MOL 2021

The 17th Meeting on the Mathematics of Language

Proceedings of the Meeeting

December 13, 2021
University of Montpellier
Montpellier, France (online)

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 17th Meeting on the Mathematics of Language (MOL 2021), held online on December 13, 2021 as part of the federated conference *Mathematical Linguistics* (MALIN), which was hosted by the University of Montpellier and organized by Christian Retoré.

The volume contains six papers, which have been selected from a total of thirteen submissions, using the EasyChair conference management system.

We would like to express our sincere gratitude to the program committee members and the reviewers for MOL 2021, and to all who helped with the organization of MALIN.

Henrik Björklund and Frank Drewes (editors)

Organisers:

Program Chairs: Henrik Björklund, Umeå University (Sweden)
Frank Drewes, Umeå University (Sweden)
Organizeing Chair: Christian Retoré, University of Montpellier (France)

Program Committee:

Henrik Björklund, Umeå University (Sweden)
David Chiang, University of Notre Dame (USA)
Alexander Clark, King's College London (UK)
Philippe de Groote, INRIA Nancy – Grand Est (France)
Frank Drewes, Umeå University (Sweden)
Giorgio Magri, CNRS (University of Paris 8 (France)
Carlos Gómez-Rodríguez, Universidade da Coruña (Spain)
Jeffrey Heinz, Stony Brook University (USA)
Makoto Kanazawa, Hosei University (Japan)
Gegory Kobele, University of Leipzig (Germany)
Andras Kornai, Boston University (USA)
Andreas Maletti, University of Leipzig (Germany)
Jens Michaelis, Universität Bielefeld (Germany)
Nelma Moreira, University of Porto (Portugal)
Glyn Morrill, Universitat Politècnica de Catalunya (Spain)
Larry Moss, Indiana University Bloomington (USA)
Gerald Penn, University of Toronto (Canada)
James Rogers, Earlham College (USA)
Mehrnoosh Sadrzadeh, University College London (UK)
Anssi Yli-Jyrä, University of Helsinki (Finland)

Invited Speakers:

Miloš Stanojević, DeepMind London (UK)
Jane Chandlee, Haverford College (USA)

Table of Contents

A Generative Process for Lambek Categorial Proof Nets

Jinman Zhao and Gerald Penn
Department of Computer Science
University of Toronto
Toronto, Canada
{jzhao,gpenn}@cs.toronto.edu

Abstract

In this work, we present a stochastic, generative model for Lambek categorial proof sequents (Lambek, 1958). When a set of primitive categories is provided, this model, called PLC is able to generate all sequents of categories that are derivable in the Lambek Calculus with it. We also introduce a simple method to numerically estimate the parameters of the model from an annotated corpus. Then we compare our model with probabilistic context-free grammars (PCFGs). We show that there are several trade-offs with respect to using PLC in place of PCFG. Overall, PLC provides a layer of generalization in exposing numerical parameters of the formalism that is not directly accessible to PCFGs.

1 Introduction

Stochastic variants of different grammars have been proposed over the last several decades, and stochastic methods are very important for natural language processing. For example, stochastic context-free grammars (Huang and Fu, 1971), also known as probabilistic context-free grammar(PCFG), assign a probability to each production rule, normalized over their left-hand sides. Maximum Likelihood Estimation (MLE) and the Inside-Outside algorithm (Lari and Young, 1991) can be used to estimate these rule probabilities, given a training set. Other stochastic models have been proposed for Combinatory Categorial Grammar (Osborne and Briscoe, 1997). Bonfante and de Groote (2004) proposed a stochastic model for Lambek Categorial Grammar. In their work, probabilities are assigned to each leaf of a proof net, with the interpretation that a leaf will appear as the left conclusion of an axiom link with this probability. Probabilities are not attached to derivation rules and their model is fully lexicalized. It is not, however, generative, in the sense that it cannot deterministically produce proof nets from no input. In this respect, it more resembles a supertagger or an automaton-based probabilistic model, such as those that have been proposed for TAG (Bangalore and Joshi, 2010) or dependency grammar (Kübler et al., 2009). These are very useful for parsing. On the other hand, it would still be very instructive to have a generative process for Lambek proof nets more akin to a PCFG.

Pentus proved that both Lambek Grammars and product-free Lambek Grammars are context-free (Pentus, 1993, 1997). Using those constructive proofs, it is possible to convert a PCFG to a generative model of Lambek sequent derivability. The conversion takes exponential time as a function of the original PCFG's size, however, and what the numerical parameters of the PCFG correspond to in terms of proof nets provides little additional illumination. What we propose here is a "native" generative process defined directly for Lambek proof nets.

While the incremental enforcement of certain semantic criteria as necessary side conditions to proof-net construction (Roorda, 1991) is very difficult when a candidate sequent is known at the outset (leading, in one view, to the NP-completeness of the sequent derivability problem), in the generative orientation, it turns out not to be so difficult, as we show below. We also provide a simple parameter estimation method, and compare the results of training a PLC model through MLE to those of training a PCFG on an analogous annotated corpus.

2 Preliminaries

2.1 Lambek Calculus

The Lambek calculus was introduced by Lambek (1958). Given a set of primitive types, $Prim = \{p_1, p_2, p_3, \ ...\}$, and, in this work, only the two connectives \, /, we have the following rules:

Proceedings of the 17th Meeting on the Mathematics of Language, pages 1–13, Montpellier (online),
December 13, 2021. © 2021 Association for Computational Linguistics

$$\frac{\Gamma X \to Y}{\Gamma \to Y/X}\ (/R)$$

$$\frac{X\Gamma \to Y}{\Gamma \to X\backslash Y}\ (\backslash R)$$

$$\frac{\Gamma \to X \quad \Delta Y \Theta \to Z}{\Delta Y/X \Gamma \to Z}\ (/L)$$

$$\frac{\Gamma \to X \quad \Delta Y \Theta \to Z}{\Delta \Gamma X\backslash Y \Theta \to Z}\ (\backslash L)$$

$$\frac{\Gamma \to X \quad \Delta X \Theta \to Y}{\Delta \Gamma \Theta \to Y}\ (\mathrm{CUT})$$

along with the axioms, $p_i \to p_i$, for primitive types only. Note that we use the formulation of Pentus (1993). His restriction of these axioms to primitives will be important to us below. The primitive types, together with their closure under the available connectives, will be called *categories*.

We take Lambek Categorial Grammar(LCG) to be a 3-tuple $G = (\Sigma, D, f)$, where Σ is a finite alphabet, D is a distinguished category, and f is a mapping of members of the alphabet $t \in \Sigma$ to a single category, where the possible categories have been derived from the set *Prim* by induction over the connectives. Note that this is a departure from Pentus (1993), in that there is no lexical ambiguity here, as our input will consist of candidate proof sequents, in which a single category is already known for each word. In general, $f(t)$ is a finite set of categories. Lambek (1958) did not define lexical mappings.

2.2 Proof Nets

Roorda (1991) adapted the proof nets of linear logic to the Lambek calculus. The treatment here is taken from Penn (2004). A (Lambek) proof net consists of a lexically unfolded sequence of terminal formulae, a spanning linkage of the resulting sequence of axiomatic formulae and a variable substitution.

As a running example of how to parse with a proof net, let us consider the sequent:

$$S/(NP\backslash S) \quad (NP\backslash S)/NP \quad NP \models S$$

2.2.1 Labelled Terms

Given a sequent, the first step is to add polarities to each category. By convention, all LHS categories receive negative polarity and the RHS category receives positive polarity. We also label each formula with a variable. The result on our example would be:

$$S/(NP\backslash S)^{-}{:}a \quad (NP\backslash S)/NP^{-}{:}b \quad NP^{-}{:}c \quad S^{+}{:}d$$

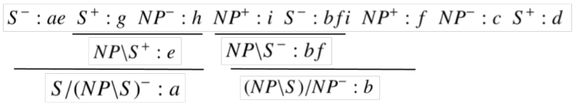
2.2.2 Lexical Unfolding

We then apply the following substitution rules to each labelled category from the previous step until no more rules can be applied:

$$(A\backslash B)^{-}{:}t \to A^{+}{:}u \;\; B^{-}{:}tu$$
$$(A\backslash B)^{+}{:}v \to B^{+}{:}v' \;\; A^{-}{:}u[v := \lambda u.v']$$
$$(A/B)^{-}{:}t \to A^{-}{:}tu \;\; B^{+}{:}u$$
$$(A/B)^{+}{:}v \to B^{-}{:}u \;\; A^{+}{:}v'[v := \lambda u.v']$$

Here, we are expanding every labelled category into a string of labelled categories until only primitive labelled types remain. Note that when positive-polarity categories unfold, they add new variables, u and v'. We will refer to v as a *lambda node* or *lambda variable* when this happens, in honour of the correspondence between these labels and lambda-terms in the corresponding labelled deduction system (Roorda, 1991). u and v' will likewise be referred to as the *daughters* of v.

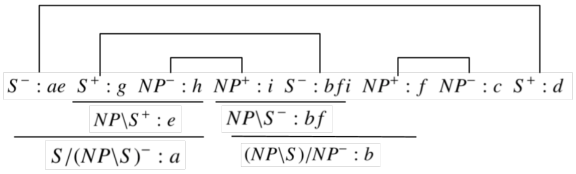
The above example produces:


$S^{-}:ae$ $S^{+}:g$ $NP^{-}:h$ $NP^{+}:i$ $S^{-}:bfi$ $NP^{+}:f$ $NP^{-}:c$ $S^{+}:d$
$NP\backslash S^{+}:e$
$NP\backslash S^{-}:bf$
$S/(NP\backslash S)^{-}:a$
$(NP\backslash S)/NP^{-}:b$

Note that negative-polarity categories are generally labelled with strings of variables, not merely one variable.

2.2.3 Axiomatic Linkage

Next, we must link matching primitives of opposite polarities together in a half-planar graph. Each polarized primitive must be linked exactly once. One possible linkage for the above example is:


$S^{-}:ae$ $S^{+}:g$ $NP^{-}:h$ $NP^{+}:i$ $S^{-}:bfi$ $NP^{+}:f$ $NP^{-}:c$ $S^{+}:d$
$NP\backslash S^{+}:e$
$NP\backslash S^{-}:bf$
$S/(NP\backslash S)^{-}:a$
$(NP\backslash S)/NP^{-}:b$

The edges of this half-planar graph will be known as *axiom links.*

2.3 LC-graph construction

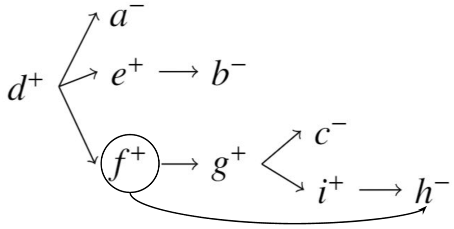
The first two steps in this process are guaranteed to succeed exactly once. The third step is not — a half-planar linkage may not exist or more than one may exist. Furthermore, even if one does exist, the fourth step below, which can be undertaken in parallel, may fail, requiring the search for more half-planar linkages to continue.

Given a sequence of axiomatic formulae and a partial linkage, every link induces the addition of one or more edges to an LC-graph, in which every node corresponds to a single variable in the category labels.

Let the candidate sequent's LC-graph be a directed graph $G = \langle V, E \rangle$, such that V is the set of all variables that appear in its category labels and E is the smallest set such that:

- for every $v \in V$, if v is a lambda-variable, then for both daughter variables of v, u and v', $(v, u) \in E$, and $(v, v') \in E$, and
- for every axiom link matching $p^+ : u$ and $p^- : t$ and for every v in the string t, $(u, v) \in E$.

The LG-graph for above linkage and sequence of axiomatic formulae is:



2.3.1 Integrity Criteria

Penn (2004) proved that an LCG sequent is derivable iff the following three criteria are true of the LC graph, G, of some axiomatic linkage:

- **I(1)** there is a unique node in G with in-degree 0, from which all other nodes are path-accessible.
- **I(2)** G is acyclic.
- **I(3)** for every lambda-node $v \in V$, there is a path from its plus-daughter, v', to its minus-daughter, u.
- **I(CT)** for every lambda-node $v \in V$, there is a path in G, $v \rightsquigarrow x$, such that x labels a negative-polarity category, x has out-degree 0 and there is no lambda-node $v' \in V$ such that $v \rightsquigarrow v' \to x$.

The second criterion can be enforced incrementally. In general, it appears that the others may result in a violation that would cause backtracking and the selection of other axiomatic links in the third step. The value of annotating proof nets with probabilities inheres in its ability to direct us towards axiom links that are likely to lead to the successful construction of a proof net.

3 Generative Process

A generative process is not necessarily just for parsing. It can begin with no input whatsoever, and generate a derivable sequent, along with a numerical score that could correspond to the probability of a corresponding string of words. It can also be made to backtrack after every output sequent, thereby enumerating an infinite sequence of derivable sequents.

Note that in our formulation of the generative process here, however, there is no guarantee that a sequent in fact corresponds to a string of words. To achieve this, we must condition our process to stop only at categories that are attested in the lexicon. What we present here is a generative process for derivable sequents.

To generate any well-formed sequent, we may begin only with $p \to p$, for a primitive type p. Note that p is not required to be D, the distinguished category of an intended grammar, and because there is no requirement that D be primitive, some choices of D will not be allowed here. Nevertheless, because well-formed proof nets are closed under cyclic permutations of their polarized primitive categories, we can assume without loss of generality that the right-hand side of the generated sequent is a primitive category.

Our generative process proceeds in two phases. Both expand the proof net as well as its corresponding LC graph. The first phase adds non-lambda nodes to the LC graph, and the second phase adds lambda nodes to the LC graph. Corollary 4.14 will prove that any proof net can be generated in this fashion: all of the non-lambda nodes first, followed by all of the lambda nodes.

Adding non-lambda nodes affects the number of axiomatic formulae; adding lambda nodes does not. Unlike the method outlined in the previous section for derivability when an input candidate sequent is given, the correctness conditions of the LC-graph will be satisfied throughout the derivation. They will therefore be satisfied in the final proof net.

We will need the following definition for growing larger categories from smaller categories:

Definition 3.1 (Term tree)**.** *Every labelled category in a sequent corresponds to a binary tree of labelled categories. Each subtree in the term tree is rooted in a labelled category $X^{+/-} : v$, where X is a category and v is a sequence of variables. Every interior subtree has a positive child and a negative child, and is licensed by one of the rules shown in Section 2.2.2.*

Figure 1 shows the term tree of $(A/B)/C^- : a$ as an example. Unlike Section 2.2.2, we will think of term trees as corresponding not to top-down unfoldings of complex categories, but to bottom-up compositions of more complex categories from simpler categories.

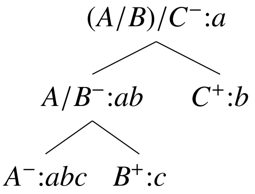
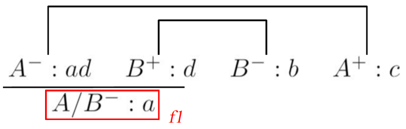


Figure 1: An example term tree.

3.1 Adding Non-lambda Nodes

This phase of the process consists of the iteration of the following steps in sequence zero or more times, until we elect to stop. Each iteration will add one more link and one more matching pair of axiomatic primitives (of opposite sign).

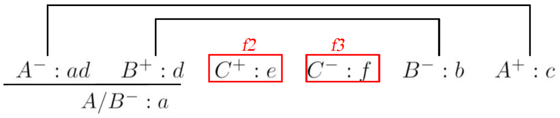
Suppose, for example, that we have already derived the following proof net:



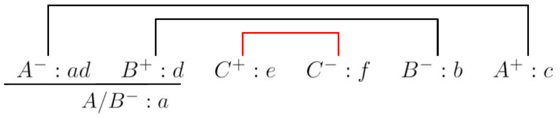
We first pick any negative term (not a subterm) from the existing sequent, e.g., $f_1 = A^- : a$, as the redex. Then we add two new, adjacent polarized primitives, connected by a new link of distance 1 to the immediate left or immediate right of that redex. Both new primitives will have a different, new variable as their labels. The new positive primitive should be the closer of the two to the negative redex.

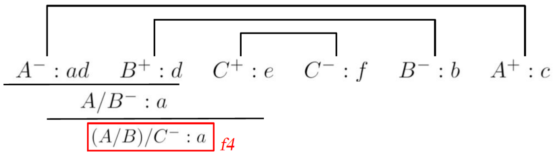
We then combine the positive primitive and the redex to form a new complex, negative category. At the same time, we give this new complex category the old semantic label of the redex, and assign the string consisting of this old label followed by the fresh variable on the new positive primitive as the new label of the redex.

In our example, suppose we elect to add the new primitives to the right of $f1$. Then we have:

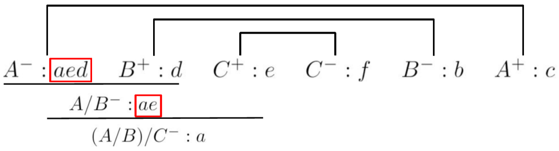


We add a link to join $f2$ and $f3$, and create a new formula $f4 = (A/B)/C^- : a$. In different words, we combine $f1$ and $f2$ into a larger term tree with the root, $f4$.





Then we replace the label on $f1$ with *ae*. We also replace *a* with *ae* in all of the descendants of $f1$'s term tree, if any.



We could have instead added $f3 = C^- : f$ and $f2 = C^+ : e$ (in this order) on the *left* of $f1$, and joined $f2$ and $f1$ with $f4 = (C\backslash A)^- : a$.

Regardless of the choice of left or right, we must also modify the corresponding LC-graph:

1. Add new nodes e and f,

2. for the unique value of x for which there is already an edge (x, a), add a new edge (x, e), and

3. add the new edge (e, f).

These measures will retain the correctness criteria in the LC-graph because, in this phase, there are no lambda nodes yet. So the new sequent will be derivable in the Lambek calculus if the sequent that began this iteration was.

We can repeat these steps as many times as we like in the first phase, and then decide to stop. If we apply zero iterations of the first phase, we cannot apply any iterations of the second phase either, and we will be left with the sequent $p \to p$.

3.2 Adding Lambda Nodes

Like the first phase, the second phase consists of the iterative application of a sequence of steps that we may elect to apply any number of times.

Every step in the second phase will preserve the number of axiomatic primitives and links, but will add one more lambda node. The following definition is useful:

Definition 3.2 (Anterior node)**.** *An anterior node, a, in an LC-graph is a positive node for which there is no lambda node on the path that leads from the root to a apart from possibly a itself.*

Definition 3.3 (Terminal node)**.** *A terminal node, t, in an LC-graph is a node with an out-degree of 0 and in-degree of 1.*

In Figure 2, j and i are both lambda nodes, but only j is an anterior lambda node. a, c, e and g are all positive non-lambda nodes, but only a, the root, is anterior. b and d are terminal nodes. f and h are negative, but not terminal because of their in-degrees. Terminal nodes are always negative because of their out-degree. Because of the restriction on their in-degrees, they correspond exactly to the labels of the LHS categories of the sequent.

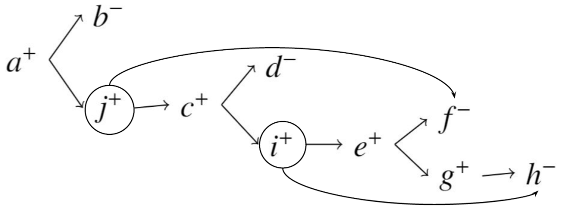


Figure 2: An example LC graph.

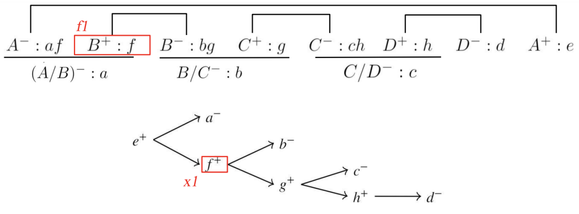


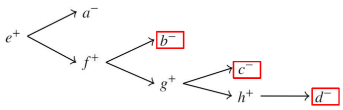
Figure 3: Running example for adding lambda nodes.

The redex, x_1, in each step of this phase will always be a positive node in the LC-graph that satisfies two other requirements:

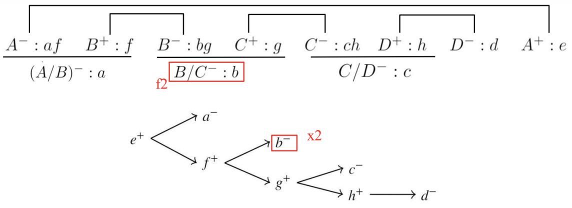
1. x_1 is anterior, and
2. x_1 has more than one terminal descendant.

Let $f1$ be the term or subterm labelled with x_1 (i.e. $f1 = X_1^+ : x_1$) in the proof net. We can do the following to the proof net (take Figure 3 as an example):

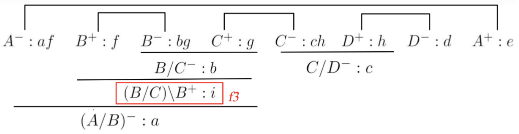
1. Let S be the set of terminal LC-graph nodes that are path-accessible from x_1.



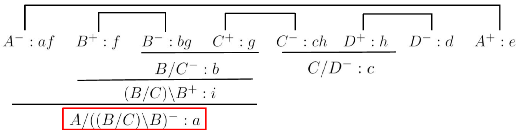
2. Pick $x_2 \in S$ as an outermost (either leftmost or rightmost) variable in S, arranged by the order of the elements in S within the current sequent. Let $f2$ be the (sub)term labelled with x_2 (i.e $f2 = X_2 - : x_2$). In the running example, we choose x_2 to be the leftmost variable.



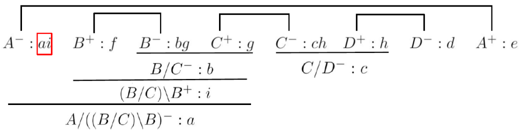
3. Create new formula $f3 = X_2 \backslash X_1 + : x_3$, with $f1$ as the left child of $f3$ and $f2$ as the right child of $f3$. Insert $f3$ into the unfolding below $f1$.



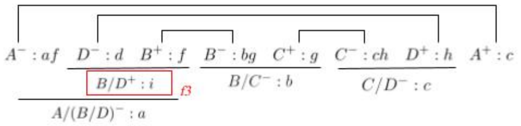
4. Update the categories in $f3$'s ancestors.



5. Replace x_1 with x_3 in all terms that are descendants of $f3$'s parent but not of $f3$.



In our example, we could have alternatively chosen $x_2 = d$, the rightmost variable in S, in which case $f3 = X_1/X_2+ : x_3$, with $f2$ as the left child $f1$ as the right child.



With either choice, we then modify the corresponding LC-graph:

1. Add x_3 as a new lambda node.

2. If x_1 is not the root, then for the unique y such that (y, x_1), replace this edge with (y, x_3). If x_1 is the root, do nothing.

3. Add edges (x_3, x_1) and (x_3, x_2).

Because x_1 was chosen to have more than one terminal descendant, Lemma 4.2 in the next section ensures that, after this step, the integrity criteria of the LC-graph are still satisfied. So the new sequent will be derivable in LCG.

3.3 End-to-end Example

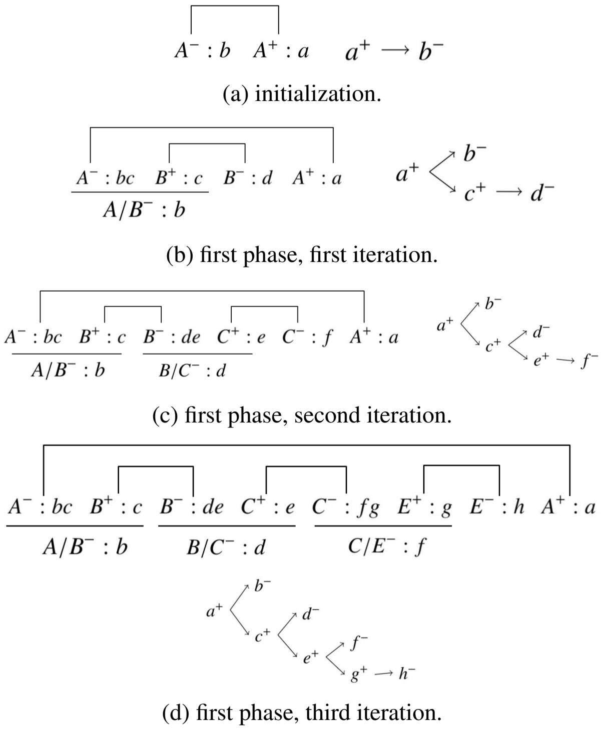
Figure 4 and Figure 5 depict the first and second phases, respectively, of a run of the generative process.

4 Coverage

Theorem 4.1. *Every sequent that is derivable in the product-free Lambek calculus is derivable through the PLC generative process.*

All derivable sequents are witnessed by a valid proof net P paired with a corresponding LC-graph G. We will prove the above theorem in this section, by induction on the number of nodes in G. The base case, sequents of the form $A \models A$, where A is a primitive, are trivial to generate. In other instances, it suffices to show that every sequent, consisting of a valid proof net, P with corresponding LC-graph, G, can be derived from some other sequent (P', G') by running one more iteration of either the first phase or the second.

We will consider the following two cases: G contains lambda nodes, and G contains no lambda nodes.



(a) initialization.

(b) first phase, first iteration.

(c) first phase, second iteration.

(d) first phase, third iteration.

Figure 4: An example run of non-lambda node generation.

4.1 LC-graph contains lambda nodes

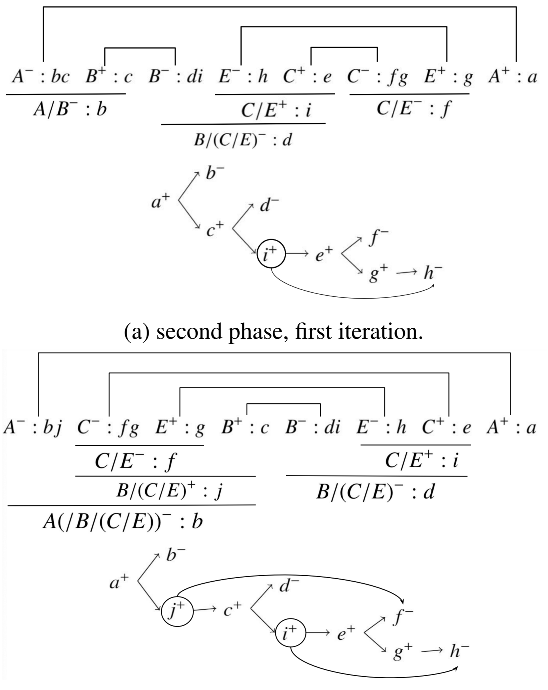
Lemma 4.2. *In any integral LC-graph, for any anterior lambda node v, there must be at least one terminal node v' such that $v \rightsquigarrow v'$.*

Proof. Every lambda node has two immediate descendants, one positive and one negative, and only these negative immediate descendants have indegrees of more than 1 (specifically, 2). If all negative nodes v' such that $v \rightsquigarrow v'$ had indegrees of 2, then they would all be negative immediate descendants of lambda nodes. Because v is anterior, those lambda nodes cannot be ancestors of v, and therefore must be v itself or descendants of v, and thus the x implied by **I(CT)** does not exist. □

Lemma 4.3. *If x, y are positive, w is negative, $x \rightsquigarrow w$ and $y \rightsquigarrow w$, then either $x \rightsquigarrow y$ or $y \rightsquigarrow x$.*

Proof. If w is the negative immediate descendant of a lambda node, z, then it is possible that one of x, y reaches w via z, and the other reaches it via the positive sibling of w, q, by **I(3)**. But $z \rightarrow q$, and so the result holds.

If w is not the negative immediate descendant of a lambda node, then its in-degree is 1 and its ancestors are all positive, with in-degrees of 1. So again the results holds. □



(a) second phase, first iteration.

(b) second phase, second iteration.

Figure 5: An example run of lambda node generation.

Lemma 4.4. *In any integral LC-graph, G, for any nodes $a, b, c \in G$, if $a \not\rightsquigarrow b$ and $b \not\rightsquigarrow a$, then there is no c such that $a \rightsquigarrow c$ and $b \rightsquigarrow c$.*

Proof. The only nodes with in-degrees greater than 1 (i.e., negative immediate descendants of lambda nodes) have out-degrees of 0. Thus if there were such a c, it would need to be a negative immediate descendant of some lambda node, d, itself. But even in this case, either $a \rightsquigarrow d \rightsquigarrow b$ or $b \rightsquigarrow d \rightsquigarrow a$ because of **I(3)**. □

Lemma 4.5. *For each term tree T in P, if $f2 = F2^{+/-} : y$ is a subterm of $f1 = F1^- : x$ in T such that $|y| = 1$ and there is no positive subterm of $f1$ of which $f2$ is a proper subterm, then for any ancestor, w, of x in G, $w \rightsquigarrow y$ in G.*

Proof. If $f1 = f2$, then $x = y$. Otherwise, the action of the negative unfolding rules on $f1$ guarantees that the primitive negative term label that reflects x will also reflect y, and so any node in G that leads to x through an axiom link also leads to y. We may consider the case when x is the negative immediate descendant of a lambda node, z, to be exceptional, because z points directly to x as a result of the action of the positive unfolding rules. But even in this case, by **I(3)**, $z \rightsquigarrow x$ through its positive immediate descendant, and thus also through an axiom link. □

Lemma 4.6. *For each term tree T in P, if $f2 = F2^{+/-} : y$ is a subterm of $f1 = F1^+ : x$ in T, where x is a lambda node and $|y| = 1$ (a variable), then $x \rightsquigarrow y$ in G.*

Proof. If $f1 = f2$, then $x = y$. Otherwise, because x is a lambda node, it has a positive immediate descendant, a, and negative immediate descendant, b, in G. b, in particular, corresponds to the negative immediate subterm of $f1$, $f3 = F3^- : b$. If $f2 = a$ or $f2 = b$, the lemma follows from the action of the positive unfolding rules.

Otherwise, we can obtain the result by induction on the number n of positive subterms of $f1$, q, for which $f2$ is a proper subterm of q. In the base case, $n = 1$, $f2$ is nested inside exclusively negative subterms of $f3$, and so by Lemma 4.5, $x \rightsquigarrow y$.

If $n > 1$, then let $f4 = F4^+ : z$ be the smallest positive subterm containing $f2$ other than possibly $f2$. By induction, $x \rightsquigarrow z$ in G. Because $f4$ contains $f2$, $F4$ is not primitive, and so z is a lambda node. Thus again by induction, $z \rightsquigarrow y$. □

Lemma 4.7. *For each term tree T in P, if $f2 = F2^{+/-} : y$ is a proper subterm of $f1 = F1^- : x$ in T, if $|y| = 1$ (a variable), then for any ancestor, w, of x in G, $w \rightsquigarrow y$ in G.*

Proof. Proof by induction on the number n of positive subterms of $f1$, q, such that $f2$ is a proper subterm of q. The base case, $n = 0$, follows by Lemma 4.5.

Otherwise, $n > 0$. Again, let $f3 = F3^+ : z$ be the smallest positive subterm of T containing $f2$ except possibly $f2$. By induction, every ancestor of x, $w \rightsquigarrow z$ in G. Because $f3$ contains $f2$, $F3$ is not primitive, and so z is a lambda node. Thus the result follows by Lemma 4.6. □

Lemma 4.8. *For every sequent with a primitive right-hand side $p^+ : r$, the axiom link emanating from $p^+ : r$ is incident to the axiomatic reflection of the label of some left-hand side term, i.e., $r \rightarrow b$ in G, for some left-hand side term, $B^- : b$.*

Proof. The axiom link must be incident to a negative-polarity primitive in the unfolding, which is either labelled with a string that includes the label of some left-hand side term, or with a string that includes the negative immediate descendant of some lambda node. In the latter case, the negative immediate descendant would be cut off from its positive immediate descendant sibling, in violation of **I(3)**, because p is primitive (and so there are no paths that would transit to b via r). □

Lemma 4.9. *Let X be the set of all axiomatic formulae of P, v be an anterior lambda node in G, and $D(v)$ be the set of terminal nodes w in G for which $v \rightsquigarrow w$ in G. Given a set of nodes, V, let $Q(V) = X \cap \{f \mid$ there exists a formula $f1$ with a label in V and f as one of its subterms in some term tree$\}$. Divide X into two subsets: $X1 = Q(\{v\}) \cup Q(D(v))$ and $X2 = X \backslash X1$. There is no axiomatic link connecting one formula in $X1$ with another one in $X2$.*

Proof. If v is the root of G, then $X1 = X$ and $X2 = \phi$. Otherwise, suppose there is such a link connecting $x1 \in X1$ and $x2 \in X2$. Let $A^{-/+} : a$ be the root of the term tree that contains $x2$.

Case 1: a is positive. A must be a primitive, otherwise, a would be a lambda node, and thus the only anterior node in G would be a itself. So $x2 = A^{+} : a$. By Lemma 4.8, there exists a negative node b that labels the root of some term tree in G such that $a \to b$. Let $R^{-} : r$ be the root of the term tree that contains $x1$. If $b = r$, then by Lemma 4.7, b is the label of both $x1$ and its witness, $f1$. $b \neq v$ (wrong polarity), and so $b \in D(v)$, $v \rightsquigarrow b$, and because b is terminal, $v \rightsquigarrow a$, which would mean that G has a cycle. Thus $b \neq r$, but then b labels the root of a different term tree, and so $x2 = a \twoheadrightarrow x1$ after all. But a is the root of G, and so $x1 \twoheadrightarrow a = x2$ either.

Case 2: a is negative, and so it labels a left-hand-side term in the sequent, and thus it is terminal. So there exists a unique node b such that $b \to a$ in G. Because a dominates $x2 \in X2$, $v \not\rightsquigarrow a$, and so $v \not\rightsquigarrow b$. Furthermore, every node in the label of $x1$ is accessible from v. If $x1 \in Q(\{v\})$, then this follows from Lemma 4.6. If $x1 \in Q(D(v))$, then it follows from Lemma 4.7 and the definition of $D(v)$.

Case 2a: $b \not\rightsquigarrow v$ either. By Lemma 4.4, there is no node v' in G such that $v \rightsquigarrow v'$ and $b \rightsquigarrow v'$. A link between $x1$ and $x2$ will guarantee that there is such a v', however, because, by Lemma 4.7, every node in the label of $x2$ is accessible from b.

Case 2b: $b \rightsquigarrow v$. First, we begin by noting that $x2$ cannot simultaneously be negative and contain a in its label, or else the label of $x1$ is a single node that points to a, and hence $v \rightsquigarrow a$. So there must be a positive $C^{+} : c$ subterm of $A^{-} : a$ which in turn contains $x2$ as a subterm. Choose the highest such subterm. Then because c is in the negative reflection of a, $b \to c$. Furthermore, either c is the label of $x2$ or, by Lemma 4.6, every node of $x2$ is accessible from c.

Let w be the negative member of $\{x1, x2\}$ and the target of the link between them. The nodes labelling w are accessible from both v (via $x1$) and c (via $x2$). By Lemma 4.3, either $v \rightsquigarrow c$ or $c \rightsquigarrow v$.

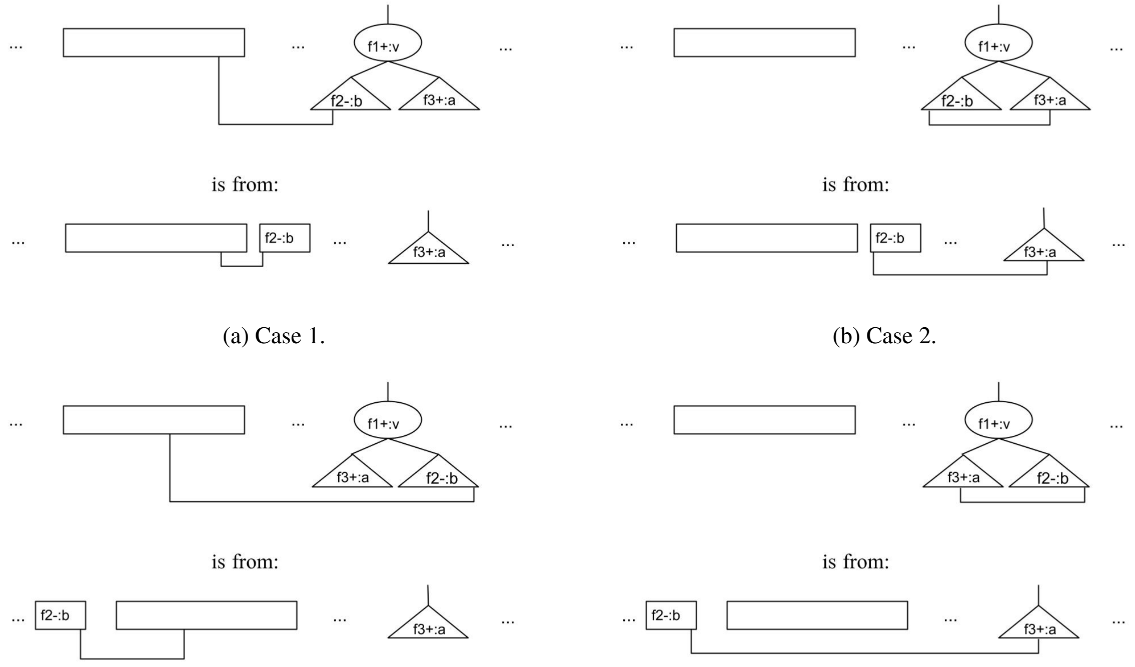
If $v \rightsquigarrow c$, then recall that both $b \to c$ and, by assumption, $b \rightsquigarrow v$, which means either $v = b$, which contradicts $v \not\rightsquigarrow a$, or $v = c$, which contradicts $x2 \in X2$. If $c \rightsquigarrow v$, then because v is anterior, c is the label of $x2$, which is positive, and therefore the source of the link between $x1$ and $x2$. As a result, the nodes of $x1$ are accessible from v, and immediately accessible from $x2$. Similar to the proof of Lemma 4.3, This might happen if $x1$ is the negative immediate descendant of some lambda node, z, but now we know that $x1$ and $x2$ are connected by an axiom link, and so $v \rightsquigarrow z$ and, by **I(3)**, $z \rightsquigarrow x2 = c$. Thus $v = c$, which is again a contradiction. □

Lemma 4.10. *For every anterior lambda node v in G, the LHS terms labelled by each of v's accessible terminal nodes must be contiguous in the sequent.*

Proof. This follows from Lemma 4.9. The terms that are labelled by each of the $D(v)$ are not connected by axiom links to the other terms, with the exception of the term that contains v itself, and even then only to the subterm rooted at v. If some of these other terms were interspersed among the former, then either the linkage would not be half-planar, or the underlying LC-graph would not be connected. Even the term that contains v itself cannot appear between two terms labelled by members of $D(v)$, because either v itself, which is positive, is the label of the term, and so must be the RHS category of the sequent, or it is not, in which case part of that term resides in $X2$, leading to the same contradiction. □

Theorem 4.11. *If G contains at least one lambda node, then (P, G) could have been generated in PLC from a proof net with any arbitrary anterior lambda node removed.*

Proof. Suppose v is the variable of the anterior lambda node in G, b is its negative immediate descendant in G and a, its positive immediate descendant.



(a) Case 1.

(b) Case 2.

(c) Case 3.

(d) Case 4.

Figure 6: Four cases of the respective ordering of $f2$, $f3$ and the contiguous subsequence of terms labelled with accessible terminal nodes, as given by Lemma 4.10 (depicted with a rectangle).

Let $f1$ be the formula labeled with v, $f2$ be the formula labeled with b and $f3$ be the formula labeled with a. In the parlance of Lemma 4.9, $X1$ consists of $f2$, $f3$, and the subsequence of terms labelled by the accessible terminal nodes of v. By Lemma 4.10, this subsequence is contiguous. By Lemma 4.9, the axiom links that have one side incident to any of these have both sides incident to these. There are therefore $2 \times 2 \times 2 = 8$ possible combinations: (1) whether $f2$ occurs to the left or right of $f3$, (2) whether the leftmost (resp. rightmost) axiomatic reflection of $f2$ connects to $f3$ or to a reflection within the contiguous subsequence and (3) whether the contiguous subsequence occurs to the left or right of $f1$. For brevity, Figure 6 shows the four possible cases in which we choose "left" for (3) — the other four are the symmetric duals of these.

In each case, such a derivation exists through a single iteration of the second phase of the generative process, as shown in Figure 6.

Note: the contiguous subsequence is, in fact, the left-hand side of the subsequent that derives $f1$. □

4.2 LC-graph does not have lambda nodes

Lemma 4.12. *For an arbitrary valid proof net P with its corresponding LC-graph G, if G does not contain any lambda node, then the sequent must be one of:*

$$A \models A$$

or

$$..., \phi/A, A... \models ...$$

or

$$..., A, A\backslash\phi.... \models ...$$

where A is a primitive and ϕ is any category.

Proof. By structural induction on the proof of the sequent. We will refer to A as a *reduction point* in the sequent. □

Theorem 4.13. *If G has at least three nodes and contains no lambda nodes, then* (P, G) *could have been generated in PLC from a proof net with any arbitrary reduction point removed.*

Proof. If G contains only two nodes, then the sequent must be $A \models A$. According to Lemma 4.12, in the second case, a reduction point exists, and corresponds to the result of a non-lambda-node addition step in PLC, as shown in Figure 7. Note that the term trees involving the reduction point must be adjacent in the sequent.

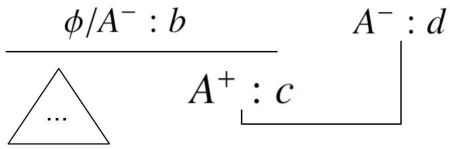


Figure 7: Formula f, f_1 and f_2 in proof net.

The third case is symmetric to the second. □

Corollary 4.14. *Any proof net and its corresponding LC-graph can be generated by starting with $p \rightarrow p$, followed by a sequence of steps of adding non lambda nodes then followed by a sequence of steps of adding lambda nodes.*

Proof. by Thm 4.11 and Thm 4.13, we can always put steps of adding non lambda nodes before the steps of adding lambda nodes. □

5 Adding Probability

Maximum likelihood parameter estimation can likewise be accomplished by separately parameterizing the two phases.

5.1 Phase 1: Adding Non-lambda Nodes

The first phase iteratively picks a category A and expands it to either $A/P \cdot P$ or $P \cdot P\backslash A$ for some primitive P. So for all primitives P, we must assign a probability to these two expansion schemes, as well as to $A \rightarrow A$, which means that A has been skipped with no expansion. The estimates are then simply the relative frequencies with which these rules have been chosen in this phase.

Below is a tiny example corpus with only two proof nets and one primitive:

$p1 :\ N1/N2 \quad N3 \quad \models N4$
axiom links : $(1,4),(2,3)$
$p2 :\ N1/N2 \quad N3 \quad N4\backslash N5 \quad \models N6$
axiom links : $(3,4),(2,5),(1,6)$

Here, the primitive category is N, and the numbers index the instances of it within the sequent.

For $p1$, we start from $N1 \models N4$, and proceed with these steps:

1. $N1 \rightarrow N1/N2 \quad N3$
2. $N1/N2 \rightarrow N1/N2$ (no expansion)
3. $N3 \rightarrow N3$ (no expansion)

For $p2$, we start from $N1 \models N6$, and follow with these steps:

1. $N1 \rightarrow N1/N2 \quad N5$
2. $N5 \rightarrow N3 \quad N4\backslash N5$
3. $N1/N2 \rightarrow N1/N2$ (no expansion)
4. $N3 \rightarrow N3$ (no expansion)
5. $N4\backslash N5 \rightarrow N4\backslash N5$ (no expansion)

Between the two derivations, there are a total of 8 steps, and so we can conclude that:

1. $P(A \rightarrow A/N \quad N) = \frac{2}{8}$
2. $P(A \rightarrow A) = \frac{5}{8}$
3. $P(A \rightarrow N \quad N\backslash A) = \frac{1}{8}$

Using this distribution, we know that:

$$\begin{aligned} P(p1) &= \tfrac{2}{8} \times \tfrac{5}{8} \times \tfrac{5}{8} = \tfrac{25}{256} \\ P(p2) &= \tfrac{2}{8} \times \tfrac{1}{8} \times \tfrac{5}{8} \times \tfrac{5}{8} \times \tfrac{5}{8} = \tfrac{125}{16384}. \end{aligned}$$

5.2 Phase 2: Adding Lambda Nodes

Once every category has been touched with a decision not to expand, it is time to start adding lambda nodes in the second phase. Let us modify the above corpus so that it requires a second phase, and yet can be derived through an identical sequence of steps as in the first phase:

$p1 :\ N1/N2 \quad \models N4/N3$
axiom links : $(1,4),(2,3)$
$p2 :\ N1/(N3\backslash N2) \quad N4\backslash N5 \quad \models N6$
axiom links : $(3,4),(2,5),(1,6)$

Both $p1$ and $p2$ contain one lambda node. During the second phase, we must first determine the place to add a lambda node, and then select either $(\backslash R)$ or $(/R)$. $p1$ applies $(/R)$ once and $p2$ applies $(\backslash R)$ once. So, for this corpus, $P(/R) = P(\backslash R) = \frac{1}{1+1} = \frac{1}{2}$.

The only remaining problem is how to determine where to add lambda nodes. Once this problem is solved, then:

$$\begin{aligned} P(p1) &= \tfrac{25}{256} \times P(\textit{pick positions in } p1) \times P(/R) \\ P(p2) &= \tfrac{125}{16384} \times P(\textit{pick positions in } p2) \times P(\backslash R) \end{aligned}$$

Now we will present our method for selecting places to add lambda nodes.

Definition 5.1 (Breakable)**.** *A node v in an integral LC-graph is **breakable** iff we can first:*

1. *if v is not the root, add a lambda node a between the edge $u \rightarrow v$, or*

2. *if v is the root, add a lambda node a such that $a \rightarrow v$,*

then pick a terminal node w such that $v \rightsquigarrow w$, then add (a, w), and the integrity of the LC-graph is preserved.

If v can be picked as $x1$ in Section 3.2, then it is breakable, but the inverse may not be true.

Lemma 5.1. *An LC-graph satisfies I(CT) iff for every lambda node v:*

$$\begin{aligned} &|\{u \mid u \text{ lambda node } \& \ v \rightsquigarrow u\}| \\ &\quad < \ |\{w \mid w \text{ negative } \& \ v \rightsquigarrow w\}| \,. \end{aligned}$$

Proof. Every lambda node u points to its positive immediate descendant. So:

$$\begin{aligned} &|\{u \mid u \text{ lambda node } \& \ v \rightsquigarrow u\}| \\ &\quad \leq \ |\{w \mid w \text{ negative } \& \ v \rightsquigarrow w\}| \,. \end{aligned}$$

And the node x asserted by I(CT) is a negative node that is not accessible from any lambda node. So the inequality is strict. □

Lemma 5.2. *A node v is breakable iff:*

$$\begin{aligned} &|\{u \mid u \text{ lambda node } \& \ v \rightsquigarrow u) + 1\}| \\ &\quad < \ |\{w \mid w \text{ negative } \& \ v \rightsquigarrow w\}| \,. \end{aligned}$$

Proof. This readily follows from Lemma 5.1. □

Theorem 5.3. *If a node v is breakable, then picking it as $x1$ during Section 3.2 will not change the breakability of any node u such that $v \not\rightsquigarrow u$.*

Proof. If $u \not\rightsquigarrow v$, the constraints of breakability in Lemma 5.2 will not be changed. If $u \rightsquigarrow v$, then the constraints of Lemma 5.2 must have been satisfied both before and after adding the lambda node. So in both cases, the breakability of u will not change. □

So in the second phase, we will add lambda nodes from bottom to top. So we first gather all breakable nodes into a set denoted as S. During each iteration, we find the subset $U \subseteq S$ with maximum depth. We then either apply Section 3.2 by picking nodes in U or reject U as candidates and jump to the next lower lower depth. Thus, we need a probability p that a given breakable node will be selected. Eventually S will be empty. Here is the algorithm:

Algorithm 1: Redex selection during the second phase.

```
   Result: proofnet and LC-graph
1  $S$ = all breakable nodes;
2  while $S$ is not empty do
3      $MaxDepth$ = max depth of node in $S$;
4      $U = \{u | u \in S, u's\ depth = MaxDepth\}$;
5      $S = S \setminus U$;
6      while $U$ is not empty do
7          $u$ = pick node from $U$ uniformly;
8          $flag$ = Sample from Bernoulli($p$);
9          if flag==1 then
10             let $u$ be $x1$ and add lambda node $a$;
11             $U = U - u$;
12             if $a$ is breakable then
13                 $U = U + a$;
14             end
15         else
16             $U = U - u$;
17         end
18     end
19 end
```

The set U is finite, so lines 6–18 will eventually terminate. During each iteration of line 2–19, the size of S strictly decreases, so the entire algorithm eventually terminates.

At line 4, $\nexists u, v \in U$ such that $u \neq v$ and $u \rightsquigarrow v$. During lines 6–18, denote U^i as the U at line 6 and at iteration i. For all $u \in U^i$, all $v \in U^{i+1}, u \not\rightsquigarrow v$. Theorem 5.3 ensures that lines 9–17 will not change the breakability of the nodes in U^{i+1}. Also, if we denote U^i as the U in line 4, the nodes in U^{i+1} have smaller depth, so for all $u \in U$, all $v \in U^{i+1}, u \not\rightsquigarrow v$. Theorem 5.3 also ensures lines 3–18 will not change the breakability of the nodes in U^{i+1}.

Returning to our tiny corpus, during the second phase, $p1$ traverses lines 10–14 once and line 16 zero times, while $p2$ traverses lines 10–14 once and line 16 once. Therefore $p=\frac{2}{3}$. Thus:

$$\begin{aligned} P(p1) &= \tfrac{25}{256} \times \tfrac{2}{3} \times \tfrac{1}{2} = \tfrac{25}{768} \\ P(p2) &= \tfrac{125}{16384} \times \tfrac{2}{3} \times \tfrac{1}{3} \times \tfrac{1}{2} = \tfrac{125}{147456}. \end{aligned}$$

6 Experimental Investigation

6.1 Dataset

We trained and tested our probabilistic generative model on LCGbank (Fowler, 2016), a conversion of CCGbank (Hockenmaier and Steedman, 2007) to LCG. CCGbank is in turn a conversion based upon the Penn TreeBank (Marcus et al., 1993). We use LCGbank section 23, which contains 2416 proof nets as the test set. For the training set, we use sections 1-22 and 24 which contain 44870 proof nets. LCGbank uses the 5 primitives, ”S”, ”NP”, ”N”, ”conj” and ”PP”. The number of tokens is slightly different between CCGbank and PTB for the following two reasons:

- Some tokens like punctuation do not have categories in CCGbank. So we ignore those tokens in both LCGbank and PTB.
- Tokens like ”interest-rate” count as one token in CCGbank but count as three in PTB.

6.2 Comparison with Probabilistic Context Free Grammar

Since our model uses MLE, we also estimate the parameters for a PCFG using MLE on the PTB (using the same sections as the training and test set). Also, we exclude the lexica for both the PCFG and PLC. For the PCFG, the task is merely to generate sequences of POS tags, and for PLC, it is to generate sequents with the right-hand side of S.

Our model can generate every sequence in the test set because all primitives in the test set had been seen in the training set. Thus it can also assign a positive probability to every proof net in the test set. Some production rules in the PTB test set, however, never appear in the training set, and so PCFG fails to assign a non-zero probability to some sentences in the test set. Table1 shows that 260 sentences have zero probability in the resulting PCFG. On the other hand, the majority of sentences assigned a non-zero probability by both PLC and PCFG are assigned a lower probability by PLC than by PCFG. Table2 shows the number of cases (excluding zero-probability cases) that receive larger and smaller probabilities using PCFG. Table 3 shows the log probability of the entire test set (sum of log probability of each sentence) divided by the number of tokens. This evidence is consistent with the conclusion that PLC spreads probabilities more evenly across the seen and unseen category sequences induced by the set of primitives, at the expense of the data likelihood of the corpus.

	PCFG	PLC
$P \geq 0$	2144	2416
$P = 0$	272	0

Table 1: Number of sentences receiving positive and zero probabilities.

Positive	Negative
1964	180

Table 2: Number of sentences for which $p(PCFG) - p(PLC)$ is positive and negative, respectively.

PCFG	PLC
-2.95	-4.06

Table 3: Log probability of test set normalized by the number of tokens.

7 Conclusion and Future work

In this work, we have presented a probabilistic generative model for sequent derivability in the Lambek calculus. Any sequent that is derivable in the Lambek Calculus can be generated by our model. We also compared PLC with PCFG using MLE, both trained and tested on comparable corpora. The results show a trade-off to using PLC, in which the probabilities are more evenly distributed.

This probabilistic model may be used to parse with Lambek Grammar, although further investigation of early stopping when categories expand outside the coverage of the lexicon is still needed.

Another advantage of numerically parametrizing a grammar is to improve the speed of parsing, typically by forcing early failures. This is a direction that we have not yet pursued.

References

S. Bangalore and A. Joshi, editors. 2010. *Supertagging: Using Complex Lexical Descriptions in Natural Language Processing*. MIT Press.

G. Bonfante and P. de Groote. 2004. Stochastic lambek categorial grammars. *Electronic Notes in Theoretical Computer Science*, 53:34–40.

T. A. Fowler. 2016. *Lambek Categorial Grammars for Practical Parsing*. Ph.D. thesis, University of Toronto.

J. Hockenmaier and M. Steedman. 2007. CCGbank: A corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396.

T. Huang and K.S. Fu. 1971. On stochastic context-free languages. *Information Sciences*, 3(3):201–224.

S. Kübler, R. McDonald, and J. Nivre. 2009. *Dependency Parsing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool.

J. Lambek. 1958. The mathematics of sentence structure. *The American Mathematical Monthly*, 65(3):154–170.

K. Lari and S.J. Young. 1991. Applications of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech & Language*, 5(3):237–257.

M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

M. Osborne and T. Briscoe. 1997. Learning stochastic categorial grammars. In *Proc. CoNLL*, pages 80–87.

G. Penn. 2004. A graph-theoretic approach to sequent derivability in the lambek calculus. *Electronic Notes in Theoretical Computer Science*, 53:274–295.

M. Pentus. 1993. Lambek grammars are context free. In *[1993] Proceedings Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 429–433.

M. Pentus. 1997. Product-free lambek calculus and context-free grammars. *J. Symb. Log.*, 62:648–660.

D. Roorda. 1991. *Resource Logics: Proof-Theoretical Investigations*. Ph.D. thesis, Universiteit van Amsterdam.

German Verb Particle Constructions in CCG

Pablo Lopez Alonso
Department of Linguistics
Stony Brook University
Stony Brook, NY
al.pablo20@gmail.com

Abstract

As a theory of grammar, CCG (Steedman, 2000) is said to keep theory closely linked to psychological and computational mechanisms, to model how child or computer can learn any language (Steedman, 2017). In this paper, I test a Combinatory Categorial Grammar (CCG) against German Verb Particle Constructions (VPCs). Following previous work on English VPCs (Constable and Curran, 2009) and a German CCGbank (Hockenmaier, 2006), I analyze three types of German VPC constructions in CCG – main clauses, embedded clauses, and coordination clauses. Problems with modeling these sentences using CCG are discussed, and an alternative to deriving coordination sentences is presented in the Minimalist framework.

1 Introduction

An effective grammar formalism must represent the computational and psychological reality of natural language. To do so, it must be able to account elegantly and easily for natural language phenomena. Its mechanisms should correctly accommodate empirical data and make the correct predictions and rule out incorrect ones.

The central goal of this paper is to test Combinatory Categorial Grammar (CCG) (Steedman, 2000) against a set of compound verbs known as Verb Particle Constructions (VPCs), which present two possible orders:

(1) **Continuous Order**

a. The police **tracked down** the thief.

b. Anna **looked up** the book.

(2) **Discontinuous Order**

a. The police **tracked** the thief **down**.

b. Anna **looked** the book **up**.

VPCs have been extensively studied, but their syntactic status remains controversial (Dehé, 2015), (Haiden, 2017). They form a single semantic unit composed by two lexemes that show paradoxical behavior – they behave as both a word and a phrase. The analysis in this paper consists of two parts: first, we look at CCG's ability to deal with these verbs in German in simple main and embedded clauses. Then, we look at how effective this formalism is at dealing with a more complex structure, namely coordination. Crucially, we want CCG to deal with three types of VPC coordination – coordination of two different verb stems, coordination of one verb stem with two different particles, and dual valency coordination. This latter one involves a verb that is both a regular standalone verb and a verb that is part of a VPC.

The approach I follow here, borrows directly from Constable and Curran's (Constable and Curran, 2009) analysis of VPCs in CCGBank for English, as well as from Hockenmaier's (Hockenmaier, 2006) CCG Bank for German. I build on both approaches by adding Constable and Curran's new label for particles in VPCs to Hockenmaier's feature-rich German CCG to derive the correct German word order.

My proposal indicates that the approach used for English by Constable and Curran can account for some, but not all, German VPC sentences in CCG. Crucially, embedded sentences pose a challenge as VPCs in German have a different configuration in these constructions than they do in English. Furthermore, Across the Board Movement (ATB) in Minimalist Syntax is better able to capture dual valency VPC coordination constructions.

The paper proceeds as follows. Section 2 presents the preliminaries of German Syntax. Examples of VPCs in main and embedded clauses are presented, as well as those of coordination with VPCs with dual valency. Section 3 presents a brief

Proceedings of the 17th Meeting on the Mathematics of Language, pages 14–22, Montpellier (online),
December 13, 2021. © 2021 Association for Computational Linguistics

introduction to CCG, as well as Hockenmaier's CCG rules for a German CCGBank. Section 4 delves into the analysis of VPC constructions in CCG. First, I summarize Constable and Curran's approach for VPCs in English, then I present my own analysis combining their analysis with Hockenmaier's. This section contains the bulk of the paper, and presents the problematic cases of VPCs for the proposed German CCG. In the last section, I discuss how Minimalist Syntax is better equipped to deal with cases of dual valency coordination.

2 German Syntax

2.1 Word order

German has three different word orders depending on the sentence type. Main clauses are verb second (V2) (3), embedded and relative clauses are verb final (4), and interrogatives and imperatives are verb-initial (5) (examples from (Hockenmaier, 2006)).

(3) a. *Peter gibt Maria das Buch*
'Peter gives Mary the book.'
b. *ein Buch gibt Peter Maria.*
c. *dann gibt Peter Maria das Buch.*

(4) a. *dass Peter Maria das Buch gibt.*
b. *das Buch, das Peter Maria gibt.*

(5) a. *Gibt Peter Maria das Buch?*
b. *Gib Maria das Buch!*

For the purposes of this paper, we look only at VPCs in the context of V2 and verb-final sentences.

2.2 VPCs in German

VPCs in German must follow the same word order outlined in section 2.1:

(6) a. Ich stehe auf.
I stand PRT
'I stand up.'
b. Ich höre dir zu.
I hear you.DAT PRT
'I listen to you.'

In a main clause (6), the verb appears in its usual V2 position and the particle remains stranded in the final position. However, in an embedded clause (7), the verb and particle form one lexical unit and both appear in the final position.

(7) a. dass ich aufstehe.
that I PRT.stand
'that I stand up.'
b. dass ich dir zuhöre.
that I you.DAT PRT.hear
'that I listen to you.'

Embedded sentences, in particular, pose a challenge for dealing with VPCs in CCG as they display idiosyncratic behavior. I take this issue up in section 4.2.

3 CCG

3.1 Combinatory Categorial Grammar

Combinatory Categorial Grammar (Steedman, 2000) is a highly lexicalized grammar formalism. In CCG, every word is associated with a syntactic type made up of atomic categories and directional slashes. A category of the type X/Y is a functor that takes a type Y to the right and returns type X after application. CCG uses function application to combine different constituents; however, other types of functions are also available.

Application:

$$\begin{array}{llll} \sigma/\tau & \tau & \mapsto & \sigma \\ \tau & \sigma\backslash\tau & \mapsto & \sigma \end{array}$$

Composition (B):

$$\begin{array}{llll} \sigma/\tau & \tau/\rho & \mapsto & \sigma/\rho \\ \tau\backslash\rho & \sigma\backslash\tau & \mapsto & \sigma\backslash\rho \end{array}$$

Crossing Composition (B$_x$):

$$\begin{array}{llll} \sigma/\tau & \tau\backslash\rho & \mapsto & \sigma\backslash\rho \\ \tau/\rho & \sigma\backslash\tau & \mapsto & \sigma/\rho \end{array}$$

Type-Raising (T):

$$\begin{array}{lll} \tau & \mapsto & \sigma/(\sigma\backslash\tau) \\ \tau & \mapsto & \sigma\backslash(\sigma/\tau) \end{array}$$

As a theory of grammar, CCG keeps the syntax and semantics closely linked together. It also purports to be consistent with linguistic facts, keeping the theory as close as possible to computational and psychological mechanisms, allowing any language to be learned by both child and computer (Steedman, 2017).

CCG deals elegantly with certain linguistic phenomena like conjunction (Steedman and Baldridge, 2006). However, it has been noted that CCG Bank (Hockenmaier and Steedman, 2007), the main corpus for CCG-related work, which uses the formalism's combinatory rules, has several flaws. For example, it struggles to deal with complement/adjunct distinctions, compound nouns, and

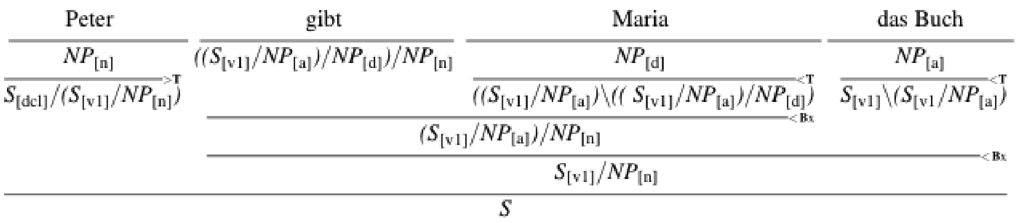


Figure 1: Standard main clause in Hockenmaier's German CCG

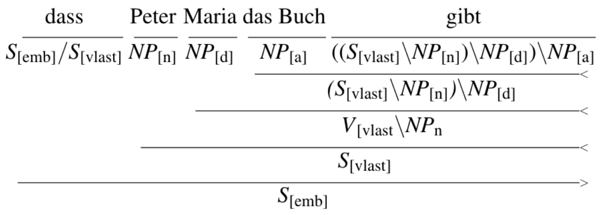


Figure 2: Embedded clause in Hockenmaier's German CCG

phrasal verbs or Verb Particle Constructions (Constable and Curran, 2009). I discuss this issue further in section 4.

3.2 A CCG for German

A first step to analyzing German VPCs in CCG is to capture the German word order. Hockenmaier (Hockenmaier, 2006) creates a CCG Bank for German, in which she advocates for augmenting CCG Bank with features to derive the correct word order (see Fig.1 and Fig.2 above). Features such as $S_{[v1]}$ and $S_{[vlast]}$ are introduced to capture the correct word order for V2 in main clauses and verb-final embedded clauses, for example. Other features such as $[n]$, $[a]$, and $[d]$ are introduced for nouns inflected for case.

Admittedly, contrary to Steedman (Steedman, 2000), Hockenmaier assumes that German is verb-initial (whence the $[v1]$ feature). This notion, however, runs counter to the literature on German verb headedness (Harbert, 2006), (Haider, 2010).

3.3 CCG Treatment of VPCs

CCGBank (Hockenmaier and Steedman, 2007), as mentioned in section 3.1, is the primary corpus for CCG-related work. It has been noted that it varies in its management of particles, but it tends to treat them as adverbial modifiers (Constable and Curran, 2009). This notion is problematic since particles are a core part of the VPC construction. Similarly, current German CCG tools do not seem to have a clear way to handle particles or VPCs (German CCG Bank, (Hockenmaier, 2006)) or they only deal with them in their sentence-final position (CCGWeb, (Evang et al., 2019)).

At least one attempt has been made to remedy this problem for English (Constable and Curran, 2009). Constable and Curran add a new atomic category *RP* to the existing set (*N, NP, S, PP*). This approach adds the particle directly into the verb's subcategorization and ensures the particle is a required part of the construction instead of an adverbial modifier. Admittedly, this model tends to over-generate, and it cannot rule out ungrammatical VPCs with a pronominal object in the split configuration (**she took away it*). Moreover, Constable and Curran observed a decrease in performance when parsing using this new mechanism. Nonetheless, the model is more consistent with linguistic facts than previous treatments of VPCs in CCG. As such, I take it up and use it to augment Hockenmaier's German CCG Bank rules in the next section.

My analysis is presented in contrast to Steedman's (personal communication, 2019) proposal that verbs such as *geben* 'give' and its VPC counterpart *zugeben* 'to admit' are "accidental homophones." Steedman assumes that these verbs are two distinct lexical units – a regular verb and a light-verb that combines with a particle. This proposal weakens the semantic connection between the two verbal constructions that intuitively we know exists (cf. *stehen* 'stand' v. *aufstehen* 'stand up').

4 German VPCs in CCG

In this section I analyze some VPC sentences in German CCG. First, I look at both intransitive and

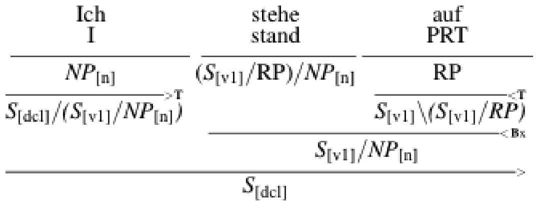


Figure 3: Main clause VPC sentence in German CCG

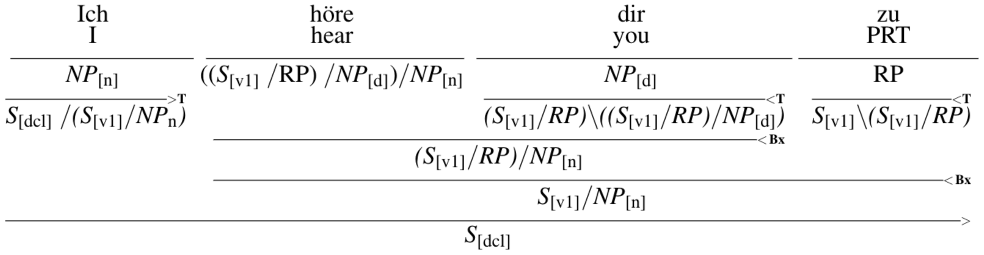

Ich
I
höre
hear
dir
you
zu
PRT
$NP_{[n]}$
$((S_{[v1]}/RP)/NP_{[d]})/NP_{[n]}$
$NP_{[d]}$
RP
>T
$S_{[dcl]}/(S_{[v1]}/NP_{n})$
<T
$(S_{[v1]}/RP)\backslash((S_{[v1]}/RP)/NP_{[d]})$
<T
$S_{[v1]}\backslash(S_{[v1]}/RP)$
< Bx
$(S_{[v1]}/RP)/NP_{[n]}$
< Bx
$S_{[v1]}/NP_{[n]}$
>
$S_{[dcl]}$

Figure 4: Transitive VPC construction in German CCG (dative object)

transitive VPCs in a main clause. Then, I move on to embedded sentences, and finally I look at examples of VPC coordination and verbs with dual valency.

4.1 Main Clauses

The combined approach I proposed in section 3.3, deals with intransitive sentences easily (Fig. 3). Following Hockenmaier's rules for German CCG Bank, the subject is type raised to be able to combine with the VPC. Similarly, here the Particle must type raise to combine with the verb. Unlike Constable and Curran (Constable and Curran, 2009), I have chosen to make the verb select for the particle first, and only then select for its other arguments. This decision was based on two facts: first, it allows us to more closely follow Hockenmaier's rules for German CCG derivations, and second, it captures the notion that the verb and particle form one unit despite their distance in the sentence.

This approach easily handles main clauses with transitive VPCs as well. Two examples are presented here, one with a dative object (Fig.4 above) and one with an accusative object (Fig.5 below).

Both examples follow the same template. First, the verb combines with its object (*accusative or dative*) through crossing composition, then with the particle through the same process. Lastly, the subject selects for the rest of the sentence, giving a declarative sentence as a result.

4.2 Embedded Clauses

Dealing with embedded VPC clauses in CCG presents the biggest challenge. As mentioned in section 2.2, VPCs in German show as a single lexical unit in the final position in embedded clauses. Since we know in a main clause the verb behaves as two separate lexical units, this leaves us with two options in CCG.

First, we could assume that the verb is one word only (Fig.6). In this approach we switch the functionality of the functor and make the verb select for the subject to its left. This is in line with what Hockenmaier (Hockenmaier, 2006) does for embedded clauses as well.

The second option is to split the verb into the particle and verb stem, much like we have done for the main clause (Fig. 7).

None of the previous approaches are without fault. The first approach requires that we stipulate that the main clause VPC and the embedded clause VPC have two different lexical categories. This is problematic as it increases the category ambiguity of the words by introducing a new category for each instance of the verb. In principle, CCG opposes this type of variation as it prefers to handle those differences by using combinatory rules.

The second approach allows us to keep the category of the verb consistent across main and embedded clauses by keeping the verb-particle split. However, in this instance, there is no mechanism that would prevent us from allowing adjuncts to come between the verb and the particle. It overgenerates

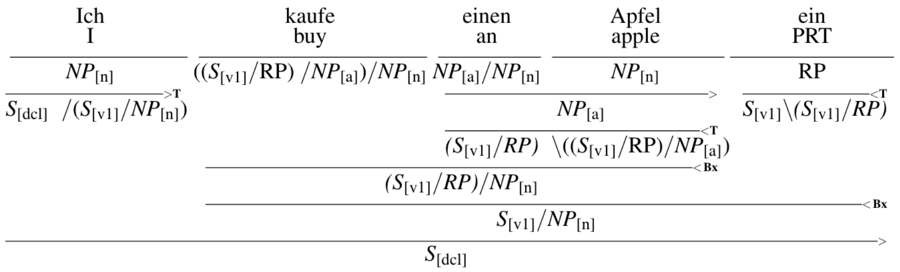


Figure 5: Transitive VPC construction in German CCG (accusative object)

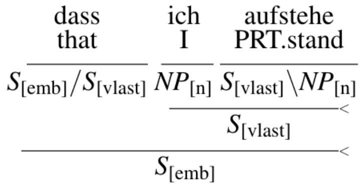


Figure 6: Embedded clause with VPC as a single lexical unit

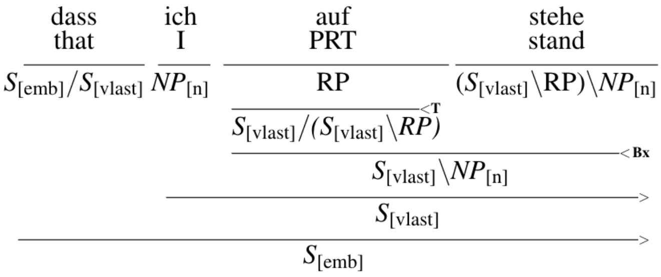


Figure 7: Embedded clause with VPC as a phrase

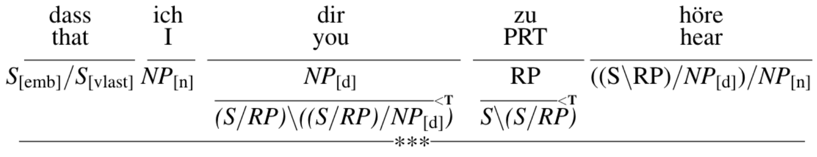


Figure 8: Failed CCG derivation for an embedded transitive VPC

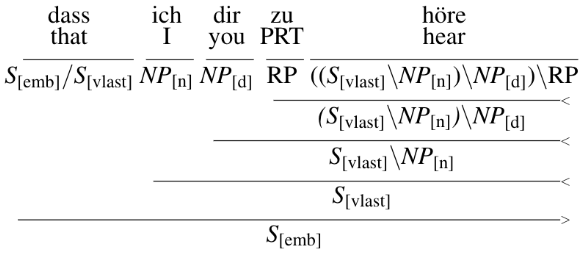


Figure 9: Fully derived VPC CCG derivation; verb category has changed

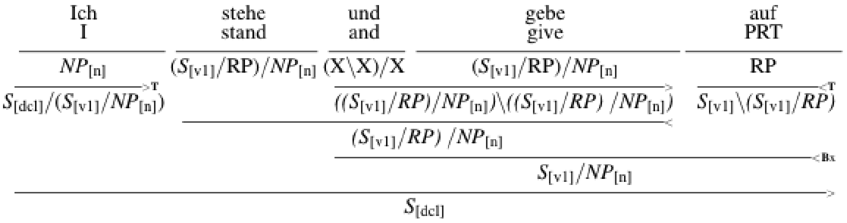


Figure 10: CCG derivation for particle sharing conjunction

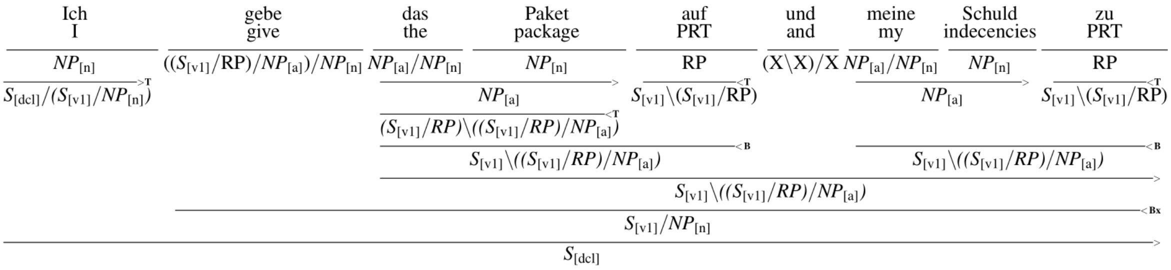


Figure 11: CCG derivation for verb sharing conjunction

for ungrammatical words like **aufgesternstehen* 'PRT-*yesterday*-stand'. At the moment, I do not have a solution for this problem, but if solved it would make CCG able to handle VPCs in embedded sentences.

Transitive VPCs in embedded clauses present even more of a challenge (Fig.8). In addition to the issue of whether the verb should be split from the particle or not, the verb has to select for an object.

It is impossible to derive this sentence and keep the verb category consistent with that of the main clause. Here again, changing the category assigned to the verb could give us a fully derived sentence. However, as mentioned above this is undesirable as it increases category ambiguity, thus assuming that the main clause verb and the embedded clause verb are two different lexical items. I present one possible derivation here for illustration purposes (Fig.9). Note, however, that this is an undesirable CCG derivation within our framework.

4.3 Coordination

VPC coordination presents even more of a challenge for CCG. Three types of sentences were analyzed for coordination – particle sharing conjunction, verb sharing conjunction, and dual valency conjunction. The first type of coordination involves two different verb stems in each conjunct associated with the same particle. The second type of coordination involves a verb stem that is associated with two different particles. The last type is a clause where the verb acts as both an intransitive and transitive across the conjuncts.

Particle sharing instances are easily handled (Fig.10). The verbs combine with one another first, then with the particle, and last with the subject. One elegant feature of this type of derivation is that the verb types remain the same as those for the regular main clause sentences, thus keeping the design simplified for the formalism.

Instances of verb sharing are handled much like Steedman and Baldridge (Steedman and Baldridge, 2006) handle some instances of Across The Board (ATB) movement (Fig. 11). First, we combine the two object conjuncts, and then we compose them with the verb. Last, by application we derive the full declarative sentence.

Once again, this example allows us to keep the category of the verb $((S_{[v1]}/RP)/NP_{[a]})/NP_{[n]}$ consistent with that of the main clause verb keeping the formalism simple.

Although CCG handles verb and particle sharing easily, it is unable to handle dual valency coordination at all (Fig.12). Crucially, the verb stem needs to select for an object for one conjunct, but for no object for the other. That is to say, the verb needs to be both transitive and intransitive. There is no formal mechanism to assign the verb two different categories.

As a transitive VPC, the verb must have the category $((S_{[v1]}/RP)/NP_{[a]})/NP_{[n]}$. However, as an intransitive verb, it needs the category $S_{[v1]}/RP/NP_{[n]}$. Since CCG deals with these surface variations only through combinatory rules,

Ich	gebe	auf	und	meine	Schuld	zu
I	give	PRT	and	my	indecencies	PRT
$NP_{[n]}$	$((S_{[v1]}/\mathrm{RP})/NP_{[a]})/NP_{[n]}$	RP	$(X\backslash X)/X$	$NP_{[a]}/NP_{[n]}$	$NP_{[n]}$	RP
			***			

Figure 12: Failed CCG derivation for dual valency conjunction

there is no way to assign both valencies to the verb, and the derivation is impossible.

As a grammar formalism, CCG is unable to handle VPCs in a cohesive and consistent manner. In this section I have highlighted only two problems that are readily apparent – embedded clauses and coordination. Embedded clauses present two problems. In their intransitive configuration the verb has to be categorized as a lexical unit with the particle or as a phrasal one. This problem is not unique to CCG and remains unresolved in the literature (Haiden, 2017; Dehé, 2015). However, in contrast to other analyses, both approaches are too powerful. The lexical approach is too restrictive and presents no mechanism for verb inflection. The phrasal approach is too permissive in that it allows for ungrammatical derivations of the verb. The second problem of German VPCs in CCG is that of coordination. Intransitive and transitive coordination fare decently under the formalism, however, dual valency coordination is impossible. The coordination problem receives a simpler analysis under the Minimalist Program (Chomsky, 1995). In the next section I discuss those analyses as an alternative to CCG.

5 VPC Coordination in Minimalism

Within the Minimalist Program (Chomsky, 1995), VPCs have received considerable attention (cf. (Haiden, 2017) for a general review and (Dehé, 2002) for VPCs in English). The syntactic status of VPCs as either complex heads or small clauses remains unresolved. Wurmbrand (Wurmbrand, 2000) provides an analysis that splits the complex head/small clause debate along semantic lines and which I follow here. The coordination problem, however, remains, to my knowledge, unadressed. As such, in this section, I make use of Wurmbrand's approach to show how Minimalism is better equipped to deal with the coordination cases mentioned in section 4.3.

Wurmbrand's analysis splits VPCs into transparent and idiomatic VPCs. Transparent VPCs receive a small clause treatment, while idiomatic ones receive a complex head treatment. This analysis easily accounts for different properties of VPCs in Germanic languages such as topicalization (Fig.13)

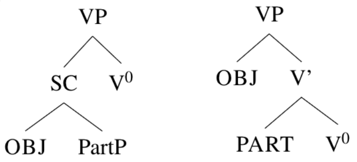


Figure 13: Transparent VPC (left) vs. (Semi-)idiomatic VPC structures (right)

The transparent/idiomatic distinction is not always clear-cut, however, and as such is largely ignored here. Note that for our purposes this distinction does not affect the analysis proposed here. Under Wurmbrand's analysis a main clause VPC looks as follows.

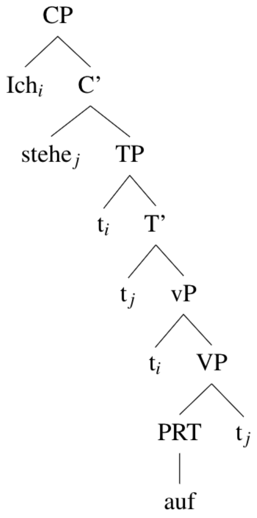


Figure 14: Minimalist derivation for *Ich stehe auf* 'I stand up'

I follow Haider (Haider, 2010) in assuming that no functional heads in German are head-final, only the verb is head-final. Canonically, the verb raises to the T head before reaching its final position in the C head for its V2 position in main clauses. The subject must then raise to Spec C in order to derive the correct word order. Crucially, the particle remains in its base position giving rise to particle stranding in German VPCs.

Coordination cases are easily accommodated by this analysis (Fig.15). For intransitive cases of particle sharing, the derivation starts with the VPC conjunction *aufgebe und aufstehe*. Through ATB movement the particle moves to a position above the coordination and the verbs must move out of the coordination to reach their V2 position as well. Just how this mechanism would fully work is out of the scope of this paper, but it would follow some type of sideward movement like that proposed in (Torr and Stabler, 2016).

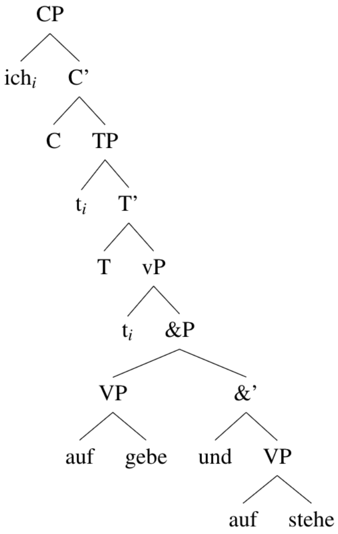


Figure 15: Verb stem coordination in Minimalism

Crucially, Minimalism has no difficulty in dealing with cases of dual valency mentioned in section 4.3.

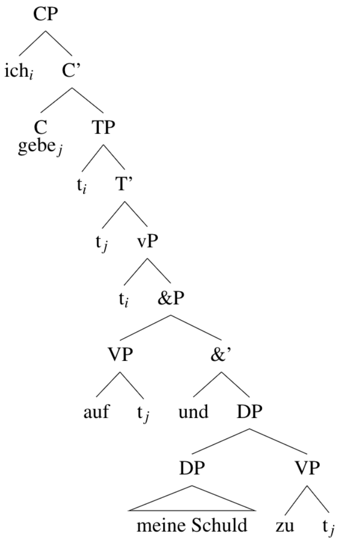


Figure 16: Dual valency coordination

The derivation starts with the conjunction of the two VPCs *aufgeben* and *meine Schuld zugeben*. Through ATB the verb follows its canonical movement to the T head and then to the C head to land in its V2 position. This operation results in the correct word order and the fully derived sentence. ATB in Minimalism avoids the problem CCG has of assigning two different categories to the same lexical item and thus creating ambiguity in the formalism. It remains to be seen if the minimalist approach would overgenerate, however, I leave that work for future research.

6 Conclusion

The account proposed in this paper draws from Constable and Curran (Constable and Curran, 2009) and Hockenmaier (Hockenmaier, 2006) to derive an analysis for German VPCs in CCG. The analysis presented here is able to accommodate VPCs in their main clause configuration, but fails at deriving the embedded clause counterparts. CCG is unable to capture distinctions such as the complex head/small clause analysis that has been proposed for VPCs in the literature. We are forced to choose one of the two analyses and the formalism becomes too powerful. Either it overgenerates for ungrammatical sentences under the small clause account or extremely restricts the verb under the complex head account, disallowing verbal inflection. In instances of embedded clauses with transitive VPCs, CCG seems completely unable to derive these clauses as the combinatory rules do not allow any possible combination of the constituents. Steedman's (p.c.) account for VPCs in CCG is equally unsatisfactory as it hinges on the idea that the verb in the VPC is a light-verb equivalent of a regular verb. At best, this analysis weakens a semantic connection between the verbs and at worst it completely denies it. Additionally, as a language formalism, CCG is unable to deal with examples of coordination such as dual valency.

While the question of the syntactic status of VPCs remains open, it is clear that as a theory of grammar, CCG offers no new insights to this matter. If we are to embrace this formalism as a theory of grammar that enables any child or computer to learn any language, further work needs to occur to refine the formalism in a manner that can better account for syntactic constructions such as the ones examined here.

References

Noam Chomsky. 1995. *The Minimalist Program*. MIT Press, Cambridge, Massachussets.

James Constable and James Curran. 2009. Integrating verb-particle constructions into CCG parsing. In *Proceedings of the Australasian Language Technology Association Workshop 2009*, pages 114–118, Sydney, Australia.

Nicole Dehé. 2002. *Particle Verbs in English: Syntax, Information Structure, and Intonation (Linguistik Aktuell/Linguistics Today 59)*.

Nicole Dehé. 2015. Particle verbs in germanic. In Peter O. Müller, editor, *Word-Formation : an International Handbook of the Languages of Europe ; vol. 1*, number 40,1 in Handbücher zur Sprach- und Kommunikationswissenschaft, pages 611–626. De Gruyter Mouton, Berlin; Boston.

Kilian Evang, Lasha Abzianidze, and Johan Bos. 2019. CCGweb: a new annotation tool and a first quadrilingual CCG treebank. In *Proceedings of the 13th Linguistic Annotation Workshop*, pages 37–42, Florence, Italy. Association for Computational Linguistics.

Martin Haiden. 2017. *Verb Particle Constructions*. The Wiley Blackwell Companion to Syntax.

Hubert Haider. 2010. *The Syntax of German*. Cambridge Syntax Guides. Cambridge University Press.

Wayne Harbert. 2006. *The Germanic Languages*. Cambridge Language Surveys. Cambridge University Press.

Julia Hockenmaier. 2006. Creating a CCGbank and a wide-coverage CCG lexicon for German. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 505–512, Sydney, Australia. Association for Computational Linguistics.

Julia Hockenmaier and Mark Steedman. 2007. CCGbank: A corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396.

M. Steedman and Jason Baldridge. 2006. Combinatory categorial grammar. *Encyclopedia of Language Linguistics*, 2.

Mark Steedman. 2000. *The Syntactic Process*. MIT Press, Cambridge, MA, USA.

Mark Steedman. 2017. Combinatory categorial grammar: An introduction.

John Torr and Edward P. Stabler. 2016. Coordination in Minimalist Grammars: Excorporation and across the board (head) movement. In *Proceedings of the 12th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+12)*, pages 1–17, Düsseldorf, Germany.

Susi Wurmbrand. 2000. The structure (s) of particle verbs. *Ms., McGill University*.

Strong learning of some Probabilistic Multiple Context-Free Grammars

Alexander Clark
Department of Philosophy, Linguistics and Theory of Science,
University of Gothenburg
alexsclark@gmail.com

Abstract

This paper presents an algorithm for strong learning of probabilistic multiple context free grammars from a positive sample of strings generated by the grammars. The algorithm is shown to be a consistent estimator for a class of well-nested grammars, given by explicit structural conditions on the underlying grammar, and for grammars in this class is guaranteed to converge to a grammar which is isomorphic to the original, not just one that generates the same set of strings.

1 Introduction

A fundamental theoretical goal of linguistics is to characterise a set of possible human languages that is sufficiently large and expressive to describe the attested natural languages while sufficiently restricted that it is learnable given information plausibly available to the child learner in the course of first language acquisition. There is a broad consensus. modulo some concerns about copying (Kobele, 2006), that a class of mildly context-sensitive grammars may be descriptively adequate (Stabler, 2013). Such grammars can generate a rich class of structural descriptions (trees) that serve as latent variables for the syntax/semantics interface, but the acquisition of such richer grammars is harder than the acquisition of simpler formalisms such as context-free grammars (CFGs). Yoshinaka (2009) showed how ideas of distributional learning for CFGs (Clark and Eyraud, 2007) could be extended to these mildly context-sensitive grammar formalism using multiple context free grammars (MCFGs) (Seki et al., 1991). However these algorithms, and their extensions (Yoshinaka, 2010) are only *weak* learning algorithms, that converge only to a grammar that is weakly equivalent to — generates the same set of strings as — the original target grammar. For the purposes of theoretical linguistics this is inadequate (Berwick et al., 2011), as we want a grammar that is strongly equivalent to — generates the same set of trees as — the original grammar. Moreover from a probabilistic perspective weakly equivalent grammars are insufficient (Scicluna and de la Higuera, 2016). Recently Clark and Fijalkow (2020) presented a strong learning algorithm for probabilistic context-free grammars (PCFGs), using only a sample of strings generated by the grammar (Horning, 1969). In this paper we extend that approach to a class of (probabilistic) MCFGs; or rather to several different classes. The approach of Clark and Fijalkow (2020) relies on the existence of what they call, following Stratos et al. (2016), *anchors*, which are terminal symbols that are unambiguously generated by a single nonterminal. On this assumption, the distribution of the terminal will be equal to the distribution of the nonterminal and with some ancillary technical conditions, this suffices to establish identifiability of the class of grammars and thence their learnability.

It is natural to extend this idea to the MCFG case, where some nonterminals generate not strings but tuples of strings; pairs of strings only in this paper. We can therefore define the anchoring condition for nonterminals which generate pairs of strings — of dimension 2 — as saying that there are a pair of terminals a, b which are generated only by a single production from a nonterminal of dimension 2.

However there is a significant technical problem to be overcome: the distribution of this pair may be much larger than the distribution of the nonterminal it anchors since we may have more than one occurrence of each symbol. For example if we have the string *aabb*, then we know that there must be two occurrences of the anchoring production in any derivation of this string but we don't know which pairs of *a* and *b* "match". We will state this problem more formally in section 3, but what is necessary is some way of restricting the

Proceedings of the 17th Meeting on the Mathematics of Language, pages 23–37, Montpellier (online),
December 13, 2021. © 2021 Association for Computational Linguistics

distribution of the substring in some way, and we will present a method that works with well-nested MCFGs (Kanazawa et al., 2011), Moreover the additional power of the formalism introduces new sorts of structural indeterminacy of the derivation trees which need to be handled carefully.

This is the first strong learning algorithm for these sorts of grammars and the goal of this paper is not to provide a complete solution to the problem but rather to identify the key difficulties and get an understanding of some of the ways in which these difficulties can be overcome. Nonetheless the class of grammars that can be learned, though descriptively inadequate in several important ways, have some interesting properties which we discuss at the end, and can represent the sorts of cross-serial dependencies that motivated the development of these mildly context-sensitive formalisms (Joshi, 1985).

2 Definitions

We assume some familiarity with MCFGs, or equivalent formalisms, and standard definitions of formal languages, as for example Yoshinaka et al. (2010). We assume a finite alphabet Σ; Σ^* is the set of finite strings of elements of Σ. We use λ for the empty string, since we need ϵ for another purpose. We write concatenation of two strings as uv and occasionally as $u \cdot v$ when we want to emphasize it. We will write $n(u; w)$ for the number of times u occurs in w, where $u, w \in \Sigma^*$, and we write $|w|$ for the length of a string.

We need to deal with various types of string; tuples of strings, like ordered pairs, and strings with gaps in. We will do this by using two additional symbols, $\Box$ which represents a gap, and $|$ which represents a boundary. We write $\mathbf{\Sigma}$ for $\Sigma \cup \{\Box, |\}$. Define $\mathbb{S}^k_g$ to be the set of strings with $k-1$ boundaries and g gaps.

$$\mathbb{S}^k_g = \{\mathbf{w} \in \mathbf{\Sigma}^* \mid n(\Box; w) = g, n(|; w) = k - 1\}$$

This represents a k-tuple of strings with g gap symbols occurring somewhere. Notationally we will use $\mathbf{w}$ for elements of $(\Sigma \cup \{\Box, |\})^*$ and w for elements of Σ^*. We will occasionally write the boundary symbol as a comma where there is no risk of confusion.

In this paper we consider $k \in \{1, 2\}$ and $g \in \{0, 1, 2\}$. The vast majority of the time we are just going to be using S^k_0 and S^1_k for $k \in \{1, 2\}$. These are k-tuples of strings, and contexts of k-tuples of strings; so things like u,$u|v$, $l\Box r$ and $l\Box m\Box r$.

The gap symbols represent variables, or spaces to be filled and so elements of $\mathbb{S}^k_g$ represent simple functions from $(\Sigma^*)^g \to (\Sigma^*)^k$. In particular a $\mathbb{S}^1_1$ like $l\Box r$ represents a function from strings to strings that we might write as $f(x) = lxr$, and an $\mathbb{S}^1_2$ like $l\Box m\Box r$ represents $f(x, y) = lxmyr$. As functions, $\Box$ and $\Box|\Box$ are the identities on strings and pairs of strings.

More generally and formally we can combine an element of $\mathbb{S}^i_j$ with one of $\mathbb{S}^j_k$ to get one of $\mathbb{S}^i_k$, defined by $\mathbf{u} \odot \mathbf{v}$ in the natural way by replacing all of the gap symbols in u with corresponding tuple elements of v; we lift this to sets in the natural way.

We now define the distribution of a set of k-tuples of strings, $\mathbf{U} \subseteq \mathbb{S}^k_0$, in a language L as

$$\mathbf{U}^{\triangleright} = \{\mathbf{c} \in \mathbb{S}^1_k \mid \mathbf{c} \odot_k \mathbf{U} \subseteq L\}$$

Conversely for $\mathbf{C} \subseteq \mathbb{S}^1_k$

$$\mathbf{C}^{\triangleleft} = \{\mathbf{u} \in \mathbb{S}^k_0 \mid \mathbf{C} \odot_k \mathbf{u} \subseteq L\}$$

For singleton sets $\{w\}$, we will just write $w^{\triangleright}$.

2.1 Grammars

We now define the class of grammars that we use, which are a restricted subclass of MCFGs (Seki et al., 1991) of dimension 2.

Definition 1. *An* MCFG, *a grammar, is a tuple* $\langle \Sigma, V, S, P \rangle$ *where* Σ *is an unranked finite set of terminal symbols,* V *is a finite nonempty ranked set of nonterminal symbols of ranks 1 or 2, which we call the dimension of the nonterminal. We write* $V^{(1)}$ *and* $V^{(2)}$ *for the sets of dimensions 1 and 2 respectively.* $S \in V^{(1)}$ *is a distinguished start symbol and* P *is a set of productions which we define below.*

We only consider non-deleting, non permuting, λ-free productions so we can use a slightly lighter notation than is normal. We have a countably infinite set of variables indexed by a pair of positive integers, $\mathcal{X} = \{x_{i,j} \mid i \in \mathbb{N}_+, j \in \{1, 2\}\}$. The variable $x_{i,j}$ refers to the jth component of the tuple generated by the ith nonterminal on the right hand side of the production. We will abbreviate $x_{1,j}$ as x_j and $x_{2,j}$ as y_j and x_1 as x and y_1 as y. A production, π, is a triple $\langle f, A, \alpha \rangle$ written $A \to f(\alpha)$, where $A \in V, \alpha \in V^*$ and f is a finite string of variables, terminals and boundary symbols. A is the left hand side of the production, α is the right hand side of the production, and f represents a function which constructs a tuple of strings from a tuple of tuple of strings, via substitution of the variables. If $\alpha = \lambda$ then we omit the brackets and write $A \to f$. In this case f is just a tuple of strings, which represents a leaf node in the derivation. The rank of a production is defined as $\operatorname{rank}(\pi) = |\alpha|$, and $\dim(\pi) = \dim(A)$.

We need to restrict the function f in various ways to make π a well-formed MCFG rule, and further to put it in a restricted normal form. We assume that terminals are introduced by special lexical or bilexical rules. Possible values of f are as follows:

- The rank is 0 and either $\dim(A) = 1$ and $f = a$ for some $a \in \Sigma$, or $\dim(A) = 2$ and $f = a|b$ for some $a, b \in \Sigma$. We call these lexical or bilexical rules, and we can write them as $A \to \mathbf{a}$.

- Alternatively the rank is nonzero, and f contains no terminal symbols. If $\dim(A) = 1$, then f contains only variables and if $\dim(A) = 2$ then it contains one occurrence of the boundary symbol. If there is a boundary symbol then it cannot occur at the beginning or end of the string (the production is λ-free). f contains exactly one occurrence each of variables corresponding to the nonterminals in α; So if $\alpha = B_1 \ldots B_k$ then for each i there is exactly one occurrence of the variable $x_{i,j}$ for $1 \leq j \leq \dim(B_i)$, and no other variables. Moreover if $\dim(B_i) = 2$ then $x_{i,1}$ occurs before $x_{i,2}$, (non-permuting) and if $i < j$, then $x_{i,1}$ occurs before $x_{j,1}$.

Given a particular f then, the rank is fixed, as are the dimensions of all nonterminals in the production. We will typically assume that the dimensions of all nonterminals are compatible with f. We say that a production $A \to f(B_1 \ldots B_k)$ is non-well-nested if there are two nonterminals on the right hand side B_i, B_j both of dimension 2, such that $f = \ldots x_{i,1} \ldots x_{j,1} \ldots x_{i,2} \ldots x_{j,2} \ldots$. It is well-nested if it is not non-well-nested. A grammar is well-nested if all of its productions are well-nested.

We will also use a Horn clause notation for productions (Kanazawa, 2009) when we want to discuss particular productions: For example, a CFG production like $A \to BC$ would be written in that notation as $A(xy) \leftarrow B(x)C(y)$, and would be formally the triple $A \to x_{1,1}x_{2,1}(BC)$. If the right hand side is empty then we write the production as $A(a)$ or $A(a, b)$ or $A(\mathbf{b})$. See Clark (2014) for an introduction to this notation and to MCFGs more generally. Other rules are for example the rank 1 production $A(x_1x_2) \leftarrow B(x_1, x_2)$; which takes a dimension 2 production, and concatenates the two components, and $A(x_1y_1, y_2x_2) \leftarrow B(x_1, x_2), C(y_1, y_2)$ which is a well-nested production which combines two dimension 2 nonterminals.

We assume further that S cannot appear on the right hand side of any productions, nor can it appear on the left hand side of any lexical productions. We do allow unary productions with S on the left hand side, $S(x) \to A(x)$; which are not allowed with any other nonterminal on the left hand side. Otherwise the only rank 1 production allowed is of the form $A(x_1x_2) \leftarrow B(x_1, x_2)$.

We define $\mathcal{F}_S$ and $\mathcal{F}$ to be the sets of possible well-nested functions for productions with S on the left hand side, and with some other nonterminal on the left hand side, respectively, that satisfy these conditions. If we restrict the functions to these sets then we define a normal form for the class of well-nested MCFGs of dimension 2; in the sense that for every language generated by a well-nested MCFG of dimension 2, there is a grammar in this class that generates the same language, that uses only productions with functions in these sets, as can be demonstrated using standard techniques (e.g. Kanazawa et al. (2011)), adapted from the standard approaches for converting CFGs into Chomsky normal form. Indeed although we allow in this paper productions of rank greater than 2, restricting productions to those of rank 2 does not change the set of languages generated as these well-nested grammars can be binarised.

2.2 Derivation trees

We will give a tree based semantics for this formalism, since we are interested in learning these trees and defining probability distributions over trees. A derivation tree for a grammar uses the set of productions as a ranked alphabet, where the rank of each production is equal to the number of nonterminals on the right hand side. For each nonterminal A we define an additional symbol $\square_A$ of rank 0. This symbol represents a hole or gap where an A is needed. The *sort* of a tree is the left hand side of the production labeling the root of the tree, or A if the tree is $\square_A$.

These trees must satisfy the condition that if π is a production of rank k, then the sort of the nth child of a node it labels must be equal to the nth nonterminal on the right hand side of π. Obviously if the production is of rank 0, then the node has no children and the condition is vacuous. Let $\mathbb{T}(G)$ be the set of all these trees. These include degenerate trees which just have a single node labeled with a rank 0 symbol; we won't use a special symbol for these. Given such a tree τ, we can count the number of occurrences of a label in the tree using the notation $n(\pi;\tau)$ or $n(\square_A;\tau)$.

Let $\Omega(G,A)$ be the set of such trees of sort A, and which have no occurrences of any gap symbols. Let $\Xi(G,A)$ be the set of such trees of sort S where there is exactly one occurrence of a gap symbol which is $\square_A$. We can combine an element ξ of $\Xi(G,A)$ with an element τ of $\Omega(G,A)$ to get an element of $\Omega(G,S)$ by replacing the single element of $\square_A$ in ξ with τ. We call the result $\xi \oplus \tau$. We will also lift this in the natural way to sets: for example $\Xi(G,A) \oplus \Omega(G,A)$ is the set of all derivation trees with at least one production of sort A in. Note that since S can only occur on the left hand side of a production, $\Xi(G,S)$ is a set consisting of a single tree with one node labeled $\square_S$.

We can map trees and derivation contexts into strings and string contexts using a string yield function (**sy**). A tree in $\Omega(G,A)$ has a string yield which is an element of $\mathbb{S}_0^{\dim(A)}$. A derivation context in $\Xi(G,A)$ has a string yield which is in $\mathbb{S}^1_{\dim(A)}$. The function string f represents a function: when we want to evaluate the function we will use the notation $\tilde{f}$; this occurs by substituting the variable $x_{i,j}$ for the jth component of the ith argument and concatenating the results. We can define the string yield function recursively bottom up as follows:

If A is of dimension 1, then $\mathbf{sy}(\square_A) = \square$ and if it is of dimension 2 then $\mathbf{sy}(\square_A) = \square|\square$. Otherwise: if the production is $A \to f(B_1,\ldots,B_k)$, then

$$\mathbf{sy}(\pi(\tau_1,\ldots,\tau_K)) = \tilde{f}(\mathbf{sy}(\tau_1),\ldots,\mathbf{sy}(\tau_K))$$

This also covers the case of a rank 0 production: the string yield of such a production $(A \to f)$ is just the constant f, which takes no arguments.

For a set $\mathbf{U} \subseteq \mathbb{S}_0^{\dim(A)}$, we define $\Omega(G,A,\mathbf{U}) = \{\tau \in \Omega(G,A) \mid \mathbf{sy}(\tau) \in \mathbf{U}\}$, and likewise for derivation contexts. Where appropriate we will omit G, and the set brackets so for example, $\Omega(A,w)$ is shorthand for $\{\tau \in \Omega(G,A) \mid \mathbf{sy}(\tau) = w\}$. Given the restrictions on the grammars we have imposed, $\Omega(G,A,w)$ is always finite.

Given this we define

$$\mathcal{L}(G,A) = \{\mathbf{sy}(\tau) \mid \tau \in \Omega(G,A)\}$$

and the set of contexts of a nonterminal as

$$C(G,A) = \{\mathbf{sy}(\xi) \mid \xi \in \Xi(G,A)\}.$$

The language defined by the grammar, $\mathcal{L}(G)$, is then $\mathcal{L}(G,S)$. Note of course that for every context $\mathbf{c} \in C(G,A)$ and every string $\mathbf{w} \in \mathcal{L}(G,A)$, $\mathbf{c} \odot \mathbf{w} \in \mathcal{L}(G)$.

We assume that every production occurs in at least one element of $\Omega(S)$. Under this assumption it's enough to give the list of productions to specify a MCFG, as the nonterminals and terminals can be read off the productions, and the start symbol is the unique nonterminal that occurs only on the left hand side of a production.

2.3 Examples

We give some trivial examples including one with cross-serial dependencies as seen in some natural languages (Huybrechts, 1984; Shieber, 1985)

Example 1. *$V^{(1)} = \{S\}$,$V^{(2)} = \{A\}$, $\Sigma = \{a,a',b,b'\}$ and we have productions: $\pi_A = A(a,a')$,$\pi_B = A(b,b')$, and*

$$\pi_S = S(x_1x_2) \leftarrow A(x_1,x_2)$$
$$\pi_X = A(x_1y_1,y_2x_2) \leftarrow A(x_1,x_2), A(y_1,y_2).$$

This defines the language

$$L_1 = \{aa', abbb'b'a', baa'b', \ldots\}$$

Note this grammar is ambiguous: $abbb'b'a'$ for example has two parse trees, as shown in fig. 1.

If we change π_X to

$$A(x_1y_1,y_2x_1) \leftarrow A(x_1,x_2), A(y_1,y_2)$$

this gives us a non-context-free language, with a non well-nested MCFG, L_{COPY}, which contains $\{aa', abba'b'b', aba'b', \ldots\}$. This is again ambiguous and the two trees above now both have the yield $abba'b'b'$.

Finally we give a well-nested grammar that generates a non context-free language.

Example 2. *$A(a,b)$,$C(c)$, $D(d)$, $E(e)$,$F(f)$,$C(c')$, $D(d')$, $E(e')$,$F(f')$ and*

$$S(x_1x_2) \leftarrow A(x_1,x_2)$$
$$A(x_1y,x_2z) \leftarrow A(x_1,x_2), C(y), E(z)$$
$$A(x_1y,x_2z) \leftarrow A(x_1,x_2), D(y), F(z)$$

Note this grammar is unambiguous, well nested, and generates a non-context-free language, L_{SWISS} related to the Swiss German example of Shieber (1985), which contains strings like $accd'beef$, of the form $aubv$ where u and v must contain matching sequences.

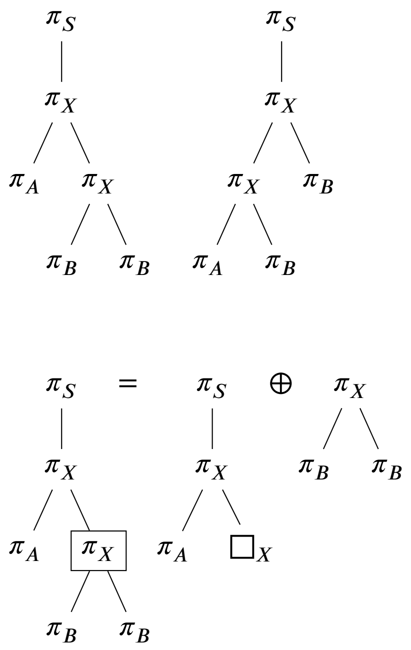


Figure 1: Sample derivation trees from example 1 and a decomposition. On the top we have two derivation trees both of which have yield $abbb'b'a'$. In the bottom we decompose the first of these trees at the boxed node into a context $\xi \in \Xi(G, X)$, where $\mathbf{sy}(\xi) = a\Box\Box a'$, and a tree $\tau \in \Omega(G, X)$ where $\mathbf{sy}(\tau) = bb|b'b'$. If we combine them we have $\mathbf{sy}(\xi \oplus \tau) = a\Box\Box a' \odot bb|bb' = abbb'b'a'$.

3 The problem

Clark and Fijalkow (2020) present a learning algorithm for PCFGs using distributional learning based on anchors. In the CFG case an anchor is a terminal which is derived uniquely from a single nonterminal: in the notation of this paper, a is an anchor for $A \in \Sigma^{(1)}$, if $A(a)$ is the unique production involving a in the grammar. In this case of course, the observable distribution of the terminal a is equal to the unobserved distribution of the nonterminal A:

$$a^{\rhd} = C(G, A)$$

Moreover for any context $\mathbf{c} \in a^{\rhd}$,

$$\Omega(S, \mathbf{c} \odot a) = \Xi(A, \mathbf{c}) \oplus A(a)$$

These two properties allow one to infer the parameters of productions, indeed the very existence of productions, using distributional analysis.

We can then naturally extend this to define an anchor for a nonterminal of dimension 2, A, as being a pair of distinct terminals (a, a') such that $A(a, a')$ is the only production using a or a'. But now neither of these two properties hold in general. For example, suppose we have the language L_{COPY}. If we let $\mathbf{a} = a, a'$ which is an anchor for A, and consider the string $w = aaa'a'$, then there are not 2 but 4 contexts $\mathbf{c}$ such that $\mathbf{c} \odot \mathbf{a} = w$, namely $\{\Box a\Box a', \Box aa'\Box, a\Box\Box a', a\Box a'\Box\}$ only two of which are in $C(G, A)$. So $\mathbf{a}^{\rhd} \supsetneq C(G, A)$. This is because we don't know which pairs of a, a' are being generated in the same production.

We will say that a context $\mathbf{c} \in \mathbf{a}^{\rhd}$, is an unambiguous context of $\mathbf{a}$, when it is a context of A (i.e. in $C(A)$) and additionally satisfies $\Omega(S, \mathbf{c} \odot \mathbf{a}) = \Xi(A, \mathbf{c}) \oplus A(\mathbf{a})$. For the approach to work, we need some way of restricting the observed contexts of 2-anchors to their unambiguous contexts. Note that all contexts of 1-anchors are unambiguous contexts in this sense.

4 Dyck contexts

Before we describe the key element of our solution, we need to recall the Dyck language (Ginsburg, 1966) and some of its properties. The Dyck language over the two letter alphabet p, q is defined by the unambiguous context-free grammar $S \to \lambda, S \to SpSq$; we write this language as $\mathcal{D}$. We say two occurrences of p and q match in an element of $\mathcal{D}$ if they are derived in the same step. Equivalently we can define the Dyck language as the set of all strings w in $\{p, q\}^*$ such that $n(p; w) = n(q; w)$ and where for every prefix v of w, $n(p; v) \geq n(q; v)$.

Suppose that we have a language L defined by an MCFG where the nonterminal A is anchored by a, a'. Then clearly in every string $w \in L$, we have that $n(a; w) = n(a'; w)$, since every time we introduce an a we also introduce an a'. Moreover, the occurrences of a will always precede the corresponding occurrences of a' since the productions are non permuting, so if u is a prefix of w then $n(a; u) \geq n(a'; u)$. It is therefore clear that in such a language, the occurrences of a, a' will form a subset of the Dyck language if we ignore all the other terminals. Let us formalise this precisely now.

For distinct terminals a, a' define the string homomorphism $h_{a,a'}$ via:

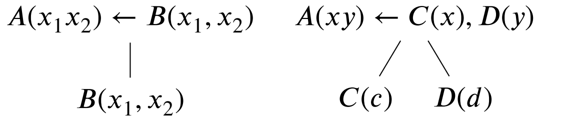
$$h_{a,a'}(b) = \begin{cases} p \text{ if } b = a \\ q \text{ if } b = a' \\ \lambda \text{ otherwise} \end{cases} \tag{1}$$

Definition 2. *A pair of distinct terminals (a, a') is a Dyck pair in a language L if for all $w \in L$, $h_{a,a'}(w) \in \mathcal{D}$.*

Lemma 1. *If A is anchored by a, a' in an* MCFG *defining the language L then a, a' is a Dyck pair.*

Consider L_1: here the nonterminal A is anchored by a, a' which is a Dyck pair; $h_{a,a'}(L_{\text{COPY}}) = \{p^n q^n \mid n \geq 0\} \subset \mathcal{D}$.

The converse is not true: it is not the case that in general the Dyck pairs will correspond to the anchors. There are two issues: first we might have productions like $A(xy) \leftarrow B(x)C(y)$ where B and C only generate b and c respectively, and only occur on the right hand side of this particular rule. A similar situation can occur with any production with two distinct dimension 1 nonterminals on the right hand side. Secondly, we might have the situation where we have two (or more) productions like $A(c, c')$ and $B(c, c')$ and no other productions using c or c'. In this case c, c' will form a Dyck pair but will clearly not be an anchor. This latter case can be handled exactly the way we handle ambiguity for dimension 1 nonterminals, but the former requires some additional assumptions. At its most fundamental the problem is this: suppose we have a language which consists only of the single length two string cd. We can represent this in two obvious ways.


$A(x_1x_2) \leftarrow B(x_1, x_2)$
$B(x_1, x_2)$
$A(xy) \leftarrow C(x), D(y)$
$C(c)$
$D(d)$

We need to resolve this structural indeterminacy in some way, if we want strong learning to be possible. There are a number of ways to proceed here: here we assume that every nonstart dimension 1 nonterminal must have two anchors. This is unnecessarily strong for this particular case, but it also serves to rule out some other problems later on.

Definition 3. *A grammar G is doubly anchored if for every $A \in V$ there are two terminals a, a' such that if $\dim(A) = 2$ then $A(a, a')$ is the only production using either, and if $\dim(A) = 1$ then $A(a)$ and $A(a')$ are the only productions using either.*

For instance, example 2 is doubly anchored.

Lemma 2. *Suppose G is a doubly anchored* MCFG*. If a, a' is a Dyck pair then a and a' can occur only in a production of the form $A(a, a')$.*

Proof. Suppose a occurred in a derivation tree τ using a production π of sort B. We can replace a single occurrence of π with π', a anchoring production of B that does not contain a to get a tree τ'. Since the yield of τ' must have matching numbers of a and a', π must be either $B(a, a')$ or $B(a', a)$. Suppose $\pi = B(a', a)$. This can't be the only production for this Dyck pair, or $h_{a,a'}$ would always start with a', so there is some anchor for π. Replace all other occurrences of π with some other anchor for B to obtain a tree τ'. Then $h_{a,a'}(\mathbf{sy}(\tau')) = qp \notin \mathcal{D}$, which is a contradiction. □

Let us suppose now that we have some a, a' which are anchors for a nonterminal A. We can say that an occurrence of a and a' in a string $w = lama'r$ match if there is a derivation of w where they are generated by the same production $A(a, a')$; in other words if their context $l\square m\square r \in C(G, A)$. Now we cannot in general determine in such a string the matching pairs as discussed above. But if the grammar is *well-nested* then the matching is always determined, even if the string is ambiguous; indeed they are generated by the same production iff their images match in the Dyck language.

Let PAIRS(L) be the set of Dyck pairs of the language L and define

$$\mathcal{D}_L = \{\mathbf{w} \in \Sigma^* \mid \forall \mathbf{a} \in \text{PAIRS}(L), h_{\mathbf{a}}(\mathbf{w}) \in \mathcal{D}\}$$

Clearly $L \subseteq \mathcal{D}_L$. We define the following four sets:

$$\begin{aligned}
\mathbb{D}^1_0 &= \mathcal{D}_L \\
\mathbb{D}^2_0 &= \{u|v \in \mathbb{S}^2_0 \mid u \cdot v \in \mathcal{D}_L\} \\
\mathbb{D}^1_1 &= \{l\square r \in \mathbb{S}^1_1 \mid l \cdot r \in \mathcal{D}_L\} \\
\mathbb{D}^1_2 &= \{l\square m\square r \in \mathbb{S}^1_2 \mid m \in \mathcal{D}_L, l \cdot r \in \mathcal{D}_L\}
\end{aligned}$$

Note that by the properties of the Dyck language, $\mathbb{D}^1_2 \odot \mathbb{D}^2_0 \subseteq \mathcal{D}_L$, and $\mathbb{D}^1_1 \odot \mathbb{D}^1_0 \subseteq \mathcal{D}_L$. The crucial observation here is the following lemma, which means that for these grammars it is enough to consider the subsets $\mathbb{D}$ and not the whole sets $\mathbb{S}$.

Lemma 3. *If G is well-nested and double anchored, then for all nonterminals $A \in V^{(k)}$, $\mathcal{L}(G, A) \subseteq \mathbb{D}^k_0$ and $C(G, A) \subseteq \mathbb{D}^1_k$.*

Proof. By induction on $\Omega(G, A)$ in the first case, and in the second case on $\Xi(G, A)$ based on the path from the root of the tree to the gap symbol, $\square_A$. □

Lemma 4. *If G is well-nested and double anchored, and $\mathbf{a}$ is an anchor for A, then $\mathbf{a}^{\triangleright} \cap \mathbb{D}^1_{\dim(A)} = C(G, A)$, and all of these contexts are unambiguous contexts.*

We now define for $\mathbf{U} \subseteq \mathbb{S}_0^k$

$$\mathbf{U}^{\blacktriangleright} = \mathbf{U}^{\triangleright} \cap \mathbb{D}_k^1$$

and for $\mathbf{C} \subseteq \mathbb{S}_k^1$

$$\mathbf{C}^{\blacktriangleleft} = \mathbf{C}^{\triangleleft} \cap \mathbb{D}_0^k$$

These form a pair of Galois connections between $\mathbb{D}_0^k$ and $\mathbb{D}_k^1$, which obey the usual identities, which we use below. So for example $\mathbf{U}^{\blacktriangleright} = \mathbf{U}^{\blacktriangleright\blacktriangleleft\blacktriangleright}$.

Crucially, these Galois connections also satisfy a fundamental lemma, analogous to the CFG condition that for all sets of strings X, Y $(X \cdot Y)^{\triangleright\triangleleft} = (X^{\triangleright\triangleleft} \cdot Y^{\triangleright\triangleleft})^{\triangleright\triangleleft}$.

Lemma 5. *Suppose f is a well-nested function of rank r, and let L be some language. Then for all $\mathbf{X}_i \in \mathbb{D}_0^{k_i}$ which are sets of tuples of strings of suitable dimension,*

$$f(\mathbf{X}_1, \dots, \mathbf{X}_r)^{\blacktriangleright\blacktriangleleft} = f(\mathbf{X}_1^{\blacktriangleright\blacktriangleleft}, \dots, \mathbf{X}_r^{\blacktriangleright\blacktriangleleft})^{\blacktriangleright\blacktriangleleft}$$

Proof. Clearly the left is a subset of the right, since $\mathbf{X}_i \subseteq \mathbf{X}_i^{\blacktriangleright\blacktriangleleft}$. Let d_i be the arity of X_i. Suppose $\mathbf{c} \in f(\mathbf{X}_1, \dots, \mathbf{X}_r)^{\blacktriangleright}$; if not then it will trivially true.

This means that $\mathbf{c} \odot f(\mathbf{X}_1, \dots, \mathbf{X}_r) \subseteq L$. So $\mathbf{c} \odot f(\square_{d_1}, \dots, \mathbf{X}_r) \subseteq \mathbf{X}_1^{\blacktriangleright}$, since f is well-nested. And $\mathbf{X}_1^{\blacktriangleright} = \mathbf{X}_1^{\blacktriangleright\blacktriangleleft\blacktriangleright}$. So $\mathbf{c} \odot f(\square_{d_1}, \dots, \mathbf{X}_r) \subseteq \mathbf{X}_1^{\blacktriangleright\blacktriangleleft\blacktriangleright}$. So $\mathbf{c} \odot f(\mathbf{X}_1^{\blacktriangleright\blacktriangleleft}, \dots, \mathbf{X}_r) \subseteq L$. Therefore $\mathbf{c} \in f(\mathbf{X}_1^{\blacktriangleright\blacktriangleleft}, \dots, \mathbf{X}_r)^{\blacktriangleright}$. Repeating for X_2 etc.:

$$f(\mathbf{X}_1, \dots, \mathbf{X}_r)^{\blacktriangleright} \subseteq f(\mathbf{X}_1^{\blacktriangleright\blacktriangleleft}, \dots, \mathbf{X}_r^{\blacktriangleright\blacktriangleleft})^{\blacktriangleright}$$

Therefore

$$f(\mathbf{X}_1, \dots, \mathbf{X}_r)^{\blacktriangleright\blacktriangleleft} \supseteq f(\mathbf{X}_1^{\blacktriangleright\blacktriangleleft}, \dots, \mathbf{X}_r^{\blacktriangleright\blacktriangleleft})^{\blacktriangleright\blacktriangleleft}$$

□

This restricted distribution is well-behaved so we can redefine the standard notion of validity to use this. Since S cannot be anchored, and to avoid handling it as a special case, we can stipulate that we have a special terminal symbol s where $s^{\triangleright} = \{\square\}$, which is of course equal to $C(G, S)$, since the start symbol can occur only at the root of a derivation tree.

Definition 4. *Suppose we have a well nested doubly anchored grammar, where $A, B_1, \dots B_k$ are nonterminals anchored by $\mathbf{a}, \mathbf{b_1}, \dots \mathbf{b_k}$, respectively. Let $\pi = A \to f(B_1, \dots, B_k)$ be a production we say that it is valid if $\mathbf{a}^{\blacktriangleright} \subseteq \tilde{f}(\mathbf{b_1}, \dots, \mathbf{b_k})^{\blacktriangleright}$.*

Note that this is now well-defined (independent of the choice of anchors) by lemma 5. Clearly all productions in the grammar are valid, since $\tilde{f}(\mathbf{b_1}, \dots, \mathbf{b_k}) \in \mathcal{L}(G, A)$. We can also show something that is approximately the converse.

Lemma 6. *Suppose $G = \langle \Sigma, N, S, P \rangle$ generates the language L; Let $G' = \langle \Sigma, N, S, P' \rangle$ and where $P' \supseteq P$, and all of $\pi \in P'$ are valid with respect to L.*

If $\mathbf{a}$ is an anchor of A, then $\mathcal{L}(G', A) \subseteq \mathbf{a}^{\blacktriangleright\blacktriangleleft}$.

Note that this then implies that $\mathcal{L}(G') = \mathcal{L}(G)$, since $s^{\blacktriangleright\blacktriangleleft} = L$.

Proof. Proof by induction on the derivation trees in $\Omega(G', A)$.

Base case:suppose $\tau = \pi = A(\mathbf{b})$, so $\mathbf{sy}(\tau) = \mathbf{b}$. Then since this production is valid $\mathbf{a}^{\blacktriangleright} \subseteq \mathbf{b}^{\blacktriangleright}$ and so $\mathbf{a}^{\blacktriangleright\blacktriangleleft} \supseteq \mathbf{b}^{\blacktriangleright\blacktriangleleft}$, and since $\mathbf{b} \in \mathbf{b}^{\blacktriangleright\blacktriangleleft}$, $\mathbf{b} \in \mathbf{a}^{\blacktriangleright\blacktriangleleft}$.

Now do inductive step: suppose topmost production is $\pi = A \to f(B_1, \dots B_k)$ and $\tau = \pi(\tau_1, \dots, \tau_k)$. By the inductive hypothesis $\mathbf{sy}(\tau_i) \in \mathbf{b}_i^{\blacktriangleright\blacktriangleleft}$. So $\mathbf{sy}(\tau) \in f(\mathbf{b}_1^{\blacktriangleright\blacktriangleleft}, \dots, \mathbf{b}_k^{\blacktriangleright\blacktriangleleft})$, and so $\mathbf{sy}(\tau) \in f(\mathbf{b}_1^{\blacktriangleright\blacktriangleleft}, \dots, \mathbf{b}_k^{\blacktriangleright\blacktriangleleft})^{\blacktriangleright\blacktriangleleft}$

By lemma 5, $\mathbf{sy}(\tau) \in f(\mathbf{b}_1, \dots, \mathbf{b}_k)^{\blacktriangleright\blacktriangleleft}$. So $\mathbf{sy}(\tau)^{\blacktriangleright} \supseteq f(\mathbf{b}_1, \dots, \mathbf{b}_k)^{\blacktriangleright}$. Validity of π says that $\mathbf{a}^{\blacktriangleright} \subseteq f(\mathbf{b_1}, \dots, \mathbf{b_k})^{\blacktriangleright}$. Therefore $\mathbf{sy}(\tau)^{\blacktriangleright} \supseteq \mathbf{a}^{\blacktriangleright}$. So $\mathbf{sy}(\tau)^{\blacktriangleright\blacktriangleleft} \subseteq \mathbf{a}^{\blacktriangleright\blacktriangleleft}$, and $\mathbf{sy}(\tau) \in \mathbf{a}^{\blacktriangleright\blacktriangleleft}$. □

5 Other conditions

Beyond the anchoring condition we need two additional conditions which are essentially the same as in Clark (2021a); the first is a bound on the ambiguity, which will allow for identifiability of the parameters.

Definition 5. *A MCFG G is locally unambiguous (LUA) if for every production π in the grammar of the form $A \to f(B_1 \dots B_r)$, there is a string w in the language which can be decomposed into a $\mathbf{c} \in \mathbb{D}^1_{\dim(A)}$, and a sequence of tuples $\mathbf{u}_1, \dots \mathbf{u}_r$, so $\mathbf{c} \odot_k \tilde{f}(\mathbf{u}_1, \dots \mathbf{u}_r)) = w$, such that:*

$$\Omega(S, w) = \Xi(A, \mathbf{c}) \oplus \pi(\Omega(B_1, \mathbf{u_1}), \dots, \Omega(B_r, \mathbf{u_r}))$$

Note that this is a generalisation of the definition of an unambiguous context earlier: If $A(\mathbf{a})$ is an anchoring production then the unambiguous contexts are just the ones that satisfy this LUA condition.

The second condition is based on lemma 6; as a result of this lemma, we can see that adding valid productions to a grammar will not increase the set of strings generated. This means that we can stipulate that the grammar must contain all valid productions. However this may lead to excessive ambiguity, which will then violate the LUA condition. Accordingly we want to eliminate "redundant" productions.

We can combine two productions into another: $\pi_1 = A \to f(\alpha B \gamma)$ and $\pi_2 = B \to g(\beta)$ can be combined to get $\pi_3 = A \to h(\alpha\beta\gamma)$ where h is the suitable function formed from f and g; we will skip the formal definition since it is obvious but fiddly since we need to renumber the variables. Clearly $\text{rank}(\pi_3) = \text{rank}(\pi_2) + \text{rank}(\pi_1) - 1$. So for example $A(x_1x_2) \leftarrow B(x_1, x_2)$ and $B(x_1y, x_2) \leftarrow C(x_1, x_2), D(y)$ can be combined to $A(x_1yx_2) \leftarrow C(x_1, x_2), D(y)$. Intuitively, if we already have the first two, we don't also need the third.

We fix a maximal rank r and define $\mathcal{F}$ and $\mathcal{F}_S$ to be all well-nested productions of rank at most r. We order the productions so that if $\text{rank}(f_1) < \text{rank}(f_2)$ then f_1 precedes f_2, and such that dimension 2 functions precede dimension 1 functions. For the case of $r = 2$, we give the explicit sets:

$$\begin{aligned}\mathcal{F} = \langle & x_1x_2, x|y, x|y_1y_2, xy_1|y_2, x_1|x_2y, x_1|yx_2,\\ & x_1x_2|y, x_1y|x_2, x_1|x_2y_1y_2, x_1|y_1y_2x_2, x_1x_2|y_1y_2,\\ & x_1y_1|y_2x_2, x_1x_2y_1|y_2, x_1y_1y_2|x_2, xy, x_1x_2y,\\ & x_1yx_2, xy_1y_2, x_1x_2y_1y_2, x_1y_1y_2x_2, \rangle\end{aligned}$$

and

$$\begin{aligned}\mathcal{F}_S = \langle & x, x_1x_2, xy, xy_1y_2, x_1x_2y, x_1yx_2,\\ & x_1x_2y_1y_2, x_1y_1y_2x_2\rangle\end{aligned}$$

Note that these are ordered so that if we compose two productions to form a third, the first two will precede the others in these lists. We need to define the notion of a possible production now: given an anchored grammar G with alphabet Σ and nonterminals V, we define $P(G, \mathcal{F}, \mathcal{F}_S)$ to be the union of these sets:

$$\begin{aligned}\{A(b) \mid A \in V^{(1)} \setminus \{S\}, b \in \Sigma\}\\ \{A(b, c) \mid A \in V^{(2)}, b, c \in \Sigma\}\\ \{A \to f(B_1, \dots B_k) \mid A, B_i \in V \setminus \{S\}, f \in \mathcal{F}\}\\ \{S \to f(B_1, \dots B_k) \mid B_i \in V \setminus \{S\}, f \in \mathcal{F}_S\}\end{aligned}$$

where the dimensions of the nonterminals in the last two are restricted to those matching the dimensions of f.

Definition 6. *Given an anchored* MCFG *G, a valid production is redundant wrt $\mathcal{F}, \mathcal{F}_S$ if it is the composition of two other valid productions in $P(G, \mathcal{F}, \mathcal{F}_S)$.*

Definition 7. *(Clark, 2021a) An* MCFG *is complete wrt $\mathcal{F}, \mathcal{F}_S$ if the set of productions is those in $P(G, \mathcal{F}, \mathcal{F}_S)$ that are valid and not redundant.*

This definition then means that the set of productions is determined by the language given the anchored nonterminals. Together these definitions mean that the grammar itself, the nonterminals and the set or productions and of course the terminals are determined by the language, which we prove in the next section.

5.1 Identifiability of Well-Nested Class

We define the class of grammars $\mathfrak{G}_r$ to be all doubly anchored MCFGs where the functions are well nested of rank at most r and which are complete.

Theorem 1. *Suppose G and G' are in $\mathfrak{G}_r$ such that $\mathcal{L}(G) = \mathcal{L}(G')$. Then G is isomorphic to G'.*

Proof. Let $L = \mathcal{L}(G)$. Suppose that $\mathbf{a}$ is a Dyck pair; and $A(\mathbf{a})$ is in the grammar. Let $\mathbf{b}$ be an anchor for A. Then $\mathbf{b}^{\blacktriangleright} \subseteq \mathbf{a}^{\blacktriangleright}$. Therefore consider the set of distributions $\{\mathbf{a}^{\blacktriangleright} \mid \mathbf{a} \text{ is a Dyck pair}\}$. The minimal elements of this ordered by set inclusion will correspond to the dimension 2 nonterminals via $C(G, A) = \mathbf{a}^{\blacktriangleright}$; by completeness we cannot have more than one nonterminal for each of these minimal elements. Consider now the set of bilexical productions: by completeness, these must consist of exactly those productions $A(\mathbf{b})$, where if $\mathbf{a}$ is an anchor for A, $\mathbf{a}^{\blacktriangleright} \subseteq \mathbf{b}^{\blacktriangleright}$. Therefore the set of bilexical productions is defined by L. Consider now the set of all terminals, T, that do not occur in any bilexical productions. Consider the set of distributions $\{a^{\triangleright} \mid a \in T\}$. The minimal elements of this set must correspond to the nonterminals of dimension 1, via the correspondance $a^{\triangleright} = C(A)$. Again the lexical productions must be exactly the set of all valid productions; namely those where $a^{\triangleright} \subseteq b^{\triangleright}$, where a is an anchor for A.

There is therefore a bijection ϕ between the nonterminals of G and G' where $\phi(A) = A'$ if $C(G, A) = C(G', A')$.

It remains to be shown that the set of productions is uniquely determined given these productions. The set of valid productions is clearly determined by the language; the only issue then is to verify that given this set there is a unique set of non-redundant productions. If a production π is formed by the composition of two productions π_1

and π_2 then we can see that functions of π_1 and π_2 will strictly precede the function of π. This ensures uniqueness.

□

6 Weighted MCFGs and Probabilistic MCFGs

The learning paradigm we use is probabilistic so we start by defining the probabilistic grammars that we use. A stochastic language is a probability distribution over Σ^*, wrtten $\mathbb{P}$, and this will be defined by a probabilistic MCFG, (Levy, 2005; Kato et al., 2006; Kallmeyer and Maier, 2013) which we define via a slightly more general formalism called weighted MCFGs. We use two different parameterisations, the top-down which corresponds to the stochastic MCFGs studied by Kato et al. (2006) and the bottom-up introduced in the CFG case by Clark and Fijalkow (2020) which is useful from a learnability perspective. The presentation here follows that in that paper.

Definition 8. *A weighted* MCFG $\langle G; \theta \rangle$ *is a* MCFG *together with a parameter function θ, from productions to positive real numbers.*

$$\theta : P \to \mathbb{R}^+.$$

We define a weight function $w : \mathbb{T}(P) \to \mathbb{R}^+$. For each such tree τ we define:

$$w(\tau) = \prod_{\pi \in P} \theta(\pi)^{n(\pi;\tau)}$$

Note that for $\xi \in \Xi(G, A)$ and $\tau \in \Omega(G, A)$,

$$w(\xi \oplus \tau) = w(\xi)w(\tau)$$

The weight of a set of trees Ω, is $w(\Omega) = \sum_{\tau \in \Omega} w(\tau)$. For a nonterminal A define two quantities which are normalization constants, the sums of the inside and outside probabilities to use the CFG terminology (Eisner, 2016):

$$I(A) = w(\Omega(G, A))$$
$$O(A) = w(\Xi(G, A))$$

The quantity $I(S)$ is called the partition function in the case of PCFGs (Nederhof and Satta, 2008). If $I(S) = 1$ this then defines a probability distribution over $\Omega(G, S)$ and via that a stochastic language whose support is $\mathcal{L}(G)$, by summing over all derivation trees with a given yield as follows:

$$w(u) = \sum_{\tau \in \Omega(G,S):\mathbf{sy}(\tau)=w} w(\tau) = w(\Omega(G, S, \{w\})) \tag{2}$$

Given such a weighted MCFG, with $I(S) = 1$, for every production π, we can define $\mathbb{E}(\pi)$ the expected number of times that the production π occurs in a tree. This is

$$\mathbb{E}(\pi) = \sum_{\tau \in \Omega(G)} w(\tau)n(\pi; \tau) \tag{3}$$

We can define the expectation of a nonterminal as the sum of expectations of all productions with that nonterminal on the left hand side.

$$\mathbb{E}(A) = \sum_{A \to f(\alpha) \in P} \mathbb{E}(A \to f(\alpha)) \tag{4}$$

We assume throughout that the expected length of strings under these distributions, $\sum_{a \in \Sigma} \mathbb{E}(a)$, is finite, which implies that all these expectations are finite too. We can see that for all nonterminals A,

$$I(A)O(A) = \mathbb{E}(A)$$

Note that $\mathbb{E}(S) = 1$ under these assumptions, as it occurs exactly once in every tree in $\Omega(G, S)$, at the root; indeed clearly $I(S) = O(S) = 1$, since $\Xi(G, A)$ has one element, $\square_S$, which has weight 1.

There is an indeterminacy in these parameters in that one can always pick some nonterminal that isn't S, say A, and scale the parameters of productions with $A \to f(B_1, \ldots, B_r)$ on the left hand side by some $\alpha > 0$, and scale all productions with A on the right hand side $B \to f(C_1, \ldots, C_r)$ by α^{-n} where n is the number of times A occurs on the right hand side, and the distribution over trees will remain unchanged. There are two natural ways of resolving this, two parameterisations: one where we stipulate that for all A, $O(A) = 1$ (and as a result $I(A) = \mathbb{E}(A)$) and one where we stipulate that for all A, $I(A) = 1$, and therefore $O(A) = \mathbb{E}(A)$. The latter gives us the regular probabilistic top down generative process. The former gives us the bottom up parameterisation, which is crucial to the success of these primal distributional learning algorithms.

Note that for any production $\pi = A \to f(B_1, \ldots, B_r)$

$$\mathbb{E}(\pi) = O(A)\theta(\pi) \prod_i I(B_i)$$

Therefore in the bottom up parameterization, where $O(A) = 1$ and $I(A) = \mathbb{E}(A)$ this gives us

$$\mathbb{E}(\pi) = \theta(\pi) \prod_i \mathbb{E}(B_i)$$

and so

$$\theta(\pi) = \frac{\mathbb{E}(\pi)}{\prod_i \mathbb{E}(B_i)} \tag{5}$$

6.1 Anchored grammars with bottom-up parameters

Given a stochastic language $\mathbb{P}$, for a terminal a we define $\mathbb{E}(a) = \sum_{\mathbf{c}\in a^{\triangleright}} \mathbb{P}(\mathbf{c} \odot a)$. This is the expected number of times the terminal a will appear in a random string. Now suppose that $\mathbf{a} = a, a'$ is a Dyck pair; clearly $\mathbb{E}(a) = \mathbb{E}(a')$ so we define $\mathbb{E}(\mathbf{a}) = \mathbb{E}(a)$.

Then if $\mathbf{a}$ is an anchor for A then in the bottom up parameterisation

$$\theta(A(\mathbf{a})) = \mathbb{E}(\mathbf{a})$$

Suppose a context $\mathbf{c} \in \mathbf{a}^{\blacktriangleright}$. Then

$$\Omega(G, S, \mathbf{c} \odot \mathbf{a}) = \Xi(G, A, \mathbf{c}) \oplus A(\mathbf{a})$$

So

$$\mathbb{P}(\mathbf{c} \odot \mathbf{a}) = w(\Xi(G, A, \mathbf{c}))\theta(A(\mathbf{a}))$$

Therefore

$$w(\Xi(G, A, \mathbf{c})) = \frac{\mathbb{P}(\mathbf{c} \odot \mathbf{a})}{\mathbb{E}(\mathbf{a})}$$

Consider some other tuple of terminals $\mathbf{b}$; not necessarily an anchor, and a context $\mathbf{c} \in \mathbf{a}^{\blacktriangleright}$. Then

$$\Omega(G, S, \mathbf{c} \odot \mathbf{b}) \supseteq \Xi(G, A, \mathbf{c}) \oplus A(\mathbf{b})$$

Therefore:

$$\mathbb{P}(\mathbf{c} \odot \mathbf{b}) \geq w(\Xi(G, A, \mathbf{c}))\theta(A(\mathbf{b}))$$

So:

$$\theta(A(\mathbf{b})) \leq \frac{\mathbb{P}(\mathbf{c} \odot \mathbf{b})\mathbb{E}(\mathbf{a})}{\mathbb{P}(\mathbf{c} \odot \mathbf{a})}$$

Since this is true for all contexts $\mathbf{c} \in \mathbf{a}^{\blacktriangleright}$ we have:

$$\theta(A(\mathbf{b})) \leq \inf_{\mathbf{c}\in\mathbf{a}^{\blacktriangleright}} \frac{\mathbb{P}(\mathbf{c} \odot \mathbf{b})\mathbb{E}(\mathbf{a})}{\mathbb{P}(\mathbf{c} \odot \mathbf{a})} \tag{6}$$

More generally, suppose that $\pi = A \to f(B_1, \ldots, B_r)$ is some production with A, B_i anchored by $\mathbf{a}, \mathbf{b_i}$. Then again for any unambiguous context $\mathbf{c}$, writing $u = \mathbf{c} \odot f(\mathbf{b_1}, \ldots, \mathbf{b_r})$

$$\mathbb{P}(u) \geq w(\Xi(G, A, \mathbf{c}))\theta(\pi)\prod_i \theta(B_i(\mathbf{b_i}))$$

Rearranging and minimizing with respect to $\mathbf{c}$:

$$\theta(\pi) \leq \inf_{\mathbf{c}\in\mathbf{a}^{\blacktriangleright}} \frac{\mathbb{P}(\mathbf{c} \odot f(\mathbf{b_1}, \ldots, \mathbf{b_r}))\mathbb{E}(\mathbf{a})}{\mathbb{P}(\mathbf{c} \odot \mathbf{a})\prod_i \mathbb{E}(\mathbf{b_i}))} \tag{7}$$

Importantly the right hand sides of eqs. (6) and (7) do not depend on the grammar, but only on the distribution over strings. If the grammar is LUA, then in these two equations the minimum will be attained, and we will have an equality.[1]

7 Algorithm

Given the identifiability of the class $\mathfrak{G}_r$ from strings, and the parameter identities above it is straightforward to design an algorithm which will learn this class from a sample of strings generated by the target grammar; we define the convergence criterion below in theorem 2.

The algorithm takes as input a sample of strings, and hyperparameters $\mathcal{F}, \mathcal{F}_S$; these implicitly define an upper bound r on the rank of the productions. It outputs a weighted MCFG, which can then be converted into a probabilistic MCFG (Clark and Fijalkow, 2020). The grammar will uses tuples of terminal symbols, i.e. anchors, directly as nonterminals — with this grammar format there is no need to distinguish the terminal and nonterminal symbols; but it's important to remember that we use $\tilde{f}$ to denote the application of a function f. So we might have a production $a \to f(b, c)$ where $f = xy$, and $a, b, c \in \Sigma$ are anchors that represent nonterminals. $\tilde{f}(b, c)$ is then the application of that concatenation function to b, c taken as terminals, namely the string bc. The start nonterminal will be a special distinguished symbol s as discussed.

7.1 Estimators

We start by defining some naive plug in estimators (Clark and Fijalkow, 2020). Assume we have a sample of N strings $w_1, \ldots, w_N$. Let $\hat{\Sigma}$ be the observed terminal symbols. For a string $w \in \hat{\Sigma}^*$, let $N(w)$ be the number of times $w = w_i$, and $\hat{\mathbb{P}}(w) = N(w)/N$. For $a \in \hat{\Sigma}$ define

$$\hat{\mathbb{E}}(a) = \frac{1}{N}\sum_i n(a; w_i)$$

If $\mathbf{a} = a_1, a_2$

$$\hat{\mathbb{E}}(\mathbf{a}) = \hat{\mathbb{E}}(a_1)$$

Lemma 7. *All of these estimators are consistent; $\hat{\mathbb{P}}(w)$ is a consistent estimator for $\mathbb{P}(w)$ and $\hat{\mathbb{E}}(a)$ is a consistent estimator for $\mathbb{E}(a)$, and if $\mathbf{a}$ is a Dyck pair, then $\hat{\mathbb{E}}(\mathbf{a})$ is a consistent estimator for $\mathbb{E}(\mathbf{a})$.*

For a k-tuple $\mathbf{a}$ define the frequent contexts to be

$$F(\mathbf{a}) = \{\mathbf{c} \in \mathbf{a}^{\blacktriangleright} : N(\mathbf{c} \odot \mathbf{a}) > \sqrt{N}\}$$

For a suitable linear function f, and tuples of appropriate arity, $\mathbf{a}, \mathbf{b}_1, \ldots, \mathbf{b}_r$:

$$\hat{\theta}(\mathbf{a} \to f(\mathbf{b_1}, \ldots, \mathbf{b_r})) = \frac{\hat{\mathbb{E}}(\mathbf{a})}{\prod_{i=1}^{r} \hat{\mathbb{E}}(\mathbf{b_i})} \min_{\mathbf{c}\in F(\mathbf{a})} \frac{\hat{\mathbb{P}}(\mathbf{c} \odot \tilde{f}(\mathbf{b_1}, \ldots, \mathbf{b_r}))}{\hat{\mathbb{P}}(\mathbf{c} \odot \mathbf{a})} \tag{8}$$

[1] Proving this is a little involved as we need to verify that the context is still LUA when all of the subtrees are just anchoring productions, which requires verifying some properties of $(\cdot)^{\blacktriangleright}$.

For tuples of the same dimension **a**, **b**, we will write the estimate for the rank 0 production as a $\hat{\theta}(\mathbf{a} \to \mathbf{b})$.

Lemma 8. *Suppose the strings are generated by a weighted* MCFG, $G; \theta$ *where grammar* $G \in \mathfrak{G}_r$*; given a production* $\pi_* = A \to f(B_1, \ldots, B_r)$*, let* π *be the result of replacing all nonterminals with their anchors.*

If π *is not valid, then for any* $\delta > 0$*, there is an* N *such that with probability greater than* $1 - \delta$*,* $\hat{\theta}(\pi) = 0$*.*

If $\pi_* \in P$*, then for any* $\delta > 0, \epsilon > 0$*, there is an* N *such that with probability greater than* $1 - \delta$*,* $|\hat{\theta}(\pi) - \theta(\pi_*)| < \epsilon$*.*

Proof. Since it is not valid there is some $\mathbf{c} \in \mathbf{a}^{\blacktriangleright}$, such that $\mathbf{c} \odot \tilde{f}(\mathbf{b_1}, \ldots, \mathbf{b_k}) \notin \mathcal{L}(G)$. Let N be sufficiently large that $\mathbf{c} \in F(\mathbf{a})$ with probability at least $1 - \delta$, and the result follows. □

7.2 Algorithm

Given these estimators we define the following algorithm.

- Let Σ be the observed alphabet and let s be an additional symbol. $P \leftarrow \emptyset, B \leftarrow \emptyset$
- We first compute the set of possible Dyck pairs $D \subseteq \Sigma \times \Sigma$.
- We use Algorithm FindExtremal with D as input to identify a set of 2-anchors, V_2.
- For every pair $\mathbf{b} \in \Sigma \times \Sigma$, and every $\mathbf{a} = (a_1, a_2) \in V_2$: if $\hat{\theta}(\mathbf{a} \to \mathbf{b}) > 0$, then $P \leftarrow P \cup \{\mathbf{a}(\mathbf{b})\}$ with parameter $\hat{\theta}(\mathbf{a} \to \mathbf{b})$ and $B \leftarrow B \cup \{a_1, a_2\}$.
- We use Algorithm FindExtremal with $\Sigma \setminus B$ as input, and initial symbol s to identify a set of 1-anchors, V_1.
- For every $b \in \Sigma$ and $a \in V_1 \setminus \{s\}$, if $\hat{\theta}(a \to b) > 0$, $P \leftarrow P \cup \{a(b)\}$ with parameter $\hat{\theta}(a \to b)$.
- For every $f \in \mathcal{F}$, if r is the rank of f then for every $A, B_1, \ldots B_r$ in V_1 or V_2 of appropriate dimension, let $\pi = \mathbf{a} \to f(\mathbf{b_1}, \ldots, \mathbf{b_r})$. If π is not redundant wrt P then let $\alpha = \hat{\theta}(\pi)$, If $\alpha > 0$ then add this production to P with parameter α,
- Do the same for every $f \in \mathcal{F}_S$ where A is restricted to be s.
- Construct a weighted MCFG with nonterminals of dimension 2, V_2, of dimension 1, $V_1 \cup \{s\}$, start symbol s, set of productions P, with these associated parameters.
- Convert the weighted MCFG to a probabilistic MCFG via one iteration of the expectation-maximisation algorithm.

This algorithm is polynomial in the size of the input data. The only computationally nontrivial part of this is the final conversion step, which is of the same complexity as parsing (Eisner, 2016). In the case of well-nested MCFGs, the grammars can be binarised and complexity is $\mathcal{O}(|w_i|^6)$ (Gómez-Rodríguez et al., 2010). This step can be omitted if all that is needed is the conditional distribution of trees given strings.

```
Input: Strings $w_1, w_2, \ldots, w_N$, and
       $k$-tuples $A$, optional initial symbol $s$
Output: A set of $k$-anchors $A_k \subseteq A$
$A_k \leftarrow \emptyset$ or $\{s\}$;
for $\mathbf{a} \in A$ do
    if $\forall \mathbf{b} \in A, \hat{\theta}(\mathbf{a} \to \mathbf{b}) > 0$ or $\hat{\theta}(\mathbf{b} \to \mathbf{a}) =$
      $0$ then
        if $\forall \mathbf{b} \in A_k, \hat{\theta}(\mathbf{b} \to \mathbf{a}) = 0$ then
            $A_k \leftarrow A_k \cup \{\mathbf{a}\}$;
        end
    end
end
return $A_k$
```

Algorithm FindExtremal: Find extremal elements of a set of tuples, of dimension 1 or 2.

8 Correctness

We will now prove the *correctness* of the algorithm for a certain class of probabilistic grammars that we will now define as $\mathfrak{P}_r$. For clarity we will gather together the various conditions that we require in one place.

- First, we have the normal form restrictions defined in section 2.1. These restrict the languages generated to those generated by well-nested MCFGs of dimension 2, a well studied class, except for the empty string which is not generated by any of these grammars.
- Double anchoring (definition 3): this requires that for each nonterminal we have some representative structures that are generated only by those nonterminals.

- Local unambiguity (definition 5): a weak condition on the ambiguity of the grammar.
- Completeness: the requirement that all productions that are valid either are in the grammar or can be derived from other productions: definition 7

We define the class $\mathfrak{P}_r$ to be the set of probabilistic grammars $\langle G; \theta\rangle$ where G satisfies the conditions above, and where θ is a top down parameterisation of finite expected length.[2] We claim that the algorithm is a consistent estimator for $\mathfrak{P}_r$, as stated in the following theorem.

Theorem 2. *For any* $\langle G_*; \theta_*\rangle \in \mathfrak{P}_r$ *for any* $\epsilon > 0, \delta > 0$*, there is an* N *such that if the grammar is provided with at least* N *samples then it outputs a probabilistic* MCFG $\langle \hat{G}; \hat{\theta}\rangle$ *such that, with probability at least* $1 - \delta$, $\hat{G}$ *is isomorphic to* G_* *and for all productions* π *in the original grammar* $|\theta_*(\pi) - \hat{\theta}(\hat{\pi})| < \epsilon$*, where* $\hat{\pi}$ *is the production in* $\hat{G}$ *isomorphic to* π.

Proof. (Sketch)

The backbone of the proof is Theorem 1; here we just need to show that we can derive this probabilistically. The outline of the proof is that we make a series of assumptions that will hold probabilistically. We can then bound the probability of an error in each case by some fraction of δ; using a union bound we can then bound the probability of any of them being false. Let D be the set of observed strings.

- We assume every element of the original alphabet occurs in some string in D.
- We assume that the Dyck pairs of the language are equal to the Dyck pairs of D.
- We assume that for every two Dyck pairs $\mathbf{a}, \mathbf{b}$ $\mathbf{a}^{\blacktriangleright} \subseteq \mathbf{b}^{\blacktriangleright}$ iff $\hat{\theta}(\mathbf{a} \to \mathbf{b}) > 0$.
- We assume that if a is a 1-anchor and b is any other terminal, then $a^{\blacktriangleright} \subseteq b^{\blacktriangleright}$ iff $\hat{\theta}(a \to b) > 0$, and $b^{\blacktriangleright} \subseteq a^{\blacktriangleright}$ iff $\hat{\theta}(b \to a) > 0$.

Under these assumptions, the nonterminals in the hypothesis grammar will correspond to the original nonterminals where $\mathbf{a}$ corresponds to A iff $\mathbf{a}^{\blacktriangleright} = C(G_*, A)$.

- We assume that for every possible production π; π is valid iff $\hat{\theta}(\pi) > 0$.
- We assume that for every possible production π that does correspond to a production π_* in the original grammar: $|\hat{\theta}(\pi) - \theta(\pi_*)| \leq \epsilon'$.

Under these assumptions, we will construct only the productions that are valid and non redundant, and the convergence of the parameters to the bottom-up parameters follows. We can then show that the conversion to the probabilistic MCFG gives bounded errors using some elementary analysis. □

The conditions stated are merely sufficient conditions for the learnable class: the actual class learned is larger. For example the finite language $\{a\}$ is not problematic, but violates the requirement that each dimension 1 nonterminal has two anchors.

9 Discussion

First, note that the strings themselves are enough to infer the "movement" structure but only when there is a fairly explicit clue in at least some of the strings that exhibit this structure. Secondly, well-nestedness (Kuhlmann and Möhl, 2007; Kanazawa et al., 2011) seems to make things a lot simpler; this tends to support Kanazawa et al. (2011)'s arguments for well-nestedness as a condition on mildly context-sensitive language formalisms to go with the parsing efficiency arguments (Gómez-Rodríguez et al., 2010) and the corpus based arguments of, among others, Kuhlmann and Nivre (2006). Note that for example MIX (Kanazawa and Salvati, 2012) is not in the class of languages defined; the strong learning classes are much more restricted in linguistically significant ways that the weak learning algorithms considered in Hao (2019).

Finally, we don't use the additional MCFG machinery only in those cases where it is strictly required. Rather some context-free languages, and indeed even some finite languages will have nonterminals of dimension 2; in this approach, all long distance dependencies must be handled by the mildly context-sensitive part even if they could be handled by some regular or CFG component.

The plugin estimators are computationally trivial but very slow to converge: but this can be improved using standard NLP techniques as is shown by Clark and Fijalkow (2020). The algorithm is relativised to a set of admissible production types $\mathcal{F}$. It is not the case that increasing the set of types will strictly increase the set of grammars learned: typically there will be some languages/grammars that

[2] For all nonterminals A, $I(A) = 1$ and $O(A)$ is finite.

will no longer be complete in the larger class. This approach then defines a family of different learnable classes.

The approach is quite close to the (weak) primal distributional learning approach for MCFGs developed in Yoshinaka (2010).

Language Acquisition The learning model here is highly idealised: the learner only has access, passively, to a sample of strings generated from the grammar. The child learner of course normally has access to much additional information derived from the situational context via other modalities, and additionally is an active participant, being able to interact and produce utterances of their own. The model is therefore highly pessimistic in the assumptions about the information available to the child; the positive result we are able to obtain in this model is thus correspondingly strengthened. Nonetheless, the classic term, the *primary linguistic data* correctly highlights the importance of this data in language acquisition since it is the only data source that is essential in the sense that when it is unavailable, language acquisition fails, see Clark and Lappin (2011) for discussion. At least in the early phases of language acquisition, this seems a reasonable assumption. However when we consider the acquisition of mildly context-sensitive grammars as we do here, we are considering fairly subtle properties of adult syntax which may only emerge later.

The anchoring condition is clearly unrealistic: natural languages to a certain extent have one dimensional anchors but they simply don't have the two dimensional anchors at least if we consider terminal symbols to be undisambiguated sequences of acoustic features. But the model we use here does not rely on this assumption about the terminal words. In parameter setting models of language acquisition (Yang and Roeper, 2011) it is common to assume that the input is a sequence of more abstract grammatical categories. If we consider the input data then to be sequences of some sort of shallow chunks, or non recursive phrases, then the double anchoring assumption starts to seem more realistic. In summary, from a modeling perspective it is a bit naive to attempt to model all of syntax learning using only the strings as input, even if it is cleaner from the mathematical perspective we adopt in this paper: some additional information derived from semantic constraints is certainly active in language acquisition.

Future work That said, there are no in principle reasons why this approach couldn't be extended to grammars that are only partially or approximately anchored. It's more realistic to remove productions like $A(a, b)$ entirely, and introduce lexical items only via dimension 1 nonterminals.

However a lot of the problems are because MCFGs themselves are inadequate representationally: we need a formalism with a little more structure. Alternatively one can learn some simpler constituent structure, and then subsequently learn a richer movement structure (Rogers, 2003) on top of that as in Clark (2021b). Again this seems to require the restriction to the well-nested class.

Unary rules like $A(x) \leftarrow B(x)$ and the 2-dimensional analogue, seem straightforward to introduce but will add some algorithmic complexity as in Clark (2021a). Moving to well-nested MCFGs of higher dimension seems straightforward, though the corresponding anchoring assumption is still less plausible: the corresponding generalisation of a Dyck language is, in the dimension 3 case that given by $S \to SpSqSr, \to \lambda$. In contrast the extension to non well-nested MCFGs has many technical difficulties; it is not clear whether this approach can be extended beyond the well-nested class. Moving from dimension 2 to dimension 3 seems particularly tricky then because the appropriate generalisation of the Dyck language, $D3$, is poorly understood (Moortgat, 2014).

Conclusion This paper has presented a family of algorithms for strong learning of a representative mildly context-sensitive grammar formalism, using only strings as input, with good theoretical guarantees. A realistic learning model, a representationally adequate grammar formalism, and a sufficiently strong convergence criterion: these are all good, but the class of grammars is still too small, and the grammars are too big, as the MCFG formalism itself is not succinct enough.

Acknowledgments

I am grateful to the reviewers for their helpful comments.

References

Robert C. Berwick, Paul Pietroski, Beracah Yankama, and Noam Chomsky. 2011. Poverty of the stimulus revisited. *Cognitive Science*, 35:1207–1242.

Alexander Clark. 2014. An introduction to multiple context free grammars for linguists. Unpublished manuscript.

Alexander Clark. 2021a. Beyond Chomsky normal form: Extending strong learning algorithms for PCFGs. In *Proceedings of the Fifteenth International Conference on Grammatical Inference*, volume 153 of *Proceedings of Machine Learning Research*, pages 4–17. PMLR.

Alexander Clark. 2021b. Strong learning of probabilistic tree adjoining grammars. *Proceedings of the Society for Computation in Linguistics*, 4(48).

Alexander Clark and Rémi Eyraud. 2007. Polynomial identification in the limit of substitutable context-free languages. *Journal of Machine Learning Research*, 8:1725–1745.

Alexander Clark and Nathanaël Fijalkow. 2020. Consistent unsupervised estimators for anchored PCFGs. *Transactions of the Association for Computational Linguistics*, 8:409–422.

Alexander Clark and Shalom Lappin. 2011. *Linguistic Nativism and the Poverty of the Stimulus*. Wiley-Blackwell, Malden, MA.

Jason Eisner. 2016. Inside-outside and forward-backward algorithms are just backprop (tutorial paper). In *Proceedings of the Workshop on Structured Prediction for NLP*, pages 1–17.

Seymour Ginsburg. 1966. *The Mathematical Theory of Context-Free Languages*. McGraw-Hill, Inc., New York, NY, USA.

Carlos Gómez-Rodríguez, Marco Kuhlmann, and Giorgio Satta. 2010. Efficient parsing of well-nested linear context-free rewriting systems. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 276–284.

Yiding Hao. 2019. Learnability and overgeneration in computational syntax. *Proceedings of the Society for Computation in Linguistics*, 2(1):124–134.

James Jay Horning. 1969. *A study of grammatical inference*. Ph.D. thesis, Computer Science Department, Stanford University.

Riny A .C. Huybrechts. 1984. The weak inadequacy of context-free phrase structure grammars. In G. de Haan, M. Trommelen, and W. Zonneveld, editors, *Van Periferie naar Kern*. Foris, Dordrecht, Holland.

Aravind K. Joshi. 1985. Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions? In David R. Dowty, Lauri Karttunen, and Arnold M. Zwicky, editors, *Natural Language Parsing: Psychological, Computational, and Theoretical Perspectives*, Studies in Natural Language Processing, page 206–250. Cambridge University Press.

Laura Kallmeyer and Wolfgang Maier. 2013. Data-driven parsing using probabilistic linear context-free rewriting systems. *Computational Linguistics*, 39(1):87–119.

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

Makoto Kanazawa, Jens Michaelis, Sylvain Salvati, and Ryo Yoshinaka. 2011. Well-nestedness properly subsumes strict derivational minimalism. In Sylvain Pogodalla and Jean-Philippe Prost, editors, *Logical Aspects of Computational Linguistics*, volume 6736 of *Lecture Notes in Computer Science*, pages 112–128. Springer Berlin Heidelberg.

Makoto Kanazawa and Sylvain Salvati. 2012. MIX is not a tree-adjoining language. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 666–674. Association for Computational Linguistics.

Yuki Kato, Hiroyuki Seki, and Tadao Kasami. 2006. Stochastic multiple context-free grammar for RNA pseudoknot modeling. In *Proceedings of the Eighth International Workshop on Tree Adjoining Grammar and Related Formalisms*, pages 57–64.

Gregory M. Kobele. 2006. *Generating Copies: An investigation into structural identity in language and grammar*. Ph.D. thesis, University of California Los Angeles.

Marco Kuhlmann and Mathias Möhl. 2007. Mildly context-sensitive dependency languages. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 160–167.

Marco Kuhlmann and Joakim Nivre. 2006. Mildly non-projective dependency structures. In *Proceedings of the COLING/ACL 2006 main conference poster sessions*, pages 507–514.

Roger Levy. 2005. *Probabilistic models of word order and syntactic discontinuity*. Ph.D. thesis, Stanford university.

Michael Moortgat. 2014. A note on multidimensional Dyck languages. In *Categories and Types in Logic, Language, and Physics*, pages 279–296. Springer.

Mark-Jan Nederhof and Giorgio Satta. 2008. Computing partition functions of PCFGs. *Research on Language and Computation*, 6(2):139–162.

James Rogers. 2003. Syntactic structures as multi-dimensional trees. *Research on Language and Computation*, 1(3-4):265–305.

James Scicluna and Colin de la Higuera. 2016. Grammatical inference of PCFGs applied to language modelling and unsupervised parsing. *Fundamenta Informaticae*, 146(4):379–402.

Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88(2):229.

Stuart M. Shieber. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8:333–343.

Edward P Stabler. 2013. The epicenter of linguistic behavior. In Montserrat Sanz, Itziar Laka, and Michael K. Tanenhaus, editors, *Language Down the Garden Path: The Cognitive and Biological Basis of Linguistic Structures*, pages 316–323. Oxford University Press.

Karl Stratos, Michael Collins, and Daniel Hsu. 2016. Unsupervised part-of-speech tagging with anchor hidden markov models. *Transactions of the Association for Computational Linguistics*, 4:245–257.

Charles Yang and Tom Roeper. 2011. Minimalism and language acquisition. In Cedric Boeckx, editor, *Oxford Handbook of Linguistic Minimalism*, chapter 24, pages 551–573. Oxford University Press, Oxford.

Ryo Yoshinaka. 2009. Learning mildly context-sensitive languages with multidimensional substitutability from positive data. In *International Conference on Algorithmic Learning Theory*, volume 5809 of *Lecture Notes in Computer Science*, pages 278–292. Springer.

Ryo Yoshinaka. 2010. Polynomial-time identification of multiple context-free languages from positive data and membership queries. In *Proceedings of the International Colloquium on Grammatical Inference*, pages 230–244.

Ryo Yoshinaka, Yuichi Kaji, and Hiroyuki Seki. 2010. Chomsky-Schützenberger-type characterization of multiple context-free languages. In *International Conference on Language and Automata Theory and Applications*, pages 596–607.

More efficiently identifying the tiers of strictly 2-local tier-based functions

Phillip Burness
University of Ottawa
pburn036@uottawa.ca

Kevin McMullin
University of Ottawa
kevin.mcmullin@uottawa.ca

Abstract

String-to-string functions that consider purely local information have proven useful for modelling local phonological processes, and similar modelling of long-distance processes is possible when the assessment of locality is relativized to subsets of the segment inventory (usually called *tiers*). Such tier-based functions can be learned in quadratic time and data when the tier(s) are known in advance, but existing methods for inducing the tier(s) run in quintic time. Current algorithms tailored specifically to learn tier-based functions are thus much slower overall than the cubic upper bound established for learning the superclass of subsequential functions. We show that the bottlenecks responsible for this comparatively inefficient runtime can be circumvented by judiciously using a Prefix Tree Transducer when inducing the tier(s). Doing so brings us down to a quadratic upper bound on overall runtime.

1 Introduction

It is generally accepted that the *regular* region of the Chomsky hierarchy (Chomsky, 1956) is sufficiently expressive to describe the attested range of human phonological patterns (Johnson, 1972; Kaplan and Kay, 1994). Equally well-established, though, is that regular languages are not learnable in the limit from positive data alone (Gold, 1967). Accordingly, various subsets of the regular languages and functions (i.e., *subregular* classes) have been explored as alternatives.

Local phonotactics and processes enjoy particularly strong learning results in this regard, as they can respectively be modelled using Strictly Local languages (see for example Rogers and Pullum, 2011; Rogers et al., 2013) and Strictly Local functions (see for example Chandlee and Heinz, 2018). Strictly Local languages are those that ban particular contiguous sequences from appearing in raw strings, and the runtime of their associated learning algorithm is linearly proportional to the size of the sample (Garcia et al., 1990). For their part, Strictly Local functions are those where the transformation of an input segment depends only on its immediately surrounding material in the raw string, and the runtime of their associated learning algorithms is quadratically proportional to the size of the sample (Chandlee et al., 2014, 2015).

Non-local phonotactics also enjoy strong learning results, since they can be modelled as Tier-based Strictly Local languages (see for example McMullin and Hansson, 2016). These are essentially relativized versions of Strictly Local languages that prohibit contiguous sequences after the raw strings have been projected onto a tier (Heinz et al., 2011; Lambert and Rogers, 2020). The runtime of their associated learning algorithm is linear in the size of the sample when the tier is known beforehand, but is quadratic when the tier must be identified (Jardine and McMullin, 2017). As for non-local processes, they have commonly been modelled as subsequential functions (Heinz and Lai, 2013; Luo, 2017; Payne, 2017) which can be learned in cubic time (Oncina et al., 1993).

A separate line of recent work, however, shows that non-local processes can equally be modelled using the weaker class of Tier-based Strictly Local functions, which extend Tier-based Strictly Local languages to functions in the same way that Strictly Local functions extend Strictly Local languages (see for example Andersson et al., 2020; Burness et al., 2021). Previous work on learning tier-based functions found that, while a transducer computing the function could be constructed in quadratic time when the tier is known in advance, learning the tier itself (where this is possible) took quintic time (Burness and McMullin, 2019). This had the odd consequence of making it less efficient to learn a Tier-based Strictly Local function than a subse-

Proceedings of the 17th Meeting on the Mathematics of Language, pages 38–49, Montpellier (online),
December 13, 2021. © 2021 Association for Computational Linguistics

quential function, even though the former class is strictly less expressive than the latter. In this paper, we amend this discrepancy, showing that inefficiencies in the existing tier induction methods can be eliminated by manipulating a Prefix Tree Transducer, a data structure commonly used in grammatical inference. Doing so reduces time complexity by three polynomial degrees, making it possible to identify a tier in quadratic time.

The rest of the paper is organized as follows. Section 2 outlines notation and definitions to be used throughout; it also provides formal background on tier-based functions and important properties thereof. Next, Section 3 discusses the first portion of Burness and McMullin's (2019) learning algorithm which extracts particular information from the training sample necessary for subsequent steps; we identify the bottleneck responsible for the relative inefficiency of this procedure and show that it can be avoided using a Prefix Tree Transducer. Then, Section 4 discusses the portion of Burness and McMullin's (2019) algorithm that identifies the target function's tier; this procedure faces a similar bottleneck to the preceding one which is also avoidable by using a Prefix Tree Transducer. Finally, Section 5 concludes and discusses directions for future research.

2 Preliminaries

2.1 Notation

Given a string w made of symbols from some alphabet Σ, we write $|w|$ to denote the length of that string. Below, strings will frequently be flanked by the special non-alphabet symbols $\rtimes$ and $\ltimes$, which denote the start and end of a string, respectively. Given an alphabet Σ, we write Σ^* to denote all possible strings made from that alphabet. The unique string of length 0 (i.e. the empty string) is written as λ. Given two strings u and v, we write $u \cdot v$ to denote their concatenation, though when context allows, we will save space by simply writing uv.

A suffix of some string w is any string s such that $w = x \cdot s$ and $x, s \in \Sigma^*$. Similarly, a prefix of some string w is any string p such that $w = p \cdot x$ and $p, x \in \Sigma^*$. Note that any string is a prefix and a suffix of itself, and that the empty string λ is a prefix and a suffix of every string. When $|w| \geq n$, $\texttt{suff}^n(w)$ denotes the unique suffix of w with a length of n; when $|w| < n$, it simply denotes w itself. Similarly, when $|w| \geq n$, $\texttt{pref}^n(w)$ denotes the unique prefix of w with a length of n; when $|w| < n$, it simply denotes w itself. We also write $\texttt{pref}^*(w)$ to denote the set of all prefixes of any length in w.

A string-to-string function pairs every $w \in \Sigma^*$ with one $y \in \Delta^*$, where Σ and Δ are the input alphabet and output alphabet respectively. Given a set of input strings $I \subseteq \Sigma^*$, $f(I) = \bigcup_{i \in I}\{f(i)\}$ is the set of all outputs associated to at least one of the inputs. Given a set of strings S, we write $\texttt{lcp}(S)$ to denote the *longest common prefix* of S, which is the string u such that u is a prefix of every $w \in S$, and there exists no other string v such that $|v| \geq |u|$ and v is also a prefix of every $w \in S$.

An important concept is that of the *tails* of an input string w with respect to a function f. In words, $\texttt{tails}_f(w)$ pairs every possible string $y \in \Sigma^*$ with the portion of $f(wy)$ that is directly attributable to y. Stated differently, $\texttt{tails}_f(w)$ describes the effect that w has on the output of any subsequent string of input symbols. When $\texttt{tails}_f(w_1) = \texttt{tails}_f(w_2)$ we say that w_1 and w_2 are *tail-equivalent* with respect to f.

Definition 1. Tails (Oncina and Garcia 1991)
Given a function f and an input $w \in \Sigma^*$:

$$\texttt{tails}_f(w) = \{(y, v) \mid f(wy) = uv \wedge u = \texttt{lcp}(f(w\Sigma^*))\}$$

Throughout the rest of this paper, we will need to be able to pick out the portion of the output that corresponds to actual input material. Given a transducer representation of the relevant function, this boils down to a distinction between the writing that occurs while reading segments from sigma Σ and any writing that occurs when the end of the word is reached (i.e., when $\ltimes$ is read). To make this distinction, Chandlee et al. (2015) defined the prefix function f^p associated with a subsequential function f as below. An example where $f(w)$ and $f^p(w)$ differ would be a function that appends a to the end of every input string. In this case, f^p is simply the identity map, so $f^p(abc) = abc$ whereas $f(abc) = abca$.

Definition 2. Prefix function (Chandlee et al. 2015)
Given a function f, its associated prefix function f^p is such that:

$$f^p(w) = \texttt{lcp}(f(w\Sigma^*))$$

Finally, a useful concept related to tails, tail-equivalency, and prefix functions is the *contribution* of a symbol $\sigma \in \Sigma$ relative to a string $w \in \Sigma^*$ with respect to a function f. In words, for an input string x that has the prefix $w\sigma$, the contribution of the σ in $w\sigma$ is the portion of $f(x)$ that is uniquely and directly attributable to that instance of σ. We also define a special case for the word-end symbol $\ltimes$ that is not part of Σ. The notation $x^{-1} \cdot w$ represents the string w with x removed from its front, so $a^{-1} \cdot aba = ba$ for example.

Definition 3. Contribution
Given a function f and some $w \in \Sigma^*$:

- *For $\sigma \in \Sigma$:* $\texttt{cont}_f(\sigma, w) = f^p(w)^{-1} \cdot f^p(w\sigma) = \texttt{lcp}(f(w\Sigma^*))^{-1} \cdot \texttt{lcp}(f(w\sigma\Sigma^*))$
- *For $\ltimes \notin \Sigma$:* $\texttt{cont}_f(\ltimes, w) = f^p(w)^{-1} \cdot f(w) = \texttt{lcp}(f(w\Sigma^*))^{-1} \cdot f(w)$

2.2 Single-tiered functions

Where a Strictly Local (SL) function divides Σ^* into tail-equivalence classes based on suffixes of raw strings (Chandlee, 2014; Chandlee et al., 2014, 2015), a Tier-based Strictly Local (TSL) function's tail-equivalence classes are based on suffixes of strings *after masking irrelevant elements* (Burness and McMullin, 2019; Hao and Andersson, 2019; Hao and Bowers, 2019). Relevant elements are those that belong to the specified *tier* (a subset of the alphabet) and the masking is accomplished with an *erasure* function, sometimes also called a *tier projection*.

Definition 4. Erasure function
Given a tier $T \subseteq \Sigma$, the erasure function *applied by T on Σ^* is such that:*

$$\begin{aligned}
\texttt{erase}_T(\lambda) &= \lambda \\
\texttt{erase}_T(w) &= \texttt{erase}_T(u) \cdot \sigma \text{ if } \\
&\quad w = u \cdot \sigma \wedge \sigma \in T \\
\texttt{erase}_T(w) &= \texttt{erase}_T(u) \text{ if } \\
&\quad w = u \cdot \sigma \wedge \sigma \notin T
\end{aligned}$$

SL and TSL functions are really divided into two types. On the one hand are the the Input (Tier-based) Strictly Local or I(T)SL functions which care about suffixes of the input string. On the other hand are the Output (Tier-based) Strictly Local or O(T)SL functions which care about suffixes of the output string. For reasons of space, we focus on the output-oriented OTSL functions in this paper, but the results herein are easily extended to the input-oriented ITSL case. As indicated in the following formal definition, the OTSL functions are further subdivided based on the length of suffix that is being tracked, although the learning results below apply only for $k = 2$. Note that we write $\texttt{suff}^n_T(w)$ as shorthand for $\texttt{suff}^n(\texttt{erase}_T(w))$.

Definition 5. Output Tier-based Strictly k-Local Functions (Burness and McMullin, 2019)
A function f is OTSL$_k$ if there is a tier $T \subseteq \Delta$ such that for all w_1, w_2 in Σ^*:

$$\texttt{suff}^{k-1}_T(f^p(w_1)) = \texttt{suff}^{k-1}_T(f^p(w_2)) \implies$$

$$\texttt{tails}_f(w_1) = \texttt{tails}_f(w_2)$$

The tier-induction strategy of Burness and McMullin (2019), which we optimize in this paper, relies on some important properties of OTSL$_2$ functions. First, many OTSL$_2$ functions can be described using a variety of tiers (e.g., the identity map can be described using any subset of the output alphabet), but taking the union of two potential tiers will always result in another potential tier (i.e., potential tiers can be freely combined).

Lemma 1. Free combination of tiers
Given an OTSL$_2$ function f, if $A \subseteq \Delta$ and $B \subseteq \Delta$ are both tiers for f, then $\Omega = A \cup B$ is also a tier for f.

Proof. See the proof of Lemma 4 in Burness and McMullin (2019). □

The above Lemma implies the existence of a unique largest tier for any OTSL$_2$ function that is a superset of its other possible tiers (if any others exist). Following Burness and McMullin (2019), we call this the *canonical* tier for f.

Definition 6. Canonical tier
Given an OTSL$_2$ function f, the tier $T \subseteq \Delta$ is the canonical *tier for f if and only if there is no other tier $\Omega \subseteq \Delta$ for f such that $|\Omega| \geq |T|$.*

Burness and McMullin (2019) go on to show that, if one attempts to describe an OTSL$_2$ function using a superset of its canonical tier, then there will always be at least one input-output pair which acts as evidence that one of the superfluous tier elements cannot be a member of *any* tier for f.

Lemma 2. Absolute non-tier status
Let f be an OTSL$_2$ function where $T \subseteq \Delta$ is the canonical tier. For every Ω such that $T \subset \Omega$ there will exist $a \in (\Omega - T)$, $w_1, w_2 \in \Sigma^*$, and $x \in \Sigma \cup \{\ltimes\}$ such that $\texttt{suff}^1_\Omega(f^p(w_1)) = \texttt{suff}^1_\Omega(f^p(w_2)) = a$ and $\texttt{cont}_f(x, w_1) \neq \texttt{cont}_f(x, w_2)$.

Proof. See the proof of Lemma 5 in Burness and McMullin (2019). □

Taking advantage of Lemma 1 and Lemma 2 together, we can begin by hypothesizing that the canonical tier is equal to the entire output alphabet Δ and whittle this hypothesis down as needed until we converge on the canonical tier. To do so, we look through our sample for evidence that some element cannot be on the tier, and if such an element is found, we remove it from the hypothesized tier. When no elements can be flagged for removal, we will have found the canonical tier.

2.3 Multi-tiered functions

With a TSL function, we are limited to a single tier. This is not necessarily an issue when we are considering isolated long-distance processes, but it severely limits our capacity to describe fuller phonological systems. Burness and McMullin (2021) address this issue at least partially by defining a class of Multi-Tiered Strictly Local (MTSL) functions that tracks multiple independent tier projections in parallel. The class they develop imposes a particular relationship between the tiers and the input alphabet. Namely, the contribution (see Definition 3) of a given input element can always be linked back to the effects of a set tier, although different input elements can be affected by different tiers. Viewed another way, each input element specifies a tier to which it pays exclusive attention.

In light of space limitations, and in order to cut down on redundancy in the proofs below, we henceforth stick to single-tiered functions, noting that the results herein are straightforwardly extended to Burness and McMullin's (2021) *strongly target-specified* MTSL functions just described. The major motivation behind this particular type of MTSL function was that Burness and McMullin's (2019) method for learning the single tier of a TSL function readily generalizes to the "one tier per input element" case. Our changes to the single-tier learner below do not affect any of the properties that allowed for Burness and McMullin's (2021) generalization to such multiple independent tiers.

3 Estimating the prefix function

Identifying the tier of a target function is done by comparing contributions and checking for any that do not match when the current hypothesis says they should. Calculating contributions requires knowledge of the target function's associated prefix function f^p, so the first step in the tier-learning pipeline is to extract as much knowledge as possible about f^p from the given sample. Burness and McMullin (2019) devised a method of doing so whose worst-case run time is in $\mathcal{O}(|S|^4)$, where $|S|$ is the size of the training sample. The relative inefficiency of this method comes from the fact that it must read through the sample once for each prefix in the sample, and must calculate the longest common prefix of the set returned by each of these nested reads. We present an alternative method in this section whose worst-case run time is in $\mathcal{O}(|S|^2)$ and which also allows us to greatly improve the efficiency of later learning steps. By creating and manipulating an auxiliary data structure, we completely eliminate the need for the problematic nested reads.

3.1 Building an onward PTT

We start with what is known as a Prefix Tree Transducer (PTT), defined in Definition 7 which is adapted from Chandlee et al. (2014). The PTT corresponding to a sample of input-output pairs effectively generates all and only the pairs in the sample, writing the entire output in one fell swoop after reading the entire input. For example, the PTT corresponding to {(s, s), (ss, ss), (sʃ, ss), (so, so), (sos, sos), (soʃo, soso), (sooʃ, soos)} is shown in Figure 1. To avoid visual clutter, all transitions landing in the designated final state (q_f) are incorporated into the label of their origin state.

Definition 7. Prefix Tree Transducer (adapted from Chandlee et al. 2014)
A Prefix Tree Transducer (PTT) for the finite set D of pairs (w, w') from some function f is $PTT(D) = (Q, q_0, q_f, \Sigma, \Delta, \delta)$ where:

- $Q = \bigcup_{(w,w')\in D}\{\texttt{pref}^*(w)\}$
- $(\forall u \in \Sigma^*)(\forall a \in \Sigma)$
  $[u, ua \in Q \Longleftrightarrow (u, a, \lambda, ua) \in \delta]$
- $(w, w') \in D \Longleftrightarrow (w, \ltimes, w', q_f) \in \delta$
- $(q_0, \rtimes, \lambda, \lambda) \in \delta$

In this initial form, a PTT is maximally lazy, waiting as late as possible before writing any output. For tier learning, we need to modify the PTT so that it is *minimally* lazy, producing as much output as it can as early as it can. To perform such a conversion, we can perform a depth-first parse of the sample working backwards from the leaves of the prefix tree towards the root. For each state along the way, we calculate the longest common prefix of the output edges on its outgoing transitions, pushing that string onto the output edge of its

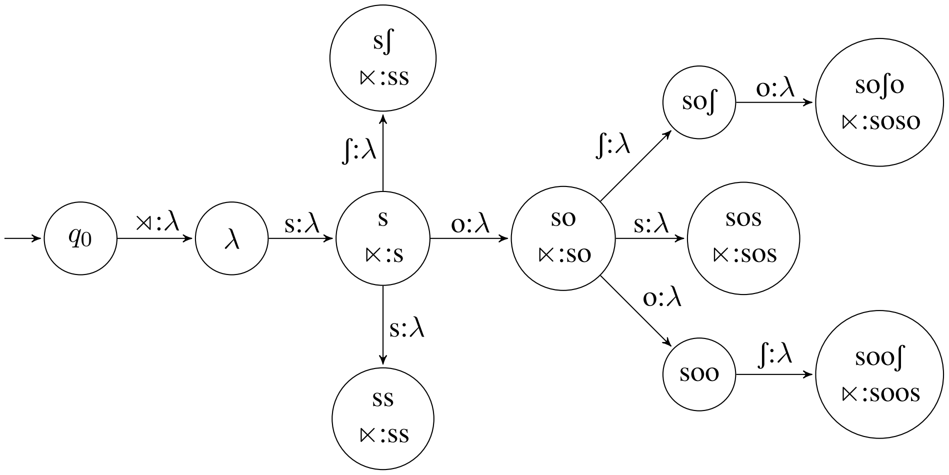


Figure 1: The PTT for the sample {(s, s), (ss, ss), (sʃ, ss), (sa, sa), (sas, sas), (saʃa, sasa), (saaS, saas)}

lone incoming transition (de la Higuera, 2010, pp. 377-379). We use the term onward PTT to refer to such a converted PTT and write $\texttt{onward}(M)$ to denote the process being applied to M. For the full details of how to build a PTT and make it onward, see chapter 18 of de la Higuera (2010). Figure 2 shows the result of onwarding the PTT in Figure 1.

Onward PTTs are used by the Onward Subsequential Transducer Inference Algorithm (OSTIA) of Oncina et al. (1993) and are used by the learning algorithm for ISL functions (Chandlee et al., 2014) but not the one for OSL functions (Chandlee et al., 2015). Interestingly, both OSTIA and the ISL function learning algorithm obtain a transducer representation of the target function by applying a process of state merging to an onward PTT. In contrast, the learning algorithm in this paper merely uses an onward PTT as a sort of oracle, consulting it for essential pieces of information without modifying it in any way.

3.2 Extracting useful information

A PTT that has been made onward exhibits some important properties that will be exploited below. First, given a state $q \in Q$ that has an outgoing transition for all $x \in \Sigma \cup \{\ltimes\}$, we will have produced exactly $f^p(q)$ so far when we enter the state q. We call such a state a *supported* state. Assuming that $\Sigma = \{$s, o, ʃ $\}$, the supported states in Figure 2 are 's' and 'so'.

Definition 8. Supported state
Given an onward PTT $P = (Q, q_0, q_f, \Sigma, \Delta, \delta)$, the state $q \in Q$ is supported *if and only if:*

$$(\forall x \in \Sigma \cup \{\ltimes\})[\exists(q, x, y, q') \in \delta]$$

Lemma 3. *Let P be the onward PTT constructed according to sample S drawn from function f. Given a supported state $q \in Q$, it is the case that we will have written exactly $f^p(q)$ upon entering q after starting in q_0.*

Proof. Since q is supported, it is the case that $(\forall x \in \Sigma \cup \{\ltimes\})[\exists(q, x, y, q') \in \delta]$. This in turn means that we have $(q, f(q)) \in S$ and for each $a \in \Sigma$ we have $(qab, f(qab)) \in S$ for some $b \in \Sigma^*$. An onward PTT is deterministic and acyclic, so inputs will only pass through q if they have q as a prefix, and all such inputs in S are guaranteed to do so. Let the set M_q (for "matching q") be this subset of the inputs in S. Because all and only the inputs in S that are also in M_q will pass through q, the process of making the PTT onward will push $\texttt{lcp}(f(M_q))$ past q towards the root such that exactly this $\texttt{lcp}$ will have been written when q is entered after starting in q_0. Now recall that $f^p(w) = \texttt{lcp}(\{u \mid u = f(wy) \wedge y \in \Sigma^*\})$. It is sufficient to use a set containing $f(w)$ and at least one $f(wv) = f(wab)$ for each $a \in \Sigma$ (where $b \in \Sigma^*$) because every $v \in \Sigma^*$ is either λ or begins with some $a \in \Sigma$. The set M_q fulfills this criterion and so $\texttt{lcp}(f(M_q)) = f^p(q)$. □

The other crucial property of an onward PTT follows from the first: given a transition $(q, x, y, q') \in \delta$ such that q is a supported state and q' is either another supported state or q_f, the string y is guaranteed to be equal to $\texttt{cont}_f(x, q)$ provided that f is a subsequential function. The transitions meeting these criteria in Figure 2 are (s, $\ltimes, \lambda, q_f$), (s, o, o, so) and (so, $\ltimes, \lambda, q_f$).

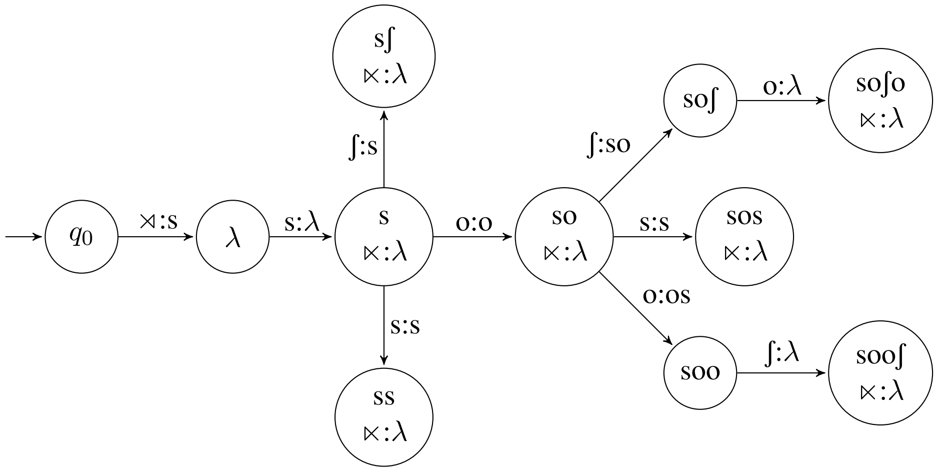


Figure 2: The onward version of Figure 1

Remark 1. *Let P be the onward PTT constructed according to a sample drawn from function f. A corollary of Lemma 3 is that, given a supported q:*

- $(q, \ltimes, u, q_f)$ *is such that* $u = \mathtt{cont}_f(\ltimes, q) = f^p(q)^{-1} \cdot f(q)$
- (q, σ, v, r) *for* $\sigma \in \Sigma$ *is such that* $v = \mathtt{cont}_f(\sigma, q) = f^p(q)^{-1} \cdot f^p(q\sigma)$ *if r is also supported.*

The procedure `estimate_fp` shown in Algorithm 1 sends a sample S once through its onward PTT and returns all pairs $(q, f^p(q))$ such that q is a supported state. This set will be exploited along with the PTT when inducing the tier; the tier-learning algorithm below essentially treats this PTT and the constructed set as a sort of oracle that it can consult for information about f^p. We close this section by showing that extracting information about f^p from a sample using Algorithm 1 takes quadratic time in the worst case.

Lemma 4. Quadratic time (`estimate_fp`)
For any sample S of input-output pairs, Algorithm 1 runs in $\mathcal{O}(|S|^2 \cdot |\Sigma|)$ time.

Proof. Let $l = \sum_{(w,u)\in S} |w|$ be the summed lengths of all inputs in the sample, let $o = max\{|u| : (w, u) \in S\}$ be the longest output length in the sample, and let s be the number of pairs in the sample. These magnitudes are all linear in the size of the sample.

Constructing $P = PTT(S)$ requires a single read through S, taking l steps. Making this P onward takes at most ol steps (for details, see chapter 18 of de la Higuera, 2010). There are at most l non-initial/non-final states in P and `estimate_fp` starts by checking each of these once. For each, it verifies whether all possible $|\Sigma| + 1$ outgoing transitions exist. There are at most $l + s$ transitions in P, meaning that checking whether a transition exists takes at most $l + s$ steps. The first portion of `estimate_fp` thus takes at most $l((l+s)(|\Sigma|+1))$ steps. The second portion sends the sample through P one input letter at a time, checking on each step whether the landing state is in A. Checking whether a state is in A takes at most l steps, so the second portion of `estimate_fp` takes at most l^2 steps. Taken together, the run time is in $\mathcal{O}(l+ol+l(l+s)|\Sigma|+l^2)$, which is quadratic in the size of S and linear in the size of Σ. □

4 Identifying tier(s)

4.1 Overview of the process

The full tier induction process is shown in Algorithm 2. This algorithm adapts the overall strategy from Burness and McMullin (2019) so that it can be used with the PTT objects described above. In their original implementation of the strategy, the learner had to (1) sift through the sample for pairs meeting a certain criterion and (2) calculate contributions relative to each member of the collected subset. The latter step requires an additional scan through the sample for each pair acting as the basis of comparison, and like above, this nested reading of the sample creates a significant bottleneck which ultimately makes the process run in $\mathcal{O}(|S|^5)$ time.

The key insight in this paper is that certain transitions in an onwarded PTT will be equal to their corresponding contribution. Remark 1 tells us that these are easily identified by checking them against the set of pairs $(q, f^p(q))$ produced by the revised

```
Data: A sample $S$
Result: An onward PTT $P$ and the set $A$
        containing the pair $(q, f^p(q))$ for
        each supported $q \in Q$
Function estimate_fp($S$):
    $P \leftarrow PTT(S) = (Q, q_0, q_f, \Sigma, \Delta, \delta)$;
    $P \leftarrow \mathtt{onward}(P)$;
    $A \leftarrow \emptyset$;
    $B \leftarrow \emptyset$;
    for $q \in Q$ do
        if $\forall x \in \Sigma \cup \{\ltimes\}$,
        $\exists(q, x, y, q') \in \delta$ then
            $B \leftarrow B \cup \{q\}$
    for each $(x, y) \in S$ do
        $a \leftarrow \lambda$;
        $\mathtt{b} \leftarrow z$ such that $(q_0, \rtimes, z, \lambda) \in \delta$;
        if $\mathtt{a} \in B$ then
            $A \leftarrow A \cup \{(\mathtt{a}, \mathtt{b})\}$;
        for $n$ from $1$ to $|x|$ do
            $\mathtt{r} \leftarrow$ the $n$-th letter of $x$;
            $\mathtt{w} \leftarrow u$ such that
             $(\mathtt{a}, \mathtt{r}, u, q) \in \delta$;
            $\mathtt{a} \leftarrow q$ such that
             $(\mathtt{a}, \mathtt{r}, \mathtt{w}, q) \in \delta$;
            $\mathtt{b} \leftarrow \mathtt{b} \cdot \mathtt{w}$;
            if $\mathtt{a} \in B$ then
                $A \leftarrow A \cup \{(\mathtt{a}, \mathtt{b})\}$;
    return $A, P$
```

Algorithm 1: Prefix function estimation

estimate_fp. Instead of calculating and recalculating contributions to find mismatches, then, we can simply cycle through the list of transitions in the onwarded PTT, sorting them into bins based on the current tier hypothesis. Doing so eliminates the problematic nesting and accordingly reduces the upper bound on runtime by three polynomial degrees to $\mathcal{O}(|S|^2)$.

This reduction in the degree of nesting is similar to how the ISL learning algorithm's quadratic time complexity relates to OSTIA's cubic time complexity. Both learning algorithms take an onward PTT and merge pairs of states until they terminate. Where n is the number of states in the provided PTT, the number of possible merges performed by OSTIA is in $\mathcal{O}(n^2)$ since it can reject and undo merges (Oncina et al., 1993); in contrast, the number of possible merges performed by the ISL learning algorithm is in $\mathcal{O}(n)$ since it cannot reject and undo merges (Chandlee et al., 2014). For each state merging, both algorithms apply operations that run in $\mathcal{O}(|S|)$, and as a result, the overall complexity of OSTIA and the ISL learning algorithm are cubic and quadratic in the size of the sample, respectively.[1]

We start by hypothesizing that $T = \Delta$ (i.e., that all members of the output alphabet are on the tier). Then, for each transition (q, x, y, r) in the onward PTT, the algorithm checks whether q is a supported state (i.e., whether there is a pair in A associated to it) and whether r is a supported state or q_f. If both of these conditions hold, then the output edge y of the transition is equal to $\mathtt{cont}_f(x, q)$. Accordingly, the algorithm takes $(q, f^p(q)) \in A$, calculates $t = \mathtt{suff}^1_T(f^p(q))$, and adds y to the bin $C_{x,t}$ (the set of contributions for x when the tier suffix is t) if it is not already there. If t is on the function's canonical tier, the cardinality of this bin should always be equal to or less than 1 since the target function is OTSL$_2$ and so the contribution should be the same whenever the output tier suffix is equal to t.

After scanning through all transitions in the onward PTT, the algorithm looks for any constructed bins of contributions $C_{x,t}$ with cardinality greater than 1. If none of the bins associated with a specific $t \in T$ has cardinality greater than 1, the element t will get added to the auxiliary set K (for "keep") containing elements that are safe to keep on the tier for now. If any of the bins associated to $t \in T$ have cardinality greater than 1, the element t is removed from the tier hypothesis since it cannot possibly be a member of the function's canonical tier. If at any point some symbol gets removed from T, the set K is immediately emptied. The algorithm repeatedly alternates between scanning the PTT transitions and checking the cardinality of contribution sets until every t in the current hypothesis for T gets added to the set K, in which case it has found the canonical tier of the target function.

4.2 Proofs of correctness/efficiency

In this subsection, we establish that our revision of Burness and McMullin's (2019) algorithm achieves the same result in much less time.

Lemma 5. Quadratic time (get_tier)
For any input sample S, get_tier(S) produces a tier T in $\mathcal{O}(|S|^2 \cdot |\Sigma| \cdot |\Delta|^2)$ time.

[1] The quadratic time complexity of the OSL learning algorithm, for its part, comes from repeatedly calculating the longest common prefix of stringsets lifted from the sample (Chandlee et al., 2015).

```
Data: A sample $S$
Result: A tier $T \subseteq \Delta$
Function get_tier($S$):
    $A, P \leftarrow$ estimate_fp($S$);
    $T \leftarrow \Delta$;
    $K \leftarrow \emptyset$;
    while $K \neq T$ do
        for each $t \in T$ do
            for each $x \in \Sigma \cup \{\ltimes\}$ do
                $C_{x,t} = \emptyset$;
        for each $(q, x, y, r) \in \delta$ from $P$ do
            if $[\exists(q, a) \in A] \wedge [[r = q_f] \vee [\exists(r, b) \in A]]$ then
                $t \leftarrow \texttt{suff}^1_T(a)$;
                $C_{x,t} \leftarrow C_{x,t} \cup \{y\}$;
        for each $t \in T$ do
            for each $x \in \Sigma \cup \{\ltimes\}$ do
                if $|C_{x,t}| > 1$ then
                    $T \leftarrow T - \{t\}$;
                    $K \leftarrow \emptyset$;
            if $t \in T$ then
                $K \leftarrow K \cup \{t\}$
    return $T$
```

Algorithm 2: Single tier induction

Proof. Let $l = \sum_{(w,u) \in S} |w|$ be the summed lengths of all inputs in the sample, let $o = max\{|u| : (w, u) \in S\}$ be the longest output length in the sample, let $i = max\{|w| : (w, u) \in S\}$ be the longest input length in the sample, and let s be the number of pairs in the sample. These are all linear in the size of the sample.

The first step is to run estimate_fp on the sample which Lemma 1 already established as running in $\mathcal{O}(|S|^2)$. Following that, the *while* loop can run up to $|\Delta|$ times. The first *for* loop initializes the contribution sets that will be constructed, of which there are at most $|\Delta| \cdot |\Sigma|$. Then, for each of the up to $l + s$ transitions in P, the second *for* loop it searches A up to two times to check whether the origin is supported and whether the destination is supported or final. A single search of A take at most l steps, and if both conditions are met we calculate the relevant output tier suffix, taking at most o steps. Finally, the third *for* loop inspects all the transitions in P, we check the cardinality of each contribution set, which takes at most $|\Delta| \cdot |\Sigma|$ steps. The overall run time of get_tier(S) is thus in $\mathcal{O}(|\Delta|^2|\Sigma| + |\Delta|(l + s)(l + o))$, which is quadratic in the size of S, linear in the size of Σ and quadratic in the size of Δ. □

The remaining lemmata of this section will show that for each total OTSL$_2$ function f, there is a finite kernel of data consistent with f that is a characteristic set for the algorithm (i.e., if the training set subsumes this kernel, the algorithm is guaranteed to succeed). The OTSL$_k$ functions divide Σ^* into a finite number of equivalence classes according to sets of tails, meaning that the OTSL$_k$ functions are also subsequential functions. Oncina and Garcia (1991) show how the finite partition of Σ^* lets us build the smallest finite-state transducer that computes a given subsequential function. Given a state q in this canonical transducer $\mathcal{F}$, we write w_q to denote the length-lexicographically earliest input string that reaches the state q, and define the characteristic set as follows. Note that this same characteristic set is used by Burness and McMullin (2021) for their multi-tier learner; they showed that its size is in $\mathcal{O}(|\mathcal{F}|^2)$.

Definition 9. Characteristic set
A sample S contains a characteristic set iff it contains the following for each state q in $\mathcal{F}$:

1. *The input-output pair $(w_q, f(w_q))$.*
2. *For all triples $a, b, c \in \Sigma$:*
   i. *some pair $(w_q a, f(w_q a))$,*
   ii. *some pair $(w_q ab, f(w_q ab))$, and*
   iii. *some pair $(w_q abcv, f(w_q abcv))$, where $v \in \Sigma^*$*

Lemma 6. Quadratic data
There exists a characteristic set whose size is in $\mathcal{O}(|\mathcal{F}|^2)$.

Proof. See the proof of Lemma 17 in Burness and McMullin (2021). □

Lemma 7. Evidence availability
If a learning sample S contains a characteristic set and P is the onward PTT for S then for all $w \in \Sigma^*$ and all pairs $x, y \in \Sigma$ there is at least one transition in P corresponding to each of the following that (1) leaves a supported state and (2) ends in q_f or a supported state:

- $\texttt{cont}_f(x, w)$
- $\texttt{cont}_f(\ltimes, w)$
- $\texttt{cont}_f(y, wx)$
- $\texttt{cont}_f(\ltimes, wx)$

Proof. For any input string $w \in \Sigma^*$, reading w will lead to some non-initial and non-final state q in $\mathcal{F}$. The target function is subsequential which means that that either $w = w_q$ or else can be replaced thereby since subsequentiality implies that $\texttt{cont}_f(i, w) = \texttt{cont}_f(i, w_q)$ for any $i \in \Sigma \cup \{\ltimes\}$. By the definition of the seed, for every state q in $\mathcal{F}$ and for every triple $x, y, z \in \Sigma$, the learner will see w_q, w_qx, w_qxy, and w_qxyzv where $v \in \Sigma^*$. This means that the states w_q, w_qx, and w_qxy in P are all supported. As Remark 1 notes, Lemma 3 then implies that:

- (w_q, x, a_1, w_qx) in P is such that $a_1 = f^p(w_q)^{-1} \cdot f^p(w_qx) = \texttt{cont}_f(x, w_q)$
- $(w_q, \ltimes, a_2, q_f)$ in P is such that $a_2 = f^p(w_q)^{-1} \cdot f(w_q) = \texttt{cont}_f(\ltimes, w_q)$
- (w_qx, y, a_3, w_qxy) in P is such that $a_3 = f^p(w_qx)^{-1} \cdot f^p(w_qxy) = \texttt{cont}_f(y, w_qx)$
- $(w_qx, \ltimes, a_4, q_f)$ in P is such that $a_2 = f^p(w_qx)^{-1} \cdot f(w_qx) = \texttt{cont}_f(\ltimes, w_qx)$

□

Lemma 8. Tier convergence
Given a learning sample S that contains a characteristic sample, $\texttt{get\_tier}(S)$ will produce the canonical tier of f.

Proof. Let T be the canonical tier of f, and let H be the tier constructed by the algorithm. The algorithm begins with $H = \Delta$, and so either $H = T$ already, or else $H \supset T$. The algorithm is designed to consider all and only the transitions (q, x, y, r) in $P = \texttt{onward}(PTT(S))$ such that q is a supported state and r is a supported state or q_f. As Remark 1 notes, Lemma 3 implies that $y = \texttt{cont}_f(x, q)$ for all such transitions. For each considered transition (q, x, y, r) in P, the algorithm sorts y into the bin associated simultaneously with x and with $z = \texttt{suff}^1_H(p)$, where p is equal to the output produced upon reading q in P. Note that the algorithm has easy access to the string p because it is paired with q in the auxiliary set A. Note also that since q is a supported state, Lemma 3 tells us that $p = f^p(q)$.

Now, we know from Lemma 2 that if $H \supset T$, there will exist a pair of input strings w_1 and w_2 in the domain of f such that $\texttt{cont}_f(x, w_1) \neq \texttt{cont}_f(x, w_2)$ even though $\texttt{suff}^1_H(f^p(w_1)) = \texttt{suff}^1_H(f^p(w_2)) = a$ for some $a \in (H - T)$ and some $x \in \Sigma \cup \{\ltimes\}$. Furthermore, Lemma 7 tells us that for all non-initial/non-final states s in the minimal FST $\mathcal{F}$ producing f, each transition along every possible sequence of two or fewer steps out of s will have at least one equivalent transition in P that is considered by the algorithm. If the first transition along one of these paths produces any elements not in T, we stand the chance of incorrectly binning the second transition when $H \supset T$. Since we see all possible paths of two transitions, at least one pair of unequal contributions (which *should* be placed into two different bins linked to two different members of T, since f is OTSL$_2$) will be placed together into a bin that should not exist (because that bin is linked to a non-member of T) when $H \supset T$.

Accordingly, at least one of the bins associated with some $b \in (H - T)$ will have a cardinality greater than 1 (assuming repeated strings are counted only once) when $H \supset T$. The algorithm will thus flag and remove at least one $b \in (H - T)$ when $H \supset T$. Conversely, there will be no pair of input strings w_3 and w_4 in the domain of f such that $\texttt{cont}_f(x, w_3) \neq \texttt{cont}_f(x, w_4)$ when $\texttt{suff}^1_H(f^p(w_3)) = \texttt{suff}^1_H(f^p(w_4)) = c$ for any $c \in T$ and any $x \in \Sigma \cup \{\ltimes\}$. Consequently, none of the bins associated with any $c \in T$ will ever surpass a cardinality of 1 (assuming repeated strings are counted only once). When $H = T$, then, the algorithm will add all $d \in H$ to K, at which point $K = H = T$. □

Theorem 1. *$\texttt{get\_tier}$ identifies the canonical tier of any total OTSL$_2$ function in $\mathcal{O}(|S|^2)$ time and $\mathcal{O}(|\mathcal{F}|^2)$ data.*

Proof. Immediate from Lemmata 5, 6, and 8. □

The sample and the tier returned by $\texttt{get\_tier}(S)$ can then be fed to the transducer building algorithm from Burness and McMullin (2019), which is a generalization of the transducer building algorithm from Chandlee et al. (2015) and whose worst-case runtime is also in $\mathcal{O}(|S|^2)$. Since all three components of the tier-based function learning pipeline now have a quadratic upper bound on runtime, the overall process from start to finish now also has a quadratic upper bound.

5 Discussion and conclusion

SL functions, aside from closely approximating the typology of local processes (Chandlee and Heinz, 2018), are highly useful from a learnability standpoint. With their quadratic upper bounds on runtime, it can be preferable to use the SL function learning algorithms (Chandlee et al., 2014, 2015) over the Onward Subsequential Transducer Inference Algorithm (OSTIA) of Oncina et al. (1993). While OSTIA can learn all the same functions as the SL function learners and more (since the subsequential functions properly contain the SL functions), its run time is cubic in the worst case. Sacrificing expressiveness, in this case, is offset by a gain in efficiency, making it worthwhile in appropriate circumstances.

Prior to this paper, the same tradeoff was only true for TSL and MTSL functions when the learner already knew the necessary tier(s). Such advance knowledge permits basic generalizations of the SL learning algorithms that preserve their complexity bounds (Burness and McMullin, 2019). Of course, it is not realistic for a learner to come equipped with foreknowledge of the relevant tier(s), so a focus of research on tier-based functions has been whether and how tiers can be identified from positive examples drawn from the target function. Initial methods from this enterprise were limited in that they (i) only work for functions with a window length (the parameter k) of 2 and (ii) are less efficient than OSTIA by two polynomial degrees, with a quintic worst-case runtime. Accordingly, the fact that some tiers could be learned from positive data was effectively a technical curiosity from the perspective of learning performance.

Practically speaking, a learner was better off attempting to build a subsequential function than a TSL or MTSL function when an SL function was not sufficient. Our contribution here was to show that the inefficiencies of tier learning could be overcome by manipulating a Prefix Tree Transducer (a data structure also used by OTSIA and the ISL learning algorithm) rather than just manipulating the sample. Doing so circumvents the need for nested reading of the sample, which we identified as the major bottleneck of previous methods. Our revision of the methods from Burness and McMullin (2019, 2021) reduces their upper bound on runtime by three polynomial degrees. As was the case for the SL functions, then, the sacrifice in expressiveness from eschewing a subsequential function in favour of a TSL$_2$ function (or a *strongly-target specified* MTSL$_2$ function; Burness and McMullin, 2021) is offset by an appreciable gain in efficiency.

While we have focused mainly on concerns of practicality in this paper, we do note that there are also conceptual grounds for using TSL and MTSL functions over subsequential functions as models of long-distance phonological process. It is well-established that subsequential computation is sufficiently expressive to model non-local vowel harmony (Heinz and Lai, 2013), consonant harmony (Luo, 2017), and consonant dissimilation (Payne, 2017) with a handful of exceptions in the form of *unbounded circumambience* (Jardine, 2016; McCollum et al., 2020). That being said, the relativized locality underpinning tier-based functions has been shown to more intuitively capture attested long-distance behaviours (Andersson et al., 2020; Burness et al., 2021), while excluding some pathological behaviours like modulo counting which are otherwise amenable to a subsequential analysis (Burness et al., 2021). Combining this work with the learnability results in the current paper solidifies the appropriateness of TSL and MTSL functions as models of long-distance phonological processes.

Several hurdles, however, still remain to be overcome in the area of tier-based function learning. First and foremost, the results herein require a window size (k) of 2; the properties exploited by the learner do not hold for larger window sizes. This is in stark contrast to tier-based *languages*, whose tiers are efficiently learnable for arbitrary window sizes (Jardine and McMullin, 2017; Lambert, 2021). Second, the learner developed above is a batch learner (as are OSTIA and the SL learners), making it unlikely as a model of real human phonological learning. In this regard as well, the existing work on languages outpaces the work on functions, since an online learner was recently developed for TSL languages (Lambert, 2021). Finally, the way in which we manipulate the PTT during tier learning assumes that the function is total. To learn partial functions, it may be necessary to provide the learner with some additional information, like how Oncina and Varó (1996) and Castellanos et al. (1998) augment OSTIA by giving it access to domain and range information, respectively.

References

Samuel Andersson, Hossep Dolatian, and Yiding Hao. 2020. Computing vowel harmony: The generative capacity of search and copy. In *Proceedings of the 2019 Annual Meeting on Phonology*.

Phillip Burness and Kevin McMullin. 2019. Efficient learning of Output Tier-Based Strictly 2-Local functions. In *Proceedings of the 16th Meeting on the Mathematics of Language*, pages 78–90. Association for Computational Linguistics.

Phillip Burness and Kevin McMullin. 2021. Learning multiple independent tier-based processes. In *Proceedings of the Fifteenth International Conference on Grammatical Inference*, volume 153 of *Proceedings of Machine Learning Research*, pages 66–80. PMLR.

Phillip Burness, Kevin McMullin, and Jane Chandlee. 2021. Long-distance phonological processes as tier-based strictly local functions. *Glossa*, 6.

Antonio Castellanos, Enrique Vidal, Miguel A. Varó, and José Oncina. 1998. Language understanding and subsequential transducer learning. *Computer Speech and Language*, 12:193–228.

Jane Chandlee. 2014. *Strictly Local Phonological Processes*. Doctoral dissertation, University of Delaware.

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. 2014. Learning Strictly Local subsequential functions. *Transactions of the Association for Computational Linguistics*, 2:491–503.

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. 2015. Output Strictly Local functions. In *Proceedings of the 14th Meeting on the Mathematics of Language (MOL 2015)*, pages 112–125.

Jane Chandlee and Jeffrey Heinz. 2018. Strict Locality and phonological maps. *Linguistic Inquiry*, 49:23–60.

Noam Chomsky. 1956. Three models for the description of language. *IRE Transactions on Information Theory*, 2:113–124.

Colin de la Higuera. 2010. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, New York.

Pedro Garcia, Enrique Vidal, and José Oncina. 1990. Learning Locally Testable languages in the strict sense. In *Proceedings of the Workshop on Algorithmic Learning Theory*, pages 325–338. Japanese Society for Artificial Intelligence.

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

Yiding Hao and Samuel Andersson. 2019. Unbounded stress in subregular phonology. In *Proceedings of the 16th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology and Morphology*, pages 135–143, Florence, Italy. Association for Computational Linguistics.

Yiding Hao and Dustin Bowers. 2019. Action-sensitive phonological dependencies. In *Proceedings of the 16th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology and Morphology*, pages 218–228, Florence, Italy. Association for Computational Linguistics.

Jeffrey Heinz and Regine Lai. 2013. Vowel harmony and subsequentiality. In *Proceedings of the 13th Meeting on the Mathematics of Language (MOL 13)*, pages 52–63, Sofia, Bulgaria. Association for Computational Linguistics.

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, OR. Association for Computational Linguistics.

Adam Jardine. 2016. Computationally, tone is different. *Phonology*, 33:247–283.

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 (LATA 2017)*, pages 64–76.

C. Douglas Johnson. 1972. *Formal Aspects of Phonological Description*. Mouton, The Hague.

Ronald M. Kaplan and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20:331–378.

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. PMLR.

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 (SCiL) 2020*, pages 330–337, New Orleans, Louisianna.

Huan Luo. 2017. Long-distance consonant agreement and subsequentiality. *Glossa: A Journal of General Linguistics*, 2:1–25.

Adam G. McCollum, Eric Baković, Anna Mai, and Eric Meinhardt. 2020. Unbounded circumambient patterns in segmental phonology. *Phonology*, 37:215–255.

Kevin McMullin and Gunnar Ólafur Hansson. 2016. Long-distance phonotactics as Tier-Based Strictly 2-Local Languages. In *Proceedings of the 2014 Annual Meeting on Phonology*, Washington, DC. Linguistic Society of America.

José Oncina and Pedro Garcia. 1991. Inductive learning of subsequential functions. Technical Report DSIC II-34, University Politecnia de Valencia.

José Oncina, Pedro Garcia, and Enrique Vidal. 1993. Learning subsequential transducers for pattern recognition tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:448–458.

José Oncina and Miguel A. Varó. 1996. Using domain information during the learning of a subsequential transducer. In Laurent Miclet and Colin de la Higuera, editors, *Grammatical Interference: Learning Syntax from Sentences*, number 1147 in Lecture Notes in Artificial Intelligence, pages 301–312. Springer, Berlin.

Amanda Payne. 2017. All dissimilation is computationally subsequential. *Language*, 93:353–371.

James Rogers, Jeffrey Heinz, Margaret Fero, Jeremy Hurst, Dakotah Lambert, and Sean Wibel. 2013. Cognitive and sub-regular complexity. In *Formal Grammar*, number 8036 in Lecture Notes in Artificial Intelligence, pages 90–108. Springer.

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

Tier-based modeling of gradience and distance-based decay in phonological processes

Kevin McMullin
University of Ottawa
kevin.mcmullin@uottawa.ca

Phillip Burness
University of Ottawa
pburn036@uottawa.ca

Abstract

Current computational approaches to long-distance phonological processes use string-to-string function classes that operate over phonological tiers, but these are necessarily deterministic devices and are thus limited to enforcing categorical application. We show that probabilistic relations that act like a tier-based function (aside from being non-deterministic) perform well as models of gradient long-distance processes. In particular, they offer a cognitively plausible characterization of *distance-based decay* (Zymet, 2015) with other desirable properties, exemplified by two case studies. The first, examining rounding dissimilation in Malagasy, demonstrates that tier-based models of decay can be made sensitive to phonetic similarity in interesting ways. The second, examining Hungarian backness harmony, demonstrates that tier-based models of decay can handle scenarios where a process is obligatory at short distances but vanishingly unlikely at increasing distances.

1 Introduction

Taking inspiration from foundational results in *Autosegmental Phonology* (e.g., Goldsmith 1976), recent computational work has shown that long-distance phonological processes can be fruitfully modelled using string-to-string function classes that operate according to a relativized notion of strict locality. These classes include the Tier-based Strictly Local (TSL) functions explored by Burness and McMullin (2019), Hao and Andersson (2019), Hao and Bowers (2019), and Andersson et al. (2020), as well as the Multi-tiered Strictly Local (MTSL) functions (Burness and McMullin, 2020). While these functions have offered valuable insights into the computational characteristics of non-local phonology, they are limited in that they assume every input has exactly one output. Consequently, these functions can only describe either mandatory application or mandatory non-application of a process. Real language data is, however, not always this clean; many phonological processes apply *optionally*, and long-distance processes are no exception to this fact. This paper will explore how the requirement of determinism can be relaxed in order to describe the probabilistic application of a process, while still maintaining the advantages of (tier-based) strict locality. In particular, by augmenting the tier-based structures with duplicate transitions for non-tier elements, we are able to model phonological processes with a well-known property of *distance-based decay*, wherein the probability that a long-distance process applies will exponentially diminish as more and more transparent segments intervene between the trigger and target (Zymet, 2015).

The paper is structured as follows. First, Section 2 provides the necessary background on tier-based functions and their automata-theoretic characterization. Then, Section 3 looks at some optional long-distance patterns and shows how strategically adding transitions to a TSL or MTSL FST and weighting them can describe the desired probabilistic distribution of output forms for a given input. After that, Section 4 considers distance-based decay, and proposes that weighted transducers built according to a TSL or MTSL template can derive distance-based decay in a cognitively plausible manner. Section 5 concludes.

2 Categorical tier-based functions

We begin this section with a modicum of notation and definitions An *alphabet* is a set of elements from which strings can be built. The concatenation of two strings u and v is written as $u \cdot v$, although this is shortened to uv when context permits. Given an alphabet Σ, we write Σ^* to denote the set of all strings of any length (including 0) that can be

Proceedings of the 17th Meeting on the Mathematics of Language, pages 50–63, Montpellier (online),
December 13, 2021. © 2021 Association for Computational Linguistics

constructed using Σ. Here and throughout, we use λ to denote the unique *empty string*, which has a length of 0 and satisfies $\lambda \cdot w = w \cdot \lambda = w$. Given an alphabet Σ of input elements and an alphabet Γ of output elements, a (partial) *string-to-string function* is a mapping from Σ^* to Γ^* where each $w \in \Sigma^*$ is paired with (at most) one string in Γ^*.

We will mainly demonstrate and discuss tier-based functions (and their probabilistic variants) with reference to the *finite-state transducers* (FSTs) that compute them. A (one-way) FST produces an output string incrementally by reading an input string one element at a time in a single direction. Such a machine consists of a finite set of states (which can be thought of as a primitive sort of memory) and a finite set of transitions between these states (which are the machine's instructions for what to write at each step). The machine begins in a designated initial state, and traverses a path through the state space by following transitions in response to the input that it reads. Each state is given a (potentially empty) *final string* which is appended to the output when the machine lands in that state after consuming the whole input string. Figure 1 presents a visual diagram of an FST. States are represented using circles and the initial state is marked with an unlabeled incoming arrow. Transitions are represented with labelled arrows between states; a label 'a:b' is an instruction to take that transition when reading 'a' from the input and write 'b' to the output. Final strings are shown underneath the state label, formatted like a transition label for the special end-marker ⋉. The transducer in Figure 1 operates over the input alphabet {a, b}, transforming all odd-numbered positions to 'a', transforming all even-numbered positions to 'b', and appending 'b' to the end if it runs out of input after writing 'a' (i.e., if it ends in the state labelled '1'). For example it maps /bab/ to [abab] and maps /aabbab/ to [ababab].

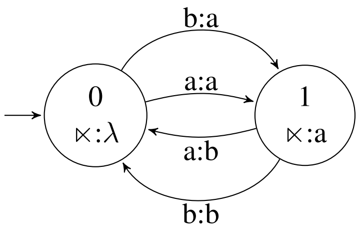


Figure 1: A simple finite-state transducer

The tier-based functions expanded upon in this paper are themselves extensions of the Strictly Local functions (Chandlee, 2014; Chandlee et al., 2014, 2015, 2018; Chandlee and Heinz, 2018). A Strictly k-Local (SL$_k$) FST operates according to a memory window with a fixed and finite maximum size k, the state labels acting as a record of this window's contents. When the window pays attention to the input string we say that the function is Input Strictly k-Local (ISL$_k$) and the label of the currently occupied state always corresponds to the most recent (up to) $k-1$ elements read. Similarly, when the window pays attention to the output string we say that the function is Output Strictly k-Local (OSL$_k$) and the label of the currently occupied state always corresponds to the most recent (up to) $k-1$ elements written. Input Tier-based Strictly k-Local (ITSL$_k$) and Output Tier-based Strictly k-Local (OTSL$_k$) FSTs operate exactly like their ISL$_k$ and OSL$_k$ cousins except that only a subset of the relevant alphabet (the function's *tier*) is allowed to occupy space in the memory window. Restricting the transducers attention in this manner permits the modelling of non-local processes where the distance between trigger and target can be arbitrarily large.

To demonstrate, consider the process of regressive sibilant harmony in Slovenian. The Slovenian process is optional, though we will assume for the purposes of this section that it is categorical, postponing a discussion of its optionality to Section 3. The Slovenian pattern causes a sibilant to become [−anterior] if it is followed at any distance by another [−anterior] sibilant unless a coronal stop intervenes (Jurgec, 2011, pp. 329-333). Examples of successful sibilant harmony are provided in (1a-b) and examples of blocked sibilant harmony are provided in (1c-d) with the second singular suffix acting as the potential trigger in each case.

(1) Slovenian sibilant harmony, blocking by coronal stops (Jurgec, 2011, pp. 330-331)

a.	/spi-ʃ/	[ʃpi-ʃ]	'sleep-2SG'
b.	/pozabi-ʃ/	[poʒabi-ʃ]	'forget-2SG'
c.	/stoji-ʃ/	[stoji-ʃ]	'stand-2SG'
d.	/zida-ʃ/	[zida-ʃ]	'build-2SG'

The transducer in Figure 2 shows what an idealized and mandatory version of the Slovenian pattern would look like as an OTSL$_2$ function operating relative to the tier {s, ʃ, z, ʒ, t, d}. Since the process is regressive, the machine reads input strings from right to left. State labels are enclosed in square brackets to highlight the fact that this transducer tracks the output string. To save on space, states with the same behaviour are collapsed into a single circle with multiple labels and transitions that share an origin and a destination are collapsed into a single arrow with multiple labels. Note that transitions labelled '*:*' represent an arbitrary non-tier segment mapping faithfully to itself. An input [+anterior] sibilant (i.e., /s/ or /z/) will map faithfully to itself if no tier-elements have been produced thus far, if the most recently produced tier element was a coronal stop, or if the most recently produced tier element was another [+anterior] sibilant. On the other hand, if the most recently produced tier element was a [−anterior] sibilant (i.e., [ʃ] or [ʒ]), an input [+anterior] sibilant will instead palatalise to become its [−anterior] equivalent. Palatalization will happen no matter how many non-tier elements (i.e., non-sibilants other than [t] or [d]) intervene between the [−anterior] trigger and the [+ anterior] target, because producing such an element never causes a change of state in this machine.

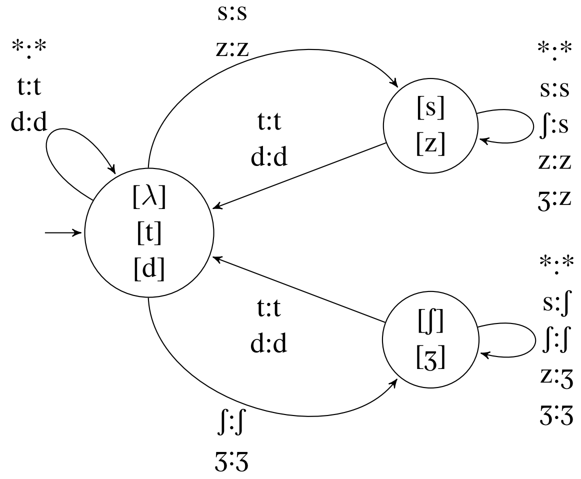


Figure 2: An OTSL$_2$ transducer that produces mandatory sibilant harmony

ITSL$_k$ and OTSL$_k$ functions are equipped with just one tier, which works well for many cases, but some patterns require multiple memory windows that each track a different tier. By allowing for multiple tiers in this way we delve into the class of Multi-Tiered Strictly k-Local functions (Burness and McMullin, 2020). All of the MTSL$_k$ transducers that will appear in this paper adhere to a restriction which Burness and McMullin (2020) call *target-specification*. The restriction states that (i) each input element is associated with a set of tiers that on their own can fully determine what the element is mapped to on a given step and (ii) this *target-specified* set of tiers must form a strict superset-subset hierarchy. Target-specified MTSL functions essentially track multiple, related sources of information when deciding how to process a particular input element. Tiers that are not part of a input element's specified set are ignored when reading that input element, since they provide either irrelevant or redundant information.

3 Probabilistic variants

In the previous section, we mentioned that the Slovenian process of sibilant harmony was optional. When the transducer in Figure 2 reads an input like /pozabiʃ/ from right to left, it will be in the [−anterior] state as it goes to read /z/, and the corresponding transition will enforce harmony. We want, however, to have the possibility of faithfully producing [z] for /z/ while in the [−anterior] state, since harmony is optional in Slovenian (Jurgec, 2011). We can create the possibility of optional faithfulness by adding transitions from the [−anterior] state to the [+anterior] state labelled 's:s' and 'z:z' and transitions from the [+anterior] state to the [−anterior] state labelled 'ʃ:ʃ' and 'ʒ:ʒ'. Figure 3 shows the resulting transducer, which aside from the added non-determinism, exhibits all the required behaviour of an OTSL$_2$ transducer (i.e., all transitions still land in the state corresponding to the most recently written tier element). For clarity, the harmony-enforcing transitions are shown as dotted lines and the harmony-ignoring (faithful) transitions are shown as dashed lines.

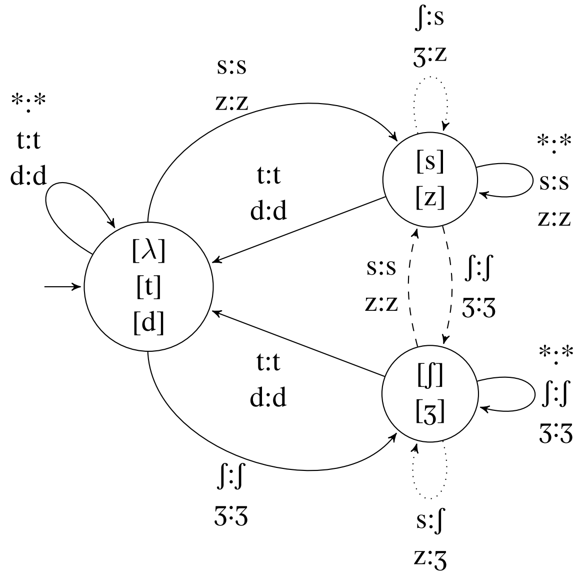


Figure 3: A quasi-OTSL$_2$ transducer that produces optional sibilant harmony

Now we have two transitions out of the [−anterior] state for the input /z/, one that enforces harmony and one that enforces faithfulness. Assuming that we choose randomly between the two available transitions, the /z/ in /pozabiʃ/ then has a 50% chance of harmonizing with the nearby [ʃ]. It is very important that the new faithful transition leads to the [+anterior] state rather than looping back to the [−anterior] state. This is because, while the new faithful transition does not produce harmony, it nonetheless produces a tier element. By ensuring that any additional transitions all lead to the state associated with the most recently produced tier element, we maximally preserve the intuitions of the TSL functions, even though we are abandoning determinism and thus no longer meet the definition of a TSL function. We instead have a *quasi*-TSL *relation*, where the set of possible outputs at a given step is directly determined by the most recently produced tier element.

Of course, we may want to achieve a rate of harmony higher than 50% while still allowing for the possibility of faithfulness. To do so, we can assign a numerical weight to each transition in the machine (Vidal et al., 2005a). When more than one transition could be followed at a given step in the derivation, the probability that we choose a given member from that set of transitions is proportional to its share of the summed weights of the set. For ease of interpretation, we assume that the weight of a transition is equal to the probability that it is followed, meaning that given a state q and an input symbol a, the weights of all transitions leaving q for the input a must add up to 1. Suppose now that the harmony-ignoring (dashed) transitions are weighted 0.2, the harmony-enforcing (dotted) transitions are weighted 0.8, and the remaining transitions are weighted 1. The input /sapozabiʃ/ has three possible outcomes whose probabilities sum to 1: a fully faithful candidate [sapozabiʃ], a single harmony candidate [sapoʒabiʃ], and a full harmony candidate [ʃapoʒabiʃ]. The fully faithful candidate has a probability of $0.2 * 1 = 0.2$ since the probability of the /z/ remaining faithful is 0.2, and if it does so, the /s/ is guaranteed to be faithful. The remaining output probabilities can be calculated in a similar manner: the single harmony candidate has a probability of $0.8 * 0.2 = 0.16$, and the full harmony candidate has a probability of $0.8 * 0.8 = 0.64$. More generally, a weighted transducer set up in the above manner produces a conditional distribution over a finite set of output strings for each possible input string. The number of output possibilities as well as their shape and share of the probability can of course change from input to input and from transducer to transducer, but the distributions so-defined are crucially related to and dictated by tier-based strict locality.

A particularly interesting case of optionality in a long-distance process comes from Bukusu. In this language, underlying /l/ becomes output [r] if the nearest leftward surface liquid is [r] (de Blois, 1975; Odden, 1994; Hansson, 2010). The pattern of liquid harmony affects the applicative suffix /-ila/, exemplified by the data in (2). The suffix's underlying /l/ surfaces faithfully when the root contains no liquids as in (2a) or when the only liquids in the root are all instances of /l/ as in (2b). When the base contains an /r/, though, the liquid in the applicative suffix alternates to obey harmony. This happens across a single vowel as in (2c) and at further distances as in (2d).

(2) Bukusu liquid harmony (Odden, 1994)

a.	xam-ila	'milk-APPL'
b.	lim-ila	'cultivate-APPL'
c.	kar-ira	'twist-APPL'
d.	rum-ira	'send-APPL'

Importantly, harmony is obligatory across a single vowel (i.e., in transvocalic contexts) but becomes optional at further distances (Hansson, 2010). For example, /ruk-ila/ 'plait-APPL' may surface as [ruk-ila] without harmony or as [ruk-ira] with harmony. Another long-distance pattern that is cited as being obligatory in transvocalic contexts but optional at further distances would be the sibilant harmony in Kinyarwanda (Kimenyi, 1979; Coupez, 1980; Hansson, 2010; Walker and Mpiranya, 2006; Walker et al., 2008). Such a transvocalic / beyond-transvocalic dichotomy is not possible to describe using a probabilistic quasi-OTSL$_2$ transducer as we did for Slovenian sibilant harmony above, but *is* possible to describe using a probabilistic quasi-OMTSL$_2$ transducer.

Consider the transducer in Figure 4, where 'V' stands for an arbitrary vowel and 'C' stands for an arbitrary non-liquid consonant. Ignoring the dashed transition for now (but including the dotted transitions), this machine represents a target-specified OMTSL$_2$ function that computes a fully obligatory version of the Bukusu pattern over a tier of liquid consonants $A = \{\text{r}, \text{l}\}$ and a tier of all consonants $B = \{\text{r}, \text{l}, \text{C}\}$. The left and right symbol of each state label correspond respectively to the suffix on A (i.e., the most recently produced liquid consonant) and B (i.e., the most recently produced consonant). Reaching the [r, r] state can be interpreted to mean that the most recent consonant we have seen is an [r] (since it is on both the liquid and consonantal tiers). Compare this to being in the [r, C] state, which means that the most recent consonant we have seen is a non-liquid, and that this consonant is preceded by an [r]. In a transducer that does not contain the dashed transition, both of these states enforce the harmonic /l/ → [r] change, as indicated by the dotted transitions. However, by adding the dashed transition, harmony becomes optional just in those cases where [r] is the most recently produced liquid but not the most recently produced consonant. In a language like Bukusu with mostly open CV syllables, these two states more-or-less reflect the difference between a transvocalic and beyond-transvocalic distance from the most recent liquid consonant. In particular it is the superset-subset relationship imposed onto the tierset by target specification (Burness and McMullin, 2020) that allows us to have harmony be obligatory across 0 non-liquid consonants and be optional across 1+ non-liquid consonants.

An important question arises when we model optional processes using probabilistic transducers. Namely, how do we determine the transition weights that best reflect the target pattern? This type of optimization problem is well-studied in the literature on Probabilistic Finite-state Acceptors (PFAs) which are exactly like probabilistic transducers except that rather than taking an input string and producing an output string, they take an input string and return a value reflecting the input's well-formedness. The Slovenian and Bukusu transducers above can be reinterpreted as acceptors if we think of their transition labels as atomic elements of an alphabet and rewrite input-output pairs as a string of such "transducer actions". Conveniently, reinterpreting the Slovenian and Bukusu transducers in this manner makes them deterministic since, while a given input string can follow potentially multiple paths through the transducers to produce different outputs, a given input-output pair can only be achieved by following a single, specific path through the transducers. Finding the transition weights for a given deterministic PFA that maximise the probability of a set of training data has a well-known, simple, and efficient solution. For each transition,[1] we calculate the number of times it was followed when reading the sample and divide this number by the total number of times its origin state was visited when reading the sample (Vidal et al., 2005a,b; de la Higuera, 2010). One small modification is needed for our purposes since the weights resulting from the above method will describe a single distribution over input-output pairs, rather than a separate distribution over output strings for each input. This is because the weights of all transitions out of a given state will sum to 1, whereas we want all transitions out of a given state *for a given input element* to sum to 1. To remedy this, we can normalize the transducer by taking each combination of state and input symbol, adding together the weights of all transitions leaving that state for that input element, then dividing each of the implicated transition weights by this sum.

4 Distance-based decay

The analyses of the Slovenian and Bukusu cases above are relatively simplistic in that the probability with which the process applies remains constant. In many cases, however, we see that the probability of application is inversely correlated to the distance between trigger and target. This phenomenon is known as *distance-based decay* (Zymet, 2015) and can be observed in Malagasy vowel rounding dissimilation (Zymet, 2015), Hungarian backness vowel harmony (Hayes and Londe, 2006; Hayes et al., 2009), Latin liquid dissimilation (Zymet, 2015), and Navajo sibilant harmony (Martin, 2005), among others. Current descriptions of the phenomenon are couched within stochastic constraint-based frameworks like Noisy Harmonic Grammar (Coetzee and Pater, 2011) and Maximum Entropy grammar (Goldwater and Johnson, 2003; Hayes and Wilson, 2008). These descriptions propose that the weight of a process-enforcing constraint is scaled down proportionally to the distance between trigger and target (Kimper, 2011; Zymet, 2015). Distant trigger-target pairs incur smaller penalties than more local pairs, and as a result, the process applies at a lower probability in the distant pair than in the more local pair (Kimper, 2011; Zymet, 2015). Focusing on Malagasy and Hungarian, we will show how distance-based decay can equally be captured through minor modifications

[1] The final strings associated to states are treated as transitions for the purposes of this optimization, effectively acting as a transitions that lead to a dedicated "stopping" state with no associated string of its own.

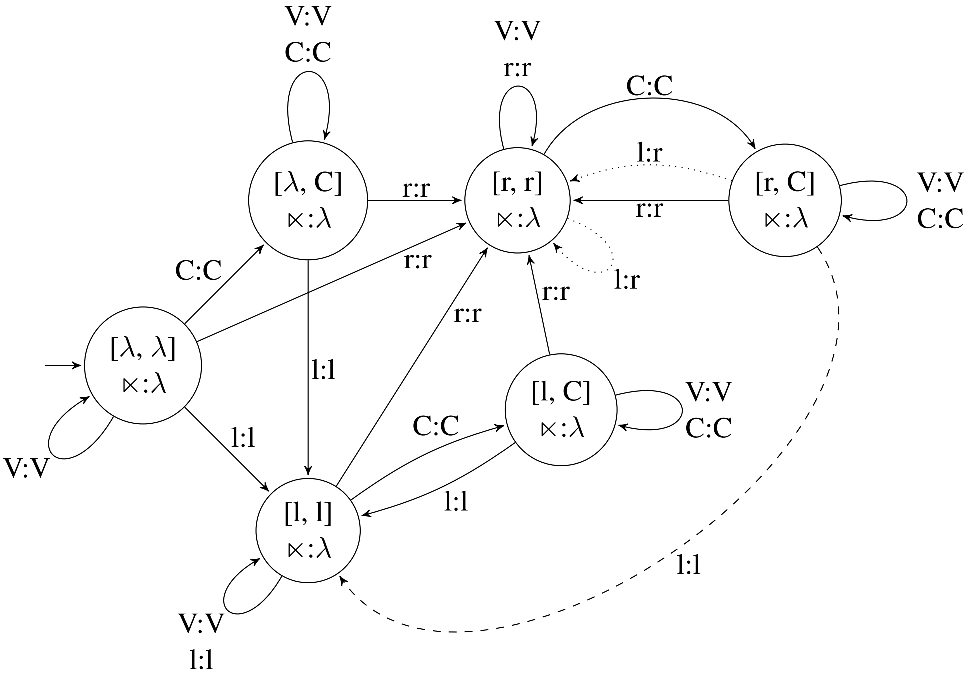


Figure 4: A quasi-OMTSL$_2$ transducer that computes Bukusu liquid harmony

to a TSL or MTSL transducer.

4.1 Malagasy

Malagasy has a process of vowel rounding dissimilation whereby the passive imperfective suffix /-u/ becomes [-i] when preceded by an [u], as can be seen in /babu-u/ → [babu-i] 'plunder-PASS.IMP'. Front vowels are opaque to the process as can be seen with /turi-u/ → [turi-u] 'preach-PASS.IMP' and /ure-u/ → [ure-u] 'massage-PASS.IMP'. In contrast, the vowel /a/ is transparent to dissimilation, as can be seen with /gurabah-u/ → [gurabah-i] 'splutter-PASS.IMP'. If we ignore its dashed transition (discussed further below), the OTSL$_2$ transducer in Figure 5 captures a mandatory version of the Malagasy pattern just described. Dissimilation specifically affects the passive imperative suffix rather than /u/ in general (Zymet, 2020), so for convenience we assume that this transducer only ever reads verb stems and adds the appropriate suffix allomorph upon reaching the end of the base.

Malagasy dissimilation is not categorical, however, and exhibits distance-based decay. According to Zymet's (2015) survey of de la Beaujardière's (2004) online Malagasy dictionary, the probability of dissimilation is 0.99 (989/993) when the trigger and target are in adjacent syllables, 0.51 (201/397)

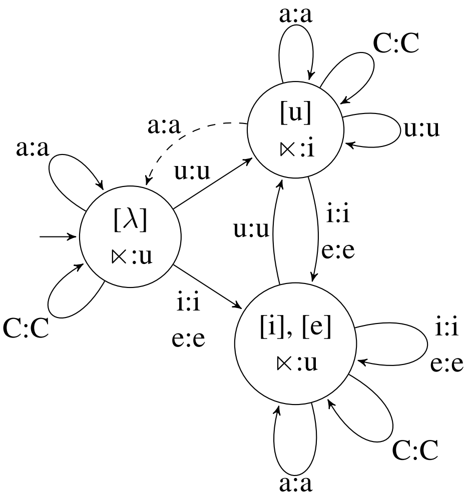


Figure 5: A quasi-OTSL$_2$ transducer that produces Malagasy dissimilation

when they are separated by one transparent syllable, 0.13 (4/32) when they are separated by two transparent syllables, and 0 (0/4) when they are separated by three transparent syllables. Due to the language's mostly open-syllable nature, the number of transparent syllables corresponds with how many transparent vowels fall between the trigger and target, and the probability of dissimilation is roughly $1/2^x$, where x represents the number of intervening transparent vowels. This opens an interesting route to deriving the distance-based decay using the structure of the transducer in Figure 5: whenever the transducer reads /a/ and produces [a] while in the [u] state, it has a roughly 50% chance to "forget" that it previously produced an instance of [u]. In this case, the transducer will follow the dashed transition which returns to the [λ] state instead of looping back to the [u] state, and will consequently fail to dissimilate the passive imperative suffix. As more transparent vowels are encountered while in the [u] state, the machine is exponentially less likely to remember that it encountered a dissimilation trigger, giving us the negative exponential curve in the probability of dissimilation. A cognitive interpretation of forgetful transitions would be that the memory of the most recent tier element decays over time.

Two related questions arise when modelling distance-based decay by augmenting a TSL transducer with forgetfulness parameters. First, how do we decide which forgetful transitions should be added to the TSL transducer? Any transition that fails to produce an element from the tier can presumably be given a forgetful version, but including too many or too few of these could negatively impact the accuracy of our model. Second, given a fixed set of forgetful transitions, how do we determine their optimal weights? We answer the latter question first, since its solution will be considered when approaching the former question.

Recall from Section 3 that to optimize the weights of a deterministic acceptor, it is sufficient to read through the provided sample once and count the number of times that each transition is followed. Unfortunately, even after reinterpreting a forgetful quasi-TSL transducer as an acceptor, it is still non-deterministic. Given an input-output pair we can generally tell whether forgetting did or did not occur, but we cannot tell exactly where the forgetting took place when there is a sequence of more than one transparent element. Because we cannot always know the exact path that an input-output pair followed through the machine, we cannot accurately count the number of times each transition get traversed when the sample is read. It is, however, possible to estimate these counts given the machine's current transition weights using what are called forward and backward probabilities. Consider the element x in the string $w = u \cdot x \cdot v$. For a transition labeled x leaving state q and landing in state q' we can calculate the probability that we are in state q after having read u (the forward probability) and the probability that we produce v when starting in state q' (the backward probability).[2] Multiplying the current weight of a given transition by its forward and backward probability and then dividing by the probability of the whole string gives us the probability that we actually traversed the transition on that reading step (de la Higuera, 2010, pp. 362-363).

By using estimated traversal probabilities as our traversal counts, we can calibrate the weights of a non-deterministic acceptor using the same division operations as for a deterministic acceptor. If we cycle through the estimation and calibration processes just described, the parameter weights will get adjusted by smaller and smaller amounts until they converge. This is known as the Baum-Welch algorithm,[3] originally developed by Baum et al. (1970) and Baum (1972). It is a type of maximum likelihood estimation (MLE) that, metaphorically, climbs the "hill" of sample probabilities by adjusting the available parameter values, and stops when it reaches a peak and cannot increase the sample probability any further.

The algorithm is guaranteed to converge on such an optimum, but non-deterministic machines can have multiple optima in addition to the global optimum, and the algorithm may get trapped in one of these (Vidal et al., 2005b; de la Higuera, 2010). Returning to the hill metaphor, there can be multiple peaks of varying heights and we want the algorithm to find the highest one, but it cannot tell whether the peak it reaches is actually the highest, it simply stops once it finds *any* peak. The only guaranteed way around this is to try several times with different starting values, and then pick the result that gives the best probability, in the hope that the chosen iteration found the global optimum (de la Higuera, 2010, p. 323). Luckily, this was not a serious issue during the tests described further below. Whenever a transducer needed optimizing, the optimization process was run several times with random initializations of the transducer's transition weights, and each machine always achieved the same approximate log-likelihood no matter its initialization, suggesting that (at least in these cases) there were no local optima in which the optimizer

[2] Chapter 5 of de la Higuera (2010) shows how to efficiently calculate forward and backward probabilities.

[3] See chapter 17 of de la Higuera (2010) for a more thorough presentation.

could get trapped. The results reported for each machine below are relative to the optimization that achieved the best log-likelihood.

Moving on to the question of which forgetful transitions to include, we could take the stance that we want only the forgetful transitions that significantly affect model performance. So long as the set of forgetful transitions in one optimized transducer are a strict superset of the forgetful transitions in another optimized transducer, it is in theory possible to perform a log-likelihood ratio test to assess whether the additional transitions significantly improve model performance. Given a calibrated transducer, we calculate its log-likelihood by running the training sample through it. For each input-output pair (x, y) we calculate the probability that the machine produces y given x, which is equal to the sum of the probability of all paths through the machine that produce y given x. This might seem difficult to do efficiently since the number of possible paths through a non-deterministic transducer is in the worst-case exponentially proportional to the length of the input string, but we can bypass this issue by calculating forward probabilities (which takes just one pass through the string) and summing over those instead (de la Higuera, 2010, pp. 90-92). If we then take the log of each pair's probability, adding them up gives us the model's log-likelihood. One way to find the best set of forgetful transitions, then, would be to take a forwards selection approach. Starting with no forgetful parameters, we iteratively add the one that would contribute the most until we cannot significantly improve our model any more.

To test the effectiveness of the FST decay model, we created custom Python code that implements the Baum-Welch algorithm and log-likelihood calculation procedures described above, then ran it against the Malagasy data from Zymet (2015). Calibrating the base model affects only the probability that dissimilation occurs while in the [u] state, and the optimal value of 83.73% gives a log-likelihood of -633.30. Most additional forgetful transitions significantly improved model fit on their own,[4] but 'a' had by far the strongest contribution, increasing log-likelihood all the way to -315.34 ($\chi^2_1 = 635.93, p = 2.57 \times 10^{-140}$). The next highest contribution came from 'l', which increased log-likelihood to -609.67 ($\chi^2_1 = 47.27$, $p = 6.2 \times 10^{-12}$). A second round of tests using the 'a' model only found two additional forgetful transitions to be significant: these were 'dʒ' ($\chi^2_1 = 3.85, p = 0.050$) and 'z' ($\chi^2_1 = 3.93, p = 0.048$). This might seem odd considering how most were highly significant on the first round of tests. Looking at the largely CV syllable structure of Malagasy, though, the presence of an intervening 'a' heavily implies the presence of an intervening consonant. Significant contributions from the lone consonantal parameters may thus have been indirect inheritances from instances of 'a'. Because forgetful 'dʒ' and 'z' transitions are just barely significant given a threshold of $p < 0.05$, we opted not to include either and stop further testing, leaving us with just a forgetful 'a' transition. One reason that 'a' may have near-exclusive entitlement to a forgetful transition is its high similarity to the tier elements, all of which are vowels. Encountering a non-tier element that is highly similar to elements on the tier would intuitively interfere with the maintenance of a tier suffix in long-term memory, although we leave the confirmation of this hypothesis for future research.

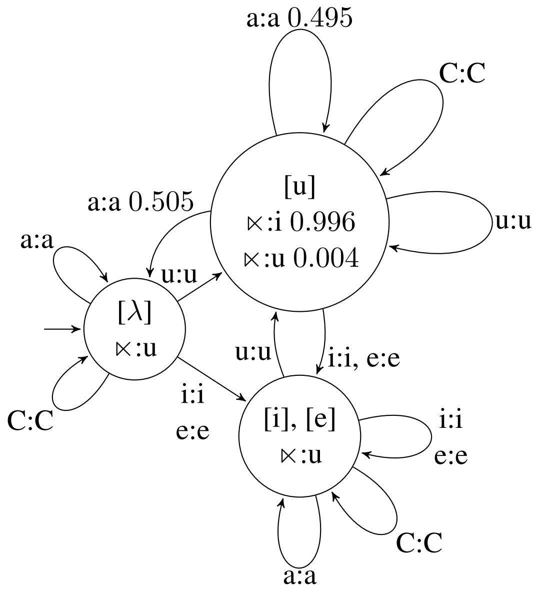


Figure 6: An optimized quasi-OTSL$_2$ transducer for Malagasy

The optimized Malagasy transducer is shown in Figure 6; all transitions without a displayed weight have a weight of 1. Earlier we mentioned that, as reported by (Zymet, 2015), the probability of dissimilation is $0.996(989/993)$ when the trigger and

[4] Only 'v' ($\chi^2_1 = 3.58, p = 0.06$), 't' ($\chi^2_1 = 2.66, p = 0.1$), 'f' ($\chi^2_1 = 0.49, p = 0.48$) and 'h' ($\chi^2_1 = 0, p = 1$) did not. The last case is particularly interesting in that the optimal weight for a lone forgetful 'h' transition was 0, equivalent to the absence of such a transition.

target are in adjacent syllables, $0.506(201/397)$ when they are separated by one transparent syllable, $0.125(4/32)$ when they are separated by two transparent syllables, and $0.00(0/4)$ when they are separated by three transparent syllables. A single forgetful transition for [a] pretty faithfully reproduces the probability of adjacent dissimilation (0.996) and dissimilation across one transparent syllable $(0.996 * 0.495 = 0.493)$, but modestly overestimates the probability of dissimilation across two intervening syllables $(0.996 * 0.495^2 = 0.244)$ and three intervening syllables $(0.996 * 0.495^3 = 0.121)$. Zymet's (2015) constraint-based model more closely reproduces the latter two probabilities, but this may be an instance of overfitting, considering how few forms in the corpus contain 2+ intervening syllables. In any case, the model with a forgetful [a] transition drastically outperforms the base model, which predicts dissimilation with a probability of 0.837 at any distance.

4.2 Hungarian

Including forgetfulness parameters into a single-tiered transducer is sufficient for the Malagasy case, but not all cases of distance-based decay are so easy. Take for instance the backness vowel harmony in Hungarian, to which [i], [e], and [ɛ] are transparent (Hayes and Londe, 2006; Hayes et al., 2009; Kimper, 2011; Ozburn, 2019). While it is generally true that a higher number of transparent vowels between trigger and target will exponentially diminish the probability of harmony, there is an important exception: harmony remains nearly obligatory across a single transparent vowel (Hayes and Londe, 2006; Hayes et al., 2009; Kimper, 2011; Ozburn, 2019). For example, the [ɔ] in [pɔpiːr] 'paper' always triggers the back variant of the dative suffix ([pɔpiːr-nɔk] 'paper-DAT') since it is followed by only one transparent vowel, but the [ɔ] in [ɔspirin] 'aspirin' only optionally triggers the back variant since it is followed by two transparent vowels ([ɔspirin-nɔk] ∼ [ɔspirin-nɛk] 'aspirin-DAT').

This is impossible to model using a single-tiered transducer, even with forgetful transitions. To see why, consider the transducer fragment in Figure 7, which determines the appropriate allomorph of the dative suffix /-nEk/ for bases containing only high vowels.[5] The underspecified suffix vowel /E/ must harmonize while in either the /u/ or /y/ state, and defaults to front while in the [λ] state. The vowel /i/ is transparent to harmony and so each of these states has a looping transition labelled 'i:i'. We could try modelling the distance-based decay using the forgetful transitions marked with dashed lines, but these do not distinguish between having one transparent vowel and having two or more transparent vowels between trigger and target. We want forgetfulness to begin applying only in the latter case, but there is no way to set such a threshold on the required number of transparent segments in a single-tiered transducer. In such a transducer, a transparent segment either always or never has the opportunity to cause forgetfulness.

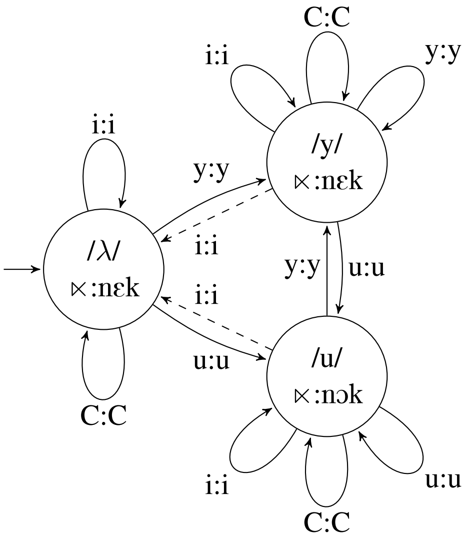


Figure 7: A quasi-ITSL$_2$ transducer fragment that enforces Hungarian suffixal harmony

Interestingly, the desired 2+ threshold can be modelled using a multi-tiered transducer as the base. Suppose we have one tier V that tracks all vowels (i.e., including the transparent [i], [e], and [ɛ]), and another tier H that tracks only the harmony-triggering vowels (i.e., excluding the transparent [i], [e], and [ɛ]). This allows us to distinguish cases where the most recently produced vowel is a harmony trigger (i.e., the suffixes on V and H coincide) and cases where the most recently produced vowel is transparent but is itself preceded by a harmony trigger (i.e., the suffixes on V and H do not coincide). These two cases constitute two different states in the corresponding multi-tiered transducer, and by ensuring that forgetful transitions only originate from states where the two tier suffixes do not coincide, harmony will be obligatory across a single transparent vowel, although any additional transparent vowels will cause distance-

[5] We are assuming here for simplicity that harmony only affects underspecified suffix vowels, and that all base-internal vowels are fully specified in underlying forms.

based decay.

Consider now the transducer fragment in Table 1, which again determines the appropriate allomorph of the dative suffix /-nEk/ for bases containing only high vowels. Drawing the transducer in a legible manner is tricky, so we have opted to represent it as a collection of tables where each row corresponds to a transition. Rows are organized according to their origin state, and a label /a, b/ corresponds to having /a/ as the suffix on H (the tier of harmonic vowels) and /b/ as the suffix on V (the tier of all vowels). Notice how the /u,u/ and /u,i/ states both enforce harmony, but only the latter has a forgetful transition labelled 'i:i' (i.e. it has two rows for input /i/ in the table). Being in the /u,u/ state means that we have not read any transparent vowels after reading the most recent harmony-triggering vowel, while being in the /u,i/ state means that we have read *at least one* transparent vowel after reading the most recent harmony-triggering vowel. The transition labelled 'i:i' leaving /u,u/ and landing in /u,i/ does not have a forgetful counterpart, and so harmony remains obligatory across this one transparent vowel. The transition labelled 'i:i' leaving /u,i/ and looping back to /u,i/ does, however, have a forgetful counterpart. There is thus a chance that reading a second transparent vowel (and third, and fourth, etc.) will cause us to forget having read /u/. Forgetting that we read /u/ will bring us to the /λ,i/ state, which corresponds to thinking that we have not read a harmony trigger yet (or at the very least, not remembering the identity of the most recent harmony trigger). The decaying probability of harmony then results from the fact that it is increasingly unlikely to remain in the harmony-enforcing /u,i/ state as we read more and more transparent vowels.

Hayes and Londe (2006) and Hayes et al. (2009) collected a corpus of Hungarian noun bases (available at `https://linguistics.ucla.edu/people/hayes/HungarianVH/index.htm`), marking the percentage of times each base appears with the back versus front allomorph of the dative suffix (determined using Google search results). To ascertain whether and how much an MTSL model of decay outperforms a TSL model, we optimized a TSL-like transducer and an MTSL-like transducer against this corpus using a modified Baum-Welch algorithm. Unlike for the Malagasy tests above, we are not trying to maximize the probability of each datum in the sample, but trying to replicate the indicated probability of each datum as closely as possible. Each 'base + allomorph' combination with greater than 0 probability was thus treated as a separate datum, and the amount contributed by a reading step in that datum to a transition's estimated traversal count was multiplied by the datum's probability. Essentially, this would treat a training datum with an indicated probability of, for example, 0.75 as being 75% of a datum (i.e., a datum that was observed 0.75 times). Consequently, the optimization procedure is maximizing a weighted version of model log-likelihood rather than the regular log-likelihood. Where $P(o \mid i)$ is the probability of the input-output pair (i, o) as indicated in the sample S, and where $\hat{P}(o \mid i)$ is the probability of the input-output pair (i, o) predicted by the model M, regular and weighted log-likelihood can be expressed as in (3). There were only ever up to two output possibilities o_1 and o_2 for a given input string i,[6] and their observed probabilities $P(o_1 \mid i)$ and $P(o_2 \mid i)$ always summed to 1, as did their predicted probabilities $\hat{P}(o_1 \mid i)$ and $\hat{P}(o_2 \mid i)$. Maximizing the weighted log-likelihood ensures that we are on average minimizing the distance between the points $\langle \hat{P}(o_1 \mid i),\ \hat{P}(o_2 \mid i)\rangle$ and $\langle P(o_1 \mid i),\ P(o_2 \mid i)\rangle$ for the input strings in the sample.

(3) Regular model log-likelihood:

$$L(M \mid S) = \sum_{(i,o)\in S} \log(\hat{P}(o \mid i))$$

Weighted model log-likelihood:

$$L_W(M \mid S) = \sum_{(i,o)\in S} \log(\hat{P}(o \mid i)) \cdot P(o \mid i)$$

The TSL-like transducer had three states: /λ/ when there was no known preceding harmonic vowel, /F/ when the most recent harmonic vowel was front, and /B/ when the most recent harmonic vowel was back. It had forgetful transitions for /iː/, /i/, /e/, and /ɛ/ leading to the /λ/ state out of the /B/ state. The /F/ state had no forgetful transitions since Hayes and Londe (2006) found that transparent vowels never block front harmonic vowels from imposing a front allomorph. The MTSL-like transducer had the 7 states listed in (4). The states /Biː/, /Bi/, /Be/, and /Bɛ/ were kept separate, rather than having a single 'back + transparent' state since

[6] The one with the front allomorph of the dative suffix and the one with the back allomorph of the dative suffix.

Origin	Input	Output	Landing	Origin	Input	Output	Landing
/λ,λ/	/C/	[C]	/λ,λ/	/λ,i/	/C/	[C]	/λ,i/
/λ,λ/	/i/	[i]	/λ,i/	/λ,i/	/i/	[i]	/λ,i/
/λ,λ/	/y/	[y]	/y,y/	/λ,i/	/y/	[y]	/y,y/
/λ,λ/	/u/	[u]	/u,u/	/λ,i/	/u/	[u]	/u,u/
/λ,λ/	/⋉/	[nɛk]	NA	/λ,i/	/⋉/	[nɛk]	NA
Origin	Input	Output	Landing	Origin	Input	Output	Landing
/y,y/	/C/	[C]	/y,y/	/u,u/	/C/	[C]	/u,u/
/y,y/	/i/	[i]	/y,i/	/u,u/	/i/	[i]	/u,i/
/y,y/	/y/	[y]	/y,y/	/u,u/	/y/	[y]	/y,y/
/y,y/	/u/	[u]	/u,u/	/u,u/	/u/	[u]	/u,u/
/y,y/	/⋉/	[nɛk]	NA	/u,u/	/⋉/	[nɔk]	NA
Origin	Input	Output	Landing	Origin	Input	Output	Landing
/y,i/	/C/	[C]	/y,i/	/u,i/	/C/	[C]	/u,i/
/y,i/	/i/	[i]	/y,i/	/u,i/	/i/	[i]	/u,i/
/y,i/	/i/	[i]	/λ,i/	/u,i/	/i/	[i]	/λ,i/
/y,i/	/y/	[y]	/y,y/	/u,i/	/y/	[y]	/y,y/
/y,i/	/u/	[u]	/u,u/	/u,i/	/u/	[u]	/u,u/
/y,i/	/⋉/	[nɛk]	NA	/u,i/	/⋉/	[nɔk]	NA

Table 1: A quasi-IMTSL$_2$ transducer fragment that enforces Hungarian suffixal harmony

Hayes and Londe (2006) found that the height of the most recent transparent vowel has a significant effect on the probability of a back allomorph. Each of the states /Biː/, /Bi/, /Be/, and /Bɛ/ had forgetful transitions for /iː/, /i/, /e/, and /ɛ/ leading to the state /N/.

(4) States in the MTSL-like Hungarian transducer

- /λ/ = no preceding vowel OR the most recent vowel is transparent with no known preceding harmonic vowel
- /F/ = the most recent harmonic vowel is front
- /BB/ = the most recent vowel is back
- /Biː/ = the most recent harmonic vowel is back but the most recent vowel is iː
- /Bi/ = the most recent harmonic vowel is back but the most recent vowel is i
- /Beː/ = the most recent harmonic vowel is back but the most recent vowel is e
- /Bɛ/ = the most recent harmonic vowel is back but the most recent vowel is ɛ

Unfortunately, a log-likelihood ratio test cannot compare the two models because their parameters are not strictly nested; none of their forgetful transitions originate from equivalent states. Accordingly, their relative performance was assessed using their Akaike Information Criterion (AIC). Lower AIC values are preferred, and a model's AIC is equal to 2 times its number of free parameters minus 2 times its log-likelihood. The TSL-like model had a weighted log-likelihood of -266.90 and had 7 free parameters, so its AIC is 547.8. For its part, the MTSL-like model had a weighted log-likelihood of -249.73 and had 23 free parameters, so its AIC is 545.46. Going from the TSL model to the MTSL model reduces AIC by 2.34, which is not substantial, but nonetheless favours the MTSL model.

A reviewer points out that the closely related Bayesian Information Criterion (BIC) will likely heavily favor the TSL model as opposed to the MTSL model, even though both criteria tend to favour the same models in practice. The BIC more harshly penalizes parameter count: it is obtained by multiplying number of free parameters by the natural logarithm of the sample size and then subtracting 2 times the log-likelihood. There were 9427 input-output pairs in the training sample, so the TSL model has a BIC of 597.86 and the MTSL model has a BIC of 709.94; the reviewer thus correctly speculates that BIC vastly prefers the TSL model over the MTSL one. We are unsure of how to reconcile the discrepancy between the opposite preferences of AIC and BIC in this case, but we lean towards siding with the BIC's preference since it is much stronger. Nevertheless, it is worth examining where the MTSL model's higher accuracy comes from, since it may be worthwhile in other cases.

Visual inspection of the weights in the optimized transducers suggests that the greater accuracy of the MTSL model comes from the fact that it can distinguish between spans of 1 versus $2+$ transparent vowels, a distinction not possible in the TSL model. For example, the MTSL model assigns a probability of about $0.97 * 0.55 = 0.53$ to the form [ɔspirin-nɔk] 'aspirin-DAT' since the probability of harmony while in the state /Bi/ (i.e., across a single instance of /i/) is about 0.97 and the probability of /i/ not causing forgetfulness out of the state /Bi/ is about 0.55. Compare this to the TSL transducer which assigns the same form a probability of about $0.99 * 0.93^2 = 0.86$ since the probability of harmony while in the state /B/ is about 0.99 and the probability of /i/ not causing forgetfulness out of the state /B/ is about 0.93. The actually observed frequency of harmony for this noun is 0.21 and so the MTSL transducer, while still a fair ways off, is much closer than the TSL transducer.

Consistent with this interpretation of the models' differing performance, Table 2 compares the observed average probability of harmony against the average probabilities predicted by the TSL and MTSL models, broken down by the number of intervening transparent syllables. Taking every noun for which back a back allomorph is possible (i.e., whose rightmost harmonic vowel is back), we find that adjacent harmony has an average probability of 0.99 (5317 eligible nouns), harmony across a single transparent vowel has an average probability of 0.67 (370 eligible nouns), harmony across two transparent vowels has an average probability of 0.18 (63 eligible nouns) and harmony across three transparent vowels has an average probability of 0.00 (8 eligible nouns). In particular, we see that the TSL model overestimates the probability of harmony across two transparent vowels, whereas the MTSL model closely matches the observed probabilities in all cases.

Transparent syllables	Observed average	TSL average	MTSL average
0	0.99	0.99	0.99
1	0.67	0.64	0.67
2	0.18	0.33	0.19
3	0.00	0.03	0.01

Table 2: Average probability of Hungarian harmony by number of transparent syllables

Interestingly, both the trained TSL model and the trained MTSL model reproduce an additional aspect of Hungarian harmony whereby vowel height gradiently affects the degree to which a front unrounded vowel is transparent. Specifically, lower front unrounded vowels are "less transparent" than higher front unrounded vowels (Hayes and Londe, 2006; Hayes et al., 2009; Kimper, 2011; Rebrus and Törkenczy, 2016; Ozburn, 2019). In the optimized TSL model, the forgetfulness parameters out of state /B/ for /i/, /iː/, /eː/, and /ɛ/ are weighted 0.065, 0.11, 0.23, and 0.91 respectively, so lower vowels cause more forgetfulness than higher vowels. In the optimized MTSL model, the probability of appending the back allomorph upon ending in the /Bi/, /Biː/, /Beː/ and /Bɛ/ states is respectively 0.97, 0.99, 0.80, and 0.11, so backness harmony is more likely across a higher vowel than a lower vowel. The same height effect is also somewhat apparent in the MTSL model's forgetfulness parameters: the parameters for /iː/, /eː/, and /ɛ/ have average weights of 0.51, 0.78, and 0.91 respectively, mimicking the trend in the TSL model's forgetfulness parameters. The mimicking is not perfect, however, as the average forgetful weight for the vowel /i/ is unexpectedly high at 0.8. This mismatch could perhaps be because back vowels are only uncommonly followed by a chain of multiple front unrounded vowels, making the weights of the MTSL model's forgetfulness parameters a less reliable reflection of the height effect. Indeed, there were cases where the forgetful and non-forgetful transitions for the same vowel out of the same state both had a weight of 0, meaning that neither transition was ever crossed by the sample. These transitions effectively do not exist and were not considered when calculating the average weights.

5 Conclusion

In the existing computational work that models long-distance phonological processes using tiers, there is the tacit and convenient assumption that the processes are an all or nothing affair, but long-distance processes are often optional in reality. We showed here that probabilistic tier-based transducers with a structure similar to that of their categorical counterparts can capture gradient application while maintaining the relative computational simplicity afforded to us by tier-based strict locality.

In particular, we demonstrated that distance-based decay can be modeled by strategically adding duplicate non-tier transitions that make the transducer forget the identity of the most recent tier-element, in a sense causing the machine's memory to deteriorate over time. Using established techniques for optimizing the weights of probabilistic automata, we found that these models performed well relative to two real-language data sets. The Malagasy case study suggested that the presence and strength of forgetful transitions might be tied to similarity. For its part, he Hungarian case study showed that an MTSL model can distinguish between spans of 1 versus 2+ transparent elements, which may be a useful ability for some patterns. Finally, it should be said that all of the simulations presented above assume that the necessary tiers are known in advance, although phonological learning ideally involves as little *a priori* knowledge as possible. Methods exist for learning the tier of a TSL_2 function efficiently from positive data (Burness and McMullin, 2019), although it remains to be seen whether they can be adapted to probabilistic cases.

References

Samuel Andersson, Hossep Dolatian, and Yiding Hao. 2020. Computing vowel harmony: The generative capacity of search and copy. In *Proceedings of the 2019 Annual Meeting on Phonology*.

Leonard E. Baum. 1972. An inequality and associated maximization technique occurring in the statistical estimation for probabilistic functions of Markov processes. *Inequalities*, 3:1–8.

Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. 1970. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The Annals of Mathematical Statistics*, 41:164–171.

Phillip Burness and Kevin McMullin. 2019. Efficient learning of Output Tier-Based Strictly 2-Local functions. In *Proceedings of the 16th Meeting on the Mathematics of Language*, pages 78–90. Association for Computational Linguistics.

Phillip Burness and Kevin McMullin. 2020. Multi-tiered strictly local functions. In *Proceedings of the 17th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 245–255. Association for Computational Linguistics.

Jane Chandlee. 2014. *Strictly Local Phonological Processes*. Doctoral dissertation, University of Delaware.

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. 2014. Learning Strictly Local subsequential functions. *Transactions of the Association for Computational Linguistics*, 2:491–503.

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. 2015. Output Strictly Local functions. In *Proceedings of the 14th Meeting on the Mathematics of Language (MOL 2015)*, pages 112–125.

Jane Chandlee and Jeffrey Heinz. 2018. Strict Locality and phonological maps. *Linguistic Inquiry*, 49:23–60.

Jane Chandlee, Jeffrey Heinz, and Adam Jardine. 2018. Input strictly local opaque maps. *Phonology*, 35:171–205.

Andries Coetzee and Joe Pater. 2011. The place of variation in phonological theory. In John Goldsmith, Jason Riggle, and Jason Yu, editors, *The Handbook of Phonological Theory*, 2nd edition, pages 401–431. Blackwell.

André Coupez. 1980. *Abrège de Grammaire Rwanda*. Institut National de Recherche Scientifique, Butare.

Kornelis F. de Blois. 1975. *Bukusu Generative Phonology an Aspects of Bantu Structure*. Number 85 in Annales. Musée Royal de l'Afrique Centrale, Tervuren.

Jean-Marie de la Beaujardière. 2004. Malagasy dictionary and encyclopedia of Madagascar.

Colin de la Higuera. 2010. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, New York.

John Goldsmith. 1976. *Autosegmental Phonology*. Ph.D. thesis, MIT.

Sharon Goldwater and Mark Johnson. 2003. Learning OT constraint rankings using a maximum entropy model. In *Proceedings of the Workshop on Variation within Optimality Theory*, pages 111–120.

Gunnar Ólafur Hansson. 2010. *Consonant Harmony: Long-Distance Interaction in Phonology*. Number 145 in University of California Publications in Linguistics. University of California Press, Berkeley, CA.

Yiding Hao and Samuel Andersson. 2019. Unbounded stress in subregular phonology. In *Proceedings of the 16th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology and Morphology*, pages 135–143, Florence, Italy. Association for Computational Linguistics.

Yiding Hao and Dustin Bowers. 2019. Action-sensitive phonological dependencies. In *Proceedings of the 16th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology and Morphology*, pages 218–228, Florence, Italy. Association for Computational Linguistics.

Bruce Hayes and Zsuzsa Londe. 2006. Stochastic phonological knowledge: The case of Hungarian vowel harmony. *Phonology*, 23:59–104.

Bruce Hayes and Colin Wilson. 2008. A maximum entropy model of phonotactics and phonological learning. *Linguistic Inquiry*, 39:379–440.

Bruce Hayes, Kie Zuraw, Peter Siptar, and Zsuzsa Londe. 2009. Natural and unnatural constraints in Hungarian vowel harmony. *Language*, 85:822–863.

Peter Jurgec. 2011. *Feature Spreading 2.0: A Unified Theory of Assimilation*. Doctoral dissertation, University of Tromso.

Alexandre Kimenyi. 1979. *Studies in Kinyarwanda and Bantu Phonology*. Linguistic Research, Carbondale, IL.

Wendell A. Kimper. 2011. *Competing Triggers: Transparency and Opacity in Vowel Harmony*. Doctoral dissertation, University of Massachusetts Amherst.

Andrew Martin. 2005. *The Effects of Distance on Lexical Bias: Sibilant Harmony in Navajo Compounds*. Master's thesis, University of California, Los Angeles.

David Odden. 1994. Adjacency parameters in phonology. *Language*, 70:289–330.

Avery Ozburn. 2019. A target-oriented approach to neutrality in vowel harmony: Evidence from Hungarian. *Glossa*, 4:1–36.

Péter Rebrus and Miklós Törkenczy. 2016. A non-cumulative pattern in vowel harmony: A frequency-based account. In *Proceedings of the 2015 Annual Meeting on Phonology*.

Enrique Vidal, Franck Tollard, Colin de la Higuera, Francisco Casacuberta, and Rafael Carrasco. 2005a. Probabilistic finite-state machines - Part I. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1013–1025.

Enrique Vidal, Franck Tollard, Colin de la Higuera, Francisco Casacuberta, and Rafael Carrasco. 2005b. Probabilistic finite-state machines - Part II. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1026–1039.

Rachel Walker, Dani Byrd, and Fidèle Mpiranya. 2008. An articulatory view of Kinyarwanda coronal harmony. *Phonology*, 25:499–535.

Rachel Walker and Fidèle Mpiranya. 2006. On triggers and opacity in coronal harmony. In *Proceedings of the 31stth Annual Meeting of the Berkeley Linguistics Society*, University of California, Berkeley.

Jesse Zymet. 2015. Distance-based decay in long-distance phonological processes. In *Proceedings of the 32nd West Coast Conference on Formal Linguistics*, pages 72–81, Sommerville, MA. Cascadilla Press.

Jesse Zymet. 2020. Malagasy ocp targets a single affix: Implications for morphosyntactic generalization in learning. *Linguistic Inquiry*, 51:624–634.

Embedding Intensional Semantics into Inquisitive Semantics

Philippe de Groote
LORIA, UMR 7503
Université de Lorraine, CNRS, Inria
54000 Nancy, France
philippe.degroote@loria.fr

Valentin D. Richard
LORIA, UMR 7503
Université de Lorraine, CNRS, Inria
54000 Nancy, France
valentin.richard@loria.fr

Abstract

Ciardelli, Roelofsen, and Theiler (2017) have shown how a Montague-like semantic framework based on inquisitive logic allows for a uniform compositional treatment of both declarative and interrogative constructs. In this setting, a natural question is the one of the relation between the intensional and the inquisitive interpretation of a declarative sentence. We tackle this problem by defining an embedding of intensional semantics into inquisitive semantics, in the spirit of de Groote's and Kanazawa's (2013) intensionalization procedure. We show that the resulting *inquisitivation* procedure preserves intensional validity and entailment.

1 Introduction

Inquisitive semantics (Ciardelli et al., 2013, 2018) provides a semantic framework for analysing the information conveyed by linguistic utterances. It is based on a formal notion of *issue* that is reminiscent of alternative semantics and that allows several linguistic constructs to be assigned a meaning. In particular, it offers a uniform treatment of both declarative and interrogative forms.

Taking advantage of this new semantic framework, Ciardelli, Roelofsen, and Theiler (2017) have introduced a typed inquisitive logic, based on the simply typed λ-calculus, that can be used to provide a compositional semantics to fragments of language that contain interrogative constructs. This opens the door to Montague grammars based on inquisitive logic, but raises the question of the relation between the intensional and the inquisitive interpretation of a declarative utterance. If one sticks to the case of first-order logic, the question can be easily settled. This, unfortunately, is not sufficient. Indeed, a Montague grammar typically contains higher-order constructs, as in the following example that might correspond to the lexical semantic of the word *seek*:

$$\lambda os.\, s\,(\lambda x.\, \mathbf{try}\, x\,(\lambda x.\, o\,(\lambda y.\, \mathbf{find}\, x\, y)))$$

In the above λ-term, constant $\mathbf{try}$ is assigned the following type:

$$\mathbf{e} \to (\mathbf{e} \to \mathbf{s} \to \mathbf{t}) \to \mathbf{s} \to \mathbf{t}$$

Then, the question we must answer is the following one: which inquisitive interpretation should we assign to constant $\mathbf{try}$ so that the intended meaning of *seek* is preserved?

In order to solve this problem, we propose an *inquisitivation* procedure akin to de Groote's and Kanazawa's (2013) intensionalization. This procedure is based on an embedding of the intensional interpretations of the types into their inquisitive interpretations. We then prove that our inquisitivation procedure is adequate in the sense that it preserves validity and entailment.

The rest of our paper is organized as follows:

- Section 2 contains a brief introduction to inquisitive semantics.
- In Section 3, we present the necessary mathematical preliminaries, and we fix the type-theoretic setting in which we are working. In particular, we remind one of the definition of the simply typed λ-calculus, and we give its intensional interpretation.
- In Section 4, we define an embedding of intensional semantics into inquisitive semantics, and we provide the simply typed λ-calculus, with an inquisitive interpretation.
- Section 5, contains the proof that our inquisitivation procedure preserves validity and entailment.
- In Section 6, we compare the inquisitive interpretations of the logical connectives, as defined in

Proceedings of the 17th Meeting on the Mathematics of Language, pages 64–75, Montpellier (online),
December 13, 2021. © 2021 Association for Computational Linguistics

inquisitive semantics, with their inquisitive interpretations, as resulting from the inquisitivation procedure. We then propose a syntactic translation that allows inquisitive logic to be used as object language.

- In Section 7, we discuss briefly the inquisitivation of modal operators.

2 Inquisitive semantics

Montague semantics, in its original formulation (Montague, 1970, 1973), is only concerned with declarative sentences. To remedy this situation, Hamblin (1973) introduced alternative semantics, which allows for a treatment of interrogative sentences. Hamblin's idea is to interpret a question as the set of its possible answers. Hence, if an answer is modeled by a proposition, a question, in turn, must be modeled by a set of propositions. At the semantic level, a proposition being interpreted as a set s of possible worlds, a question is then interpreted as a set of sets of possible worlds.

This different treatment of declarative vs. interrogative propositions has some disadvantages though. Traditional set-theoretic operations cannot be used on interrogative propositions to interpret common semantic coordination, e.g. conjunction. Moreover, alternatives fail to predict even basic declarative entailments such as *John walks* $\models$ *John moves* (Groenendijk and Stokhof, 1984).

Inquisitive semantics (Ciardelli et al., 2013, 2018) elaborates on the idea of alternative semantics and circumvents some of its drawbacks.

Technically, in inquisitive logic, a *proposition* (also known as an *issue*) is defined to be a non-empty set of sets of possible worlds that is downward-closed with respect to set inclusion. As a consequence, conjunction, disjunction, and entailment can be defined in a standard way, i.e., as intersection, union, and inclusion, respectively. Let us illustrate this by an example.

Consider a discourse universe with three individuals, namely, *Mary*, *John*, and *Ash*, and assume a situation where it is known that exactly one of them is sleeping. Accordingly, we define a set of possible worlds, $W = \{\textsc{m}, \textsc{j}, \textsc{a}\}$, where each possible world corresponds respectively to the fact that Mary, John, or Ash is sleeping. Then, the proposition φ_1 that *Mary sleeps* and the proposition φ_2 that *John sleeps* are interpreted as follows:

$$\llbracket \varphi_1 \rrbracket = \{\{\textsc{m}\}, \varnothing\}$$
$$\llbracket \varphi_2 \rrbracket = \{\{\textsc{j}\}, \varnothing\}$$

Then, the inquisitive disjunction of φ_1 and φ_2 is interpreted as the union of their interpretations:

$$\llbracket \varphi_1 \vee \varphi_2 \rrbracket = \{\{\textsc{m}\}, \{\textsc{j}\}, \varnothing\}$$

This disjunction does not correspond to a proposition asserting that either Mary or John is sleeping, but rather to the question *whether it is Mary or John who sleeps*. The mere assertion, φ_3, that *Mary or John is sleeping* is interpreted in a different way:

$$\llbracket \varphi_3 \rrbracket = \{\{\textsc{m}, \textsc{j}\}, \{\textsc{m}\}, \{\textsc{j}\}, \varnothing\}$$

The proposition, φ_4 asserting that *Mary does not sleep* is interpreted as follows:

$$\llbracket \varphi_4 \rrbracket = \{\{\textsc{j}, \textsc{a}\}, \{\textsc{j}\}, \{\textsc{a}\}, \varnothing\}$$

Then, the inquisitive disjunction of φ_1 and φ_4 corresponds to the polar question *whether Mary is sleeping*:

$$\llbracket \varphi_1 \vee \varphi_4 \rrbracket = \{\{\textsc{j}, \textsc{a}\}, \{\textsc{m}\}, \{\textsc{j}\}, \{\textsc{a}\}, \varnothing\}$$

In inquisitive semantics, a proposition has both an informative and an inquisitive content. For instance, the informative content of proposition $\varphi_1 \vee \varphi_2$ is that Ash is not sleeping, and its inquisitive content is the issue whether Mary or John is sleeping. The proposition may then be paraphrased as follows: *knowing that Ash does not sleep, one wonders whether Mary or John is sleeping*. A mere assertion such as φ_1 has a trivial inquisitive content. Its paraphrase would be: *knowing that Mary is sleeping, one wonders whether she is sleeping*. Similarly, a mere question such as $\varphi_1 \vee \varphi_4$ has a trivial informative content: *knowing that Mary sleeps or does not sleep, one wonders whether she is sleeping*. Inquisitive semantics features two projection operators, ! and ?, that respectively trivialize the inquisitive content and the informative content of a proposition. Then, for any proposition φ, one has:

$$\varphi = {!\varphi} \wedge {?\varphi}$$

We end this short introduction to inquisitive semantics by presenting first-order inquisitive logic.

Let $\langle \mathcal{F}, \mathcal{R} \rangle$ be the signature of a first-order language, where $\mathcal{F}$ is the set of function symbols, and

$\mathcal{R}$ is the set of relation symbols. From this signature together with a set $\mathcal{X}$ of first-order variables, the notions of terms and of first-order formulas are defined in the standard way.

The notion of a model does not differ from the one used for intensional logic. A model is a triple $\langle D, W, \mathcal{I}\rangle$, where D is the domain of interpretation, W is the set of possible worlds, an $\mathcal{I}$ is the symbol interpretation function such that:[1]

$$\begin{array}{ll} \mathcal{I}(F) \in D^{D^n} & \text{for } F \in \mathcal{F} \text{ of arity } n \\ \mathcal{I}(R) \in \mathscr{P}(W)^{D^n} & \text{for } R \in \mathcal{R} \text{ of arity } n \end{array}$$

Given a valuation ξ from $\mathcal{X}$ into D, the interpretation $[\![t]\!]_\xi$ of a term t is defined as usual, and the interpretation of a first-order formula is given by the following equations:

$$\begin{aligned}
[\![R(t_1,\ldots,t_n)]\!]_\xi &= \mathscr{P}(\mathcal{I}(R)([\![t_1]\!]_\xi,\ldots,[\![t_n]\!]_\xi)) \\
[\![\neg\varphi]\!]_\xi &= \{s \mid \forall t \in [\![\varphi]\!]_\xi .\, s \cap t = \varnothing\} \\
[\![\varphi \wedge \psi]\!]_\xi &= [\![\varphi]\!]_\xi \cap [\![\psi]\!]_\xi \\
[\![\varphi \vee \psi]\!]_\xi &= [\![\varphi]\!]_\xi \cup [\![\psi]\!]_\xi \\
[\![\varphi \to \psi]\!]_\xi &= \\
&\{s \mid \forall t \subseteq s.\, t \in [\![\varphi]\!]_\xi \to t \in [\![\psi]\!]_\xi\} \\
[\![\forall x.\, \varphi]\!]_\xi &= \textstyle\bigcap_{d\in D} [\![\varphi]\!]_{\xi[x:=d]} \\
[\![\exists x.\, \varphi]\!]_\xi &= \textstyle\bigcup_{d\in D} [\![\varphi]\!]_{\xi[x:=d]}
\end{aligned}$$

As for the projection operators ! and ?, they may be added as defined connectives:

$$\begin{aligned} !\varphi &= \neg\neg\varphi \\ ?\varphi &= \varphi \vee \neg\varphi \end{aligned}$$

3 Type-theoretic setting

Since Montague (1973), it is usual in the field of natural language semantics to use the simply typed λ-calculus as an object language to express the compositional semantics of linguistic constructs. In this paper, we adhere to this tradition, and we take advantage of the present section to remind the reader of some notions related to the simply typed λ-calculus, in order to fix the notations.

We take for granted the notions of (untyped) λ-term, β-redex, β-reduction, and β-equivalence. The notations we use, when they are not explicitly introduced, are taken from (Barendregt, 1984). In particular, we write $t \twoheadrightarrow_\beta u$ for the relation of β-reduction.

A λ-term that does not contain any β-redex is called β-normal (or normal, for short). This notion can be explicitly defined in a syntactic way.

Definition 1. The notions of a neutral λ-term and of a normal λ-term are defined by mutual recursion as follows:

1. every λ-variable is a neutral λ-term;
2. every constant is a neutral λ-term;
3. if t a neutral λ-term and u a normal λ-term, then $t\,u$ is a neutral λ-term.
4. every neutral λ-term is a normal λ-term;
5. if t a normal λ-term, so is $\lambda x.\, t$.

The object language we consider comprises two atomic types: IND (the type of individuals) and PROP (the type of proposition).[2] Accordingly, the definition of a simple type is the following one.

Definition 2. The set of simple types $\mathscr{T}$ is inductively defined as follows:

1. IND, PROP $\in \mathscr{T}$;
2. if $\alpha, \beta \in \mathscr{T}$ then $(\alpha \to \beta) \in \mathscr{T}$.

We provide the λ-terms with a type system *à la Church*. To this end, we consider a pairwise disjoint family of countable sets of λ-variables, $(\mathcal{X}_\alpha)_{\alpha\in\mathscr{T}}$, and a pairwise disjoint family of countable sets of constants, $(\mathcal{C}_\alpha)_{\alpha\in\mathscr{T}}$. Given these two families of sets, the notion of a simply-typed lambda-term obeys the next definition.

Definition 3. The family of sets $(\Lambda_\alpha)_{\alpha\in\mathscr{T}}$ of simply-typed λ-terms of type α is inductively defined as follows:

1. For all $\alpha \in \mathscr{T}$, $\mathcal{X}_\alpha \subseteq \Lambda_\alpha$;
2. For all $\alpha \in \mathscr{T}$, $\mathcal{C}_\alpha \subseteq \Lambda_\alpha$;
3. For all $\alpha, \beta \in \mathscr{T}$, if $t \in \Lambda_{\alpha\to\beta}$ and $u \in \Lambda_\alpha$ then $(t\,u) \in \Lambda_\beta$;
4. For all $\alpha, \beta \in \mathscr{T}$, if $x \in \mathcal{X}_\alpha$ and $t \in \Lambda_\beta$ then $(\lambda x.\, t) \in \Lambda_{\alpha\to\beta}$.

We usually let x range over λ-variables, c over constants, and t, u (possibly with subscripts) over λ-terms. If $t \in \Lambda_\alpha$, we say that the term t is of type α, or that α is the type of t. In order to stress the

[1] For the sake of simplicity, we use rigid models, i.e., models in which the interpretation of a term does not vary from one possible world to the other. This assumption does not affect the results we establish in this paper.

[2] IND and PROP are reminiscent of Montague's $\mathbf{e}$ and $\mathbf{t}$, respectively. It is not the case, however, that PROP will be semantically interpreted as the set $\{0, 1\}$.

type of a term, we sometimes decorate it with types, using an exponent like notation. For instance, we write $(\lambda x^{\alpha}.\, t^{\beta})$ when $(\lambda x.\, t) \in \Lambda_{\alpha\to\beta}$.

The simply-typed λ-terms enjoy several interesting properties, in particular, the subject-reduction property and the normalisation-property. The first one says that the sets $(\Lambda_{\alpha})_{\alpha\in\mathscr{T}}$ are closed by β-reduction. The second one says that every simply-typed λ-term has a normal form. We state them explicitly because we will use them in the sequel.

Proposition 4 (Subject Reduction). *Let t and u be two λ-terms such that $t \twoheadrightarrow_{\beta} u$. If $t \in \Lambda_{\alpha}$ then $u \in \Lambda_{\alpha}$.*

Proposition 5 (Normalization). *Let $t \in \Lambda_{\alpha}$. Then there exists a β-normal form $u \in \Lambda_{\alpha}$ such that $t \twoheadrightarrow_{\beta} u$.*

We end this section by providing the simple types and the simply-typed λ-terms with their set-theoretic semantic interpretation.

In order to give a semantic interpretation to the types, we posit two sets, D and W, that are used to give an interpretation to the atomic types. D, the *domain of interpretation*, is the semantic counterpart of type IND. As for W, the set of *possible worlds*, it is used to provide a semantic interpretation to type PROP.

Definition 6. The semantic interpretation $[\alpha]_{\mathsf{i}}$ of a simple type α is inductively defined by the following equations.

$$\begin{aligned}[\textsc{ind}]_{\mathsf{i}} &= D\\ [\textsc{prop}]_{\mathsf{i}} &= \mathscr{P}(W)\\ [\alpha\to\beta]_{\mathsf{i}} &= [\beta]_{\mathsf{i}}{}^{[\alpha]_{\mathsf{i}}}\end{aligned}$$

According to the above definition, a type $\alpha \to \beta$ is interpreted in standard[3] way as the set of set-theoretic functions from the interpretation of α into the interpretation of β. The interpretation of type PROP, however, is not the set of Booleans, $\{0, 1\}$, but the powerset of the set of possible worlds. This corresponds to an intensional (or modal) interpretation, where a proposition is interpreted as a subset of the set of possible worlds (hence, the subscript i in the notation).

We now turn to the interpretation of the λ-terms. To this end, we introduce the notion of a model.

Definition 7. A model $\mathcal{M} = \langle D, W, \mathcal{I}\rangle$ consists of:

1. a set D, called the domain of interpretation;
2. a set W, called the set of possible worlds;
3. a family $\mathcal{I} = (\mathcal{I}_{\alpha})_{\alpha\in\mathscr{T}}$ of interpretation functions $\mathcal{I}_{\alpha}$ from $\mathcal{C}_{\alpha}$ into $[\alpha]_{\mathsf{i}}$.

From now on and throughout the rest of this paper, we consider that a such a model $\mathcal{M} = \langle D, W, \mathcal{I}\rangle$ is given.

The third component of the model, namely $\mathcal{I}$, allows the constant to be given an interpretation. We need a similar notion in order to interpret the λ-variables. Accordingly, we define a valuation $\xi = (\xi_{\alpha})_{\alpha\in\mathscr{T}}$ to be a family of functions ξ_{α} from $\mathcal{X}_{\alpha}$ into $[\alpha]_{\mathsf{i}}$. Let $\xi = (\xi_{\alpha})_{\alpha\in\mathscr{T}}$ be such a valuation, and let $x \in \mathcal{X}_{\alpha}$ and $a \in [\alpha]_{\mathsf{i}}$. Then, $\xi[x{:=}a]$ stands for the valuation $(\xi'_{\alpha})_{\alpha\in\mathscr{T}}$ such that:

1. $\xi'_{\alpha}(x) = a$;
2. for every $y \in \mathcal{X}_{\alpha}$, if $y \neq x$ then $\xi'_{\alpha}(y) = \xi_{\alpha}(y)$;
3. for every $\beta \in \mathscr{T}$, if $\beta \neq \alpha$ then $\xi'_{\beta} = \xi_{\beta}$.

We are now in a position of defining the interpretation of the λ-terms.

Definition 8. Let $\xi = (\xi_{\alpha})_{\alpha\in\mathscr{T}}$ be a valuation. The interpretation $[\![t]\!]_{\mathsf{i}\,\xi}$ of a λ-term t is inductively defined by the following equations:

$$\begin{aligned}[\![x^{\alpha}]\!]_{\mathsf{i}\,\xi} &= \xi_{\alpha}(x)\\ [\![c^{\alpha}]\!]_{\mathsf{i}\,\xi} &= \mathcal{I}_{\alpha}(c)\\ [\![t^{\alpha\to\beta}\, u^{\alpha}]\!]_{\mathsf{i}\,\xi} &= [\![t^{\alpha\to\beta}]\!]_{\mathsf{i}\,\xi}([\![u^{\alpha}]\!]_{\mathsf{i}\,\xi})\\ [\![\lambda x^{\alpha}.\, t^{\beta}]\!]_{\mathsf{i}\,\xi} &= a \in [\alpha]_{\mathsf{i}} \mapsto [\![t^{\beta}]\!]_{\mathsf{i}\,\xi[x^{\alpha}:=a]}\end{aligned}$$

The semantic interpretation of Definition 8 is sound with respect to β-equivalence. We state this proposition explicitly because we will use it later on.

Proposition 9 (Soundness). *Let $\alpha \in \mathscr{T}$, $t, u \in \Lambda_{\alpha}$, and ξ be any valuation. If $t =_{\beta} u$ then $[\![t]\!]_{\mathsf{i}\,\xi} = [\![u]\!]_{\mathsf{i}\,\xi}$.*

The interpretation $[\![t]\!]_{\mathsf{i}\,\xi}$ of a closed λ-term t does not depend upon the valuation ξ. Accordingly, when t is a closed term, we simply write $[\![t]\!]_{\mathsf{i}}$ to denote its interpretation.

A closed λ-term of type PROP is called a formula. Let φ be a formula. We say that φ is valid, and we write $\models_{\mathsf{i}} \varphi$, if and only if $[\![\varphi]\!]_{\mathsf{i}} = W$. Similarly, we say that a sequence of formulas $\varphi_1, \ldots, \varphi_n$ entails a formula φ, which we write $\varphi_1, \ldots, \varphi_n \models_{\mathsf{i}} \varphi$, if and only if $[\![\varphi_1]\!]_{\mathsf{i}} \cap \cdots \cap [\![\varphi_n]\!]_{\mathsf{i}} \subseteq [\![\varphi]\!]_{\mathsf{i}}$.

[3] We use the so-called *standard* interpretation just to keep the definition of a model simple. In fact, everything we do in this paper could be done in the more general setting of Henkin models (Henkin, 1950).

4 Inquisitivation

Definitions 6 and 8 provide to the types and the terms of the object language an intensional interpretation. As explained in the introduction, our objective is to built from this intensional interpretation an inquisitive one.

A first step towards this goal is to provide the type system with an inquisitive interpretation. This consists mainly in interpreting type PROP as the set of inquisitive propositions, i.e., as the set of *sets of sets of possible worlds*. This motivates the next definition.

Definition 10. The inquisitive semantic interpretation $[\alpha]_{\mathsf{i}}$ of a simple type α is inductively defined by the following equations.

$$\begin{aligned}[\mathrm{IND}]_{\mathsf{q}} &= D\\ [\mathrm{PROP}]_{\mathsf{q}} &= \mathscr{P}(\mathscr{P}(W))\\ [\alpha \to \beta]_{\mathsf{q}} &= {[\beta]_{\mathsf{q}}}^{[\alpha]_{\mathsf{q}}}\end{aligned}$$

The next step would be to adapt Definition 8 to the inquisitive case. This adaptation seems almost straightforward, except for the constants. Indeed, the interpretation function of the model interprets a constant of type α as an element of $[\alpha]_{\mathsf{i}}$, not as an element of $[\alpha]_{\mathsf{q}}$. Consequently, what we need is a way of transforming an element of $[\alpha]_{\mathsf{i}}$ into an element of $[\alpha]_{\mathsf{q}}$, while preserving the information it carries.

In other words, what we need for each type α is an embedding $\mathbb{E}_\alpha$ from $[\alpha]_{\mathsf{i}}$ into $[\alpha]_{\mathsf{q}}$. At the level of type PROP, such an embedding exists. Indeed, if $A \subseteq W$ is an intensional proposition, $\mathscr{P}(A)$ is an inquisitive proposition that is purely informative and that carries the same informative content as A.

Now, in order to lift up this embedding at every type, we also need projection operators, $\mathbb{P}_\alpha$, from $[\alpha]_{\mathsf{q}}$ onto $[\alpha]_{\mathsf{i}}$. Again, at the level of type PROP, such a projection exists. It consists of the operation that takes the union of all the elements of a set of sets. Indeed, for every set A, we have that $\bigcup \mathscr{P}(A) = A$. It remains to lift up this embedding-projection pair at every type. This is achieved by the next definition.

Definition 11. The family of embeddings $(\mathbb{E}_\alpha)_{\alpha\in\mathscr{T}}$ and the family of projections $(\mathbb{P}_\alpha)_{\alpha\in\mathscr{T}}$ are defined by mutual recursion over the types as follows:

$$\begin{aligned}\mathbb{E}_{\mathrm{IND}}(a) &= a\\ \mathbb{E}_{\mathrm{PROP}}(p) &= \mathscr{P}(p)\\ \mathbb{E}_{\alpha\to\beta}(f)(a) &= \mathbb{E}_\beta(f(\mathbb{P}_\alpha(a)))\end{aligned}$$

$$\begin{aligned}\mathbb{P}_{\mathrm{IND}}(a) &= a\\ \mathbb{P}_{\mathrm{PROP}}(p) &= \bigcup p\\ \mathbb{P}_{\alpha\to\beta}(f)(a) &= \mathbb{P}_\beta(f(\mathbb{E}_\alpha(a)))\end{aligned}$$

The operators $\mathbb{E}_\alpha$ are what we need to give an inquisitive version of Definition 8. First, let us define an inquisitive valuation to be a family of functions $\xi = (\xi_\alpha)_{\alpha\in\mathscr{T}}$ from $\mathscr{X}_\alpha$ into $[\alpha]_{\mathsf{q}}$. The inquisitive interpretation of a λ-term is then defined as follows.

Definition 12. Let $\xi = (\xi_\alpha)_{\alpha\in\mathscr{T}}$ be an inquisitive valuation. The inquisitive interpretation $\llbracket t\rrbracket_{\mathsf{q}\,\xi}$ of a λ-term t is inductively defined by the following equations:

$$\begin{aligned}\llbracket x^\alpha\rrbracket_{\mathsf{q}\,\xi} &= \xi_\alpha(x)\\ \llbracket c^\alpha\rrbracket_{\mathsf{q}\,\xi} &= \mathbb{E}_\alpha(\mathcal{I}_\alpha(c))\\ \llbracket t^{\alpha\to\beta}\, u^\alpha\rrbracket_{\mathsf{q}\,\xi} &= \llbracket t^{\alpha\to\beta}\rrbracket_{\mathsf{q}\,\xi}(\llbracket u^\alpha\rrbracket_{\mathsf{q}\,\xi})\\ \llbracket \lambda x^\alpha.\, t^\beta\rrbracket_{\mathsf{q}\,\xi} &= a \in [\alpha]_{\mathsf{q}} \mapsto \llbracket t^\beta\rrbracket_{\mathsf{q}\,\xi[x:=a]}\end{aligned}$$

It turns out that inquisitivation is a particular case of de Groote (2015).[4]

The proof of Proposition 9 does not depend on the interpretation of the constants. Consequently, it also holds for Definition 12.

Proposition 13 (Soundness)**.** *Let $\alpha \in \mathscr{T}$, $t, u \in \Lambda_\alpha$, and ξ be any inquisitive valuation. If $t =_\beta u$ then $\llbracket t\rrbracket_{\mathsf{q}\,\xi} = \llbracket u\rrbracket_{\mathsf{q}\,\xi}$.*

As for the notions of inquisitive validity and of inquisitive entailment, they are defined as expected: $\models_{\mathsf{q}} \varphi$ if and only if $\llbracket\varphi\rrbracket_{\mathsf{q}} = \mathscr{P}(W)$, and $\varphi_1, \ldots, \varphi_n \models_{\mathsf{q}} \varphi$ if and only if $\llbracket\varphi_1\rrbracket_{\mathsf{q}} \cap \cdots \cap \llbracket\varphi_n\rrbracket_{\mathsf{q}} \subseteq \llbracket\varphi\rrbracket_{\mathsf{q}}$.

5 Preservation of validity and entailment

In this section, we prove that our inquisitivation procedure preserves the validity of the propositions, that is, a proposition is valid according to its intensional interpretation, if and only if it is valid according to its inquisitive interpretation. We also establish a similar result for entailment.

[4] In our case, the operators of de Groote (2015) are instantiated so: $T\alpha = \alpha$, $\mathsf{U}\,t = t$, $t \bullet u = t\,u$ and $\mathsf{C}\,t = t$

We start by showing that the operators of embedding and projection are indeed embedding-projection pairs, i.e., that $\mathbb{P}_\alpha \circ \mathbb{E}_\alpha$ is the identity for every type α.

Lemma 14. *Let $\alpha \in \mathscr{T}$ be any type. For all $a \in [\alpha]_\mathsf{i}$, $\mathbb{P}_\alpha(\mathbb{E}_\alpha(a)) = a$.*

Proof. The proof proceeds by induction on the structure of α.

1. $\alpha = \text{IND}$.

$$\begin{aligned}\mathbb{P}_{\text{IND}}(\mathbb{E}_{\text{IND}}(a)) &= \mathbb{E}_{\text{IND}}(a)\\ &= a\end{aligned}$$

2. $\alpha = \text{PROP}$.

$$\begin{aligned}\mathbb{P}_{\text{PROP}}(\mathbb{E}_{\text{PROP}}(a)) &= \bigcup \mathbb{E}_{\text{PROP}}(a)\\ &= \bigcup \mathscr{P}(a)\\ &= a\end{aligned}$$

3. $\alpha = \alpha_1 \to \alpha_2$. For all $x \in [\alpha_1]_\mathsf{i}$, we have:

$$\begin{aligned}&\mathbb{P}_{\alpha_1\to\alpha_2}(\mathbb{E}_{\alpha_1\to\alpha_2}(a))(x)\\ &\quad= \mathbb{P}_{\alpha_2}(\mathbb{E}_{\alpha_1\to\alpha_2}(a)(\mathbb{E}_{\alpha_1}(x)))\\ &\quad= \mathbb{P}_{\alpha_2}(\mathbb{E}_{\alpha_2}(a(\mathbb{P}_{\alpha_1}(\mathbb{E}_{\alpha_1}(x)))))\\ &\quad= a(\mathbb{P}_{\alpha_1}(\mathbb{E}_{\alpha_1}(x))) \quad \text{by induction hypothesis}\\ &\quad= a(x) \quad \text{by induction hypothesis}\end{aligned}$$

□

It is not the case that $\mathbb{E}_\alpha(\mathbb{P}_\alpha(a)) = a$ for every $a \in [\alpha]_\mathsf{q}$. For instance, at type PROP, $\mathbb{E}_{\text{PROP}}(\mathbb{P}_{\text{PROP}}(a)) = \mathscr{P}(\bigcup a)$, which is different from a, in general. In fact, the only inquisitive propositions for which $\mathbb{E}_{\text{PROP}}(\mathbb{P}_{\text{PROP}}(a)) = a$ holds are those propositions that are equal to $\mathscr{P}(b)$ for some $b \subseteq W$. These propositions are called *purely informative* because they do not raise any issue. We generalize this notion by defining an element $a \in [\alpha]_\mathsf{q}$ to be *purely informative* if and only if there exists some $b \in [\alpha]_\mathsf{i}$ such that $a = \mathbb{E}_\alpha(b)$.

Lemma 15. *Let $\alpha \in \mathscr{T}$ be any type, and let $a \in [\alpha]_\mathsf{q}$. $\mathbb{E}_\alpha(\mathbb{P}_\alpha(a)) = a$ if and only if a is purely informative.*

Proof. If $\mathbb{E}_\alpha(\mathbb{P}_\alpha(a)) = a$ then a is purely informative, by definition.

Now suppose that a is purely informative, i.e., that there exists $b \in [\alpha]_\mathsf{i}$ such that $a = \mathbb{E}_\alpha(b)$. Then, we have:

$$\begin{aligned}\mathbb{E}_\alpha(\mathbb{P}_\alpha(a)) &= \mathbb{E}_\alpha(\mathbb{P}_\alpha(\mathbb{E}_\alpha(b)))\\ &= \mathbb{E}_\alpha(b) \quad \text{by Lemma 14}\\ &= a\end{aligned}$$

□

We are now in a position of stating and proving the main technical lemma of this section, from which we will derive conservativity results. We first introduce some additional vocabulary and notation.

Let $\xi = (\xi_\alpha)_{\alpha\in\mathscr{T}}$ be an inquisitive valuation. We say that ξ is purely informative if and only if for every $\alpha \in \mathscr{T}$ and $x \in \mathscr{X}_\alpha$, $\xi_\alpha(x)$ is a purely informative element of $[\alpha]_\mathsf{q}$.

For $\xi = (\xi_\alpha)_{\alpha\in\mathscr{T}}$ an inquisitive valuation, we write $\mathbb{P} \circ \xi$ for the intensional valuation $(\mathbb{P}_\alpha \circ \xi_\alpha)_{\alpha\in\mathscr{T}}$, i.e., the intensional valuation $\xi' = (\xi'_\alpha)_{\alpha\in\mathscr{T}}$ such that $\xi'_\alpha(x) = \mathbb{P}_\alpha(\xi_\alpha(x))$.

Lemma 16. *Let $t \in \Lambda_\alpha$ be any λ-term of type α, and let ξ be an inquisitive valuation that is purely informative.*

(a) *If t is neutral,* $[\![t]\!]_{\mathsf{q}\,\xi} = \mathbb{E}_\alpha([\![t]\!]_{\mathsf{i}\,\mathbb{P}\circ\xi})$.

(b) *If t is normal,* $\mathbb{P}_\alpha([\![t]\!]_{\mathsf{q}\,\xi}) = [\![t]\!]_{\mathsf{i}\,\mathbb{P}\circ\xi}$.

Proof. We prove both (a) and (b) by a simultaneous induction on the structure of t.

1. $t = x$.

(a) $$\begin{aligned}[\![x]\!]_{\mathsf{q}\,\xi} &= \xi_\alpha(x)\\ &= \mathbb{E}_\alpha(\mathbb{P}_\alpha(\xi_\alpha(x))) \quad \text{by Lemma 15}\\ &= \mathbb{E}_\alpha([\![x]\!]_{\mathsf{i}\,\mathbb{P}\circ\xi})\end{aligned}$$

(b) Follows from (a), by Lemma 14.

2. $t = c$.

(a) $$\begin{aligned}[\![c]\!]_{\mathsf{q}\,\xi} &= \mathbb{E}_\alpha(\mathcal{I}_\alpha(c))\\ &= \mathbb{E}_\alpha([\![c]\!]_{\mathsf{i}\,\mathbb{P}\circ\xi})\end{aligned}$$

(b) Follows from (a), by Lemma 14.

3. $t = t_1\,t_2$, with $t_1 \in \Lambda_{\beta\to\alpha}$ and $t_2 \in \Lambda_\beta$, for some type β.

(a) $$\begin{aligned}[\![t_1\,t_2]\!]_{\mathsf{q}\,\xi} &= [\![t_1]\!]_{\mathsf{q}\,\xi}([\![t_2]\!]_{\mathsf{q}\,\xi})\\ &= \mathbb{E}_{\beta\to\alpha}([\![t_1]\!]_{\mathsf{i}\,\mathbb{P}\circ\xi})([\![t_2]\!]_{\mathsf{q}\,\xi})\\ &\qquad \text{by induction hypothesis (a)}\\ &= \mathbb{E}_\alpha([\![t_1]\!]_{\mathsf{i}\,\mathbb{P}\circ\xi}(\mathbb{P}_\beta([\![t_2]\!]_{\mathsf{q}\,\xi})))\\ &= \mathbb{E}_\alpha([\![t_1]\!]_{\mathsf{i}\,\mathbb{P}\circ\xi}([\![t_2]\!]_{\mathsf{i}\,\mathbb{P}\circ\xi}))\\ &\qquad \text{by induction hypothesis (b)}\\ &= \mathbb{E}_\alpha([\![t_1\,t_2]\!]_{\mathsf{i}\,\mathbb{P}\circ\xi})\end{aligned}$$

(b) Follows from (a), by Lemma 14.

4. $t = \lambda x.\,t_1$, with $x \in \mathscr{X}_{\alpha_1}$ and $t_1 \in \Lambda_{\alpha_2}$, for some types α_1 and α_2.

(a) Holds vacuously beause t is not neutral.

(b) For every $a \in [\alpha_1]_\mathsf{i}$, we have:

$$\begin{aligned}
&\mathbb{P}_{\alpha_1\to\alpha_2}(\llbracket \lambda x.\, t_1\rrbracket_{\mathsf{q}\,\xi})(a)\\
&\quad= \mathbb{P}_{\alpha_2}(\llbracket \lambda x.\, t_1\rrbracket_{\mathsf{q}\,\xi}(\mathbb{E}_{\alpha_1}\, a))\\
&\quad= \mathbb{P}_{\alpha_2}((a \mapsto \llbracket t_1\rrbracket_{\mathsf{q}\,\xi[x:=a]})(\mathbb{E}_{\alpha_1}(a)))\\
&\quad= \mathbb{P}_{\alpha_2}(\llbracket t_1\rrbracket_{\mathsf{q}\,\xi[x:=\mathbb{E}_{\alpha_1}(a)]})\\
&\quad= \llbracket t_1\rrbracket_{\mathsf{i}\,(\mathbb{P}\circ(\xi[x:=\mathbb{E}_{\alpha_1}\, a]))}\\
&\qquad\qquad \text{by induction hypothesis (b)}\\
&\quad= \llbracket t_1\rrbracket_{\mathsf{i}\,\mathbb{P}\circ\xi[x:=\mathbb{P}_{\alpha_1}(\mathbb{E}_{\alpha_1}(a))]}\\
&\quad= \llbracket t_1\rrbracket_{\mathsf{i}\,\mathbb{P}\circ\xi[x:=a]} \qquad \text{by Lemma 14}\\
&\quad= (a \mapsto \llbracket t_1\rrbracket_{\mathsf{i}\,\mathbb{P}\circ\xi[x:=a]})(a)\\
&\quad= \llbracket \lambda x.\, t_1\rrbracket_{\mathsf{i}\,\mathbb{P}\circ\xi}(a)
\end{aligned}$$

□

We may now establish our main result as an immediate consequence of Lemma 16.

Proposition 17. *Let φ be a proposition. Then, $\llbracket\varphi\rrbracket_{\mathsf{i}} = a$ if and only if $\llbracket\varphi\rrbracket_{\mathsf{q}} = \mathscr{P}(a)$.*

Proof. Suppose $\llbracket\varphi\rrbracket_{\mathsf{i}} = a$. Since φ is a simply-typed λ-term, it has a β-normal form φ', which is of type PROP by Proposition 4. Then, φ' being a normal form of atomic type, it is neutral. Hence:

$$\begin{aligned}
\llbracket\varphi\rrbracket_{\mathsf{q}} &= \llbracket\varphi'\rrbracket_{\mathsf{q}} && \text{by Proposition 13}\\
&= \mathbb{E}_{\text{PROP}}(\llbracket\varphi'\rrbracket_{\mathsf{i}}) && \text{by Lemma 16(a)}\\
&= \mathbb{E}_{\text{PROP}}(\llbracket\varphi\rrbracket_{\mathsf{i}}) && \text{by Proposition 9}\\
&= \mathbb{E}_{\text{PROP}}(a)\\
&= \mathscr{P}(a)
\end{aligned}$$

Conversely, if $\llbracket\varphi\rrbracket_{\mathsf{q}} = \mathscr{P}(a)$, we obtain the expected result in a similar way, using Lemma 16 (b). □

As a particular case of Proposition 17, we obtain that our inquisitivation procedure preserves validity.

Corollary 18. *Let φ be a proposition. Then, $\models_{\mathsf{q}} \varphi$ if and only if $\models_{\mathsf{i}} \varphi$.*

Finally, observing that $A \subseteq B$ if and only if $\mathscr{P}(A) \subseteq \mathscr{P}(B)$, and that $\mathscr{P}(A) \cap \mathscr{P}(B) = \mathscr{P}(A \cap B)$, we obtain that entailment is also preserved.

Corollary 19. *Let $\varphi, \varphi_1, \ldots, \varphi_n$ be propositions. Then, $\varphi_1, \ldots, \varphi_n \models_{\mathsf{q}} \varphi$ if and only if $\varphi_1, \ldots, \varphi_n \models_{\mathsf{i}} \varphi$.*

6 Using inquisitive logic as the object language

Our inquisitivation procedure, as defined by Definitions 11 and 12, leaves the treatment of the logical connectives completely implicit. Somehow, we assumed that the set of constants $\mathcal{C}_{\text{PROP}\to\text{PROP}}$ contains a constant corresponding to negation, that $\mathcal{C}_{\text{PROP}\to\text{PROP}\to\text{PROP}}$ contains constants corresponding to conjunction, disjunction, and implication, and that $\mathcal{C}_{(\text{IND}\to\text{PROP})\to\text{PROP}}$ contains constants corresponding to the quantifiers. In addition, we also assumed family $\mathcal{I} = (\mathcal{I}_\alpha)_{\alpha\in\mathscr{T}}$ of interpretation functions assigns to the logical connectives their standard intensional meaning. That is:

$$\begin{aligned}
\mathcal{I}_{\text{PROP}\to\text{PROP}}(\neg) &= a \mapsto W \setminus a\\
\mathcal{I}_{\text{PROP}\to\text{PROP}\to\text{PROP}}(\wedge) &= a\, b \mapsto a \cap b\\
\mathcal{I}_{\text{PROP}\to\text{PROP}\to\text{PROP}}(\vee) &= a\, b \mapsto a \cup b\\
\mathcal{I}_{\text{PROP}\to\text{PROP}\to\text{PROP}}(\to) &= a\, b \mapsto (W \setminus a) \cup b\\
\mathcal{I}_{(\text{IND}\to\text{PROP})\to\text{PROP}}(\forall) &= p \mapsto \textstyle\bigcap_{d\in D} p(d)\\
\mathcal{I}_{(\text{IND}\to\text{PROP})\to\text{PROP}}(\exists) &= p \mapsto \textstyle\bigcup_{d\in D} p(d)
\end{aligned}$$

Then, according to Definition 12, our inquisitive interpretation of the logical connectives is given by $\mathbb{E} \circ \mathcal{I}$, not by their very inquisitive meaning as defined at the end of Section 2. Spelling it out, our inquisitive interpretation of the logical connectives is as follows:

$$\begin{aligned}
\llbracket\neg\rrbracket_{\mathsf{q}} &= a \mapsto \mathscr{P}(W \setminus (\textstyle\bigcup a))\\
\llbracket\wedge\rrbracket_{\mathsf{q}} &= a\, b \mapsto \mathscr{P}((\textstyle\bigcup a) \cap (\textstyle\bigcup b))\\
\llbracket\vee\rrbracket_{\mathsf{q}} &= a\, b \mapsto \mathscr{P}((\textstyle\bigcup a) \cup (\textstyle\bigcup b))\\
\llbracket\to\rrbracket_{\mathsf{q}} &= a\, b \mapsto \mathscr{P}((W \setminus (\textstyle\bigcup a)) \cup (\textstyle\bigcup b))\\
\llbracket\forall\rrbracket_{\mathsf{q}} &= p \mapsto \mathscr{P}(\textstyle\bigcap_{d\in D}(\bigcup p(d)))\\
\llbracket\exists\rrbracket_{\mathsf{q}} &= p \mapsto \mathscr{P}(\textstyle\bigcup_{d\in D}(\bigcup p(d)))
\end{aligned}$$

Let us call the above interpretation of the logical connectives their *weak inquisitive interpretation*. By contrast, let us call the very inquisitive interpretation of the connectives, as given at the end of Section 2, their *strong inquisitive interpretation*. Then, given an inquisitive valuation ξ, let us define the strong inquisitive interpretation $\llbracket t\rrbracket_{\mathsf{sq}\,\xi}$ of a λ-term t as in Definition 12, except for the logical connectives that are assigned their strong inquisitive interpretation.

We now wonder whether the weak and the strong interpretations of the connectives coincide. For negation, this is indeed the case.

Lemma 20.

$$\llbracket\neg\rrbracket_{\mathsf{sq}} = \llbracket\neg\rrbracket_{\mathsf{q}}$$

Proof.

$$\begin{aligned}\llbracket\neg\rrbracket_{\mathsf{sq}}(a) &= \{s \mid \forall t \in a.\, s \cap t = \varnothing\}\\ &= \{s \mid \forall w \in s.\, \forall t \in a.\, w \notin t\}\\ &= \{s \mid \forall w \in s.\, w \notin \textstyle\bigcup a\}\\ &\qquad \text{because } a \text{ is downward-closed}\\ &= \mathscr{P}(W \setminus (\textstyle\bigcup a))\\ &= \llbracket\neg\rrbracket_{\mathsf{q}}(a)\end{aligned}$$

□

For the other connectives, the weak and the strong interpretations do not coincide in general. Let us exhibit some counterexamples. Let $W = \{w, v\}$, $D = \{1, 2\}$, and Define a to be $\{\{w\}, \{v\}, \varnothing\}$. For conjunction, we have:

$$\llbracket\wedge\rrbracket_{\mathsf{sq}}(a)(a) = a \cap a = a$$

which is different from:

$$\begin{aligned}\llbracket\wedge\rrbracket_{\mathsf{q}}(a)(a) &= \mathscr{P}((\textstyle\bigcup a) \cap (\bigcup a))\\ &= \mathscr{P}(W)\end{aligned}$$

For implication, define b to be $\mathscr{P}(W)$. Then, we have:

$$\begin{aligned}&\llbracket\rightarrow\rrbracket_{\mathsf{sq}}(b)(a)\\ &\quad = \{s \mid \forall t \subseteq s.\, t \in b \rightarrow t \in a\}\\ &\quad = a\end{aligned}$$

which is different from:

$$\begin{aligned}\llbracket\rightarrow\rrbracket_{\mathsf{q}}(b)(a) &= \mathscr{P}((W \setminus (\textstyle\bigcup b)) \cup (\bigcup a))\\ &= \mathscr{P}(W)\end{aligned}$$

For universal quantification, define p to be $\{(1, a), (2, a)\}$. We then obtain a counterexample similar to the one for conjunction, with $\llbracket\forall\rrbracket_{\mathsf{sq}}(p) = a$, which is different from $\llbracket\forall\rrbracket_{\mathsf{q}}(p) = \mathscr{P}(W)$.

For disjunction, define c to be $\{\{w\}, \varnothing\}$, and d to be $\{\{v\}, \varnothing\}$. Then we have:

$$\begin{aligned}\llbracket\vee\rrbracket_{\mathsf{sq}}(c)(d) &= c \cup d\\ &= \{\{w\}, \{v\}, \varnothing\}\end{aligned}$$

which is different from

$$\begin{aligned}\llbracket\vee\rrbracket_{\mathsf{q}}(c)(d) &= \mathscr{P}((\textstyle\bigcup c) \cup (\bigcup d))\\ &= \mathscr{P}(W)\end{aligned}$$

For existential quantification, one obtains a counterexample similar to the one for disjunction by defining q to be $\{(1, c), (2, d)\}$. Then we have that $\llbracket\exists\rrbracket_{\mathsf{sq}}(q) = \{\{w\}, \{v\}, \varnothing\}$, and that $\llbracket\exists\rrbracket_{\mathsf{q}}(q) = \mathscr{P}(W)$.

Because of the non-coincidence of the weak and the strong inquisitive interpretations of the logical connective, we do not have, in general, that for any formula φ:

$$\llbracket\varphi\rrbracket_{\mathsf{q}} = \llbracket\varphi\rrbracket_{\mathsf{sq}} \tag{1}$$

Consequently, Proposition 17 in which $\llbracket\cdot\rrbracket_{\mathsf{q}}$ would be replaced by $\llbracket\cdot\rrbracket_{\mathsf{sq}}$ does not hold.

In order to circumvent this problem, we will introduce a syntactic translation of the λ-terms, $\overline{\cdot}$, such that for every formula φ, $\llbracket\overline{\varphi}\rrbracket_{\mathsf{sq}} = \llbracket\varphi\rrbracket_{\mathsf{q}}$. With such a translation, the picture of our inquisitivation process is as follows:

$$\begin{tikzcd}
& \Lambda \arrow[rr, "\overline{\cdot}"] \arrow[dl, "\llbracket\cdot\rrbracket_{\mathsf{i}}"', ""{name=I}] \arrow[dr, "\llbracket\cdot\rrbracket_{\mathsf{q}}", ""'{name=Q}] & & \Lambda \arrow[dl, "\llbracket\cdot\rrbracket_{\mathsf{sq}}"'] \\
\mathcal{M} & & \mathcal{M} &
\arrow[from=I, to=Q, Rightarrow, bend right, "\mathbb{E}"']
\end{tikzcd}$$

Remark, however, that an exact coincidence between the weak and the strong interpretation of the connectives is not needed in order to have that Equation 1 holds. What is needed is that the weak and the strong interpretations coincide on the image of the embedding $\mathbb{E}$, that is that they coincide for the purely informative elements. For conjunction, implication, and universal quantification, this is the case, as shown by the next lemma.

Lemma 21. *Let* $a, b \in [\text{PROP}]_{\mathsf{q}}$, *and* $p \in [\text{IND} \rightarrow \text{PROP}]_{\mathsf{q}}$ *be purely informative elements.*

(a) $\llbracket\wedge\rrbracket_{\mathsf{sq}}(a)(b) = \llbracket\wedge\rrbracket_{\mathsf{q}}(a)(b)$

(b) $\llbracket\rightarrow\rrbracket_{\mathsf{sq}}(a)(b) = \llbracket\rightarrow\rrbracket_{\mathsf{q}}(a)(b)$

(c) $\llbracket\forall\rrbracket_{\mathsf{sq}}(p) = \llbracket\forall\rrbracket_{\mathsf{q}}(p)$

Proof.

(a) Conjunction. Remark that a being purely informative, we have that $a = \mathscr{P}(\bigcup a)$, and similarly for b. Then, we have:

$$\begin{aligned}\llbracket\wedge\rrbracket_{\mathsf{sq}}(a)(b) &= a \cap b\\ &= \mathscr{P}(\textstyle\bigcup a) \cap \mathscr{P}(\bigcup b)\\ &= \llbracket\wedge\rrbracket_{\mathsf{q}}(a)(b)\end{aligned}$$

(b) Implication.

$$\begin{aligned} & [\![\rightarrow]\!]_{\mathsf{sq}}\,(a)(b) \\ &= \{s \mid \forall t \subseteq s.\, t \in a \rightarrow t \in b\} \\ &= \{s \mid \forall t \subseteq s.\, t \in \mathscr{P}(\textstyle\bigcup a) \rightarrow t \in \mathscr{P}(\textstyle\bigcup b)\} \\ &= \{s \mid \forall t \subseteq s.\, \forall w \in t.\, w \in (\textstyle\bigcup a) \rightarrow w \in (\textstyle\bigcup b)\} \\ &= \{s \mid \forall w \in s.\, w \in (W \setminus (\textstyle\bigcup a)) \cup (\textstyle\bigcup b)\} \\ &= \mathscr{P}((W \setminus (\textstyle\bigcup a)) \cup (\textstyle\bigcup b)) \\ &= [\![\rightarrow]\!]_{\mathsf{q}}\,(a)(b) \end{aligned}$$

(c) Universal quantification. This case is similar to conjunction. □

As for disjunction and existential quantification, their strong and weak interpretations do not coincide, even for the purely informative elements. This is shown, indeed, by the above counterexamples. Nevertheless, we may simulate the weak interpretations of these connective using the projection operator !.

Lemma 22. *Let $\varphi, \psi \in \Lambda_{\text{PROP}}$, $\upsilon \in \Lambda_{\text{IND}\rightarrow\text{PROP}}$, and ξ be an inquisitive valuation.*

(a) *If $[\![\varphi]\!]_{\mathsf{q}\,\xi} = [\![\varphi]\!]_{\mathsf{sq}\,\xi}$ and $[\![\psi]\!]_{\mathsf{q}\,\xi} = [\![\psi]\!]_{\mathsf{sq}\,\xi}$ then $[\![\varphi \vee \psi]\!]_{\mathsf{q}\,\xi} = [\![!(\varphi \vee \psi)]\!]_{\mathsf{sq}\,\xi}$.*

(b) *If for all $d \in S$, $[\![\upsilon]\!]_{\mathsf{q}\,\xi[x:=d]} = [\![\upsilon]\!]_{\mathsf{sq}\,\xi[x:=d]}$ then $[\![\exists x.\, \upsilon]\!]_{\mathsf{q}\,\xi} = [\![!(\exists x.\, \upsilon)]\!]_{\mathsf{sq}\,\xi}$*

Proof.

(a) Disjunction. Remark that for every inquisitive proposition $a \in \mathscr{P}(\mathscr{P}(W))$, $[\![!]\!]_{\mathsf{sq}}\,(a) = \mathscr{P}(\bigcup a)$.

$$\begin{aligned} [\![\varphi \vee \psi]\!]_{\mathsf{q}\,\xi} &= \mathscr{P}((\textstyle\bigcup [\![\varphi]\!]_{\mathsf{q}\,\xi}) \cup (\textstyle\bigcup [\![\psi]\!]_{\mathsf{q}\,\xi})) \\ &= \mathscr{P}(\textstyle\bigcup ([\![\varphi]\!]_{\mathsf{q}\,\xi} \cup [\![\psi]\!]_{\mathsf{q}\,\xi})) \\ &= \mathscr{P}(\textstyle\bigcup ([\![\varphi]\!]_{\mathsf{sq}\,\xi} \cup [\![\psi]\!]_{\mathsf{sq}\,\xi})) \\ &= \mathscr{P}(\textstyle\bigcup [\![\varphi \vee \psi]\!]_{\mathsf{sq}\,\xi}) \\ &= [\![!(\varphi \vee \psi)]\!]_{\mathsf{sq}\,\xi} \end{aligned}$$

(b) Existential quantification. This case is handled similarly. □

Taking advantage of the above lemma, we define the syntactic translation $\overline{\cdot}$ as follows:

$$\begin{aligned} \overline{x} &= x \\ \overline{\neg\varphi} &= \neg\overline{\varphi} \\ \overline{\varphi \wedge \psi} &= \overline{\varphi} \wedge \overline{\psi} \\ \overline{\varphi \vee \psi} &= !(\overline{\varphi} \vee \overline{\psi}) \\ \overline{\varphi \rightarrow \psi} &= \overline{\varphi} \rightarrow \overline{\psi} \\ \overline{\forall x.\, \varphi} &= \forall x.\, \overline{\varphi} \\ \overline{\exists x.\, \varphi} &= !(\exists x.\, \overline{\varphi}) \\ \overline{c} &= c \qquad \text{for the other constants} \\ \overline{t\,u} &= \overline{t}\,\overline{u} \\ \overline{\lambda x.\, t} &= \lambda x.\, \overline{t} \end{aligned}$$

Finally, we obtain the following proposition.

Proposition 23. *For any λ-term u, and any inquisitive valuation ξ that is purely informative,*

$$[\![u]\!]_{\mathsf{q}\,\xi} = [\![\overline{u}]\!]_{\mathsf{sq}\,\xi}$$

Proof. By induction over the λ-terms, using Lemmas 20, 21, and 22. □

7 Application to an epistemic modality

Epistemic modalities are logical operators that can be added to intensional logic to model natural language expressions involving the knowledge of an agent. In epistemic logic (Hintikka, 1962), the semantics of *know that + declarative subclause* uses an operator named $\mathbf{K}$.

Ciardelli and Roelofsen (2015) developed a new operator $\mathbf{K}_{\mathsf{q}}$ in view of 1. adapting $\mathbf{K}$ to inquisitive semantics and 2. modeling the semantics of *know + interrogative subclause*.

Let us take the following sentences as illustrations:

(1) a. $\mathbf{Kj}\ (\mathbf{sleep\ m})$ (John knows that Mary sleeps)
  b. $\mathbf{Kj}\ (?\,(\mathbf{sleep\ m}))$ (John knows whether Mary sleeps)
  c. $\mathbf{Kj}\ (\exists x.\, \mathbf{sleep}\ x)$ (John knows who sleeps)

These last two λ-terms of Λ^{sq} have to be interpreted by the strong inquisitive interpretation so that $\exists$ generates an alternative for every $d \in D$. Similarly, we must interpret $\mathbf{K}$ as the inquisitive epistemic operator of Ciardelli and Roelofsen (2015).

This raises the question whether the strong inquisitive interpretation of (1-a) is still consistent with the one obtained by embedding the intensional version of $\mathbf{K}$.

This section investigates to which group of logical constants $\mathbf{K}$ belongs.

7.1 Traditional modal knowledge

We expose here the traditional treatment of *know* in modal logic (Kripke, 1959).

To interpret $\mathbf{K}$ we need an accessibility relation indexed by individuals $d \in D$:

$$\sigma_d : W \to \mathscr{P}(W)$$

in any model $\mathcal{M}$. In particular, $w\ \sigma_d\ v$ iff agent d cannot distinguish worlds w and v by her knowledge.

Then we can define for every x^{IND} and proposition φ^{PROP},

$$[\![\mathbf{K}\ x\ \varphi]\!]_{\mathsf{i}\ \xi} = \{w \in W \mid \sigma_{[\![x]\!]_{\mathsf{i}\ \xi}}(w) \subseteq [\![\varphi]\!]_{\mathsf{i}\ \xi}\}$$

Embedding this operation yields

$$[\![\mathbf{K}\ x\ \varphi]\!]_{\mathsf{q}\ \xi} = \mathscr{P}(\{w \mid \sigma_{[\![x]\!]_{\mathsf{q}\ \xi}}(w) \subseteq \bigcup [\![\varphi]\!]_{\mathsf{q}\ \xi}\})$$

7.2 Inquisitive knowledge

We can see σ_d as a function from worlds to intensional propositions. The idea of Ciardelli and Roelofsen (2015) is to extend it to a function $\Sigma_d : W \to \mathscr{P}(\mathscr{P}(W))$, mapping worlds to inquisitive propositions, called the inquisitive states of agent d. This way, the inquisitive knowledge modality can take inquisitive propositions as inputs.

The intensional counterpart of Σ_d can be retrieved by taking the truth set of the inquisitive state at world w:

$$\sigma_d(w) = \bigcup(\Sigma_d(w))$$

$\Sigma_d(w)$ represents the issue $\mathcal{P}$ that agent d entertains at world w. The informational content of $\mathcal{P}$ is where d locates the current world, so what d knows. The inquisitive content of $\mathcal{P}$ is related to what d wonders. Therefore, to interpret knowledge, we only need to use $\bigcup \Sigma_d(w)$, i.e. $\sigma_d(w)$.

The strong inquisitive interpretation of the knowledge operator is

$$[\![\mathbf{K}\ x\ \varphi]\!]_{\mathsf{sq}\ \xi} = \{s \mid \forall w \in s.\, \sigma_{[\![x]\!]_{\mathsf{sq}\ \xi}}(w) \in [\![\varphi]\!]_{\mathsf{sq}\ \xi}\}$$

For $\mathbf{K}\ x\ \varphi$ to be true at s, the knowledge of agent x at every world w of s has to settle the proposition expressed by φ. This way, $\mathbf{K}$ can be used to interpret both *know that* + *declarative* and *know* + *interrogative* in a single formulation.

7.3 Inquisitivation of K

The modality $\mathbf{K}$ belongs to group 2: $[\![\mathbf{K}]\!]_{\mathsf{sq}\ \xi}$ coincides with $[\![\mathbf{K}]\!]_{\mathsf{q}\ \xi})$ on the image of $\mathbb{E}$.

Let us first provide a counterexample against their coicidence in the general case.

Again, take the model having $W = \{w, v\}$, $D = \{d\}$ and $\Sigma_d(w) = \Sigma_d(v) = \{\{w\}, \{v\}, \varnothing\}$. Therefore, $\sigma_d(w) = \sigma_d(v) = W$. Set $Q = \Sigma_d(w)$. Then,

$$[\![\mathbf{K}]\!]_{\mathsf{sq}\ \xi}(d)(\mathcal{Q}) = \mathscr{P}(\{w \mid \sigma_d(w) \in \mathcal{Q}\}) = \{\varnothing\}$$

whereas

$$[\![\mathbf{K}]\!]_{\mathsf{q}\ \xi}(d)(\mathcal{Q}) = \mathscr{P}(\{w \mid \sigma_d(w) \subseteq \bigcup \mathcal{Q}\}) = \mathscr{P}(W)$$

Proposition 24. *Let $\mathcal{P}$ be a purely informative issue and $d \in D$.*

$$[\![\mathbf{K}]\!]_{\mathsf{sq}\ \xi}(d)(\mathcal{P}) = [\![\mathbf{K}]\!]_{\mathsf{q}\ \xi}(d)(\mathcal{P})$$

Proof. The derivation goes like this

$$\begin{aligned}[\![\mathbf{K}]\!]_{\mathsf{sq}\ \xi}(d)(\mathcal{P}) &= \{s \mid \forall w \in s.\, \sigma_d(w) \in \mathscr{P}(\bigcup \mathcal{P})\} \\ &= \{s \mid \forall w \in s.\, \sigma_d(w) \subseteq \bigcup \mathcal{P}\} \\ &= \mathscr{P}(\{w \mid \sigma_d(w) \subseteq \bigcup \mathcal{P}\}) \\ &= [\![\mathbf{K}]\!]_{\mathsf{q}\ \xi}(d)(\mathcal{P})\end{aligned}$$

□

This proves that the inquisitive epistemic modality is indeed a "natural" extension of traditional $\mathbf{K}$, as suggested in Ciardelli and Roelofsen (2015).

8 Conclusion

We designed a transformation that creates inquisitive lexical representations out of intensional lexical interpretations. This transformation, called inquisitivation, can be used as a procedure to embed an intensional interpretation into the inquisitive world, where more operations are available, e.g. to express questions.

We proved that inquisitivation preserves validity and entailment.

We classified logical connectives into three groups w.r.t. how their inquisitivation coincides with their counterparts defined by inquisitive logic. Group 1 includes negation and exhibits an exact coincidence. In group 2, connectives (e.g. conjunction) exhibit a coincidence on the image of inquisitivation (i.e. on purely informative issues). The connectives of group 3 (e.g. disjunction) do not coincide in general. But they are definable by their inquisitive counterpart.

We finally showed that the knowledge operator **K** shares properties with its adaptation to inquisitive logic defined by Ciardelli and Roelofsen (2015). As such, it belongs to group 2.

Inquisitivation offers a tool to easily transfer any system based on intensional semantics to inquisitive semantics. Future works may focus on other such systems, like dynamic semantics.

It would also be interesting to try to emulate an inquisitive logic out of another basis than intensional semantics. For example, events may have a rich enough structure to allow an inquisitive logic based on (set of) events instead of information states.

References

Henk Barendregt. 1984. *The Lambda Calculus. Its Syntax and Semantics*, 2 edition, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North Holland.

Ivano Ciardelli, Jeroen Groenendijk, and Floris Roelofsen. 2013. Inquisitive Semantics: A New Notion of Meaning. *Language and Linguistics Compass*, 7(9):459–476.

Ivano Ciardelli, Jeroen Groenendijk, and Floris Roelofsen. 2018. *Inquisitive Semantics*. Oxford Surveys in Semantics and Pragmatics. Oxford University Press, Oxford, New York.

Ivano Ciardelli and Floris Roelofsen. 2015. Inquisitive dynamic epistemic logic. *Synthese*, 192(6):1643–1687.

Ivano Ciardelli, Floris Roelofsen, and Nadine Theiler. 2017. Composing alternatives. *Linguistics and Philosophy*, 40(1):1–36.

Philippe de Groote. 2015. On Logical Relations and Conservativity. In *EPiC Series in Computing*, volume 32, pages 1–11. EasyChair.

Philippe de Groote and Makoto Kanazawa. 2013. A Note on Intensionalization. *Journal of Logic, Language and Information*, 22(2):173–194.

Jeroen Groenendijk and Martin Stokhof. 1984. *Studies on the Semantics of Questions and the Pragmatics of Answers*. Ph.D. thesis, University of Amsterdam.

Charles Leonard Hamblin. 1973. Questions in Montague English. *Foundations of Language*, 10(1):41–53.

Leon Henkin. 1950. Completeness in the Theory of Types. *The Journal of Symbolic Logic*, 15(2):81–91.

Jaakko Hintikka. 1962. Knowledge and Belief: An Introduction to the Logic of the Two Notions. *Studia Logica*, 16:119–122.

Saul A. Kripke. 1959. A Completeness Theorem in Modal Logic. *The Journal of Symbolic Logic*, 24(1):1–14.

Richard Montague. 1970. *English as a Formal Language*. De Gruyter Mouton.

Richard Montague. 1973. The Proper Treatment of Quantification in Ordinary English. In K. J. J. Hintikka, J. M. E. Moravcsik, and P. Suppes, editors, *Approaches to Natural Language: Proceedings of the 1970 Stanford Workshop on Grammar and Semantics*, Synthese Library, pages 221–242. Springer Netherlands, Dordrecht.