On the Mathematical Properties of Linguistic Theories

C. Raymond Perrault

Dept. of Computer Science
University of Toronto
Toronto, Ontario, Canada M5S 1A4

ABSTRACT

Meta-theoretical results on the decidability, generative capacity, and recognition complexity of several syntactic theories are surveyed. These include context-free grammars, transformational grammars, lexical functional grammars, generalized phrase structure grammars, and tree adjunct grammars.

1. Introduction.

The development of new formalisms in which to express linguistic theories has been accompanied, at least since Chomsky and Miller's early work on context-free languages, by the study of their meta-theory. In particular, numerous results on the decidability, generative capacity, and more recently the complexity of recognition of these formalisms have been published (and rumoured!). Strangely enough, much less attention seems to have been devoted to a discussion of the significance of these mathematical results. As a preliminary to the panel on formal properties which will address the significance issue, it seemed appropriate to survey the existing results. Such is the modest goal of this paper.

We will consider context-free languages, transformational grammars, lexical functional grammars, generalized phrase structure grammars, and tree adjunct grammars. Although we will not examine them here, formal studies of other syntactic theories have been undertaken: e.g. Warren [51] for Montague's PTQ [30], and Borgida [7] for the stratificational grammars of Lamb [25]. There follows a brief summary of some comments in the literature about related empirical issues, but we avoid entirely the issue of whether one theory is more descriptively adequate than another.

2. Preliminary Definitions

We assume the reader is familiar with the basic definitions of regular, context-free (CF), context-sensitive (CS), recursive, and recursively enumerable (r.e.) languages and with their acceptors as can be found in [14].

Some elementary definitions from complexity theory may be useful. Further details may be found in [2]. Complexity theory is the study of the resources required of algorithms, usually space and time. Let $f(x)$ be a function, say the recognition function for a language L. The most interesting results we could obtain about f would be a *lower bound* on the resources needed to compute f on a machine of a given architecture, say a von Neumann computer or a parallel array of neurons. These results over whole classes of machines are very difficult to obtain, and none of any significance exist for parsing problems.

Restricting ourselves to a specific machine model and an algorithm M for f, we can ask about the **cost** (e.g time or space) $c(x)$ of executing M on a specific input x. Typically c is too fine-grained to be useful: what one studies instead is a function c_w whose argument is an integer n denoting the *size* of the input to M, and which gives some measure of the cost of processing inputs of length n. Complexity theorists have been most interested in the **asymptotic** behaviour of c_w, i.e. the behaviour of c_w as n gets large.

If one is interested in *upper bounds* on the behaviour of M, one usually defines $c_w(n)$ as the *maximum* of $c(x)$ over all inputs x of size n. This is called the **worst-case complexity function** for M. Notice that other definitions are possible: one could define the **expected complexity function** $c_e(n)$ for M as the average of $c(x)$ over all inputs of length n. c_e might be more useful than c_w if one had an idea of what the distribution of inputs to M could be. Unfortunately, the introduction of probabilistic considerations makes the study of expected complexity technically more difficult that of worst case complexity. For a given problem, expected and worst case measures may be quite different.

It is quite difficult to get detailed descriptions of c_w and for many purposes a cruder estimate is sufficient. The next abstraction involves "lumping" classes of c_w functions into simpler ones that more clearly demonstrate their asymptotic behaviour and are easier to manipulate. This is the purpose of **O-notation**. Let $f(n)$ and $g(n)$ be two functions. f is said to be $O(g)$ if a constant multiple of g is an upper bound for f, for all but a finite number of values of n. More precisely, f is $O(g)$ if there is are constants K and n_0 such that for all $n>n_0$, $f(n) < Kg(n)$.

Given an algorithm M, we will say that its worst-case time complexity is $O(g)$ if the worst-case time cost function $c_w(n)$ for M is $O(g)$. Notice that this merely says that almost all inputs to M of size n can be processed in time at most a constant times $g(n)$. It does *not* say that all inputs require $g(n)$ time, or even that any do even on M, let alone on any other machine that implements f. Also, if two algorithms A_1 and A_2 are available for a function f, and if their worst-case complexity can be given respectively as $O(g_1)$ and $O(g_2)$, and $g_1 < g_2$, it may still be the case that for a large number of cases (maybe even for all cases one is likely to encounter in practice) that A_2 will be the preferable algorithm, simply because the constant K_1 for g_1 may be much smaller than K_2 for g_2.

This research was sponsored by the National Science and Engineering Research Council of Canada under Grant A9285.

In examining known results about the recognition complexity of various theories, it is useful to consider how "robust" they are in the face of changes in the machine model from which they were derived. These models can be divided into two classes: **sequential** models and **parallel** models. Sequential models [2] include the familiar single- and multi-tape Turing Machines (TMs) as well as **Random Access Machines** (RAMs) and **Random Access Stored Program Machines** (RASPs). A RAM is like a TM except that its working menory is random access rather than sequential. A RASP is like a RAM but stores its program in its memory. Of all these models, it is most like a von Neumann computer.

All these sequential models can simulate each other in ways that do not cause great changes in time complexity. For example, a k-tape Turing Machine that runs in time $O(t)$ can be simulated by a RAM in time $O(t)$, and conversely, a RAM running in $O(t)$ can be simulated by a k-tape TM in time $O(t^2)$. In fact, all familiar sequential models are **polynomially related**: they can simulate each other with at most a polynomial loss in efficiency.

Thus if a syntactic model is known to have a difficult recognition problem on one sequential model, then it will not have a much easier one on another.

Transforming a sequential algorithm to one on a parallel machine with a fixed number K of processors provides at most a factor K improvement in speed. More interesting results are obtained when the number of processors is allowed to grow with the size of the problem, e.g. with the length of the string to be parsed. If we view these processors as connected together in a circuit, with inputs values entering at one end and outputs being produced at the other, then a problem that has a solution on a *sequential* machine in polynomial time and in space s will have a solution on a *parallel* machine with a polynomial number of processors and *circuit depth* (or maximum number of processors data must be passed through from input to output) $O(s^2)$. Since the depth of a parallel circuit corresponds to the (parallel) *time* required to complete the computation, this means that algorithms with sequential solutions requiring small space (such as deterministic CSLs) have fast parallel solutions. For a comprehensive survey of parallel computation, see Cook[9].

3. Context-Free Languages.

Recognition techniques for context-free languages are well-known [3]. The so-called "CKY" or "dynamic programming" method is attributed by Hays [15] to J. Cocke, and it was discovered independently by Kasami [54] and Younger [53] who showed it to be $O(n^3)$. It requires the grammar to be in Chomsky Normal Form, and putting an arbitrary grammar in CNF may square the size of the grammar.

Earley's algorithm recognizes strings in arbitrary CFGs in time $O(n^3)$ and space $O(n^2)$, and in time $O(n^2)$ for unambiguous CFGs. Graham, Harrison and Ruzzo [13] give an algorithm that unifies CKY and Earley's [10] algorithm, and discuss implementation details.

Valiant [50] showed how to interpret the CKY algorithm as the finding of the transitive closure of a matrix and thus reduced CF recognition to matrix multiplication, for which sub-cubic algorithms exist. Because of the enormous constants of proportionality associated with this method, it is not likely to be of much practical use, either an implementation method or as a description of the function of the brain.

Ruzzo [55] has shown how CFLs can be recognized by boolean circuits of depth $O(\log(n)^2)$, and thus that parallel recognition can be done in time $O(\log(n)^2)$. The required circuit has size polynomial in n.

So as not to get mystified by the *upper bounds* on CF recognition, it is useful to remember that no known CFL requires more than linear time, nor is there a (non-constructive) proof of the existence of such a lang[illegible].

For an empirical comparison of various parsing methods, see Slocum [44].

4. Transformational Grammar.

From its earliest days, discussions of transformational grammar (TG) have included mention of matters computational.

Peters and Ritchie [33] provided the first non-trivial results on the generative power of TGs. Their model reflects the "Aspects" version quite closely, including transformations that could move and add constituents, and delete them subject to recoverability. All transformations are obligatory, and applied cyclically from the bottom up. They show that every recursively enumerable (r.e.) set can be generated by a TG using a context-sensitive base. The proof is quite simple: the right-hand sides of the type-0 rules that generate the r.e. set are padded with a new "blank" symbol to make them at least as long as their left-hand sides. Rules are added to allow the blank symbols to commute with all others. These context-sensitive rules are then used as the base of a TG whose only transformation deletes the blank symbols.

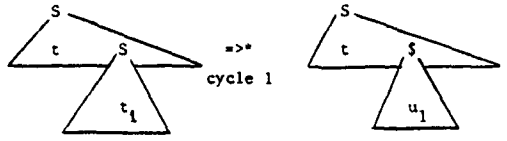
Thus if the transformational formalism itself is supposed to *characterize* the grammatical strings of possible natural languages, then the only languages being excluded are those which are not enumerable under *any* model of computation.

At the expense of a considerably more intricate argument, the previous result can be strengthened [32] to show that every r.e. set can be generated by a context-free based TG, as long as a **filter** (intersection with a regular set) can be applied to the phrase-markers output by the transformations. In fact, the base grammar can be *independent* of the language being generated. The proof involves simulating a TM by a TG. The transformations first generate an "input tape" for the TM being simulated, and then apply the TM productions, one per cycle of the grammar. The filter insures that the base grammar generated just as many S nodes as necessary to generate the input string and do the simulation. Again, if the transformational formalism is supposed to characterize the possible natural languages, then the **Universal Base Hypothesis** [31] according to which all natural languages can be generated from the same base grammar is empirically vacuous: *any* recursively enumerable language can.

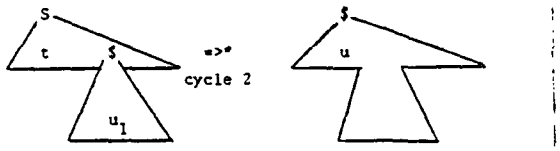
Several attempts were then made to find a restricted form of the transformational model that was descriptively adequate and yet whose generated languages are recursive (see e.g. [27]). Since a key part of the proof in [32] involves the use of a filter on the final derivation trees, Peters and Ritchie examined the consequences of forbidding final filtering [35]. They show that if S is the only recursive symbol in the CF base then the generated language L is *predictably enumerable* and *exponentially bounded*. A language L is **predictably enumerable** if there is an "easily" computable function $t(n)$ that gives an upper bound on the number of tape squares needed by its enumerating TM to enumerate the first n elements of L. L is **exponentially bounded** if there is a constant K such that for every string x in L there is another string x' in L whose length is at most K times the length of x.

The class of non-filtering languages is quite unusual, including all the CFLs (obviously), but also some (but not all) CSLs, some (but not all) recursive languages, and some (but not all) r.e. languages.

The source of non-recursivity in transformationally generated languages is that transformations can delete arbitrarily large parts of the tree, thus producing surface trees arbitrarily smaller than the deep structure trees they were derived from. This is what Chomsky's recoverability of deletions condition was meant to avoid. In his thesis, Petrick [36] defines the following **terminal-length-increasing** condition on transformational derivations: consider the following two p-markers from a derivation, where the right one is derived from the left one by applying the cycle of transformations to subtree c producing the subtree u_1.



Continuing the derivation, apply the cycle to tree t yielding tree u.



A derivation satisfies the terminal-length-increasing condition if the yield of u is always longer than the yield of u_1.

Petrick shows that if all recursion in the base "passes through S" and if all derivations satisfy the terminal-length-increasing condition, then the generated language is recursive. Using a slightly more restricted model of transformations, Rounds [42] strengthens this result by showing that the resulting languages are in fact context-sensitive.

In an unpublished paper, Myhill shows that if the condition is weakened to terminal-length-non-decreasing, then the resulting languages can be recognized in space at most *exponential* in the length of the input. This implies that the recognition can be done in at most double-exponential time, but Rounds [43] shows that not only can recognition be done in *exponential time*, but that every language recognizable in exponential time can be generated by a TG satisfying the terminal-length-non-decreasing condition and recoverability of deletions.

This is a very strong result, because of the closure properties of the class of exponential-time languages. To see why this is so requires a few more definitions.

Let **P** be the class of all languages that can be recognized in polynomial time on a deterministic TM, and **NP** the class of all languages that can be recognized in polynomial time on a non-deterministic TM. P is obviously contained in NP, but the converse is not known, although there is much evidence that is false.

There is a class of problems, the so-called **NP-complete** problems, which are in NP and "as difficult" as any problem in NP in the following sense: if *any* of them could be shown to be in P, then *all* the problems in NP would also be in P. One way to show that a language L is NP-complete is to show that L is in NP and that every other language L_0 in NP can be **polynomially transformed** into L, i.e. that there is a deterministic TM, operating in polynomial time, that will transform an input w to L into an input w_0 to L_0 such that w is in L if and only w_0 is in L_0. In practice, to show that a language is NP-complete, one shows that it is in NP, and that some already-known NP-complete language can be polynomially transformed to it.

All the known NP-complete languages can be recognized in exponential time on a deterministic machine, and none are known to have sub-exponential solutions. Thus since the restricted transformational languages of Rounds characterize the exponential languages, then if all of them were to be in P, then P would be equal to NP. Putting it another way, if P is not equal to NP, then some transformational languages (even those satisfying the terminal-length-non-increasing condition) have no "tractable" (i.e. polynomial time) recognition procedures on any deterministic TM. Note that this result also holds for all the other known sequential models of computation, and even for parallel machines with as many as a *polynomial* number of processors.

5. Lexical Functional Grammar.

In part, transformational grammar seeks to account for a range of constraints or dependencies within sentences. Of particular interest are subcategorization dependencies and predicate-argument dependencies. These dependencies can hold over arbitrarily large distances. Several recent theories suggest different ways of accounting for these dependencies, but without making use of transformations. We will examine three of these, Lexical Functional Grammar, Generalized Phrase Structure Grammar, and Tree Adjunct Grammars, in the next few sections.

Lexical Functional Grammar (LFG) of Kaplan and Bresnan [24] aims to provide a descriptively adequate syntactic formalism without transformations. All the work done by transformations is instead encoded in structures in the lexicon and in links established between nodes in the constituent structure

LFG languages are CS and properly include the CFLs [24]. Berwick [5] shows that a set of strings whose recognition problem is known to be NP-complete, namely the set of satisfiable boolean formulas, is an LFG language. Therefore, as was the case for Rounds's restricted class of TGs, if P is not equal to NP, then some languages generated by LFGs do not have polynomial time recognition algorithms. Indeed only quite "basic" parts of the LFG mechanism are necessary to the reduction. This includes mechanisms necessary for feature agreement, for forcing verbs to take certain cases, and lexical ambiguity. Thus no simple change to the formalism is likely to avoid the combinatorial consequences of the full mechanism.

Berwick has also examined the relation between LFG and the class of languages generated by **indexed grammars** [1], a class known to be a proper subset of the CSLs, but including some NP-complete languages [42]. He claims (personal communication) that the indexed languages are a proper subset of the LFG languages.

6. Generalized Phrase Structure Grammar.

In a series of papers, Gerald Gazdar and his colleagues [21] have argued for a joint account of the syntax and semantics of English like LFG in eschewing the use of transformations but unlike it in positing only one level of

syntactic description. The syntactic apparatus is based on a non-standard interpretation of phrase-structure rules and on the use of meta-rules. The formal consequences of both these moves have been investigated.

6.1. Node Admissibility

There are two ways of interpreting the function of CF rules. The first, and most usual, is as rules for *rewriting* strings. Derivation trees can then be seen as canonical representatives of classes of derivations producing the same string, and differing only in the order of application of the same productions.

The second interpretation of CF rules is as **constraints** on derivation trees: a legal derivation tree is one where each node is "admitted" by a rule, i.e. each node dominates a sequence of nodes in a way sanctioned by a rule. For CF rules, the two interpretations obviously generate the same strings *and* the same set of trees.

Following a suggestion of McCawley's, Peters and Ritchie [34] showed that if one considered *context-sensitive* rules from the node-admissibility point of view, the languages defined were still CF. Thus the use of CS rules in the base to impose sub-categorization restrictions, for example, does not increase the weak generative capacity of the base component. (For some different restrictions of context-sensitive rules that guarantee that only CFLs will be generated, see Baker [4].)

Rounds [40] gives a simpler proof of Peters and Ritchie's node-admissibility result using the techniques from tree-automata theory, a generalization to trees of finite state automata theory for strings. Just as a finite state automaton (FSA) **accepts** a string by reading it one character at a time, changing its state at each transition, a finite state tree automaton (FSTA) traverses trees, propagating states. The **top-down FSTA** "attaches" a starting state (from a finite set) to the root of the tree. Transitions are allowed by productions of the form

$$(q, a, n) \dashrightarrow (q_1, \ldots, q_n)$$

such that if state q is being applied to a node labelled a and dominating n descendants, then state q_i should be applied to its ith descendant. Acceptance occurs if all leaves of the tree end up labelled with states in the accepting subset. The **bottom-up FSTA** is similar: starting states are attached to the leaves of the tree and the productions are of the form

$$(a, n, q_1, \ldots, q_n) \rightarrow q$$

indicating that if a node labelled a dominating n descendants each labelled with states q_1 to q_n, then node a gets labelled with state q. Acceptance occurs when the root is labelled by a state from the subset of accepting states.

As is the case with FSAs, FSTAs of both flavours can be either deterministic or non-deterministic. A set of trees is said to be **recognizable** if it is accepted by a non-deterministic bottom-up FSTA. Again as with FSAs, any set of trees accepted by a non-deterministic bottom-up FSTA is accepted by a deterministic bottom-up FSTA, but the result does not hold for top-down FSTA, although the recognizable sets are exactly the languages recognized by non-deterministic top-down FSTAs.

A set of trees is **local** if it is the set of derivation trees of a CF grammar. Clearly, every local set is recognizable by a one-state bottom-up FSTA that checks at each node that it satisfies a CF production. Also, the **yield** of a recognizable set of trees (the set of strings it generates) is CF. Although not all recognizable sets are local, they can all be mapped into local sets by a simple (homomorphic) mapping.

Rounds's proof [41] that CS rules under node-admissibility generate only CFLs involves showing that the set of trees accepted by the rules is recognizable, i.e. that there is a non-deterministic bottom-up FSTA that can check at each node that some node-admissibility condition holds there. This requires checking that the "strictly context-free" part of the rule holds, and that some proper analysis of the tree passing through the node satisfies the "context-sensitive" part of the rule.

The difficulty comes from the fact that the bottom-up automaton cannot generate the set of proper analyses, but must instead propagate (in its state set) the proper analysis conditions necessary to "admit" the nodes of its subtrees. It must, of course, also check that those rules get satisfied.

A more intuitive proof using *tree transducers* as well as FSTAs is sketched inthe Appendix.

Joshi and Levy [21] strengthened Peters and Ritchie's result by showing that the node admissibility conditions could also include arbitrary Boolean combinations of **dominance** conditions: a node could specify a bounded set of labels that must occur immediately above it along a path to the root, or immediately below it on a path to the frontier.

In general the CF grammars constructed in the proof of weak equivalence to the CS grammars under node admissibility are much larger than the original, and not useful for practical recognition. Joshi, Levy and Yueh [22], however, show how Earley's algorithm can be extended to a parser that uses the local constraints directly.

6.2. Metarules.

The second important mechanism used by Gazdar [11] is **metarules,** or rules that apply to rules to produce other rules. Using standard notation for CF rules, one example of a metarule that could replace the transformation known as "particle movement" is:

$$V \dashrightarrow V\ N\ Pt\ X \Longrightarrow V \dashrightarrow V\ Pt\ N[-PRO]\ X$$

X here is a variable behaving like variables in structural analyses of transformations. If such variables are restricted to being used as *abbreviations*, that is if they are only allowed to range over a *finite* subset of strings over the vocabulary, then closing the grammar under the metarules produces only a finite set of derived rules, and thus the generative power of the formalism is not increased. If, on the other hand, X is allowed to range over strings of *unbounded* length, as are the **essential variables** of transformational theory, then the consequences are less clear. It is well known, for example, that if the right-hand sides of phrase structure rules are allowed to be arbitrary regular expressions, then the generated languages are still context-free. Might something like this not be happening with essential variables in metarules? It turns out not.

The formal consequences of the presence of essential variables in metarules depends on the presence of another device, the so-called **phantom categories**. It may be convenient in formulating metarules to allow, in the left-hand sides of rules, occurrences of syntactic categories that are never introduced by the grammar, i.e. that never appear in the right-hand sides of rules. In standard CFLs, these are called *useless categories*, and rules containing them can simply be dropped, with no change in generative capacity. Not so with metarules: it is possible for metarules to rewrite rules containing phantom categories into rules without them. Such a device was proposed at one time as a way to implement passives in the GPSG framework.

Uszkoreit and Peters [49] have shown that essential variables in metarules are powerful devices indeed: CF grammars with metarules that use at most one essential variable and allow phantom categories can generate all recursively enumerable sets. Even if phantom categories are banned, as long as the use of at least one essential variables is allowed, then some non-recursive sets can be generated.

Possible restrictions on the use of metarules are suggested in Gazdar and Pullum [12]. Shieber et al.[45] discuss some empirical consequences of these moves.

7. Tree Adjunct Grammar.

The Tree Adjunct Grammars (TAGs) of Joshi and his colleagues presents a different way of accounting for syntactic dependencies ([17], [19]). A TAG consists of two (finite) sets of (finite) trees, the **centre** trees and the **adjunct** trees.

The centre trees correspond to the surface structures of the "kernel" sentences of the languages. The root of the adjunct trees is labelled with a non-terminal symbol which also appears exactly once on the frontier of the tree. All other frontier nodes are labelled with terminal symbols. Derivations in TAGs are defined by repeated application of the operation of **adjunction**. If c is a centre tree containing an occurrence of a non-terminal A, and if α is an adjunct tree whose root (and one node n on the frontier) is labelled A, then the adjunction of α to c is performed by "detaching" from c the subtree t rooted at A, attaching α in its place, and reattaching t at node n. Adjunction may then be seen as a tree analogue of a context-free derivation for strings [40]. The string languages obtained by taking the yields of the tree languages generated by TAGs are called **Tree Adjunct Languages**, or TALs.

In TAGs all long-distance dependencies are the result of adjunctions separating nodes that at one point in the derivation were "close". Both crossing and non-crossing dependencies can be represented [18]. The formal properties of TAGs are fully discussed in [20], [52], [23]. Of particular interest are the following.

TALs properly contain the CFLs and are properly contained in the indexed languages, which in turn are properly contained in the CSLs. Although the indexed languages contain NP-complete languages, TALs are much better behaved: Joshi and Yokomori report [personal communication] an $O(n^4)$ recognition algorithm and conjecture that an $O(n^3)$ bound may be possible.

8. A Pointer to Empirical Discussions

The literature on the empirical issues underlying the formal results reported here is not extensive.

Chomsky argues convincingly [8] that there is no argument for natural languages *necessarily* being recursive. This, or course, is different from the possibility that languages are *contingently* recursive. Putnam [39] gives three reasons he claims "point in this direction": (1) "speakers can presumably classify sentences as acceptable or unacceptable, deviant or non-deviant, et cetera, without reliance on extra-linguistic contexts. There are of course exceptions to this rule ", (2) grammaticality judgements can be made for nonsense sentences, and (3) grammars can be learned. (2) and (3) are irrelevant and (1) contains its own counter-argument.

Peters and Ritchie [33] contains a suggestive but hardly open-and-shut case for contingent recursivity: (1) every TG has an exponentially bounded cycling function, and thus generates only recursive languages, (2) every natural language has a descriptively adequate TG, and (3) the complexity of languages investigated so far is typical of the class.

Hintikka[16] presents a very different argument against the recursivity of English based on the distribution of the words *any* and *every*. His account of why *John knows everything* is grammatical while *John knows anything* is not is that *any* can appear only in contexts where replacing it by *every* changes the meaning. Taking meaning to be logical equivalence, this means that grammaticality is dependent on the determination of logical equivalence of logical formulas, an undecidable problem. Chomsky [8] argues that a simpler solution is available, namely one that replaces logical equivalence by syntactic identity of some kind of logical form.

Pullum and Gazdar [38] is a thorough survey of, and argument against, published claims (mainly the "respectively" examples [26], Dutch cross-serial dependencies, and nominalization in Mohawk [37]) that some natural languages cannot be weakly generated by CF grammars. No claims are made about the strong adequacy of CFGs.

9. Seeking Significance.

When can the supporter of a weak (syntactic) formalism (i.e. low recognition complexity, low generative capacity) claim that it superior to a competing more powerful formalism?

Linguistic theories can differ along several dimensions, with generative capacity and recognition capacity being only two (albeit related) ones. The evaluation must take into consideration at least the following others:

Coverage. Do the theories make the same grammatical predictions?

Extensibility. The linguistic theory of which the syntactic theory is a part will want to express well-formedness constraints other than syntactic ones. These constraints may be expressed over syntactic representations, or over different representations, presumably related to the syntactic ones. One theory may make this connection possible when another does not. This of course underlies the arguments for strong descriptive adequacy.

Also relevant here is how the linguistic theory as a whole is decomposed. The syntactic theory can obviously be made simpler by transferring some of the explanatory burden to another constituent. The classic example in programming languages is the constraint that all variables must be declared before they are used. This constraint cannot be imposed by a CFG but can be by an indexed grammar, at the cost of a dramatic increase in recognition complexity. Typically, however, the requirement is simply not considered part of "syntax", which thus remains CF, and imposed separately. In this case, the overall recognition complexity remains some low-order polynomial. Some arguments of this kind can be found in [38].

Separating the constraints into different subtheories will not in general make the problem of recognizing strings that satisfy all the constraints any more efficient, but it may allow limiting the power of each constituent. To take an extreme example, every r.e. set is the homomorphic image of the intersection of two context-free languages.

Implementation. This is probably the most subtle set of issues determining the significance of the formal results, and I don't claim to understand them.

Comparison between theories requires agreement between the machine models used to derive the complexity results. As mentioned above, the sequential models are all polynomially related, and no problem not having a

polynomial time solution on a sequential machine is likely to have one on a parallel machine limited to at most a polynomial number of processors, at least if P is not equal to NP. Both these results restrict the improvement one can obtain by changing implementation, but are of little use in comparing algorithms of low complexity. Berwick and Weinberg [6] give examples of how algorithms of low complexity may have different implementations differing by large constant factors. In particular, changes in the form of the grammar and in its representation may have this effect.

But of more interest I believe is the fact that implementation is often accompanied by some form of resource limitation that has two effects. First it is also a change in *specification*. A context-free parser implemented with a bounded stack recognizes only a finite-state language.

Second, very special implementations can be used if one is willing to restrict the size of the *problem* to be solved, or even use special-purpose methods for limited problems. Marcus's parser [28] with its bounded look-ahead is another good example. Sentences parsable within the allowed look-ahead have "quick" parses, but some grammatical sentences, such as "garden path" sentences cannot be recognized without an extension to the mechanism that would distort the complexity measures.

There is obviously much more of this story to be told. Allow me to speculate as to how it might go. We may end up with a space of linguistic theories, differing in the idealization of the data they assume, in the way they decompose constraints, and in the procedural specifications they postulate (I take it that two theories may differ in that the second simply provides more detail than the first as to how constraints specified by the first are to be used.) Our observations, in particular our measurements of necessary resources, are drawn from the "ultimate implementation", but this does not mean that the "ultimately low-level theory" is necessarily the most informative, witness many examples in the physical sciences, or that less procedural theories are not useful stepping stones to more procedural ones.

It is also not clear that theories of different computational power may not be useful as descriptions of different parts of the syntactic apparatus. For example, it may be easier to learn statements of constraints within the framework of a general machine. The constraints once learned might then be subjected to transformation to produce more efficient special-purpose processors also imposing resource limitations. Indeed, the "possible languages" of the future may be more complex than the present ones, just as earlier ones may have been syntactically simpler. Were ancient languages regular?

Whatever we decide to make of existing formal results, it is clear that continuing contact with the complexity community is important. The driving problems there are the $P = NP$ question, the determination of lower bounds, the study of time-space tradeoffs, and the complexity of parallel computations. We still have some methodological house-cleaning to do, but I don't see how we can avoid being affected by the outcome of their investigations.

ACKNOWLEDGEMENTS

Thanks to Bob Berwick, Aravind Joshi, Jim Hoover, and Stan Peters for their suggestions.

APPENDIX

Rounds [41] proves that context-sensitive rules under node-admissibility generate only context-free languages by constructing a non-deterministic bottom-up tree automaton to recognize the accepted trees. We sketch here a proof that makes use of several *deterministic transducers* instead.

FSTAs can be generalized so that instead of simply accepting or rejecting trees, they transform them, by adding constant trees, and deleting or duplicating subtrees. Such devices are called **finite state tree transducers** (FSTT), and like the FSTA they can be top-down or bottom-up. First motivated as models of syntax-directed translations for compilers, they have been extensively studied (e.g. [47], [48], [40]) but a simple subset is sufficient here.

The idea is this. Let T be the set of trees accepted by the CS-based grammar. Let t be in T. FSTTs can be used to label each node n of t with the set of all proper analyses passing through n. It will then be simple to check that each node satisfies one of the node admissibility conditions by sweeping through the labelled tree with a bottom-up FSTA.

The node labelling is done by two FSTTs τ_1 and τ_2. Let m be the maximum length of any left or right-context of any node admissibility condition. Thus we need only label nodes with sets of strings of length at most m, and over a finite alphabet there are only a finite number of such strings.

τ_1 operates bottom-up on a tree t, and labels each node n of t with three sets *Prefix(n)*, *Suffix(n)*, and *Yield(n)* of proper analyses: if P is the set of all proper analyses of the subtree rooted at n, then *Prefix(n)* is the set of all substrings of length at most m that are prefixes of strings of P. Similarly, *Suffix(n)* is the set of all suffixes of length at most m, and *Yield(n)* is the set of all strings of P of length at most m. It can easily be shown that for any set of trees T, T is recognizable if and only if $\tau_1(T)$ is.

Applying to the output of τ_1, the second transducer τ_2, operating top-down, labels each node n with all the proper analyses going through n, i.e. with a pair of sets of strings. The first set will contain all left-contexts of node n and the second all right-contexts. τ_2 also preserves recognizability. A bottom-up FSTA can now be defined to check at each node that both the context-free part of a rule as well as its context conditions are satisfied.

This argument also extends easily to cover the dominance predicates of Joshi and Levy: transducers can be added to label each node with all its "top contexts" and all its "bottom- contexts" The final FSTA must then check that the nodes satisfy whatever Boolean combination of dominance and proper analysis predicates are required by the node admissibility rules.

REFERENCES

[1] Aho A.V., *Indexed grammars: an extension of the context-free grammars*, **JACM** 15, 647-671, 1968.

[2] Aho A.V., Hopcroft J.E. and Ullman J.D., **The Design and Analysis of Computer Algorithms**, Addison-Wesley, Reading Mass, 1974.

[3] Aho A.V., and Ullman J.D., **The Theory of Parsing, Translation, and Compiling, vol 1: Parsing**, Prentice Hall, Englewood Cliffs N.J., 1972.

[4] Baker B.S., *Arbitrary grammars generating context-free languages*, TR 11-72, Center for Research in Computing Technology, Harvard Univ., 1972.

[5] Berwick R. C., *Computational complexity and lexical functional grammar*, 19th ACL, 1981.

[6] Berwick R. C. and Weinberg A., *Parsing efficiency, computational complexity, and the evaluation of grammatical theories*, **Ling. Inq.** 13, 165-191, 1982.

[7] Borgida A. T., **Formal Studies of Stratificational Grammars,** PhD Thesis, University of Toronto, 1977.

[8] Chomsky N., **Rules and Representations,** Columbia University Press, 1980.

[9] Cook S. A., *Towards a complexity theory of synchronous parallel computation*, **L'Enseignement Mathématique** 27, 99-124, 1981.

[10] Earley J., *An efficient context-free parsing algorithm*, **Comm. of ACM** 13,2, 94-102, 1970.

[11] Gazdar G., *Phrase structure grammar*, in Jacobson P. and Pullum G. (eds.), **The Nature of Syntactic Representation,** Reidel, 1982.

[12] Gazdar G. and Pullum G., **Generalized Phrase Structure Grammar: A Theoretical Synopsis,** Indiana Univ. Ling. Club, 1982.

[13] Graham S. L., Harrison M. A., and Ruzzo W.L., *An improved context-free recognizer*, **ACM Trans. on Prog. Lang. and Systems**, 2, 3, 415-462, 1980.

[14] Hopcroft J.E. and Ullman J., **Introduction to Automata Theory, Languages and Computation,** Addison Wesley, 1979.

[15] Hays D.G., *Automatic language data processing*, In **Computer Applications in the Behavioral Sciences,** H. Borko (ed.), Prentice Hall, Englewood Cliffs N.J., 1962.

[16] Hintikka, J.K.K., *Quantifiers in natural languages: some logical problems 2*, **Ling. and Phil.**, 153-172, 1977.

[17] Joshi A. K., *How much context-sensitivity is required to provide reasonable structural descriptions: tree adjoining grammars*, to appear in Dowty D., Karttunen L. and Zwicky A. (eds.), **Natural Language Processing: Psycholinguistic, Computational and Theoretical Properties**, Cambridge Univ. Press.

[18] Joshi A.K., *Factoring recursion and dependencies: an aspect of Tree Adjoining Grammars and a comparison of some formal properties of TAG's, GPSG's, PLG's and LFG's*, these Proceedings, 1983.

[19] Joshi A.K. and Levy L.S., *Phrase structure trees bear more fruit than you would have thought*, 18th ACL, 1980.

[20] Joshi A.K., Levy L.S. and Takahashi M., *Tree adjunct grammars*, **J. of Comp. and Sys. Sc.** 10, 1, 136-163, 1975.

[21] Joshi A.K., Levy L.S., *Constraints on structural descriptions: local transformations*, **SIAM J. on Computing**, 1977.

[22] Joshi A.K., Levy L.S. and Yueh K., *Local constraints on programming languages, Part 1: Syntax*, **Th. Comp. Sc.** 12, 265-290, 1980.

[23] Joshi A. K. and Yokomori T, *Some characterization theorems for tree adjunct languages and recognizable sets*, forthcoming.

[24] Kaplan R. and Bresnan J., *Lexical-Functional Grammar: a formal system for grammatical representation*, in Bresnan J (ed.), **The Mental Representation of Grammatical Relations,** MIT Press, 1982.

[25] Lamb S., **Outline of Stratificational Grammar,** Georgetown University Press, Washington, 1966.

[26] Langendoen D.T., *On the inadequacy of Type-2 and Type-3 grammars for human languages*, in P.J. Hopper (ed.) **Studies in Historical Linguistics: festschrift for Winfred P. Lehman,** John Benjamin, Amsterdam, 159-171, 1977.

[27] LaPointe S., *Recursiveness and deletion*, **Ling. Anal.** 3, 227-265, 1976.

[28] Marcus M.P., **A Theory of Syntactic Recognition for Natural Language**, MIT Press, 1980.

[29] Matthews R., *Are the grammatical sentences of a language a recursive set?*, **Synthese** 40, 209-224, 1979.

[30] Montague R., *The proper treatment of quantification in ordinary English*, in Hintikka, J., Moravcsik J., and Suppes P.(eds.), **Approaches to Natural Language**, Reidel, Dordrecht, 1973.

[31] Peters P.S. and Ritchie R.W., *A note on the universal base hypothesis*, **J. of Linguistics**, 5, 150-2, 1969.

[32] Peters P.S. and Ritchie R.W., *On restricting the base component of transformational grammars*, **Inf. and Control**, 18, 483-5-1, 1971.

[33] Peters P.S. and Ritchie R.W., *On the generative power of transformational grammars*, **Inf. Sc.** *6, 49-83, 1973.*

[34] Peters P.S. and Ritchie R.W., *Context-sensitive immediate constituent analysis - context-free languages revisited*, **Math. Sys. Theory** 6, 324-333, 1973.

[35] Peters P.S. and Ritchie R.W., *Non-filtering and local filtering grammars*, in J.K.K. Hintikka, J.M.E. Moravcsik, and P. Suppes (eds.) **Approaches to Natural Language**, Reidel, 180-194, 1973.

[36] Petrick S. R., **A Recognition Procedure for Transformational Grammars,** PhD Thesis, MIT, 1965.

[37] Postal P.M., *Limitations of phrase-structure grammars*, in J.A. Fodor and J.J. Katz (eds.), **The structure of language: Readings in the philosophy of language,** Englewood Cliffs: Prentice Hall, 137-151, 1964.

[38] Pullum G.K. and Gazdar G., *Natural and context-free languages*, **Ling. and Phil.**, 4, 471-504, 1982.

[39] Putnam H., *Some issues in the theory of grammar*, in **Proc. of Symposia in Applied Mathematics,** American Math. Soc., 1961.

[40] Rounds W. C., *Mappings and grammars on trees*, **Math. Sys. Th.**,4,3, 257-287, 1970.

[41] Rounds W. C., *Tree-oriented proofs of some theorems on context-free and indexed languages*, **2nd ACM Symp. on Th. Comp. Sc.**, 109-116, 1970.

[42] Rounds W. C., *Complexity of recognition in intermediate-level languages*, **14th Symp. on Sw. and Aut. Th.** 1973.

[43] Rounds W. C., *A grammatical characterization of exponential-time languages*, **IEEE Symp. on Found. of Comp. Sc.**, 135-143, 1975.

[44] Slocum J., *A practical comparison of parsing strategies*, 19th ACL, 1981.

[45] Shieber S.M., Stucky S. U., Uszkoreit H. and Robinson J. J., *Formal constraints on metarules*, these Proceedings, 1983.

[46] Thatcher J.W., *Characterising derivation trees of context-free grammars through a generalization of finite automata theory*, **J. of Comp. and Sys. Sc.** 1, 317-322, 1967

[47] Thatcher J.W., *Generalized² sequential machine maps*, **J. of Comp. and Sys. Sc.** 4, 339-67, 1970.

[48] Thatcher J.W., *Tree automata: an informal survey*, in A. Aho (ed.), **Currents in the theory of computing,** Prentice Hall, 143-172, 1973.

[49] Uszkoreit H. and Peters P. S., *Essential variables in metarules*, forthcoming.

[50] Valiant L., *General context-free recognition in less than cubic time*, **J. of Comp. and Sys. Sc.** 10, 308-315, 1975.

[51] Warren D. S., **Syntax and Semantics of Parsing: An Application to Montague Grammar**, PhD Thesis, University of Michigan, 1979.

[52] Yokomori T. and Joshi A. K., *Semi-linearity, Parikh-boundedness and tree adjunct languages*, to appear in **Inf. Pr. Letters**, 1983.

[53] Younger D. H., *Recognition and parsing of context-free languages in time n^3*, **Inf. and Control**, 10, 2, 189-208, 1967.

[54] Kasami T., *An efficient recognition and syntax algorithm for context-free languages*, Air Force Cambridge Research Laboratory report AF-CRL-65-758, Bedford MA, 1965.

[55] Ruzzo W. L., *On uniform circuit complexity (extended abstract)*, **Proc. of 20th Annual Symp. on Found. of Com. Sc.**, 312-318, 1979.