 2004; Kovalev, 2024 *inter alia*) for a future occasion.

First, we need to know how *for* temporal PPs are analyzed. Obviously, they measure the length of times, and I say that this is achieved by means of integration of functions from T into $\mathbb{R}$. For concreteness, let's consider the following sentence:

(24) Ymir ran for six hours yesterday.

Assuming that the verb *ran* denotes a relation between times and entities, the denotation of *Ymir ran* will be the set of times where Ymir ran.[14] To calculate the total length of Ymir's running, we need to get the set R of time points at which Ymir was running, and this may be given by the following (cf. our definition and discussion of the mass noun denotation for time points):

$$R = \bigcup \{\, G \mid G \text{ open}, G \neq \varnothing, [G] \in \llbracket\text{Ymir ran}\rrbracket \,\}.$$

The PP *for six hours* says that the intersection of this set and the set of time points in yesterday is at least $6 \cdot 60^2$ seconds long. Assuming that μ_{L} measures times in seconds, this can be expressed, using a Lebesgue integral, as

$$\int_{\mathsf{ystd}} \chi_R\, d\mu_{\mathrm{L}} \geq 6 \cdot 60^2,$$

where χ_R is the characteristic function of R.

[14] Authors like Taylor (1977) and Dowty (1979) argue that activities (such as *run*) have minimal parts, and if that is the case, $\llbracket$Ymir ran$\rrbracket$ will not be an ideal in $\mathcal{T}$.

Let's return to (21). We can regard *the calf consumed milk* as denoting the set of times where the calf consumed milk. If we are to trace our analysis of (24) above, we should derive from this the set of time points at which the calf was consuming milk, and integrate its characteristic function. However, in this case, we can directly get to the time points at which the calf was consuming milk, as they must coincide with the time points at which the rate of milk consumption is positive.[15] Using ϱ_{C} from the previous section, (21) will then be analyzed as follows:

(25) $$\int_{\mathsf{ystd}} \chi_{\{\, p \in T \mid \varrho_{\mathsf{C}}(p) > 0 \,\}} \, d\mu_{\mathrm{L}} \geq 90.$$

It should now be apparent why (22-b) does not work. Both the expressions *9.8 liters* and *for 90 seconds* require integration of a function from T into $\mathbb{R}$, so we need two integrals. However, once the time-point variable is "used up" by one integral, it will no longer be available for the other. This point might become transparent if we rewrite the integrands in (20) and (25) in λ notation:

(20′) $$\int_{\mathsf{ystd}} [\lambda p \in T.\, \varrho_{\mathsf{C}}(p)] \, d\mu_{\mathrm{L}} \geq 9.8.$$

(25′) $$\int_{\mathsf{ystd}} [\lambda p \in T.\, \varrho_{\mathsf{C}}(p) > 0] \, d\mu_{\mathrm{L}} \geq 90.$$

As you can see, the time-point variable p gets bound in an integral, so it cannot be further used to define a meaningful integrand for another integral. This situation is analogous to the variable binding in the formula $\forall x \exists x P(x)$; the inner quantifier $\exists x$ binds the occurrence of x in $P(x)$, and as a result, the outer quantifier $\forall x$ cannot bind it.

The contrast in (23) presents difficulties to theories like Krifka's, which will expect the VP to be atelic because proper parts of some milk are still some milk. Our approach treats (23) in much the same way as (22). (23-a) involves integration of the rate of milk-consumption just like (22-a), except that the consumed amount is unspecified, so it is analyzed as follows:

$$\int_{\mathsf{ystd}} \varrho_{\mathsf{C}} \, d\mu_{\mathrm{L}} > 0.$$

(23-b) is bad for the same reason as (22-b) is.

Finally, we note that *in* temporal PPs will correspond to intervals of integration. It has been observed in the literature that *in* temporal PPs go with telic VPs, as demonstrated by (26).

(26) The calf consumed 9.8 liters of milk in two days.

Here, the PP *in two days* plays the same role as *yesterday* in (22-a), except that it is quantified. Therefore, it can be analyzed as something like the following:

(27) There is a time interval I (beginning at some contextually salient time point) such that $\mu_{\mathrm{L}}(I) = 2 \cdot 24 \cdot 60^2$ and

$$\int_I \varrho_{\mathsf{C}} \, d\mu_{\mathrm{L}} \geq 9.8.$$

In fact, a *for* temporal PP and an *in* temporal PP can co-occur as in (28), and this can be straightforwardly analyzed in much the same way, as shown in (29), where R is as defined earlier.

(28) Ymir ran for six hours in two days.

(29) There is a time interval I (beginning at some contextually salient time point) such that $\mu_{\mathrm{L}}(I) = 2 \cdot 24 \cdot 60^2$ and

$$\int_I \chi_R \, d\mu_{\mathrm{L}} \geq 6 \cdot 60^2.$$

7 Conclusion

By examining sentences involving mass nouns, we outlined an ontology based on the view that to be is to measure positively. Regarding mass noun denotations, we argued that our intuition is best captured if they are thought to give rise to Galois connections, and showed that it is possible to derive mass noun denotations for time points from those for time intervals, and vice versa. Our approach allows one to talk rigorously about changes of states and about events of continuous nature. In particular, we argued that sentences of continuous production or consumption inherently require mathematical integration. Since our theory is framed in terms of measure theory, they receive a natural treatment with Lebesgue integration. In fact, regardless of whether or not mass nouns (or events of continuous nature) are involved, it is expected that a great deal of existential quantification may be eliminated

[15] Accordingly, $[\![$the calf consumed milk$]\!]$ as a set of times will be given as the following principal ideal in $\mathcal{T}$:

$$[\![\text{the calf consumed milk}]\!] = {\downarrow}[\{\, p \in T \mid \varrho_{\mathsf{C}}(p) > 0 \,\}].$$

from the truth conditions of natural language sentences in favor of conditions on Lebesgue integrals; one would only need to invoke appropriate measures for involved nouns and verbs. Finally, we sketched how we might develop a theory of telicity on our approach. I hope to have demonstrated both the merit and necessity of a measure-theoretic approach to natural language semantics.

Obviously, we have only scratched the surface of this new direction of development, and much remains to be investigated and to be elaborated upon. For instance, we treated nouns and verbs as denoting relations between times and entities, but a more precise analysis should include eventualities. In such an analysis, mass nouns will be associated with states, and verbs of production or consumption with processes of some sort. The times that we have so far associated with nouns and verbs will correspond to (the equivalence classes of) the projections of eventualities onto the time axis via something like Krifka's (1989) temporal trace function. The different properties exhibited by denotations of nouns and verbs should then be ascribed to the different types of underlying eventualities. Also, while the Lebesgue-integral method yields a satisfactory treatment as far as meaning is concerned, it gives one a good deal of headache trying to work out how all this may be achieved compositionally.

Finally, a note on possible worlds is in order. So far, we have fixed the world parameter and focused exclusively on temporal changes, but much similar development is expected for worlds as well. If we trace our train of discussion, we should be dealing mainly with (equivalence classes of) sets of possible worlds of positive measure rather than with possible worlds *per se*, just as we have decided to deal mainly with sets of time points of positive measure rather than time points. Those sets of possible worlds could be viewed as representing partial information of a world, so they might be comparable to situations (Barwise and Perry, 1983; Kratzer, 1989). The obvious candidate for the measure on these "situations" will be one that assigns them their probabilities. By positing a "situational trace function" similar to the temporal trace function, predicates could be analyzed as relations between situations and entities, if the temporal parameter is fixed. Lebesgue integrals over situations will calculate expected values. All of this is mere speculation at this point, but I hope to delve deeper into these and other matters in future research.

Acknowledgments

This paper is a revision of the piece of writing that I contributed to the online Festschrift to celebrate the 60th birthdays of Makoto Kanazawa and Chris Tancredi, edited by Yasutada Sudo and Wataru Uegaki (`https://semanticstokyo.wordpress.com`). That writing was in turn a major revision of the paper I read at the open symposium "The Semantics of Intensional Phenomena" within the 37th annual conference of the English Linguistic Society of Japan, held at Kwansei Gakuin University in 2019, when Chris Tancredi invited me to give a talk as a member of his gang. The comments I gratefully received from Makoto when I presented this work at the 60th birthday event on January 6, 2025 at Keio University have been reflected in the present version.

References

Jon Barwise and John Perry. 1983. *Situations and Attitudes*. MIT Press, Cambridge, MA.

H. C. Bunt. 1979. Ensembles and the formal semantic properties of mass terms. In Francis Jeffery Pelletier, editor, *Mass Terms: Some Philosophical Problems*, pages 249–277. D. Reidel, Dordrecht.

Helen Cartwright. 1975. Amounts and measures of amount. *Noûs*, 9(2):143–164.

C.-Y. Cheng. 1973. Comments on Moravcsik's paper. In J. M. E. Moravcsik J. Hintikka and P. Suppes, editors, *Approaches to Natural Language*, pages 286–288. D. Reidel, Dordrecht.

David Dowty. 1979. *Word Meaning and Montague Grammar*. Kluwer, Dordrecht.

Steven Givant and Paul Halmos. 2009. *Introduction to Boolean Algebras*. Springer, New York.

Paul R. Halmos. 1950. *Measure Theory*. Springer, New York.

Jim Higginbotham. 1994. Mass and count quantifiers. *Linguistics and Philosophy*, 17(4):447–480.

Pavel Kovalev. 2024. *Modeling Telicity with Dependent Types*. Ph.D. thesis, Indiana University.

Angelika Kratzer. 1989. An investigation of the lumps of thought. *Linguistics and Philosophy*, 12(5):607–653.

Manfred Krifka. 1989. Nominal reference, temporal constitution and quantification in event semantics. In Renate Bartsch, Johan van Benthem, and Peter van Emde Boas, editors, *Semantics and Contextual Expressions*, pages 75–115. Foris, Dordrecht.

Manfred Krifka. 1992. Thematic relations as links between nominal reference and temporal constitution. In Ivan Sag and Anna Szabolcsi, editors, *Lexical Matters*, pages 29–53. CSLI, Stanford, CA.

Manfred Krifka. 1998. The origins of telicity. In Susan Rothstein, editor, *Events and Grammar*, pages 197–235. Kluwer, Dordrecht.

Gödehard Link. 1983. The logical analysis of plurals and mass terms: A lattice-theoretical approach. In Rainer Bäuerle, Christoph Schwarze, and Arnim von Stechow, editors, *Meaning, Use, and Interpretation of Language*, pages 303–323. Walter de Gruyter, Berlin.

Jan Tore Lønning. 1987. Mass terms and quantification. *Linguistics and Philosophy*, 10(1):1–52.

Saunders Mac Lane. 1998. *Categories for the Working Mathematician*, second edition. Springer, New York.

Renate Musan. 1995. *On the Temporal Interpretation of Noun Phrases*. Ph.D. thesis, MIT.

W. V. Quine. 1960. *Word and Object*. MIT Press, Cambridge, MA.

Peter Roeper. 1983. Semantics for mass terms with quantifiers. *Noûs*, 17(2):251–265.

Susan Rothstein. 2004. *Structuring Events*. Blackwell, Malden, MA.

Junri Shimada. 2009. *Measurement That Transcends Time: A Lebesgue Integral Approach to Existential Sentences*. Ph.D. thesis, MIT.

Robert M. Solovay. 1970. A model of set-theory in which every set of reals is Lebesgue measurable. *Annals of Mathematics, Second Series*, 92(1):1–56.

Zoltán Gendler Szabó. 2006. Counting across times. *Philosophical Perspectives*, 20:399–426.

Barry Taylor. 1977. Tense and continuity. *Linguistics and Philosophy*, 1(2):199–220.

Henk J. Verkuyl. 1972. *On the Compositional Nature of the Aspects*. Reidel, Dordrecht.

Giuseppe Vitali. 1905. *Sul problema della misura dei gruppi di punti di una retta*. Tip. Gamberini e Parmeggiani, Bologna.

Sandro Zucchi and Michael White. 2001. Twigs, sequences and the temporal constitution of predicates. *Linguistics and Philosophy*, 24(2):223–270.

The Computational Cost of Quantifier Raising

Ned Sanger
University of California, Los Angeles
nedsanger@ucla.edu

Abstract

Quantifier Raising (QR) is a tremendously popular tool for repairing type mismatches and analyzing scope in LF-based approaches to semantic interpretation. But despite its prominence, its computational properties are barely known. To fill this gap, I study the time complexity of the following decision problem: given a tree whose leaves are assigned semantic types, can repeated applications of QR lead to a well-typed logical form (LF)? The main result is that this problem is NP-complete, meaning that there exists no general and efficient algorithm for building interpretable LFs using QR (assuming P $\neq$ NP). The problem remains NP-complete even if one constrains QR by limiting the type of traces, limiting the number of atomic semantic types, banning parasitic scope, and banning cyclic QR.

1 Introduction

Any compositional semantic analysis has to answer two questions: what are the meaningful pieces, and how do they fit together? The need for answers becomes both pressing and difficult as soon as a syntactic structure contains type mismatches, or a subexpression has to take scope over a portion of its context. Semanticists have met these challenges over the years with a battery of methods like Quantifier Raising (May, 1977; Heim and Kratzer, 1998), Quantifying In (Montague, 1973), Cooper Storage (Cooper, 1983), Continuation-Passing Style (Barker and Shan, 2014; Kiselyov and Shan, 2014), and Flexible Montague Grammar (Hendriks, 1993). Varied as they are, at a high level these tools are all just ways of chopping up trees and piecing them back together in a fashion that makes the computation of meaning possible. One way or another, every compositional theory of semantic interpretation ends up sawing and gluing.

This paper studies that process of semantic reshuffling by taking a close look at Quantifier Raising. QR is a particularly simple and popular way to solve the problem of type mismatches and displaced scope, but despite fifty years of widespread use, surprisingly little is known about it computationally. In order to improve our formal understanding, I therefore answer some basic but unresolved questions about its complexity:

- Given a logical form that is *not* well-typed, how difficult is it to determine whether repeated applications of QR can turn it into one that *is*?
- Does the difficulty depend on the number of atomic types one uses? Does it depend on the possibility of higher-order traces, or the possibility of parasitic scope, or the possibility of QR applying to an element multiple times?
- Do we need any purely formal constraints on quantifier raising to keep it well-behaved?

My answer to the first question above is: *very* difficult (NP-complete). My answer to the second is: *no*, none of those constraints affects the difficulty (as measured by asymptotic, worst-case time complexity). My answer to the third question is: *yes*, we need a formal constraint that prevents QR from targeting λ-abstractions that were themselves created by QR.

To date, the only formal study of QR is Barker (2020). This paper looks at the operation in the same spirit that Barker did and builds on the foundation he laid.

The rest of the paper is organized as follows. Section 2 provides a formal definition of logical forms, of Quantifier Raising, and of the decision problem LF REPAIR: given a possibly ill-typed logical form, can repeated applications of QR make it well typed? The section also proves a couple of useful facts about logical forms. Section 3 proves that LF REPAIR belongs to the complexity class NP, following Barker (2020) closely. Section 4 proves that

LF REPAIR is NP-hard by a reduction from the DIRECTED HAMILTONIAN CYCLE problem. Section 5 makes a few concluding remarks.

I assume that the reader is familiar with basic notions from computational complexity theory, in particular the theory of NP-completeness. For a general introduction, see Papadimitriou (1994) and Arora and Barak (2009). For overviews focused on linguistic issues, see Pratt-Hartmann (2010) and Barton et al. (1987).

2 Preliminaries and Notation

2.1 Types, Logical Forms, Contexts

Given a finite set $\mathbf{A}$ of atomic types, the set of *types* over it is

$$\mathbf{T} ::= \mathbf{A} \mid \mathbf{T} \to \mathbf{T}.$$

In the rest of the paper, lowercase greek letters $(\alpha, \beta, \gamma, \ldots)$ will range over types. As usual, '$\to$' associates to the right. I often drop the '$\to$', so that $\alpha\beta\gamma$, for example, is a shorthand for the type $\alpha \to (\beta \to \gamma)$. I use $\alpha \xrightarrow{m} \beta$ as a shorthand for

$$\underbrace{\alpha \to \cdots \to \alpha}_{m \text{ times}} \to \beta$$

A useful way to stratify the set of types is by assigning each one an *order*. If we think of types as describing functions, then the order quantifies how complex that function's arguments are. Formally, the order of an atomic type p is $\operatorname{ord}(p) = 1$, and the order of a complex type $\alpha \to \beta$ is

$$\operatorname{ord}(\alpha \to \beta) = \max(1 + \operatorname{ord}(\alpha), \operatorname{ord}(\beta)).$$

Types of order n describe functions whose arguments have types of order less than n. Some examples: if $e, t \in \mathbf{A}$, then $\operatorname{ord}(e) = 1$, $\operatorname{ord}(eet) = 2$, and $\operatorname{ord}\big((et)et\big) = 3$.

Let $\{\mathbf{V}_\alpha\}_{\alpha \in \mathbf{T}}$ be a disjoint family of countable sets indexed by $\mathbf{T}$, and let

$$\mathbf{V} = \bigcup_{\alpha \in \mathbf{T}} \mathbf{V}_\alpha.$$

The members of each set $\mathbf{V}_\alpha = \{x, y, \ldots\}$ are *variables of type* α. Nodding at linguistic practice, I sometimes call variables *traces*. When the type of a variable x is relevant, I will write it as x^α to indicate that $x \in \mathbf{V}_\alpha$.

The set $\mathbf{L}$ of *logical forms* is

$$\mathbf{L} ::= \mathbf{T} \mid \mathbf{V} \mid \mathbf{L} \bullet \mathbf{L} \mid \lambda \mathbf{V} \circ \mathbf{L}.$$

In words, an LF is either a type, a variable, the concatenation of two LFs, or a λ-abstraction over an LF. An *abstraction-free* LF is one built from only the first three cases, i.e., an LF that contains no λ-abstractions. It's usual in semantics to place typed lexical items rather than types themselves at the leaves of logical forms. But since this paper is only concerned with type coherence, the definition above is simpler.

The constructor '$\bullet$' associates to the left and '$\circ$' to the right. Both are commutative, meaning $\alpha \bullet \beta$ and $\beta \bullet \alpha$ are considered equivalent, as are $\alpha \circ \beta$ and $\beta \circ \alpha$. '$\circ$' binds more tightly than '$\bullet$'.

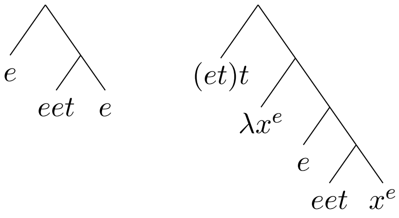
If $e, t \in \mathbf{A}$, then

$$e \bullet (eet \bullet e) \quad \text{and} \quad (et)t \bullet \lambda x^e \circ (e \bullet (eet \bullet x^e))$$

are each examples of logical forms. From now on I'll leave out the '$\circ$' after λ-abstractions, representing for instance the second LF above as

$$(et)t \bullet \lambda x^e (e \bullet (eet \bullet x^e)).$$

I sometimes present LFs as unordered binary trees in a way familiar to linguists, e.g.



Later on I will need to talk about LFs with a "hole" that can be plugged by another LF. I call these *contexts*. Formally, the set of contexts is

$$\mathbf{C} ::= [\,] \mid \mathbf{L} \bullet \mathbf{C} \mid \lambda \mathbf{V} \circ \mathbf{C}.$$

If Γ is a context and Δ is an LF (or context), then $\Gamma[\Delta]$ is the LF (or context) that results from substituting Δ for $[\,]$ in Γ. In Section 4, it will be useful to have notation for repeatedly nesting a context inside itself: if Γ is a context, let $\Gamma^0 = [\,]$ and $\Gamma^n = \Gamma[\Gamma^{n-1}]$. In the rest of the paper, capital Greek letters (Γ, Δ, etc.) will denote either LFs or contexts.

The *typing function* $\tau : \mathbf{L} \to \mathbf{T}$ is the following partial map:

$$\tau(\Gamma) = \begin{cases} \alpha & \text{if } \Gamma = \alpha, \\ \beta & \text{if } \Gamma = \Delta \bullet \Theta, \tau(\Delta) = \alpha\beta \\ & \quad \text{and } \tau(\Theta) = \alpha, \\ \alpha\beta & \text{if } \Gamma = \lambda x^\alpha\, \Delta[x^\alpha] \text{ and} \\ & \quad \tau(\Delta[\alpha]) = \beta, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The second and third cases correspond respectively to the semantic rules of *function application* and *predicate abstraction* in Heim and Kratzer (1998). When $\tau(\Gamma) = \alpha$ holds, I will usually just say Γ has type α. I occasionally use the notation $\Gamma : \alpha$ to mean that either $\tau(\Gamma) = \alpha$ or $\Gamma = x$ for some $x \in \mathbf{V}_\alpha$. (Note that $\tau(x^\alpha)$ is formally undefined.)

An LF Γ is *well typed* iff $\tau(\Gamma)$ is defined. The two LFs mentioned above, $e \bullet (eet \bullet e)$ and $(et)t \bullet \lambda x^e\, (e \bullet (eet \bullet x^e))$, each have type t and are therefore well typed.

A trace x in an LF is *bound* if it occurs within a subtree of the form $\lambda x\, \Sigma[x]$. Otherwise it's *unbound*. Notice that an LF containing an unbound trace cannot be well typed. The definition of the typing function therefore enforces a form of the empirically motivated proposal that remnant movement must reconstruct (Huang, 1993; Bhatt and Dayal, 2007): movement steps that create an unbound trace are always "undone" before interpretation begins.

2.2 Quantifier Raising

The surface syntax of a sentence is often ill-typed and therefore not suitable for semantic interpretation. A classic example comes from transitive sentences with a quantifier in object position ("the critic panned every film"), which correspond to the ill-typed LF $e \bullet (eet \bullet (et)t)$. Function application can't combine the type of a transitive verb, eet, with the type of a generalized quantifier, $(et)t$.

In this case, we know what LF we would *like* to interpret (Heim and Kratzer, 1998, 178–79). The proper semantic argument for the quantifier in object position is an abstraction over its context, also known as its continuation, so the LF we want is

$$\underbrace{(et)t}_{\text{quantifier}} \bullet \underbrace{\lambda x^e\, (e \bullet (eet \bullet x^e))}_{\text{continuation}}.$$

Quantifier Raising is just an operation for moving from ill-typed LFs like $e \bullet (eet \bullet (et)t)$ to the well-typed one above. Given an LF $\Gamma[\Delta[\Theta]]$, QR "raises" Θ so that it becomes Δ's sister, puts a variable in Θ's old position, and adds a binder for that variable between Θ and Δ. The result is $\Gamma[\Theta \bullet \lambda x\, \Delta[x]]$. In the case above,

$$\Gamma = [\,], \ \ \Delta = e \bullet (eet \bullet [\,]), \ \ \Theta = (et)t.$$

My goal is to study QR in its most general form. Accordingly, I won't constrain it by—just to name a few possibilities—imposing scope islands, scope economy (Fox, 2000), or limits on the types of traces (Poole, 2024). Whatever empirical merits they might have, these restrictions would only weaken the mathematical results in this paper. More importantly, to understand the effect (if any) that specific constraints have on complexity, it's necessary to first understand the complexity of the general case.

That said, *one* formal constraint is necessary to keep QR from running totally amok: the operation should be disallowed from targeting abstractions, that is, LFs of the form $\lambda x\, \Gamma$. To illustrate what can go wrong without this constraint, look at Figure 1. The leftmost tree shows an innocent-looking analysis of the sentence *Mary gave John the book*. It might not seem like QR can do anything interesting to this tree, because no subtree has a complex enough type to take scope. But the ability to raise abstractions lets us create a scope-taker.

The middle tree is the result of applying QR to *gave*, moving it to a position right below *Mary* and leaving a trace of type eet. The rightmost tree is the result of quantifier raising the *abstraction* created by the previous step to the root of the tree and leaving a trace of type e. Not only is the resulting LF well typed, but (assuming natural denotations for the lexical items) it means *the book gave John Mary*.

Completely unrestricted QR therefore allows us to create scope-takers on the fly, even in sentences where nothing ought to be taking scope, and the semantic effect is disastrous. It does not behave in the way linguists expect QR to behave and it lacks good theoretical properties. The definition below therefore explicitly prevents QR from targeting abstractions.[1]

Definition 2.1. *Quantifier Raising* is the smallest binary relation $\rightarrow_{\text{QR}} \subseteq \mathbf{L} \times \mathbf{L}$ such that for all contexts Γ and Δ, all LFs Θ that are *not* abstractions, and all fresh variables x,

$$\Gamma[\Delta[\Theta]] \rightarrow_{\text{QR}} \Gamma[\Theta \bullet \lambda x\, \Delta[x]].$$

The reflexive, transitive closure of $\rightarrow_{\text{QR}}$ is $\rightarrow^*_{\text{QR}}$.

[1] A more elegant way to ban QR of abstractions is to change the structure of LFs so that no constituent corresponds to an abstraction in the first place. Variable binding would instead be handled by an index on moved items, so that QR would relate an LF $\Gamma[\Delta[\Theta]]$ to something like $\Gamma[\Theta_x \bullet \Delta[x]]$ (Heim, 1997; Heim and Kratzer, 1998, 187–88). But this approach can't easily accommodate *parasitic scope* (Barker, 2007). Since I want a formalization of quantifier raising that covers all of its uses in linguistics, I've therefore stuck with tradition and used λ-abstraction to bind variables.

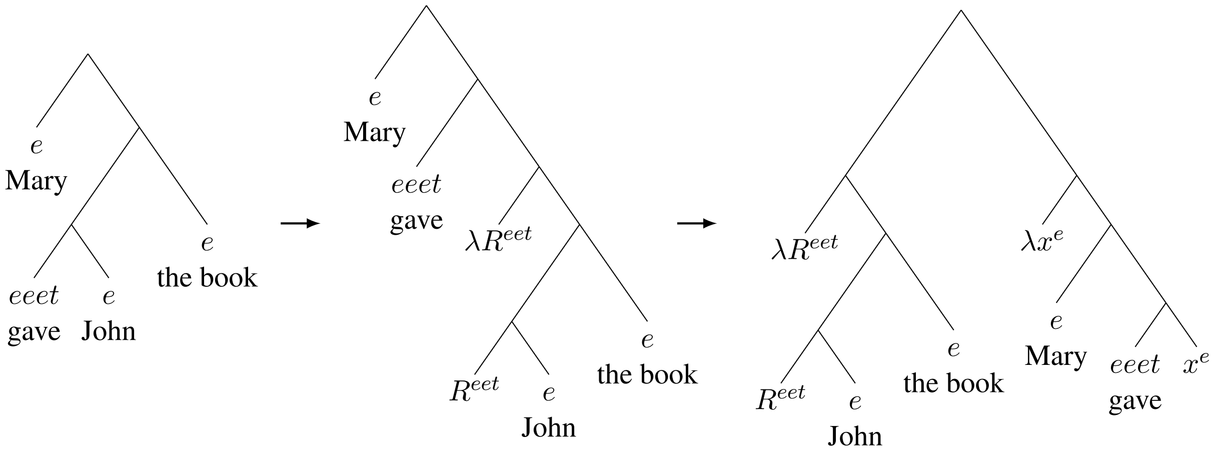


Figure 1: Raising abstractions is dangerous.

Because this operation targets not just quantifiers but any part of an LF at all, the name “Quantifier Raising” is misleading. “Scope Taking,” as Barker (2020) says, is more accurate. But like him, I’ve stuck with the traditional, firmly entrenched name.

Now we can define the central decision problem of this paper:

> LF REPAIR
> Given a set of atomic types **A** and an LF Γ, is there an LF Δ of type α such that $\Gamma \rightarrow^*_{\text{QR}} \Delta$?

The name comes from the idea that ill-typed LFs are “broken,” and that repairing them involves raising the scope-takers and finding appropriate trace types in such a way that the result is well-typed.

Here is an example. Let $\mathbf{A} = \{e, t\}$ and let Γ be the ill-typed LF

$$\big[(et)e \bullet \big[(((et)et)et)et \bullet et\big]\big] \bullet \big[eet \bullet (et)t\big].$$

Is there a Δ of type t such that $\Gamma \rightarrow^*_{\text{QR}} \Delta$? The answer, not immediately obvious, is yes:

$$\begin{aligned}\Gamma \rightarrow_{\text{QR}}\ & (et)t \bullet \lambda x^e\big[\big[(et)e \bullet \big[(((et)et)et)et \bullet et\big]\big] \\ & \bullet \big[eet \bullet x^e\big]\big] \\ \rightarrow_{\text{QR}}\ & (et)t \bullet \big[(((et)et)et)et \bullet \lambda Q^{(et)et}\lambda x^e \\ & \big[\big[(et)e \bullet \big[Q^{(et)et} \bullet et\big]\big] \bullet \big[eet \bullet x^e\big]\big]\big],\end{aligned}$$

and this last LF has type t.

2.3 Normalization

QR’s purpose is to let a function take some portion of its semantic context as an argument. But it’s also possible for the raised element to become an argument rather than a function of its context: for example, $e \bullet et \rightarrow_{\text{QR}} e \bullet \lambda x^e(x^e \bullet et)$. Such steps are licit but pointless—they will never help turn an ill-typed LF into a well-typed one. In general, if an LF has the form $\Delta[\Pi \bullet \lambda x\, \Sigma[x]]$, where $\Pi : \alpha$ and $\lambda x\, \Sigma[x]$ has type $\alpha\beta$, then I call the abstraction $\lambda x\, \Sigma[x]$ *vacuous*. A well-typed LF is *normalized* iff it contains no vacuous abstractions.

The next lemma justifies the label “vacuous.”

Lemma 2.1. *Let Γ be a variable- and abstraction-free LF. If $\Gamma \rightarrow^*_{\text{QR}} \Delta$, where Δ has type α, then there exists a normalized LF Δ' of type α such that $\Gamma \rightarrow^*_{\text{QR}} \Delta'$.*

Proof. If Δ isn’t already normalized, then it contains a vacuous abstraction and therefore has the form $\Theta[\Pi \bullet \lambda x^\beta\, \Sigma[x^\beta]]$, where $\Pi : \beta$ and $\lambda x^\beta\, \Sigma[x^\beta]$ has type $\beta\gamma$. By definition, QR cannot target an abstraction. And since Δ is well-typed, no step in the derivation $\Gamma \rightarrow^*_{\text{QR}} \Delta$ raises x^β outside the scope of its binder λx^β. The derivation $\Gamma \rightarrow^*_{\text{QR}} \Delta$ therefore has the following form:

$$\begin{aligned}\Gamma \rightarrow^*_{\text{QR}}\ & \Theta_1[\Sigma_1[\Pi_1]] \\ \rightarrow_{\text{QR}}\ & \Theta_1[\Pi_1 \bullet \lambda x^\beta\, \Sigma_1[x^\beta]] \\ \rightarrow_{\text{QR}}\ & \Theta_2[\Pi_2 \bullet \lambda x^\beta\, \Sigma_2[x^\beta]] \\ \rightarrow_{\text{QR}}\ & \cdots \\ \rightarrow_{\text{QR}}\ & \Theta_n[\Pi_n \bullet \lambda x^\beta\, \Sigma_n[x^\beta]],\end{aligned}$$

where the Θ_i and Σ_i are contexts; the Π_i are LFs; and $\Theta_n = \Theta$, $\Sigma_n = \Sigma$, and $\Pi_n = \Pi$. Then

$$\begin{aligned}\Gamma \rightarrow^*_{\text{QR}}\ & \Theta_1[\Sigma_1[\Pi_1]] \\ \rightarrow_{\text{QR}}\ & \Theta_2[\Sigma_2[\Pi_2]] \\ \rightarrow_{\text{QR}}\ & \cdots \\ \rightarrow_{\text{QR}}\ & \Theta_n[\Sigma_n[\Pi_n]]\end{aligned}$$

is a valid derivation of $\Theta[\Sigma[\Pi]]$ from Γ. Like Δ, the LF $\Theta[\Sigma[\Pi]]$ also has type α, but it contains one fewer vacuous abstraction.

By repeating this process, we can eliminate vacuous abstrations one by one until we obtain a normalized LF Δ' of type α such that $\Gamma \to^*_{\text{QR}} \Delta'$. □

The following lemma, needed in Section 4, makes it easier to check whether a variable- and abstraction-free LF Γ can be repaired. It turns out that the order of the traces created by each QR step never has to exceed a constant fixed by Γ.

Lemma 2.2. *Suppose Γ is a variable- and abstraction-free LF, Δ is a normalized LF, and $\Gamma \to^*_{\text{QR}} \Delta$. Let m be the maximum order among Γ's leaves. If x^α is a trace occurring in Δ, then* $\operatorname{ord}(\alpha) < m - 1$.

Proof. Let n be the maximum order among all of Δ's traces, and suppose $n \geq m - 1$. Consider a trace x^α of order n and write

$$\Delta = \Theta[\Pi \bullet \lambda x^\alpha\, \Sigma[x]].^{2}$$

Because Δ is normalized, Π's type has the form $(\alpha \to \tau(\Sigma[\alpha])) \to \beta$ for some β, and so

$$\operatorname{ord}(\tau(\Pi)) > \operatorname{ord}(\alpha) + 1 = n + 1. \qquad (1)$$

Every node within the subtree Π is either (i) a leaf of order at most $m \leq n + 1$, (ii) function application of two elements of order at most $n + 1$, or (iii) an abstraction over a variable of order at most n. It follows (by an induction from the leaves to the root) that every node of Π has order at most $n + 1$, so $\operatorname{ord}(\tau(\Pi)) \leq n + 1$. This contradicts (1). Therefore $n < m - 1$. □

3 LF REPAIR is in NP

This goal of this section is to prove that LF REPAIR is in NP. This result would be a simple corollary of the proof in Barker (2020) that LF REPAIR is decidable, but it turns out the proof is not valid.[3] The rest of this section provides a corrected proof, and uses it to show that LF REPAIR is in NP. Because many of the details from Barker's paper don't need to change, the proofs in this section move fast; see his paper for further information.

The plan, following Barker's lead, is to define a type-logical grammar called QRT (Quantifier Raising with Types) that captures the logic of quantifier raising in the following sense: if Γ is an LF, then $\Gamma \vdash A$ is a theorem of QRT if and only if $\Gamma \to^*_{\text{QR}} \Delta$ for some Δ of type A.[4] I show below that provability in QRT is in NP. Since LF REPAIR clearly has the same time complexity as provability in QRT, it must also be in NP.

The axiom and inference rules of QRT are shown in Figure 2. Γ, Δ and Θ represent LFs or contexts, and A and B represent types. $\lambda^{\uparrow}$ and $\lambda^{\downarrow}$ are subject to the restriction that the LF Θ not be a λ-abstraction (or equivalently, that Θ be either a type or of the form $\Sigma \bullet \Omega$). This restriction, which is the main difference between the version of QRT in this paper and the one in Barker (2020), will guarantee that the critical Theorem 3.2 holds. Unrestricted raising—which is what caused problems in Barker's proof—wreaks havoc when combined with the commutativity of '•'. For instance, it lets us freely swap an inner and outer context above a '•'-node, which results in massive overgeneration:

$$\begin{array}{cl}
\Gamma[\Delta[A \bullet B]] \vdash C & \\ \cline{1-1}
\Gamma[A \bullet \lambda x\, \Delta[x \bullet B]] \vdash C & \lambda^{\downarrow} \\ \cline{1-1}
(\lambda x\, \Delta[x \bullet B]) \bullet \lambda y\, \Gamma[A \bullet y] \vdash C & \lambda^{\downarrow} \\ \cline{1-1}
\Delta[(\lambda y\, \Gamma[A \bullet y]) \bullet B]) \vdash C & \lambda^{\uparrow} \\ \cline{1-1}
\Delta[\Gamma[A \bullet B]] \vdash C & \lambda^{\uparrow}
\end{array}$$

This issue is not really different from the one in Figure 1, and my solution is similar: forbidding $\lambda^{\uparrow}$ and $\lambda^{\downarrow}$ from targeting abstractions.

Figure 3 shows a small QRT proof.

There is a natural decision problem associated with QRT. As mentioned above, it has exactly the same time complexity as LF REPAIR.

> QRT PROVABILITY
> Given a sequent, is it a theorem of QRT?

The next theorem shows that cut-elimination is possible in proofs of QRT sequents that are *abstraction-free*, that is, sequents $\Gamma \vdash A$ where Γ contains no λ-abstractions.

Theorem 3.1. *Every abstraction-free theorem of QRT has a* CUT*-free proof.*

Proof. By the normal method of permuting cuts upward. When the cut formula is not principal in

[2] Technically, there could be more abstractions intervening, so that I should write $\Theta[\Pi \bullet \lambda y_1^{\beta_1} \ldots \lambda y_k^{\beta_k}\, \lambda x^\alpha\, \Sigma[x^\alpha]]$. But this doesn't affect the rest of the proof.

[3] Barker points out the problem himself in a footnote added after the early-access version of the paper was published.

[4] For a proof which carries over with minimal changes to the version of QRT in this paper, see the appendix to Barker (2020).

$$\dfrac{}{A \vdash A}\,\textsc{Ax} \qquad \dfrac{\Gamma \vdash A \qquad \Delta[B] \vdash C}{\Delta[AB \bullet \Gamma] \vdash C}\,{\to}\mathrm{L} \qquad \dfrac{\Gamma[A] \vdash B}{\lambda x^A\,\Gamma[x^A] \vdash AB}\,{\to}\mathrm{R}$$

$$\dfrac{\Gamma[\Theta \bullet \lambda x\,\Delta[x]] \vdash A}{\Gamma[\Delta[\Theta]] \vdash A}\,\lambda^{\uparrow} \qquad \dfrac{\Gamma[\Delta[\Theta]] \vdash A}{\Gamma[\Theta \bullet \lambda x\,\Delta[x]] \vdash A}\,\lambda^{\downarrow} \qquad \dfrac{\Gamma \vdash A \qquad \Delta[A] \vdash B}{\Delta[\Gamma] \vdash B}\,\textsc{Cut}$$

Figure 2: Axiom and Rules of QRT.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{e \vdash e}\,\textsc{Ax} \quad \dfrac{\dfrac{}{et \vdash et}\,\textsc{Ax} \quad \dfrac{}{t \vdash t}\,\textsc{Ax}}{(et)t \bullet et \vdash t}\,{\to}\mathrm{L}}{(et)t \bullet (eet \bullet e) \vdash t}\,{\to}\mathrm{L}}{\lambda x^e\,((et)t \bullet (eet \bullet x^e)) \vdash et}\,{\to}\mathrm{R} \qquad \dfrac{}{t \vdash t}\,\textsc{Ax}}{(et)t \bullet \lambda x^e\,((et)t \bullet (eet \bullet x^e)) \vdash t}\,{\to}\mathrm{L}}{(et)t \bullet (eet \bullet (et)t) \vdash t}\,\lambda^{\uparrow}$$

Figure 3: An example QRT derivation.

both premises of Cut, the proof transformations are straightforward.

The subtle case is when the cut formula *is* principal in both premises:

$$\dfrac{\dfrac{\Gamma[A] \vdash B}{\lambda x\,\Gamma[x] \vdash AB}\,{\to}\mathrm{R} \qquad \dfrac{\Delta \vdash A \qquad \Theta[B] \vdash C}{\Theta[AB \bullet \Delta] \vdash C}\,{\to}\mathrm{L}}{\Theta[(\lambda x\,\Gamma[x]) \bullet \Delta] \vdash C}\,\textsc{Cut}$$

Notice that Δ cannot be an abstraction. To see why, first define a sequent to be *stuck* if its left-hand side has the form $\Sigma[\Phi \bullet \Psi]$ where Φ and Ψ are both abstractions. A straightforward induction shows that if a sequent in a QRT proof is stuck, then every subsequent sequent—and in particular the conclusion—is stuck too. Now if Δ *were* an abstraction, then $\Theta[(\lambda x\,\Gamma[x]) \bullet \Delta] \vdash C$ would be stuck and the proof's conclusion would contain an abstraction, contrary to assumption. Δ is therefore not an abstraction, and we can replace the cut with two smaller ones:

$$\dfrac{\dfrac{\dfrac{\Delta \vdash A \qquad \Gamma[A] \vdash B}{\Gamma[\Delta] \vdash B}\,\textsc{Cut} \qquad \Theta[B] \vdash C}{\Theta[\Gamma[\Delta]] \vdash C}\,\textsc{Cut}}{\Theta[(\lambda x\,\Gamma[x]) \bullet \Delta] \vdash C}\,\lambda^{\downarrow}$$

□

Theorem 3.2. *Every abstraction-free theorem of QRT has a* Cut- *and* $\lambda^{\downarrow}$*-free proof.*

Proof. Take a Cut-free proof of an abstraction-free theorem, and consider an occurrence of $\lambda^{\downarrow}$ of maximal depth. Because the proof's conclusion is abstraction-free, there must be a lower occurrence of $\lambda^{\uparrow}$ eliminating the abstraction introduced by $\lambda^{\downarrow}$. If the occurrence of $\lambda^{\uparrow}$ is immediately below, then the application of $\lambda^{\downarrow}$ is eliminable:

$$\dfrac{\dfrac{\Gamma[\Delta[\Theta]] \vdash A}{\Gamma[\Theta \bullet \lambda x\,\Delta[x]] \vdash A}\,\lambda^{\downarrow}}{\Gamma[\Delta[\Theta]] \vdash A}\,\lambda^{\uparrow} \quad \rightsquigarrow \quad \Gamma[\Delta[\Theta]] \vdash A$$

Otherwise we can permute the occurrence of $\lambda^{\downarrow}$ downward across any occurrence of →L, →R, and $\lambda^{\uparrow}$ until it meets up with a suitable lower occurrence of $\lambda^{\uparrow}$ and can be removed. (The requirement that $\lambda^{\uparrow}$ and $\lambda^{\downarrow}$ not target abstractions guarantees that it is possible to permute $\lambda^{\downarrow}$ across $\lambda^{\uparrow}$ whenever needed.)

By repeating this procedure, we can eliminate each occurrence of $\lambda^{\downarrow}$ one by one. Since the procedure doesn't introduce any new applications of Cut, the end result is a Cut- and $\lambda^{\downarrow}$-free proof. □

Theorem 3.3. QRT PROVABILITY *is in NP.*

Proof. A sequent $\lambda x^A\,\Gamma[x^A] \vdash AB$ is provable if and only if $\Gamma[A] \vdash B$ is, and $\Gamma[\Sigma \bullet \lambda x\,\Delta[x]] \vdash A$ is provable if and only if $\Gamma[\Delta[\Sigma]] \vdash A$ is. Given an arbitrary sequent, we can use these facts to efficiently find an *abstraction-free* sequent that is provable iff the original sequent is. QRT-provability for arbitrary sequents is therefore reducible in polynomial time to QRT-provability of *abstraction-free* sequents, which we now show is in NP.

Theorem 3.2 shows that every abstraction-free theorem of QRT has a proof using only the rules AX, →L, →R, and $\lambda^{\uparrow}$. Read bottom to top, each application of →L and →R eliminates an occurrence of '→', and no rule introduces new ones. The number of uses of each of these rules in a proof is therefore bounded by the number of '→'s in the conclusion. Since each application of $\lambda^{\uparrow}$ introduces a λ-abstraction that must be removed by a corresponding application of →R, it can only be used as many times as →R is. Every abstraction-free theorem of QRT therefore has a proof whose size is polynomial in the theorem's length and which can serve as an efficient witness of theoremhood. □

Since LF REPAIR has the same time complexity as provability in QRT, the result we set out to show follows:

Theorem 3.4. LF REPAIR *is in NP.*

4 LF REPAIR is NP-Hard

4.1 Directed Graphs and Hamiltonian Cycles

The proof in this section uses some terminology from graph theory. As a reminder, a directed graph (or digraph) is a pair $\langle V, E\rangle$, where V is a finite set of *vertices* and $E \subseteq V \times V$ a set of *edges*. The first component of an edge is its *tail*, the second its *head*. Given a vertex v, its *indegree* $\deg^-(v)$ and *outdegree* $\deg^+(v)$ are the number of edges of which v is a head and tail, respectively.

A *Hamiltonian cycle* in a digraph is a sequence of edges $e_1, e_2, \ldots, e_n$ such that for $0 < i < n$, the head of e_i is the tail of e_{i+1}, the head of e_n is the tail of e_1, and every vertex occurs exactly once as a head and a tail of an edge in the sequence. The following decision problem is NP-complete (see, e.g., Garey and Johnson 1979):

> DIRECTED HAMILTONIAN CYCLE
> Given a digraph, does it have a Hamiltonian cycle?

4.2 Encoding Hamiltonian Cycles in LFs

This section describes a polynomial-time reduction from DIRECTED HAMILTONIAN CYCLE to LF REPAIR. Let $G = \langle V, E\rangle$ be a digraph with $n > 0$ vertices and $m \geq n$ edges. It's convenient to assume that the vertices of G are some initial segment of the natural numbers, i.e. $V = \{1, \ldots, n\}$. I will construct an instance of LF REPAIR whose answer is "yes" iff G has a Hamiltonian cycle. The method has some similarities to the one in Krantz and Mobile (2001), who encode DIRECTED HAMILTONIAN CYCLE in the multiplicative fragment of linear logic.

The set of atomic types for the instance consists of a type v for each $v \in V$, plus the additional types a, b, c, d, e, and t. That is,

$$\mathbf{A} = \{1, \ldots, n\} \cup \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}, \mathsf{t}\}.$$

When describing the reduction below, I'll use the following abbreviations for types:

$$\begin{aligned} \mathrm{E}^v_u &:= (\mathsf{a} \to u) \to \mathsf{b} \to v, \\ \mathrm{V}_v &:= (\mathsf{a} \to v) \to \mathsf{c} \to v, \\ \mathrm{G}_v &:= (\mathsf{a} \to \mathsf{t}) \ \to \mathsf{d} \to v, \\ \mathrm{G}^v &:= (\mathsf{a} \to v) \to \mathsf{e} \to \mathsf{t}. \end{aligned}$$

The rest of this subsection shows how to build an LF Γ such that $\Gamma \to^*_{\mathrm{QR}} \Delta$ for some Δ of type t if and only if G has a Hamiltonian cycle. Without loss of generality, I assume the cycle begins at the vertex 1. To make the construction more concrete, I will illustrate it as I go using the digraph in Figure 4. Notice that the graph has a Hamiltonian cycle: $(1,3), (3,2), (2,4), (4,1)$.

I'll build Γ up in four layers, defining an LF Γ_1 and three contexts $\Gamma_2, \Gamma_3, \Gamma_4$, such that $\Gamma = \Gamma_4[\Gamma_3[\Gamma_2[\Gamma_1]]]$. The plan is as follows. Γ_1 will contain a leaf E^v_u for each edge (u, v) of G, as well as a leaf V_v for every vertex. Each one of these leaves will need to undergo QR to have a chance of making Γ into a well-typed LF. The context Γ_2 will have n positions to which n of the leaves E^v_u from Γ_1 can undergo QR, as well as n positions for the V_v. By design, the types will properly compose if and only if the n corresponding edges in G form a Hamiltonian cycle.

The $m - n$ leaves E^v_u corresponding to edges not occurring in the cycle will also need a place that they can QR to. The context Γ_4 will provide $m - n$ positions for them. Since a Hamiltonian cycle enters and exits every vertex exactly once, for each vertex v there are $\deg^-(v) - 1$ edges with head v and $\deg^+(v) - 1$ edges with tail v that do not occur in the cycle. Γ_3 will accordingly contain $\deg^+(v) - 1$ leaves G_v and $\deg^-(v) - 1$ leaves G^v. For each E^v_u that QRs into Γ_4, G_u and G^v will QR right below and above it in Γ_4. This last step is needed to make all the types combine.

Now for the technical details. Let $(u_1, v_1), \ldots, (u_m, v_m)$ be G's edges, and let

$$\Gamma_1 = (\mathsf{a} \xrightarrow{m+n} 1) \bullet \mathrm{E}^{v_1}_{u_1} \bullet \ldots \bullet \mathrm{E}^{v_m}_{u_m} \bullet \mathrm{V}_1 \bullet \ldots \bullet \mathrm{V}_n.$$

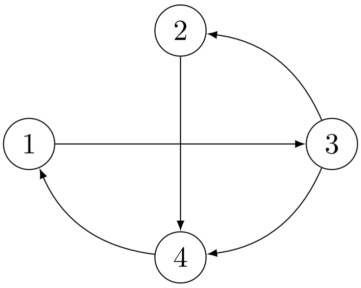
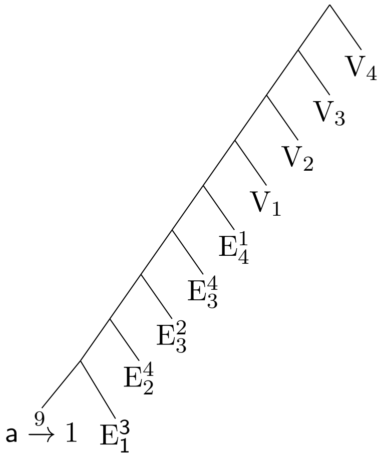


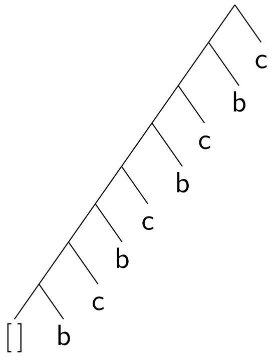
Figure 4: A directed graph.

For the digraph in Figure 4, Γ_1 would be



The intention is that every E^v_u and V_v will take scope and leave a trace of type a. The leaf of type $\mathsf{a} \xrightarrow{m+n} 1$ will consume these traces one by one, producing a subtree of type 1.

Let $\Delta = [\,] \bullet \mathsf{b} \bullet \mathsf{c}$ and let $\Gamma_2 = \Delta^n$. For the digraph in Figure 4, $\Gamma_2 = \Delta^4$, which looks like



If E^v_u and V_v each QR directly below the occurrences of b and c respectively in $\Delta[u]$, then the resulting subtree will have type v. Δ is therefore a gadget for simulating the traversal of an edge from u to v. The left side of Figure 5 illustrates the simulation with E^3_1 and V_3. Just for intuition, one can imagine that QR of E^v_u takes us from the state "exiting u" to the state "entering v", and QR of V_v takes us from the state "entering v" to the state "exiting v".

By nesting n copies of Δ, Γ_2 can simulate a path of length n in G. Notice that simulating the traversal of an edge with head v requires raising an occurrence of V_v. Since Γ will contain exactly one leaf V_v for each vertex v, the paths Γ_2 can simulate are those that visit each vertex exactly once, that is, Hamiltonian paths.

Γ_3 is most complicated. First define

$$a \dot{-} b = \max(a-b, 0).$$

Now for each $v \in V$ let

$$\Phi_v = [\,]\bullet \underbrace{\mathrm{G}_v \bullet \ldots \bullet \mathrm{G}_v}_{\deg^+(v) \dot{-} 1 \text{ times}}, \quad \Psi_v = [\,]\bullet \underbrace{\mathrm{G}^v \bullet \ldots \bullet \mathrm{G}^v}_{\deg^-(v) \dot{-} 1 \text{ times}}$$

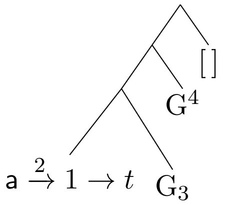
and let

$$\Phi = \Phi_1[\Phi_2[\ldots \Phi_n]] \quad \text{and} \quad \Psi = \Psi_1[\Psi_2[\ldots \Psi_n]].$$

Also let

$$k = \sum_{v=1}^{n} (\deg^+(v) \dot{-} 1) + (\deg^-(v) \dot{-} 1)$$

Then $\Gamma_3 = \Psi[\Phi[\mathsf{a} \xrightarrow{k} 1 \rightarrow t]] \bullet [\,]$. In the case of the digraph in Figure 4 (where only Φ_3 and Ψ_4 are nontrivial), Γ_3 would be



To make Γ well typed, each G_u and G^v will need to undergo QR and leave behind a trace of type a. The leaf of type $\mathsf{a} \xrightarrow{k} 1 \rightarrow t$ will consume these traces, resulting in a subtree of type $1 \rightarrow \mathsf{t}$.

Finally, let $\Sigma = [\,] \bullet \mathsf{d} \bullet \mathsf{b} \bullet \mathsf{e}$, and let $\Gamma_4 = \Sigma^{m-n}$. In the case of the digraph in Figure 4, Γ_4 would be

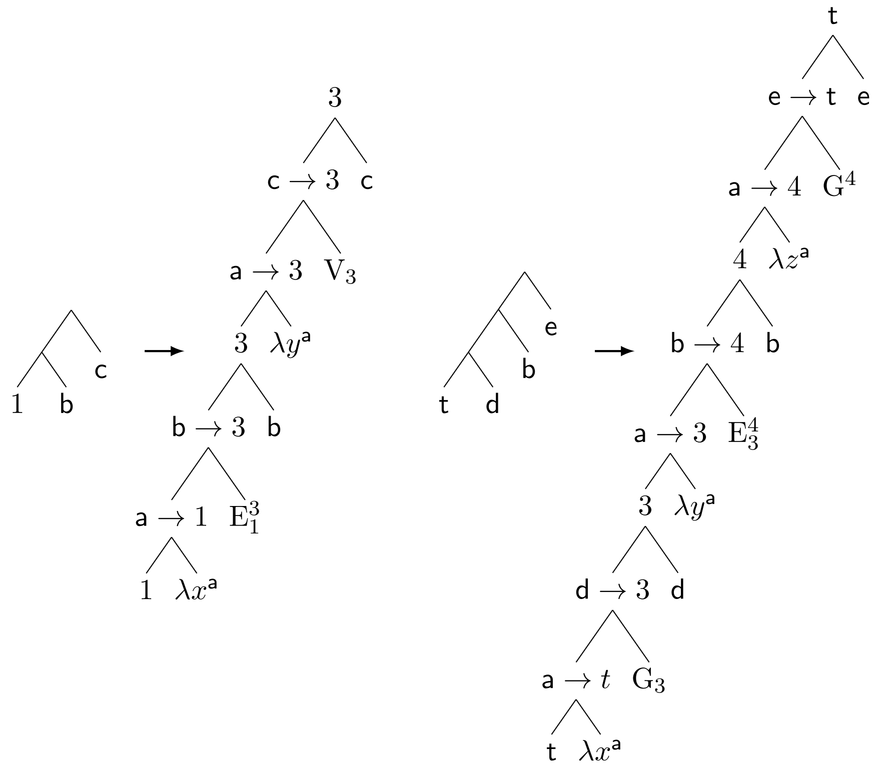
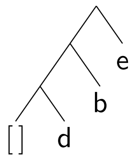


Figure 5: Simulating the traversal of an edge (left) and disposing of an unused edge (right). Interior nodes are annotated with the type of the subtree they dominate. λ-abstractions are now shown on the right, just for cosmetics (remember that LFs are *unordered* binary trees).



The context Σ is a gadget that works more or less like Δ. If G_u, E_u^v, and G^v each QR directly below the occurrences of d, b, and e respectively in $\Sigma[\mathsf{t}]$, the resulting subtree will have type t. The right side of Figure 5 illustrates the procedure with G_3, E_3^4, and G^4. Together $\Sigma[\mathsf{t}]$, G_u, and G^v work as a kind of disposal mechanism for the leaf E_u^v, providing it a place to which it can QR and returning a subtree of type t.

Γ_4 consists of $m - n$ nested copies of Σ. When G has a Hamiltonian cycle, Γ_1 will contain $m - n$ leaves E_u^v coming from edges not used in the cycle. Γ_4's purpose is to act as a waste bin of sorts for these unused edges. Notice that "disposing" of a leaf E_u^v requires raising both a leaf G_u and a leaf G^v from Γ_3. As mentioned earlier, for each vertex v there are $\deg^-(v) - 1$ edges with head v and $\deg^+(v) - 1$ edges with tail v that do not occur in the cycle. Γ_3 therefore provides exactly the right number of leaves G_u and G^v to dispose of all the unused edges.

Figure 6 shows the final LF $\Gamma = \Gamma_4[\Gamma_3[\Gamma_2[\Gamma_1]]]$. Notice that Γ will always be linear in the size of the input, and can be constructed in polynomial time.

4.3 Correctness

It remains to show that the reduction is correct, i.e., that G has a Hamiltonian cycle iff the answer to the LF REPAIR-instance constructed in the previous subsection is "yes".

First the easy direction.

Theorem 4.1. *If G has a Hamiltonian cycle, then there exists an LF Δ of type* t *such that $\Gamma \rightarrow^*_{\mathrm{QR}} \Delta$.*

Proof. Let

$$(u_1, u_2), (u_2, u_3), \ldots, (u_{n-1}, u_n), (u_n, u_1)$$

be a Hamiltonian cycle in G. Without loss of generality, assume that $u_1 = 1$. We can derive an appropriate Δ of type t from Γ as follows. Note that every QR step in the derivation below will leave a trace of type a.

First QR $\mathrm{E}_{u_1}^{u_2}$ and V_{u_2} directly below the lowest occurrence of b and c respectively in Γ_2 (compare the left side of Figure 5). Then QR $\mathrm{E}_{u_2}^{u_3}$ and V_{u_3} directly below the *second*-lowest b and c respectively in Γ_2. Continue this process for the rest of the cycle, finishing with $\mathrm{E}_{u_n}^{u_1}$ and V_{u_1}.

Exactly $m - n$ leaves of the form E_u^v in Γ_1 are still waiting to QR, each one of them corresponding to an edge not used in the Hamiltonian cycle. For each vertex v, exactly $\deg^-(v) - 1$ of these leaves correspond to an edge with head v and $\deg^+(v) - 1$ to an edge with tail v.

For each b in Γ_4, QR one of the $m - n$ remaining leaves E_u^v's directly below it. Then QR an instance of G_u and G^v from Γ_3 directly below the adjacent occurrences of d and e, respectively (compare the right side of Figure 5). Since the number of leaves G_u and G^v is $\deg^+(v) - 1$ and $\deg^-(u) - 1$ respectively, by the end of this process every G_u and G^v from Γ_3 will have undergone QR.

The LF resulting from these $3m - n$ applications of QR is the Δ we wanted. An easy check shows that it has type t. □

Now for the harder direction, which comes down to showing that the procedure in the previous proof is the only way to make Γ well typed.

Theorem 4.2. *Suppose that $\Gamma \rightarrow^*_{\mathrm{QR}} \Delta$ for some Δ of type* t. *Then G has a Hamiltonian cycle.*

Proof. By Lemma 2.1, we can assume that Δ is normalized. We now make a series of claims about Δ. Below, I refer to a leaf that occurs in Γ as a "Γ-leaf."

(i) Every trace in Δ is of order one and therefore of atomic type. The claim follows from Lemma 2.2 and the fact that all of Γ's leaves have order at most three. A consequence is that nothing in the derivation $\Gamma \rightarrow^*_{\mathrm{QR}} \Delta$, undergoes QR more than once, since this would leave a trace of atomic type next to an abstraction, contradicting the fact that Δ is normalized.

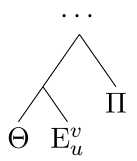
(ii) Every internal node of Δ has a type of order at most two. To prove the claim, suppose it were false and consider a maximally deep internal node of order greater than two. By (i), all abstractions are over variables of order one, so the node is not an abstraction (otherwise there would be a deeper internal node of the same order). It therefore has the form $\Theta \bullet \Lambda$, where Θ has type $\alpha\beta$, Λ has type α, and $\mathrm{ord}(\beta) > 2$. Since $\mathrm{ord}(\tau(\Theta)) = \mathrm{ord}(\alpha\beta) > 2$, Θ cannot be an internal node, so it must be a leaf. But each leaf of Δ is either a trace of order one or a Γ-leaf, and by inspection no Γ-leaf has the form $\alpha\beta$ where $\mathrm{ord}(\beta) > 2$. So no such internal node can exist.

(iii) Δ contains no "double abstractions," i.e., it is impossible to write $\Delta = \Theta[\lambda x\,(\lambda y\,\Lambda)]$. Suppose otherwise, and consider a "double abstraction" of *minimal* depth, so that we can write Δ as $\Theta[\Pi \bullet \lambda x^\alpha\,(\lambda y^\beta\,\Lambda)]$. Because Δ is normalized, Π must have a type of the form $(\alpha\beta\gamma)\delta$. This means that Π's type has order at least three, and so by (ii) Π must be a leaf. But no leaf of Δ has the form $(\alpha\beta\gamma)\delta$, a contradiction.

(iv) If an abstraction $\lambda x^\alpha\,\Pi$ occurs in Δ, then $\Delta = \Theta[l \bullet \lambda x^\alpha\,\Pi]$ for some context Θ and leaf l of type E_u^v, V_v, G_v, or G^v. This follows from (i), (ii), (iii), and the fact that Δ is normalized. A consequence is that in a derivation $\Gamma \rightarrow^*_{\mathrm{QR}} \Delta$, the only elements that undergo QR are leaves of type E_u^v, V_v, G_v, or G^v.

(v) Every trace in Δ has type a. If x^α is a trace in Δ, then it must be bound by some abstraction, so by (iv) we can write Δ as $\Theta[l \bullet \lambda x^\alpha\,\Pi[x^\alpha]]$, where l is a Γ-leaf. Since Δ is normalized, $l = (\alpha\beta)\gamma$ for some types β, γ. Inspecting the leaves of Γ shows that α can only be a.

(vi) Call the sister of the mother of a node in Δ its *aunt*. Then the aunt of each leaf E_u^v, V_v, G_v, and G^v in Δ is a Γ-leaf of type b, c, d or e, respectively. Consider, for example, a leaf E_u^v occurring in Δ with sister Θ and aunt Π, as shown below:



Θ would need order at least 4 to take

$$\mathrm{E}_u^v := (\mathsf{a} \rightarrow u) \rightarrow \mathsf{b} \rightarrow v$$

as an argument, but no node in Δ has order higher than 3. So Θ must have type $\mathsf{a} \rightarrow u$ and $\Theta \bullet \mathrm{E}_u^v$ must have type $\mathsf{b} \rightarrow v$. For Π to take $\mathsf{b} \rightarrow v$ as an argument, it would need to have order at least 3. The only such elements of Δ are the Γ-leaves, but no Γ-leaf has a type of the form $(\mathsf{b} \rightarrow v) \rightarrow \gamma$. So Π must have type b. Since all the traces in Δ have type a, the only nodes of type b in Δ are the m Γ-leaves coming from Γ_2 and Γ_4, and therefore Π is a Γ-leaf of type b, as we wanted to show. Similar arguments apply to the leaves of type V_v, G_v, and G^v. Notice that none of the leaves E_u^v, V_v, G_v, and G^v has an aunt of type b, c, d, or e in their base positions in Γ, so they must have all undergone QR in the derivation $\Gamma \rightarrow^*_{\text{QR}} \Delta$.

The upshot of all these observations is that in the derivation $\Gamma \rightarrow^*_{\text{QR}} \Delta$, every leaf of type E^v_u, V_v, G_v, and G^v undergoes QR *exactly once* (by (i) and (vi)); lands in a position directly below a unique leaf labeled b, c, d, or e respectively (by (vi)); leaves a trace of type a (by (v)); and *nothing else* undergoes QR (by (iv)).

We can now extract a Hamiltonian cycle from Δ. Exactly $2n$ leaves E^v_u and V_v in Δ underwent QR into Γ_2. Order them by decreasing depth:

$$\mathrm{E}^{v_1}_{u_1}, \mathrm{V}_{w_1}, \mathrm{E}^{v_2}_{u_2}, \mathrm{V}_{w_2}, \ldots, \mathrm{E}^{v_n}_{u_n}, \mathrm{V}_{w_n}.$$

In order for Δ to be well-typed, it must be the case that $u_1 = v_n = w_n = 1$ and

$$v_i = w_i = u_{i+1}, \quad 1 \leq i < n.$$

So we can rewrite the sequence above as

$$\mathrm{E}^{u_2}_{u_1}, \mathrm{V}_{u_2}, \mathrm{E}^{u_3}_{u_2}, \mathrm{V}_{u_3}, \ldots, \mathrm{E}^{u_1}_{u_n}, \mathrm{V}_{u_1}.$$

Since $V_{u_1}, \ldots, \mathrm{V}_{u_n}$ are pairwise distinct, so are $u_1, \ldots, u_n$, and

$$(u_1, u_2), (u_2, u_3), \ldots, (u_n, u_1),$$

is therefore a Hamiltonian cycle in G. □

Putting Theorems 3.4, 4.1, and 4.2 together proves the main claim of this paper:

Theorem 4.3. LF REPAIR *is NP-complete.*

Section 3 mentioned that LF REPAIR has the same time complexity as provability in QRT. So a corollary of the above is:

Corollary 4.1. QRT PROVABILITY *is NP-complete.*

5 Conclusion

The main result of this paper is that finding well-typed logical forms using quantifier raising is NP-complete, and therefore belongs to a well-known class of intractable problems. Assuming P $\neq$ NP, this has an immediate practical consequence: there is no efficient program that takes in an input tree and computes the different scopal readings licensed by QR.

In this paper I've chosen, in part for mathematical convenience, to work with a particularly powerful version of QR. I've allowed it to raise almost any subtree to any position and to leave a trace of any type, so long as the final LF is well typed. A reader might wonder: does the intractability result depend on this choice? Not really. Even if one worked with a vastly weaker version of QR, the problem of repairing LFs would remain intractable. Just to give an example, notice that the reduction in Section 4

(i) did not need higher-order traces (in fact, every trace had type a),

(ii) did not need parasitic scope (Barker, 2007),

(iii) did not need any elements to undergo QR multiple times.

So even if one changed the definition of QR in Section 2 to always leave traces of a single atomic type, or to ban parasitic scope, or to disallow elements from undergoing QR multiple times, and even if one imposed all of these constraints simultaneously, the reduction in the previous section would still be valid and the problem of finding a well-typed LF would remain NP-complete. There may be good empirical arguments for or against (i)–(iii), but as far as computational tractability is concerned, none of them are consequential.

The reduction does help itself to an unbounded inventory of atomic types, but only a finite number are actually needed. Given the atomic types $\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}, \mathsf{t}$ and an additional type v, it's possible to encode the types $1, \ldots, n$ in "unary": replace 1 with v, 2 with $\mathsf{v} \rightarrow \mathsf{v}$, 3 with $\mathsf{v} \rightarrow \mathsf{v} \rightarrow \mathsf{v}$, etc. This will preserve the reduction in the previous section. So even if we only considered LFs over a small, fixed set of atomic types, repairing them with QR would remain NP-complete.

Despite being generally intractable, the instances of LF REPAIR that semanticists solve in practice tend to be easy. What could explain this discrepancy? Semanticists have most likely been working with a class of LFs subject to implicit constraints that vastly reduce the complexity of the problem. For example, the higher-order meanings that semanticists actually handle are extremely uniform in that almost all of them have a type ending in t.[5] The higher-order types appearing in the LFs from Section 4 by contrast were extremely *non*-uniform. This variety is what made it possible to construct a computationally difficult "domino game" where input and output types link together in an intricate arrangement. If this is on the right track, then the true source of complexity, the true reason why LF REPAIR is difficult, is just the enormous number

[5] This is a strong tendency, not a rule. A simple counterexample are choice functions, which have type $(et)e$.

of derivational possibilities that arise when higher-order types are variegated enough that they can interact in nontrivial ways.

A natural question for further research is whether a reduction similar to the one used in this paper could prove NP-completeness results for other scope-taking formalisms. The type-logical grammar NL_λ, for example, is a very close relative of the grammar QRT from Section 3, and should have a similar time complexity (Barker, 2019; Moot, 2020). Another good candidate are continuized CCGs (Barker and Shan, 2014; White et al., 2017).

Acknowledgments

I'm thankful to Dylan Bumford, Tim Hunter, and three anonymous reviewers for some tremendously helpful suggestions.

References

Sanjeev Arora and Boaz Barak. 2009. *Computational Complexity: A Modern Approach*. Cambridge University Press, Cambridge, UK.

Chris Barker. 2007. Parasitic Scope. *Linguistics and Philosophy*, 30:407–444.

Chris Barker. 2019. NL_λ as the Logic of Scope and Movement. *Journal of Logic, Language and Information*, 28:217–237.

Chris Barker. 2020. The Logic of Quantifier Raising. *Semantics and Pragmatics*, 30:1–40.

Chris Barker and Chung-chieh Shan. 2014. *Continuations and Natural Language*. Oxford University Press.

G. Edward Barton, Jr., Robert C. Berwick, and Eric Sven Ristad. 1987. *Computational Complexity and Natural Language*. MIT Press, Cambridge, Massachusetts.

Rajesh Bhatt and Veneeta Dayal. 2007. Rightward Scrambling as Rightward Remnant Movement. *Linguistic Inquiry*, 38(2):287–301.

Robin Cooper. 1983. *Quantification and Syntactic Theory*. D. Reidel, Dordrecht.

Danny Fox. 2000. *Economy and Semantic Interpretation*. Linguistic Inquiry Monographs 35. MIT Press, Cambridge, MA.

Michael R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP Completeness*. W. H. Freeman and Company, New York.

Irene Heim. 1997. Predicates or Formulas? Evidence from Ellipsis. In *Proceedings of SALT 7*, pages 197–221.

Irene Heim and Angelika Kratzer. 1998. *Semantics in Generative Grammar*. Blackwell Publishers Ltd, Oxford, UK.

Herman Hendriks. 1993. *Studied Flexibility*. PhD thesis, Universiteit van Amsterdam.

C.-T. James Huang. 1993. Reconstruction and the Structure of VP: Some Theoretical Consequences. *Linguistic Inquiry*, 24(1):103–138.

Oleg Kiselyov and Chung-chieh Shan. 2014. Continuation Hierarchy and Quantifier Scope. In *Formal Approaches to Semantics and Pragmatics: Japanese and Beyond*, pages 105–134. Springer Netherlands, Dordrecht.

Thomas Krantz and Virgile Mobile. 2001. Encoding Hamiltonian Circuits into Multiplicative Linear Logic. *Theoretical Computer Science*, 266:987–996.

Robert May. 1977. *The Grammar of Quantification*. PhD thesis, Massachusetts Institute of Technology.

Richard Montague. 1973. The Proper Treatment of Quantification in Ordinary English. In *Approaches to Natural Language: Proceedings of the 1970 Stanford Workshop on Grammar and Semantics*, pages 221–242. Springer.

Richard Moot. 2020. Proof-theoretic Aspects of NL_λ. *Preprint*, arXiv:2010.12223.

Christos H. Papadimitriou. 1994. *Computational Complexity*. Addison-Wesley, Reading, MA.

Ethan Poole. 2024. (Im)possible Traces. *Linguistic Inquiry*, 55(2):287–326.

Ian Pratt-Hartmann. 2010. Computational Complexity in Natural Language. In *The Handbook of Computational Linguistics and Natural Language Processing*, pages 43–73. Wiley-Blackwell.

Michael White, Simon Charlow, Jordan Needle, and Dylan Bumford. 2017. Parsing with Dynamic Continuized CCG. In *Proceedings of the 13th International Workshop on Tree Adjoining Grammars and Related Formalisms*, pages 71–83. Association for Computational Linguistics.

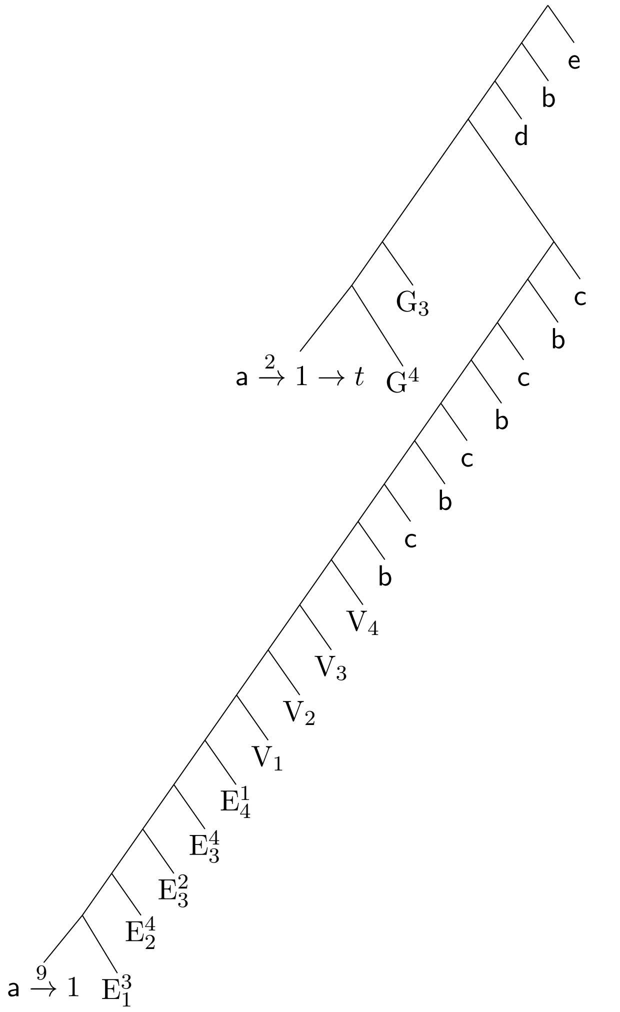


Figure 6: The LF REPAIR-instance for the digraph in Figure 4.

Dependency Structures Representation: Meaning Text Theory's Deep-Syntax Encoding With Abstract Categorial Grammars

Marie Cousin
Université de Lorraine, CNRS, Inria, LORIA / F-54000 Nancy, France
marie.cousin@loria.fr

Abstract

We present an encoding of the meaning-text theory into abstract categorial grammars, a grammatical formalism based on λ-calculus. It adresses some shortcomings of a previous encoding (Cousin, 2025). This encoding shows some limitations, especially the articulation of dependencies within structures and modifiers behavior (predicative role of adjectives and adverbs at semantic and syntactic levels, number of modifiers, etc.). Reusing Cousin's (2025) grammars composition, we present a representation of syntactic dependency structures based on de Groote (2023) that overcomes these limitations, with a focus on deep-syntax.

1 Introduction

The work presented in this article takes place in a context of encoding the meaning-text theory (MTT) into abstract categorial grammars (ACG). It extends previous work presented in (Cousin, 2023b,a, 2025). While using the same ACG architecture, we provide here a new encoding of dependency structures based on (de Groote, 2022, 2023) that concerns MTT's deep-syntactic and surface-syntactic representations. This work falls within a text generation context with formal methods. Similar motivations are to be found in Grammatical Framework (Ranta, 2004). This work's aims are to have strong control over generated text in order to be sure that it indeed conveys the message to be expressed, and to implement MTT's linguistic structures through which this control operates. In order to do so, we rely on a formal model, ACGs.

1.1 MTT and ACGs

MTT (Mel'čuk et al., 2012, 2013, 2015, see section 2) is a linguistic theory describing the link between the meaning and the text of an utterance. MTT can be used for generation as well as for analysis purposes. It is composed of seven representation levels (including semantic, deep-syntactic, and surface-syntactic levels) and six transition modules between these levels. Each representation is composed of at least both predicative and communicative structures. MTT heavily relies on the key concepts of lexical functions (LF) and paraphrase. Systems presented in (Wanner et al., 2010) and (Lareau et al., 2018) are two examples of MTT applied to text generation. These systems use graph transducers to perform generation, while we use ACGs which have the advantage of being reversible.

ACGs (de Groote, 2001, see section 3) are a grammatical framework based on λ-calculus, enabling the representation of other grammatical formalisms. For instance, Pogodalla (2017) shows an implementation of TAGs into ACGs. ACGs can also be used in generation and analysis (Kanazawa, 2007), since they are reversible. Their reversibility is a property we take avantage of in this work.

In this article, we focus on an encoding of MTT into ACGs.

1.2 Related Work

1.2.1 MTT Into ACG Encoding

(Cousin, 2023b,a) shows such an encoding of MTT into ACGs, that only takes predicative structures into account and completely ignores communicative structures. This encoding enables several linguistic phenomena's handling, such as collocations or paraphrase generation. Such phenomena are enabled thanks to the encoding of LFs among others. This article reworks Cousin's (2023a) ACG architecture and composition.

(Cousin, 2025) reuses this implementation to take the communicative structure into account. Theme-rheme opposition (Mel'cuk, 2001; Polguère, 1990), included in communicative structures, plays a crucial role in the determination of the deep-syntactic tree corresponding to the semantic graph of an utterance. Indeed, depending on the communicative structure, for two semantic representations sharing the same predicative structure, the obtained expressions differ. It is the theme-rheme opposition, that, for instance, makes a copula appear (or not) in deep-syntax when an adjective modify a noun, and therefore decides if the adjective will be expressed as an attributive (“*[the* ***purple*** *dragon]*$_{Rheme}$") or as a subject complement (“*[the dragon]*$_{Theme}$ *[is* ***purple****]*$_{Rheme}$"). Two expressions obtained from the same semantic predicative structure are different if they have different communicative structures. In this case, the two expressions are usually not paraphrases.

Since the communicative structure plays an important role in MTT’s generation process, we keep Cousin’s (2025) addition of this structure. However, (Cousin, 2023b,a, 2025) show some limitations, like the inability to use multiple modifiers on one same predicate for instance. These limitations are detailed below.

1.2.2 ACGs and Dependency Strutures

de Groote (2022, 2023) presents an ACG encoding of a semantic-syntax interface to derive formal logical semantic representations *à la* Montague. The syntactic part of de Groote’s interface uses dependency structures. We use here a similar approach to add non-mandatory dependencies to syntactic structures, such as adverbial and adjectival ones.

However, several aspects show differences:

- de Groote (2023) uses higher-order ACGs: their parsing complexity is not polynomial, and possibly only semi-decidable (de Groote, 2015). We want to use ACGs from a specific class, ACGs of order 2, to have polynomial complexity;

- de Groote’s (2023) encoding is made toward analysis (syntax to semantics), we are working toward generation (semantic to syntax);

- de Groote (2023) defines the coherence principle: regardless of the computation order of dependents (given one common governor), computed structures are logically equivalent and yield the same semantic interpretation. The coherence principle is not respected here. Indeed, respecting de Groote’s coherence principle would mean computationally intractable or even undecidable ACGs. Since we focus on the opposite direction to him (i.e., generation), and heavily rely on ACG parsing, it’s something we choose to avoid.

1.3 Contributions

This article present a new version of MTT into ACG encoding, with a focus on deep-syntax realization. It reuses original aspects of MTT, such as paraphrase and lexical functions (that enable the representation and handling of idiomatic expressions for instance). We are interested in the way linguistic structures, especially MTT’s ones, can be expressed in ACGs. The ACG encoding shows some advantages. First, it can be used in generation as well as analysis, since ACGs are reversible (see section 3). Second, some similarities with other formalisms can be found, which enable a comparison to them, and to reuse some aspects of these formalisms in our encoding. It is for example the case of modification presented in section 5 and inspired by Pogodalla’s (2017) TAG into ACG encoding.

Our contributions are the following:

- syntactic dependency structures allowing the use of multiple modifiers (for both deep-syntax and surface-syntax; this article focuses on deep-syntax), less general than de Groote’s (2023) ones, but which parsing is polynomial;

- decidable ACGs (as opposed to Cousin (2025) who had higher order ACGs);

- an encoding that naturally falls within Cousin’s (2023a) ACG architecture, with a better encoding of predicative structures having multiple dependents expressed by modifiers (adjectives and adverbs).

Sections 2 and 3 present MTT and ACGs. Sections 4 and 5 detail the used ACG architecture and the deep-syntax encoding. Section 6 presents obtained results before the conclusion in section 7.

2 Meaning-Text Theory (MTT)

MTT (Mel’čuk et al., 2012, 2013, 2015) is a linguistic theory aiming at representing the link between the meaning and the textual representation of an utterance. The *meaning* is “a linguistic content to be communicated" (Milićević, 2006) and the *text* is “any fragment of speech" (Milićević, 2006). In order to do so, MTT uses a meaning-text model (MTM, see. figure 1) composed of seven representation levels and six transition modules. MTT uses key concepts of lexical functions (LFs, Mel’Čuk and Polguère, 2021) and paraphrase (Iordanskaja et al., 1991; Milićević, 2007). These concepts are not detailed here.

2.1 MTT Architecture

A MTM (see. figure 1) is composed of seven representation levels, each one corresponding to an eponym utterance representation, namely the semantic (SemR), deep-syntactic (DSyntR), surface-syntactic (SSyntR), deep-morphologic and surface-morphologic (resp. DMorphR and SMorphR), deep-phonologic and surface-phonologic (resp. DPhonR and SPhonR) representations.

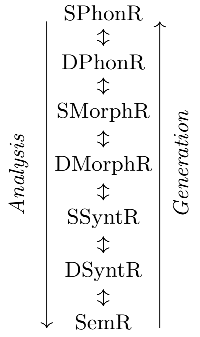


Figure 1: Schema of a MTM (Mel'čuk et al., 2012)

Between each pair of adjacent levels operates a transition module, named after the level (of the pair) nearest to the semantic level. For instance, the semantic transition module operates the transition between SemR and DSyntR.

Each representation level is made of several substructures, among which a predicative structure (graphs for SemR, trees for DSyntR and SSyntR, strings for the four remaining ones) and a communicative structure. The latter bears (among other communicative oppositions) the theme-rheme opposition (Polguère, 1990), that decorate and complement the predicative structure (see figure 3).

A transition module carry out all necessary operations between two levels to transform the previous representation into the next one. It handles structure changes, such as the semantic module that transforms semantic graphs into deep-syntactic trees, but can also perform other operations such as paraphrase steps (see section 2.3).

2.2 DSyntR

DSyntR is made of four substructures (see figure 2), namely: the deep-syntactic structure (DSyntS), the deep-syntactic communicative structure (DSynt-CommS), the deep-syntactic prosodic structure (DSynt-ProsS) and the deep-syntactic anaphoric structure (DSynt-AnaphS) (Mel'čuk et al., 2013). In this article, we only consider the first two ones, DSyntS and DSynt-CommS.

DSyntR =
⟨ DSyntS, DSynt-CommS, DSynt-ProsS, DSynt-AnaphS ⟩

Figure 2: Composition of a DSyntR

DSyntS are dependency tree structures, which nodes are deep-syntactic lexemes and edges are labeled with deep-syntactic relations (see figure 3). Several examples of DSyntS are given in section 5. Deep-syntactic relations are mainly actantial (*actant* **I**, *actant* **II**, etc.), attributive (**ATTR**), or coordinative (**COORD**) relations. Its lexemes are deep lexemes (or "rough" lexemes): collocations and idiomatic expressions are represented by one single lexeme (and not by as many lexemes as there are words composing the expression, like in SSyntS), and some lexical categories, like prepositions and determinants, are not represented (prepositions appear in SSyntS and determinants in the morphologic levels).

DSynt-CommS is represented by markers that decorate the DSyntS with communicative oppositions. For instance, theme and rheme markers are represented as boxes that encapsulates subtrees (see figure 3). The subtree in the theme-box (respectively rheme-box) is labeled as theme (resp. rheme) and therefore represents the theme (resp. rheme).

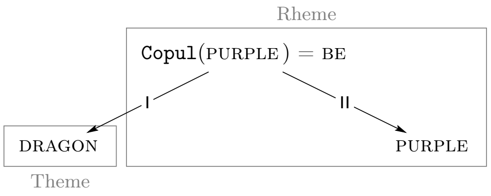


Figure 3: DSyntS and DSynt-CommS representing "*The dragon is purple.*"

2.3 Semantic Transition Module

The semantic module performs the transition between SemR and DSyntR. SemS are graphs which nodes are semantemes and which edges are semantic relations (labelled *1*, *2*, etc. depending on the semantic actant (first, second, etc.) the edge is pointing at). Hence, the semantic module performs structural operations to transform a SemS into a DSyntS. While doing so, some LF might appear, such as the ones expressing support verbs (like $\mathtt{Oper}_i$), that are semantically empty and therefore no semanteme represent them in SemS, or the one indicating a copula (`Copul`) that is also missing from SemSs. Figure 3 shows an example of DSyntS bearing the `Copul` LF.

On top of such structural operations, the semantic module also performs paraphrase steps, such as semantic paraphrasing and deep-syntactic paraphrasing. Semantic paraphrasing can bee seen as unfolding the semantic graph with semanteme definitions. Deep-syntactic paraphrasing relies heavily on LFs. Both paraphrase steps are described in (Milićević, 2007).

3 Abstract Categorial Grammars (ACGs)

ACGs (de Groote, 2001, whose definitions are used here) are a grammatical framework based on λ-calculus and used to represent other grammatical formalisms. An ACG is composed of two vocabularies (abstract and object vocabularies) that are linked by a lexicon. The abstract language, build upon the abstract vocabulary, corresponds to the set of all abstract grammatical structures, such as analysis trees. The object language, build upon the object vocabulary, corresponds to the set of surface realizations of abstract language structures, such as strings. Section 3.1 gives necessary definitions to define ACGs in section 3.2. Section 3.3 present some ACGs operations and properties.

3.1 Types, Signatures, and Lexicons

Definition 1 *Be A a set of atomic types. $\mathcal{T}(A)$ is the set of* ***linear implicative types****, obtained inductively over A:*

- *if $a \in A$ then $a \in \mathcal{T}(A)$*
- *if $\alpha, \beta \in \mathcal{T}(A)$ then $(\alpha \rightarrow \beta) \in \mathcal{T}(A)$*

Definition 2 *A* ***higher order signature****. Σ is a tuple $\Sigma = \langle\, A,\ C,\ \tau\, \rangle$, where:*

- *A is a set of atomic types,*
- *C is a set of constants,*
- *$\tau : C \longrightarrow \mathcal{T}(A)$ is a function associating a type to constants.*

$\vdash_{\Sigma_1} t : s$ expresses that the type of a λ-term t is s in the signature Σ (or $t : s$ if there is no ambiguity on the used signature). $\Lambda(\Sigma)$ denotes the set of λ-terms obtained using constants of C, variables, abstractions and applications.

For instance, $dragon^{dt} : T$ means that the constant $dragon^{dt}$ has type T and $invite^{ds}$: $\text{MOD} \rightarrow G' \rightarrow G \rightarrow G$ means that the constant $invite^{ds}$ has type[1] $\text{MOD} \rightarrow G' \rightarrow G \rightarrow G$, so that it expects a first argument of type MOD (a modifier), a second argument of type G' (an optional argument) and a third argument of type G (a mandatory argument) to obtain a term of type G (a graph). $\Sigma_{\textit{deep-syntactic-0tr}}$ and $\Sigma_{\textit{dsynt-tree}}$, illustrated in figure 5, are higher-order signatures which extracts are given in figures 8a (page 8) and 6 (page 6).

Definition 3 *Let Σ_1 and Σ_2 be two signatures. A* ***lexicon*** *$\mathcal{L}_{12}$ from Σ_1 to Σ_2 is a pair of morphisms $\langle F, G\rangle$ such that $F : \tau(A_1) \longrightarrow \tau(A_2)$ and $G : \Lambda(\Sigma_1) \longrightarrow \Lambda(\Sigma_2)$.*

We write $\mathcal{L}_{12}(t) = \gamma$ to express that γ is the interpretation of t by $\mathcal{L}_{12}$ (or $t := \gamma$ if there is no ambiguity on the used lexicon), regardless of whether the used morphism is F and both t and γ are types or the used morphism is G and they are terms.

Then, $\mathcal{L}_{\textit{dsyntRel}}(dragon^{ds}) = \lambda\, A.\ A\ dragon^{dt}$ means that the constant $dragon^{ds}$ (from $\Sigma_{\textit{deep-syntactic-0tr}}$) is interpreted by $\mathcal{L}_{\textit{dsyntRel}}$ in $\Sigma_{\textit{dsynt-tree}}$ as the term $\lambda\, A.\ A\ dragon^{dt}$. Similarly, $\mathcal{L}_{\textit{dsyntRel}}(invite^{ds}_{rtr}) = \lambda\ A\ X\ Y.\ A\ (\lambda x.\ A_1(A_2\ invite^{dt}\ Y)\ x)\ X$ means that the constant $invite^{ds}_{rtr}$ of $\Sigma_{\textit{deep-syntactic-0tr}}$ is interpreted by $\mathcal{L}_{\textit{dsyntRel}}$ in $\Sigma_{\textit{dsynt-tree}}$ as the term $\lambda\ A\ X\ Y.\ A\ (\lambda x.\ A_1\ (A_2 invite^{dt}\, Y)\ x)\ X$. In figure 5, $\mathcal{L}_{\textit{dsyntRel}}$ is the lexicon from $\Sigma_{\textit{deep-syntactic-0tr}}$ to $\Sigma_{\textit{dsynt-tree}}$. Its extract is given in figure 9 (page 9).

3.2 ACG

We may now define notions of ACG, abstract language and object language:

Definition 4 *An* ***abstract categorial grammar*** *is a tuple $\mathcal{G} = \langle \Sigma_1, \Sigma_2, \mathcal{L}_{12}, s\rangle$ where:*

- *$\Sigma_1 = \langle A_1, C_1, \tau_1\rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2\rangle$ are two higher order signatures,*
- *$\mathcal{L}_{12} = \Sigma_1 \longrightarrow \Sigma_2$ is the lexicon,*
- *$s \in \mathcal{T}(A_1)$ is the distinguished type of the grammar.*

Definition 5 *The* ***abstract language*** *$\mathcal{A}$ and the* ***object language*** *$\mathcal{O}$ of an ACG $\mathcal{G} = \langle \Sigma_1, \Sigma_2, \mathcal{L}_{12}, s\rangle$ are:*

- *$\mathcal{A} = \{t \in \Lambda(\Sigma_1) |\vdash_{\Sigma_1} t : s$ is derivable$\}$*
- *$\mathcal{O} = \{t \in \Lambda(\Sigma_2) | \ \exists u \in \mathcal{A}(\mathcal{G})$ such that $t = \mathcal{L}_{12}(u)\}$*

This article uses $\beta\eta$-equivalence as equality between λ-terms.

[1] More detail about these types is given in section 5.

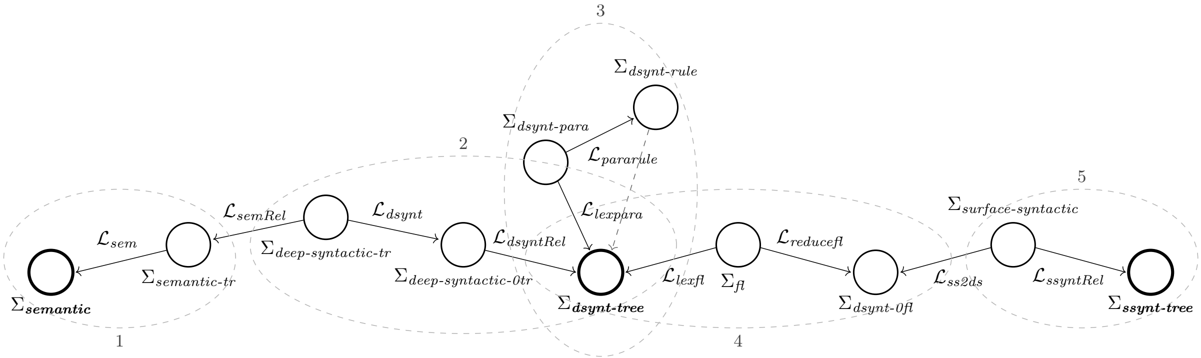

3
$\Sigma_{dsynt\text{-}rule}$
$\Sigma_{dsynt\text{-}para}$
$\mathcal{L}_{pararule}$
2
5
$\mathcal{L}_{semRel}$
$\mathcal{L}_{dsynt}$
$\mathcal{L}_{lexpara}$
$\Sigma_{surface\text{-}syntactic}$
$\mathcal{L}_{sem}$
$\mathcal{L}_{dsyntRel}$
$\mathcal{L}_{reducefl}$
$\Sigma_{deep\text{-}syntactic\text{-}tr}$
$\Sigma_{semantic\text{-}tr}$
$\Sigma_{deep\text{-}syntactic\text{-}0tr}$
$\mathcal{L}_{lexfl}$
Σ_{fl}
$\mathcal{L}_{ss2ds}$
$\mathcal{L}_{ssyntRel}$
$\Sigma_{\textbf{semantic}}$
$\Sigma_{\textbf{dsynt-tree}}$
$\Sigma_{dsynt\text{-}0fl}$
$\Sigma_{\textbf{ssynt-tree}}$
1
4

Figure 4: ACG composition of MTT encoding

Both abstract and object languages of an ACG are sets of λ-terms obtained by induction over eponym signatures of the ACG. For example, the set of deep-syntactic trees corresponds to the object language of the ACG (illustrated in figure 5) made of $\Sigma_{deep\text{-}syntactic\text{-}0tr}$ (its abstract signature), $\Sigma_{dsynt\text{-}tree}$ (its object signature), and $\mathcal{L}_{dsyntRel}$.

3.3 Operations and Properties

ACGs allow the use of three main operations, as illustrated in figure 5. Application is the operation of applying the lexicon of an ACG from its abstract signature to its object signature ($\mathcal{L}_{dsynt}(\gamma^{dst}) = \gamma^{ds}$). Lexicons are reversible, and the operation of reversing the lexicon from the object signature of an ACG to its abstract signature is called parsing ($\mathcal{L}^{-1}_{semRel}(\gamma^{str}) = \{\gamma^{dst}\}$). Both operations (application and parsing) can be composed to form a transduction operation ($\mathcal{L}_{dsynt}(\mathcal{L}^{-1}_{semRel}(\gamma^{str})) = \{\gamma^{ds}\}$). Two transduction examples are shown in figure 5.

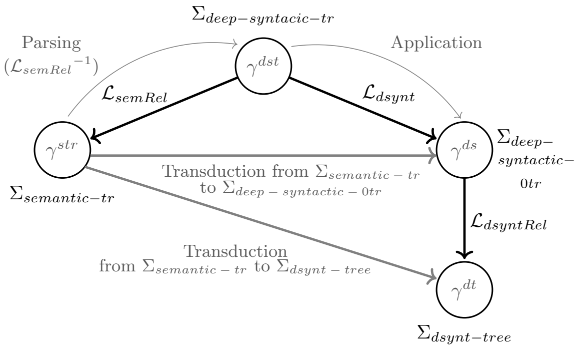

$\Sigma_{deep-syntacic-tr}$
Parsing
$(\mathcal{L}_{semRel}{}^{-1})$
Application
γ^{dst}
$\mathcal{L}_{semRel}$
$\mathcal{L}_{dsynt}$
γ^{str}
γ^{ds}
$\Sigma_{deep-syntactic-0tr}$
Transduction from $\Sigma_{semantic-tr}$ to $\Sigma_{deep-syntactic-0tr}$
$\Sigma_{semantic-tr}$
$\mathcal{L}_{dsyntRel}$
Transduction from $\Sigma_{semantic-tr}$ to $\Sigma_{dsynt-tree}$
γ^{dt}
$\Sigma_{dsynt-tree}$

Figure 5: Example of application, parsing and transduction operations

An ACG is characterized by its order and its complexity. The complexity of an ACG is its parsing complexity.

Definition 6 *The order of a type $\tau \in \mathcal{T}(A)$ is inductively defined:*

- $order(\tau) = 1$ *if* $\tau \in A$,
- $order(\alpha \to \beta)$ $= max(1 + order(\alpha), order(\beta))$ *otherwise.*

The order of an abstract constant is the order of its type, and the ***order of an ACG*** *is the maximum of the order of its abstract constants.*

The ***complexity*** *of an ACG is the maximum of the orders of its atomic types realizations. An ACG of order γ and of complexity η is written $ACG_{(\gamma,\eta)}$.*

We aim to use a specific class of ACGs, ACGs said of order two, since their parsing complexity is decidable and polynomial (Salvati, 2005; Kanazawa, 2017)

4 MTT Into ACG Encoding

We use the ACG architecture of (Cousin, 2025) (that uses Cousin, 2023b,a architecture while adding theme-rheme opposition to the semantic to deep-syntactic transition). However, we rewrite several ACGs to model DSyntR and SSyntR dependency trees based on de Groote's (2023) ACGs. We use the ACGtk software (Guillaume et al., 2024) to implement this model.

Figure 4 shows the architecture and composition of Cousin (2023b, 2025) that we use here. It is divided into five areas:

- Areas 1 - 2: transduction between $\Sigma_{semantic\text{-}tr}$ and $\Sigma_{dsynt\text{-}tree}$ allows transitions between SemR and DSyntR. Area 1 corresponds to semantic and area 2 corresponds to deep-syntax;
- Area 3: transduction between $\Sigma_{dsynt\text{-}tree}$ and $\Sigma_{dsynt\text{-}rule}$ performs deep-syntactic paraphrase steps;

- Area 4: transduction between $\Sigma_{\textit{dsynt-tree}}$ and $\Sigma_{\textit{dsynt-0fl}}$ realizes LFs eventually present in DSyntR;
- Area 5: transduction between $\Sigma_{\textit{dsynt-0fl}}$ and $\Sigma_{\textit{ssynt-tree}}$ transforms DSyntR (without LFs) into SSyntS. Area 5 corresponds to surface-syntax.

Our modifications concern areas 2, 4 and 5. This article only present the ones about the second area, *i.e.*, about deep-syntax.

Notations: In the following, $\Sigma_{\textit{deep-syntactic-tr}}$ is referenced as $\Sigma_{\textit{dst}}$ (deep-syntax with thematicity), $\Sigma_{\textit{deep-syntactic-0tr}}$ as $\Sigma_{\textit{ds}}$ (deep-syntax without thematicity), and $\Sigma_{\textit{dsynt-tree}}$ as $\Sigma_{\textit{dt}}$ (deep-syntactic trees). A constant representing a lexeme LEX in the signature Σ_S is written lex^S. For instance, $dragon^{dt}$ is the constant representing the lexeme DRAGON in $\Sigma_{\textit{dt}}$. A term representing an expression E in $\Lambda(\Sigma_S)$ is written γ_E^S. For instance, the term encoding the DSyntR associated to the expression (cpd): "*the calm purple dragon*" in $\Sigma_{\textit{dt}}$ is written $\gamma_{\textit{cpd}}^{dt}$.

5 Deep-Syntax Encoding

We adapt de Groote's (2023) dependency structures. This section details how it is done for deep-syntactic dependency structures. Figures 8a, 8b, 6, 8c and 9 (on the following pages) respectively show extracts of $\Sigma_{\textit{dst}}$, $\Sigma_{\textit{ds}}$, $\Sigma_{\textit{dt}}$, $\mathcal{L}_{\textit{dsynt}}$ and $\mathcal{L}_{\textit{dsyntRel}}$ that are used below.

5.1 DSyntS Encoding ($\Sigma_{\textit{dt}}$)

$\Sigma_{\textit{dt}}$ (deep-syntactic trees, see figure 6) enables DSyntS representation. DSyntS are dependency tree structures, which nodes are deep lexemes and branches are deep-syntactic relations, such as **I**, **II**, or **ATTR** for example. These three examples respectively correspond to relations expressing the first actant, the second actant, or an attributive dependent of a deep lexeme. A relation is represented as a branch in the deep-syntactic tree that starts at the governor and points toward its dependent.

Each deep lexeme is encoded in this implementation by a constant of type T (see (1)). In MTT, categories are distinguished at the deep-syntactic level. However, for simplification purposes, we chose to represent them only at the surface-syntactic level of the present encoding and not at its deep-syntactic level. Therefore, grammatically incorrect structures can be produced. However, they will be filtered out by lexicons and won't be generated at the surface syntactic level (see section 5.3). Hence, no distinction is made between grammatical categories in $\Sigma_{\textit{dt}}$.

$$dragon^{dt}, invite^{dt} : T \tag{1}$$

Since a relation starts at one lexeme and points toward another, it is represented by a constant that takes two arguments of type T (being resp. the governor and the dependent), and that has a resulting type T. Hence, each deep-syntactic relation (such as the ones in (2)) is of type $T \to T \to T$.

$$\mathbf{I}, \mathbf{ATTR} : T \to T \to T \tag{2}$$

Since the resulting type of a relation is T, a dependency (*i.e.* a governor, the relation, and its dependent) is represented by a term of type T as well. This term can in turn be used as a governor (to add another dependency to its governor) or as a dependent (to make it depend from another lexeme). Hence, type T is used for constants and terms encoding trees, sub-trees, and single nodes.

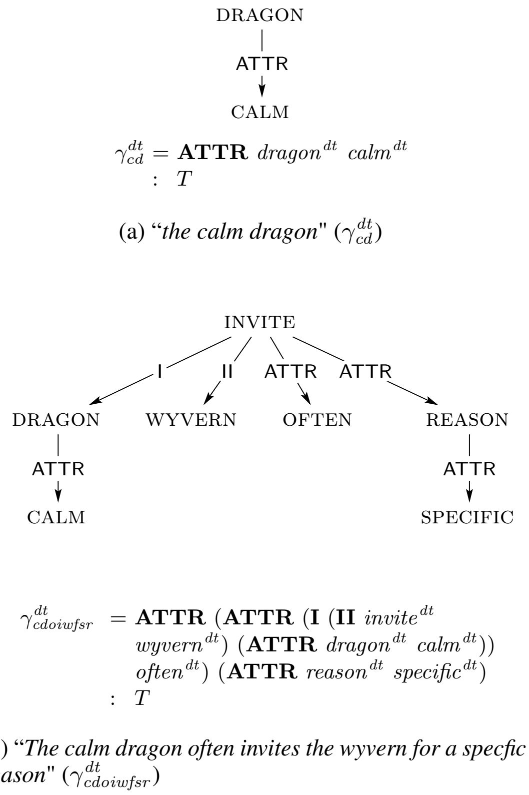
$$\begin{array}{rl}
\multicolumn{2}{c}{\Sigma_{\textit{dsynt-tree}}} \\
T & : type \\
\\
\mathbf{I}, \mathbf{II}, \mathbf{III} & : T \to T \to T \\
\mathbf{ATTR} & : T \to T \to T \\
\\
Conv_{21} & : T \to T \to T \\
\\
calm^{dt} & : T \\
invite^{dt} & : T \\
dragon^{dt} & : T \\
often^{dt} & : T \\
purple^{dt} & : T \\
reason^{dt} & : T \\
specific^{dt} & : T \\
wyvern^{dt} & : T
\end{array}$$

Figure 6: Extract of $\Sigma_{\textit{dsynt-tree}}$ (noted $\Sigma_{\textit{dt}}$, that enables the representation of deep-syntactic trees)

Figure 7 shows two examples of DSyntS along with terms of $\Lambda(\Sigma_{\textit{dt}})$ encoding them. Figure 7a pictures a rather simple DSyntS, having only one dependency, while figure 7b pictures a more complex one, with several dependencies.

5.2 DSyntR Encoding ($\Sigma_{\textit{dst}}$ and $\Sigma_{\textit{ds}}$)

As stated in the introduction (section 1), a DSyntR contains a communicative structure, DSynt-CommS. This structure is absent from $\Sigma_{\textit{dt}}$ (deep-syntactic trees) and $\Sigma_{\textit{ds}}$ (deep-syntax without thematicity), but is present in $\Sigma_{\textit{dst}}$ (deep-syntax



$$\begin{aligned}\gamma^{dt}_{cd} &= \textbf{ATTR}\ \mathit{dragon}^{dt}\ \mathit{calm}^{dt}\\ &:\ T\end{aligned}$$

(a) "*the calm dragon*" (γ^{dt}_{cd})

$$\begin{aligned}\gamma^{dt}_{cdoiwfsr} &= \textbf{ATTR}\ (\textbf{ATTR}\ (\textbf{I}\ (\textbf{II}\ \mathit{invite}^{dt}\\ &\quad \mathit{wyvern}^{dt})\ (\textbf{ATTR}\ \mathit{dragon}^{dt}\ \mathit{calm}^{dt}))\\ &\quad \mathit{often}^{dt})\ (\textbf{ATTR}\ \mathit{reason}^{dt}\ \mathit{specific}^{dt})\\ &:\ T\end{aligned}$$

(b) "*The calm dragon often invites the wyvern for a specfic reason*" ($\gamma^{dt}_{cdoiwfsr}$)

Figure 7: Representation of DSyntS associated to expressions "*the calm dragon*" and "*The calm dragon often invites the wyvern for a specific reason*" and their associated object terms (from $\Lambda(\Sigma_{dt})$).

with thematicity). Indeed, theme-rheme opposition plays a crucial role in the determination of the DSyntR that corresponds to a SemR. In order to represent this role and enable the potential corresponding choices inherent to the theme-rheme opposition of a SemR, thematic annotations are encoded in the type of constants from Σ_{str} (semantic with theme-rheme opposition) and Σ_{dst} (deep-syntax with thematicity).

In both Σ_{dst} and Σ_{ds}, lexemes are encoded with constants of type G-like. These types stand for graph, subgraph or node and follows the same use principle as type T. Variants can be found, such as G_t or G_r, that respectively express a theme or a rheme marking. If a prime symbol is used (like G' or G'_t), it denotes an optionally expressible argument[2]. Type $G_{f,r}$ is used to indicate that a predicative structure contains a thematic opposition (i.e. both theme and rheme areas) with the index $-_f$, and that its dominant node (the future tree root) is marked as rheme with the index $-_r$.

For instance, $\mathit{invite}^{dst}_{rtr}$ (which type expression is given in (3)) is a constant encoding a verb which predicative structure expects two arguments, the first one being marked as theme (type G_t), the remaining part of the predicative structure being marked as rheme (type G_r). Its first argument (of type MOD) stands for a potential modifier[3], such as an adverbial group.

$$\mathit{invite}^{dst}_{rtr}\ :\ \mathrm{MOD} \to G_t \to G_r \to G_{f,r} \tag{3}$$

Thematicity markers in the constants' types are lost between Σ_{dst} and Σ_{ds}. $\mathcal{L}_{dsynt}$ only purpose is to erase these types markings (see Figure 8c). For instance, $\mathcal{L}_{dsynt}(\mathit{invite}^{dst}_{rtr}) = \mathit{invite}^{ds}_{rtr}$ is given in (4). Therefore, expressions of terms from $\Lambda(\Sigma_{dst})$ and $\Lambda(\Sigma_{ds})$ are really similar (see figures 8a and 8b, page 8).

$$\mathit{invite}^{ds}_{rtr}\ :\ \mathrm{MOD} \to G \to G \to G_f \tag{4}$$

It is $\mathcal{L}_{dsyntRel}$ (see Figure 9) that will really build all DSyntS in $\Lambda(\Sigma_{dt})$ from $\Lambda(\Sigma_{ds})$. In other words, Σ_{dst} is used to express and control thematicity contraints and optional arguments issues (see section 5.3), while $\mathcal{L}_{dsyntRel}$ realizes DSyntS. $\mathcal{L}_{dsynt}$ and Σ_{ds} are intermediary lexicon and signature used for encoding purposes.

Σ_{dst} bears the encoding of all constraints and is therefore used in examples from the following section about these constraints. However, for reading simplicity purposes, we use in section 5.5 examples from Σ_{ds} (see figure 8b). Their pendants from Σ_{dst} do not add any information for section 5.5 purpose and can be found in figures 8a, and their interpretation by $\mathcal{L}_{dsynt}$ in figure 8c (page 8).

5.3 Constraints Encoding (Σ_{dst} and $\mathcal{L}_{dsyntRel}$)

Since all lexemes of Σ_{dt} are encoded by constants of type T, incorrect structures can be build upon $\Lambda(\Sigma_{dt})$, like the one illustrated in (5). In this example, two major issues occur. First, in a DSyntR, according to well-formedness rules of MTT, each node (or lexeme) has at most one actancial relation of each kind; dragon^{dt} can't be the governor of two **I** relations. Second, invite^{dt} is a lexeme expecting at least one dependent ("*Y is invited*"), and usually two, since its predicative structure contains two arguments: $X\ \mathit{invites}\ Y$.

$$\begin{aligned}\gamma^{dt}_{illicit} &= \textbf{I}\ (\textbf{I}\ \mathit{dragon}^{dt}\ \mathit{wyvern}^{dt})\ \mathit{invite}^{dt}\\ &:\ T\end{aligned} \tag{5}$$

[2] Optionally expressible arguments are detailed in section 5.3.

[3] Modifiers are described in section 5.5.

$$\begin{array}{rl}
\multicolumn{2}{c}{\Sigma_{\textit{deep-syntacic-tr}}} \\
G_r,\ G_t,\ G'_r & : \mathit{type} \\
\mathrm{MOD}_{rr} & : \mathit{type} \\
\mathrm{MOD}_{tr} & : \mathit{type} \\
\mathrm{MOD}'_{rr} & : \mathit{type} \\
\mathrm{MOD}j_{rr} & : \mathit{type} \\
 & \\
C_{r\to t} & : G_r \to G_t \\
I^{dst}_{\mathrm{MOD},rr} & : \mathrm{MOD}_{rr} \\
I^{dst}_{\mathrm{MOD},tr} & : \mathrm{MOD}_{tr} \\
I^{dst}_{\mathrm{MOD}',rr} & : \mathrm{MOD}'_{rr} \\
I^{dst}_{\mathrm{MOD}j,rr} & : \mathrm{MOD}j_{rr} \\
 & \\
calm^{dst}_{rr} & : \mathrm{MOD}_{rr} \to \mathrm{MOD}j_{rr} \to \mathrm{MOD}j_{rr} \\
invite^{dst}_{rtr} & : \mathrm{MOD}_{tr} \to G_t \to G_r \to G_{f,r} \\
invite^{dst}_{rrt} & : \mathrm{MOD}'_{rr} \to G'_r \to G_t \to G_{f,r} \\
dragon^{dst}_{r} & : \mathrm{MOD}j_{rr} \to G_r \\
often^{dst}_{tr} & : \mathrm{MOD}_{tr} \to \mathrm{MOD}_{tr} \\
purple^{dst}_{rr} & : \mathrm{MOD}_{rr} \to \mathrm{MOD}j_{rr} \to \mathrm{MOD}j_{rr} \\
fsr^{dst}_{tr} & : \mathrm{MOD}_{tr} \to \mathrm{MOD}_{tr} \\
wyvern^{dst}_{r} & : \mathrm{MOD}j_{rr} \to G_r
\end{array}$$

(a) Extract of $\Sigma_{\textit{deep-syntactic-tr}}$ (noted Σ_{dst})

$$\begin{array}{rl}
\multicolumn{2}{c}{\Sigma_{\textit{deep-syntacic-0tr}}} \\
G,\ G' & : \mathit{type} \\
\mathrm{MOD} & : \mathit{type} \\
\mathrm{MOD}' & : \mathit{type} \\
\mathrm{MOD}j & : \mathit{type} \\
 & \\
I^{ds}_{\mathrm{MOD}} & : \mathrm{MOD} \\
I^{ds}_{\mathrm{MOD}'} & : \mathrm{MOD}' \\
I^{dst}_{\mathrm{MOD}j} & : \mathrm{MOD}j \\
 & \\
calm^{ds} & : \mathrm{MOD} \to \mathrm{MOD}j \to \mathrm{MOD}j \\
invite^{ds}_{rtr} & : \mathrm{MOD} \to G \to G \to G \\
invite^{ds}_{rrt} & : \mathrm{MOD}' \to G' \to G \to G \\
dragon^{ds} & : \mathrm{MOD}j \to G \\
often^{ds} & : \mathrm{MOD} \to \mathrm{MOD} \\
purple^{ds} & : \mathrm{MOD} \to \mathrm{MOD}j \to \mathrm{MOD}j \\
fsr^{ds} & : \mathrm{MOD} \to \mathrm{MOD} \\
wyvern^{ds} & : \mathrm{MOD}j \to G
\end{array}$$

(b) Extract of $\Sigma_{\textit{deep-syntactic-0tr}}$ (noted Σ_{ds})

$$\begin{array}{rl}
\multicolumn{2}{c}{\mathcal{L}_{dsynt}} \\
G_r, G_t & := G \\
G'_r & := G' \\
\mathrm{MOD}_{rr}, \mathrm{MOD}_{tr} & := \mathrm{MOD} \\
\mathrm{MOD}'rr & := \mathrm{MOD}' \\
\mathrm{MOD}j_{rr} & := \mathrm{MOD}j \\
 & \\
C_{r\to t} & := \lambda p.\ p \\
I^{dst}_{\mathrm{MOD},rr}, I^{dst}_{\mathrm{MOD},tr} & := I^{ds}_{\mathrm{MOD}} \\
I^{dst}_{\mathrm{MOD}',rr} & := I^{ds}_{\mathrm{MOD}'} \\
I^{dst}_{\mathrm{MOD}j,rr} & := I^{ds}_{\mathrm{MOD}j} \\
calm^{dst}_{rr} & := calm^{ds} \\
invite^{dst}_{rtr} & := invite^{ds}_{rtr} \\
invite^{dst}_{rrt} & := invite^{ds}_{rrt} \\
dragon^{dst}_{r} & := dragon^{ds} \\
often^{dst}_{tr} & := often^{ds} \\
purple^{dst}_{rr} & := purple^{ds} \\
fsr^{dst}_{tr} & := fsr^{ds} \\
wyvern^{dst}_{r} & := wyvern^{ds}
\end{array}$$

(c) Extract of $\mathcal{L}_{dsynt}$

Figure 8: Extracts of $\Sigma_{\textit{deep-syntactic-0tr}}$ (deep-syntax without thematicity), $\Sigma_{\textit{deep-syntactic-tr}}$ (deep-syntax with thematicity) and $\mathcal{L}_{dsynt}$, lexicon from $\Sigma_{\textit{deep-syntactic-tr}}$ to $\Sigma_{\textit{deep-syntactic-0tr}}$

In order to tackle these issues, two main strategies are used. Regarding the number of dependents (and their nature), *we use abstract signatures to control obtained structures by encoding the constraints in the constants' types.* Hence, these constraints are dealt with in abstract signatures Σ_{dst} and Σ_{ds}. Figures 8a and 8b give extracts of Σ_{dst} and Σ_{ds} as examples.

Each predicate has a predicative structure that expects mandatory dependents. Among these dependents, some are necessarily expressible, and other may be omitted. Let's take the example of INVITE. When using this verb, we know that *someone*$_1$ invites another *someone*$_2$. However, sentences like "*Charlie is invited*" are correct (with "*Charlie*" being the invited *someone*$_2$). Hence INVITE has two mandatory dependents (*someone*$_1$ and *someone*$_2$), but the first one is optionally expressible. Such constraints are expressed in the type of constants of Σ_{dst}, with the prime notation indicating an optionally expressible argument. In the case of a noun like DRAGON, no dependent is expected, hence no argument is mandatory. (6) and (7) [4] give the types of constants $invite^{dst}_{rrt}$ and $dragon^{dst}_{r}$ illustrating the encoding of this constraint.

$$invite^{dst}_{rtr} : \mathrm{MOD}_{tr} \to G_t \to G_r \to G_{f,r} \tag{6}$$

$$\begin{aligned} invite^{dst}_{rrt} &: \mathrm{MOD}'_{rr} \to G'_r \to G_t \to G_{f,r} \\ dragon^{dst}_{r} &: \mathrm{MOD}j_{rr} \to G_r \end{aligned} \tag{7}$$

Nevertheless, (7) shows two constants for INVITE, $invite^{dst}_{rtr}$ and $invite^{dst}_{rrt}$. The former has its first argument marked as theme, and will produce sentences like "*[The dragon]*$_{Theme}$ *[invites the wyvern]*$_{Rheme}$", where the first dependent of INVITE is not optionally expressible (since it is the theme, hence it can not be omitted). The latter has its second argument marked as theme, and will produce sentences like "*[The wyvern]*$_{Theme}$ *[is invited by the dragon]*$_{Rheme}$", where the first semantic dependent of INVITE, DRAGON, is marked as rheme and can easily be omitted ("*[The wyvern]*$_{Theme}$ *[is invited]*$_{Rheme}$"). So, on top of constraints inherent to the predicative structure (optionally expressible arguments), we need to take the thematic opposition and its consequences into account. This example (with INVITE) is one illustration of the role of the thematic opposition. This opposition is to be found in communicative structures of MTT, that complement its predicative structures. Theme-rheme opposition is not limited to active/passive distinction[5]. Another example of its implication can be found in the introduction (subsection 1.2.1)

[4] MOD-like types (MOD_{tr}, MOD'_{rr}, etc.) are modifiers types and explained in section 5.5. Their theme-rheme markings reflect the thematic markings of their semantic actants.

[5] See (Polguère, 1990) and (Mel'cuk, 2001) for further details about MTT's communicative structure and the thematic opposition. Some similarities can be found between theme-rheme opposition and information structure (Kruijff-Korbayova and Steedman, 2003; Steedman, 2024).

$$\mathcal{L}_{\textit{dsyntRel}}$$

$$\begin{aligned}
G,\ G'_r &:= T\\
\textsc{mod}, \textsc{mod}' &:= (T \to T) \to (T \to T)\\
\textsc{mod}j &:= T \to T\\
\\
I^{ds}_{\textsc{mod}} &:= \lambda v.\ v\\
I^{ds}_{\textsc{mod}'} &:= \lambda v.\ v\\
I^{ds}_{\textsc{mod}j} &:= \lambda n.\ n\\
\\
\mathit{calm}^{ds} &:= \lambda M\ A\ P.\ (\mathbf{ATTR}\ (A\ P)\ (M\ (\lambda m.\ m)\ \mathit{calm}^{dt}))\\
\mathit{invite}^{ds}_{rtr} &:= \lambda M\ X\ Y.\ M\ (\lambda x.\ \mathbf{I}\ (\mathbf{II}\ \mathit{invite}^{dt}\ Y)\ x)\ X\\
\mathit{invite}^{ds}_{rrt} &:= \lambda M\ X\ Y.\ M\ (\lambda x.\ \mathbf{I}\ (\mathbf{II}\ (\mathit{Conv}_{21}\ \mathit{invite}^{dt})\ x)\ Y)X\\
\mathit{dragon}^{ds} &:= \lambda A.\ A\ \mathit{dragon}^{dt}\\
\mathit{often}^{ds} &:= \lambda M\ PX.\ M\ (\lambda x.\ \mathbf{ATTR}\ (P\ x)\ \mathit{often}^{dt})X\\
\mathit{purple}^{ds} &:= \lambda M\ A\ P.\ (\mathbf{ATTR}\ (A\ P)\ (M\ (\lambda m.\ m)\ \mathit{purple}^{dt}))\\
\mathit{fsr}^{ds} &:= \lambda M\ PX.\ M\ (\lambda x.\ \mathbf{ATTR}\ (P\ x)\ (\mathbf{ATTR}\ \mathit{reason}^{dt}\ \mathit{specific}^{dt})\ X\\
\mathit{wyvern}^{ds} &:= \lambda A.\ A\ \mathit{wyvern}^{dt}
\end{aligned}$$

Figure 9: Extract of $\mathcal{L}_{\textit{dsyntRel}}$, lexicon from $\Sigma_{\textit{deep-syntactic-0tr}}$ to $\Sigma_{\textit{dsynt-tree}}$

where it determine the apparition (or not) of a copula. As shown in (7), the thematic opposition is also encoded in the type of abstract constraints (see (Cousin, 2025) for further details on the thematic opposition encoding).

Regarding the well-formedness of DSyntR, it is enforced in $\mathcal{L}_{\textit{dsyntRel}}$ (see Figure 9) that associates abstract constants from Σ_{ds} to their object terms in $\Lambda(\Sigma_{dt})$. (8)[6] shows the interpretations by $\mathcal{L}_{\textit{dsyntRel}}$ of a predicate expecting two dependents, respectively an actant **I** and an actant **II** ($\mathit{invite}^{ds}_{rtr}$) and the interpretation of a noun which predicative structure does not except any dependents (dragon^{ds}).

$$\begin{aligned}
&\mathcal{L}_{dsyntRel}(\mathit{invite}^{ds}_{rtr})\\
&:= \ \lambda\, M\ X\ Y.\ M\ (\lambda\, x.\ \mathbf{I}\ (\mathbf{II}\ \mathit{invite}^{dt}\ Y)\ x)\ X\\
&\mathcal{L}_{dsyntRel}(\mathit{dragon}^{ds})\\
&:= \ \lambda\, A.\ A\ \mathit{dragon}^{dt}
\end{aligned} \tag{8}$$

The encoding of interpretations by the lexicon solves the issue of having only well-formed structures in $\Lambda(\Sigma_{dt})$. Since only correct interpretations are encoded, only correct structures will be generated. Reversely, if an incorrect structure is build upon Σ_{dt}, no antecedent will be found by parsing of $\mathcal{L}_{\textit{dsyntRel}}$, and hence no SemR can be obtained (when continuing the analysis).

We therefore have the following equalities[7]:

$$\begin{aligned}
&\mathcal{L}_{dsyntRel}(\mathcal{L}_{dsynt}(\mathit{dragon}^{dst}_{r}\ (\mathit{calm}^{dst}_{rr}\ I^{dst}_{\textsc{mod},rr}\ I^{dst}_{\textsc{mod}j})))\\
&= \mathbf{ATTR}\ \mathit{dragon}^{dt}\ \mathit{calm}^{dt}\\
&\mathcal{L}_{dsyntRel}(\mathcal{L}_{dsynt}(\mathit{invite}^{ds}_{rtr}\ I^{dst}_{\textsc{mod},rt}\ (C_{r\to t}\\
&(\mathit{dragon}^{dst}_{r}\ I^{dst}_{\textsc{mod}j,rr}))(\mathit{wyvern}^{dst}_{r}\ I^{dst}_{\textsc{mod}j,rr})))\\
&= \mathbf{I}\ (\mathbf{II}\ \mathit{invite}^{dt}\ \mathit{wyvern}^{dt})\ \mathit{dragon}^{dt}
\end{aligned} \tag{9}$$

5.4 Discussion

This encoding presents some major differences with de Groote's (2023) one, since we made choices opposed to his, as stated earlier in section 1.2.2. He encodes dependency relations in the abstract signature, while we do so in the object signature. In (de Groote, 2023), relations (or rather, constants encoding them) constrain the grammatical categories of their governor and dependent. Here, we do not have grammatical categories, at least not at the deep-syntactic level. In the encoding presented here, lexemes themselves (and not relations) constrain their dependents: they force the mandatory ones to be represented (by a lexeme or a dummy constant indicating that the dependent is, in fact, not expressed) and the obligatory expressible constant to be expressed (in this case the dummy constant of type G' can't be used). Dependency relations are encoded in the object signature, and do not constrain anything nor bear any constraints.

On top of that, de Groote (2023) respects what he calls the coherence principle: regardless of the dependencies' computation order, obtained structures are logically equivalent. This result is obtained using affine constants. However, we want linear constants and ACGs of order 2. It is a constraint we fix ourselves for parsing computation and complexity issues, and in order to encode the model in

[6] Abstract varibles A and M encode modifiers. Their behavior is explained in section 5.5.

[7] Constants $I^{dst}_{\textsc{mod}}$ and its variants (that depends on the thematic markings of their governor predicative structure) have a MOD-like type (*i.e.*, a variant of MOD) and are interpreted by $\mathcal{L}_{\textit{dsyntRel}} \circ \mathcal{L}_{\textit{dsynt}}$ as the identity.

ACGtk to automate the computation process. Thus, respecting the coherence principle seems incompatible with this constraint. Moreover, the order in which relations are computed does matter in our case. At the deep-syntactic level, relations are not ordered, so one could think the order is of no importance. However, their order does matter at the surface syntactic level, since it will be used in the linearisation process toward the morphological levels. Consequently, two structures having the same dependencies but in different orders are to yield two different expressions in MTT. Such representations have to be considered different and not logically equivalent. Hence, the dependency order does matter at the surface-syntactic level. Then, even if the order doesn't matter at the deep-syntactic level, it is also useless to care for the coherence principle at that level since it will have to be broken at the next level. Additionally, not respecting the coherence principle makes both syntactic levels have homogeneous encoding. For all these reasons, ACGs presented here do not respect the coherence principle.

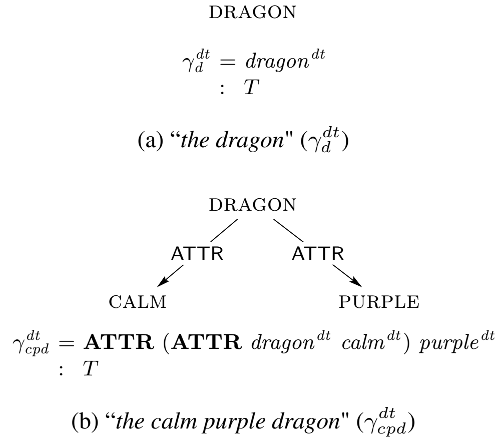
Next section describes the encoding of modifiers. One straightforward possibility to do so would have been to allow a bounded arbitrary number of optional dependents slots in the lexemes' lexical entry. This, however, would significantly complexify the deep-syntactic paraphrase part of the encoding (area 3 of figure 4), and one could always find a case where the arbitrary number of slots would be too small to represent the wanted expression. For these reasons, we chose to use a recursive solution, drawn from Pogodalla's (2017) TAG encoding, and described below.

5.5 Modifiers Encoding

Modifiers, such as adjectives and adverbial groups, have a specific behavior at the deep-syntactic level.

We may observe that they have a specific behavior in the semantic module as well. In a SemR, the modifier predicates over the entity that is modified (the modifier governs the predicate), while in a DSyntR, dependencies between a modifier and a predicate usually go from the predicate to the modifier: the predicate governs the modifier (some exceptions can be found, but that is the general case). Hence, a dependency inversion is to be found in the semantic module during the transition from SemR to DSyntR (Mel'čuk et al., 2013, p.248). We account for this, but since it is not our point here, we focus on DSyntR themselves.

In a DSyntR, modifiers are not mandatory dependents of a lexeme. A lexeme can have zero modifier (as illustrated in figure 10a), one modifier (as illustrated in figures 7a and 7b for DRAGON), or more (as illustrated for DRAGON in figure 10b and 7b for INVITE).



$$\gamma_d^{dt} = dragon^{dt} : T$$

(a) "*the dragon*" (γ_d^{dt})

$$\gamma_{cpd}^{dt} = \mathbf{ATTR}\ (\mathbf{ATTR}\ dragon^{dt}\ calm^{dt})\ purple^{dt} : T$$

(b) "*the calm purple dragon*" (γ_{cpd}^{dt})

Figure 10: Representations of DSyntS associated to nominal expressions "*the dragon*" and "*the calm purple dragon*", and associated object terms (from $\Lambda(\Sigma_{dt})$).

In Montague semantics, a modifier like an adverb usually have a type like $(NP \to S) \to (NP \to S)$ (Carpenter, 1998). Since we encode constraints in the abstract signature and do not make distinctions between grammatical categories (see section 5.1), such a type would be translated with a type like $(G \to G) \to (G \to G)$, *i.e.*, a higher order type. To avoid such higher order types, we draw from Pogodalla's (2017) TAG encoding into ACGs and introduce MOD-like atomic types, used for modifiers in both Σ_{dst} and Σ_{ds}.

In Σ_{dt}, modifiers are considered like any other lexemes and encoded with a constant of type T (see figure 6, page 6). In Σ_{dst} and Σ_{ds}, MOD-like types are used.

5.5.1 Adverbial Modifiers

In a SemR, an adverbial group can govern up to two semantemes, namely the semanteme it modifies, and the first semantic actant of this semanteme (see figure 11, adverbial group "*for a specific reason*"). Therefore, an adverb is encoded with a constant of type[8] MOD → MOD, and MOD is interpreted by $\mathcal{L}_{dsyntRel}$ as $(T \to T) \to (T \to T)$, *i.e.*, a type similar to the classical one mentioned above. (10) gives the example of $often^{ds}$. Then, each constant of Σ_{ds} encoding a predicate that accepts an

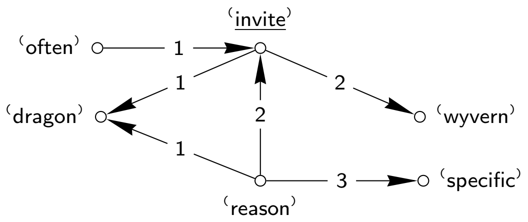


Figure 11: SemS associated to the expression "*The dragon often invites the wyvern for a specific reason*" that illustrate the semantic behavior of adverbial groups "*often*" and "*for a specific reason*".

adverbial modifier (*i.e.* verbs and adjectives[8]) have, in Σ_{ds}, its first argument being of type MOD (or MOD′ depending on the expressibility of the first argument), to enable a possible modification to take place. Following arguments have type G or G' (depending on the expressibility of arguments), and the final argument has type G (see (10)[9] for the example of $invite^{ds}_{rtr}$).

$$\begin{aligned} often^{ds} &: \text{MOD} \rightarrow \text{MOD} \\ invite^{ds}_{rtr} &: \text{MOD} \rightarrow G \rightarrow G \rightarrow G \\ \mathcal{L}_{dsyntRel}(\text{MOD}) &:= (T \rightarrow T) \rightarrow (T \rightarrow T) \\ \mathcal{L}_{dsyntRel}(often^{ds}) &:= \lambda\, M\, P\, X.\, M\, (\lambda\, x. \\ &\qquad \mathbf{ATTR}\ (P\ x)\ often^{dt})\ X \end{aligned} \tag{10}$$

With such an encoding, adverbs are represented by constants of order 2, and several adverbs can be used to modify one predicate (provided its type accepts adverbial modifiers) as shown in (11)[10].

$$\begin{aligned} & invite^{ds}_{rtr}\ (often^{ds}\ I^{ds}_{\text{MOD}}) \\ & :\ G \rightarrow G \rightarrow G \\ & invite^{ds}_{rtr}\ (often^{ds}\ (fsr^{ds}\ I^{ds}_{\text{MOD}})) \\ & :\ G \rightarrow G \rightarrow G \end{aligned} \tag{11}$$

5.5.2 Adjectival Modifiers

Adjectives have the same encoding except that, in their SemR, they only have one dependent, the semanteme they modify. Hence, another MOD-like type is used for adjectives: MODj. An adjective can be modified by an adverb ("*the **often** **calm** dragon*"), so the first argument of a constant encoding an adjective is of type MOD, followed by the core of the adjective constant type: MODj → MODj (see (12)[9]). Similarly to constants admitting adverbial modifiers, constants admitting adjectival modifiers also have their first argument being of type MODj.

$$\begin{aligned} purple^{ds} &: \text{MOD} \rightarrow \text{MOD}j \rightarrow \text{MOD}j \\ dragon^{ds} &: \text{MOD}j \rightarrow G \\ \mathcal{L}_{dsyntRel}(purple^{ds}) &:= \lambda M\ A\ P.\ (\mathbf{ATTR}\ (A\ P) \\ &\qquad (M\ (\lambda m.\ m)\ purple^{dt})) \end{aligned} \tag{12}$$

We then have the following equalities (expressions of γ^{dt}_{d}, γ^{dt}_{cpd} and $\gamma^{dt}_{cdoiwfsr}$ are given in figures 10a, 10b and 7b respectively, along with their DSyntS representation):

$$\begin{aligned} \gamma^{ds}_{d} &= dragon^{ds}\ I_{\text{MOD}j} \\ \mathcal{L}_{dsyntRel}(\gamma^{ds}_{d}) &:= \gamma^{dt}_{d} \\ \gamma^{ds}_{cpd} &= dragon^{ds}(calm^{ds}\ I^{ds}_{\text{MOD}} \\ &\qquad (purple^{ds}\ I^{ds}_{\text{MOD}}\ I^{ds}_{\text{MOD}j})) \\ \mathcal{L}_{dsyntRel}(\gamma^{ds}_{cpd}) &:= \gamma^{dt}_{cpd} \\ \gamma^{ds}_{cdoiwfsr} &= invite^{ds}_{rtr}(often^{ds}(fsr^{ds}\ I^{ds}_{\text{MOD}})) \\ &\qquad (dragon^{ds}(calm^{ds}\ I^{ds}_{\text{MOD}}\ I^{ds}_{\text{MOD}j})) \\ &\qquad (wyvern^{ds}\ I^{ds}_{\text{MOD}j}) \\ \mathcal{L}_{dsyntRel}(\gamma^{ds}_{cdoiwfsr}) &:= \gamma^{dt}_{cdoiwfsr} \end{aligned} \tag{13}$$

6 Evaluation and Results

6.1 Evaluation

Lexicons and vocabularies used here have between 30 and 50 lexemes approximately. It's a toy grammar, but it would be possible to extract data from SUD annotated corpora (Gerdes et al., 2018) or from the RL-fr (ATILF, 2025) to extend our coverage. Syntactic structures of MTT are dependency structures that roughly correspond to the ones of SUD. Labels on relations are different, but tree structures of MTT surface-syntactic representations are very similar to SUD representations. Syntactic representations of the encoding presented here are in accordance with the ones of annotated corpora from Grew (Guillaume, 2021). These ACGs are tested on 63 terms build upon $\Lambda(\Sigma_{dst})$. Tests scripts perform transduction operations toward $\Sigma_{semantic}$ as well as toward $\Sigma_{ssynt\text{-}tree}$. For 63 input terms, 430 output terms (*i.e.*, surface-syntactic structures) are generated and manually verified.

[8] An adverbial modifier can modify another adverbial modifier. Hence, adverb type should be MOD → MOD → MOD. For reasons of readability and computational complexity we won't consider this case here and use type MOD → MOD instead. However, it can be treated in the exact same way as adverbial modification on adjectives (see subsection 5.5.2).

[9] Interpretations of $invite^{ds}_{rtr}$ and $dragon^{ds}$ by $\mathcal{L}_{dsyntRel}$ are given in (8)

[10] fsr^{ds} is a constant representing the adverbial group "*for a specific reason*"

6.2 Results

This encoding of MTT into ACGs enables the representation of concepts such as LFs, semantic paraphrasing, surface-syntactic paraphrasing, the communicative structure (theme-rheme opposition) and its role, as well as some deep-syntactic paraphrasing. It also allows the representation and handling of several lexical phenomena, among which synonymy, idiomatic expressions, and modifiers behavior. Several modifiers can be used to modify the same predicate. ACGs of this encoding are from the specific class of ACGs of order 2, which parsing is decidable and polynomial.

7 Conclusion

Inspired by (de Groote, 2022, 2023), the encoding of MTT into ACG presented in this article is closer to more classical encodings such as Pogodalla's (2017) TAG encoding into ACG. In contrast to previous encodings (Cousin, 2023b,a, 2025), this encoding is of order 2, and not of higher order. This enables, on top of polynomial ACG complexity, that all used lexicons can be parsed. Consequently, this encoding can be used for generation as well as analysis purposes. It is the case for every transition between two signatures, independently of whether the whole transition between semantic and surface-syntax is considered, or only an intermediate transition. Theme-rheme opposition is accounted for in semantic and deep-syntactic representation levels, what enables a more accurate SSyntS generation from a SemR. Additionally, the lexemes do no longer define and decide exactly how many dependents they will have. They only constrain their mandatory dependents, and do so via the type of abstract constants encoding them. The addition of optional dependents, such as modifiers (adjectives and adverbs), is permitted.

Acknowledgments

I would like to thank Sylvain Pogodalla (Université de Lorraine, CNRS, Inria, LORIA) and reviewers for their comments and constructive remarks, which helped to improve this article.

References

ATILF. 2025. Réseau lexical du français (rl-fr). ORTOLANG (Open Resources and TOols for LANGuage) –www.ortolang.fr.

Bob Carpenter. 1998. Chapter 4: Applicative categorial grammar. In *Type-Logical Semantics*. MIT Press.

Marie Cousin. 2023a. Meaning-text theory within abstract categorial grammars: Toward paraphrase and lexical function modeling for text generation. In *Proceedings of the 15th International Conference on Computational Semantics*, pages 134–143, Nancy, France. Association for Computational Linguistics.

Marie Cousin. 2023b. Vers une implémentation de la théorie sens-texte avec les grammaires catégorielles abstraites. In *Actes de CORIA-TALN 2023. Actes des 16e Rencontres Jeunes Chercheurs en RI (RJCRI) et 25e Rencontre des Étudiants Chercheurs en Informatique pour le Traitement Automatique des Langues (RÉCITAL)*, pages 72–86, Paris, France. ATALA.

Marie Cousin. 2025. Adding Communicative Structure to the MTT into ACG Encoding. A long version is in preparation.

Philippe de Groote. 2001. Towards abstract categorial grammars. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 252–259, Toulouse, France. Association for Computational Linguistics.

Philippe de Groote. 2015. Abstract categorial parsing as linear logic programming. In *Proceedings of the 14th Meeting on the Mathematics of Language (MoL 2015)*, pages 15–25, Chicago, USA. Association for Computational Linguistics.

Philippe de Groote. 2022. Deriving Formal Semantic Representations from Dependency Structures. In *Logic and Engineering of Natural Language Semantics: 19th International Conference, LENLS19, Tokyo, Japan, November 19–21, 2022, Revised Selected Papers*, volume Lecture Notes in Computer Science, pages 157–172, Tokyo (JP), Japan. Springer.

Philippe de Groote. 2023. On the semantics of dependencies: Relative clauses and open clausal complements. In *Logic and Engineering of Natural Language Semantics - 20th International Conference, LENLS20, Osaka, Japan, November 18-20, 2023, Revised Selected Papers*, volume 14569 of *Lecture Notes in Computer Science*, pages 244–259. Springer.

Kim Gerdes, Bruno Guillaume, Sylvain Kahane, and Guy Perrier. 2018. SUD or surface-syntactic Universal Dependencies: An annotation scheme near-isomorphic to UD. In *Proceedings of the Second Workshop on Universal Dependencies (UDW 2018)*, pages 66–74, Brussels, Belgium. Association for Computational Linguistics.

Bruno Guillaume. 2021. Graph matching and graph rewriting: GREW tools for corpus exploration, maintenance and conversion. In *Proceedings of the 16th*

Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations, pages 168–175, Online. Association for Computational Linguistics.

Maxime Guillaume, Sylvain Pogodalla, and Vincent Tourneur. 2024. ACGtk: A Toolkit for Developing and Running Abstract Categorial Grammars. In *Functional and Logic Programming. 17th International Symposium, FLOPS 2024*, volume Lecture Notes in Computer Science of *Functional and Logic Programming. 17th International Symposium, FLOPS 2024*, pages 13–30, Kumamoto, Japan. Springer.

Lidija Iordanskaja, Richard Kittredge, and Alain Polguère. 1991. *Lexical Selection and Paraphrase in a Meaning-Text Generation Model*, pages 293–312. Springer US, Boston, MA.

Makoto Kanazawa. 2007. Parsing and generation as datalog queries. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 176–183, Prague, Czech Republic. Association for Computational Linguistics.

Makoto Kanazawa. 2017. Parsing and generation as datalog query evaluation. *IfCoLog Journal of Logics and Their Applications*, 4(4):1103–1211.

Ivana Kruijff-Korbayova and Mark Steedman. 2003. Discourse and information structure. *Journal of Logic, Language and Information*, 12:249–259.

François Lareau, Florie Lambrey, Ieva Dubinskaite, Daniel Galarreta-Piquette, and Maryam Nejat. 2018. GenDR: A generic deep realizer with complex lexicalization. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Igor Mel'cuk. 2001. *Communicative Organization in Natural Language: The semantic-communicative structure of sentences*.

Igor Mel'Čuk and Alain Polguère. 2021. Les fonctions lexicales dernier cri. In Sébastien Marengo, editor, *La Théorie Sens-Texte. Concepts-clés et applications*, Dixit Grammatica, pages 75–155. L'Harmattan.

I.A. Mel'čuk, I.A. Mel'čuk, D. Beck, and A. Polguère. 2012. *Semantics: From Meaning to Text*, volume 1 of *Semantics: From Meaning to Text*. John Benjamins Publishing Company.

I.A. Mel'čuk, I.A. Mel'čuk, D. Beck, and A. Polguère. 2013. *Semantics: From Meaning to Text*, volume 2 of *Semantics: From Meaning to Text*. John Benjamins Publishing Company.

I.A. Mel'čuk, I.A. Mel'čuk, D. Beck, and A. Polguère. 2015. *Semantics: From Meaning to Text*, volume 3 of *Semantics: From Meaning to Text*. John Benjamins Publishing Company.

J. Milićević. 2007. *La paraphrase: modélisation de la paraphrase langagière*. Sciences pour la communication. Lang.

Jasmina Milićević. 2006. A short guide to the meaning-text linguistic theory. *Journal of Koralex*, 8:187–233.

Sylvain Pogodalla. 2017. A syntax-semantics interface for Tree-Adjoining Grammars through Abstract Categorial Grammars. *Journal of Language Modelling*, 5(3):527–605.

Alain Polguère. 1990. *Structuration et mise en jeu procédurale d'un modèle linguistique déclaratif dans un cadre de génération de texte*. Ph.D. thesis.

Aarne Ranta. 2004. Grammatical framework. *J. Funct. Program.*, 14:145–189.

Sylvain Salvati. 2005. *Problèmes de filtrage et problème d'analyse pour les grammaires catégorielles abstraites*. Ph.D. thesis, Institut National Polytechnique de Lorraine.

Mark Steedman. 2024. Combinatory categorial grammar: An introduction. Course notes from ESSLLI 2024.

Leo Wanner, Bernd Bohnet, Nadjet Bouayad-Agha, François Lareau, and Daniel Nicklaß. 2010. Marquis: Generation of user-tailored multilingual air quality bulletins. *Applied Artificial Intelligence*, 24(10):914–952.