ESSLLI 2007

19th European Summer School in Logic, Language and Information
August 6-17, 2007
http://www.cs.tcd.ie/esslli2007
Trinity College Dublin
Ireland

WORKSHOP PROCEEDINGS

ESSLLI is the Annual Summer School of FoLLI,
The Association for Logic, Language and Information
http://www.folli.org

Hodges-Figgis Bookstore

COGNITIVE SCIENCE SOCIETY

OXFORD
UNIVERSITY PRESS

Model-Theoretic Syntax at 10

James Rogers and Stephan Kepser, eds.

13–17 August 2007
organized as part of the
European Summer School on Logic, Language and Information
ESSLLI 2007
6–17 August, 2007, Dublin, Ireland
Endorsed by the
Association for Mathematics of Language
a special interest group of the
Association for Computational Linguistics

Contents

Contributors

GEOFFREY K. PULLUM
School of Philosophy, Psychology and Language Sciences, University of Edinburgh

EDWARD STABLER AND EDWARD KEENAN
UCLA Linguistics

HANS-JÖRG TIEDE Department of Mathematics and Computer Science, Illinois Wesleyan University

ADI PALM Fakultät für Informatik und Mathematik, Universität Passau

ANDERS SØGAARD Center for Language Technology, Copenhagen

RALPH DEBUSMANN Programming Systems Lab, Universität des Saarlandes

JOAN CHEN-MAIN AND ARAVIND K. JOSHI
IRCS, University of Pennsylvania

M. ANDREW MOSHIER
Department of Mathematics, Computer Science and Physics, Chapman University

GREGORY M. KOBELE, CHRISTIAN RETORÉ AND SYLVAIN SALVATI
INRIA Futurs, LaBRI (Université de Bordeaux – CNRS)

UWE MÖNNICH Linguistics Department, University of Tübingen

HANS-MARTIN GÄRTNERGÄRTNER AND JENS MICHAELIS
ZAS, Berlin and Universität Osnabrück

FRANK RICHTER Eberhard Karls Universität Tübingen

LAWRENCE S. MOSS Department of Mathematics, Indiana University

Introduction

In 1996 ESSLLI hosted a workshop on ”The Mathematics of Syntactic Structure” that covered a range of topics in the area now known as Model-Theoretic Syntax which was then just emerging. Over the ensuing decade MTS has established itself as a subdiscipline, focusing on descriptive approaches to formalizing theories of syntax by defining classes of ordinary mathematical structures directly in terms of linguistically relevant structural properties rather than in terms of generative or automata-theoretic processes. Five years later the 2001 FG/MoL meeting, affiliated with ESSLLI'01, included a symposium on the then current state of MTS.

The purpose of this workshop at ESSLLI'07 is to survey the developments in this area over its first decade and to lay the foundations for its further development in the decades to come. The workshop includes invited talks by several participants of the previous meetings as well as current papers from the broader community.

Chapter 1

The Evolution of Model-Theoretic Frameworks in Linguistics[1]

Geoffrey K. Pullum
School of Philosophy, Psychology and Language Sciences
University of Edinburgh
Edinburgh EH8 9LL, Scotland, UK
pullum@ling.ac.ed.uk

1.1 Introduction

The varieties of mathematical basis for formalizing linguistic theories are more diverse than is commonly realized. For example, the later work of Zellig Harris might well suggest a formalization in terms of CATEGORY THEORY, since Harris takes the utterances of a language to constitute a collection (possibly not even enumerable) that is not itself of particular interest, and concentrates on discussing a set of associative, composable mappings defined on it. And thoroughgoing versions of generative semantics like Pieter Seuren's seem to call for a formalization in terms of TRANSDUCERS, mapping meanings to phonetic forms and conversely. However, work in formal syntax over the past fifty years has been entirely dominated by just one kind of mathematics: the string-manipulating combinatorial systems categorized as **generative-enumerative syntax** (henceforth GES) in Pullum and Scholz (2001).

A GES grammar is a recursive definition of a specific set, *ipso facto* computably enumerable.[2] The definition is given in one of two ways: top-down or bottom-up. The classic top-down style of GES grammar is a program for a nondeterministic process of construction by successive expansion symbol strings. It amounts in effect to a program with the property that if it were left running for ever, choosing randomly but always differently among the possibilities allowed for expanding symbols, every possible string in the desired set would ultimately get constructed. Such a device is described as GENERATING the entire set of all structures the machine is capable of constructing. The production systems developed by Post (1943), developed in an effort to mathematicize the syntactic approach to logical deduction, are of this type. So are all the familiar types of phrase structure grammar, classic MIT transformational grammar (TG), Backus-Naur form, and all typical statements of the syntax of programming languages.

The other type of GES grammar consists of a finite set of primitive elements (typically a set of lexical items) and a finite set of operations for composing them into larger complex units. Such a system is taken to define the set of expressions obtained by closing the set of primitives under the combination operations. This type covers all of categorial grammar including Montagovian syntax, tree adjoining grammar, the 'minimalist program', the neominimalism of Stabler and Keenan, and nearly all statements of the formation rules for logics.

1.2 MTS frameworks

I want to try and show how the type of theoretical framework that is becoming known as MODEL-THEORETIC SYNTAX (MTS) was actually adumbrated as long ago as 1970 or even before, and a small number of linguists advocated it more explicitly by 1980, but proper mathematical development did not begin until the 1990s. But first I will sketch the hallmarks of MTS frameworks as Rogers, Scholz, and I understand them, and then take a close look at certain earlier works that represented halting steps toward MTS.

I use the term EXPRESSIONS for sentences, clauses, phrases, words, idioms, lexemes, syllables — the objects that linguists investigate. I take these to *have* syntactic structure, not merely to be analyzable in terms of structures imposed on them or posited for them by linguists. That is, I take a realist view of expressions and of their syntactic properties.

MTS frameworks, as I understand them, are distinguished by the adoption of three general positions: (I) rules are statements about expressions; (II) grammars are finite sets of such rules; (III) well-formedness of an expression consists in satisfaction of the grammar. Each of these points needs a little more discussion.

[1] This paper is based on collaborative work with James Rogers and Barbara Scholz, who should really count as co-authors. We plan a fuller discussion of these topics in a joint work (Pullum et al., in preparation), but time did not permit us to work over this presentation together, so I take responsibility for all of its errors and infelicities without taking credit for all of the thinking behind it. Barbara Scholz is thanked for detailed comments on an earlier version, and I am grateful to many other people who have helped me to understand some of the issues better, among them Gerald Gazdar, Phokion Kolaitis, Barry Mazur, Paul Postal, and James Rogers.

[2] Following Soare (1996), I consider words like 'recursive' and 'recursion' to be undesirably ambiguous, and prefer 'computably enumerable' over 'recursively enumerable'.

1.3 Rules

MTS rules are simply assertions about the structure of expressions. That is, an MTS rule makes a statement that is either true or false when evaluated in the structure of an expression. If a structure is to be grammatically well formed according to a certain rule, then the rule must be true as interpreted in that structure.

Rules within GES are not like this. A GES rule is an instruction forming part of a procedure for stepwise construction of a DERIVATION — a rule-mediated sequence of representations, the last of which is by definition well formed. Crucially, GES rules do not assert anything about the structure of well-formed expressions; they are instructions making up individual parts of an integrated procedure for building such structures, and they cannot be interpreted in isolation.

Nowhere is this clearer than in recent TG. 'Merge' cannot be understood as a condition on the structure of expressions. It is a dynamic tree-building concatenation operation, joining two items together and adding a node immediately dominating them. Notice that it is stated informally as an imperative. The same is true for 'Move α' in early TG: it is an instruction forming part of a nondeterministic random generation procedure, permitting a constituent of type α to shift to some other location at the next stage in the derivation.

The same thing is true for phrase structure rules, however. The rule 'PP → P NP' does not state that adpositions precede NPs. If the grammar contained a rule 'PP → NP P' in addition, then adpositions would be freely ordered. If it contained a rule 'P → *e*', there might be no adpositions in the generated expressions at all. Everything depends on the combined functions of the component parts of a grammar holistically defining a set.

MTS rules, by contrast, are naturally given informal statement as declarative clauses. Examples might be 'The subject noun phrase of a tensed clause is in the nominative case'; 'The head verb of the verb phrase in a tensed clause agrees in person and number with the subject of that clause'; 'Verbs always follow their direct objects'; or 'Attributive modifiers precede the heads that they modify'.

1.4 Grammars

An MTS grammar is simply a finite, unordered set of MTS rules. This means that individual rules in grammars can be developed and assessed piecemeal, without regard to any sequencing of applications.

For example, how exactly to frame the general statement of verb agreement can proceed independently of how to state the conditions on auxiliary selection in passives or positioning of relative pronouns in relative clauses. No condition on structure overrides or takes priority over another such condition. The conditions all have to be true in an expression structure if it is to count as well formed. A linguist stating a grammatical rule need only be attentive to what expressions there are and what structures they have — nothing about sequencing of operations or stages of construction is relevant.

Grammar, on the MTS view, is about *what structure expressions have*. It is not about devising a sequence of operations that would permits the construction of the entire set of all and only those structures that are grammatical.

1.5 Grammaticality

An expression is well formed according to an MTS grammar if and only if the semantic consequences of the grammar are true in its syntactic structure. Grammaticality is thus defined by reference to the SEMANTIC consequences of rules (the semantics of the formal language in which the rules are stated, that is — not the semantics of the natural language being described). An expression is fully well formed if and only if its structure complies with every requirement that is a semantic consequence of what the grammar says.

Thus a rule saying 'every direct object noun phrase in a transitive verb phrase immediately follows the verb' is satisfied only by structures in which every transitive verb phrase containing a direct object noun phrase does indeed have that noun phrase adjacent to and immediately following the verb. (The echo of Tarski's definition of truth is not just an allusion, of course; we are actually using Tarski's notion of a model here.)

The rule is vacuously true in an intransitive clause: where there is no object, there is nothing to falsify a statement fixing the positioning of objects. Ungrammaticality on the MTS view is defined by violation of one or more of the rules of the grammar.

1.6 MTS and GES

I have been trying to clarify the notion of MTS an description. This is not to be construed as claiming that such descriptions are inherently superior to non-MTS ones. They could turn out to be entirely inadequate. Linguists have made many proposals for rules or principles that that are simply impossible to state in MTS terms. For anyone who accepts these, MTS is simply untenable. One general class of examples is that MTS does not permit statement of generalizations that demand quantification over all the expressions in a language. So this forbids:
(i) the 'optionality' claim in X-bar theory that non-head constituents are always optional (see Kornai and Pullum 1990 and Pullum and Scholz 2001 for discussion);
(ii) the 'exhaustive constant partial ordering' claim (that any ordering restriction imposed on sibling constituents in a natural language must be the same under any parent node, regardless of its label; see Gazdar and Pullum 1981);
(iii) any 'ambiguity avoidance' constraint that bars structures on the basis of their being confusable with others

(Pullum and Scholz 2001 discuss a putative Russian example);
(iv) any 'blocking' constraint that bars structures on the basis that other items take priority over them;
(v) any 'economy' claim that legitimates structures by reference to the facts about alternatives being less economical.

Economy conditions, in particular, have been prominent in recent versions of GES. If any valid condition of this sort were ineliminably connected to properties that could only be stated through comparison of one structure's properties with another's, MTS would not allow for the proper description of natural language syntax at all. My belief is that not a single constraint of this sort is genuinely convincing as a part of syntax. But let there be no doubt about the fact that if there were one, MTS would have to be dismissed.

In a sense, though, MTS is founded on a very traditional idea: that a grammar should describe the syntactic structure of expressions of a language by making general statements about their syntactic properties. The rules stated in traditional grammars are of just this kind — statements imposing conditions on individual grammatical structures. And the grammatical expressions are simply those of which all the entailments of the grammar's statements are true.

Traditional grammars have been denigrated by linguists throughout most of the last century, in part because of extraneous concerns (like alleged prescriptivism) and in part because they are not explicit — their statements are not precisely stated in a formal language invented for the purpose and equipped with a denotational semantics. But the alleged failings of traditional grammar do not have to do with the idea of rules as statements about structure, or that satisfaction of the conditions is the determinant of well-formedness.

Chomsky (1962: 539) offers a revisionist view, stating that "a grammar must contain ... a 'syntactic component' that generates an infinite number of strings representing grammatical sentences," and calls such a view "the classical model for grammar." This is misleading at best. There is nothing classical about the idea that grammars should be axiomatic systems for generating infinite sets of strings. It was under ten years old when he wrote, and represented a radical break with all previous conceptions of grammar (Pullum and Scholz 2005; Scholz and Pullum 2007). Although the organic connection of GES systems to the American structuralist descriptive linguistics of the 20th century is clear, they contrast sharply with the earlier tradition of grammatical scholarship and pedagogy.

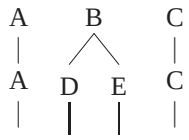
And interestingly, within about ten years after the idea of stating at least some grammatical principles as statements about expression structure began to creep back into GES.

1.7 Derivations and trees

McCawley (1968), in the context of discussing certain issues about the 'base component' in TG, raised certain doubts about whether phrase structure rules should be interpreted as rewriting instructions on strings. His paper is well known, and it has been taken to represent some kind of early adoption of the MTS point of view. I will argue that it really does not, except in an indirect way. But it does bring up some interesting and relevant issues.

As defined in 1960s TG, a rule $A \rightarrow BC$ is a rewriting operation, permitting A to be replaced by string BC in a derivation, turning a string XAY into a string $XBCY$. A separate tree-building procedure (sketched by McCawley (1968: 245) and Lasnik (2000: 17–23)) is supposed to build a parse tree from the derivation. It works from the bottom up and from the outside in, adding edges between symbols in a given line to identical symbols in the line above, and connecting up residual symbols to available nodes above. From a string *ABC*, a rule $B \rightarrow DE$ would produce *ADEC*, and the tree-building procedure would produce this intermediate stage:

(1.1)



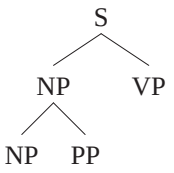
The idea is that a tree can be constructed by completing this procedure, working outside-in and bottom to top, and reducing edges like '—A—A—' to '—A—'.

But the procedure faces a problem, briefly noted by Chomsky (1959: 144, n. 8), and explored in more detail by McCawley: for some derivations it does not determine a unique tree. Worse, it may produce a tree with a structure that the rules, under their intuitive understanding, do not permit, and collapsing distinct syntactic representations.

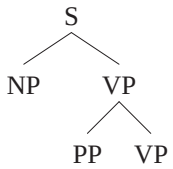
This failure to determine a unique tree stems from the fact that a derivation — the sequence of lines resulting from the rule applications — records too little information about what the rule applications were. The derivation is a record of the content of the successive lines, with no indication of which symbol was rewritten, or which rule applied, at any given stage. For example, from the partial derivation in (1.2a) the standard procedure will allow either (1.2b) or (1.2c) to be built.

(1.2) a.
$$\cdots \text{ S } \cdots$$
$$\cdots \text{ NP VP } \cdots$$
$$\cdots \text{ NP PP VP } \cdots$$

b.



c.



(Consider two possible structures for *Dogs at least bark*, one implying that there might also be other animals that bark, the other implying that there might also be other things that dogs do.)

The point is not that the problem is insoluble under the rewriting interpretation (it is not: stipulating that the symbol rewritten must always be the leftmost one for which rewriting is possible at that point permits inference of symbol was rewritten to create the current line, which permits unique correct trees to be constructed). But McCawley was interested in how trees could be more directly answerable to the content of the phrase structure rules without any such restriction on derivation construction procedures, making the connection between rules and structures directly, through a reinterpretation of phrase structure rules. This involved taking trees to be mathematically specifiable objects in themselves, rather than just diagrammatic ways of depicting properties of equivalence classes of derivations, as in Chomsky's early work (see Chomsky 1975, pp. 181ff). In this he was perhaps influenced by the work of Zwicky and Isard (1963), sent out on December 3, 1963, to a select distribution list at the MITRE Corporation, in which a set of axioms for labeled ordered trees was given and several equivalent ways of abstractly representing trees were discussed.

McCawley considered two new possibilities for interpreting of phrase structure rules. The one I will be concerned with here was suggested to him by Richard Stanley in 1965. The idea is to interpret phrase structure rules as NODE ADMISSIBILITY CONDITIONS (henceforth, NACs). An NAC is a sufficient condition for admissibility of a node given its daughter sequence. A whole tree is to be defined as well formed iff every node in it is admissible. Under this interpretation, the rule in (1.3a) would be understood as informally expressed in (1.3b):[3]

(1.3) a. S → NP VP

b. The node being evaluated is labeled 'S'; its first child is labeled 'NP'; its second child is labeled 'VP'; and there are no other child nodes.

This proposal interprets rules as monadic predicates of nodes. But it is not a proposal for MTS grammars. To see this, note that it does not respect *any* of the tenets (I)–(III).

It does not endorse (I), which says rules state necessary conditions on well-formedness of expression structures. NACs are not even defined for trees, and do not express necessary conditions anyway. Each NAC states a *sufficient* condition for admissibility of a single *node*. In consequence, (III) also fails to hold: well-formedness of a tree does not result from satisfaction of all (or in fact any) of the NACs. And strictly, the proposal in its original context did not accept (II) either: although McCawley remarks that "node admissibility conditions are by nature unordered" (p. 248), he envisions NACs in a larger context, that of providing the deep structure trees to be the inputs to the transformations, so a grammar as a whole was not envisaged as just an unordered set of NACs.

One remark McCawley makes in connection with how NACs describe trees seems to be an error: he says that "the admissibility of a tree is defined in terms of the admissibility of all of its nodes, i.e., in the form of a condition which has the form of a logical conjunction" (p. 248). It is true that each well-formed k-node tree T will be a model of a conjunction $C_{i_1}(n_1) \wedge C_{i_2}(n_2) \wedge \cdots \wedge C_{i_k}(n_k)$, where i_j is the NAC that admits the node n_j, the intuitive meaning being 'node n_1 is admissible according to NAC number i_1 and node n_2 is admissible according to NAC number $i_2 \ldots$' and so on. But it is a different statement for each tree, with a number of conjuncts corresponding to the tree size. This does not yield a general definition of well-formedness according to a grammar. Note that McCawley certainly cannot have been referring to any logical conjunction of NACs, since the conjunction of two or more distinct NACs is never true at *any* node.

The correct general definition of the set of trees defined as well formed according to a given set of NACs is in fact a DISjunction. An NAC is really a one-place predicate of nodes. For example, the NAC corresponding to the rule 'A → B C' might be expressed as (1.4), where M and $\prec$ are interpreted by the 'mother of' and 'precedes' relations.

$$\begin{aligned}(1.4)\quad \varphi(x) = {} & (\mathrm{A}(x)) \wedge \\ & ((\exists y)(\exists z)[(M(x,y)) \wedge (M(x,z)) \wedge \\ & \qquad (y \prec x) \wedge (\mathrm{B}(y)) \wedge (\mathrm{C}(z))])\end{aligned}$$

Let $\varphi_1, \ldots, \varphi_k$ be a set of such NACs. Then the formula that would need be true in a tree to make it well formed, adding the other two assumptions of McCawley's concerning the root and leaf nodes, will be (1.5), where x ranges over the node set of the tree using $S(x)$ to mean that x has the designated start symbol as its label and $T(x)$ to mean that x is labeled with a member of the terminal vocabulary:

$$\begin{aligned}(1.5)\quad & (\forall x)[\,((\mathrm{Root}(x)) \wedge S(x)) \vee && \text{[a]} \\ & \quad ((\mathrm{Leaf}(x)) \wedge T(x)) \vee && \text{[b]} \\ & \quad (\bigvee_{1 \le i \le k} \varphi_i(x))\,] && \text{[c]}\end{aligned}$$

Every node [a] is the root and is labeled with the start symbol, or [b] is a leaf node and is labeled with a terminal symbol, or [c] satisfies the disjunction of all the NACs.

Now, the set containing just this one statement (for a given $\varphi_1, \ldots, \varphi_k$) would be a very simple example of an

[3] McCawley actually introduces a special notation, '⟨S; NP VP⟩', for a phrase structure rule with this content under the Stanley interpretation, and Gazdar (1982)140 introduces another one, '[S NP VP]'. This is unnecessary, of course. Rule statements do not have to wear their interpretations on their typographical sleeves.

MTS grammar: it is a finite set of statements that may or may not be satisfied by a given tree (trivially: it is a singleton).

However, it is in fact a grammar illustrating a description language of extremely low expressive power. It interprets NACs as primitive propositions each asserting admissibility for some specific local tree. There is really no important difference between the NAC and the local tree that it uniquely describes: a grammar could be given in the form of a finite list of local trees, the interpretation being that a tree is well formed iff it is entirely composed of local trees that are on the list. This is in fact the nonstandard way of defining context-free grammars that Jim Rogers proposed at the original MTS workshop in 1996; see Rogers (1999).

As pointed out in Rogers (1997b), such a way of defining a set of trees is exactly analogous to a bigram description of a set of strings. A bigram description over an alphabet Σ is a finite list of 2-symbol sequences, and a string is grammatical according to it if every length-2 substring of the string is on the list.

But bigram descriptions define only a very small and primitive class of stringsets, the SL_2 stringsets. Local tree descriptions have much greater expressive power: every context-free stringset is the string yield of some local tree set, and every local tree set has a context-free string yield.

What McCawley apparently did not appreciate (the relevant results were not available) was that descriptions might just as well be given in a richer and more flexible description language, since no increase in weak generative capacity results from using full first-order logic on trees rather than just local tree descriptions. In fact if weak monadic second-order logic (wMSO) is used, by a crucial result that was to be obtained by Doner (1970), a stringset defined as the string yield of the set of trees satisfying some wMSO formula is always context-free, and all context-free stringsets are thus definable.

Note that the power of first-order logic on trees is sufficient to guarantee the presence of a 'trace' in some subconstituent accompanying a dislocated element, without using the GPSG device of having a chain of 'slashed categories' labeling all the nodes on the path between them. For example, to require that every constituent α contain exactly one node with the label β, we could say (writing $\mathrm{dom}(x,y)$ for 'x dominates y'):

$$(1.6)\quad (\forall x)[(\alpha(x)) \rightarrow ((\exists y)[\mathrm{dom}(x,y) \wedge \beta(y) \wedge (\forall z)[(\mathrm{dom}(x,z) \wedge \beta(z)) \rightarrow (z = y)]])]$$

The succinctness gain from the use of quantificational logic rather than just sets of NACs can be not just linear or polynomial but exponential. Jim Rogers provides the following example. Consider how to describe just a set of local trees in which the root must have m children each labeled with a distinct symbol from a list $\{A_1, \cdots, A_m\}$, any order of those children being permissible. A first-order definition needs only to say that for each of the m labels there is exactly one child with that label. That can be done with a formula of a length linearly related to m: the formula has to say for each node x that there is an A_i such that $[A_i(x) \wedge (\forall y)[(y \not\approx x) \Rightarrow \neg(A_i(x))$ (where $1 \leq i \leq m$). But the number of distinct local trees involved, and hence the size of a grammar composed of NACs, grows as an exponential function of m (linearly related to $m!$, in fact).

In short, the Stanley/McCawley proposal for reinterpreting phrase structure rules is a very interesting idea, but its role as a precursor of MTS should not be overstated, because virtually none of the MTS program is implicit in what McCawley actually suggested.

1.8 Tree sequence models

Just two or three years after McCawley's paper we find the earliest published work in linguistics that can be said to adopt all three of the hallmarks of MTS. Lakoff (1969) and the more accessible Lakoff (1971), no doubt influenced by McCawley's paper, were well known and much-cited papers, and presented a radical departure from the standard way to formalize syntactic theory. But there was essentially no effect on subsequent work. The reasons are many, but it has to be said that Lakoff's ideas were ill-delineated and sloppily illustrated. If his proposals would work at all, which is doubtful, it would apparently have been of vitiatingly unrestrained expressive power.

Lakoff's reformulation of TG was put forward in the course of a defense of the generative semantics hypothesis. To remain close to the standard assumptions of the time concerning the content of syntactic description, the syntactic structures Lakoff posited were finite sequences of finite trees, exactly as in TG. Most linguists at the time thought of transformations intuitively as operations applying to trees and producing modified trees. This was not the way Chomsky had initially formalized transformations, but it proved by far the most intuitive way to think of them. Thus a set of transformations, together with various principles governing the order and domain of their application, would determine for a given tree the structural properties of the next one in the derivation. The central challenge in Lakoff's reformulation was to be able to represent what the transformational rules express concerning which trees can follow which trees in a syntactically permissible sequence.

Lakoff proposed that transformations should be stated simply as conditions on pairs of successive trees in a sequence. He remarks (using the term 'phrase-markers' for trees):

> Since transformations define possible derivations only by constraining pairs of successive phrase-markers, I will refer to transformations as 'local derivational constraints'. ((Lakoff 1971: 233)

Lakoff defines a local derivational constraint as a conjunction of two statements C_1 and C_2, "where C_1 and C_2 are tree-conditions defining the class of input trees and class of output trees, respectively" (1971: 233).

The problem is that fixing the properties shared by the input trees and the properties shared by the output trees cannot possibly suffice to mimic the effect of a transformation. Chomsky (1972) briefly points this out, but the point is developed more fully in the only detailed critical study of Lakoff's proposals that I am aware of, Soames (1974). It is not necessary to go into matters of fine detail to see what the problem is. A conjunction of conditions saying anything remotely like what Lakoff suggests — that the input tree meets condition C_1 and the output tree meets condition C_2 — will associate *any* tree satisfying C_1 with *every* tree satisfying C_2.

As Chomsky notes (1972: 121, n. 19), Lakoff's formulation "would not, for example, distinguish between the identity transformation and a permutation of two nodes of the same category". That is, a transformation that would derive *They spoke about it to the others* from *They spoke to the others about it* could not be distinguished from a transformation that simply maps a structure containing two PPs to itself. This gives Lakoff unacknowledged technical problems like how to block infinite derivations of finite strings from meeting the definition of well-formedness, and how to define 'completed derivation'.

The problems run much deeper than that. Lakoff's reformulation of transformations does not guarantee conservativism, in the sense that it does not prevent wholesale change of lexical content in a derivational step. By allowing a tree representing *This, I believe* as an optional transform of *I believe this*, a grammar would also allow infinitely many other trees, with terminal strings like *That, they rejected* or *The others, we keep in the bathroom*, to count as well.

What is missing is what is guaranteed by the carry-over of already-written structure in the stepwise construction of transformational derivations: that a tree is rewritten in a way that alters it only in one specific way at each transformational step. The structures Lakoff actually desires to capture are characterized by a default which amounts to a massive redundancy: each tree is identical with the one preceding it in the sequence, except with regard to one part (typically quite small) where there is a specified change. (Lakoff may be acknowledging this when he remarks that one part of the condition C_1, repeated in C_2, "defines the part of the tree-condition which characterizes both" of two adjacent trees. But perceiving that something needs to be done is not the same as doing it.)

Thompson (1975) makes an attempt to work out Lakoff's ideas in more detail. Specifically, he aims to characterize the principle of CYCLIC APPLICATION in terms compatible with Lakoff's proposals. (The fact that Lakoff offers no way to express the cyclic principle is noted by Soames 1974, p. 122, n. 6.)

Thompson, referring to the framework as CORRESPONDENCE GRAMMAR, assumes that each pair of adjacent trees in a well-formed sequence must be explicitly licensed by what he calls a VALIDATING RULE. He recognizes the need "to prevent extraneous changes from occurring in the derivation" — that is, to block random differences between trees and their immediate predecessors or successors that the validating rule says nothing about — so he states a global *ceteris paribus* condition on tree sequences. He assumes that for any two finite trees a finite statement of their node-by-node differences can be stated in unique form, and so will I; call this the DIFFERENCE SET for the two trees. Thompson's own statement of the condition is partly procedural, but repairing that we can restate it thus:

(1.7) For each pair of adjacent trees T_{i-1} and T_i licensed by a validating rule R_i, any tree T_i' that is the result of eliminating from T_i some subset D of differences of their difference set, making T_i' more similar to T_{i-1}, is such that the rule R_i does not license the pair consisting of T_{i-1} and T_i'.

This says that if the second tree in an adjacent pair were altered in any way that made it more similar to the first, the validating rule for the pair would no longer apply.

I note in passing that this "explicit way of seeing to it that 'everything else remains the same' " (Thompson, p. 597) yields an exponential explosion of complexity in the problem of checking a model for compliance with a grammar. Verifying this for a given pair of trees involves checking NON-satisfaction of the relevant validating rule for a set of pairs of trees of cardinality related to the power set of the difference set. The number of subsets of a difference set of size d will be 2^d, so it must be established for each of 2^d tree pairs that the validating rule fails to license them. And this must be done for the entire set of pairs in the derivation.

Let me also note that although Thompson's ultimatel goal was to exhibit a "formal statement" of the principle of cyclic application, what he actually offers is a highly informal statement in English that is not a condition on structures at all, but a condition on validating rules, and almost certainly not even a decidable one, since it involves testing whether a rule "could apply to some tree in that derivation in two different ways, such that some variable in the rule instantiated to two different nodes in the tree *of different depth*" (emphasis in original), and given the Turing-equivalent power of the systems Lakoff is trying to mimic in MTS mode, this cannot be decidable, by Rice's theorem (see Hopcroft and Ullman 1979: 185–189.)

Problems similar to those that arise in trying to state the cyclic principle also arise in trying to define rule ordering, optional versus obligatory application, and other matters (see Soames 1974). The bottom line is that it is wildly off the mark to suggest, as Postal (1972: 139) did, that Lakoff's work provides "a fundamental theoretical clarification".[4]

[4] Postal apparently said this because he agreed with Lakoff that rule ordering in a GES grammar should be regarded as just another descriptive device like positing an extra rule. This seems sensible. But since Lakoff had no workable way of representing ordering of transformations, he can hardly be credited with having provided a theoretical clarification of them.

I am not suggesting that Lakoff's project was inherently impossible to execute. It might have been feasible. One clearly needed improvement was provision of a way to identify *corresponding nodes* in different trees in the sequence directly (see Soames 1974: 127 on this). Lakoff seems to presuppose that corresponding nodes can be located when necessary, but he does not define a correspondence relation that might make it feasible. Potts and Pullum (2002), in the course of applying MTS description to the content of constraints in optimality-theoretic phonology, assume structures that are in effect tree pairs with an added correspondence relation $\mathfrak{R}$ defined between the nodes in the first tree and the nodes in the second. Lakoff could have taken structures to be single connected graphs — tree sequences with the nodes linked to corresponding nodes in successor trees by the $\mathfrak{R}$ relation.

An analog of the highly complex *ceteris paribus* condition would still be needed, representing an embarrassingly massive redundancy in the structural representations involved. And it still needs to be shown that TG could be recast with identical descriptive power in terms of sets of conditions on graphs of the relevant sort. Lakoff cannot be taken to have done anything more than adumbrate the approach. As Zwicky (1972: 106) remarks, it is unfortunate that Lakoff and others who read it "responded to the tone of Chomsky's article rather than to its actual content."

1.9 More recent work

The failure of Lakoff's project might look like a very serious strike against the idea of a grammar as a set of constraints if the models for constraints in natural language syntax had to be of the sort Lakoff assumes. But of course they do not.

Lakoff's reasons for assuming models having the form of transformational derivations (tree sequences with some way of determining a counterpart relation for most of the nodes between successive trees) appear to have been bound up with his effort to show, within the analytical framework of TG, that the assumptions of Chomsky (1965) led inexorably toward the generative semantics hypothesis. Lakoff had a primarily rhetorical motive, in other words: he wanted to reduce to meaninglessness the question of whether deep structures or semantic representations are 'generated first'. He was not concerned with the question of whether simpler models of the structure of expressions might suffice for adequate descriptions of natural languages. That possibility was to be explored in later research.

Subsequent work by Freidin (1975), Brame (1976), and Bresnan (1978) led to a widespread weakening of the idea that transformational description was inevitable; and Gazdar (1981) finally made it explicit that perhaps purely context-free description had been much underestimated.

The 1980s saw a diverse array of developments in syntactic theory (see Jacobson and Pullum (1982) for a snapshot of the field at the beginning of the decade), but the most relevant in the present context was the arc pair grammar (APG) framework of Johnson and Postal (1980). This was the first moderately complete proposal for an MTS syntactic framework. It emerged from the relational grammar tradition, but was hardly in contact with the rest of linguistics at all.

In APG, a structure is a triple $\mathcal{A} = \langle A, R_s, R_e \rangle$, where A is a set of arcs (roughly, an arc is an edge labeled with a grammatical relation like 'subject-of' and a sequence of stratum indices) and R_s and R_e are binary relations. (R_s is called **sponsor**: an arc A_1 sponsors an arc A_2 iff the presence of A_1 is a necessary condition for A_2 to be in the structure. R_e is called **erase**: A_1 erases A_2 iff the presence of A_1 is sufficient condition for A_2 to have no relevance to superficial properties like word order, morphology, and phonology.[5]

Johnson and Postal state, in what appears to be a first-order language enriched with the power to define reflexive transitive closures, a large number of proposed universal laws of syntax and a number of proposed rules for English and other languages, but they also draw (chap. 14) a number of consequences from the idea of MTS theories, such as the observation that rules and universal principles can be stated in exactly the same logical language and have models of exactly the same sort; the point that multiple coordination with branching of unbounded degree becomes easily describable; and the suggestion that syntax can be separated completely from the lexicon, making possible an explanation of the intelligibility of expressions containing nonsense words.

Meanwhile there was a surprising (and temporary) turn taken by TG during the early 1980s, when GB started framing a significant part of the general theory of syntax in declarative terms ('An anaphor is bound in its governing category', and so on). However, there was no attempt by the practitioners of such theories to formalize them, and while the binding theory and case theory seemed implicity model-theoretic in conception, X-bar theory and Move Alpha were clearly GES ideas. GB was an rather informally developed hybrid framework: a little casual declarative superstructure built on top of an underlyingly procedural core. The conceptual nod toward the idea of giving theories in a form that involves statements about structures can be acknowledged, but it would be too great a stretch to call GB a part of the MTS project.

Elsewhere during the later 1980s there were only occasional hints of the MTS perspective, often in unpublished or fairly obscure work: the lectures Gerald Gazdar gave in 1987 advocating a fully satisfaction-based formalization of GPSG; the ideas Ron Kaplan expressed in the late 1980s concerning LFG as using a quantifier-free

[5] Johnson and Postal do not put things quite like this. They treat nodes as primitive, and define an arc as a pair of nodes associated with a grammatical relation name and a sequence of stratum indices, and then with some awkwardness treat **sponsor** and **erase** as higher-order relations of some kind. It seems preferable to formalize the theory in terms of edges as primitives, as Postal has suggested in unpublished work.

equational logic on complex models incorporating functions (see Kaplan (1995), which dates from 1989); the far-sighted work by Paul John King (1989) on development of an MTS formalization of HPSG; and so on.

Basically, MTS as a full-fledged variety of linguistic theorizing can be said to have begun with Johnson and Postal (1980). So there is a sense in which MTS is not just 10 years old this year, but more like 30. But it is of course artificial to give precise ages to intellectual movements. Like words and phrases in the history of a language, they always turn out to be a little older than the last investigator thought. What is certainly clear is that the MTS project mostly languished between 1980 and about 1993. Hardly anybody paid attention to arc pair grammar, and the one or two who did (e.g., Judith Aissen) were interested in its hypotheses about syntactic structure and its inventory of conjectured syntactic universals (see Aissen 1987 for an essentially unique APG-based descriptive study).

It was only in the 1990s, as computational linguists with a training in logic became involved, that MTS work with some real mathematical and logical sophistication began to emerge. A partial timeline:

1993: Kracht (1993) (partly inspired by Barker and Pullum 1990) and Blackburn et al. (1993) (re-formalizing Gazdar et al. 1985), both from German institutions and using modal logic on tree models, presented at the 6th EACL meeting in Dublin.

1994: James Rogers completes a dissertation at the University of Delaware (Rogers, 1994) using wMSO on tree models; Blackburn, Kracht, and Rogers meet at a workshop in Amsterdam ('Logic, Structures and Syntax, at the Centrum voor Wiskunde en Informatica, September 26–28).

1995: Backofen et al. (1995) publish their first-order axiomatization of the theory of finite trees; Kracht publishes two relevant papers (Kracht 1995a; Kracht 1995b; Kaplan (1995) publishes a clearly MTS-oriented statement of the bases of lexical functional grammar and (Blackburn and Gardent, 1995) publish a different reformalization using modal logic.

1996: Rogers presents a paper at the first conference on Logical Aspects of Computational Linguistics (LACL); ESSLLI (in Prague; see `http://folli.loria.fr/esslliyear.php?1996`) features an advanced course by Rogers called 'Topics in Model-Theoretic Syntax' — a title that Rogers proposed as a joke but was persuaded by Blackburn to keep — and also a workshop organized by Uwe Mönnich and Hans Peter Kolb of Tübingen under the title 'The Mathematics of Syntactic Structure'.

1997: proceedings of the 1994 Amsterdam workshop appear as Blackburn and de Rijke (1997); Rogers' term 'model-theoretic syntax' appears in print for the first time in the title of Blackburn and Meyer-Viol (1997); Rogers' LACL paper published as Rogers (1997b); Rogers (1997a) uses MTS to reformalize aspects of GPSG.

The explosion of MTS publication in 1997 makes it very appropriate to be holding a tenth-anniversary reunion in 2007. I have tried to point out in the brief historical review above, however, is that the flowering of this work that began in the middle 1990s was related to seeds planted some thirty years before. They were planted in stony ground, only inexpertly tended, and inadequately watered, but they were planted nonetheless. There is now an increasingly luxuriant garden to explore.

Bibliography

Aissen, Judith (1987). *Tzotzil Clause Structure*. Kluwer Academic, Dordrecht.

Backofen, Rolf, James Rogers, and K. Vijay-shanker (1995). A first-order axiomatization of the theory of finite trees. *Journal of Logic, Language, and Information*, **4**:5–39.

Barker, Chris and Geoffrey K. Pullum (1990). A theory of command relations. *Linguistics and Philosophy*, **13**:1–34.

Blackburn, Patrick and Maarten de Rijke, eds. (1997). *Specifying Syntactic Structures*. Studies in Logic, Language and Information. CSLI Publications and FoLLI, Stanford, CA.

Blackburn, Patrick and Claire Gardent (1995). A specification language for lexical functional grammars. In *Seventh Conference of the European Chapter of the Association for Computational Linguistics*, pp. 39–44. ACL, Morristown, NJ.

Blackburn, Patrick, Claire Gardent, and Wilfried Meyer-Viol (1993). Talking about trees. In *Sixth Conference of the European Chapter of the Association for Computational Linguistics*, pp. 21–29. ACL, Morristown, NJ.

Blackburn, Patrick and Wilfried Meyer-Viol (1997). Modal logic and model-theoretic syntax. In M. de Rijke, ed., *Advances in Intensional Logic*, pp. 29–60. Kluwer Academic, Dordrecht.

Brame, Michael K. (1976). *Conjectures and Refutations in Syntax and Semantics*. Elsevier North-Holland, New York.

Bresnan, Joan W. (1978). A realistic transformational grammar. In *Linguistic Theory and Psychological Reality*, pp. 1–59. MIT Press, Cambridge, MA.

Chomsky, Noam (1956). The logical structure of linguistic theory. Unpublished dittograph, microfilmed; revised version of a 1955 unpublished manuscript.

Chomsky, Noam (1959). On certain formal properties of grammars. *Information and Control*, **2**:137–167.

Chomsky, Noam (1962). Explanatory models in linguistics. In Ernest Nagel, Patrick Suppes, and Alfred Tarski, eds., *Logic, Methodology and Philosophy of Science: Proceedings of the 1960 International Congress*, pp. 528–550. Stanford University Press, Stanford, CA.

Chomsky, Noam (1965). *Aspects of the Theory of Syntax*. MIT Press, Cambridge, MA.

Chomsky, Noam (1972). Some empirical issues in the theory of transformational grammar. In P. Stanley Peters, ed., *Goals of Linguistic Theory*. Holt Rinehart and Winston, New York.

Chomsky, Noam (1975). *The Logical Structure of Linguistic Theory*. Plenum, New York. Published version of Chomsky (1956).

Doner, John (1970). Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, **4**:406–451.

Freidin, Robert (1975). The analysis of passives. *Language*, **51**:384–405.

Gazdar, Gerald (1981). Unbounded dependencies and coordinate structure. *Linguistic Inquiry*, **12**:155–184.

Gazdar, Gerald (1982). Phrase structure grammar. In Pauline Jacobson and Geoffrey K. Pullum, eds., *The Nature of Syntactic Representation*, pp. 131–186. D. Reidel, Dordrecht, Netherlands.

Gazdar, Gerald, Ewan Klein, Geoffrey K. Pullum, and Ivan A. Sag (1985). *Generalized Phrase Structure Grammar*. Basil Blackwell, Oxford.

Gazdar, Gerald and Geoffrey K. Pullum (1981). Subcategorization, constituent order, and the notion 'head'. In Michael Moortgat, Harry van der Hulst, and Teun Hoekstra, eds., *The Scope of Lexical Rules*, pp. 107–123. Foris, Dordrecht, Netherlands.

Hopcroft, John E. and Jeffrey D. Ullman (1979). *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley.

Jacobson, Pauline and Geoffrey K. Pullum, eds. (1982). *The Nature of Syntactic Representation*. D. Reidel, Dordrecht. Republished by Springer Verlag.

Johnson, David E. and Paul M. Postal (1980). *Arc Pair Grammar*. Princeton University Press, Princeton, NJ.

Kaplan, Ronald (1995). The formal architecture of lexical-functional grammar. In Mary Dalrymple, Ronald M. Kaplan, John T. Maxwell III, and Annie Zaenen, eds., *Formal Issues in Lexical-Functional Grammar*, pp. 7–27. CSLI Publications, Stanford, CA. Earlier versions appeared in *Proceedings of ROCLING II*, ed. by C.-R. Huang and K.-J. Chen (Taipei, Republic of China, 1989), 1–18, and in *Journal of Information Science and Engineering* **5**, 1989, 305–322.

King, Paul John (1989). *A Logical Formalism for Head-driven Phrase Structure Grammar*. Ph.D. thesis, University of Manchester, Manchester, UK.

Kornai, Andràs and Geoffrey K. Pullum (1990). The X-bar theory of phrase structure. *Language*, **66**:24–50.

Kracht, Marcus (1993). Mathematical aspects of command relations. In *Sixth Conference of the European Chapter of the Association for Computational Linguistics: Proceedings of the Conference*, pp. 240–249. Association for Computational Linguistics, Morristown, NJ.

Kracht, Marcus (1995a). Is there a genuine modal perspective on feature structures? *Linguistics and Philosophy*, **18**:401–445.

Kracht, Marcus (1995b). Syntactic codes and grammar refinement. *Journal of Logic, Language and Information*, **4**:41–60.

Lakoff, George (1969). On derivational constraints. In Robert I. Binnick, Alice Davison, Georgia Green, and Jerry Morgan, eds., *Papers from the Fifth Regional Meeting, Chicago Linguistic Society*, pp. 117–139. Department of Linguistics, University of Chicago, Chicago, Illinois.

Lakoff, George (1971). On generative semantics. In Danny D. Steinberg and Leon A. Jakobovitz, eds., *Semantics: An Interdisciplinary Reader in Philosophy, Linguistics and Psychology*, pp. 232–296. Cambridge University Press, Cambridge.

Lasnik, Howard (2000). Syntactic Structures *Revisited: Contemporary Lectures on Classic Transformational Theory*. MIT Press, Cambridge, MA.

McCawley, James D. (1968). Concerning the base component of a transformational grammar. *Foundations of Language*, **4**:243–269. Reprinted in James D. McCawley, *Grammar and Meaning*, 35–58 (New York: Academic Press; Tokyo: Taishukan, 1973).

Post, Emil (1943). Formal reductions of the general combinatory decision problem. *American Journal of Mathematics*, **65**:197–215.

Postal, Paul M. (1972). The best theory. In P. Stanley Peters, ed., *Goals of Linguistic Theory*, pp. 131–179. Prentice-Hall, Englewood Cliffs, NJ.

Potts, Christopher and Geoffrey K. Pullum (2002). Model theory and the content of OT constraints. *Phonology*, **19**:361–393.

Pullum, Geoffrey K., James Rogers, and Barbara C. Scholz (in preparation). *Model-Theoretic Syntax*. Oxford University Press, Oxford.

Pullum, Geoffrey K. and Barbara C. Scholz (2001). On the distinction between model-theoretic and generative-enumerative syntactic frameworks. In Philippe de Groote, Glyn Morrill, and Christian Retoré, eds., *Logical Aspects of Computational Linguistics: 4th International Conference*, number 2099 in Lecture Notes in Artificial Intelligence, pp. 17–43. Springer, Berlin and New York.

Pullum, Geoffrey K. and Barbara C. Scholz (2005). Contrasting applications of logic in natural language syntactic description. In Petr Hajek, Luis Valdes-Villanueva, and Dag Westerstahl, eds., *Proceedings of the 13th International Congress of Logic, Methodology and Philosophy of Science*, pp. 481–503. KCL Publications, London.

Rogers, James (1994). *Studies in the Logic of Trees with Applications to Grammar Formalisms*. Ph.D. thesis, University of Delaware, Newark, DE.

Rogers, James (1997a). "Grammarless" phrase structure grammar. *Linguistics and Philosophy*, **20**:721–746.

Rogers, James (1997b). Strict LT_2 : regular :: local : recognizable. In Christian Retoré, ed., *Logical Aspects of Computational Linguistics: First International Conference, LACL '96 (Selected Papers)*, number 1328 in Lecture Notes in Artificial Intelligence, pp. 366–385. Springer, Berlin and New York.

Rogers, James (1999). The descriptive complexity of generalized local sets. In Hans-Peter Kolb and Uwe Mönnich, eds., *The Mathematics of Syntactic Structure: Trees and their Logics*, number 44 in Studies in Generative Grammar, pp. 21–40. Mouton de Gruyter, Berlin.

Scholz, Barbara C. and Geoffrey K. Pullum (2007). Tracking the origins of transformational generative grammar. *Journal of Linguistics*, **43**(3). In press.

Soames, Scott (1974). Rule orderings, obligatory transformations, and derivational constraints. *Theoretical Linguistics*, **1**:116–138.

Soare, Scott (1996). Computability and recursion. *Bulletin of Symbolic Logic*, **2**:284–321.

Thompson, Henry (1975). The cycle: a formal statement. In Robin E. Grossman, L. James San, and Timothy J. Vance, eds., *Papers from the Eleventh Regional Meeting, Chicago Linguistic Society*, pp. 589–603. Chicago Linguistic Society, Chicago, Illinois.

Zwicky, Arnold M. (1972). Remarks on directionality. *Journal of Linguistics*, **8**:103–109.

Zwicky, A. M. and S. Isard (1963). Some aspects of tree theory. Working Paper W-6674, The MITRE Corporation, Bedford, MA.

Chapter 2

Universals across languages

Edward Stabler and Edward Keenan
UCLA Linguistics
Los Angeles, California, USA
stabler@ucla.edu, keenan@humnet.ucla.edu

Introduction

One motivation for model theoretic approaches to syntax is the prospect of enabling us to "abstract fully away from the details of the grammar mechanism – to express syntactic theories purely in terms of the properties of the class of structures they license" (Rogers, 1996). This is a worthy goal: in order to see the significant relations among expressions and their parts more clearly, and to describe similarities among different structures and different languages, we would like to discard those aspects of generative, derivational history which appear just because of our decision to use some particular generative device to specify it. If this is our goal, then although it is known that the derivation trees (or derived trees, or other closely related sets of structures) of various generative formalisms can be defined model-theoretically (Büchi, 1960; Thatcher and Wright, 1968; Doner, 1970; Thomas, 1997), that is not generally what we want. We want something more abstract; we want structures that "abstract fully away from ... the grammar mechanism." What are those structures? This paper takes some first, standard steps towards an algebraic, group-theoretic perspective on this question.

A generative grammar can be given by a lexicon *Lex* and some generating functions $\mathcal{F}$, defining the language L which is the closure of *Lex* with respect to $\mathcal{F}$. The structure building functions of most grammars are *partial*, that is, they apply to some but not other expressions, and typically the domains of the functions are picked out by "syntactic categories" and "syntactic features." This partiality is a very important part of grammar!

Since the structure building rules in $\mathcal{F}$ define the structure of the language, we **set the stage** for our analysis by requiring the grammars to be "balanced" in a sense defined below, with rules $\mathcal{F}$ that are neither too specific nor too general. (Few of the grammars popular in mainstream syntax are balanced in this sense, but balanced formulations are often easily obtained.) Then, in a **first** step towards a suitably abstract perspective, define the *structural* elements of a language (lexical items, properties, relations) to be those that are fixed by every automorphism of $(L, \mathcal{F})$. Two expressions then have the "same structure" if some automorphism maps one to the other. The automorphisms of course form a group with respect to composition, and so we have an instance of the familiar framework for the study of symmetries (Klein, 1893). This perspective stands well away from particular grammars with which we started, in a number of senses that we briefly explore. Although it conforms at many points with linguists' intuitions about structure, a derivation tree of a particular grammar, if interpreted in the traditional linguistic fashion, can actually be misleading about the "structure" the grammar defines, in our sense.

The automorphisms Aut_G of each grammar G are still very sensitive to small changes in the language though. In order to compare similar but non-identical grammars, we take a **second** step, again using standard concepts, finding homomorphisms that relate structural polynomials of the languages. Then we achieve a perspective in which we can recognize different languages, with different signatures, as related by homomorphisms that preserve certain 'minimal' or 'core' structures of predication and modification, even when they are realized in slightly different ways. This allows a precise formulation of some of the basic common properties that linguists notice in grammars of diverse languages.

2.1 Grammars and structure

For $Lex \subseteq \mathrm{E}$ and $\mathcal{F} = \langle f_1, f_2, \ldots \rangle$ a sequence of *partial* functions $f_i : \mathrm{E}^n \to \mathrm{E}$, we regard each $f_i : \mathrm{E}^n \to \mathrm{E}$ as a set of $n+1$-tuples, as usual. Let $[Lex]_{\mathcal{F}}$ represent the closure of *Lex* with respect to the functions in $\mathcal{F}$. Then we can regard a grammar $G = (Lex_G, \mathcal{F}_G)$ as defining the language $[Lex_G]_{\mathcal{F}}$ with structure $\mathcal{F}_G$. (When no confusion will result, we sometimes leave off subscripts.)

For example, consider Span=$(Lex, \mathcal{F})$ defined as follows (Keenan and Stabler, 2003, §4.2). Let Σ={every, some, very, gentle, intelligent, -a, -o, man, doctor, woman, obstetrician}, Cat={D, Dm, Df, Nm, Nf, M, A, Am, Af, Agrm, Agrf, NPm, NPf}, and $\mathrm{E} = \Sigma^* \times Cat$ as usual. Then let the lexicon $Lex \subseteq E$ be the following set of 12 elements

$$\begin{array}{lll} Lex = \{ & \langle \text{some}, \text{D} \rangle, & \langle \text{every}, \text{D} \rangle, \\ & \langle \text{very}, \text{M} \rangle, & \langle \text{moderately}, \text{M} \rangle, \\ & \langle \text{intelligent}, \text{A} \rangle, & \langle \text{gentle}, \text{A} \rangle, \\ & \langle \text{-o}, \text{Agrm} \rangle, & \langle \text{-a}, \text{Agrf} \rangle, \\ & \langle \text{man}, \text{Nm} \rangle, & \langle \text{doctor}, \text{Nm} \rangle, \\ & \langle \text{woman}, \text{Nf} \rangle, & \langle \text{obstetrician}, \text{Nf} \rangle \ \}. \end{array}$$

We let $\mathcal{F} = \langle g, m \rangle$, where g gender-marks determiners D and adjectives A as follows, for any $s, t \in \Sigma^*$, writing st

for their concatenation:

$$\langle\langle \mathrm{s,A}\rangle,\langle \mathrm{t,Agrm}\rangle\rangle \mapsto \langle \mathrm{st,Am}\rangle$$
$$\langle\langle \mathrm{s,A}\rangle,\langle \mathrm{t,Agrf}\rangle\rangle \mapsto \langle \mathrm{st,Af}\rangle$$
$$\langle\langle \mathrm{s,D}\rangle,\langle \mathrm{t,Agrm}\rangle\rangle \mapsto \langle \mathrm{st,Dm}\rangle$$
$$\langle\langle \mathrm{s,D}\rangle,\langle \mathrm{t,Agrf}\rangle\rangle \mapsto \langle \mathrm{st,Df}\rangle,$$

and then phrases are merged together by m as follows,

$$\langle\langle \mathrm{s,M}\rangle,\langle \mathrm{t,Am}\rangle\rangle \mapsto \langle \mathrm{st,Am}\rangle$$
$$\langle\langle \mathrm{s,M}\rangle,\langle \mathrm{t,Af}\rangle\rangle \mapsto \langle \mathrm{st,Af}\rangle.$$
$$\langle\langle \mathrm{s,Am}\rangle,\langle \mathrm{t,Nm}\rangle\rangle \mapsto \langle \mathrm{st,Nm}\rangle$$
$$\langle\langle \mathrm{s,Af}\rangle,\langle \mathrm{t,Nf}\rangle\rangle \mapsto \langle \mathrm{st,Nf}\rangle$$
$$\langle\langle \mathrm{s,Dm}\rangle,\langle \mathrm{t,Nm}\rangle\rangle \mapsto \langle \mathrm{st,NPm}\rangle$$
$$\langle\langle \mathrm{s,Df}\rangle,\langle \mathrm{t,Nf}\rangle\rangle \mapsto \langle \mathrm{st,NPf}\rangle$$

Lifting any function on E to apply coordinatewise to tuples in E^*, and then pointwise to sets of expressions or tuples of expressions, an *automorphism* $h : [Lex] \to [Lex]$ of $([Lex],\mathcal{F})$ is a bijection such that for every $f \in \mathcal{F}$, $h(f \restriction [Lex]) = f \restriction [Lex]$.

For x an expression, a tuple of expressions, a set of expressions, or set of tuples, we say x is *structural* iff x is fixed by every automorphism. And *x has the same structure as y* iff there is some automorphism h such that $h(x) = y$.

For any $\mathrm{E} = \Sigma \times Cat$, $Lex \subseteq \mathrm{E}$ and partial functions $\mathcal{F}$, consider the grammar $G = (Lex, \mathcal{F})$. For any $\mathrm{C} \in Cat$ let the phrases of category C

$$\mathrm{PH(C)} = \{\langle \mathrm{s,D}\rangle \in [Lex] |\ D = C\}.$$

Keenan and Stabler (2003) show that with the grammar Span,

- *Lex* is structural, as are $\mathrm{PH}(A), \mathrm{PH}(D), \mathrm{PH}(M)$.
- There is an automorphism that exchanges $\langle \mathrm{every,D}\rangle$ and $\langle \mathrm{some,D}\rangle$, exchanging all occurrences of the vocabulary elements *every* and *some* in the strings of each expression but leaving everything else unchanged. The existence of this automorphism establishes that $\langle$every -a very gentle -a woman, NPf$\rangle$ and $\langle$some -a very gentle -a woman, NPf$\rangle$ have the same structure.
- There are other automorphisms that exchange the masculine and feminine phrases. For example, let's define the string homorphism $\cdot^{mf}$ that exchanges these substrings:

  $$\text{-a} \leftrightarrow \text{-o}$$
  $$\text{man} \leftrightarrow \text{woman}$$
  $$\text{doctor} \leftrightarrow \text{obstetrician}$$

  And then let's extend that mapping to exchange the following categories:

  $$\mathrm{Agrm} \leftrightarrow \mathrm{Agrf}$$
  $$\mathrm{Nm} \leftrightarrow \mathrm{Nf}$$
  $$\mathrm{Dm} \leftrightarrow \mathrm{Df}$$
  $$\mathrm{Am} \leftrightarrow \mathrm{Af}$$
  $$\mathrm{NPm} \leftrightarrow \mathrm{NPf}$$

  Then define the total function $h : [Lex] \to [Lex]$ as follows:

  $$h(s,C) = (s^{mf}, C^{mf}).$$

  This function is an automorphism of Span (Keenan and Stabler, 2003, p.143).

  So $\langle$every -a very gentle -a woman, NPf$\rangle$ and $\langle$every -o very gentle -o man, NPm$\rangle$ have the same structure.

For any $G = (Lex, \mathcal{F})$, let Aut_G be the set of automorphisms of G. Clearly, $\langle \mathrm{Aut}_G, \circ\rangle$ is a group, since Aut_G includes the identity on $[Lex]$ which is also the identity with respect to composition of automorphisms, and for any automorphism, its inverse is also an automorphism (Grätzer, 1968; Plotkin, 1972).

It will be convenient to introduce some 'auxiliary' functions. An n-ary *projection function* is a total function $\varepsilon^n_i : \mathrm{E}^n \to \mathrm{E}$, for $0 < i \leq n$, defined by

$$\varepsilon^n_i(x_1,\ldots,x_i,\ldots,x_n) = x_i.$$

The set $poly(G)$ of *polynomials over* $G = (A, \mathcal{F})$ is the smallest set containing the projection functions and such that if $p_1,\ldots,p_n$ are n-ary (partial) polynomials, and m-ary (partial) $f \in \mathcal{F}$, then $f(p_1,\ldots,p_m)$ is an n-ary (partial) polynomial, whose domain is the set of $s \in \mathrm{E}^n$ such that, for $0 < i \leq m$,

$$s \in \mathrm{dom}(p_i) \text{ and } \langle p_1(s),\ldots,p_m(s)\rangle \in \mathrm{dom}(f),$$

and where the values of the polynomial are given by

$$f(p_1,\ldots,p_m)(s) = f(p_1(s),\ldots,p_m(s)).$$

So for example, the expression $\langle$every -a very gentle -a woman, NPf$\rangle$, derived in Figure 1, is the value of the 6-ary polynomial

$$m(g(\varepsilon^6_1,\varepsilon^6_2), m(m(\varepsilon^6_3, g(\varepsilon^6_4,\varepsilon^6_5)),\varepsilon^6_6))$$

applied to this element of Lex^6: $\langle\langle$every, D$\rangle$, $\langle$-a, Agrf$\rangle$, $\langle$very, M$\rangle$, $\langle$gentle, A$\rangle$, $\langle$-a, Agrf$\rangle$, $\langle$woman, Nf$\rangle\rangle$. Putting the arguments in alphabetical order and eliminating redundancies, we can get the same value with this polynomial

$$m(g(\varepsilon^5_2,\varepsilon^5_1), m(m(\varepsilon^5_4, g(\varepsilon^5_3,\varepsilon^5_1)),\varepsilon^5_5))$$

applied to this element of Lex^5: $\langle\langle$-a, Agrf$\rangle$, $\langle$every, D$\rangle$, $\langle$gentle, A$\rangle$, $\langle$very, M$\rangle$, $\langle$woman, Nf$\rangle\rangle$.

Each polynomial is represented by a term. Let's say that elements of $\mathcal{F}$ and the projection functions by themselves have *term depth* 0. And for any polynomial term $f(p_1,\ldots,p_n)$, let it's *term depth* be 1 more than the maximum depth of terms $p_1,\ldots,p_n$. Let the *depth* of any polynomial p be the minimum term depth of polynomials defining the function p.

Given any grammar $(Lex, \mathcal{F})$, it is clear that $(Lex, poly(G))$ has the same automorphisms. The addition of the polynomials does not change structure,

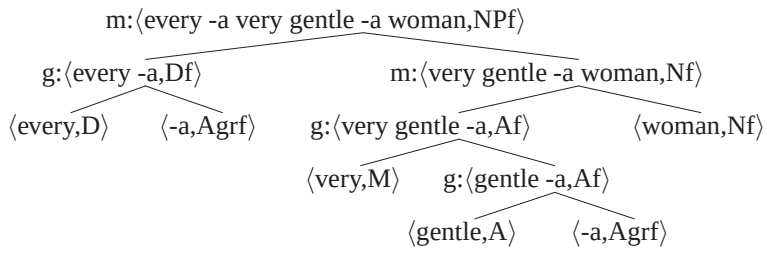


Figure 2.1: Span derivation of an NPf

even though it gives every expression a 1-step derivation (Keenan and Stabler, 2003, p.58).

We see in this setting that the mere fact that two expressions have structurally different derivations does not show that they have different structures. One and the same expression can have infinitely many derivations. Even two expressions with isomorphic derivations with the same categories, differing only in their strings, can differ in structure if the generating functions can be sensitive to the strings.

2.2 Balanced grammars

In Span, the categories serve to pick out the domains of the structure building functions. Let's say that $G = (Lex, \mathcal{F})$ is *category closed* iff for any $s_1,\ldots,s_n,t_1,\ldots,t_n \in [Lex]$ and for $0 < i \leq n$, if s_i and t_i have the same categories, then for all $f \in \mathcal{F}$

$$\langle s_1,\ldots,s_n\rangle \in \mathrm{dom}(f) \text{ iff } \langle t_1,\ldots,t_n\rangle \in \mathrm{dom}(f).$$

Let's say that G is *category functional* iff for all $f \in \mathcal{F}$ and for any $s_1,\ldots,s_n,t_1,\ldots,t_n \in \mathrm{dom}(f)$, if, for $0 < i \leq n$, s_i and t_i have the same categories, then $f(s_1,\ldots,s_n)$ and $f(t_1,\ldots,t_n)$ have the same category.

Span is category closed and category functional. We will restrict attention to grammars with these properties in the sequel except when explicitly indicated. Imposing these conditions requires that syntactic categories be explicit in a sense, reflecting all properties relevant to the application of structure building functions.

It will also be useful to require that our grammars make their operations appropriately explicit in the signature, in a sense we now define. For any partial functions $\mathcal{F}$, let $explode(\mathcal{F}) = \{\{\langle a,b\rangle\} \mid f_i(a) = b \text{ for some } f_i \in \mathcal{F}\}$. And for any $G = (A,\mathcal{F})$, let $explode(G) = (A, explode(F))$. (The order of the functions in $explode(\mathcal{F})$ will not matter for present purposes.) Then for any grammar G, the grammar $explode(G)$ defines the same language, but will often have fewer automorphisms. In $explode(G)$, every expression that is in the domain or range of any function is structural. So the only non-trivial automorphisms, if any, are those that exchange lexical items not in the domain or range of any function.

The grammar explode(Span) has infinitely many generating functions, and is "unbalanced" in the sense that there are regularities in m and g that we see in the automorphisms of Span, but not in automorphisms of explode(Span).

Let's say functions f,g are *compatible* iff they agree on any elements common to both of their domains; so functions with disjoint domains are always compatible. Since the functions g and m of Span are compatible, consider the grammar collapse(Span)=(Lex,$\langle g \cup m\rangle$) with a single generating function. This grammar is "unbalanced" too, in the sense that while collapse(Span) and Span have the same automorphisms, taking the union of g and m does not reveal anything new.

Let's say that a grammar $G = (A,\mathcal{F})$ is *balanced* iff both

- there are no two distinct, compatible, non-empty functions $f_i, f_j \in \mathcal{F}$ such that removing f_i, f_j and adding $f_i \cup f_j$ strictly increases the set of automorphisms, and
- there are no two distinct, compatible, non-empty functions g,g' such that $g \cup g' = f_i$ for some $f_i \in \mathcal{F}$, where the result of replacing f_i by g and g' yields a grammar with the same automorphisms as G has.

Balance matters. As noted above, it affects the automorphisms. And it affects grammar type, the signature. In the present context, balance matters because the elements of $\mathcal{F}$ determine the available structural polynomials that are useful in comparing grammars, as explained below.

In addition to the noun phrase grammar Span above, Keenan and Stabler (2003) define a "little English" Eng (p15), a "little Korean" case marking language Kor (p47), a "free word order" case marking language FWK (p54), a little verb-marking language Toba (p67), and a classical categorial grammar CG1 (p105).

Theorem 2.2.1. *None of the grammars Span, Eng, Kor, FWK, Toba, or CG1 are balanced.*

Proof: It suffices to show that in each grammar, there is a function $f \in \mathcal{F}$ that can be replaced by distinct nonempty $g1,g2$ such that $f = g1 \cup g2$, without changing the automorphisms. For Span, let $g1 = g \cap (\mathrm{PH}(D) \times \mathrm{E} \times \mathrm{E})$ and $g2 = g \cap (\mathrm{PH}(A) \times \mathrm{E} \times \mathrm{E})$. Then $g1,g2$ are compatible, $g = g1 \cup g2$, and since $\mathrm{PH}(D)$ and $\mathrm{PH}(A)$ are

already structural in Span, the automorphisms of Span are unchanged by the replacement of g by $g1$ and $g2$. The other grammars mentioned above have similarly over-unified structure-building functions. □

Define the grammar bal(Span) with $\Sigma, \mathrm{Cat}, Lex$ unchanged from Span, but $\mathcal{F} = \langle g1, g2, m1, m2, m3\rangle$, where $g1$ gender-marks determiners D as follows,

$$\langle\langle \mathrm{s}, \mathrm{D}\rangle, \langle \mathrm{t}, \mathrm{Agrm}\rangle\rangle \mapsto \langle \mathrm{st}, \mathrm{Dm}\rangle$$
$$\langle\langle \mathrm{s}, \mathrm{D}\rangle, \langle \mathrm{t}, \mathrm{Agrf}\rangle\rangle \mapsto \langle \mathrm{st}, \mathrm{Df}\rangle;$$

$g2$ gender-marks adjectives A:

$$\langle\langle \mathrm{s}, \mathrm{A}\rangle, \langle \mathrm{t}, \mathrm{Agrm}\rangle\rangle \mapsto \langle \mathrm{st}, \mathrm{Am}\rangle$$
$$\langle\langle \mathrm{s}, \mathrm{A}\rangle, \langle \mathrm{t}, \mathrm{Agrf}\rangle\rangle \mapsto \langle \mathrm{st}, \mathrm{Af}\rangle;$$

$m1$ produces complex Am,Af:

$$\langle\langle \mathrm{s}, \mathrm{M}\rangle, \langle \mathrm{t}, \mathrm{Am}\rangle\rangle \mapsto \langle \mathrm{st}, \mathrm{Am}\rangle$$
$$\langle\langle \mathrm{s}, \mathrm{M}\rangle, \langle \mathrm{t}, \mathrm{Af}\rangle\rangle \mapsto \langle \mathrm{st}, \mathrm{Af}\rangle;$$

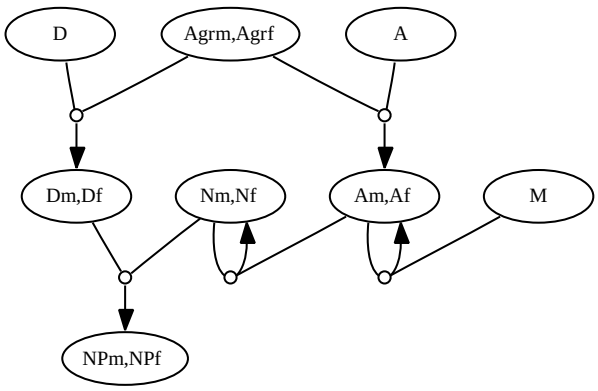
$m2$ produces complex Nm,Nf:

$$\langle\langle \mathrm{s}, \mathrm{Am}\rangle, \langle \mathrm{t}, \mathrm{Nm}\rangle\rangle \mapsto \langle \mathrm{st}, \mathrm{Nm}\rangle$$
$$\langle\langle \mathrm{s}, \mathrm{Af}\rangle, \langle \mathrm{t}, \mathrm{Nf}\rangle\rangle \mapsto \langle \mathrm{st}, \mathrm{Nf}\rangle;$$

and $m3$ produces noun phrases:

$$\langle\langle \mathrm{s}, \mathrm{Dm}\rangle, \langle \mathrm{t}, \mathrm{Nm}\rangle\rangle \mapsto \langle \mathrm{st}, \mathrm{NPm}\rangle$$
$$\langle\langle \mathrm{s}, \mathrm{Df}\rangle, \langle \mathrm{t}, \mathrm{Nf}\rangle\rangle \mapsto \langle \mathrm{st}, \mathrm{NPf}\rangle.$$

The grammar bal(Span) is category closed, category functional, and balanced. We can diagram the relations between the functions (circles), their domains and ranges (ovals):



Note that this kind of graph does not provide full information about the domain of the functions. It does not show for example how the determiner and noun genders must match, so that gender in effect splits the nominal system into two similar systems. These details matter.

It is easy to see that combinatory categorial grammars (Steedman, 1989) are unbalanced in the way Span is. Standard tree adjoining grammars (TAGs) (Joshi and Schabes, 1997) with $\mathcal{F} = \langle$substitution, adjunction$\rangle$ are unbalanced too, as are minimalist grammars (Chomsky, 1995) with $\mathcal{F} = \langle$merge, move$\rangle$ and variants. These grammars can usually be converted into "balanced" forms by replacing each generating function f by the set of functions obtained by restricting f to each of the structural subsets of its domain. This makes structural distinctions more explicit in $\mathcal{F}$, and thereby increases the options for building the polynomials which we will exploit in the next section.

2.3 Comparing grammars

We have seen that grammars of the sort defined in §1 are partial algebras that define groups of automorphisms. We introduce some standard notions for comparing different languages. Following (Grätzer, 1968, ch.2), we define three different notions of homomorphism for our partial algebras. Function $h : A \to B$ is a *homomorphism* from $(A, \langle f_1, \ldots\rangle)$ to $(B, \langle g_1, \ldots\rangle)$ iff for $0 < i$, both

1. whenever $\langle s_1, \ldots, s_n\rangle \in \mathrm{dom}(f_i)$, $\langle h(s_1), \ldots, h(s_n)\rangle \in \mathrm{dom}(g_i)$, and
2. $h(f_i(s_1, \ldots, s_n)) = g_i(h(s_1), \ldots, h(s_n))$.

A homomorphism is *full* iff for $0 < i$ and for all $s_1, \ldots, s_n, s \in A$,

a. $\langle h(s_1), \ldots, h(s_n)\rangle \in \mathrm{dom}(g_i)$ and

b. $g_i(h(s_1), \ldots, h(s_n)) = h(s)$

imply that there are $t_1, \ldots, t_n, t \in A$ such that

c. $h(s_1) = h(t_1), \ldots, h(s_n) = h(t_n), h(s) = h(t)$, and

d. $\langle t_1, \ldots, t_n\rangle \in \mathrm{dom}(f_i)$, $f_i(t_1, \ldots, t_n) = t$.

And a homomorphism is *strong* iff for $0 < i$,

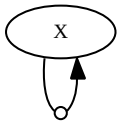
$$\langle s_1, \ldots, s_n\rangle \in \mathrm{dom}(f_i) \text{ iff } \langle h(s_1), \ldots, h(s_n)\rangle \in \mathrm{dom}(g_i).$$

To compare grammars with different types, different signatures, we find polynomials that can be homomorphically related. Let's say that there is a (full, strong) *polynomial homomorphism* of $(A, \mathcal{F})$ into $(B, \mathcal{G})$ iff there are polynomials $P_1, P_2, \ldots,$ over $(A, \mathcal{F})$ such that there is a (full, strong) homomorphism from $(A, \langle P_1, P_2, \ldots\rangle)$ to $(B, \mathcal{G})$.

Let's define a minimal recursive language $\mathcal{R} = (Lex, \mathcal{F})$ as follows.

$$\Sigma = \{\mathrm{a,b,w}\}, \text{ and}$$
$$\mathrm{Cat} = \{\mathrm{X,W}\},$$
$$Lex = \{\ \langle \mathrm{a,X}\rangle,\ \langle \mathrm{b,X}\rangle,\ \langle \mathrm{w,W}\rangle\ \}$$

and $\mathcal{F} = \langle m\rangle$, where m is the identity function on $\mathrm{PH}(X)$. Keenan and Stabler (2003, p165) propose that grammatical constants often play a special role in the grammar – these include many 'grammatical morphemes' etc. The grammar $\mathcal{R}$ includes two elements to indicate that the recursion involves a category that includes non-constant elements. And we include $\langle \mathrm{w,W}\rangle$ to indicate there can be elements that do not participate in the recursion. $\mathcal{R}$ has the following diagram:



And let's define a minimal "one step" language $O = (Lex, \mathcal{F})$ as follows.

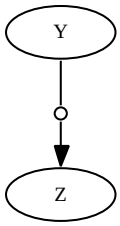
$$\Sigma=\{\text{c,d,e,f}\}, \text{ and}$$
$$\text{Cat}=\{\text{Y,Z}\},$$

$$Lex = \{\ \langle \text{c}, \text{Y}\rangle,\ \langle \text{d}, \text{Y}\rangle\ \},$$

and $\mathcal{F} = \langle n \rangle$, where, for any $s,t \in \Sigma^*$, n maps expressions as follows:

$$\langle \text{c}, \text{Y}\rangle \mapsto \langle \text{e}, \text{Z}\rangle$$
$$\langle \text{d}, \text{Y}\rangle \mapsto \langle \text{f}, \text{Z}\rangle.$$

We can diagram O:



Theorem 2.3.1. *There is a full polynomial homomorphism from O to $\mathcal{R}$, but no strong one.*

Proof: Consider the function $h : [Lex_O] \to [Lex_{\mathcal{R}}]$ given by the following mappings:

$$\langle c,Y\rangle \mapsto \langle a,X\rangle \qquad \langle d,Y\rangle \mapsto \langle b,X\rangle$$
$$\langle e,Z\rangle \mapsto \langle a,X\rangle \qquad \langle f,Z\rangle \mapsto \langle b,X\rangle.$$

This is a homomorphism from $([Lex_O], \langle n \rangle)$ to $([Lex_{\mathcal{R}}], \langle m \rangle)$ since whenever $s \in \text{dom}(n)$, $h(s) \in \text{dom}(m)$, and $h(n(s)) = m(h(s))$. This homomorphism is *full*, since whenever $h(s) \in \text{dom}(m)$ and $m(h(s)) = s'$, there are $t,t' \in [Lex_O]$ such that $h(s) = h(t), h(s') = h(t')$, and $t \in \text{dom}(n)$, $n(t) = t'$. For example, $h(e,Z) = \langle a,X\rangle \in \text{dom}(m)$ and although $\langle e,Z\rangle \notin \text{dom}(n)$ there are elements $\langle c,Y\rangle, \langle e,Z\rangle \in [Lex]_O$ such that $h(e,Z) = h(c,Y), h(e,Z) = h(e,Z)$, such that $\langle c,Y\rangle \in \text{dom}(n)$ with $n(c,Y) = \langle e,Z\rangle$. However, homomorphism h is not *strong* since it is not the case that

$$s \in \text{dom}(n) \text{ iff } h(s) \in \text{dom}(m).$$

In particular, $h(e,Z) \in \text{dom}(m)$ but $\langle e,Z\rangle \notin \text{dom}(n)$. Not only is h not a strong polynomial homomorphism from O to $\mathcal{R}$, but it is easy to see that no such thing exists, since in $\mathcal{R}$, everything is in the range of m is also in its domain, while in O, n maps elements from its domain to things outside that domain. □

2.4 Predication and modification

Human languages differ in their most basic constituent order, and in their argument and agreement marking properties, as for example Keenan and Stabler (2003) illustrate with tiny fragments of Spanish, English, Korean, free word order languages, and Toba Batak. In these languages and, we claim, human languages, certain semantically identified relations are structural. But one puzzle left unanswered in that work was: How can we describe the significant syntactic similarities among languages as different as these, in a clear and illuminating way? We might like to say, for example, that human languages all have transitive and intransitive predication; all languages have modification of both arguments and predicates, and so on. Now we have the tools to maks such claims precise.

Let's define a minimal predicative language $\mathcal{P} = (Lex, \mathcal{F})$ as follows:

$$\Sigma=\{\text{a,b,p,q,r,s,w}\},$$
$$\text{Cat}=\{\text{P0,P1,P2,W}\},$$

$$\begin{aligned} Lex = \{\ & \langle \text{a}, \text{D}\rangle,\ \ \langle \text{b}, \text{D}\rangle, \\ & \langle \text{p}, \text{P1}\rangle,\ \langle \text{q}, \text{P1}\rangle, \\ & \langle \text{r}, \text{P2}\rangle,\ \langle \text{s}, \text{P2}\rangle,\ \langle \text{w}, \text{W}\rangle\ \}, \end{aligned}$$

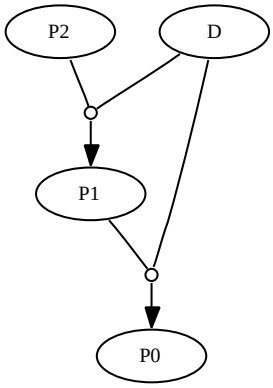
and $\mathcal{F} = \langle m1, m2\rangle$, where $m1$ saturates unary 'predicates' as follows, for any $s,t \in \Sigma^*$,

$$\langle\langle \text{s}, \text{D}\rangle, \langle \text{t}, \text{P1}\rangle\rangle \mapsto \langle \text{st}, \text{P0}\rangle,$$

and $m2$ maps binary 'predicates' to unary ones,

$$\langle\langle \text{s}, \text{D}\rangle, \langle \text{t}, \text{P2}\rangle\rangle \mapsto \langle \text{st}, \text{P1}\rangle.$$

We can diagram $\mathcal{P}$:



And let's define a minimal modifier language $\mathcal{M} = (Lex, \mathcal{F})$ as follows.

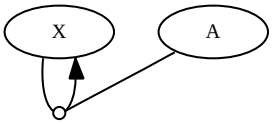
$$\Sigma=\{\text{a,b,p,q,w}\}, \text{ and}$$
$$\text{Cat}=\{\text{A,X,W}\},$$

$$\begin{aligned} Lex = \{\ & \langle \text{a}, \text{A}\rangle,\ \langle \text{b}, \text{A}\rangle, \\ & \langle \text{p}, \text{X}\rangle,\ \langle \text{q}, \text{X}\rangle,\ \langle \text{w}, \text{W}\rangle\ \}, \end{aligned}$$

and $\mathcal{F} = \langle m \rangle$, where m 'modifies' some elements X as follows, for any $s,t \in \Sigma^*$,

$$\langle\langle \mathrm{s}, \mathrm{A}\rangle, \langle \mathrm{t}, \mathrm{X}\rangle\rangle \mapsto \langle \mathrm{st}, \mathrm{X}\rangle.$$

We can diagram $\mathcal{M}$:



As noted above, Keenan and Stabler (2003) define the languages Span, Eng, Kor, FWK, Toba, and CG1. Theorem 1 points out that none of these is balanced, but a balanced grammar bal(Span) is provided above, and balanced grammars of the other languages are also easy to formulate.

Theorem 2.4.1. *There is a strong polynomial homomorphism from bal(Span) to $\mathcal{M}$.*

Proof: Define $h : [Lex_{bal(Span)}] \rightarrow [Lex_{\mathcal{M}}]$ as follows:

$$h(s) = \begin{cases} \langle \mathrm{a}, \mathrm{A}\rangle & \text{if } s = \langle \mathrm{x}, \mathrm{Am}\rangle \text{ for any } x \in \Sigma^* \\ \langle \mathrm{b}, \mathrm{A}\rangle & \text{if } s = \langle \mathrm{x}, \mathrm{Af}\rangle \text{ for any } x \in \Sigma^* \\ \langle \mathrm{p}, \mathrm{X}\rangle & \text{if } s = \langle \mathrm{x}, \mathrm{Nm}\rangle \text{ for any } x \in \Sigma^* \\ \langle \mathrm{q}, \mathrm{X}\rangle & \text{if } s = \langle \mathrm{x}, \mathrm{Nf}\rangle \text{ for any } x \in \Sigma^* \\ \langle \mathrm{w}, \mathrm{W}\rangle & \text{otherwise.} \end{cases}$$

Letting $m2$ be the polynomial over Span, this is a homomorphism from $(Lex_{bal(Span)}, \langle m2\rangle)$ to $\mathcal{M}$ since whenever $\langle s_1, s_2\rangle \in \mathrm{dom}(m1)$, $\langle h(s_1), h(s_2)\rangle \in \mathrm{dom}(m)$, and $h(m1(s_1,s_2)) = m(h(s_1), h(s_2))$. It is strong since $\langle s_1, s_2\rangle \in \mathrm{dom}(m1)$ iff $\langle h(s_1), h(s_2)\rangle \in \mathrm{dom}(m)$. □

It's clear that there are other strong polynomial homomorphisms from bal(Span) to $\mathcal{M}$, finding recursion in either the N modifiers or in the A modifiers. It is similarly easy to show that there are strong polynomial homomorphisms from Eng, Kor, FWK, Toba, and CG1 of Keenan and Stabler (2003) to $\mathcal{P}$. We propose,

Hypothesis 2.4.2. *For every category closed, category functional, balanced grammar for a human language G, there are strong polynomial homomorphisms from G to $\mathcal{P}$, to $\mathcal{M}$, and to $\mathcal{R}$.*

In other grammars of human languages, we find the encoding of predicative, modifier, and other recursive relations sometimes elaborated by marking arguments or predicates, or making other small adjustments, but we expect it will always be easy to find structures like these in any human language.

Hypothesis 2.4.3. *There are low complexity polyomials satisfying Hypothesis 2.4.2, polynomials with depths in the range of 2 or 3.*

Keenan and Stabler (2003) observe that the automorphism *mf* of Span has a different status than the automorphisms that permute elements inside of each category. One difference noted there is that *mf* is disrupted by the addition of a single new element of category Nf; with this change, the categories Nm and Nf become structural. But now we can notice in addition, that for *mf* (as for any other element of *Aut*), given any polynomial homomorphism h from *Span* to A, $h(mf)$ is category preserving. This is an immediate consequence of the fact that $\mathcal{M}$ itself does not have enough structure to distinguish masculine and feminine systems of modification, and provides a precise sense in which we can see that the agreement marking that introduces the category changing automorphisms into the modification systems, does not participate in the modification system; it is collapsed by every strong polynomial homomorphism to $\mathcal{M}$ into a category preserving automorphism. Extending Span to include predication, we find the agreement distinctions similarly collapsed in strong polynomial homomorphisms from that system to $\mathcal{P}$.

2.5 The syntactic status of sentences

Do sentences, semantically identified as the bearers of propositional content, have any special syntactic status across languages? Given a grammar of an arbitrary natural language with the categories and lexical items renamed, and without any semantics, could you tell which categories were clauses? One positive idea, and another universal claim about human languages, can be formulated as follows.

For any grammar $G = (Lex, \mathcal{F})$, define $Lex_0 = Lex$, and $Lex_{n+1} = Lex_n \cup \{ f(\vec{e}) |\ \vec{e} \in Lex_n^* \cap \mathrm{dom}(f), f \in \mathcal{F}\}$. Clearly the language $[Lex] = \bigcup_{i \geq 0} Lex_i$. Standard notions of immediate constituency ICON and related notions can be defined as follows. Let $e\,\mathrm{ICON}\,e'$ iff there is $\langle d_1, \ldots, d_n\rangle \in [Lex]^*$ such that $e = d_i$ for some $0 < i \leq n$ and there is some $f \in \mathcal{F}$ such that $e' = f(d_1, \ldots, d_n)$. Then let PCON be the transitive closure of ICON, and let CON be the reflexive, transitive closure of ICON.

Let's distinguish those expressions whose derivations do not include any recursion, in any category, $e \in [Lex]$ is *non-recursive*, $\mathrm{NR}(e)$ iff there are no $d, d' \in [Lex]$ such that $\mathrm{Cat}(d) = \mathrm{Cat}(d')$, $d\,\mathrm{PCON}\,d'$, and $d'\,\mathrm{CON}\,e$. Now we can define the height of the most complex but non-recursive elements of a category. The *non-recursive height* of category C,

$$nrh(C) = \max_i \exists e \in Lex_i,\ \mathrm{NR}(e),\ \mathrm{Cat}(e) = C.$$

Then we can know say what it is to be the most complex category without recursion, as follows: C is non-recursively maximal iff there is no $C' \in Cat$ such that $nrh(C') > nrh(C)$.

It is easy to show that the set of expressions that have a non-recursively maximal category is a structural set, in the sense defined above. In the example grammars Eng, Kor, FWK, Toba, CG1 of Keenan and Stabler (2003), mentioned above, there is a unique non-recursively max-

imal category, the 'sentence' category (named P0 or S in those grammars).

Hypothesis 2.5.1. *In every category closed, category functional, balanced grammar for a human language, there are non-recursively maximal categories that hold of the expressions semantically identified as bearers of propositional content ('clauses').*

Note that in a grammar that marks extractions of X with /X features in the category system, if it allows extractions of arbitrarily many Xs, there can easily fail to be any non-recursively maximal category. If any human language allows unboundedly many elements to be extracted from a single constituent – contra the kinds of limits in TAGs (Joshi and Schabes, 1997) and minimalist grammars (Stabler, 1997), etc. – then this last hypothesis will need to be reformulated.

2.6 Conclusions

We have defined an approach to language that is suitably abstract for stating the purely syntactic component of semantically loaded universals of language like these:

- All human languages exhibit transitive and intransitive predication.
- All human languages exhibit modification of at least one category.
- All human languages have recursion.

To capture the purely syntactic part of these, we propose,

> **Hypothesis 2.4.2** For any category closed, category functional, balanced grammar G for a human language, there are strong polynomial homomorphisms from G to $\mathcal{P}$, to $\mathcal{M}$, and to $\mathcal{R}$.
>
> **Hypothesis 2.4.3** There are low complexity polyomials satisfying Hypothesis 2.4.2, polynomials with depths in the range of 2 or 3.

Finally, we propose that clausal categories are maximal in a certain sense:

> **Hypothesis 2.5.1** In every category closed, category functional, balanced grammar for a human language, there are non-recursively maximal categories that hold of the expressions semantically identified as bearers of propositional content ('clauses').

It should be possible to use this kind of approach to articulate precise versions of a range of familiar universal claims about syntax. As these claims become more precise, it may be possible to establish whether they are really correct. Notice that these claims are not tied to an particular grammar formalism. For example, we already observed that a particular grammar $G = (Lex, \mathcal{F})$ satisfies these hypotheses iff $G = (Lex, poly(G))$ does. It does not matter which grammar we select from any of the infinitely many that define the same automorphisms.

Bibliography

Büchi, J. Richard (1960). Weak second-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, **6**:66–92.

Chomsky, Noam (1995). *The Minimalist Program*. MIT Press, Cambridge, Massachusetts.

Doner, John (1970). Tree acceptors and their applications. *Journal of Computer and System Sciences*, **4**:406–451.

Grätzer, George (1968). *Universal Algebra*. van Nostrand, NY.

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

Keenan, Edward L. and Edward P. Stabler (2003). *Bare Grammar: Lectures on Linguistic Invariants*. CSLI Publications, Stanford, California.

Klein, Felix (1893). A comparative review of recent researches in geometry. *Bulletin of the New York Mathematical Society*, **2**:215–249. Translation by M.W. Haskell of the original October 1872 publication, with a prefatory note by the author.

Plotkin, Boris I. (1972). *Groups of automorphisms of algebraic systems*. Wolters-Noordhoff, Groningen. English translation by K.A. Hirsh of the Russian *Gruppi avtomorfismov algebraicheskikh sistem*, Moscow, 1966.

Rogers, James (1996). A model-theoretic framework for theories of syntax. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*.

Stabler, Edward P. (1997). Derivational minimalism. In Christian Retoré, ed., *Logical Aspects of Computational Linguistics*, pp. 68–95. Springer-Verlag (Lecture Notes in Computer Science 1328), NY.

Steedman, Mark J. (1989). Grammar, interpretation, and processing from the lexicon. In William Marslen-Wilson, ed., *Lexical Representation and Process*, pp. 463–504. MIT Press, Cambridge, Massachusetts.

Thatcher, J.W. and J.B. Wright (1968). Generalized finite automata theory with an application to a decision problem of second order logic. *Mathematical Systems Theory*, **2**:57–81.

Thomas, Wolfgang (1997). Languages, automata and logic. In G. Rozenberg and A. Salomaa, eds., *Handbook of Formal Languages, Volume 3: Beyond Words*, pp. 389–455. Springer, NY.

Chapter 3

Applications of Modal Logic in Model Theoretic Syntax[1]

Hans-Jörg Tiede
Department of Mathematics and Computer Science
Illinois Wesleyan University
htiede@iwu.edu

3.1 Logics of Trees

Since extending the logic of strings to capture more complex string languages than the regular languages often leads to undecidability (see e.g. Lautemann et al. (1995)), one approach to extending the coverage of logic is to describe more complex structures: move from strings to trees. Thus, the Kripke structures we will be considering are trees, and the logics will contain more complicated modalities to describe trees. One immediate advantage of this approach for linguistic purposes is that these logics will automatically be connected to strong generative capacity, since they describe sets of trees. One disadvantage is that the *recognition* or *parsing* problem, which in the string case just amounts to model checking, now involves satisfiability checking (see below).

The extension of the descriptive approach to trees was originally also motivated by decidability questions Thatcher and Wright (1968). Even though the connections to CFLs were pointed out by Thatcher (1967), this line of research did not find applications in linguistics until the development of constraint based grammar formalisms which replaced the derivational approach to natural language syntax. The work of Rogers (1998), Kracht (2003), and others provided formal models for these constraint based grammar formalisms and established formal language theoretic results for them at the same time.

As mentioned above our Kripke structures will now be trees. We will use the concept of tree domains Gorn (1967) to define such Kripke structures. A (finite, binary) *tree domain*, T, is a finite subset of $\{0,1\}^*$, such that for all $u,v \in \{0,1\}^*$

1. if $uv \in T$, then $u \in T$, and
2. if $u1 \in T$, then $u0 \in T$.

A string in T describes a path from the root to a node, where 0 means "go left" and 1 means "go right". We identify nodes with the path leading to them. Thus, ε is the root. The first condition above says that if there is a path to a node, then there is a path to any node above it (this is called prefix closure). The second condition says that if a node has a right daughter, then it has a left daughter (called left sibling closure).

[1]This paper is an excerpt of Moss and Tiede (2006)

The main relations between nodes in a tree that are of interest in linguistics are domination and linear precedence. We say that that a node $u \in T$ *dominates* a node $v \in T$ if for some $w \in \{0,1\}^*$, $v = uw$. A special case of domination is the parent-of relation, defined by: u is the parent of v if $v = u0$ or $v = u1$. We say that u *linearly precedes* v if for some $x,y,z \in \{0,1\}^*$, $u = x0y$ and $v = x1z$. Following Rogers (1998), we will denote the domination relation by $\lhd^*$, the parent-of relation by $\lhd$, and linear precedence by $\prec$. Thus, our Kripke frames will be variations of the form $(T, \lhd, \lhd^*, \prec)$, where T is a tree domain.

3.1.1 Regular tree languages

In order to generalize from strings to labeled trees, we will now consider *ranked alphabets* in which each symbol has an arity or rank. For surveys of tree languages see Gécseg and Steinby (1997) or Thatcher (1973). Let Σ be a ranked alphabet. We will denote the set of n-ary symbols in Σ by Σ_n. The set of terms over Σ is denoted by T_Σ. A subset of T_Σ is called a *tree language*.

In a number of settings, trees are considered to be labeled with boolean features, rather than with ranked symbols. We note that these two approaches commensurable using the following representation. Given a finite set of boolean features $F = \{f_1, \ldots, f_n\}$, the *binary ranked alphabet based on* F, Σ^F, is defined as

$$\Sigma^F = \{f_1, \neg f_1\} \times \cdots \times \{f_n, \neg f_n\} \times \{0,2\}$$

where each $f_i, \neg f_i$ represents whether or not a feature holds at a given node and 0 or 1 represent the arity of the symbol. Thus, $(f_1, \neg f_2, 0)$ would be a leaf symbol, and $(f_1, \neg f_2, 2)$ would be an internal node symbol. The previous definition can be easily generalized to trees of any arity.

The *yield* of a tree, t, is the string over Σ_0 which is obtained by concatenating the symbols at the leaves of t from left to right, or more formally:

$$yield(c) = c, \text{ for } c \in \Sigma_0$$
$$yield(f(t_1, \ldots, t_n)) = yield(t_1) \ldots yield(t_n), \text{ for } f \in \Sigma_n$$

A (bottom-up, non-deterministic) *finite tree automaton* (FTA) M is a structure of the form (Σ, Q, F, Δ) where Σ

is a ranked alphabet, Q is a finite set of states, $F \subseteq Q$ is the set of final states, and Δ is a finite set of transition rules of the form $f(q_1,\ldots,q_n) \to q$ with $f \in \Sigma_n$ and $q, q_1, \ldots, q_n \in Q$. An FTA is *deterministic* if there are no two transition rules with the same left-hand-side. It can be shown that the bottom-up variety of finite tree automata can be determinized, while the top-down variety cannot.

A *context* s is a term over $\Sigma \cup \{x\}$ containing the zero-ary term x exactly once. We write $s[x \mapsto t]$ for the term that results from substituting x in s with t. Given a finite tree automaton $M = (\Sigma, Q, F, \Delta)$ the derivation relation $\Rightarrow_M \subseteq T_{Q\cup\Sigma} \times T_{Q\cup\Sigma}$ is defined by $t \Rightarrow_M t'$ if for some context $s \in T_{\Sigma\cup Q\cup\{x\}}$ there is a rule $f(q_1,\ldots,q_n) \to q$ in Δ, and

$$t = s[x \mapsto f(q_1,\ldots,q_n)]$$
$$t' = s[x \mapsto q]$$

We use $\Rightarrow_M^*$ to denote the reflexive, transitive closure of $\Rightarrow_M$. A finite automaton M *accepts* a term $t \in T_\Sigma$ if $t \Rightarrow_M^* q$ for some $q \in F$. The *tree language accepted* by a finite tree automaton M, $L(M)$, is

$$L(M) = \{t \in T_\Sigma \mid t \Rightarrow_M^* q, \text{ for some } q \in F\}.$$

A tree language, L, is *regular* if $L = L(M)$ for some FTA M.

The following example is concerned with the Circuit Value Problem (CVP), in which the trees labeled with boolean functions are evaluated. It is interesting to note that a number of separation results of logically defined tree languages use trees labeled with boolean functions Potthoff (1994a).

Example 3.1.1. *Let $\Sigma = \{\wedge, \vee, 0, 1\}$. The tree language $CVP \subseteq T_\Sigma$ such that each tree in CVP evaluates to true can be accepted by the following FTA, $M = (\Sigma, Q, F, \Delta)$, where*

$$Q = \{t, f\}$$
$$F = \{t\}$$

and

$$\Delta = \left\{ \begin{array}{ll} 0 \to f, & 1 \to t, \\ \wedge(t,t) \to t, & \wedge(t,f) \to f, \\ \wedge(f,t) \to f, & \wedge(f,f) \to f, \\ \vee(t,t) \to t, & \vee(t,f) \to t, \\ \vee(f,t) \to t, & \vee(f,f) \to f \end{array} \right\}$$

Given a finite sets of feature $F = \{f_1,\ldots,f_n\}$ and a feature $f_i \in F$, we define the *projection*, π, that eliminates f_i in the natural way:

$$\pi : \Sigma^F \to \Sigma^{F-\{f_i\}}$$

This definition can be extended to arbitrary subsets $G \subseteq F$, where

$$\pi : \Sigma^F \to \Sigma^{F-G}$$

Given a projection $\pi : \Sigma^F \to \Sigma^{F-G}$, we extend π to a tree homomorphism $\hat{\pi} : T_{\Sigma^F} \to T_{\Sigma^{F-G}}$ as follows:

$$\hat{\pi}(c) = \pi(c)$$
$$\hat{\pi}(f(t_1,\ldots,t_n)) = \pi(f)(\hat{\pi}(t_1),\ldots,\hat{\pi}(t_n))$$

with $c \in \Sigma_0$ and $f \in \Sigma_n, n > 0$. For a tree language L, we define $\hat{\pi}(L) = \{\hat{\pi}(t) \mid t \in L\}$.

We will consider the relationship between regular tree languages and the derivation trees of CFGs.

Proposition 3.1.2. *(Thatcher (1967)) If $L \subseteq T_\Sigma$ is a regular tree language, then*

$$\{yield(t) \mid t \in L\}$$

is a CFL.

While the yields of regular tree languages are CFLs, regular tree languages are more complex than the derivation trees of CFG. In order to compare the regular tree languages to the derivation trees of CFGs, we formalize the latter using the local tree languages.

The *fork* of a tree t, $fork(t)$, is defined by

$$fork(c) = \varnothing$$
$$fork(f(t_1,\cdots,t_n)) =$$
$$\{(f, root(t_1),\ldots,root(t_n))\} \cup \bigcup_{i=1}^{n} fork(t_i)$$

with $c \in \Sigma_0$, $f \in \Sigma_n, n > 0$, and *root* being the function that returns the symbol at the root of its argument. For a tree language L, we define

$$fork(L) = \bigcup_{t\in L} fork(t)$$

The intuition behind the definition of $fork$ is that an element of $fork(T_\Sigma)$ corresponds to a rewrite rule of a CFG. Note that $fork(T_\Sigma)$ is always finite, since Σ is finite.

A tree language $L \subseteq T_\Sigma$ is *local* if there are sets $R \subseteq \Sigma$ and $E \subseteq fork(T_\Sigma)$, such that, for all $t \in T_\Sigma, t \in L$ iff $root(t) \in R$ and $fork(t) \subseteq E$.

We quote without proof the following two theorems by Thatcher (1967).

Theorem 3.1.3. *Thatcher (1967) A tree language is a set of derivation trees of some CFG iff it is local.*

Theorem 3.1.4. *Thatcher (1967) Every local tree language is regular.*

While there are regular tree languages that are not local, the following theorem, also due to Thatcher (1967), demonstrates that we can obtain the regular tree languages from the local tree languages via projections. We will review the main points of the proof, because we will use some of its details later on.

Theorem 3.1.5. *Thatcher (1967) For every regular tree language L, there is a local tree language L' and a one-to-one projection π, such that $L = \hat{\pi}(L')$.*

Syntax	**Formulas** φ
	$p_i \mid \neg\varphi \mid \varphi \wedge \psi \mid [\pi]\varphi$
	Programs π
	$\rightarrow \mid \leftarrow \mid \uparrow \mid \downarrow \mid \pi^*$
Semantics	**Main Clauses**
	$[\![\rightarrow]\!]_T = \{(u0,u1) \mid u1 \in T\}$
	$[\![\leftarrow]\!]_T = \{(u1,u0) \mid u1 \in T\}$
	$[\![\downarrow]\!]_T = \{(u,ui) \mid i \in \{0,1\}, ui \in T\}$
	$[\![\uparrow]\!]_T = \{(ui,u) \mid i \in \{0,1\}, ui \in T\}$

Figure 3.1: Modal logic of trees: $\mathcal{L}_{core}$

Proof. Let L be a regular tree language. Assume that L is accepted by the deterministic FTA $M = (\Sigma, Q, F, \Delta)$. We define L' terms of R and E as follows: $R = \Sigma \times F$ and

$$\begin{aligned} E \quad = \quad & \{((f,q),(f_1,q_1),\ldots,(f_n,q_n)) \\ & \mid f(q_1,\ldots,q_n) \rightarrow q \in \Delta, f_1,\ldots,f_n \in \Sigma\} \end{aligned}$$

We then define $L' = \{t \in T_{\Sigma\times Q} \mid root(t) \in R, fork(t) \subseteq E\}$. Notice that the trees in L' encode runs of M. The tree homomorphisms $\hat{\pi}$ based on the projection $\pi : \Sigma \times Q \rightarrow \Sigma$ maps L' to L as can be easily verified.

It should be noted that, since M is deterministic, there exists exactly one accepting run for each tree in $L(M)$ and thus the homomorphism $\hat{\pi} : L' \rightarrow L$ is one-to-one. □

This rather technical result is of some importance in the context of linguistic application, for it implies that we can use frameworks of lower complexity to describe the same structures as a more complex framework *if we use more complex categories or features*. Since we can also add new categories as names for the more complex ones, we can use a less complex framework to describe the same structures as a more complex framework by adding more categories. Thus, parsimony would seem to imply that we should always use the simpler framework. However from the point of linguistics, the use of complex or additional features needs to be justified. To further elaborate on the previous point, we will have to keep in mind that all of the logics we will consider can define the local tree languages and all the languages they can define are regular. Thus undefinability will always mean undefinability over a fixed finite set of propositional variables, since we can always define a regular, undefinable tree language by using more features.

3.1.2 The basic modal logic of trees: $\mathcal{L}_{core}$

To the best of our knowledge, the first explicit use of modal logic to define tree languages can be found in Blackburn et al. (1993). Two variations of this logic were considered in Blackburn and Meyer-Viol (1994); Blackburn et al. (1996), of which we will consider the latter. The basic modal logic of trees, $\mathcal{L}_{core}$, is defined in Figure 3.1. Again, we say that a tree T satisfies a formula φ if $T, \varepsilon \models \varphi$. A language L is *definable* in this (or another language) if there is a sentence φ so that L is exactly the set of trees satisfying a sentence φ.

The following proposition establishes that $\mathcal{L}_{core}$ is expressive enough to define any binary branching, local tree language. The restriction to binary branching is only due to the fact that we defined our tree domains to be binary branching.

Proposition 3.1.6. *Let $L \subseteq T_\Sigma$ be a local tree language. There is a sentence φ_G in $\mathcal{L}_{core}$ that defines L.*

Proof. By Theorem 3.1.3, there is a CFG G such that L is equal to the derivation trees of G. Let $G = (\Sigma, \Gamma, P, S)$. Since we are only considering binary branching trees, every rule in P is of the form $A \rightarrow BC$ or $A \rightarrow a$ with $A, B, C \in \Gamma$ and $a \in \Sigma$. We can simply encode the rules directly in our logic:

$$A \rightarrow \bigvee_{A \rightarrow BC \in P} \langle\downarrow\rangle(B \wedge \langle\rightarrow\rangle C)$$

and

$$A \rightarrow \bigvee_{A \rightarrow a \in P} (\langle\downarrow\rangle a)$$

This ensures that the models of φ_G are parse trees of G. However, we further need to ensure only the parse trees of G model φ_G. So, we need to express that each node makes exactly one symbol true:

$$[\downarrow^*](\bigvee_{a \in (\Sigma \cup \Gamma)} a \wedge \bigwedge_{a \neq b} (\neg a \vee \neg b))$$

that the start symbol of the grammar is true at the root: S, that the terminal symbols are true at the leaves:

$$[\downarrow^*](\bigvee_{a \in \Sigma} a \rightarrow \neg\langle\downarrow\rangle\top)$$

and that the non-terminal symbols are true at the internal nodes

$$[\downarrow^*](\bigvee_{A \in \Gamma} A \rightarrow \langle\downarrow\rangle\top)$$

□

As is observed by Blackburn and Meyer-Viol, this translation of a CFG into logical formulas brings with it a change in perspective. Instead of a *procedural* or *derivational* perspective that considers CFG rules to be rewrite rules, we move to a *declarative* or *descriptive* perspective that considers CFG rules to be *constraints*. This change in perspective is the main motivation for the application of logic in syntax, because of a similar change in perspective that occurred in a number of grammar formalisms proposed by linguists in the 1980s, most notably Chomsky's "Government and Binding" (GB) (Chomsky, 1981) and Gazdar, Klein, Pullum, and Sag's "Generalized Phrase Structure Grammar" (GPSG) (Gazdar et al., 1985).

Syntax
$p_i \mid \neg\varphi \mid \varphi \wedge \psi \mid$
$\mathcal{U}_{\rightarrow}(\varphi,\psi) \mid \mathcal{U}_{\leftarrow}(\varphi,\psi) \mid \mathcal{U}_{\uparrow}(\varphi,\psi) \mid \mathcal{U}_{\downarrow}(\varphi,\psi)$
Semantics
$T,u \models \mathcal{U}_{\downarrow}(\varphi,\psi)$ iff there exists a v such that $u \lhd^* v$,
$T,v \models \varphi$, and for all w such that $u \lhd^* w \lhd^* v$,
$T,w \models \psi$

Figure 3.2: Temporal logic of trees: $\mathcal{X}_{until}$ (only one clause in the semantics)

Syntax	**Formulas** φ
	$p_i \mid \neg\varphi \mid \varphi \wedge \psi \mid [\pi]\varphi$
	Programs π
	$\rightarrow \mid \leftarrow \mid \uparrow \mid \downarrow \mid \pi_\varphi \mid \pi^*$
Semantics	**Main Clauses**
	$[\![\pi_\varphi]\!]_T = \{(u,v) \mid (u,v) \in [\![\pi]\!]_T, T,u \models \varphi\}$

Figure 3.3: Conditional path logic of trees: $\mathcal{L}_{cp}$

3.1.3 ID/LP Grammars

The rules of a CFG encode two kinds of information: the categories of a node and its children, and the order in which the categories of the children occur. Thus, a rule of the form $A \rightarrow BC$ tells us that a node labeled A can have two children, one labeled B, the other C, and that the node labeled B precedes the node labeled C. Linguists have observed that separating these two notions can lead to more compact grammars. Thus, ID/LP grammars have been proposed that consist of *unordered* rewrite (immediate dominance or ID) rules, $A \rightarrow B,C$, and *linear precedence* (LP) rules, $B < C$. Linear precedence rules only apply to sisters, which is why we used $<$ rather than $\prec$ which applies to arbitrary nodes.

ID/LP grammars can be very naturally expressed in $\mathcal{L}_{core}$; in fact ID/LP grammars are, in some sense, a very limited logic for trees. See Gazdar et al. (1985) or Shieber (1984) for applications and detailed examinations of ID/LP grammars.

3.1.4 Variations of $\mathcal{L}_{core}$

Two additional basic modal logics of trees have been considered by Blackburn and associates Blackburn et al. (1993); Blackburn and Meyer-Viol (1994). The first includes the connectives $\varphi \Rightarrow \psi$ and $\bullet(\varphi_1,\ldots\varphi_n)$. The latter is used in the context of trees with n children, so we will only consider the case where n is 2. Their semantics are given by $T,v \models \varphi \Rightarrow \psi$ iff for all u, $T,u \models \varphi \rightarrow \psi$, and $T,v \models \bullet(\varphi,\psi)$ iff $T,u0 \models \varphi$ and $T,u1 \models \psi$. Notice that the purpose of $\bullet$ is to combine immediate dominance and linear precedence into one connective.

Blackburn and Meyer-Viol (1994) define a modal logic of trees that differs from $\mathcal{L}_{core}$ in that it contains modalities for the left and right daughter: $\downarrow_1, \downarrow_2$.

3.1.5 Temporal Logic of Trees

We now move on to an extension of $\mathcal{L}_{core}$, temporal logic. The syntax and semantics of propositional tense logic on trees, $\mathcal{X}_{until}$, is defined in Figure 3.2. The main application of $\mathcal{X}_{until}$ was given by Palm (1999), though with a different formulation which we will consider below. We follow here the formulation of Marx (2004), because it lends itself to a more direct proof of equivalence with FO.

Theorem 3.1.7. *Marx (2004) The following are equivalent for a tree language $L \subseteq T_\Sigma$:*

1. *L is FO definable.*
2. *L is definable in $\mathcal{X}_{until}$.*

While the notion of regular expressions can be generalized to trees, the correspondence between star-free expressions and FO (or $\mathcal{X}_{until}$) definability breaks down at this level. In fact, Thomas and Potthoff (1993) showed that every regular language that does not contain unary branching symbols is star-free. The question whether FO definability of regular tree language is decidable is still open.

3.1.6 Variations of $\mathcal{X}_{until}$

As was mentioned above, Palm's (1999) application of $\mathcal{X}_{until}$ was carried out using a different formulation which he called propositional tense logic and which Afanasiev et al. (2005) called conditional path logic, $\mathcal{L}_{cp}$. The syntax and semantics of $\mathcal{L}_{cp}$ are defined in Figure 3.3.

3.1.7 X-bar theory

As was mentioned above, which non-terminals are used in a natural language grammar matters to linguists. The point again is that the label assigned to a node in a tree signifies the grammatical category of the constituent it dominates. One theory of the organization of non-terminals and their rules is X-bar theory, which provides the foundation for a variety of grammar formalisms, including GB and GPSG. There are many variations of X-bar theory, so the particular formulation discussed here may not agree with those found in other places.

In terms of the organization of the non-terminals of a grammar, X-bar theory stipulates that there is a finite set of *lexical categories*, like $N(oun)$, $V(erb)$, $P(reposition)$, $A(djective)$, $Adv(erb)$, corresponding to the parts of speech, and that all other non-terminals are *projections* of the lexical categories. The idea of a projection is best motivated by the following example. The constituent *tall man* consists of two words, a noun and an adjective. When considering what the category of the constituent

should be, we should take into account that *tall man* behaves more like a noun than like an adjective, which can be verified by substituting *tall man* for a noun in a sentence, preserving grammaticality, and substituting it for an adjective in a sentence, not preserving grammaticality. Thus, the category of *tall man* should be derived from the category of *man*. The category that X-bar theory assigns to the phrase is called N' (pronounced N-bar). N' is a projection of N. While X-bar theory within GB considered N and N' as atomic categories, the idea that the bar-level of a node is a syntactic feature is due to GPSG.

While there are various proposal for X-bar theory, we will assume that all rules of an X-bar grammar should be of the form

$$X'' \to X', Y'' \tag{3.1}$$
$$X' \to X', Y'' \tag{3.2}$$
$$X' \to X, Y'' \tag{3.3}$$

The non-terminal Y'' has different roles in the three rule schemata, each of which has a name in X-bar theory. In rule schema 3.1, Y'' is called the *specifier*; in rule schema 3.2, it is called the *adjunct*, and in rule schema 3.3, it is called the *complement*. In each of the rules, the X or X' on the right hand side is called the *head*.

It has been observed in a variety of contexts Kornai and Pullum (1990); Kracht (1997b); Palm (1999) that it is desirable to dispense with the bar-feature and to define the constraints posed by the X-bar schemata in terms of projections. Thus, we would like to define a constraint that states that every node has a path to a leaf such that the node, the leaf, and all the nodes on the path have the same lexical features. This can expressed in $\mathcal{L}_{cp}$ as follows. First, we state that a feature φ belongs to a head:

$$hd\ \varphi \equiv \varphi \wedge head$$

Then, we state that a feature φ is projected from a leaf:

$$proj\ \varphi \equiv \langle\downarrow^*_{hd\ \varphi}\rangle(hd\ \varphi \wedge leaf)$$

Finally, we can restate the X-bar convention by requiring every node to be a projection, given a finite set of lexical features *Lex*:

$$[\downarrow^*](\bigvee_{\varphi \in Lex} proj\ \varphi)$$

Notice that we would need a feature to indicate that a node is the head in case two siblings share the same lexical feature. Furthermore, there are certain regularities that this head feature has to observe, such as that no two sisters may both be heads:

$$[\downarrow^*](head \to \neg(\langle\leftarrow\rangle head \vee \langle\rightarrow\rangle head))$$

3.1.8 Dynamic Logic of Trees

The first descriptive characterization of the regular tree languages was obtained by Doner (1970), and Thatcher and Wright (1968). They generalized Büchi's theorem to trees.

Syntax **Formulas** φ
$p_i \mid \neg\varphi \mid \varphi \wedge \psi \mid [\pi]\varphi$
Programs π
$\rightarrow \mid \leftarrow \mid \uparrow \mid \downarrow \mid ?\varphi \mid \pi;\sigma \mid \pi \cup \sigma \mid \pi^*$

Figure 3.4: Dynamic logic of trees

Theorem 3.1.8. *The following are equivalent for a tree language $L \subseteq T_\Sigma$:*

1. *L is regular.*
2. *L is definable in MSO.*

Kracht (1995) introduced PDL on trees in the context of model theoretic syntax.

While the correspondence between $\mathcal{X}_{until}$ and FO continues to hold in the generalization from strings to trees, the same is not true for the correspondence between PDL and MSO on strings, as was shown by Kracht, a topic we shall investigate in detail in the next section.

3.1.9 Undefinability: Inessential Features

The relationships between the three logics discussed above are well-understood, in that $\mathcal{L}_{core}$ is properly included in $\mathcal{X}_{until}$, which is properly included in PDL, which in turn is properly included in MSO. There is a central property that can be used to describe the languages that can be defined in one logic, but not in another. This property was first introduced by Kracht (1997a) and it is defined in terms of *inessential features*.

Let F be a finite set of features, $G \subseteq F$, $L \subseteq T_{\Sigma^F}$, and $\pi : \Sigma^F \to \Sigma^{F-G}$ be a projection. We call the features in G *inessential for* L if the homomorphism $\hat{\pi} : L \to T_{\Sigma^{F-G}}$ based on π is one-to-one. The intuition for this definition of inessential features is that no two trees in L can be distinguished using features in G. Thus, given a tree t in $\hat{\pi}(L)$, we can recover the features from G in t using $\hat{\pi}^{-1}$, since $\hat{\pi}$ is one-to-one.

Example 3.1.9. *The bar feature of the version of X-bar theory sketched above is inessential. To see that, notice that there is only one head (bar-level 0) which has a maximal projection (bar-level 2) and all projections in between are of bar-level 1.*

While being an inessential feature is defined with respect to a language, being eliminable is defined with respect to a logic and a language. Let F be a finite set of features, $G \subseteq F$, $L \subseteq T_{\Sigma^F}$, $\pi : \Sigma^F \to \Sigma^{F-G}$ be a projection, and $\mathcal{L}$ be a logic. Suppose that L is definable in $\mathcal{L}^F$. We say that G is *eliminable in $\mathcal{L}$ for L* if $\hat{\pi}(L)$ is definable in $\mathcal{L}^{F-G}$.

It should be noted that this definition of eliminability does not coincide with Kracht's (1997a), who defines eliminable as being globally explicitly definable.

Kracht's definition implies the definition used here, and thus is stronger. However, since we are interested in *ineliminability*, by contraposition, the definition employed here implies Kracht's definition of ineliminability.

The following, well-known, inclusions follow primarily from the definition of the three modal logics.

Theorem 3.1.10. $\mathcal{L}_{core} \leq \mathcal{L}_{cp} \leq PDL_{tree} \leq MSO$

Proof. The first two inclusions follow from the definitions of these logics. The third inclusion follows from the fact that transitive closure is MSO-definable. □

Next, we consider strictness of these inclusions.

Proposition 3.1.11. *(Schlingloff, 1992) Let $F = \{a,b\}$. The tree language $L_1 \subseteq T_{\Sigma^F}$ such that each tree in L_1 contains a path from the root to a leaf at which exactly one a holds is not $\mathcal{L}_{core}$-definable, but is $\mathcal{L}_{cp}$-definable.*

Proposition 3.1.12. *Let $\Sigma = \{\wedge, \vee, 0, 1\}$. The tree language $CVP \subseteq T_\Sigma$ such that each tree in CVP evaluates to true is not $\mathcal{L}_{cp}$-definable, but is PDL_{tree}-definable.*

Proof. Potthoff (1994b) showed that CVP is not definable in an extension of first-order logic with modular counting quantifiers, and since $\mathcal{L}_{cp}$ is equivalent to first-order logic on trees Afanasiev et al. (2005), the undefinability follows. That CVP is definable in PDL_{tree} is shown in Afanasiev et al. (2005). □

Proposition 3.1.13. *(Kracht, 1999, 2001) Let $F = \{p,q\}$. Let $L_2 \subseteq T_{\Sigma^F}$ where each tree in L is a ternary branching tree such that p is true along a binary branching subtree and q is true at all leaves at which p is true. The language $L_3 \subseteq T_{\Sigma\{q\}}$ obtained from the projection that eliminates p is not PDL_{tree}-definable, but is MSO-definable.*

These three propositions demonstrate the strictness of the inclusion of the three modal logics and MSO. Next, we will consider how languages that are undefinable in one of these logics can be defined with additional features.

Theorem 3.1.14. *(Tiede, 2005) There exists a set of features F, a tree language $L \subseteq T_{\Sigma^F}$, and a subset $G \subseteq F$, such that G is ineliminable in $\mathcal{L}_{core}$ (resp. $\mathcal{L}_{cp}$) but eliminable in $\mathcal{L}_{cp}$ (resp. PDL_{tree}).*

Proof. Both of these constructions work the same way. Given two of our logics $\mathcal{L}_1, \mathcal{L}_2$, with $\mathcal{L}_1 < \mathcal{L}_2$, pick a tree language, L, that is not definable in $\mathcal{L}_1$ but is definable in $\mathcal{L}_2$, which exists by propositions 3.1.11 and 3.1.12.

By Theorem 3.1.10, we know that L is regular, and by Theorem 3.1.6, we know that any local tree language is definable in $\mathcal{L}_1$. Given a deterministic FTA $M = (\Sigma, Q, F, \Delta)$, with $L = L(M)$, we can use theorem 3.1.5 to construct a local tree language $L' \subseteq T_{\Sigma \times Q}$ such that $\hat{\pi}(L') = L$. Now, the features in Q are inessential, since M is deterministic, but ineliminable, since L is undefinable in $\mathcal{L}_1$. However, since L is definable in $\mathcal{L}_2$, the features in Q are eliminable in $\mathcal{L}_2$. □

The previous theorem can be strengthened in that it can be used to *characterize* the tree languages that are undefinable in some logic $\mathcal{L}_1$ but definable in some other logic $\mathcal{L}_2$, with $\mathcal{L}_1 \leq \mathcal{L}_2$.

Theorem 3.1.15. *(Tiede, 2005) Any tree language that is not definable in $\mathcal{L}_{core}$ (resp. $\mathcal{L}_{cp}$) but is definable in $\mathcal{L}_{cp}$ (resp. PDL_{tree}) can be defined with additional, inessential features in $\mathcal{L}_{core}$ (resp. $\mathcal{L}_{cp}$) that are not eliminable in $\mathcal{L}_{core}$ (resp. $\mathcal{L}_{cp}$).*

3.1.10 Model Theoretic Syntax and Parsing

Recall that we generalized from strings to trees because we wanted to retain decidability and because we wanted to have a formalism that associates grammatical structure to an unstructured string. While decidability has been retained by this move, we need to say a little bit about how model theoretic syntax associates structures with strings. It should be noted that CFGs are formalisms that generate strings and that the structures that they assign to the strings arise in the process of generating the string, i.e. that trees are not a primary but a derived notion for formal grammars, Tree Adjoining Grammars being a notable exception. It should also be noted that, in our move to logics of trees, strings are no longer a primary notion because we are talking about trees directly. However, when we are interested in, say, checking whether a particular sentence is grammatical, we are given a string. So, while parsing, the process of determining whether a given grammar generates a given string, for CFG amounts to checking whether the grammar generates the string, this is not quite as straightforward here. The following quote from Backofen et al. (1995) gives an outline of how parsing in the logical framework might look like:

> The intent here is to translate a given grammar G into a formula φ_G such that the set of trees generated by the grammar is exactly the set of trees that satisfy φ_G. Parsing, then, is just identifying the set of models of φ_G that yield a given string.

Following an idea proposed by Cornell (2000) in the context of parsing with finite tree automata, we can improve on the above parsing procedure by observing that we can describe the set of all trees that yield a given string w, φ_w, and then simply check whether $\varphi_w \wedge \varphi_G$ is satisfiable. Notice, though, that having moved from logics of strings to logic of trees entails that the complexity of parsing, which in the string case is that of *model checking*, now is that of *satisfiability checking*. For all of the modal logics considered here, satisfiability checking is EXPTIME-complete. This is still significantly better than MSO or even FO with $\triangleleft^*$ both of which are non-elementary. However, model checking for the modal logics considered here is linear. For another approach to parsing and model theoretic syntax, see Palm (2001).

3.1.11 Variations

Just as in the case of strings, monadic transitive closure (MTC) and least fixed point (MLFP) logic and logics with modular counting quantifiers have been considered on trees (Potthoff, 1994a), as well as Thomas' chain and anti-chain logics (Thomas, 1997). While, over trees, MLFP is equally expressive as MSO, the question whether this equivalence also holds for MTC is currently open.

Kracht (2003) also considers a modal logic with quantifiers ranging over propositions which is equivalent to MSO over trees.

3.1.12 Extensions

While the fact that natural languages are not regular has been known since the 1950s, examples of non-context-free phenomena in natural languages were only found in the 1980s; see Shieber (1985). Thus, we again need to consider how to strengthen the logics employed here if we want this approach to be applicable to all natural languages.

One approach, a generalization of the logical characterization of CFLs to trees, is Langholm's (2001) characterization of the indexed languages by an extension of MSO which generalizes the logical characterization of CFLs mentioned above to trees. The indexed languages are located strictly between the CFLs and the context-sensitive languages. However, as was pointed out above, since parsing with tree logics involves testing for satisfiability rather than model checking, using an undecidable logic makes this approach uninteresting to computational linguistics.

Other approaches to extending model theoretic syntax to non-regular tree languages include Rogers' (2003) extension of MSO to n-dimensional trees and the approach by Mönnich and colleagues (Kolb et al., 2003) that encodes non-regular tree language in regular tree languages. Both approaches have in common that they introduce a new level of abstraction, since the direct connection between a logical formula and the tree it encodes is only available via a translation, which is explicit only in the latter approach. While this move from trees to more complex structures is analogous to the move from strings to trees, the latter move still corresponds to structures employed by linguists (derivation trees) while the former does not. However, both approaches retain decidability. Whether decidable, non-regular extensions of PDL can be used to define interesting classes of tree languages is, at present, an open problem.

3.1.13 Assessment: Why Modal Logic for Syntax and Which One?

The foregoing multitude of tree logics raises two questions: what are the advantages and disadvantages of modal logics over classical logics for the description of trees, and similarly between the different modal logics? With respect to classical logic, the advantage is not, as in the general case, that modal logics are decidable while classical logic is not, since even MSO over trees is decidable. However, there is an advantage in complexity: all the modal logics considered are EXPTIME-complete (Afanasiev et al., 2005), while MSO and FO with $\triangleleft^*$ are not elementary. One exception is FO with two successors, S_1, S_2 which is elementary (Ferrante and Rackoff, 1979), but not very expressive, since not even $\triangleleft^*$ is FO definable from S_1, S_2. For further discussions of complexity theoretic aspects of MSO, see Libkin (2004).

Another more general question: why should logic be used at all to formalize grammatical theories? The first advantage that the approach outlined in this chapter has is that it connects a descriptive approach to grammars to a procedural approach: grammars formalized in these logics can be translated into tree automata which can be implemented. Another issue has to do with methodology in linguistics. While some linguists have become downright hostile towards formalization, the methodological paradigm of Government and Binding theory was to formulate more and more "principles;" i.e., general statements about the structure of sentences that were supposed to be true for all languages. However, it was quite unclear how one would check whether or not any new principle was consistent with all the previously stated principles. Formalizing principles from GB in one of these logics would allow to check whether an adding a given principle would make a particular theory contradictory. For further discussions of methodological issues in GB, see Hintikka and Sandu (1991).

3.2 Conclusion and Open Problems

Open problems in model theoretic syntax include computational implementations, for which some progress has already been made by the existing implementations of monadic second order logic Klarlund (1998). However similar implementations of modal logics of trees or applications of the existing applications to linguistic problems do not seem to exist. The relationship between the different approaches to extending model theoretic syntax to non-regular tree languages outlined above is also currently open. For example, is there an easy way to translate between Rogers' extension in Rogers (2003) of MSO to n-dimensional trees and the approach by Mönnich and colleagues (Kolb et al., 2003) that encodes non-regular tree language in regular tree languages? Finally, while the different modal logics in this chapter were separated using the tree languages in Propositions 3.1.11, 3.1.12 and 3.1.13, it would be interesting to find linguistically motivated tree languages that can also separate these logics. Until such examples are found, very little motivation seems to exist to use the more expressive logics.

Bibliography

Afanasiev, Loredana, Patrick Blackburn, Ioanna Dimitriou, Bertrand Gaiffe, Evan Goris, Maarten Marx, and Maarten de Rijke (2005). PDL for ordered trees. *Journal of Applied Non-Classical Logic*, **15**(2):115–135.

Backofen, Rolf, James Rogers, and K. Vijay-Shanker (1995). A first-order axiomatization of the theory of finite trees. *Journal of Logic, Language and Information*, **4**(1):5–39.

Blackburn, Patrick, Claire Gardent, and Wilfried Meyer Viol (1993). Talking about trees. In Steven Krauwer, Michael Moortgat, and Louis des Tombe, eds., *Sixth Conference of the European Chapter of the Association for Computational Linguistics — Proceedings of the Conference*. ACL.

Blackburn, Patrick and Wilfried Meyer-Viol (1994). Linguistics, logic and finite trees. *Bulletin of the Interest Group in Pure and Applied Logics*, **2**(1):3–29.

Blackburn, Patrick, Wilfried Meyer-Viol, and Maarten de Rijke (1996). A proof system for finite trees. In Dirk van Dalen and Marc Bezem, eds., *Computer science logic (Paderborn, 1995)*. Springer, Berlin.

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

Cornell, Thomas (2000). Parsing and grammar engineering with tree automata. In Anton Nijholt Dirk Heylen and Giuseppe Scollo, eds., *Algebraic Methods in Language Processing AMiLP 2000, Iowa City, Iowa*.

Doner, John (1970). Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, **4**:406–451.

Ferrante, Jeanne and Charles W. Rackoff (1979). *The computational complexity of logical theories*. Springer, Berlin.

Gazdar, Gerald, Ewan Klein, Geoffrey Pullum, and Ivan Sag (1985). *Generalized Phrase Structure Grammar*. Harvard University Press, Cambridge.

Gécseg, Ferenc and Magnus Steinby (1997). Tree languages. In *Handbook of formal languages, Vol. 3*. Springer, Berlin.

Gorn, Saul (1967). Explicit definitions and linguistic dominoes. In *Systems and Computer Science (Proc. Conf., London, Ont., 1965)*. Univ. Toronto Press, Toronto, Ont.

Hintikka, Jaakko and Gabriel Sandu (1991). *On the Methodology of Linguistics*. Basil Blackwell, Oxford.

Klarlund, Nils (1998). Mona & fido: The logic-automaton connection in practice. In Georg Gottlob, Etienne Grandjean, and Katrin Seyr, eds., *Computer Science Logic (Brno, 1998)*. Springer, Berlin.

Kolb, Hans-Peter, Jens Michaelis, Uwe Mönnich, and Frank Morawietz (2003). An operational and denotational approach to non-context-freeness. *Theoretical Computer Science*, **293**(2):261–289.

Kornai, Andràs and Geoffrey K. Pullum (1990). The X-bar theory of phrase structure. *Language*, **66**:24–50.

Kracht, Marcus (1995). Syntactic codes and grammar refinement. *Journal of Logic, Language and Information*, **4**(1):41–60.

Kracht, Marcus (1997a). Inessential features. In Alain Lecomte, François Lamarche, and Guy Perrier, eds., *Logical aspects of computational linguistics*. Springer, Berlin.

Kracht, Marcus (1997b). On reducing principles to rules. In Patrick Blackburn and Maarten de Rijke, eds., *Specifying Syntactic Structures*. CSLI Publications, Stanford, CA.

Kracht, Marcus (1999). *Tools and techniques in modal logic*. North-Holland, Amsterdam.

Kracht, Marcus (2001). Logic and syntax—a personal perspective. In Michael Zakharyaschev, Krister Segerberg, Maarten de Rijke, and Heinrich Wansing, eds., *Advances in modal logic, Vol. 2*. CSLI Publications, Stanford, CA.

Kracht, Marcus (2003). *The Mathematics of Language*. de Gruyter, Berlin.

Langholm, Tore (2001). A descriptive characterisation of indexed grammars. *Grammars*, **4**(3):205–262.

Lautemann, Clemens, Thomas Schwentick, and Denis Thérien (1995). Logics for context-free languages. In Leszek Pacholski and Jerzy Tiuryn, eds., *Computer science logic (Kazimierz, 1994)*. Springer, Berlin.

Libkin, Leonid (2004). *Elements of finite model theory*. Springer, Berlin.

Marx, Maarten (2004). Conditional XPath, the first order complete XPath dialect. In *Proceedings of PODS '04*.

Moss, Lawrence S. and Hans-Jörg Tiede (2006). Applications of modal logic in linguistics. In Patrick Blackburn, Johan van Benthem, and Frank Wolter, eds., *Handbook of Modal Logic*. ELSEVIER, Amsterdam.

Palm, Adi (1999). Propositional tense logic for finite trees. In *Proceedings of Mathematics of Language (MOL 6)*.

Palm, Adi (2001). Model theoretic syntax and parsing: An application to temporal logic. In *Proceedings of Formal Grammar and Mathematics of Language (FG-MOL)*.

Potthoff, Andreas (1994a). *Logische Klassifizierung regulärer Baumsprachen*. Ph.D. thesis, Christian-Albrechts-Universität zu Kiel.

Potthoff, Andreas (1994b). Modulo-counting quantifiers over finite trees. *Theoretical Computer Science*, **126**(1):97–112.

Potthoff, Andreas and Wolfgang Thomas (1993). Regular tree languages without unary symbols are star-free. In *Fundamentals of computation theory (Szeged, 1993)*. Springer, Berlin.

Rogers, James (1998). *A descriptive approach to language-theoretic complexity*. Studies in Logic, Language and Information. CSLI Publications, Stanford, CA.

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

Schlingloff, Bernd-Holger (1992). On the expressive power of modal logics on trees. In Anil Nerode and Michael A. Taitslin, eds., *Logical Foundations of Computer Science - Tver '92*. Springer, Berlin.

Shieber, Stuart (1984). Direct parsing of ID/LP grammars. *Linguistics and Philosophy*, **7**:135–154.

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

Thatcher, James W. (1967). Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *Journal of Computer and System Sciences*, **1**:317–322.

Thatcher, James W. (1973). Tree automata: an informal survey. In *Currents in the theory of computing*. Prentice-Hall, Englewood Cliffs, N. J.

Thatcher, James W. and Jesse B. Wright (1968). Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, **2**:57–81.

Thomas, Wolfgang (1997). Languages, automata, and logic. In *Handbook of formal languages, Vol. 3*. Springer, Berlin.

Tiede, Hans-Jörg (2005). Inessential features, ineliminable features, and modal logics for model theoretic sytax. In *Proceedings of Formal Grammar and Mathematics of Language (FGMOL)*.

Chapter 4

Parsing Complexity and Model-Theoretic Syntax

Adi Palm
Fakultät für Informatik und Mathematik, Universität Passau
Passau, Germany
adi.palm@web.de

4.1 Introduction

As a primarily declarative approach, model theoretic syntax focuses the manner we formalize and describe the syntax of natural languages. However, in practice, we often make the experience that such formalisms suffer from excessive complexity. Depending on the underlying logical formalism, the complexity of proving the validity or satisfiability of a given (set of) formula(e) is typically EXPTIME complete or beyond. On the other hand, the parsing complexity of context-free languages is well-known to be in cubic time with regard to the length of the input word and quadratic time with regard to the size of the underlying grammar(see Younger, 1967).[1]

In this article, we consider the complexity of several parsing methods for model theoretic grammars. Instead of constructing new parsing algorithms for particular grammar formalism, we focus on their relation to context-free grammars. We mainly deal with the question, whether and how is it possible to transform a grammatical statement of a given model theoretic formalism into a context-free grammar or something similar. Afterward, we can employ well-known and efficient CFG-parsing formalisms. So the crucial issue is the method which is used to extract the context-free rules from a given logical formula.

As the formal foundation, we have chosen a modal language on labeled binary tree domains. For a more general and detailed discussion of modal logic within model theoretic syntax and bibliographical notes we refer to Hans Jörg Tiede's article in this issue. There are several reason why employing modal logic. On the one hand, it provides sufficient expressivity, i.e. many linguistic constraints can be expressed by means of this language. On the other hand, such formalism are well-known for their formal simplicity, i.e. the language is complete and the satisfiability problem is decidable. Note, that latter property means a necessary condition for parsing.

[1] More exactly, we obtain even better results when considering context-free parsing as a special case of binary matrix multiplication, see Lee (2002) for a discussion of this aspect.

4.2 Parsing and Model-theoretic Syntax

In general, model-theoretic syntax considers syntactic structures (of natural languages) as the models of particular formulae specifying a certain grammar or at least a grammatical property. Typically, these models are trees or at least tree-like structures. Thus, given a set of formulae Φ denoting a grammar, the grammatical structures $\mathcal{M}$ are the models of Φ, i.e. we have $\mathcal{M} \models \Phi$. Obviously, for a given Φ it is interesting to know, whether Φ has a model, i.e. Φ is satisfiable, and how these models look like (usually, the validity of Φ is of no interest). Now parsing puts another restriction to a given structure: Let Φ be a set of formulae specifying a certain grammar and w denotes some input word. Then we look for some model $\mathcal{M}$ with $\mathcal{M} \models \Phi$ such that w is the yield $\mathcal{M}$, i.e. $\mathcal{M} \models \mathit{yield}(w)$. The formula $\mathit{yield}(w)$ specifies the structures, which have w as their yield, i.e. the sequence of leaves leading w. It is not necessary that $\mathit{yield}(w)$ and Φ must be of the same logical language.

Formally seen, parsing in a model-theoretic framework means checking the satisfiability of $\Phi \cup \{\mathit{yield}(w)\}$. Hence, a given input word has at least one parse tree that is a model of Φ, if (and only if) $\Phi \cup \{\mathit{yield}(w)\}$ has a model. Further, if there is exactly one such model, w has a unique parse tree for the grammar Φ. In practice, parsing means to construct the tree models for w rather than checking only existence of such a structure. Now, if the satisfiability problem is decidable for a given formalism, then there is an effective method that constructs the set parse trees for w.

In the sequel, we discuss the complexity of three ways how parsing could work for a modal tree language,

1. *Standard Model Construction* (not discussed):
   We construct the Kripke-structures for $\Phi \cup \mathit{yield}(w)$. From this structure we may obtain all parsing trees for w that are models of Φ

2. *Constructing a Parsing-Grammar* (like 1):
   We construct the Kripke-structures for Φ which can be transformed to a corresponding context-free grammar (CFG). Then we can employ arbitrary stan-

dard parsing algorithms for CFGs to obtain the parsing trees for w.

3. *Dynamic Rule Generation*:
   Again, we employ a CFG-parser. But at a certain stage of the parsing processes, we always require only local section the corresponding CFG, which can be generated, dynamically. So we only construct the required local model of Φ instead of transforming Φ to the whole corresponding CFG.

4. *Restricting the underlying logic*:
   We restrict the shape of Φ in particular way, such that we can directly obtain the corresponding CFG.

4.3 The Modal Tree Language BTL

Now we turn to the logical framework and the corresponding structures. In essence, we try to find a balance between simplicity, sufficient expressivity, formal adequacy and generality. Since we focus on the issue of parsing (rather than discussing the formal adequacy of a particular approach), we emphasize the issues of simplicity and expressivity leading us to a modal tree language (which is obviously not the only solution to this problem)[2]. For a general discussion of modal logic in model theoretic syntax we refer to the paper of Hans-Jörg Tiede in this volume and to the overview in Blackburn et al. (2003). In detail, we employ a variation on the language of Palm (2001), which consists of two simple modal operators accessing the left-hand and right-hand child of a node. In addition, this language employs a kind of until-operator for the conditional access to some descendant of a node. We define the binary tree language BTL as follows:

$$\varphi,\psi ::= \top \mid p \mid \mathit{leaf} \mid \neg\varphi \mid \varphi\wedge\psi \mid \downarrow_1\varphi \mid \downarrow_2\varphi \mid \downarrow^*_\psi\varphi$$

Beside the usual propositional operators $\wedge$ and $\neg$, our simple binary tree language consists the boolean constants *leaf*, $\top$, a set of propositional constants $\mathcal{P} = \{p,q,\ldots\}$ and of the modal operators $\downarrow_1$ '*left child*', $\downarrow_2$ '*right child*', and $\downarrow^*_\psi$ 'some descendant'.

Next we turn to the structures of BTL. A complete binary tree domain $D \subseteq \{1,2\}^*$ denotes a set of words, that is closed with respect to prefixes, i.e. $x{\cdot}y \in D \Rightarrow x \in D$ for each $x,y \in \{1,2\}^*$, and with respect to sisters, i.e. $w{\cdot}1 \in D \Leftrightarrow w{\cdot}2 \in D$ for each $x \in \{1,2\}^*$. A binary tree $T = (D,h)$ consists of a complete binary tree domain D and a labeling function $h\colon D \to 2^P$. The semantics of a BTL formula for some tree node $n \in D$ in a binary tree $T = (D,h)$ can be defined as follows:

- $T,n \models p$ iff $p \in h(n)$ for each $p \in \mathcal{P}$
- $T,n \models \mathit{leaf}$ iff $n{\cdot}1 \notin D$ (and $n{\cdot}2 \notin D$)
- $T,n \models \downarrow_i\varphi$ iff $n{\cdot}i \in D$ and $T,n{\cdot}i \models \varphi$ for each $i \in \{1,2\}$.
- $T,n \models \downarrow^*_\psi \varphi$ iff there is some $w \in \{1,2\}^*$ such that $n{\cdot}w \in D$, $T,n{\cdot}w \models \varphi$ and for all proper prefixes w' of w it is $T,n{\cdot}w' \models \psi$

Obviously, we have $T,n\downarrow_1(\top) \Leftrightarrow T,n\downarrow_2(\top)$. The operator $\downarrow^*_\psi\varphi$ denotes the until-operator, i.e. there is a path of nodes satisfying the intermediate condition ψ until there is a node satisfying the target condition φ. The atomic proposition *leaf* indicates the leaf nodes of a tree. Note that for parsing and most other linguistic issues, we are only interested in finite trees. However, due to the compactness, finiteness cannot be expressed by means of temporal logic.

4.4 Constructing Tree Models

As noted earlier, modal logic offers simple methods for constructing the model for a satisfiable formula. Subsequently, we will provide a more or less simple algorithm on how to construct a tree-model for a given set of satisfiable BTL formulae. Furthermore, this will also lead to the completeness and the decidability of BTL. In essence, there are similar results and techniques used for several related formalisms as for propositional dynamic logic (Fischer and Ladner, 1979; Kracht, 1995); for temporal logic in general (Schlingloff, 1990); for propositional linear-temporal logic PTL (Lichtenstein and Pnueli, 2000), for modal logic on (finite) trees (Blackburn and Meyer-Viol, 1994; Blackburn et al., 1996; Blackburn and Meyer-Viol, 1997); for tree-automata (Cornell, 2000) and for several other tree-related formalisms (Kamp, 1968; van Benthem, 1984; Kracht, 1995; Palm, 1997, 1999a; Rogers, 1998) and many others.

The construction of tree models for BTL-formulae works as follows. At first we establish certain sets of formulae called atoms. Each atom describes the kind of tree nodes satisfying the formulae occurring in the atom. Next we construct the reachability relation among atoms. Finally, we obtain the tree models by combing the atoms with the reachability relation.

Let φ be a BTL-formula. The *closure* of φ, $Cl(\varphi)$, is the smallest set of formulae containing φ, $\neg\varphi$, and all subformulae of φ and their negations. In addition, the closure must include *leaf*, $\neg$*leaf*. Obviously, we can show by induction on the structure of φ that $|Cl(\varphi)| \leq 2\cdot|\varphi|+2$. Now, we can define the set of atoms of φ, $At(\varphi)$as follows

Definition 4.4.1 (atom). *A set $A \subseteq Cl(\varphi)$ is an atom, if every $\psi \in Cl(\varphi)$ meets the following conditions:*

$(A1)$ $\psi \in A \;\Leftrightarrow\; \neg\psi \notin A$
$(A2)$ $\psi \equiv \psi_1\wedge\psi_2 :\; \psi_1\wedge\psi_2 \in A \;\Leftrightarrow\; \psi_1,\psi_2 \in A$
$(A3)$ $\psi \equiv \downarrow_i\psi' :\; \downarrow_i\psi' \in A \;\Rightarrow\; \neg\mathit{leaf} \in A$
$(A4)$ $\psi \equiv \downarrow^*_{\psi_1}\psi_2 :\; \downarrow^*_{\psi_1}\psi_2 \in A \;\Rightarrow\; \{\psi_1,\psi_2\}\cap A \neq \emptyset$
$(A5)$ $\psi \equiv \downarrow^*_{\psi_1}\psi_2 :\; \psi_2 \in A \;\Rightarrow\; \downarrow^*_{\psi_1}\psi_2 \in A$

[2] Many aspects discussed in the sequel would also apply to more complex formalism. However, the amount for defining and proving certain properties would increase at least in the same way.

Then $At(\varphi)$ denotes the set of atoms of φ.

Obviously, an atom denotes a (propositionally) consistent subset of $Cl(\varphi)$ and, clearly, it is $AT(\varphi) \subset 2^{Cl(\varphi)}$ and $|AT(\varphi)| \leq 2^{2\cdot|\varphi|+2}$. An atom is called *initial* (or a root atom) if it includes the underlying formula φ. An atom A is called *terminal* (or leaf atom), if *leaf* belongs to A. Each atom corresponds to a certain kind of tree nodes (or state in the Kripke-structure). Therefore, the formulae occurring in some atom describe the formal properties of this kind of tree node (or state). Due to the maximality of atoms, we know the validity for each subformula. Consequently, an atom completely specifies its propositional and its structural properties. The latter is useful for establishing a structural relation among atoms:

Definition 4.4.2 (reachability). *We define the reachability relation $(A, B_1, B_2) \in \mathcal{R}_\varphi$ as follows*

1. $\neg leaf \in A$
2. *For all $\downarrow_i \phi \in At(\Phi)$ holds: $\downarrow_i \phi \in A \Leftrightarrow \phi \in B_i$*
3. *For all $\downarrow^*_\psi \phi \in At(\Phi)$ holds:*
   (a) $\downarrow^*_\psi \phi, \neg\phi \in A \Rightarrow \downarrow^*_\psi \phi \in B_1 \cup B_2$
   (b) $\downarrow^*_\psi \phi \in B_1 \cup B_2, \psi \in A \Rightarrow \downarrow^*_\psi \phi \in A$

The reachability relation considers all structural formulae in $Cl(\varphi)$ (i.e. formulae employing the operators *leaf*, $\downarrow_1$, $\downarrow_2$ and $\downarrow^*$) and establishes the corresponding structural relation. Now we can combine atoms the reachability relation, in order to construct preliminary tree models of φ.

Definition 4.4.3 (pre-tree). *Let $T = (D, h)$ be an $At(\varphi)$-labeled binary tree. T is called a pre-tree for $At(\varphi)$ if $\varphi \in h(\varepsilon)$ and the following conditions apply to each $d \in D$*

1. $leaf \in h(d) \Leftrightarrow \{d.1, d.2\} \cup D = \emptyset$.
2. $d.1, d.2 \in D \Rightarrow (h(d), h(d.1), h(d.2)) \in \mathcal{R}_\varphi$.

If $T = (D, h)$ is finite, the we call T a finite pre-tree for $At(\varphi)$.

So far, the construction is not complete, since it fails to consider the properties of the $\downarrow^*$-operator in a sufficient way. Any formula $\downarrow^*_\psi \varphi$ is only considered locally. So it is still possible, that we have an infinite path atoms including ψ and $\downarrow^*_\psi \varphi$ without reaching an atom including φ.

Definition 4.4.4 (full-filling path). *Let $d_0, d_1, \ldots, d_n$ be a finite path in a pre-tree $T = (D, h)$, i.e. $d_{i+1} \in \{d_i.1, d_i.2\}$ for each $0 \leq i < n$. Then $d_0, d_1, \ldots, d_n$ is a* full-filling path *for some $\downarrow^*_{\psi_1} \psi_2 \in At(\varphi)$, if $\downarrow^*_{\psi_1} \psi_2 \in h(d_i)$ for each $0 \leq i \leq n$ and $\psi_2 \in h(d_n)$.*

We obtain the well-structured pre-trees by restricting them in such a way, that only full-filling paths may occur:

Definition 4.4.5 (well-structured pre-trees). *A pre-tree $T = (D, h)$ for $At(\varphi)$ is called well-structured, if for every $\downarrow^*_{\psi_1} \psi_2 \in At(\varphi)$ and every $d_0 \in D$ with $\downarrow^*_{\psi_1} \psi_2 \in h(d_0)$ exits a full-filling path $d_0, d_1, \ldots, d_n$ in $T = (D, h)$.*

Actually, for finite trees the restriction to full-filling paths is not required:

Lemma 4.4.6. *A finite pre-tree $T = (D, h)$ for $At(\varphi)$ is always well-structured .*

Proof. By the properties of the reachability relation, for each $n \in D$ with $\downarrow^*_{\psi_1} \psi_2 \in h(n)$ we have $\psi_2 \in h(n)$ or $\neg\psi_2, \psi_1 \in h(n)$ and $\downarrow^*_{\psi_1} \psi_2 \in h(n.1) \cup h(n.2)$. Accordingly, we either obtain an infinite path nodes $n, n_1, \ldots$ with $\psi_1, \neg\psi_2, \downarrow^*_{\psi_1} \psi_2 \in h(n_i)$ for every i or there is a full-filling path $n, n_1 \ldots, n_k$. Since T is finite, the latter must be true. □

Now we show that well-structured pre-trees for $At(\varphi)$ are valid models for φ.

Lemma 4.4.7. *Let $T = (D, h)$ a well-structured pre-tree for $At(\varphi)$. For each $\psi \in Cl(\varphi)$ and $n \in D$ we have $T, n \models \psi$ iff $\psi \in h(n)$.*

Proof. We show this by structural induction on ψ:

1. $\forall p \in \mathcal{P}$: $p \in h(n) \Leftrightarrow p \in \overline{h}(n) \Leftrightarrow T, n \models p$.
2. $\neg\psi \in h(n) \overset{\text{Def. At.}}{\Longleftrightarrow} \psi \notin h(n) \overset{\text{Ind. Hyp.}}{\Longleftrightarrow} T, n \not\models \psi \overset{\text{Sem.}}{\Longleftrightarrow} T, n \models \neg\psi$.
3. $\psi \wedge \psi' \in h(n) \overset{\text{Def. At.}}{\Longleftrightarrow} \psi, \psi' \in h(n) \overset{\text{Ind. Hyp.}}{\Longleftrightarrow} T, n \models \psi$ and $T, n \models \psi' \overset{\text{Sem.}}{\Longleftrightarrow} T, n \models \psi \wedge \psi'$.
4. $\downarrow_i \psi \in h(n) \overset{\text{Reach.}}{\Longleftrightarrow} \psi \in h(n.i) \overset{\text{Ind. Hyp.}}{\Longleftrightarrow} T, n.i \models \psi \overset{\text{Sem.}}{\Longleftrightarrow} T, n \models \downarrow_i \psi$.
5. $\downarrow^*_{\psi_1} \psi_2 \in h(n) \overset{\text{well str.}}{\Longleftrightarrow}$ there is a full-filling path $n_0, \ldots, n_k$ with $n = n_0, k \geq 0$ such that $\psi_1 \in h(n_i)$ for $i < k$ and $\psi_2 \in h(n_k) \overset{\text{Ind. Hyp.}}{\Longleftrightarrow} T, n_i \models \psi_1$ for $i < k$ and $T, n_k \models \psi_2 \overset{\text{Sem.}}{\Longleftrightarrow} T, n \models \downarrow^*_{\psi_1} \psi_2$.

□

Vice versa, the construction of well-structured pre-trees is complete for BTL:

Theorem 4.4.8. *A BTL-formula φ is satisfiable iff there is a well-structured pre-tree $T = (D, h)$ for $At(\varphi)$*

Proof. $\Rightarrow$: Let $T = (D, h)$ be a model of φ. We show that there is a corresponding well-structured pre-tree $\overline{T} = (D, \overline{h})$ with $h(n) = \overline{h}(n) \cap \mathcal{P}$ for each $n \in D$ such that $\overline{T} \models \varphi$. We define $\overline{h}(n) := \{\phi \in At(\varphi) \mid T, n \models \psi\}$. Obviously we have, $(\overline{h}(n), \overline{h}(n.1), \overline{h}(n.2)) \in \mathcal{R}_\varphi$, so . $\overline{T} = (D, \overline{h})$ is a well-structured pre-tree for $At(\varphi)$.
$\Leftarrow$: Let $T = (D, h)$ be a well-structured pre-tree for $At(\varphi)$. By $\varphi \in h(\varepsilon)$ and Lemma 4.4.7 we have $T, \varepsilon \models \varphi$. Thus φ is satisfiable. □

Theorem 4.4.9. *For every satisfiable BTL-formula φ there is a $At(\varphi)$-CFG G_φ, such that the set of derivation trees of G_φ is equal to the set of finite (well-structured) pre-trees of $At(\varphi)$*

Proof. We define the CFG $G_\varphi = (N_\varphi, T_\varphi, S_\varphi, P_\varphi)$ as follows:

$$\begin{aligned} N_\varphi &:= \{A \in At(\varphi) \mid \neg leaf \in A\} \\ T_\varphi &:= \{A \in At(\varphi) \mid leaf \in A\} \\ S_\varphi &:= \{A \in At(\varphi) \mid \varphi \in A\} \\ P_\varphi &:= \{A \to B_1 B_2 \mid (A, B_1, B_2) \in \mathcal{R}_\varphi\} \end{aligned}$$

Note that every derivation tree of a CFG is finite, and by lemma 4.4.6 every finite pre-tree is well-structured. Moreover, for each node n of a derivation tree of G_φ and a pre-tree of $At(\varphi)$ holds:

- If n is the root then $\varphi \in h(n)$.
- If n is a non-leaf node successors n_1 and n_2, then $\neg leaf \in h(n)$ and $(n, n_1, n_2) \mathcal{R}_\varphi$.
- If n is a leaf, then $pleaf \in h(n)$

Every finite $At(\varphi)$-labeled binary tree T, is a derivation tree of $G\varphi$ iff T is a pre-tree for $At(\varphi)$. □

4.5 Complexity of the Construction

The above method of constructing models for BTL-formulae already reveals some aspects on the worst-case complexity of the satisfiability problem. Before we turn to details, we mention some other relevant results on similar formalisms. The satisfiability problem for linear propositional temporal logic LPTL is PSPACE-complete (Sistla and Clarke, 1985; Reynolds, 2003). The satisfiability problem for CTL is EXPTIME-complete (Kozen and Tiuryn, 1990), and for CTL*, it is EX^{2}PTIME-complete (Emerson and Sistla, 1984). Since BTL denotes a special case of CTL, the satisfiability problem of BTL must be at most EXPTIME-complete. On the other hand, LPTL denotes the linear special case of BTL, so PSPACE-completeness is a lower bound for the worst-case complexity of the satisfiability problem in BTL.

It is important to note, that these complexity classes only consider the worst cases, and there is (still) the hope that for practical tasks the bad cases only occur rarely. However, we will show that such bad cases could occur, even though it is not necessary. This is due to the way we have constructed the models for a BTL formula. Nevertheless, we have already mentioned the exponential amount for checking the satisfiability, we show this now in detail.

Proposition 4.5.1. *The satisfiability problem of some BTL formulae φ can be solved in time $O(2^{|\varphi|})$.*

Proof. Obviously, we have $|At(\varphi)| \leq 2^{|Cl(\varphi)|} \leq 2^{2|\varphi|+2}$. We construct $At(\varphi)$-CFG G_φ of theorem 4.4.9 and consider the emptiness problem of the language $L(G_\varphi)$. For the size of G_φ we have $|N \cup T| = |At(\varphi)|$, $|P| \leq |At(\varphi)|^3$ and $|G| = |N \cup T| \cdot |P| \leq |At(\varphi)|^4$ which is still in $O(2^{|\varphi|})$. Eventually, the emptiness problem of context-free languages can be solved in time $O(|G\varphi)$. □

Consequently, we can check whether a BTL formula is satisfiable in at most single exponential time with respect to the length of this formula. Nevertheless, we try to find a better result, but we obtain the same result even for weaker modal languages. As shown in Blackburn et al. (1996) using the results of Spaan (1993) and Hemaspaandra (1996)(née Spaan), the satisfiability problem for a modal language for finite trees employing modalities for the immediate successors and its transitive closure, but without until-operator, is already in EXPTIME, too. Therefore, we obtain for BTL:

Theorem 4.5.2. *The satisfiability problem for BTL is* EXPTIME*-complete.*

Proof. By Proposition 4.5.1 the satisfiability problem is EXPTIME and, according to Spaan (1993); Hemaspaandra (1996), a proper subset of BTL is already EXPTIME-complete. This proper subset consists of a universal modality $[r]$ and its transitive closure $[r^*]$ (or their existential counterparts $\langle r \rangle$ and $\langle r^* \rangle$). In the case of BTL, these are the formulae of the type $\neg\downarrow_1 \neg\psi \wedge \neg\downarrow_2 \neg\psi$ (all immediate successors satisfy ψ) and $\neg\downarrow_\top^* \neg\psi$ (all descendants satisfy ψ), and r denotes the immediate successor relation. □

Furthermore, Blackburn et al. (2003) established the same complexity result for three modal tree languages, which only differ in the way of combining modal operators. In essence, they showed that the consequence problem for these languages is EXPTIME-complete. Especially, one of these languages, the 'Propositional Tense Logic for Finite Trees'(Palm, 1999b), is more or less equal to BTL, but it is not restricted to binary branching trees.

As mentioned above this is still a worst-case estimation. Therefore, it is highly interesting, how general this result is. Unfortunately, it is rather general for the construction method presented above. We illustrate this for a special case, where we know for sure that there is much better result than a CFG of exponential size. This case is the characteristic BTL formula $\chi(G)$ of an arbitrary binary context-free grammar G. Obviously, the optimal solution should be the grammar itself, i.e. $G_{\chi(G)} = G$. Unfortunately, we will see that the atom-based transformation clearly yields this grammar on the whole, but the corresponding set of atoms and the reachability relation is still of exponential size. Therefore, the exponential complexity seems to be a general property rather than a worst-case estimation.

Now we consider context-free grammars and their characterization by means of BTL formulae. In essence, a (binary) CFG $G = (N, T, P, S)$ consists of a finite set of non-terminal and terminal symbols $N = \{A_1, A_2, \ldots\}$ and $T = \{a_1, a_2, \ldots\}$ (with $N \cap T = \emptyset$), respectively, some

start symbol $S \in N$ and a finite set of productions $P \subseteq N \times (N \cup T)^2$. Then we can establish a corresponding BTL formula $\chi(G)$ representing G as follows (where $\Downarrow^* \varphi$ abbreviates $\neg\downarrow_\top^*\neg\varphi$:

- *Start Assertion*: $\chi_S = S$.
- *Exclusive Symbol Assertion*: $\Downarrow^* \chi_\Sigma = \Downarrow^* \bigvee_{p\in\mathcal{P}} p \wedge (\bigwedge_{q\in\mathcal{P}\setminus\{p\}} \neg q))$
- *Leaf Assertion*: $\Downarrow^* \chi_L = \Downarrow^* (\bigvee_{a\in T} a \wedge \mathit{leaf}) \vee (\bigvee_{A\in N} A \wedge \neg \mathit{leaf})$
- *Successor Assertion*: $\Downarrow^* \chi_P = \Downarrow^* \bigwedge_{A\in N}(A \to \bigvee_{A\to BC\in P}(\downarrow_1 B \wedge \downarrow_2 C))$

We summarize these assertions to a single formula $\chi(g)$ which is called the *characteristic CFG-formula* for G.

Definition 4.5.3 (characteristic CFG-formula). *Let $G = (N,T,P,S)$ be a binary CFG, i.e. $P \subseteq (N \to (N \cup T)^2)$. Then we define the characteristic CFG-formula $\chi(G)$ for G as follows where $\Downarrow^* \varphi \equiv \neg\downarrow_\top^*\neg\varphi$:*

$$\chi(G) \;=\; S \wedge \Downarrow^* \chi_\Sigma(G) \wedge \Downarrow^* \chi_P(G) \wedge \Downarrow^* \chi_L(G)$$

where $\mathcal{P}_G = T \cup \{A \in N \mid \exists \alpha \in (N \cup T)^2 \colon A \to \alpha \in P\}$.

The characteristic CFG-formula $\chi(G)$ translates a context-free grammar into a corresponding BTL formula. Now we establish the soundness of this transformation such that G and $\chi(G)$ accept the same set of binary branching labeled trees:

Lemma 4.5.4. *Let G be context-free grammar and $\chi(G)$ the corresponding characteristic CFG-formula as specified in Definition 4.5.3. Then we have for all binary trees t:*

$$t \models \psi(G) \;\Leftrightarrow\; t \in T(G)$$

Proof. $\Leftarrow$ By the construction of $\chi(G)$ every tree in $T(G)$ satisfies $\chi(G)$. $\Rightarrow$ We show this by induction on t. The root node obviously wears the label S. Now we consider some node n and assume that the claim is true for the structure above n. By the exclusive symbol assertion there is exactly one proposition in $\mathcal{P}$ that is true at n and that corresponds to some symbol of G. If n is a leaf, it satisfies exactly one proposition $a \in \mathcal{P}$, which must be a terminal symbol, due to the leaf assertion. Otherwise n is an inner node satisfying the successor assertions. Therefore, n must have two successors that exclusively satisfy some propositions $X \in \mathcal{P}$ and $Y \in \mathcal{P}$, respectively. Accordingly, there must be a rule $A \to XY$ in P such that G accepts t at n, too. □

At next, we examine the properties of $\chi(G)$ concerning model construction and model checking. Ignoring the special shape of a characteristic CFG-formula, the model construction works in the standard way leading to a number of atoms in $O(2^{|G|})$. According to the exclusive symbol assertion, each atom includes exactly one positive propositional constant while the other ones must be negated. Thus we have atoms of the shape

$$\{\neg p_1, \ldots, \neg p_{i-1}, p_i, \neg p_{i+1}, \ldots, \neg p_k, \ldots\}.$$

However, are there any atoms failing that scheme? Consider the negated version of the exclusive symbol assertion which is also a member of $Cl(\chi(G))$. There is no reason why such atoms should be absent, but as we shall see below, they will not contribute to the resulting $At(\chi(G))$-CFG.

Now we turn to the ternary reachability relation $R_{\chi(G)} \subseteq At(\chi(G))^3$ between atoms. At first, we only care about the universal assertions. According to the construction of atoms , we obtain the following relationship between atoms A and the $\Downarrow^*$-formulae:

$$\Downarrow^* \psi \in A \text{ iff}$$
$$\psi, \mathit{leaf} \in A \text{ or } \psi, \neg\mathit{leaf}, \downarrow_1\Downarrow^*\psi, \downarrow_2\Downarrow^*\psi \in A$$

This leads to the $\Downarrow^*$-rule for non-leaf atoms:

$$\neg\mathit{leaf}, \Downarrow^* \psi \in A \Rightarrow \psi, \downarrow_1\Downarrow^*\psi, \downarrow_2\Downarrow^*\psi \in A$$

Accordingly, all atoms $A, B_1, B_2 \in At(\chi(G))$ with $(A, B_1, B_2) \in \mathcal{R}_{\chi(G)}$ satisfy

$$\neg\mathit{leaf}, \Downarrow^* \psi \in A \;\Rightarrow \Downarrow^* \psi \in B_1 \text{ and } \Downarrow^* \psi \in B_2$$

for every universal assertion $\Downarrow^* \psi$, and, moreover we also have

$$\neg\mathit{leaf}, \Downarrow^* \psi \in A \;\Rightarrow\; \psi \in A \text{ and } \psi \in A_1 \text{ and } \psi \in A_2.$$

Therefore, $\mathcal{R}_{\chi(G)}$ links every exclusive symbol atom only with other exclusive symbols atoms. Obviously, the root atoms with

$$At_{root}(\chi(G)) = \{A \in At(\chi(G)) \mid \chi(G) \in A\}$$

are exclusive symbol atoms and so are all reachable atoms via the ternary reachability relation $\mathcal{R}_{\chi(G)}$. Consequently, all other atoms are never reachable and they might therefore be deleted. Note that computing the unreachable atoms requires a consideration of all atoms and the reachability relation, which means a time complexity of $O(|At(\chi(G)| \cdot |\mathcal{R}_{\chi(G)}|)$. Since this has an exponential upper bound by $O(2^{|\varphi|})$, searching and eliminating the useless atoms is, in general, too expensive.

So far, we know that for a good estimation of the complexity, we must focus on the number of exclusive symbol atoms. Therefore we consider the disjunctions occurring within the successor assertion

$$\Downarrow^* \bigwedge_{A\in N} (\neg A \vee \bigvee_{A\to BC\in P} (\downarrow_1 B \wedge \downarrow_2 C))$$

and the leaf assertion

$$\Downarrow^* ((\bigvee_{a\in T} a \wedge \mathit{leaf}) \vee (\bigvee_{A\in N} A \wedge \neg\mathit{leaf})).$$

When constructing the atom we must add by the latter one the formula $\neg\downarrow_1\top$ to every atom including a positive occurrence of a leaf symbol proposition, and for a negative occurrence we must add the formula $\downarrow_1\top$. Consequently, the leaf assertion puts no alternatives on the exclusive symbol atoms. In contrast, the successor assertion leads to alternative exclusive symbol atoms for each exclusive combination of propositional constants. Therefore, we have an atom for each rule $A \to BC$ such that the atom includes A, $\downarrow_1 B \wedge \downarrow_1 C$, $\downarrow_1 B$ and $\downarrow_1 C$. Note that there could be no exclusive symbol atom including $\downarrow_1 A_1$ and $\downarrow_1 A_2$ (or $\downarrow_2 A_1$ and $\downarrow_2 A_2$) for $A_1 \neq A_2$. Consequently, we have exactly one exclusive symbol atom for each rule $A \to BC \in P$ and one for each terminal symbol $a \in T$. This leads to the following Lemma.

Lemma 4.5.5. *Let $\chi(G)$ be the characteristic CFG-formula of some context-free grammar $G = (N,T,P,S)$ and $At^{\#}(\chi(G))$ denotes the set of exclusive symbol atoms. Then we have $|At^{\#}(\chi(G))| = |G|$ where $|G| = |P| + |T|$.*

Obviously, the validity of this lemma follows from the discussion we have made above. As a consequence, it is possible to construct a modified $At(\chi(G))$-CFG whose symbols correspond to exclusive symbol atoms. All other atoms are ignored since they are not reachable from the starting symbol. The resulting modified context-free grammar only includes a fraction of the atoms the original one has, though either $At(\chi(G))$-CFG accepts the same set of finite binary trees. Thus, when ignoring the special properties of the characteristic CFG-formula, we obtain a $At(\chi(G))$-CFG with a number of atoms in $O(2^{|G|}) = O(2^{|\chi(G)|})$. However by employing the original grammar G , model checking and satisfiability can be done in linear time .

This is an important result since it means the linear complexity for satisfiability and model checking of $\chi(G)$. Consequently, for every formula $\chi(G)$ there trivially exists a linear-sized context-free grammar, but this cannot be obtained by the atom-based construction. Therefore, we look for a more efficient formalism that is able to construct models for polynomial size provided there is such a model for the underlying formula.

4.6 Dynamic Rule Extraction

Instead of generating all rules of G_φ, we only consider the rules that are necessary for parsing some input string. The method presented only works for top-down parsers. However, after employing some modification, it would work for other parsing strategies as well. At first we consider the branching consequences of a BTL-formula, which is defined as follows.

Definition 4.6.1 (branching consequences). *For every formula φ we define the branching consequences $Cn_\triangle(\varphi)$ as follows:*

$$\begin{aligned}
\psi_1 \wedge \psi_2 &: \{\psi_1,\psi_2\} \to \{\}\,\{\} \\
\neg(\psi_1 \wedge \psi_2) &: \{\neg\psi_1\} \to \{\}\,\{\} \\
&\quad \mid \{\neg\psi_2\} \to \{\}\,\{\} \\
\downarrow_1\psi &: \{\} \to \{\psi\}\,\{\} \\
\neg\downarrow_1\psi &: \{\} \to \{\neg\psi\}\,\{\} \\
\downarrow_2\psi &: \{\} \to \{\}\,\{\psi\} \\
\neg\downarrow_2\psi &: \{\} \to \{\}\,\{\neg\psi\} \\
\downarrow^*_{\psi_1}\psi_2 &: \{\psi_2\} \to \{\}\,\{\} \\
&\quad \mid \{\psi_1,\neg\psi_2\} \to \{\downarrow^*_{\psi_1}\psi_2\}\,\{\} \\
&\quad \mid \{\psi_1,\neg\psi_2\} \to \{\}\,\{\downarrow^*_{\psi_1}\psi_2\} \\
\neg\downarrow^*_{\psi_1}\psi_2 &: \{\neg\psi_1,\neg\psi_2\} \to \{\}\,\{\} \\
&\quad \mid \{\psi_1,\neg\psi_2\} \to \{\neg\downarrow^*_{\psi_1}\psi_2\}\,\{\neg\downarrow^*_{\psi_1}\psi_2\}
\end{aligned}$$

We assume that a (set of) BTL formula(e) specifies the properties of a branching node. Then $Cn_\triangle(\varphi)$ consists of a set rules, that describe the logical properties of a node and either of its successors resulting form φ:

Lemma 4.6.2. *Let n be a some branching node in a well-structured pre-tree $T = (d,h)$ for $At(\varphi)$. For each $\psi \in h(n)$ there is some $(A \to B_1 B_2) \in Cn_\triangle(\psi)$ such that $A \subseteq h(n)$, $B_1 \subseteq h(n.1)$ and $B_2 \subseteq h(n.1)$.*

Proof. We only consider for two cases $\neg(\psi_1 \wedge \psi_2)$ and $\downarrow^*_{\psi_1}\psi_2$; the others can be obtained likewise.

1. $\neg(\psi_1 \wedge \psi_2) \in h(n) \overset{\text{atom}}{\iff} \psi_1 \wedge \psi_2 \notin h(n) \overset{\text{atom}}{\iff} \psi_1,\psi_2 \notin h(n) \overset{\text{atom}}{\iff} \neg\psi_1 \in h(n)$ or $\neg\psi_2 \in h(n)$.

2. $\downarrow^*_{\psi_1}\psi_2 \in h(n) \overset{\text{atom}}{\Longrightarrow} \psi_1 \in h(n)$ or $\psi_2 \in h(n) \iff$
   $\psi_2 \in h(n)$ or $\psi_1,\neg\psi_2 \in h(n) \overset{\mathcal{R}}{\Longrightarrow}$
   $\psi_2 \in h(n)$ or $\psi_1,\neg\psi_2 \in h(n), \downarrow^*_{\psi_1}\psi_2 \in h(n.1) \cup h(n.2) \iff$
   $\psi_2 \in h(n)$ or $\psi_1,\neg\psi_2 \in h(n), \downarrow^*_{\psi_1}\psi_2 \in h(n.1)$ or $\psi_1,\neg\psi_2 \in h(n), \downarrow^*_{\psi_1}\psi_2 \in h(n.2)$

For reasons of readability we omitted the atomic proposition $\neg$*leaf* which is always a member of $h(n)$. □

Likewise, we define the leaf consequences:

Definition 4.6.3 (leaf consequences). *For every formula φ we define the leaf consequences $Cn_\#(\varphi)$ as follows:*

$$\begin{aligned}
\psi_1 \wedge \psi_2 &: \{\psi_1,\psi_2\} \\
\neg(\psi_1 \wedge \psi_2) &: \{\neg\psi_1\} \mid \{\neg\psi_2\} \\
\neg\downarrow_i\psi &: \{\} \\
\downarrow^*_{\psi_1}\psi_2 &: \{\psi_2\} \\
\neg\downarrow^*_{\psi_1}\psi_2 &: \{\neg\psi_1,\neg\psi_2\} \mid \{\psi_1,\neg\psi_2\}
\end{aligned}$$

The leaf consequence presumes that the formula(e) considered describe a leaf node, i.e. *leaf* must be true.

Lemma 4.6.4. *Let n be a leaf in a well-structured pre-tree $T = (d,h)$ for $At(\varphi)$. For each $\psi \in h(n)$ there is some $A \in Cn_\#(\psi)$ such that $A \subseteq h(n)$.*

Proof. We only consider the case $\downarrow^*_{\psi_1}\psi_2$; the others can be obtained likewise.
$\downarrow^*_{\psi_1}\psi_2, leaf \in h(n) \stackrel{\text{atom}}{\Longrightarrow}$
$\psi_1, leaf \in h(n)$ or $\psi_2, leaf \in h(n) \stackrel{\mathcal{R}_\downarrow}{\Longrightarrow}$
$\psi_2, leaf \in h(n)$ or $\psi_1, \neg\psi_2, leaf, \neg leaf \in h(n) \stackrel{\text{atom}}{\Longrightarrow}$
$\psi_2, leaf \in h(n)$ □

The leaf and the branching consequences can be used to construct sets of formulae that hold for a node (and its successors), if we apply them repeatedly to a given set of formulae. This leads to their closure

Definition 4.6.5 (Branching Closure). *We call a set A of formulae consistent if $\varphi \in A$ implies $\neg\varphi \notin A$. We define the branching consequence relation $\vdash_\triangle$ as follows:*

$$A \to BC \vdash_\triangle A' \to B'C'$$

iff there is some $A_0 \to B_0C_0 \in Cn_\triangle(A)$ such that $A' = A \cup A_0$, $B' = B \cup B_0$, $C' = C \cup C_0$ and A', B', C' are consistent. Then

$$A \to BC \vdash^{\max}_\triangle A' \to B'C'$$

iff $A \to BC \vdash^*_\triangle A' \to B'C'$ and $A' \to B'C'$ is maximal, i.e. $A' \to B'C' \vdash_\triangle A' \to B'C'$. Then

$$Cn^*_\triangle(A) := \{A' \to B'C' \mid A \to \{\}\{\} \vdash^{\max}_\triangle A' \to B'C'\}$$

denotes the branching closure $Cn^*_\triangle(A)$ for a set of formulae A.

Note that the members of a $Cn^*_\triangle(A)$ are fixed-points of $\vdash_\triangle$. Analogously, we define the leaf closure:

Definition 4.6.6 (Leaf Closure). *We define the leaf consequence relation $\vdash_\#$ as follows:*

$$A \vdash_\triangle A'$$

iff there is some $A_0 \in Cn_\triangle(A)$ such that $A' = A \cup A_0$ and A is consistent. Then

$$A \vdash^{\max}_\triangle A'$$

iff $A \vdash^*_\triangle A'$ and A' is maximal, i.e. $A' \vdash_\triangle A'$. Then

$$Cn^*_\#(A) := \{A' \mid A \vdash^{\max}_\triangle A'\}$$

denotes the branching closure $Cn^*_\#(A)$ for a set of formulae A.

Now we call the union of the branching and the leaf closure the *local closure*. Like atoms, the closure specifies sets of consistent formulae, that describe certain kinds of tree nodes. However, unlike atoms that consider all combinations of subformulae, the local closure only includes the formula, that are derived from the underlying formula. Consequently, the size and the number of local closures should be less than the number of corresponding atoms. Moreover, the local closure can be employed to construct tree models of a given formula.

Definition 4.6.7 ($Cn^*(\varphi)$-constructed tree). *Let $T = (d,h)$ be a finte $2^{Cl(\varphi)}$-labeled binary tree. We call T a $Cn^*(\varphi)$-constructed tree, if it is constructed in the following way:*

1. *Root: The root has the initial label $\{\varphi\}$*
2. *Leaf nodes: A leaf with the initial label A_0, obtains a final label $A \in Cn^*_\#(A_0)$*
3. *Branching nodes: A branching with the initial label A_0, obtains a final label A and the two successor obtain the initial labels B_1 and B_2, respectively, for some $(A \to B_1B_2) \in Cn^*_\triangle(A_0)$.*

Obviously, this construction is sound, i.e. the tree nodes satisfy the formulae obtained from their label:

Lemma 4.6.8. *Let $T = (d,h)$ be $Cn^*(\varphi)$-constructed tree. Then $T, n \models h(n)$ for every node $n \in d$.*

Proof. This follows immediately from the definition of the local closure and the semantics of BTL. □

Finally, we obtain the other direction:

Theorem 4.6.9. *A BTL-formula φ is finitely satisfiable, iff there is a $Cn^*(\varphi)$-constructed tree.*

Sketch. $\Leftarrow$ follows from lemma 4.6.8. $\Rightarrow$ By theorem 4.4.8 it is sufficient to show that every finite pretree $T_p = (d,h_p)$ for $At(\varphi)$ corresponds to some $Cn^*(\varphi)$-constructed tree$T_c = (d,h_c)$ such that $h_c(n) \subseteq h_p(n)$ for each $n \in d$. Obviously, by lemma 4.6.2 every local consequence contributes to the construction of atoms, i.e if Φ_c are local consequences of Φ, then $\Phi \cup \Phi_c \subseteq A$ for some $A \in At(\varphi)$. Moreover, the branching consequences of a node do not violate the reachability relation. Consequently for $A \to B_1B_2$ of branching closure there are atoms A', B'_1, B'_2 with $A \subset A'$, $B_1 \subset B'_1$, $B_2 \subset B'_2$ and $(A', B'_1, B'_2) \in \widehat{R_\varphi}$. Similarly, by lemma 4.6.4, every leaf closure must have a corresponding atom. □

Thus, we obtained the completeness of the $Cn^*(\varphi)$-construction method for finite trees, i.e. a finite tree is a model of φ, if and only if, it can be $Cn^*(\varphi)$-constructed. Now we illustrated, how the local closure can be employed for top-down parser. For example we consider the parsing operations of an Earley parser (Earley, 1970). Typically, a top-down parsing algorithm consults the underlying context-free grammar, if it processes parsing items of the shape $A \to \alpha \bullet B\beta$, where $A, B \in N$ are nonterminal symbols, $\alpha, \beta \in (N \cup T)*$ are strings. In this situation, the parser must expand the symbol B, i.e. we must determine all rules having B at their left-hand side. Obviously, in the context of dynamic rule generation, the branching closure of B means exactly that set. A similar situation occurs for shift operation, which corresponds to parsing items of the shape $A \to \alpha \bullet x\beta$, where $x \in T$ is terminal symbol. However, the situation is quite different for parsing with BTL-formula. In this case x corresponds

to a set of formula and we must determine the leaf closure of x. Then we could read the corresponding input symbol w, if the atomic propositions of w matches the ones of x, i.e. $x \cup w$ must be consistent.

Now let's turn to the issue of complexity. Obviously, the technique of determining the branching and leaf closure avoids constructing the whole set of atoms, as discussed in the previous section. Unfortunately, we generally cannot exclude the worst cases, neither that the local closure consist of atoms (instead of proper subsets), nor that all rules must be computed. For instance consider the formula $\varphi = \downarrow_{\top}^{*} \neg(a \wedge b)$ which has the leaf and branching closure:

$$
\begin{array}{l}
\{\ \varphi,\ \mathit{leaf}, \neg(a \wedge b),\ a, \neg b\ \} \\
\{\ \varphi,\ \mathit{leaf}, \neg(a \wedge b), \neg a,\ b\ \} \\
\{\ \varphi,\ \mathit{leaf}, \neg(a \wedge b), \neg a, \neg b\ \} \\
\{\neg\varphi,\ \mathit{leaf},\ (a \wedge b),\ a,\ b\ \} \\
\{\neg\varphi, \neg\mathit{leaf},\ (a \wedge b),\ a,\ b\ \} \rightarrow \{\}\,\{\} \\
\{\ \varphi, \neg\mathit{leaf},\ (a \wedge b),\ a,\ b\ \} \rightarrow \{\varphi\}\,\{\} \\
\{\ \varphi, \neg\mathit{leaf},\ (a \wedge b),\ a,\ b\ \} \rightarrow \{\}\,\{\varphi\} \\
\{\ \varphi, \neg\mathit{leaf}, \neg(a \wedge b),\ a, \neg b\ \} \rightarrow \{\}\,\{\} \\
\{\ \varphi, \neg\mathit{leaf}, \neg(a \wedge b), \neg a,\ b\ \} \rightarrow \{\}\,\{\} \\
\{\ \varphi, \neg\mathit{leaf}, \neg(a \wedge b), \neg a, \neg b\ \} \rightarrow \{\}\,\{\}
\end{array}
$$

Obviously, each set on the left-hand side is an atom and there are no further atoms. Moreover, we also obtain nearly all rules except the general rule $\{\} \rightarrow \{\}\,\{\}$ which is the branching consequence of the empty set. Nevertheless, the reachability relation $\mathcal{R}_\varphi$ has even $4*9*9+2*9*7 = 450$ elements, which can be obtained by employing the local closure to the sets of the right-hand side. Thus, the complexity once more has an exponential upper bound at $O(2^{|\varphi|})$. As we have already seen, it is difficult to avoid the bad case for the translation method. However, this does not apply to the local closure. Once more, we consider the characteristic BTL-formula $\chi(G)$ for an arbitrary context-free grammar.

Proposition 4.6.10. *We define the transitive closure of the local closure written $TC_\triangle(\varphi)$ as follows:*

1. *If $(A \rightarrow B_1 B_2) \in Cn^*_\triangle(\varphi)$ then $(A \rightarrow B_1 B_2) \in TC_\triangle(\varphi)$.*
2. *If $(A \rightarrow B_1 B_2) \in TC_\triangle(\varphi)$ and $(C \rightarrow D_1 D_2) \in Cn^*_\triangle(B_1) \cup Cn^*_\delta$ then $(C \rightarrow D_1 D_2)' \in TC_\triangle(\varphi)$.*
3. *Nothing else is in $TC_\triangle(\varphi)$.*

Then transitive closure of the local closure of a characteristic CFG-formula (see Definition 4.5.3) $\chi(G)$ is of linear size, i.e $|TC_\triangle(\chi(G))|$ is in $O(|\chi(G)|)$, where G denotes a binary CFG G.

Proof. Consider (a compact representation of) the local closure of each partial constraint:

$$
\begin{array}{l}
Cn^*_\triangle(\chi_S) = \{\{S\} \rightarrow \{\}\,\{\}\} \\
Cn^*_\triangle(\chi_\Sigma) = \{\{\dot{X}, \chi_\Sigma\} \rightarrow \{\chi_\Sigma\}\,\{\chi_\Sigma\} \mid X \in \mathcal{P}\ \} \\
Cn^*_\triangle(\chi_P) = \{\{A, \chi_P\} \rightarrow \{B, \chi_P\}\{C, \chi_P\} \mid A \rightarrow BC \in P\} \\
Cn^*_\triangle(\chi_L) = \{\{X, \chi_L\} \rightarrow \{\chi_L\}\,\{\chi_L\} \mid X \in N\}
\end{array}
$$

where $\dot{X}$ abbreviates the constraints, which ensure that the $X \in \mathcal{P}$ exclusively occurs at a node, i.e.$\{\dot{X}\} = \{X\} \cup \{\neg Y \mid Y \in \mathcal{P} \setminus \{X\}\}$. Now we conjunctively combine the constraints χ_Σ, χ_P, χ_L to a single constraint χ' which leads to:

$$
\begin{array}{c}
Cn^*_\triangle(\chi') = \\
\{\{\dot{A}, \chi'\} \rightarrow \{\dot{B}, \chi'\}\,\{\dot{C}, \chi'\} \mid A \rightarrow BC \in P, A \in N\}
\end{array}
$$

Now obtain the first member of the transitive closure:

$$
\begin{array}{c}
Cn^*_\triangle(\chi_S, \chi') = \\
\{\{\dot{S}, \chi'\} \rightarrow \{B, \chi'\}\,\{C, \chi'\} \mid S \rightarrow BC \in P, S \in N\}
\end{array}
$$

Then we recursively determine the local closure to the set of formulae arising at the right hand side. Finally, we obtain the transitive closure of the local closure of $\chi(G)$:

$$
\begin{array}{c}
TC_\triangle(\chi(G)) = \\
\{\{\dot{A}, \chi'\} \rightarrow \{\dot{B}, \chi'\}\,\{\dot{C}, \chi'\} \mid A \rightarrow BC \in P_S\}
\end{array}
$$

where P_S denotes the set rules in P that are reachable from S. Thus, we have $|TC_\triangle(\chi(G))| \leq |P|$ □

The result of this proposition is quite remarkable. Given a particular shape of the underlying BTL-formula φ, it is possible to construct a corresponding context grammar G_φ, such that the size of G linearly depends on the size of φ, i.e the parsing problem has the complexity $O(|\varphi|^2|w|^3)$ in this case.

4.7 Restricted Formulae

Finally, we now turn to an interesting sublanguage BTL_S. We will show that the complexity does not only depend on the size of φ, but also on the shape of φ. One basic step for reducing the complexity concerns the atomic propositions $p, q, \ldots \in \mathcal{P}$. In the sequel, the atomic propositions are assumed to be exhaustive and mutually exclusive, which exactly corresponds to the exclusive symbol constraint of $\chi(G)$ in definition 4.5.3. Obviously, this restriction does not change the expressive power of BTL, since we simply obtain such a set of exclusive labels by assuming $\mathcal{P} = 2^{\mathcal{P}'}$ for a given set $\mathcal{P}'$ of atomic propositions. For the further formulae, we often employ disjunctions of atomic propositions, written $\vee Q$ for some $Q \subseteq \mathcal{P}$. Furthermore, we associate each atomic proposition with a particular structural constraints. Let $\mathcal{P}$ be a set of mutual exclusive atomic propositions. Then an BTL_S formula consists of a conjunction of the following constraints:

(i) $\vee Q$ (initial symbol constraint)

(ii) $\Downarrow^* (\vee Q)$ (global symbol constraint)

(iii) $\Downarrow^* (\mathit{leaf} \rightarrow \vee Q)$ (leaf symbol constraint)

(iv) $\Downarrow^* (p \rightarrow \vee_{q_1,q_2} \downarrow_1 q_1 \wedge \downarrow_2 q_2)$ (successor const.)

(v) $\Downarrow^* (p \rightarrow \neg(\downarrow^*_{\vee Q_1} \neg(\vee Q_2)))$ (global path const.)

where $Q \subseteq \mathcal{P}$ and $p, q_1, q_2 \in \mathcal{P}$. Except the last one, these kinds of constraints clearly resemble the characteristic formula of context-free grammars. Note that the global path constraints are particular constraints. They state that every path of Q'_p symbols starting at some nod *n* must end with a Q_p. Then we have:

Theorem 4.7.1. *The parsing problem for BTL$_S$ has polynomial time complexity.*

Proof. Besides the global path constraints any BTL$_S$ formula corresponds to a characteristic CFG-formula. Therefore we have already seen the linear complexity for this part of φ. It remains to show that the global path constraints keep this property. The local closure of a global path constraint $\Downarrow^* (p \rightarrow \neg(\downarrow^*_{\vee Q_1} \neg(\vee Q_2)))$ yields the following rules where $\psi_U = \neg(\neg((\downarrow^*_{\vee Q_1} \neg(\vee Q_2)))$:

$$\begin{array}{rcl} \{\psi, \vee\overline{P}\} & \rightarrow & \{\psi\}\{\psi\} \text{ where } \overline{P} = (P \setminus \{p\}) \\ \{\psi, p\} & \rightarrow & \{\psi\}\{\psi\} \text{ iff } p \notin Q_1 \\ \{\psi, p\} & \rightarrow & \{\psi, \psi_U)\}\{\psi, \psi_U\} \text{ iff } p \in Q_1 \cap Q_2 \end{array}$$

Then we consider transitive closure of the local closure :

$$\begin{array}{l} \{\psi,\psi_U,\mathcal{P} \setminus Q_2\} \rightarrow \{\psi,\psi_U\}\{\psi,\psi_U\} \\ \{\psi,\psi_U,Q_1 \cap Q_2\} \rightarrow \{\psi,\psi_U,Q_2 \setminus Q_1\}\{\psi,\psi_U,Q_2 \setminus Q_1\} \\ \{\psi,\psi_U,Q_1 \cap Q_2\} \rightarrow \{\psi,\psi_U,Q_2 \setminus Q_1\}\{\psi,\psi_U,Q_2 \cap Q_1\} \\ \{\psi,\psi_U,Q_1 \cap Q_2\} \rightarrow \{\psi,\psi_U,Q_2 \cap Q_1\}\{\psi,\psi_U,Q_2 \setminus Q_1\} \\ \{\psi,\psi_U,Q_1 \cap Q_2\} \rightarrow \{\psi,\psi_U,Q_2 \cap Q_1\}\{\psi,\psi_U,Q_2 \cap Q_1\} \\ \{\psi,\psi_U,Q_2 \setminus Q_1\} \rightarrow \{\psi,\psi_U\}\{\psi,\psi_U\} \end{array}$$

Obviously, these rules mutually exclude each other. Consequently, every rule of the context-free part of φ matches at most one of these rules. Therefore, when we intersect the CFG-rules and the rules of the global path constraints, the resulting number of rules does not increase, but maybe, it is smaller. □

As we have seen above, we can obtain polynomial parsing complexity, if we restrict the formula in a particular manner. However, the restrictions of BTL$_S$ are rather strong, since they exclude existential constraints and negations. On the other hand, it is possible to encode such constraints by introducing special symbols, that are exclusively used for these constraints. Although this would increase the number symbols, the resulting effects on the parsing complexity are not so bad – compared with BTL, in general. The size of a binary CFG lies in $O(|\mathcal{P}|^3)$, therefore the parsing complexity is bound by $O(|\mathcal{P}|^6)$.

By using of global path constraints, it is possible to formulate constraints more outside the context-free paradigm. As mentioned in the proof, the corresponding triples of classes of atoms mutually exclude themselves. If we intersect these triples with the ones for the successor constraints, the resulting set of triples is not greater than greatest set of underlying triples, since one global constraint match most one successor constraint. Therefore it should be possible, to introduce further kinds of global constraints without forcing an exponential blow-up, unless the corresponding triples of classes of atoms do not mutually exclude themselves. If this is true, the resulting set of triples is always bound by the maximum of the cardinalities of the underlying sets.

4.8 Conclusion

In a model theoretic framework, the parsing problem agrees with the satisfiability problem of the underlying logical formalism. Consequently, for each formalism on trees, the complexity of the parsing problem and of the satisfiable problem are equal, in general. Therefore the parsing problem for the modal tree language BTL is EXPTIME complete. However, this is only a worst case estimation. As we have demonstrated, this is not a general limitation. Obviously, the complexity depends on the underlying formula and the way we the corresponding grammar rules, that are necessary for the parsing process. As we have seen the standard model construction based on atoms and the reachability relation, fails to find an optimal solution for the characteristic BTL formula of a context-free grammar. In contrast, the local closure method leads more or less to the underlying context-free grammar in that case. Thus, polynomial time parsing is absolutely possible. Finally, we presented a restricted version of BTL, where the parsing complexity always linearly depends on the length of the underlying formula.

Bibliography

Blackburn, P., B. Gaiffe, and M. Marx (2003). Variable free reasoning on finite trees. In R.T. Oehrle and J. Rogers, eds., *Proceedings of Mathematics of Language MOL 8*. Bloomington, Indiana, USA.

Blackburn, P. and W. Meyer-Viol (1994). Linguistics, logic and finite trees. *Bulletin of the IGPL*, **2**:2–29.

Blackburn, P. and W. Meyer-Viol (1997). Modal logic and model-theoretic syntax. In M. de Rijke, ed., *Advances in Intensional Logic*, pp. 27–58. Kluwer, Dordrecht.

Blackburn, P., W. Meyer-Viol, and M. de Rijke (1996). A proof system for finite trees. In H. Kleine Büning, ed., *Computer Science Logic*, LNCS, vol. 1092, pp. 86–105. Springer Verlag.

Cornell, T. (2000). Parsing and grammar engineering with tree automata. In D. Heylen, A. Nijholt, and G. Scollo, eds., *Algebraic Methods in Language Processing AMoLP 2000*, pp. 267–274. Iowa City, Iowa.

Earley, R. (1970). An efficient context-free parsing algorithm. *Communications of the ACM*, **13**.

Emerson, E.A. and A.P. Sistla (1984). Deciding branching time logic. *Information and Control*, **61**(2):175–201.

Fischer, M.J. and R.F. Ladner (1979). Propositional dynamic logic of regular programs. *Journal of Computer and System Science*, **18**:194–211.

Gabbay, D. and F. Guenthner, eds. (1984). *Handbook of Philosophical Logic vol. 2*. Reidel, Dordrecht.

Hemaspaandra, E. (1996). The price of universality. *Notre Dame Journal of Formal Logic*, **37**(2):174–203.

Kamp, J.A.W. (1968). *Tense Logic and the Theory of Linear Order*. Ph.D. thesis, University of Califormia, Los Angeles.

Kozen, D. and J. Tiuryn (1990). The logic of programs. In J. van Leeuwen, ed., *Handbook of Theoretical Computer Science*, pp. 789–840. Elsevier Science Publisher, Amsterdam, The Netherlands.

Kracht, M. (1995). Syntactic codes and grammar refinement. *Journal of Logic, Language and Information*, **4**:41–60.

Lee, L. (2002). Fast context-free grammar parsing requires fast boolean matrix multiplication. *Journal of the ACM*, **49**(1):1–15.

Lichtenstein, O. and A. Pnueli (2000). Propositional temporal logics: Decidability and completeness. *Logic Journal of the IGPL*, **8**(1):55–85.

Palm, A. (1997). *Transforming Tree Constraints into Rules of Grammars*, volume 173 of *DISKI*. infix-Verlag, St. Augustin.

Palm, A. (1999a). The expressivity of tree languages for syntactic structures. In H.-P. Kolb and U. Mönnich, eds., *The Mathemathics of Syntactic Strutures*, pp. 50–90. De Gruyter, Berlin.

Palm, A. (1999b). Propositional tense logic for finite trees. In *Proceedings of 6th Meeting on Mathemathics of Language (MOL6)*, pp. 285–300. Orlando, Florida.

Palm, A. (2001). Model theoretic syntax and parsing: An application to temporal logic. In G.-J. Kruijff, L. Moss, and R. Oehrle, eds., *Proceedings of FGMOL'01*, volume 53 of *ENTCS*. Elsevier, Helsinki, Finland.

Reynolds, M. (2003). The complexity of the temporal logic with "until" over general linear time. *Journal of Computer and System Science*, **66**(2):393–426.

Rogers, J. (1998). *A Descriptive Approach to Language-Theoretic Complexity*. CSLI Publication, Stanford, California.

Schlingloff, B.-H. (1990). *Zur temporalen Logik von Bäumen*. Ph.D. thesis, Technische Universität München, Munich.

Sistla, A.P. and E.M. Clarke (1985). The complexity of propositional linear temporal logic. *Journal of the ACM*, **32**(3):733–749.

Spaan, E. (1993). *The Complexity of Modal Logic*. Ph.D. thesis, University of Amsterdam, Amsterdam.

van Benthem, J. (1984). Correspondence theory. In Gabbay and Guenthner (1984), pp. 167–247.

Younger, D. (1967). Recognition and parsing of context-free languages in time n^3. *Information and Control*, **10**:189–208.

Chapter 5

Operations on polyadic structures

Anders Søgaard
Center for Language Technology
Njalsgade 80
DK-2300 Copenhagen
anders@cst.dk

abstract

Logical translations for simple attribute-value grammars are numerous (Blackburn, 1994). Only a few exist for more complex attribute-value grammars. In particular, lists and set values and manipulation of such data structures seem difficult to capture in logical languages of reasonable expressivity. A new class of logical languages is presented here to meet this need, namely *polyadic propositional dynamic logics* (PPDLs); the properties of these languages are compared to $\mathcal{L}^{++}$ (Reape, 1992) and RSRL (Richter, 2004), the two most cited alternatives.

5.1 Introduction

In model-theoretic syntax, translations into logical languages are not only used to evaluate linguistic theories, but the logical translations *are* in a sense the linguistic theories. Grammars are logical systems, and their denotation is often said to be all the strings whose logical descriptions are satisfiable in conjunction with the axioms.[1] The derivation structure of a string is the model that satisfies this conjunction. It seems natural thus to expect complete axiomatizations from model-theoretic syntacticians. Such expectations are in general not met, apart from a few, notable exceptions. In fact, an exhaustive translation is rarely found. Usually model-theoretic syntacticians only provide translation sketches. Since the complex attribute-value grammars that are in our focus, are not exhaustively specified in the first place, a translation sketch is all that can be expected. It is, of course, possible to provide an exhaustive translation of a particular grammar.

It was mentioned in the abstract that our objective is the presentation of a new class of logical languages, suitable for the translation of complex attribute-value grammars. In particular, these grammars are equipped with lists and set values and operations for manipulation of such data structures. In the literature, two languages have been suggested for such translations: $\mathcal{L}^{++}$ (Reape, 1992) and RSRL (Richter, 2004). Our focus will be on the comparison of our new class of languages to these two existing ones, rather than on the translatability of complex attribute-value grammars. Of course, a few examples are included for illustration. Our new class of languages will be argued to have three advantages compared to $\mathcal{L}^{++}$ and RSRL: It is less complex, more modular, and more intuitive. In some respects, it also has descriptive advantages.

5.1.1 $\mathcal{L}^{++}$ and RSRL

The formal introduction of $\mathcal{L}^{++}$ and RSRL is omitted in this abstract for brevity. If the reader has no prior knowledge of these two logical languages, she is urged to consult the references cited above. Otherwise she can think of $\mathcal{L}^{++}$ as a polyadic, deterministic version of quantified hybrid logic (H($\exists$)) (Blackburn and Seligman, 1995); while RSRL is a bit more unusual. The RSRL expressions come in four flavors: as variables, path-variables, formulas and components ($\mathbb{VAR}, \mathbb{PVAR}, \mathbb{FORM}, \mathbb{COMP}$) It may be useful to also think of a typed feature structure grammar over a subsumption hierarchy $\langle \mathsf{Types}, \sqsubseteq \rangle$ and some set of attributes Attr. The syntax is now defined as:

$$\begin{array}{lcl} \pi \in \mathbb{PVAR} & \doteq & x|a\pi \\ \phi \in \mathbb{FORM} & \doteq & \pi \sim \sigma|\pi \approx \pi'|\phi \wedge \psi|\neg\phi| \\ & & R(x_1,\ldots,x_{\rho(R)})|\exists x.\phi \end{array}$$

where $x \in \mathbb{VAR}, a \in \mathsf{Attr}$ and $\sigma \in \mathsf{Types}$. $\mathbb{COMP}$ is a set used to restrict quantification. The idea resembles the master modality of extended modal logic. $\mathbb{COMP}$ is defined relative to every individual u in the universe, and $\mathbb{COMP}^u$ is $\{u' \in \mathfrak{U}|$ there is a path from u to $u'\}$. Finally, $\sim$ is for type assignment and $\approx$ is for path equation. RSRL also introduces a special machinery for *chains*, a kind of virtual list that can be manipulated in the scope of negation. Chains are not introduced formally here, but it is important to understand the importance of this kind of virtual manipulation. See Richter (2004) for linguistic motivation. The new class of languages introduced later on has the expressive power to perform such manipulation.

Call the chain-free fragment of RSRL for c-RSRL. c-RSRL translates into HL($\exists$)$^{\mathsf{Reape}}$ with a reflexive, transitive modality $\langle r \rangle$ (satisfying KT4). The translation, which

[1] Other views exist. Two of them are just as influential as the one just presented; namely, saying that the language of a grammar is the set of models that satisfy the grammar, or saying that the language is the yield of the set of derivation structures that are in the exhaustive model of the grammar. The differences are ignored here; they do not influence our results.

relies on a specific axiomatization where output arguments, e.g. concatenated lists, are extracted, is defined by:

$$\begin{aligned}
\mathrm{ST}(x) &= x \\
\mathrm{ST}(a\pi) &= \langle a\rangle \mathrm{ST}(\pi) \\
\mathrm{ST}(x \sim \sigma) &= \sigma \\
\mathrm{ST}(\pi \approx \pi') &= \exists x.\mathrm{ST}(\pi)x \wedge \mathrm{ST}(\pi')x \\
\mathrm{ST}(\phi \wedge \psi) &= \mathrm{ST}(\phi) \wedge \mathrm{ST}(\psi) \\
\mathrm{ST}(\neg\phi) &= \neg\mathrm{ST}(\phi) \\
\mathrm{ST}(R_a(x_1, x_2 \ldots)) &= @_{x_1}\langle a\rangle(x_2, \ldots) \\
\mathrm{ST}(\exists x.\phi) &= \exists x.\langle r\rangle(x \wedge \mathrm{ST}(\phi))
\end{aligned}$$

Remark 5.1.1. Satisfiability is undecidable for both $\mathcal{L}^{++}$ and c-RSRL (and therefore RSRL). Consider, for instance, what it takes to code up the tiling problem. You need to ensure sideways and downwards matches, but this is trivial. You need to ensure that each field in the plane is tiled by a unique tile type, and you need to enforce grids. This is easy once you have quantification, reentrancy (nominals or path equation) and deterministic relations. See e.g. Blackburn et al. (2001, 366).

Remark 5.1.2. Model checking RSRL was proven to be undecidable by Kepser (2001) by reduction of Post systems. It is easy to see that model checking $\mathcal{L}^{++}$ or c-RSRL is PSPACE-hard, e.g. by reduction of the Geography problem. The rules of the Geography game are: P_1 first selects a vertex v; P_2 then selects a vertex v' such that $(v,v') \in E$; P_1 selects a vertex v'' such that $(v',v'') \in E$; and so on; and the player who chooses a vertex that has been visited before loses. It can be decided if P_1 has a winning strategy by evaluating at most n formulas of the type (in $\mathcal{L}^{++}$ syntax):

$$\exists x_1.\forall x_2.\exists x_3.\forall x_4.\ldots.\exists x_{n-1}.\forall x_n.x_1 \wedge \langle e\rangle(x_2 \wedge \langle e\rangle(x_3 \wedge \langle e\rangle(x_4 \wedge \ldots \wedge \langle e\rangle(x_{n-1} \wedge \langle e\rangle x_n)))) \wedge x_1 \neq x_2 \neq x_3 \neq x_4 \neq \ldots \neq x_{n-1} \neq x_n$$

PSPACE-completeness of $\mathcal{L}^{++}$ follows from an extension of the model checking procedure in Franceschet and de Rijke (2005), and PSPACE-completeness of c-RSRL from the translation above into $\mathcal{L}^{++}$.

The main motivations for the logics introduced below are complexity and modularity. In particular, there exist polynomial time model checking procedures for the relevant logics. The logics also have important expressive advantages. It is wellknown that the expressive power of first order logic is limited by the lack of a mechanism for unbounded iteration or recursion. The most notable example of a query that is not expressible in first order logic, is the transitive closure of a relation. Neither $\mathcal{L}^{++}$ nor RSRL improves on this.

Remark 5.1.3. It is possible to add a least fixpoint operator to both logics. Finite model checking will still be polynomial in the size of the model and PSPACE-complete in the length of the formula.

5.1.2 PPDLs

The new class of logical languages, *polyadic propositional dynamic logics* (PPDLs), includes languages for which model checking is in P. Some languages have PSPACE-hard model checking. In particular, we single out three languages here, PPDL^k, PPDL$^{(*)}$ and PPDL$^{\circledast}$. Complexity upper bound results for model checking are summarized in the table below:

PPDL^k	P
PPDL$^{(*)}$	P
PPDL$^{\circledast}$	EXP

In other words, PPDL$^{\circledast}$ is in EXP and PSPACE-hard. All three languages have the expressivity to manipulate lists and set values in the scope of negation, but only PPDL$^{\circledast}$ includes lists and set values in the quantification domain. Since RSRL includes lists and set values in the quantification domain, this may turn out to be relevant. Model checking PPDL$^{\circledast}$ is still significantly less complex than model checking RSRL. In a sense, you can say you get chains for free, since the complexity of PPDL$^{\circledast}$ is comparable to c-RSRL.

There is another advantage to PPDLs, namely their modularity. PPDLs is a class of propositional languages. Lists and set values are treated as first-class citizens, and operations are introduced directly in the syntax. Compare this to $\mathcal{L}^{++}$ and RSRL where lists and sets are decomposed, though sets are treated differently in the two logical languages, and first order axiomatizations of the operations are introduced. The move to first order introduces the expressive power to encode all these operations. In PPDLs, operations are introduced separately. The third claim, that PPDLs are more intuitive than its alternatives, is related to this modularity. The first order axiomatizations rely on various relations that complicate the syntax and make linguistic descriptions less compact. For this reason, most linguistic work in $\mathcal{L}^{++}$ and RSRL rely on convenient abbreviations of the syntax.

PPDL^k and PPDL$^{(*)}$ rely on a common language which we call PPDL* here.

Definition 5.1.4. Formulas and programs of PPDL* over a signature $\langle \mathsf{Labels}, \mathbb{PRROP}\rangle$ are defined as:

$$\begin{aligned}
\phi &\doteq p \mid \bot \mid \phi \wedge \psi \mid \neg\phi \mid \langle\alpha\rangle(\phi_1, \ldots, \phi_n) \\
\alpha &\doteq \varepsilon \mid a \mid \alpha;\beta \mid \alpha^* \mid \alpha \cup \beta \mid \alpha \cap \beta \mid \\
&\quad\ \alpha \sqcup \beta \mid \alpha \sqcap \beta \mid \oplus(\alpha_1, \alpha_2, \alpha_3, \alpha_4)
\end{aligned}$$

where $a \in \mathsf{Labels}$, $\alpha \in \mathsf{Programs}$ and $p \in \mathbb{PRROP}$. In addition, the formula $\langle\alpha\rangle(\phi_1, \ldots, \phi_l)$ is wellformed iff $l = \rho(\alpha) - 1$ (or $\rho(\alpha, l+1) = \top$, really, since relations are dynamic). The program α is itself wellformed iff the statement $\rho(\alpha) \vdash \alpha$ can be derived by the rules:

$$\frac{}{l \vdash \varepsilon} \qquad \frac{}{\rho(a) \vdash a} \qquad \frac{2 \vdash \alpha \quad l \vdash \beta}{l \vdash \alpha;\beta}$$

$$\frac{2 \vdash \alpha}{2 \vdash \alpha^*} \qquad \frac{l \vdash \alpha \quad l \vdash \beta}{l \vdash \alpha \cup \beta} \qquad \frac{l \vdash \alpha \quad l \vdash \beta}{l \vdash \alpha \cap \beta}$$

$$\frac{l_1 \vdash \alpha \quad l_2 \vdash \beta}{2 \vdash \alpha \sqcup \beta} \qquad \frac{l_1 \vdash \alpha \quad l_2 \vdash \beta}{2 \vdash \alpha \sqcap \beta}$$

$$\frac{l_1 \vdash \alpha_1 \quad l_2 \vdash \alpha_2 \quad l_3 \vdash \alpha_3 \quad l_4 \vdash \alpha_4}{l_1 \times l_2 + l_3 \times l_4 \vdash \oplus(\alpha_1, \alpha_2, \alpha_3, \alpha_4)}$$

If a program has arity 2, the corresponding relation is binary. In ordinary propositional dynamic logic (Fischer and Ladner, 1979), all programs have arity 2. The reader should note that the first argument of composition (;) is always binary. A truly polyadic composition operator is introduced in PPDL$^{\circledast}$ below. PPDL$^{\circledast}$ also introduces a polyadic iteration operator. The calculus above restricts ordinary iteration (*) to binary relations.

Now for the semantics. The satisfaction definitions are standard (similar to the modal core of $\mathcal{L}^{++}$). Each program α induces a relation R_α of arity $\rho(\alpha)$ over a polyadic Kripke structure that is inductively defined in Figure 5.1.

$;, *, \cup, \cap$ are standard. Intuitively, ; builds paths out of programs, $*$ iterates over programs, $\cup$ takes the union of relations (useful for disjunctive paths), and $\cap$ takes their intersection. $\sqcup$ and $\sqcap$ are union-in-a-point and intersection-in-a-point. The $\oplus$ operator is for concatenation of lists. $\sqcap$ and $\cap$ encodes set membership. You can think of Labels as Attr, linguistically, and of $\mathbb{PROP}$ as Types for some attribute-value grammar over a signature $\langle\langle \mathsf{Types}, \sqsubseteq\rangle, \mathsf{Attr}\rangle$.

Example 5.1.5. Consider, for illustration, the attribute-value structure:

$$\begin{bmatrix} \mathrm{F}_1 \left[\mathrm{F}_2 \left\langle \mathrm{a,b,c} \right\rangle \right] \\ \mathrm{F}_3 \left\langle \mathrm{d,e} \right\rangle \\ \mathrm{F}_4 \left[\mathrm{F}_2 \left\langle \mathrm{a,b,c,d,e} \right\rangle \right] \end{bmatrix}$$

The structure can, with some abuse of terminology, be said to satisfy the formulas:

(a) $\langle \oplus(\mathrm{F}_1, \mathrm{F}_2, \varepsilon, \mathrm{F}_3) \cap \mathrm{F}_4; \mathrm{F}_2 \rangle \top$
(b) $\langle (\mathrm{F}_1; \mathrm{F}_2) \cap (\mathrm{F}_4; \mathrm{F}_2) \rangle \top$

In fact, (b) follows from (a).

Remark 5.1.6. In the linguistic literature, subtraction of lists is sometimes used instead of concatenation of lists. One can easily introduce such an operator, $R_{\ominus(\pi_\alpha, \alpha, \pi_\beta, \beta)}$, defined as:

$$\begin{array}{l} \{\{s, t_1, \ldots, t_n\} \mid \\ \exists(s', t_1, \ldots, t_n, u_1, \ldots, u_m) \in R_\alpha, \\ \exists(s'', u_1, \ldots, u_m) \in R_\beta, \\ \exists(s, \ldots s' \ldots) \in \pi_\alpha, \exists(s, \ldots s'' \ldots) \in \pi_\beta\} \end{array}$$

The possible introduction of $\ominus$ illustrates the modularity of PPDL.

Call PPDL* with a k bound on operator nesting PPDL^k, and PPDL* where $*$ never outscopes another operator except $\cup$ for PPDL$^{(*)}$.

Theorem 5.1.1. *Verification in PPDL^k and PPDL$^{(*)}$ is in P.*

Proof. For all atomic relations R_a, R_b and some world $w \in \mathbb{W}$, $\rho(a) = l$, $\rho(b) = m$, $|R_a|_w = n$, $\forall w' \in \mathbb{W}.|R_b|_{w'} = o$, $|R_a| = p$, and $|R_b|_w = q$, where $|R_a|_w = n$ means that there are n many tuples in R_a whose initial argument is w. It follows that

- $|R_\varepsilon|_w = 1$
- $|R_{a \cup b}|_w \leq n + q$
- $|R_{a;b}|_w \leq n \times (l-1) \times o$
- $|R_{a^*}|_w \leq p^2$
- $|R_{a \cap b}|_w \leq \mathbf{min}(n, q)$
- $|R_{a \sqcap b}|_w \leq \mathbf{min}(n \times (l-1), q \times m)$
- $|R_{a \sqcup b}|_w \leq n \times (l-1) + q \times m$
- $|R_{\oplus(\varepsilon, a, \varepsilon, b)}|_w \leq n \times q$

Since frames are deterministic, $n, q = 1$. Since model checking procedures for modal logic have polynomial runtime, and model checking PPDL^k or PPDL$^{(*)}$ now reduces to model checking modal logic, it holds that verification in PPDL^k and PPDL$^{(*)}$ can be solved in polynomial time. □

Definition 5.1.7 (PPDL$^{\circledast}$). Formulas and programs of PPDL$^{\circledast}$ over a signature $\langle \mathsf{Labels}, \mathsf{Atoms} \rangle$ are defined as:

$$\begin{array}{lcl} \phi & \doteq & p \mid \phi \wedge \psi \mid \neg\phi \mid \langle\alpha\rangle(\phi_1, \ldots, \phi_n) \\ \alpha & \doteq & \varepsilon \mid a \mid \alpha;\beta \mid \alpha^* \mid \alpha^{\circledast} \mid \alpha \cup \beta \mid \alpha \cap \beta \mid \\ & & \alpha \sqcup \beta \mid \alpha \sqcap \beta \mid \oplus(\alpha_1, \alpha_2, \alpha_3, \alpha_4) \mid \\ & & \mathbf{un}(\alpha_1, \alpha_2, \alpha_3, \alpha_4) \end{array}$$

The semantics extend the semantics of PPDL* by the clauses in Figure 5.2.

";;" is the abbreviation

$$\alpha;;\beta \;:=\; \mathbf{un}(\varepsilon, \alpha, \alpha, \beta)$$

Note that ";;" is the straight-forward polyadic extension of the binary composition operator ";". $\circledast$ is thus polyadic iteration. Note also that polyadic composition subsumes its binary pendant in the presence of the union-in-a-point-operator: $\alpha;\beta \Leftrightarrow \alpha;;\beta \sqcup \alpha;;\beta$.

Remark 5.1.8. Satisfiability for PPDLs is *highly* undecidable. This can be shown by reduction of recurrent tiling; it is not difficult to see that binary iteration can be used to enforce an infinite number of tiles of a particular type in the first row of the tiling. See, again, Blackburn et al. (2001, 366).

Remark 5.1.9. Model checking PPDL$^{\circledast}$ is PSPACE-hard and in EXP. This can be proven, respectively, by reduction of the problem whether a non-deterministic finite state automaton accepts all permutations of its vocabulary, and by specifying an algorithm for an alternating Turing machine. This result may surprise the reader. The step from a monadic logic to its polyadic

$$
\begin{array}{rcl}
R_{\varepsilon} & \doteq & \{(s,\ldots,s) \mid s \in \mathbb{W}\} \\
R_{\alpha;\beta} & \doteq & \{(s,s') \mid \exists (s,t_1,\ldots,t_n) \in R_\alpha \text{ and } (t_i,s') \in R_\beta \text{ for some } i \in \{1,\ldots,n\}\} \\
R_{\alpha^*} & \doteq & \bigcup_k R_\alpha^k \text{ where } R_{\alpha^0} = \{(s,s) \mid s \in \mathbb{W}\} \text{ and } R_{\alpha^{k+1}} = R_{\alpha;\alpha^k} \\
R_{\alpha\cup\beta} & \doteq & R_\alpha \cup R_\beta \\
R_{\alpha\cap\beta} & \doteq & R_\alpha \cap R_\beta \\
R_{\alpha\sqcup\beta} & \doteq & \{(s,s') | \exists (s,t_1,\ldots,t_n) \in R_\alpha \cup R_\beta, \text{ s.t. } s' = t_i \text{ for some } i \in \{1,\ldots,n\}\} \\
R_{\alpha\sqcap\beta} & \doteq & \{(s,s') | \exists (s,t_1,\ldots,t_n) \in R_\alpha, \exists (s,u_1,\ldots,u_m) \in R_\beta, \exists i,j \text{ s.t. } s' = t_i = u_j\} \\
R_{\oplus(\alpha_1,\alpha_2,\alpha_3,\alpha_4)} & \doteq & \{(x,\overline{y_1},\ldots,\overline{y_n},\overline{z_1},\ldots,\overline{z_n}) | \forall i,j . \exists x', x'' . (x',\overline{y_i}) \in R_{\alpha_2}, (x'',\overline{z_j}) \in R_{\alpha_4}, \\
 & & (x,\ldots x'\ldots) \in R_{\alpha_1}, (x,\ldots x''\ldots) \in R_{\alpha_3}\}
\end{array}
$$

Figure 5.1: Definition of R_α in PPDL*.

$$
\begin{array}{rcl}
R_{\alpha^{\circledast}} & \doteq & \bigcup^k R_\alpha^k \text{ where } R_{\alpha^0} = \{(s,\ldots,s) \mid s \in \mathbb{W}\} \text{ and } R_{\alpha^{k+1}} = R_{\alpha;;\alpha^k} \\
R_{\mathbf{un}(\alpha_1,\alpha_2,\alpha_3,\alpha_4)} & \doteq & \{(x,\overline{y_1},\ldots,\overline{y_n},\overline{z_1},\ldots,\overline{z_n}) | \forall i,j . \exists x', x'' . \overline{y_i} = \{v | (x',\ldots v\ldots) \in R_{\alpha_2}\}, \\
 & & \overline{z_j} = \{v' | (x'',\ldots v'\ldots) \in R_{\alpha_4}\}, (x,\ldots x'\ldots) \in R_{\alpha_1}, (x,\ldots x''\ldots) \in R_{\alpha_3}\}
\end{array}
$$

Figure 5.2: Extension of R_α in PPDL$^{\circledast}$.

bigger sister is usually quite harmless. It should be noted, however, that $\circledast$ has unusual properties. For instance, it is not safe for bisimulation. This follows from the fact that ;; is not safe for bisimulation either. Consider, for instance, the bisimilar models $K_1 = \langle\{1,2,3,4\},\{R_b(1,2),R_a(2,3),R_a(2,4)\},\mathcal{V}_1\rangle$ and $K_2 = \langle\{1,2,3\},\{R_b(1,2),R_a(2,3)\},\mathcal{V}_2\rangle$, where $\mathcal{V}_1(p) = \{2\},\mathcal{V}_1(q) = \{1,3,4\},\mathcal{V}_2(p) = \{2\},\mathcal{V}_2(q) = \{1,3\}$. See that $\langle b;;a\rangle(p,q,q)$ is only true in model K_1, not in K_2. It is also of interest to note that, relative to the proof of Theorem 5.1.1, $|R_{a;;b}| = m!$.

5.1.3 Examples

Example 5.1.10 (Principle A). Say Principle A says that the list that is the value of LIST$_1$ is never the concatenation of the values of LIST$_2$ and LIST$_3$ in the type t. Principles of this kind motivated the introduction of chains in RSRL. In PPDLs, this is easily modelled. Simply say:

$$t \rightarrow [\mathtt{list}_1 \cup \oplus(\varepsilon,\mathtt{list}_2,\varepsilon,\mathtt{list}_3)]\bot$$

Example 5.1.11 (Principle B). Principle B, which was actually proposed by (Pollard and Sag, 1994, 402), says that:

> The CONTEXT.BACKGROUND value of a given phrase is the union of the CONTEXT.BACKGROUND values of the daughters.

The BACKGROUND attribute is supposed to contain information about the (pragmatic) appropriateness conditions associated with an utterance. The details are unimportant. Principle B ensures that the appropriateness conditions of the component parts of a sentence are accumulated, so that there is access to this information at the sentence level. In PPDL$^{\circledast}$, this amounts to:

$$phr \rightarrow \langle \mathtt{ctx;bgr} \cap \mathbf{un}(\mathtt{dtrs},\mathtt{ctx;bgr},\mathtt{dtrs},\mathtt{ctx;bgr})\rangle\top$$

In PPDL*, $\oplus$ is used rather than **un**. It is important to notice that set value effects can be obtained with lists, e.g. $\sqcap$ and $\cap$ is enough to talk about set membership.

5.1.4 Conclusion

A new class of logical languages, *polyadic propositional dynamic logics* (PPDLs), was introduced. These languages are suitable target languages for the translation of complex attribute-value grammars. It was shown that PPDLs are less complex, more modular and more intuitive than $\mathcal{L}^{++}$ and RSRL. In particular, three languages were singled out, PPDL^k, PPDL$^{(*)}$ and PPDL$^{\circledast}$. Model checking PPDL^k or PPDL$^{(*)}$ is in P, while model checking PPDL$^{\circledast}$ is PSPACE-hard (and in EXP).

Bibliography

Blackburn, Patrick (1994). Structures, languages and translations: the structural approach to feature logic. In C. J. Rupp, Rod Johnson, and Michael Rosner, eds., *Constraints, language and computation*, pp. 1–29. Academic Press, London.

Blackburn, Patrick, Maarten de Rijke, and Yde Venema (2001). *Modal logic*. Cambridge University Press, Cambridge, England.

Blackburn, Patrick and Jerry Seligman (1995). Hybrid languages. *Journal of Logic, Language and Information*, **4**:251–272.

Fischer, M. J. and R. E. Ladner (1979). Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, **18**(2):194–211.

Franceschet, Massimo and Maarten de Rijke (2005). Model checking for hybrid logics. *Journal of Applied Logic*, **4**(3):279–304.

Kepser, Stephan (2001). On the complexity of RSRL. *Electronic Notes in Computer Science*, **53**:146–162.

Pollard, Carl and Ivan Sag (1994). *Head-driven phrase structure grammar*. The University of Chicago Press, Chicago, Illinois.

Reape, Mike (1992). *A formal theory of word order: a case study of West Germanic*. Ph.D. thesis, University of Edinburgh, Edinburgh, Scotland.

Richter, Frank (2004). *A mathematical formalism for linguistic theories with an application in head-driven phrase structure grammar*. Ph.D. thesis, Universität Tübingen, Tübingen, Germany.

Chapter 6

Scrambling as the Combination of Relaxed Context-Free Grammars in a Model-Theoretic Grammar Formalism

Ralph Debusmann
Programming Systems Lab
Universität des Saarlandes
Postfach 15 11 50
66041 Saarbrücken, Germany
`rade@ps.uni-sb.de`

6.1 Introduction

After half a decade of work on Model-Theoretic Syntax (MTS), Pullum and Scholz (2001) stated that in a sense, this work had been done in the shadow of Generative-Enumerative Syntax (GES), since it had largely focused on comparing MTS and GES.

Half a decade later, we still observe that the bulk of work has been invested in reformulations of existing GES frameworks in MTS and their comparison. Reformulations of Government and Binding (GB) (Chomsky, 1981) can be found in Rogers (1996, 2003), of Lexical-Functional Grammar (LFG) (Bresnan and Kaplan, 1982) in Blackburn and Gardent (1995), of Generalized Phrase Structure Grammar (GPSG) (Gazdar et al., 1985) in Kracht (1995) and Rogers (1996, 2003), of Head-driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994) in Kepser (2000) and Kepser and Mönnich (2003), and of Tree Adjoining Grammar (TAG) (Joshi, 1987) in Rogers (2003). Comparisons of GB and GPSG can be found in Rogers (1996), and of GB, GPSG and TAG in Rogers (2003).

Recently (Rogers, 2004), there have also been attempts to step out of the shadow of GES, and use MTS not only to reformulate and compare existing GES frameworks, but to utilize the more declarative, clarifying perspective of MTS to also explore *extensions* of them. This is what we set out to do in this paper as well.

Our goal is to reformulate Context-Free Grammar (CFG) in the model-theoretic meta grammar formalism of Extensible Dependency Grammar (XDG) (Debusmann, 2006). We will see that XDG allows us to selectively *relax* constraints such as contiguity, and to *combine* grammars G_1 and G_2, such that the resulting string language is the intersection of those of G_1 and G_2.

After reformulating CFG in XDG, we can immediately explore the relaxation and the intersection of CFGs. We demonstrate this additional expressive power with a simple and elegant account of scrambling loosely based on Topological Dependency Grammar (TDG) (Duchier and Debusmann, 2001).

6.2 Extensible Dependency Grammar

Before we turn to the formalization of XDG, this subsection provides the intuitions and ideas. XDG is a description language for tuples of dependency graphs sharing the same set of nodes, which are anchored by the same string of words. The components of the tuple are called *dimensions*, and XDG analyses *multigraphs*.

Figure 6.1 shows an example multigraph with two dimensions: SYN provides a syntactic, and SEM a semantic analysis in terms of predicate-argument structure. The nodes are identified by indices (1 to 6), and associated with words (e.g. *Mary*, *wants*, etc.). The edge labels on SYN are subj for "subject", vinf for "full infinitive", part for "particle", obj for "object" and adv for "adverb". On SEM, ag stands for "agent", pat for "patient" and th for "theme".

Contrary to other dependency-based grammar formalisms such as Gaifman (1965), XDG dimensions need not be projective trees, but can in fact be general graphs as in Word Grammar (Hudson, 1990). An example is the SEM dimension in Figure 6.1, which is not a tree but a directed acyclic graph (DAG). Here, *to*, which does not have any semantic content, has no ancestor, and *Mary*, which is the agent of both *wants* and *eat*, has two.

Multigraphs are constrained by *grammars* specifying:

1. A *multigraph type* determining the possible dimensions, words, edge labels and additional attributes associated with the nodes called *node attributes*.

2. A *lexicon* determining a subset of the node attributes of each node, depending on the associated word.

3. A set of *principles* stating the well-formedness conditions of the multigraphs.

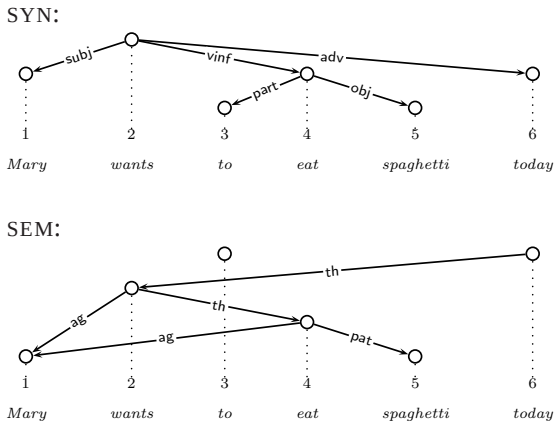


Figure 6.1: XDG multigraph for *Mary wants to eat spaghetti today*.

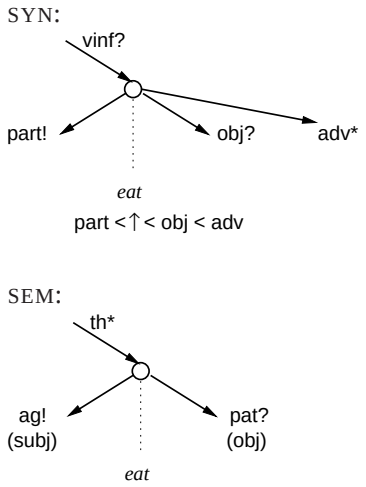


Figure 6.2: Lexical entry for the word *eat*

XDG is a *meta* grammar formalism. *Instances* of XDG are defined by fixing a multigraph type and a set of principles, and leaving the lexicon variable.

XDG comes with a *principle library* of predefined principles, from which instances can be built as with Lego bricks. Principles stipulate e.g. treeness, DAG-ness, projectivity, valency and order constraints. They can also constrain the relation of multiple dimensions, which is used e.g. in the linking principle to constrain the relation between arguments on SEM and their syntactic realization on SYN. Some principles are *lexicalized*, i.e., they constrain the analysis with respect to the lexicon.

The lexicon constrains all dimensions simultaneously, and thereby synchronizes them. Figure 6.2 depicts an example schematic lexical entry for the word *eat*. On SYN, by the lexicalized valency principle, the lexical entry licenses zero or one incoming edges labeled vinf, precisely one part, zero or one obj, and arbitrary many adv dependents, and no other incoming and outgoing edges. By the order principle, the part dependents must precede the head *eat*, which must precede the obj and the adv dependents. On SEM, the lexical entry licenses arbitrary many incoming th edges, and requires precisely one ag and one pat dependent (valency principle). It licenses no other incoming and outgoing edges. The agent is realized by a subject, and the patient by an object (linking principle).

6.2.1 Multigraph

We now turn to the formalization of XDG. Contrary to Debusmann (2006), which is higher-order, our formalization is first-order, and hence called FO XDG. We begin with the notion of multigraphs.

Definition 6.2.1 (Multigraph). *Given a finite set of of dimensions D, a finite set of words W, a finite set of edge labels L, a finite set of attributes A, and a finite set of set types T, a multigraph $M = (V, E^+, <, nw, na)$ consists of a finite set of nodes V, the set of labeled dominances $E^+ \subseteq V \times V \times L \times D$, a total order $< \subseteq V \times V$ on the set of nodes, the node-word mapping $nw \in V \to W$, and the node-attributes mapping $na \in V \to D \to A \to \cup\{ty \mid ty \in T\}$. We define V as a finite interval of the natural numbers starting with 1. $(v, v', l, d) \in E^+$ iff on dimension d, the multigraph contains an edge from v to v'' labeled l, and a path of arbitrary many edges from v'' to v' with any labels.*

The labeled dominance relation corresponds to the transitive closure of the labeled edge relation. The purpose of including this relation and not the labeled edge relation itself is to stay in a first-order logic: if we included only the labeled edge relation instead, we could not express the transitive closure without extending the logic with fixpoints or second-order quantification. This idea is adapted from XPath-related research (Filiot et al., 2007).

6.2.2 Grammar

Definition 6.2.2 (Grammar). *A grammar $G = (MT, lex, P)$ consists of a multigraph type MT, a lexicon lex, and a set of principles P.*

Definition 6.2.3 (Multigraph Type). *Given a set of atoms At, a multigraph type $MT = (D, W, L, dl, A, T, dat)$ consists of a finite set of dimensions $D \subseteq At$, a finite set of words $W \subseteq At$, a finite set of labels $L \subseteq At$, a dimension-label mapping $dl \in D \to 2^L$, a finite set of attributes $A \subseteq At$, a finite set of types $T \subseteq Ty$, and a dimension-attributes-type mapping $dat \in D \to A \to T$. Ty is the set of types built from finite domains Fd: $Ty ::= 2^{Fd_1 \times \ldots \times Fd_n}$, where $Fd ::= V \mid \{a_1, \ldots, a_n\}$, V is a placeholder for the set of nodes, and $a_1, \ldots, a_n \in At$.*

Definition 6.2.4 (Multigraph of Multigraph Type). *A multigraph $M = (V, E^+, <, nw, na)$ is of multigraph type*

$MT = (D, W, L, dl, A, T, dat)$ iff the sets of dimensions D, words W, edge labels L, attributes A and types T match, all labeled dominances on dimension $d \in D$ have only edge labels in $dl\ d$, and all node attributes $a \in A$ on dimension $d \in D$ have a value in $dat\ d\ a$.

Definition 6.2.5 (Lexicon). *The lexicon is a function from words to sets of lexical entries: $lex \in W \rightarrow 2^{D \rightarrow A' \rightarrow \cup\{ty | ty \in T'\}}$, where $A' \subseteq A$ is the subset of lexical attributes, and for all $w \in W$, if $e \in lex\ w$, then for all $d \in D$, $a \in A'$, $(e\ d\ a)$ has a value in $(dat\ d\ a)$. $T' \subseteq Ty'$, where Ty' is the set of types built from finite domains Fd': $Ty' ::= 2^{Fd'_1 \times \ldots \times Fd'_n}$, where $Fd' ::= \{a_1, \ldots, a_n\}$. That is, lexical attributes can never talk about nodes in the multigraph, whereas non-lexical attributes can—the reason for this is that the set of nodes is not known at the time of creating the lexicon.*

Definition 6.2.6 (Principles). *Principles are a finite set $P \subseteq \phi$ of first-order formulas built from terms $t ::= c \mid x$, where c is an individual constant and x an individual variable. ϕ is defined as follows:*

$$\phi \quad ::= \quad \neg\phi \mid \phi_1 \wedge \phi_2 \mid \exists x : \phi \mid t_1 = t_2 \mid \psi$$

where the predicates *ψ are defined further below. We define the usual logical operators ($\vee$, $\Rightarrow$, $\Leftrightarrow$, $\forall$, $\exists!$, $\neq$) as syntactic sugar, and allow to use variables other than x for convenience (e.g. v for nodes, l for labels, w for words and a for attributes etc.). The constants and predicates of the logic are defined with respect to a multigraph type $MT = (D, W, L, dl, A, T, dat)$. The constants are taken from the set C:*

$$\begin{aligned} C \quad = \quad & D \cup W \cup L \cup A \cup \\ & \{Fd_i \mid 2^{Fd_1 \times \ldots \times Fd_n} \in T,\ 1 \leq i \leq n\} \cup \mathbb{N} \end{aligned}$$

where $\mathbb{N}$ is the set of natural numbers. The universe of the logic is defined given a multigraph $M = (V, E^+, <, nw, na)$, and equals C with the exception that $\mathbb{N}$ is replaced by V, the actual set of nodes. All constants are interpreted by the identity function. As the universe contains only the nodes of the given multigraph, only this finite subset of the natural numbers can be interpreted, i.e., a principle mentioning node 42 can only be interpreted with respect to a multigraph with at least 42 nodes. Here are the predicates ψ:

$$\begin{aligned} \psi \quad ::= \quad & v \xrightarrow{l}_d \rightarrow^*_d v' \\ \mid \quad & v < v' \\ \mid \quad & (W\ v) = w \\ \mid \quad & (t_1 \ldots t_n) \in (d\ v).a \end{aligned}$$

where $v \xrightarrow{l}_d \rightarrow^*_d v'$ is interpreted as the labeled dominance relation, i.e., $(v, v', l, d) \in E^+$ and $v < v'$ by the total order $<$, i.e., $(v, v') \in <$. $(W\ v) = w$ is interpreted by the node-word mapping, i.e., $nw\ v = w$, and $(t_1 \ldots t_n) \in (d\ v).a$ by the node-attributes mapping, i.e., $(t_1, \ldots, t_n) \in na\ v\ d\ a$.

For convenience, we define shortcuts for strict dominance (with any label), labeled edge and edge (with any label):

$$\begin{aligned} v \rightarrow^+_d v' \quad & \stackrel{\text{def}}{=} \quad \exists l : v \xrightarrow{l}_d \rightarrow^*_d v' \\ v \xrightarrow{l}_d v' \quad & \stackrel{\text{def}}{=} \quad v \xrightarrow{l}_d \rightarrow^*_d v' \wedge \neg\exists v'' : v \rightarrow^+_d v'' \wedge v'' \rightarrow^+_d v' \\ v \rightarrow_d v' \quad & \stackrel{\text{def}}{=} \quad \exists l : v \xrightarrow{l}_d v' \end{aligned}$$

where we define labeled edge as labeled dominance between v and v' with the restriction that there must be no node v'' in between.

6.2.3 Example Principles

To get a deeper understanding of the principles of XDG, we provide an extract of the XDG *principle library*. For generality, the principles are parametrized by the dimensions that they constrain.

Definition 6.2.7 (Tree principle). *Given a dimension d, the tree principle stipulates that 1) there must be no cycles, 2) there is precisely one node without a mother (the root), and 3) all nodes have zero or one mothers:*

$$\begin{aligned} & \forall v : \neg(v \rightarrow^+_d v) \wedge \\ & \exists! v : \neg\exists v' : v' \rightarrow_d v \wedge \\ & \forall v : (\neg\exists v' : v' \rightarrow_d v) \vee (\exists! v' : v' \rightarrow_d v) \end{aligned}$$

Definition 6.2.8 (Projectivity principle). *Given a dimension d, the projectivity principle forbids crossing edges by stipulating that all nodes positioned between a head and a dependent must be below the head.*

$$\begin{aligned} & \forall v, v' : \\ & (v \rightarrow_d v' \ \wedge\ v < v' \Rightarrow \forall v'' : v < v'' \wedge v'' < v' \Rightarrow v \rightarrow^+_d v'') \wedge \\ & (v \rightarrow_d v' \ \wedge\ v' < v \Rightarrow \forall v'' : v' < v'' \wedge v'' < v \Rightarrow v \rightarrow^+_d v'') \end{aligned}$$

For example, this principle is violated on the SEM dimension in Figure 6.1, where *wants* is positioned between *eat* and *Mary*, but is not below *eat*.

To explain the lexicalized valency, order and linking principles, we show an example concrete lexical entry for *eat* in Figure 6.3, modeling the schematic lexical entry in Figure 6.2.

Definition 6.2.9 (Valency principle). *Given a dimension d, the valency principle constrains the incoming and outgoing edges of each node according to the lexical attributes in and out of type $2^{(dl\ d) \times \{!, +, ?, *\}}$, which models the function $(dl\ d) \rightarrow \{!, +, ?, *\}$ from edge labels on d to* cardinalities, *where ! stands for "one", + for "more than one", ? for "zero or one", and $*$ for "arbitrary many":*

$$\begin{aligned} & \forall v : \forall l : \\ & ((l, !) \in (d\ v).in \ \Rightarrow\ \exists! v' : v' \xrightarrow{l}_d v) \wedge \\ & ((l, +) \in (d\ v).in \ \Rightarrow\ \exists v' : v' \xrightarrow{l}_d v) \wedge \\ & ((l, ?) \in (d\ v).in \ \Rightarrow\ \neg\exists v' : v' \xrightarrow{l}_d v \vee \exists! v' : v' \xrightarrow{l}_d v) \wedge \\ & (\neg(l, !) \in (d\ v).in \wedge \neg(l, +) \in (d\ v).in \wedge \neg(l, ?) \in (d\ v).in \wedge \\ & \neg(l, *) \in (d\ v).in \ \Rightarrow\ \neg\exists v' : v' \xrightarrow{l}_d v) \wedge \\ & ((l, !) \in (d\ v).out \ \Rightarrow\ \exists! v' : v \xrightarrow{l}_d v') \wedge \\ & \ldots \end{aligned}$$

The part of the principle dealing with the outgoing edges proceeds analogously.

$$
\begin{array}{l}
eat \mapsto \\
\left\{ \begin{array}{l}
\left\{ \begin{array}{ll}
\text{SYN}: & \left\{ \begin{array}{rl}
in & : \{(\mathsf{vinf},?)\} \\
out & : \{(\mathsf{part},!),(\mathsf{obj},?),(\mathsf{adv},*)\} \\
order & : \{(\mathsf{part},\uparrow),(\mathsf{part},\mathsf{obj}), \\
 & \quad (\mathsf{part},\mathsf{adv}),(\uparrow,\mathsf{obj}), \\
 & \quad (\uparrow,\mathsf{adv}),(\mathsf{obj},\mathsf{adv})\}
\end{array} \right\} \\
\text{SEM}: & \left\{ \begin{array}{rl}
in & : \{(\mathsf{th},*)\} \\
out & : \{(\mathsf{ag},!),(\mathsf{pat},?)\} \\
linkEnd & : \{(\mathsf{ag},\mathsf{subj}),(\mathsf{pat},\mathsf{obj})\}
\end{array} \right\}
\end{array} \right\} \\
\quad , \quad \ldots
\end{array} \right\}
\end{array}
$$

Figure 6.3: Concrete lexical entry for *eat*

Given the concrete lexical entry in Figure 6.3, the principle constrains node *eat* on SYN such that there can be zero or one incoming edges labeled vinf, there must be precisely one part dependent, zero or one obj dependents, arbitrary many adv dependents, and no other incoming or outgoing edges.

Definition 6.2.10 (Order principle). *Given a dimension d, the order principle constrains the order of the dependents of each node according to the lexical attribute order of type $2^{(dl\ d)\cup\{\uparrow\}\times(dl\ d)\cup\{\uparrow\}}$. The order attribute models a partial order on $(dl\ d)\cup\{\uparrow\}$, where $\uparrow$ is the head:*

$$
\begin{array}{l}
\forall v : \forall l : \forall l' : (l,l') \in (d\ v).order \Rightarrow \\
\quad \begin{array}{lrcrcl}
\forall v': & l \doteq \uparrow & \wedge & v \xrightarrow{l'}_d v' & \Rightarrow & v < v' \ \wedge \\
\forall v': & v \xrightarrow{l}_d v' & \wedge & l' \doteq \uparrow & \Rightarrow & v' < v \ \wedge \\
\forall v', v'': & v \xrightarrow{l}_d v' & \wedge & v \xrightarrow{l'}_d v'' & \Rightarrow & v' < v''
\end{array}
\end{array}
$$

For instance, given the concrete lexical entry in Figure 6.3, the order principle orders all part dependents to the left of the head *eat*, and to the left of the obj and adv dependents of *eat*.

Definition 6.2.11 (LinkingEnd principle). *Given two dimensions d_1 and d_2, the linkingEnd principle constrains for all edges from v to v' labeled l on d_1 the incoming edge label l' of v' on d_2, according to the lexical attribute linkEnd of type $2^{(dl\ d_1)\times(dl\ d_2)}$. The linkEnd attribute models the function $(dl\ d_1) \to 2^{(dl\ d_2)}$:*

$$
\begin{array}{l}
\forall v : \forall v' : \forall l : \exists l' : v \xrightarrow{l}_{d_1} v' \ \wedge\ (l,l') \in (d_1\ v).linkEnd \ \Rightarrow \\
\exists v'' : v'' \xrightarrow{l'}_{d_2} v
\end{array}
$$

The principle is called "linkingEnd" since what is constrained is the endpoint of the path to v' on d_2. The XDG principle library contains a large number of other linking prinnciples.

In the concrete lexical entry in Figure 6.3, $d_1 = \text{SEM}$ and $d_2 = \text{SYN}$, and the linking principle stipulates e.g. that the agent of *eat* on SEM must have incoming edge label subj on SYN.

6.2.4 Models

Definition 6.2.12 (Models). *The models of a grammar $G = (MT, lex, P)$ are all multigraphs of multigraph type MT which satisfy the lexicon lex and the principles P.*

What does it mean to satisfy the lexicon, and to satisfy the principles?

Definition 6.2.13 (Lexicon Satisfaction). *Given a grammar $G = (MT, lex, P)$, a multigraph $M = (V, E^+, <, nw, na)$ satisfies the lexicon lex iff for all nodes $v \in V$, there is a lexical entry e for the word of v, and for all dimensions $d \in D$ and all lexical attributes $a \in A'$, the value of the lexical attribute a on dimension d for node v equals the value of the lexical attribute a on dimension d of e:*

$$
\begin{array}{l}
\forall v \in V : \exists e \in lex\ (nw\ v) : \forall d \in D : \forall a \in A' : \\
(na\ v\ d\ a) = (e\ d\ a)
\end{array}
$$

Definition 6.2.14 (Principles Satisfaction). *Given a grammar $G = (MT, lex, P)$, a multigraph $M = (V, E^+, <, nw, na)$ satisfies the principles P iff the conjunction of all principles in P is true.*

6.2.5 String Language

Definition 6.2.15 (String Language). *The string language of a grammar $G = (MT, lex, P)$, $L(G)$, is the set of all strings $s = w_1 \ldots w_n$ such that there is a model of G with as many nodes as words, and the concatenation of the words of the nodes following the total order of the model yields s.*

6.2.6 Grammar Intersection

XDG is closed under intersection, i.e., we can combine two grammars G_1 and G_2 to a new grammar $G_1 \cap G_2$, such that the string language of the latter is the intersection of that of G_1 and of G_2. We call this *grammar intersection*.

Definition 6.2.16 (Grammar Intersection). *Given two grammars $G_1 = (MT_1, lex_1, P_1)$ and $G_2 = (MT_2, lex_2, P_2)$ with multigraph types $MT_1 = (D_1, W_1, L_1, dl_1, A_1, T_1, dat_1)$ and $MT_2 = (D_2, W_2, L_2, dl_2, A_2, T_2, dat_2)$, the prerequisites for grammar intersection are:*

1. *The sets of dimensions must be disjoint:*

$$D_1 \cap D_2 = \emptyset$$

2. *The sets of words must be the same:*

$$W_1 = W_2$$

The intersection grammar $G = G_1 \cap G_2$ with $G = (MT, lex, P)$ has the following multigraph type $MT = (D, W, L, dl, A, T, dat)$:

$$\begin{array}{rcl} D & = & D_1 \cup D_2 \\ W & = & W_1 \\ L & = & L_1 \cup L_1 \\ dl & = & dl_1 \cup dl_2 \\ A & = & A_1 \cup A_2 \\ T & = & T_1 \cup T_2 \\ dat & = & dat_1 \cup dat_2 \end{array}$$

The lexicon lex of the intersection grammar is defined as follows for all $w \in W$:

$$lex\ w \quad = \quad \{e_1 \cup e_2 \mid e_1 \in lex_1\ w,\ e_2 \in lex_2\ w\}$$

Here, it becomes obvious why we demand that the grammars to be intersected have the same set of words—otherwise, parts of the lexicon of the intersection grammar would be undefined.

Finally, the principles P of the intersection grammar are defined as the union of those of G_1 and G_2:

$$P \quad = \quad P_1 \cup P_2$$

Theorem 6.2.17 (Grammar Intersection models Language Intersection). *We prove that $L(G_1 \cap G_2) = L(G_1) \cap L(G_2)$, assuming that G_1 and G_2 have dimensions D_1 and D_2, respectively.*

Proof. By the construction in Definition 6.2.16 we have, writing $m(G)$ for the models of grammar G and $M_{|D}$ for the multigraph restricted to dimensions D:

$$M \in m(G_1 \cap G_2) \quad \equiv \quad M_{|D_1} \in m(G_1) \ \wedge\ M_{|D_2} \in m(G_2)$$

That is, each model of $G_1 \cap G_2$ must also be a model of G_1 (restricted to D_1) and a model of G_2 (restricted to D_2).

As strings can be directly read off the models by concatenation of the words of the nodes, we get:

$$s \in L(G_1 \cap G_2) \quad \equiv \quad s \in L(G_1) \ \wedge\ s \in L(G_2)$$

This leads us to our hypothesis:

$$L(G_1 \cap G_2) \quad = \quad L(G_1) \cap L(G_2)$$

□

6.2.7 LCFGs as XDGs

Debusmann (2006) includes a constructive proof that reformulates lexicalized CFGs (LCFGs) as XDGs. Given an LCFG G, it is easy to construct an XDG G' with one dimension called DERI (for "derivation tree"). The derivation trees of the LCFG stand in the following correspondence to the models on DERI:

1. The non-terminal nodes in the derivation tree correspond to the nodes on DERI.
2. The labels of the non-terminal nodes in the derivation tree are represented by the incoming edge labels of the corresponding nodes on DERI, except for the root, which has no incoming edge.
3. The terminal nodes in the derivation tree correspond to the words on DERI.

We depict an example LCFG derivation tree and the corresponding XDG DERI tree in Figure 6.4.

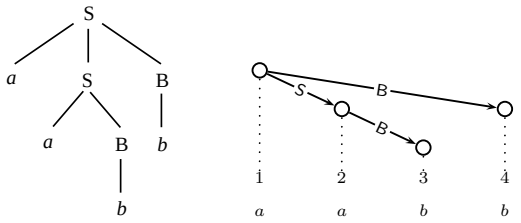


Figure 6.4: LCFG derivation tree (left) and corresponding XDG DERI tree (right)

The constructed XDG grammar uses the tree, projectivity, valency and order principles. The lexicon includes for each rule $A \rightarrow B_1 \ldots B_k a B_{k+1} \ldots B_n$ $(1 \leq k \leq n)$ of the LCFG, given that each non-terminal occurs at most once on the RHS, and given that A is not the start symbol[1], a lexical entry schematically depicted in Figure 6.5. Here, the anchor is the terminal symbol a of the RHS of the LCFG rule. We require precisely one incoming edge labeled by the LHS of the rule, i.e., A. As for the outgoing edges, we require precisely one for each non-terminal on the RHS of the rule. The order requirements model the order among the non-terminals and the anchor.

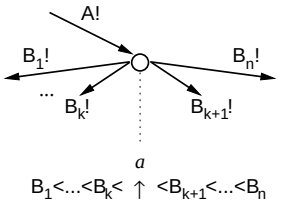


Figure 6.5: Lexical entry for LCFG rule $A \rightarrow B_1 \ldots B_k a B_{k+1} \ldots B_n$

6.3 Scrambling as the Combination of Relaxed LCFGs

In German, roughly, the word order in subordinate sentences is such that all verbs are positioned in the so-called *verb-cluster* at the right frontier, in which they follow their verbal dependents, and the nominal dependents of the verbs are positioned to the left of the verb cluster in

[1] If A is the start symbol, we license zero or one incoming edges labeled A instead of precisely one.

the so-called *Mittelfeld*. Here is an example, translated word-by-word into English for convenience:

(that) $John_1$ $Mary_1$ $Peter_2$ $animals_3$ $feed_3$ $help_2$ saw_1. (6.1)

The subscripts indicate the dependencies between the NPs and the verbs: *John* and *Mary* are dependents of *saw*, *Peter* of *help* and *animals* of *feed*. Figure 6.6 shows an LCFG called G_1 for this word order:

S	→	NP NP VP *saw*	VP	→	NP VP *help*
VP	→	NP *feed*	NP	→	*John*
NP	→	*Mary*	NP	→	*Peter*
NP	→	*animals*			

Figure 6.6: LCFG G_1 capturing *(that)* $John_1$ $Mary_1$ $Peter_2$ $animals_3$ $feed_3$ $help_2$ saw_1.

However, this grammar gives us only one analysis for (6.1), shown in Figure 6.7 (left), whereas 12 are grammatical, which is because the NPs in the Mittelfeld can occur in any permutation[2] irrespectively of the positions of their verbal governors.[3] In order to correctly model this so-called *scrambling* phenomenon, we would also have to additionally license e.g. the discontinuous analysis shown in Figure 6.7 (middle). But how can we do that, given that LCFG derivations are always contiguous?

The key is to reformulate the LCFG in XDG. In XDG, we *can* relax the global contiguity constraint by simply dropping the *projectivity* principle, even though this is still not quite the solution: although the rules for VPs still position their verbal dependents to their left, material from verbs higher up in the tree can now interrupt them, as in Figure 6.7 (right), where the VP *Peter animals feed help* is interrupted by the NPs *John* and *Mary*, and as a result, the verb *feed* wrongly ends up in the Mittelfeld.

We can get around such situations as follows. What we need first is a contiguous LCFG that orders all NPs to the left of the verbs. With this LCFG, called G_2, we attempt an analysis in the spirit of topological fields theory, as in Kathol (1995), Gerdes and Kahane (2001), Duchier and Debusmann (2001). We use the non-terminals MF standing for "Mittelfeld" and VC for "Verb Cluster". The grammar is depicted in Figure 6.8. It licenses analyses where the nouns in the Mittelfeld can occur in any order, and the verbs precede their verbal governors. An example analysis is depicted in Figure 6.9.

Second, we reformulate both grammars G_1 and G_2 in XDG, and again drop the contiguity constraint from G_1. Third, we construct their intersection. This gives us precisely the right structures, where the two combined grammars can be seen as "helping out" each other: G_1 models the syntactic dependencies or *co-occurrences* (Becker, 1994), and G_2 helps out with ordering all NPs to the left of the verbs.

[2] Any permutation is grammatical, though some are strongly marked in performance studies of German.

[3] Why 12? The verb *feed* has 4 possibilities to fill its NP argument slot, there remain 3 for *help*, and 1 for *saw*.

S	→	MF VC *saw*	VC	→	VC *help*
VC	→	*feed*	MF	→	*John*
MF	→	*John* MF	MF	→	*Mary*
MF	→	*Mary* MF	MF	→	*Peter*
MF	→	*Peter* MF	MF	→	*animals*
MF	→	*animals* MF			

Figure 6.8: Topological fields LCFG G_2

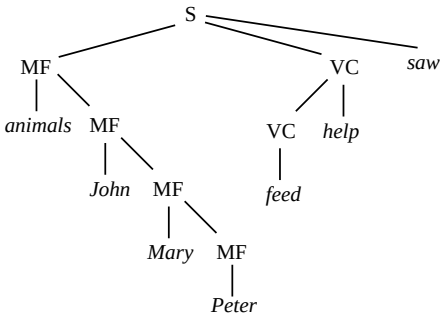


Figure 6.9: Topological derivation tree for *(that)* $animals_3$ $John_1$ $Mary_1$ $Peter_2$ $feed_3$ $help_2$ saw_1.

6.4 Conclusions

By reformulating CFG in a grammar formalism based on MTS, we obtained the flexibility to at the same time relax and extend it. This has brought us into the position to model one of the most complicated phenomena in syntax by the combination of two grammars formulated in one of the simplest of all grammar formalisms.

A related approach to model scrambling by intersection has been put forward in the context of Range Concatenation Grammars (RCG) (Boullier, 2000). In Boullier's approach, the structures generated by the two combined grammars are correlated only by their yields. In his paper "Uses and abuses of intersected languages", Chiang (2004) observes that from the point of view of strong generative capacity, this use of intersection amounts to only constraining the tail end of otherwise independent parallel processes, which he calls *weak parallelism*. He argues that it is easy to overestimate how much control this kind of of parallelism offers. As an example, he argues that the treatment of scrambling in Boullier (2000) is not general enough, as it relies on nonexistent information in the surface string.

Intersection in XDG offers more fine-grained control as Boullier's, and we argue that it thus does not fall into the category of "abuse". First, the dimensions of XDG are synchronized by the input string and the corresponding nodes, which are shared among all dimensions. Second, XDG allows to stipulate any number of additional con-

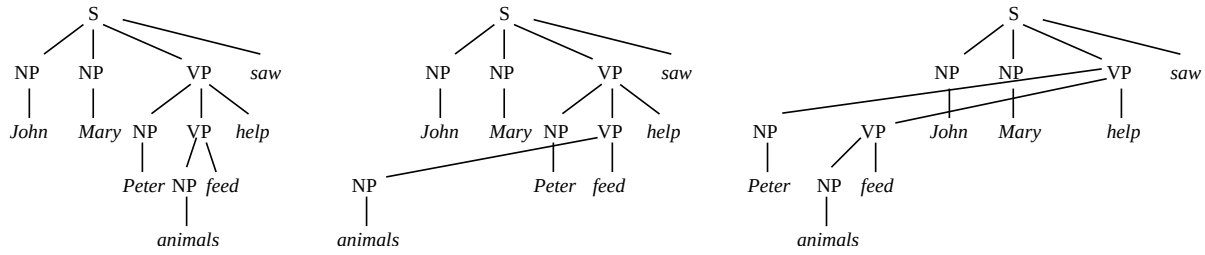


Figure 6.7: Derivation trees

straints to correlate the two intersected grammars, such as the linkingEnd principle (Definition 6.2.11) above.

Acknowledgments

I'd like to thank Prof. Gert Smolka from Programming Systems Lab in Saarbrücken, the people from the CHORUS project, and the International Graduate College (IGK) Saarbrücken/Edinburgh for supporting this research. I'd also like to thank the anonymous reviewers of this paper for their valuable suggestions.

Bibliography

Becker, Tilman (1994). *HyTAG: A New Type of Tree Adjoining Grammars for Hybrid Syntactic Representation of Free Word Order Languages*. Ph.D. thesis, Saarland University.

Blackburn, Patrick and Claire Gardent (1995). A specification language for lexical functional grammars. In *Proceedings of EACL 1995*. Dublin/IE.

Boullier, Pierre (2000). Range concatenation grammars. In *Proceedings of the Sixth International Workshop on Parsing Technologies (IWPT 2000)*, pp. 53–64. Trento/IT.

Bresnan, Joan and Ronald Kaplan (1982). Lexical-Functional Grammar: A formal system for grammatical representation. In Joan Bresnan, ed., *The Mental Representation of Grammatical Relations*, pp. 173–281. The MIT Press, Cambridge/US.

Chiang, David (2004). Uses and abuses of intersected languages. In *Proceedings of TAG+7*, pp. 9–15. Vancouver/CA.

Chomsky, Noam (1981). *Lectures on Government and Binding: The Pisa Lectures*. Foris Publications.

Debusmann, Ralph (2006). *Extensible Dependency Grammar: A Modular Grammar Formalism Based On Multigraph Description*. Ph.D. thesis, Universität des Saarlandes.

Duchier, Denys and Ralph Debusmann (2001). Topological dependency trees: A constraint-based account of linear precedence. In *Proceedings of ACL 2001*. Toulouse/FR.

Filiot, Emmanuel, Joachim Niehren, Jean-Marc Talbot, and Sophie Tison (2007). Polynomial time fragments of xpath with variables. In *Proceedings of the 26th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. Beijing/CN.

Gaifman, Haim (1965). Dependency systems and phrase-structure systems. *Information and Control*, **8**(3):304–337.

Gazdar, Gerald, Ewan Klein, Geoffrey Pullum, and Ivan Sag (1985). *Generalized Phrase Structure Grammar*. B. Blackwell, Oxford/UK.

Gerdes, Kim and Sylvain Kahane (2001). Word order in German: A formal dependency grammar using a topological hierarchy. In *ACL 2001 Proceedings*. Toulouse/FR.

Hudson, Richard A. (1990). *English Word Grammar*. B. Blackwell, Oxford/UK.

Joshi, Aravind K. (1987). An introduction to tree-adjoining grammars. In Alexis Manaster-Ramer, ed., *Mathematics of Language*, pp. 87–115. John Benjamins, Amsterdam/NL.

Kathol, Andreas (1995). *Linearization-Based German Syntax*. Ph.D. thesis, Ohio State University, Ohio/US.

Kepser, Stephan (2000). A coalgebraic modelling of head-driven phrase structure grammar. In *Proceedings of AMiLP 2000*.

Kepser, Stephan and Uwe Mönnich (2003). Graph properties of HPSG feature structures. In Gerhard Jäger, Paola Monachesi, Gerald Penn, and Shuly Wintner, eds., *Proceedings of Formal Grammar 2003*, pp. 115–124.

Kracht, Marcus (1995). Syntactic codes and grammar refinement. *Journal of Language, Logic and Information*, **4**:41–60.

Pollard, Carl and Ivan A. Sag (1994). *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago/US.

Pullum, Geoffrey K. and Barbara C. Scholz (2001). On the distinction between model-theoretic and generative-enumerative syntactic frameworks. In Philippe de Groote, Glyn Morrill, and Christian Retoré, eds., *Logical Aspect of Computational Linguistics: 4th International Conference*, Lecture Notes in Artificial Intelligence, pp. 17–43. Springer, Berlin/DE.

Rogers, James (1996). A model-theoretic framework for theories of syntax. In *Proceedings of ACL 1996*.

Rogers, James (2003). Syntactic structures as multi-dimensional trees. *Journal of Research on Language and Computation*, **1**(3/4).

Rogers, James (2004). On scrambling, another perspective. In *Proceedings of TAG+7*. Vancouver/CA.

Chapter 7

Some Observations on a "Graphical" Model-Theoretical Approach and Generative Models

Joan Chen-Main and Aravind K. Joshi
IRCS, University of Pennsylvania
Philadelphia, USA

7.1 Introduction

Algebraic and logic based model-theoretic approaches to syntax have been extensively studied, revealing interesting and linguistically relevant properties of natural language formalisms. Büchi (1960) and Elgot (1961), for example, have examined the connections between monadic second order (MSO) logic and regular string and tree languages, and Doner (1970) and Thatcher and Wright (1968) have examined the relationship between MSO and regular tree languages (recognizable sets). Morawietz and Mönnich (2001) have provided a model-theoretic account of Tree-Adjoining Grammars (TAG). Rogers (2003) has investigated a model-theoretic characterization of TAGs in terms of MSO on three dimensional trees.

Though related to word-word dependencies, the formalisms mentioned above are primarily phrase structure based. Dependency structures, which rely directly on word-word dependencies, have not been studied model-theoretically until very recently. Bodirsky et al. (2005), Kuhlmann and Möhl (2006), and Kuhlmann and Nivre (2006) initiated a research program based on the properties of the dependency structures rather than in terms of some grammar formalism that generates such structures. Because of this emphasis on the graphs themselves independent of a derivational system, we refer to their approach as a "graphical" model-theoretic approach to syntactic structure. The formal notation used in Kuhlmann and Möhl's (2007) recent work on characterizing dependency languages further facilitates viewing each of their dependency structures as a model of a set of well-formedness conditions. On the empirical front, Kuhlmann and Nivre (2006) investigate the adequacy of a number of formal properties of these graphical models for characterizing the data in two large scale dependency tree banks, the Prague Dependency Treebank (Hajič et al., 2001) and Danish Dependency Treebank (Kromann, 2003). Their results identify three properties that successfully describe more than 99% of the structures in these treebanks. Bodirsky et al. (2005) further show that two of these three properties (*well-nestedness* and a *gap degree* ≤ 1) are exactly the right properties for characterizing graphical models that correspond to derivations in Lexicalized Tree Adjoining Grammar (LTAG).[1] That is, on the theoretical front, their work can be seen as a graphical model-based perspective of the generative approach taken by TAG.

This paper reports some further explorations of the graphical model-theoretic account mentioned above to multi-component extensions of TAG (MC-TAG) and also comments on how the graphical model-theoretic approach and the TAG approach inform each other both when they converge and diverge. In Section 7.2, we provide a brief introduction to TAG, *graph drawings* (the graph models of interest), and the work of Bodirsky et al. (2005) and Möhl (2006) that relates the two. We also touch on how the convergence of the two approaches suggests that TAG provides an answer to the question of what the source of a certain type of discontinuity (*non-projectivity*) is. In Section 7.3, we review examples from Kuhlman (p.c.) and Kuhlmann and Möhl (2006) that show that their graph model is not extendible to tree-local MC-TAG. Because MC-TAGs allow only additional derivations, not additional derived phrase structures or strings (that is, they are weakly equivalent to TAGs), this divergence is somewhat surprising. Based on the success of the graph-model approach with respect to characterizing the Czech and Dutch structures in the previously mentioned treebanks, one is tempted to conclude that multi-component extensions are unnecessarily expressive. However, multi-component extensions have proven useful for characterizing a wide range of linguistic phenomena such as extraposition, anaphoric binding, quantifier scope ambiguities, that are difficult for basic TAG. In Section 7.4, we consider MC-TAGs in use. In particular, we examine the cases where MC-TAG extensions have been argued to be necessary for linguistically satisfying accounts and find that these MC-TAGs do in fact have graph models that satisfy the constraints identified by Bodirsky et al. (2005) due to certain properties

[1] The third property (*edge degree* $\leq$) has no clear relationship to TAG-derivations and is not pursued further in later work by the same authors. Thus, reference to it is omitted in the remaining text. See Kuhlmann and Nivre (2006) for the definition of edge degree.

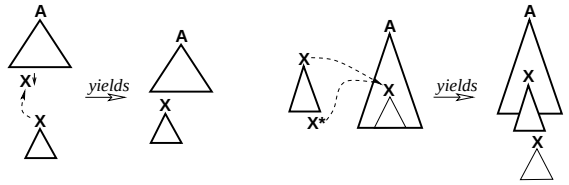


Figure 7.1: Substitution and Adjoining

of the particular multi-component grammars. The convergence of the graph model approach and MC-TAGs as used suggests that a natural class of MC-TAG grammars can be identified that is more restrictive and a closer fit to natural language than MC-TAGs in general. Finally, Section 7.5 summarizes on-going work that compares the scrambling patterns that are possible in various MC-TAG extensions with those that conform to the constraints of the graph model approach. Thus far, it appears that the two approaches diverge in the domain of scrambling. In a sense, the work presented in these last two sections may be considered as case studies of some fine grained relations between the graphical model-theoretic approach and the generative approach. We suspect that similar divergences where certain properties of structural descriptions are easier to express in one approach vs. another may exist when the logic based model-theoretic approach and the generative approach are examined in a very fine grained manner.

7.2 A Graph-Model Perspective on Tree Adjoining Grammar

7.2.1 TAG Basics

Tree Adjoining Grammar (TAG) is a structure-generating formalism for rewriting nodes of a tree as other trees (Joshi, Levy, and Takahashi, 1975).[2] The finite set of underived trees are called *elementary trees*. The rewriting is accomplished via two operations: *substitution*, which rewrites a leaf node labeled X with a tree rooted in a node labeled X, and *adjoining*, which rewrites a node X with a tree that labels both its root and a distinguished leaf node, the *foot*, with X. Each of these operations is illustrated in Figure 7.1. (Down arrows are often used to denote substitution sites, while asterisks are used to denote foot nodes.)

TAG itself is a mathematical formalism, not explicitly a theory of natural language, but has proven useful when applied to the analysis of natural language syntax. In particular, the observation that linguistic dependencies typically occur within some sort of local domain is expressed in the TAG hypothesis that all such dependencies occur within the basic building blocks of the grammar. Recursive structure is"factored out," which allows apparent non-local dependencies to be recast as local ones.

A closer look at the linguistic data, however, has motivated extensions of TAG. One of the most widely used extensions for handling cases that are difficult for classic TAG is Multi-Component TAG (Weir, 1988). Whereas basic TAG takes the basic unit of the grammar to be a single elementary tree, MC-TAG extends the domain of locality to encompass a set of elementary trees. That is, these sets are the objects over which the combinatory operations apply. The MC-extension allows for linguistically satisfying accounts for a number of attested phenomena, such as: English extraposition (Kroch and Joshi, 1987), subj-aux inversion in combination with raising verbs (Frank, 1992), anaphoric binding (Ryant and Scheffler, 2006), quantifier scope ambiguity (Joshi et al., 2003), clitic climbing in Romance (Bleam, 2000), and Japanese causatives (Heycock, 1986).

7.2.2 Graph Drawing Basics

These next two subsections are intended to provide an intuition for the terms that are defined more formally in Bodirsky et al. (2005) and Kuhlmann and Nivre (2006). In the former, the authors define their dependency structures of interest, called *graph drawings*, as a three-tuple: a set of nodes, a dominance relation, and a precedence relation. Each of these dependency structures is encodes information also carried by both a TAG-derivation and that derivation's final phrase structure. The TAG is assumed to be strictly lexicalized (LTAG). The anchor of each elementary tree of a strictly lexicalized TAG (LTAG) corresponds to a node name in the induced dependency structure. E.g., suppose tree A is anchored by lexical item *a* in the LTAG grammar. Then *a* will be a node name in any dependency structure induced by an LTAG derivation involving tree A.

To see how the dominance relation and precedence relation mirror the derivation and the final derived phrase structure, let us further suppose that LTAG tree B is anchored by lexical item *b*. The following relationships hold between and LTAG derivation and the dependency structure it induces:

- Node *a* dominates node *b* in the dependency structure iff Tree A dominates tree B in the LTAG derivation structure. (i.e., tree B must combine (substitute/adjoin) into tree A during the TAG-derivation.[3]
- Node *a* precedes node *b* in the dependency structure iff *a* precedes *b* in the derived phrase structure tree.

[2] For a readable introduction, see Chapter 1 of Frank (2002).

[3] A consequence is that TAG-induced dependency graphs do sometimes diverge from standard dependency graphs. e.g. The standard TAG analysis for cyclic *wh*-movement adjoins the matrix clause tree into the tree corresponding to its complement embedded clause. Thus, in the graph drawing, the matrix verb is a dependent on the embedded verb, while the reverse dependency holds in a standard dependency graph.

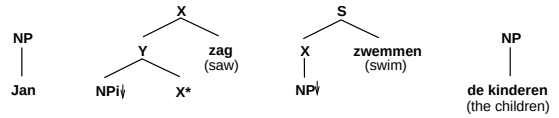


Figure 7.2: LTAG for *Jan de kinderen zag zwemmen.*

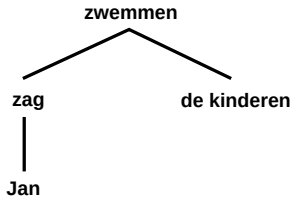


Figure 7.3: Derivation structure for *Jan de kinderen zag zwemmen.*

An example based on the cross-serial dependencies seen in Dutch subordinate clauses is given in Figures 7.2–7.5. In the graph drawing in (7.5), the four nodes names, {*Jan*, *de kinderen*, *zag*, *zwemmen*}, are the same set as the anchors of the elementary trees in (7.2), which is the same as the set of terminals in (7.4), the derived phrase structure. The ordering of the nodes in (7.5) is exactly the ordering of the terminals in (7.4). The directed edges between the nodes in (7.5) mirrors the immediate dominance relation represented in (7.3), the derivation structure showing how the trees in (7.2) combine. E.g., Just as the *zwemmen* node has the *zag* and *de kinderen* nodes as its two children in (7.3), so does the *zwemmen* node dominate *zag* and *de kinderen* in (7.5).

Whereas a TAG approach deems an expression well-formed when there exists a derivation for that expression,

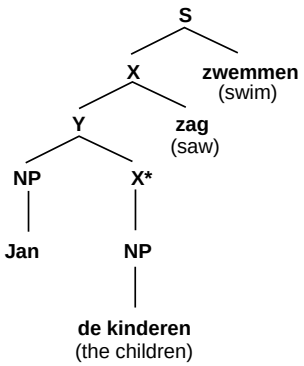


Figure 7.4: Derived phrase structure for *Jan de kinderen zag zwemmen.*

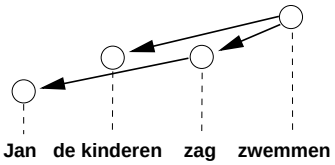


Figure 7.5: Graph drawing corresponding to derivation for *Jan de kinderen zag zwemmen*

the graph model approach considers an expression well-formed when there exists a graph model with the properties defined in the subsection below.

7.2.3 Properties of TAG-induced Graph Drawings

Möhl (2006) provides the formal details showing that the LTAG-induced dependency structures described above have the properties of being 1) well-nested and 2) gap degree ≤ 1, and, conversely, that structures with these properties have a corresponding LTAG derivation. Note that this bi-directional correspondence relates single structures only (i.e. a single graph drawing and a particular LTAG derivation). However, when it comes to a (possibly infinite) set of structures and derivations, the correspondence is clear in only one direction: given a particular LTAG, we can be sure that all derivations will each correspond to a well-nested dependency structure of gap degree at most 1, whether the set of derivations is infinite or not. Given an infinite set of well-nested dependency structures with gap degree at most 1, additional constraints are needed to guarantee the existence of a finite LTAG that derives exactly those structures. That is, there exist sets of well-nested, gap degree ≤ 1 dependency structures that cannot be derived by an LTAG (Möhl, p.c.). Kuhlmann and Möhl (2007) further discuss the relationship between dependency structure languages and the mildly-context sensitive languages that have been most often characterized derivationally.

Gap Degree

It will be useful to first define the term *projection*:

Definition: The projection of a node x is the set of nodes dominated by x (including x). (E.g., in (7.5), the projection of zag ={ *Jan*, *zag* }.)

Recall that the nodes of a graph drawing are in a precedence relation, and that this precedence relation is total.

Definition: A gap is a discontinuity with respect to precedence in the projection of a node in the drawing. (E.g., in (7.5), *Jan* and *zag* do not form a contiguous interval.)

Definition: The gap degree of a node is the number of gaps in its projection. (E.g., in (7.5), the gap degree of

node *zag* = 1.)

Definition: The gap degree of a drawing is the maximum among the gap degrees of its nodes. (E.g., in (7.5), only the projection of *zag* is interrupted by a gap. Thus, the gap degree of the graph drawing in Figure 7.5 = 1.)

When a graph drawing has a gap degree of zero (i.e. there are no discontinuities in the projections of any nodes), then the graph is *projective*.

The source of every gap in a TAG drawing comes from an interruption of the dependencies in an auxiliary tree. If B is adjoined into A, the gap is the material in A that is below the foot node of B. E.g., the gap in the projection of *zag* in Figure 7.2 is *de kinderen*. Note that the substitution site for *de kinderen* is below the X node in the tree anchored by *zwemmen*. In the derived phrase structure tree, this is the pronounced material that falls below the foot node of the *zag* tree; *de kinderen* is what interrupts the pronounced material on the left of the foot node, *Jan*, from the pronounced material on the right of the foot node, *zag*.

Understanding the relationship between gap degree in dependency structures and TAG derivations reveals that a commitment to basic TAG is actually accompanied by a general claim about the source of non-projectivity in natural language dependencies. That auxiliary trees with (slots for) pronounced material on both sides of the foot (*wrapping auxiliary trees*) are the source of every non-projective dependency [4] is an underlying assumption of the parsers reported in (Shen, 2006), and, in particular, this assumption contributes to the efficiency of the TAG parser reported in (Shen and Joshi, 2007). This is worth noting since non-projectivity is attested in natural language (as in movement and free word order) and also requires a special approach (e.g., temporarily avoiding non-projectivity, recovering non-projective structures in post-processing, as in Nivre and Nilsson (2005)) for dependency structure parsing.

Well-Nestedness

Definition: If the roots of two subtrees in the drawing are not in a dominance relation, then the trees are *disjoint*. (E.g., in (7.6), the subtrees rooted in *b* and *c* are disjoint, while the subtrees rooted in *a* and *b* are not.)

Definition: If nodes *x1*, *x2* belong to tree X, nodes *y1*, *y2* belong to tree Y, precedence orders these nodes: $x1 > y1 > x2 > y2$, and X and Y are disjoint, then trees X and Y *interleave*. (E.g., in (7.6), *b* and *d* belong to the subtree rooted in *b*, while *c* and *e* belong to the subtree rooted in *c*. These two subtrees are disjoint. Since the nodes are ordered $b > c > d > e$, the two trees interleave.)

Definition: If there is no interleaving between disjoint subtrees, then a graph drawing is *well-nested*. (e.g., (7.5) is well-nested, but (7.6) is not)

[4] This refers only to dependencies between tree anchors, not to dependencies expressed within an elementary tree (i.e. using traces).

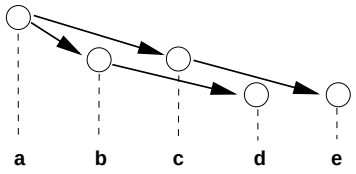


Figure 7.6: Non-well nested graph drawing

7.3 MC-TAG-induced Graphs: Gap Degree Beyond 1 and Not Well-Nested

The weak equivalence of tree-local multi-component TAGs to basic TAGs suggests that these graphical model-theoretic results may be extended to tree-local MC-TAG. This does not turn out to be the case.

7.3.1 Gap-Degree Beyond 1

As reviewed in Section 7.2, the source of every gap in a TAG drawing comes from an interruption of the dependencies in an auxiliary tree. Since the auxiliary tree only has one foot, it only has a slot for a single gap. A MC-set, however, could be comprised of two auxiliary trees. This means there are slots for two gaps, one associated with each foot. Furthermore, a gap may arise as a result of any pronounced material between the two components. Thus, when we already have at least one foot, adding an additional foot increases the maximum gap degree by 2. The maximum gap degree $= 1 + 2(n-1) = 2n-1$, where n is the maximum number of foot nodes in any elementary tree set.

As an example, consider the composition of the trees in (7.7) (Kuhlmann, p.c.) The tree set in (7.7w) is comprised of two auxiliary trees. One tree, (7.7wα), adjoins into (7.7a), and a gap is created by the material in (7.7a) that falls below the foot node of (7.7wα),namely *b*. When (7.7wβ) is adjoined into (7.7wα) at node V, a second gap is created below (7.7wβ) by *d*. A third gap is created by the material between the two components. (7.8) shows the derived phrase structure, and (7.9), the corresponding graph drawing. The projection of node *w*, $\{w, x, y, z\}$ has three discontinuities, nodes *b*, *c*, and *d*.

7.3.2 Non-Well-Nestedness

Kuhlmann and Möhl (2006) show that even a tree-local MC-TAG that allows only substitution can induce a non-well-nested graph drawing. Their example is repeated below. This derivation involves two MC-sets, (7.10b) and (7.10c). The tree anchored by *d*, (7.10d), substitutes into the second component of the set anchored by *b*, (7.10b). Similarly, the tree anchored by *e*, (7.10e), substitutes into

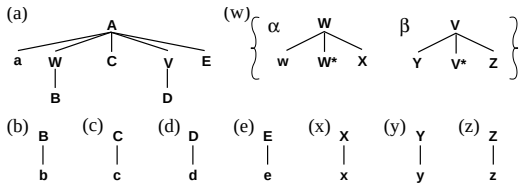


Figure 7.7: MC-TAG that induces a graph drawing of gap degree 3

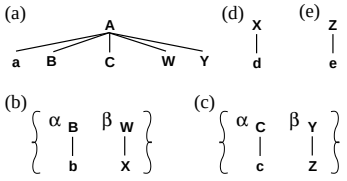


Figure 7.10: MC-TAG that induces a non-well-nested graph drawing

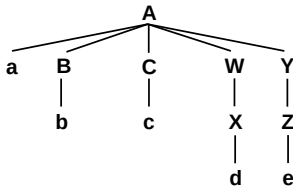


Figure 7.11: Phrase structure combining trees in (7.10)

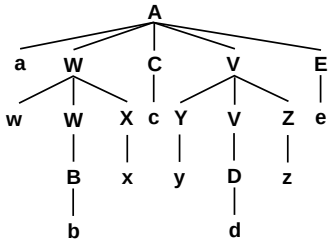


Figure 7.8: Phrase structure combining trees in (7.7)

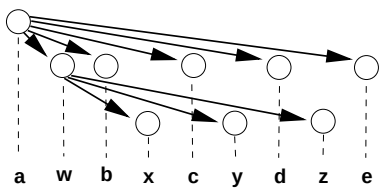


Figure 7.9: Graph drawing of gap degree 3.

the second component of the set anchored by *c*, (7.10c). Both MC-sets compose into the tree anchored by *a*, yielding the derived phrase structure in (7.11). The corresponding graph drawing is exactly our earlier example of non-well-nestedness in (7.6).

7.4 MC-TAG in Practice: Gap Degree ≤ 1 and Well-Nested

We now turn to cases in which linguists have used MC-TAGs to account for cases argued to have no satisfying solution in basic TAG. These include English extraposition (Kroch and Joshi, 1987), Japanese causatives (Heycock, 1986), *subj-aux* inversion with raising verb constructions (Frank, 1992), clitic climbing in Romance (Bleam, 2000), quantifier scope ambiguity (Joshi et al., 2003), and anaphoric binding (Ryant and Scheffler, 2006). Despite the results reported in Section 7.3, these particular MC-derivations correspond to dependency structures that are well-nested and have gap degree ≤ 1. Table 7.1 summarizes these cases. The last column indicates the type of MC-extension assumed by the analysis: tree-local MC-TAGs, tree-local MC-TAGs with *flexible composition*, the mirror operation to adjoining; if tree α adjoins into tree β, the combination can be alternatively viewed as tree β "flexibly" composing with tree α (Joshi et al., 2003; Kallmeyer and Joshi, 2003),[5] and set-local

[5]I.e. When composing A and B, we can take A as the function and B as the argument or vice versa. For CFGs, such flexibility has no added benefit. For categorical type grammars, this kind of flexibility is accomplished via type raising, which allows for some new types of constituents but does not give rise to any new word orders. For tree local MC-TAGs, such flexibility does allow more word orders (permutations) to be generated than are possible without flexible composition.

MC-TAGs. Set-local MC-TAGs are generally more powerful than TAGs, but since these particular cases induce well-nested graph drawings of gap degree ≤ 1, we can conclude that set-local MC-TAG as used in these cases is weakly equivalent to TAG.

From Table 7.1, we can draw two generalizations. First, in an MC-TAG analysis, a two-component set is typically used. One of the trees is often a very small piece of structure that corresponds to the "base position," surface position, or scope position of a single element. Second, the auxiliary tree components typically have elements with phonological content only on one side of the foot.

At this point, we make explicit an assumption that we believe aligns with Bodirsky et al. (2005). Because the TAG-related dependency structures are induced via the surface order of pronounced items and the derivation structure, some information encoded in the derived phrase structure is lost. Of particular relevance here, the relationship between a pronounced element in "surface position" and a silent element in its "base position" will not be preserved. Recall that the node names in the dependency structures correspond to the lexical anchors of elementary trees. The silent elements do not anchor an elementary tree, and thus, do not correspond to a node in the dependency structure.

7.4.1 Why the Gap-Degree Remains ≤ 1

Recall that in the example MC-TAG in (7.7), each of the two components has a foot with phonological material on both sides, giving rise to two gaps, and a third gap is created via the material between the two components. In contrast, in the MC-TAG sets shown in Table 7.1, the auxiliary tree components have pronounced material only on one side of the foot node. This eliminates the gap that would have arisen due to the interruption of material on the left side of the foot from the right side of the foot as a result of the pronounced material beneath the foot. The only way to obtain pronounced material on both sides of the foot node is to adjoin a component into one of these auxiliary trees. Interestingly, the set-local analyses (in which all components of a set must combine with components of a single set vs. tree-local MC-TAG) for clitic climbing and Japanese causatives do posit recursive components adjoining into other recursive components, but only while maintaining all pronounced material on one side of the foot.[6] In the absence of a derivational step resulting in pronounced material on both sides of a foot, the only remaining possible gap is that which arises from pronounced material that appears between the two components.

Note that the observation about the position of pronounced material applies only to auxiliary trees in sets with multiple components. That is, auxiliary trees that comprise a singleton set may still have pronounced material on both sides of the foot

[6] In fact, we have been unable to construct a clean linguistic example requiring a derived auxiliary tree component with pronounced material on both sides of its foot.

7.4.2 Why the Structures Remain Well-Nested

Since Kuhlmann and Möhl (2006) show that even a MC-TAG that allows only non-auxiliary trees in MC-sets will expand the drawings to include non-well-nested drawings, there is no way to pare back the MC-TAG via restrictions on the types of trees allowed in MC-sets so as to avoid interleaving.

To see why the graph drawings that model these MC-TAGs remain well-nested, first recall that to satisfy the definition of interleaving, it is necessary that two subgraphs with alternating nodes are not in a dominance relation in the derivation structure. Further, each of these subgraphs must be associated with a gap-inducing elementary object. At least one MC-set is involved. (The second gap-inducing object may be a MC-set or a wrapping auxiliary tree.) Both gap-inducing objects must combine into the same tree, and the site of combination of one component of the MC-set must be within the gap of the other gap-inducing object. In Kuhlmann and Möhl's example, both gap-inducing elementary objects are MC-sets. The disjointness condition is satisfied because the two MC-sets are sisters in the derivation; they combine into the same tree. In the linguistic analyses considered here, no more than one MC-set combines into the same tree and there is no interaction with wrapping auxiliary trees. For tree-local MC-TAG, it appears to be sufficient to bar more than one MC-set from combining into a single tree (in the absence of wrapping auxiliary trees).

7.5 MC-TAG and Scrambling

In subordinate clauses in Standard German, the canonical order of verbs and their subject arguments is a nested dependency order. However, other orderings are also possible. For example, in the case of a clause-final cluster of three verbs, the canonical order is as given in (7.13), $NP_1\,NP_2\,NP_3\,V_3\,V_2\,V_1$, but all the other permutations of the NP arguments are also possible orderings. All six permutations of the NPs can be derived via tree-local MC-TAG. From the graph-model perspective adopted here, this is unsurprising: All the sequences are well-nested and have gap degree ≤ 1.

(7.13)

NP_1	NP_2	NP_3	V_3
...Hans	Peter	Marie	schwimmen
...Hans	Peter	Marie	swim

V_2	V_1
lassen	sah
make	saw

"...Hans saw Peter make Marie swim."

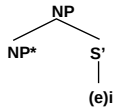
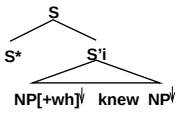
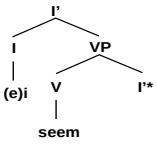
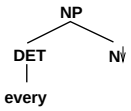
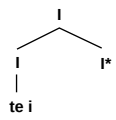
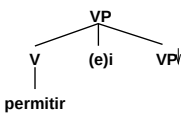
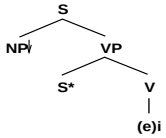
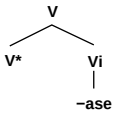
analysis source	phenomenon	first component	second component	MC-type
Kroch and Joshi 1987	English extraposition *A man arrived who knew Mary.*	auxiliary 	auxiliary 	tree-local
Frank 1992	subj-aux inversion with raising verb constructions *Does Gabriel seem to like gnocchi?*	non-auxiliary 	auxiliary 	tree-local
Ryant and Scheffler 2006	anaphoric binding *John$_i$ likes himself$_i$*	auxiliary 	non-auxiliary 	tree-local + flexible composition
Joshi, Kallmeyer, and Romero 2003	quantifier scope ambiguity $(\forall y[\text{prof}(y) \rightarrow \exists x\,[\text{agent}(x) \wedge \text{spy}(x,y)]])$ OR $(\exists(x)[\text{agent}(x) \wedge \forall y[\text{prof}(y) \rightarrow \text{spy}(x,y)]])$	auxiliary 	auxiliary 	tree-local + flexible composition
Bleam 1994	clitic climbing in Romance *Mari telo quiere permitir ver* *Mari you-it wants topermit tosee* "Mari wants to permit you to see it."	auxiliary 	non-auxiliary 	set-local
Heycock 1986	Japanese causatives *Watashi-wa Mitiko-ni Taroo-o ik-ase (-sase) -ta.* *I-TOP DAT ACC go-CS -CS -PST* "I made Mitiko make Taroo go."	auxiliary 	auxiliary 	set-local

Table 7.1: Canonical tree sets used in MC-TAG analyses of several phenomena

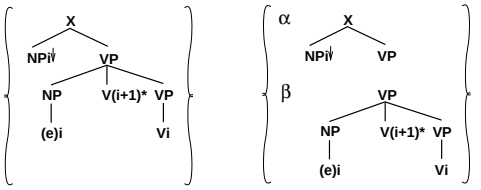


Figure 7.14: Single and two-component sets for V_i

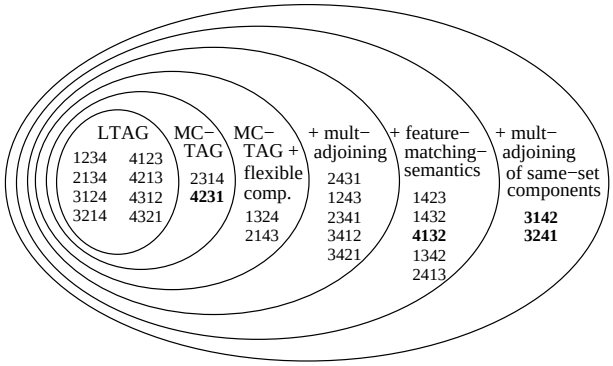


Figure 7.15: TAG variations and derivable noun sequences (sequences with gap degree = 2 are bolded)

However, with an additional level of embedding, i.e. four NPs and four verbs, the situation is different, both linguistically and formally. Our focus is on making the formal predictions of a linguistically informed system precise. We start with a tree-local MC-TAG that is restricted to linguistically motivated tree-sets and to semantically coherent derivations. The former linguistic restriction is illustrated in (7.14), the possible tree-sets anchored by a verb that takes a VP argument. The latter linguistic restriction is that the VP argument of V_i must be associated with V_{i+1}.

We examine the 24 orderings that result from permuting the four nouns while keeping the verb order fixed. (We are aware that German also allows verbs to scramble.) The diagram in (7.15) shows the orderings that we are sure can be derived as we move from LTAG to MC-TAG to MC-TAG enriched in various ways. Two caveats should be mentioned. First, while some of these subset relations are necessary (e.g., MC-TAG derivations are a subset of derivations possible with MC-TAGs that allow flexible composition), others are not (e.g., allowing multiple adjoining is orthogonal to flexible composition). The diagram reflects the order of our investigation, not logical necessity. Second, we may later find that some orders may be derived in a less enriched system.

To understand (7.15), consider the set labeled "MC-TAG+flex-comp" as an example. The two sequences listed in this set but outside the set labeled "MC-TAG," 1324 and 4231, indicate that we have found derivations for $NP_1\,NP_3\,NP_2\,NP_4\,V_4\,V_3\,V_2\,V_1$ and $NP_4\,NP_2\,NP_3\,NP_1\,V_4\,V_3\,V_2\,V_1$ using an MC-TAG with flexible composition but not using an MC-TAG without flexible composition. The orders listed in the subsets labeled MC-TAG and LTAG are also derivable using MC-TAG with flexible composition. When we allow the sort of multiple adjoining at the same node as Schabes and Shieber (1994) (i.e. at most one of those trees is a predicative tree) in addition to flexible composition, we are assured of the derivations of five more orders. For the derivations of the members of the set labeled "+mult-adj" (which include members of its subsets), the notion of semantic coherence can be described as requiring the V_i tree set to combine with only the V_{i+1} or V_{i-1} tree set (which itself has optionally had another tree set combined into it). To derive the orders in the set labeled "+feature-matching-sem," we relax our notion of semantic coherence to allow the V_i tree set to combine with any other tree set, so long as there is no semantic feature clash at any stages of the derivation. For example, the V_1 tree set is permitted to combine with the V_3 tree set, but only if the V_3 tree set has already combined with the V_2 tree set. Otherwise, the desired VP argument of V_1 is missing. Finally, when components from the same multi-component set are allowed to target the same node, derivations for all 24 orderings are possible.

Taking the dependency structures of these sequences to consist of an edge from each verb V_i to its subject NP and to the head of its argument VP, V_{i+1}, we can compare the predictions of the graph drawing approach and the MC-TAG approach.[7] It turns out that the permutations of gap degree ≤ 1 and those of gap-degree 2 do not align in an obvious way with particular enrichments. For example, $NP_4\,NP_2\,NP_3\,NP_1\,V_4\,V_3\,V_2\,V_1$ (gap degree 2) is derivable via basic tree-local MC-TAG, but $NP_3\,NP_1\,NP_3\,NP_4\,V_2\,V_3\,V_2\,V_1$ and $NP_3\,NP_2\,NP_3\,NP_4\,V_1\,V_3\,V_2\,V_1$ (also gap degree 2) appear to require both flexible composition and allowing components from the same MC-set to target the same node.

7.6 Conclusion

This paper compares and contrasts what we dub a "graphical" model theoretic approach based on dependency structures with the derivational approach of Tree Adjoining Grammar. We review the connection established in previous work between TAG derivations and the class of well-nested dependency structures of gap degree $\leq$ 1, and report several observations that build on this work. First, understanding the relationship between gaps

[7] In the cases where we exploit a feature matching semantics, the TAG induced dependency structure will not always be the same as the dependency structures described here.

in graph drawings and foot nodes of LTAG auxiliary trees highlights the implicit LTAG assumption that non-projectivity arises from the adjoining of an auxiliary tree with pronounced material on both sides of its foot. Second, Kuhlmann and Möhl (2006) show that MC-TAGs in which each set has a single lexical anchor induce graph drawings that are outside the class of well-nested, gap degree ≤ 1 dependency structures. Since MC-TAGs generate the same trees and strings as basic TAGs, differing only with respect to the possible derivations, the divergence between basic TAGs and MC-TAGs from the graph drawing perspective indicates that derivational information matters, even though there is no explicit notion of a derivation in the graph model approach and no explicit derivational history in a TAG phrase structure. Third, we observe that despite the formal examples, the graph drawings induced by MC-TAGs used in linguistic analyses continue to fall within the class of well-nested, gap degree ≤ 1 dependency structures. That is, the extra complexity of MC-TAG as reflected in the corresponding dependency structures does not appear to be utilized for linguistic purposes. Even for the crucial cases used to argue for MC-extensions, MC-TAG is used in a manner requiring less complexity than the formal system allows. Examining these particular grammars lays the groundwork for identifying a natural class of MC-TAG grammars whose derivations correspond to well-nested graph drawings of gap degree ≤ 1. Specifically, the observations suggest the class to be MC-TAGs in which 1) component sets have up to two members, and 2) auxiliary trees that are members of non-singleton MC-sets have pronounced material on only one side of the foot, whether the auxiliary member is derived or not. An additional constraint that limits the interaction of gap-inducing objects is also needed. Though these constraints appears stipulative from a formal perspective, a preliminary look suggests that natural language will not require their violation. That is, we may find linguistic justification for these constraints. Lastly, in ongoing work, we explore how a range of scrambling patterns can be derived as MC-TAG is enriched in various ways (by allowing flexible composition, multiple adjoining at the same node, and/or components from the same MC-set to target the same node). Within the domain of scrambling, we do not observe any obvious alignment of the two approaches.

Acknowledgements
This paper overlaps with material presented at the DLP2007 workshop. Julia Hockenmaier is gratefully acknowledged for helpful feedback and generous technical assistance. Thanks is also due to Jim Rogers for assistance with LaTeX.

Bibliography

Bleam, Tonia (2000). Clitic climbing and the power of Tree Adjoining Grammar. In Anne Abeillé and Owen Rambow, eds., *Tree Adjoining Grammars: formalisms, linguistic analysis and processing*, pp. 193–220. Stanford CLSI Publications. Written in 1994.

Bodirsky, Manuel, Marco Kuhlmann, and Mathias Möhl (2005). Well-nested drawings as models of syntactic structure. In *10th Conference of Formal Grammar and 9th Meeting on Mathematics of Language (FG-MoL)*. Edinburgh, UK.

Büchi, J. R. (1960). Weak second-order arithmetic and finite automata. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, **6**:66–92.

Doner, J. E. (1970). Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, **4**:406–451.

Elgot, C. C. (1961). Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, **98**:21–51.

Frank, Robert (1992). *Syntactic Locality and Tree Adjoining Grammar: Grammatical, Acquisition, and Processing Perspectives*. Ph.D. thesis, University of Pennsylvania, Philadelphia PA.

Frank, Robert (2002). *Phrase-structure composition and syntactic dependencies*. MIT Press.

Hajič, Jan, Barbora Vidova Hladka, Jarmila Panevová, Eva Hajičová, Petr Sgall, and Petr Pajas (2001). Prague dependency treebank 1.0. LDC, 2001T10.

Heycock, Caroline (1986). The structure of the Japanese causative. Technical Report MS-CIS-87-55, University of Pennsylvania.

Joshi, Aravind K., Laura Kallmeyer, and Maribel Romero (2003). Flexible composition in LTAG: quantifier scope and inverse linking. In Harry Bunt and Reinhard Muskens, eds., *Computing Meaning 3*. Kluwer, Dordrecht.

Kallmeyer, Laura and Aravind K. Joshi (2003). Factoring predicate argument and scope semantics: underspecified semantics with LTAG. *Research on Language and Computation*, **1**(1-2):3–58.

Kroch, Anthony and Aravind Joshi (1987). Extraposition in tree-adjoining grammar. In Geoffrey Huck and Almerindo Ojeda, eds., *Syntax and Semantics*, volume 20, pp. 107–151. Academic Press.

Kromann, Matthias Trautner (2003). The Danish dependency treebank and the DTAG treebank tool. In *2nd Workshop on Treebanks and Linguistic Theories (TLT)*, pp. 217–220.

Kuhlmann, Marco and Mathias Möhl (2006). Extended cross-serial dependencies in Tree Adjoining Grammars. In *Proceedings of the Eighth International Workshop on Tree Adjoining Grammar and Related Formalisms*, pp. 121–126. Association for Computational Linguistics, Sydney, Australia.

Kuhlmann, Marco and Mathias Möhl (2007). Mildly context-sensitive dependency language. In *45th Annual Meeting of the Association for Computational Linguistics (ACL)*. Association for Computational Linguistics, Prague, Czech Republic.

Kuhlmann, Marco and Joakim Nivre (2006). Mildly non-projective dependency structures. In *21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING-ACL), Companion Volume*, pp. 507–514. Association for Computational Linguistics, Sydney, Australia.

Möhl, Mathias (2006). *Drawings as Models of Syntactic Structure: Theory and Algorithms*. Master's thesis, Saarland University, Saarbrücken, Germany.

Morawietz, Frank and Uwe Mönnich (2001). A model-theoretic description of Tree Adjoining Grammars. *Electronic Notes in Theoretical Computer Science*, **53**.

Nivre, Joakim and Jens Nilsson (2005). Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, p. 99–106. Association for Computational Linguistics, Ann Arbor, Michigan.

Rogers, James (2003). wMSO theories as grammar formalisms. *Theoretical Computer Science*, **293**:291–320.

Ryant, Neville and Tatjana Scheffler (2006). Binding of anaphors in LTAG. In *Proceedings of the Eighth International Workshop on Tree Adjoining Grammar and Related Formalisms*, pp. 65–72. Association for Computational Linguistics, Sydney, Australia.

Schabes, Yves and Stuart M. Shieber (1994). An alternative conception of tree-adjoining derivation. *Computational Linguistics*, **20**(1):91–124.

Shen, Libin (2006). *Statistical LTAG Parsing*. Ph.D. thesis, University of Pennsylvania.

Shen, Libin and Aravind K. Joshi (2007). Bidirectional LTAG Dependency Parsing. Technical Report 07-02, IRCS, University of Pennsylvania.

Thatcher, James W. and Jesse B. Wright (1968). Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, **2**:57–81.

Weir, David J. (1988). *Characterizing Mildly Context-Sensitive Grammar Formalisms*. Ph.D. thesis, University of Pennsylvania.

Chapter 8

Programming Language Semantics and Model Theoretic Syntax

M. Andrew Moshier
Department of Mathematics, Computer Science and Physics
Chapman University
Orange, CA USA
moshier@chapman.edu

8.1 Everything Old is New Again

The Call for Papers for this conference delineates the following opposition: "Model Theoretic Syntax focus[es] on descriptive approaches to formalizing theories of syntax by defining classes of ordinary mathematical structures directly in terms of linguistically relevant structural properties *rather than* in terms of generative or automata-theoretic processes." (emphasis added). Although this is a fair characterization of the field, it seems to suggest a fundamental distinction between model theory (structure) and process (computation). But the field of Programming Language Semantics (PLS) – domain theory and related areas – can be regarded as the model theory of computation. This is more than a slogan. Researchers in programming language semantics employ familiar model theoretic techniques, and have added significantly to the techniques needed to study structure in the context of computation. In this informal abstract, I outline some of the connections between the field of PLS and model theory, paying particular attention to those ideas that I believe can be useful in formal linguistics.

The mid-1980's saw a flurry of activity in formal and computational linguistics around a cluster of ideas that came under the misused name of *unification grammar*. *Information-based Syntax and Semantics* (IBSS), a better term for this cluster of ideas, was coined by Pollard and Sag (1987), yet somehow did not catch on. One of the main organizing ideas of this work, which includes GPSG (Gazdar et al., 1985), HPSG (Pollard and Sag, 1987, 1988, 1992), LFG (Horn, 1983; Kaplan, 1985; Dalrymple, 1992) and others (Gazdar, 1987), was to explore a declarative approach to grammar description, in contrast to a procedural one. Model theoretic syntax (MTS), the topic of this conference, is at least in this one aspect a natural successor of work done under the unification grammar banner.

Another common idea in IBSS that has not carried forward in the MTS community is the use of the mathematical ideas of denotational semantics of programming languages, especially domain theory, to provide formal semantics of grammar formalisms. There are good reasons for this. model theory *simpliciter* is a mature branch of mathematics with a rich set of analytic techniques and with fairly clear connections to other branches of mathematics. Domain theory is less mature and, perhaps as significantly, less known in other mathematical circles. Nevertheless, domain theory (and related approaches to denotational semantics) through the 1990's into the present has developed much more deeply and has begun to find applications in some unexpected areas such as theoretical physics and biology. Roughly, the field is maturing into a general approach to abstract information that has its own analytic techniques to offer researchers in MTS.

This abstract presents an incomplete, idiosyncratic list of interesting developments in the semantics of programming languages of the last several years. The topics discussed here are not limited to Domain Theory, but do arise from its general concern for the interplay of information and computation. The hope is that the discussion offered here will spur a renewed interest in these ideas within the formal linguistics community precisely because they provide links between model theory and computation, allowing the potential to study the relations between (linguistic) structure, (linguistic) information and (linguistic) process.

8.2 Operational semantics

In the field of semantics of programming languages, Plotkin (1977; 2004) pioneered a structural approach to operational semantics as rewriting rules, thus bringing the procedural aspects of semantics under the purview of model theoretic/logical techniques. Although this approach is by now quite mature and should be regarded as part of classical computer science, it is not as well known as it deserves outside the PLS community and continues to bear useful insights. We will see two examples of this below, but in order to understand the basics, a quick review of the approach is useful.

Consider a very simple language or arithmetic terms consisting of numerals for each natural number, $\overline{0}$, $\overline{1}$, etc., a unary operation, $\mathtt{succ}\ E$, and two binary operations, $(E+F)$ and $(E*F)$. For example $(\overline{1}+\mathtt{succ}(\overline{2}*\overline{1}))$ is a term of this language. We can capture the meaning

of terms by specifying how terms can evolve into other terms. This *single step* evolution is given by an inductively defined binary relation $\rightarrow$ on terms. For example, we may have an axiom $\mathtt{succ}\overline{n} \rightarrow \overline{n+1}$. In addition, we have axioms $(\overline{n}+\overline{m}) \rightarrow \overline{n+m}$) and $(\overline{n}*\overline{m}) \rightarrow \overline{nm}$ to capture the operational meaning for successor, addition and multiplication of numerals. For general terms, however, we need five further inference rules to specify how composite terms work:

$$\frac{E \rightarrow E'}{\mathtt{succ}E \rightarrow \mathtt{succ}E'}$$

$$\frac{E \rightarrow E'}{(E+F) \rightarrow (E'+F)}$$

$$\frac{F \rightarrow F'}{(E+F) \rightarrow (E+F')}$$

and similarly for multiplicative terms.

Now let $\rightarrow^*$ denote the reflexive, transitive closure of $\rightarrow$. One easily shows that for any term E, there is a unique numeral $\overline{n}$ so that $E \rightarrow^* \overline{n}$. So we can say that the *meaning* of E is this unique natural number n.

This little example may be trivial, but even here some subtleties arise. For example, the semantics involves a non-deterministic choice whether to evolve E or F in a term $(E+F)$. Nevertheless because of a trivial Church-Rosser property, the final result is unique. If we wanted to specify an operational semantics with deterministic addition, we could replace the second inference rule for addition by the following two axioms:

$$(\overline{0}+F) \rightarrow F$$

$$(\overline{n+1}+F) \rightarrow \mathtt{succ}(\overline{n}+F)$$

The resulting evolution of terms can only reduce the second argument in a term $(E+F)$ by first reducing E to a numeral and then repeatedly applying the second axiom and eventually applying the first. That is, even in such a trivial language, fine control of computation is possible.

One of the advantages of this approach to operational semantics is that it is entirely specified in terms of the formal language itself. One does not need to specify a machine on which programs "execute", and proofs about the semantics use familiar formal language techniques such as establishment of Church-Rosser properties. An alternative to this single-step rewriting approach, known as "big-step" or "natural semantics" was introduced by Kahn in (Kahn, 1987). I will not refer to this approach to operational semantics any further, but the interested reader can find descriptions in any good introduction to PLS.

8.3 Expressivity: Full Abstraction

In classical linguistics, one of the first methods an undergraduate learns is to apply contextual tests to sentence fragments in order to develop a syntactic theory:

Bob eats cookies

* Bob planets cookies

So 'eats' and 'planets' are distinguishable grammatically based on the context 'Bob [ ] cookies.' On the other hand, 'cookies' and 'weasels' are interchangeable (syntactically) in the context 'Bob eats [ ].' This one context does not settle the question of whether 'cookies' and 'weasels' are *indistiguishable* (informally, belong to the same part of speech). So one looks for other contexts that may distinguish between the two. An important point about this is that one well-chosen context can demonstrate that 'eats' and 'planets' are distinguishable, but indistinguishability is harder to show. One must argue that *no* context will do the job.

In PLS, a formal analogue of contextual tests is also useful. Usually, the idea is framed in terms of an operational interpretation of the language (typically specified in the structural rewriting approach discussed above). That is, we are given a set of expressions (called *programs*) and a binary relation $P \rightarrow Q$ on programs that is taken to mean "program P can evolve to program Q" Programs are usually specified by some grammar and the "evolves to" relation is often given by inference rules. In addition, we are given a set of *tests*, each test T consisting of a parametric program construction $T[P]$ (P ranging over programs of a suitable type) and a criterion for when a program Q *passes* the test. Typically, the criterion is simply that Q is some canonical value indicating that no further reductions of Q are needed. But more generally, we can take the criterion to be an arbitrary set of programs (of the same type as $T[P]$). Now we say that P passes a test T provided that $T[P] \rightarrow^* Q$ for some successful value Q.

The analogy with grammaticality tests is fairly strong. We have programs (fragments of language) and tests (fragments with holes along with grammaticality judgements of native speakers). Although we do not have a formally specified "evolves to" relation, we can postulate that the native speak does have some black box operational semantics that "reduces" any particular fragment either to 'grammatical' or to 'ungrammatical.'

The set of tests determines a pre-ordering of programs: $P \leq P'$ if and only if P' passes every test that P passes. Two programs are then *indistinguishable* if and only if $P \leq P'$ and $P' \leq P$. As with linguistic tests, it is generally easy to show that P and P' are distinguishable because one test suffices to witness a difference. On the other hand, without some additional theoretic apparatus (induction at the least) it is impossible to proof that two programs are indistinguishable. This is where a denotational semantics comes into play.

For a denotational semantics, one proposes a category (usually at least a category of partial orders closed under products) in which to interpret the language. A type τ of the language (such as `Nat` for the type of natural numbers) is interpreted as an object $[\![\tau]\!]$; a term E with free variables

is interpreted as a morphism

$$[\![E]\!]\colon [\![\tau_0]\!] \times \ldots [\![\tau_{n-1}]\!] \to [\![\tau_n]\!]$$

where τ_0 through τ_{n-1} are the types of the free variables and τ_n is the type of the term E. Now one can state very simply when two terms are indistiguishable denotationally: $[\![E]\!] = [\![F]\!]$. Furthermore, one can state when one term denotationally *refines* another: $[\![E]\!] \sqsubseteq [\![F]\!]$.

Full abstraction refers to a coincidence between an operational semantics and a denotational semantics: a denotational semantics is said to be *fully abstract* with respect to a given operational semantics provided any two programs P and Q are distinguishable denotationally if and only if they are distinguishable operationally.

The idea of full abstraction was first discussed in Milner (1977) where Milner constructed a fully abstract semantics for a non-trivial language shortly after that in 1977. However, Milner's semantics was constructed by taking quotients of sets of terms, so it was essentially a syntactic construction itself and did not satisfy the semantics community. For nearly twenty years, the Full Abstraction Problem, that is, the problem of constructing a non-trivial fully abstract denotational semantics remained open for non-trivial languages. In 1993, Hyland and Ong (????), and independently Abramsky, Jagadeesan, Malacaria (????) gave a semantics of PCF in which programs are interpreted as certain kinds of games. The details are not important here, but essentially a program involves two players, one of which plays "inputs", and the other plays "outputs." Composition of games is defined by having the middle player pass his output of the first game as input to the second game.

Whether or not one is satisfied with these particular resolutions of the Full Abstraction Problem, the ideas and proof techniques may proof to be useful in other areas that study non-trivial relations between syntactic constructions. In particular, MTS may find these ideas to be useful. I will not be very technical here, but a brief outline of the strategy for proofing full abstraction will give the reader some indication of why I think MTS would benefit from importing these ideas. To simplify, I am suppressing the details about types in this discussion.

Test (pre)congruence: For all programs P and Q and all tests T, $[\![P]\!] \sqsubseteq [\![Q]\!]$ implies $[\![T[P]]\!] \sqsubseteq [\![T[Q]]\!]$.

A proof of this demonstrates that our denotational semantics respects the behavior of tests.

Adequacy: For each test with its notion of "successful state," there exists an element a such that, for all programs P, $a \sqsubseteq [\![P]\!]$ if and only if $P \to^* Q$ for some successful state Q.

There are variations on this formulation of adequacy, but the essential idea is that the denotational interpretation has enough structure to say when tests are passed operationally.

Soundness: For any two programs P and Q, if $[\![P]\!] \sqsubseteq [\![Q]\!]$ then $P \leq Q$.

This is the conjunction of test congruence and adequacy.

Definability: For any element a, there is a set of tests $\{T_i\}$ such that, for any program P, $a \sqsubseteq [\![P]\!]$ if and only if P passes all tests T_i.

In fact, one typically shows that the sets of tests that arise here have more structure.

Completeness: For any two programs P and Q, if $P \leq Q$ then $[\![P]\!] \sqsubseteq [\![Q]\!]$.

Proof. Suppose $[\![P]\!] \not\sqsubseteq [\![Q]\!]$. By definability, there is a set of tests $\{T_i\}$ so that $[\![P]\!] \sqsubseteq [\![R]\!]$ if and only if R passes one the tests T_i. In particular Q fails at least one of these and yet P succeeds them all. Hence $P \not\leq Q$. □

These proof ideas should look familiar to anyone interested in model theory. The point here is that classical model theoretic techniques adapt to a computational setting, provided we get the analogies right. In this case, "soundness" and "completeness" are understood relative to the structural operational semantics as opposed to being relative to the structural notion of provability.

8.4 Inherent parallelism

One of the computational ideas that strikes most people as vaguely, somehow important to linguistics is the distinction between sequential and parallel computation. Whatever else we know about language faculty we can be fairly certain it employs parallel (not to mention stochastic) processes. The question is whether or not parallel processes are characteristic of language or simply an artifact of the implementation. But we immediately hit a stumbling block to framing such a question because most models of computation do not support a notion of "inherent parallelism." That is, the automata themselves that provide the model of computation either are or not models of parallel processing themselves. So it is difficult to say that a particular process implemented on a particular kind of parallel machine is inherently parallel. At the Turing machine level, for example, we know that non-deterministic and well as parallel computation can be emulated on a non-parallel, deterministic machine. So inherency seems to be a vacuous idea. With structural approach to operational semantics, because it opens up the possibilities of model theoretic analysis, we can compare specific language features and build a notion of inherent parallelism that is characterized in terms of expressive power.

Consider the usual implementation of `or` in common programming languages. An expression E `or` F is evaluated sequentially by first evaluating E, and then, only if E returns `false`, evaluating F. This means, e.g., that E `or true` is not generally equal to `true`, in light of the fact that E may not terminate. In fact, it turns out that the potential of non-termination means that there can not be a sequential interpretation of "or" that is commutative.

In operational models of PCF (Plotkin, 1977), a "parallel or" operation `por` characterizes the distinction between sequential and parallel processes. The operational meaning of E `por` F is governed by the following rules:

- `true` por $F \to$ `true`.
- E por `true` $\to$ `true`.
- `false` por `false` $\to$ `false`.
- If $E \to E'$ then E por $F \to E'$ por F.
- If $F \to F'$ then E por $F \to E$ por F'.

It is quite easy to see that this is commutative, and that it fails to terminate if and only if the two operands fail to terminate. In terms of implementation, this means that the two sub-expressions are evaluated in parallel with one being short circuited if the other terminates with `true`.

Several papers investigate the expressivity of languages that employ `por` and its variations. In Escardó et al. (2004), Escardó, Streicher and Hofmann prove an interesting result that, under certain assumptions about the semantics of a type for real numbers, the operation of addition on real numbers is inherently parallel. The proof is technical, but the basic idea is to show that `por` is definable in sequential PCF with a suitable ground type for reals and with addition on reals. In other words, the result and its proof should look very familiar to a model theorist. The proof is a reduction showing that a language feature ($+$, in this case) is as expressively powerful as another (`por`). Since publication of that result, research along similar lines has grown rapidly. For example, recent work by Anberree (2007) shows that the Escardó, Streicher and Hofmann result holds under much more general assumptions about the semantics of the real number type. In a different direction, Marcial-Romero and others (Marcial-Romero, 2004; Farjudian, 2004) have developed other characterizations of limited expressivity due to restricting to sequential computation. In still another direction, Marcial-Romero investigates an approach to real number computation that replaces parallel computation with non-deterministic computation. The upshot of this work is that, with the right non-deterministic constructs, one can define deterministic functions like addition "in the limit" without resorting to parallel computation.

Although real number computation is not obviously useful in formal linguistics, the cluster of techniques that address questions of parallel versus sequential, and deterministic versus non-deterministic computation, are of a strikingly model-theoretic character. In particular, the technique of reduction (addition to `por`) yields useful information about the limits of sequential processing from structural considerations. I have long suspected that certain aspects of MP are also inherently parallel in this sense. A MTS analysis that pays attention to operational semantics might, thus, lead to insights into MP. This should not, by the way, lead to a simple-minded criticism of MP, but might be read as suggesting that psychologically plausible linguist theories will allow for inherent parallelism.

8.5 Monads and Effects

In 1998, Eugenio Moggi (1991) exploited the observation that computational effects, e.g., input, output, storage, non-deterministic choice, etc., exhibit the category theoretic structure of monads to provide a systematic treatment of the distinction between value and computation. These ideas have been developed much further since then, particularly in Levy's work on "call by push-value" semantics (Levy, 1999). The monadic approach to computational effects is now almost common place in functional programming language semantics. For example, the Haskel language (Hudak et al., 1998) includes monads as part of the language definition.

The idea is to interpret a programming language in a suitably structured category (usually, assumed to be cartesian) and to model computations in terms of Kleisli triples associated with specific monads. The specification of monads amounts to a specification of the sort of computational effect that is being modeled. A *computational monad* is a functor T together with two natural transformations $\eta\colon 1 \to T$ (called the *unit*) and $\mu\colon T^2 \to T$ (called the *multiplication*) satisfying:

$$\mu_A \circ \mu_{TA} = \mu_A \circ T(\mu_A)$$
$$\mu_A \circ \eta_{TA} = \mu_A \circ T(\eta_A) = \mathsf{id}_{TA}$$

in which each η_A is a monomorphism (a general monad satisfies the equations but does not necessarily have monomorphic unit).

The rough idea is that TA is the object that denotes computations of values in A; the unit η produces computations of values from values; the multiplication allows us to take a computation of a computation TTA and produce a computation TA.

For example, to model storage with computational side effects (in a cartesian closed category), we can think of a computation is taking a store and returning a value together with a modified store. The functor of this monad is $A \mapsto (A \times S)^S$ where S is some fixed object that models stores. The unit is the exponential transpose of identity on $(A \times S)$. Multiplication takes suitable transposes of "function application."

Now any (computational) monad T determines a category $\mathcal{C}_T$ of *T-computations*. The objects are the objects of the original category, the morphisms from A to B in $\mathcal{C}_T$ are morphisms $A \to TB$ in the original category. Identity $A \to A$ is η_A and composition is given by $g \bullet f = \mu \circ T(g) \circ f$. The reader is referred to Moggi's paper for an explanation of why this makes sense, but intuitively, we associate each value object A with its object of computations, TA.

Thus a distinction is made between a value type (an object A of the category) and type for computations on that type (the object TA). Programs, then, are morphisms that take values and produce computations: $A \to TB$. Composition of programs takes account of the apparent "type mismatch" by way of the monadic multiplication.

For an MTS audience, the importance of monadic interpretations of computational effects is that they provide modularity in the analysis of semantics. That is, a language can support more than one monad, modeling more that one effect. Moreover, the adjective "computational" is only suggestive. Monads that have no particular computational justification are amenable to this semantic treatment. Specifically, "linguistic effects," such as move-α and affect-α, the derivational mechanisms of categorial grammar and perhaps aspects of MP may be amenable to treatment in this "value/effect" set up.

For a rather simple, and trivial, example, tree adjoining can be modeled as a monad in the category in which objects are trees and morphisms are structure preserving maps between trees. The point is that model theoretic approaches to syntactic theory that seek to understand how structure and derivation are related may benefit from Moggi's analysis.

8.6 Stone Duality

Stone's original papers, Stone (1936, 1937), were concerned with the representation of Boolean algebras as subalgebras of powersets. Stone, being primarily interested in analysis, had Boolean rings, not applications to logic, in mind. Nevertheless his results have provided very useful insights into the basic methods of model theory, and more recently have had deep influences on domain theory. For the record, his original results can be stated as follows:

Theorem 8.6.1 (Stone 1937). *Every (bounded) distributive lattice is isomorphic to a (bounded) sublattice of a powerset.*

As stated here, this covers the classical case of Boolean lattices because complementation in a bounded distributive lattice is characterizable equationally. In fact, Stone proved much more.

Recall that a non-empty subset I of a lattice L is an *ideal* if and only if (i) $a \in I$ implies $a \wedge b \in L$ for all $b \in L$ and (ii) $a \in I$ and $b \in I$ implies $a \vee b \in I$. A filter is defined dually by swapping $\wedge$ and $\vee$. An ideal is *prime* if it is inaccessible by meets: $a \wedge b \in I$ implies wither $a \in I$ or $b \in I$. Prime filters are similarly inaccessible by joins. It is easy to see that in a bounded distributive lattice an ideal is prime exactly when its complement is a filter. So complementation sets up a bijection between the prime ideals and prime filters.

The following is traditionally stated in terms of prime ideals (with terminology borrowed from ring theory) and referred to as the *Prime Ideal Theorem* or "PIT". However, because of the foregoing observation there is no real difference between using ideals or filters.

Lemma 8.6.2. *In any distributive lattice, if $x \not\leq y$ then there is a prime filter containing x but not y.*

Stone's construction of a representation of a given Boolean (or bounded distributive) lattice L takes prime filters of the lattice as points of a topological space $\mathsf{spec}(L)$, known as the *spectrum* of L. The topology of the spectrum is generated by basic opens U_a for each $a \in L$, where a prime filter P belongs to U_a if and only if $a \in P$. Then an isomorphic copy of L is recovered from $\mathsf{spec}(L)$ by taking the sub-lattice of opens of the form U_a. These happen to be precisely the compact opens of the space. Stone's more elaborate result is the following.

Theorem 8.6.3 (Stone 1936, 1937).

1. *The spectra of Boolean algebras are precisely the totally disconnected compact Hausdorff spaces.*
2. *The spectra of bounded distributive lattices are precisely those topological spaces that are*
   (a) *T_0;*
   (b) *compact;*
   (c) *zero-dimensional, that is, have a basis consisting of compact open sets;*
   (d) *coherent, that is, intersections of compact open sets are compact;*[1]
   (e) *well filtered, that is, if the intersection of a filter base $(C_i)_{i\in I}$ of compact open sets is contained in an open set, then so is some C_i already.*

The totally disconnected compact Hausdorff spaces are now called *Stone spaces*; those spaces that correspond to bounded distributive lattices are now called *spectral spaces*. Without having the benefit of category theoretic terminology, in the same papers Stone essentially showed that the category of Boolean lattices is equivalent to the dual of the category of stone spaces with continuous maps, and similarly the category of bounded distributive lattices is equivalent to the dual of the category of spectral paces with continuous maps for which preimages of compact subsets are compact (these are called variously *pure* or *perfect* maps).

Stone duality has an apparent logical reading. Every Boolean algebra can be viewed as the *Lindenbaum algebra* of a propositional theory. A prime filter on the algebra corresponds to a *model* of the theory. The Prime Ideal Theorem (our Lemma 8.6.2) then states the completeness of propositional logic. Indeed the choice principle that is needed to proof PIT is essentially the same as what is used to proof completeness theorems. The duality of bounded distributive lattices likewise corresponds to propositional logic without negation, or *positive logic*.

So we can regard certain topological spaces as spaces of models of classical (or positive) logic. This idea is exploited in domain theory extensively where spectral spaces have turned out to play a very natural role because the Scott topologies of many familiar classes of domains are spectral. That is, each domain has a naturally associated positive logic. Similarly, under some assumptions about the domains in question, positive logics that satisfy

[1] This is also called *stably compact*.

additional conditions give rise to domains. The classic paper that develops this connection is Abramsky's "Domain Theory in Logical Form." (Abramsky, 1987a,b). The idea is further developed for continuous domains and in a more purely logical setting in Kegelmann (2002); Jung et al. (1999).

Stone duality certainly pre-dates these applications to programming language semantics by several decades and has applications in many other areas. Nevertheless, denotational semantics shows how productive the application of topological ideas can be in an area (computation) that seems, at first look, to be entirely in the purview of discrete mathematics.

Through the lens of Stone duality, Smyth (1983) proposed a correspondence between domain theoretic and topological notions. This correspondence has come to be known as *Smyth's dictionary*. Beginning with the notion that a data type may be modeled as some sort of structured set, Smyth argues that a semi-decidable property corresponds naturally to a topologically open set of data. That is, a property P is semi-decidable if there is a Turing machine that halts on input d if and only if $P(d)$. Now, if we have two semi-decidable properties P and Q, their conjunction is also semi-decidable (by sequentially running each on the input). Likewise, if we have a family $\{P_i\}$ of semi-decidable properties, then there infinite disjunction is semi-decidable by dovetailing the execution of the corresponding machines and halting when any one of them halts. So semi-decidable properties, being closed under finite intersection and arbitrary union, form a topology. Furthermore, a program f that takes input of type σ to output of type τ corresponds to a continuous function because f transforms any semi-decidable property P of outputs into a semi-decidable property of inputs: execute f on input d and then (semi)-decide if $P(f(d))$ holds – this is a semi-decidable property of data in σ. The following dictionary entries summarize these correspondences so far.

Data type	Topological space
Datum	point
Semi-decidable property	open set
Computable function	continuous function

More recently, Escardó (2004) has extended this dictionary to show that compact subsets of a topological space correspond naturally to subsets over which universal quantification is continuous (according to the dictionary, computable). This analysis of compact sets as universally quantifiable sets is strongly reminiscent of some of the model theoretic semantics research of Moss and Keenan (1984; 1985) and others of the mid-1980's. Furthermore, the general approach of using the notions of Stone duality to obtain correspondences between an application area and classical topological notions should proof of interest to researchers in MTS.

8.7 Conclusion

In this all too brief discussion, I have merely highlighted some areas of potential contact between model theoretic syntax and semantics of programming languages. The main moral of the story, however, is that semantics of computation has developed model theoretic (and other) approaches to the structure of computation. So it is not necessary to accept the dichotomy on between structure and process on which model theoretic syntax partly stands. Plotkin's structural approach to operational semantics allows for application of traditionally model theoretic techniques to computation. Recent analysis of inherent parallelism is an exemplary application of these ideas. On a somewhat different tack, Moggi's category theoretic approach to computational effects has the potential for application in the analysis of formalizations of theories of syntax. Finally, Stone duality shows that topology plays an unavoidable role in semantics, and also provides productive analytic techniques that have lead to the discovery of new connections between topology and computation.

It is my hope that this taste will interest active researchers in model theoretic syntax to rediscover the connections to programming language semantics that showed promise twenty years ago.

Bibliography

Abramsky, S. (1987a). *Domain Theory and the Logic of Observable Properties*. Ph.D. thesis, University of London.

Abramsky, S. (1987b). Domain theory in logical form. In *Symposium on Logic In Computer Science*, pp. 47–53. IEEE Computer Society Press.

Abramsky, S., R. Jagadeesan, and P. Malacaria (????). Games and full abstraction for PCF, preliminary announcement. Note distributed August 1993.

Anberree, T. (2007). On the non-sequential nature of domain models of real-number computation.

Dalrymple, Mary (1992). Categorial semantics for LFG. In *Proceedings of the International Conference on Computational Linguistics (COLING '92)*. Nantes, France.

Escardó, M. H. (2004). Synthetic topology of data types and classical spaces. In J. Desharnais and P. Panangaden, eds., *Domain-theoretic Methods in Probabilistic Processes*, volume 87 of *Electronic Notes in Theoretical Computer Science*, pp. 21–156. Elsevier Science Publishers B.V.

Escardó, M. H., M. Hofmann, and Th. Streicher (2004). On the non-sequential nature of the interval-domain model of real-number computation. *Mathematical Structures in Computer Science*, **14**:803–814.

Farjudian, A. (2004). *Sequentiality in real number computation*. Ph.D. thesis, University of Birmingham.

Gazdar, Gerald (1987). The new grammar formalisms - a tutorial survey. *IJCAI-87*, **2**:1172.

Gazdar, G., E. Klein, G. Pullum, and I. Sag (1985). *Generalized Phrase Structure Grammar*. Harvard University Press.

Horn, George Michael (1983). *Lexical-Functional Grammar*. Mouton de Gruyter, Berlin.

Hudak, P., J. Peterson, and J. Fasel (1998). A gentle introduction to haskel.

Hyland, M. and L. Ong (????). Dialogue games and innocent strategies: An approach to intensional full abstraction for PCF (preliminary announcement). Cambridge University, UK. Note distributed August 1993.

Jung, A., M. Kegelmann, and M. A. Moshier (1999). Multi lingual sequent calculus and coherent spaces. *Fundamenta Informaticae*, **37**:369–412.

Kahn, G. (1987). Natural semantics. *Lecture Notes in Computer Science*, **247**:22–39.

Kaplan, Ronald M. (1985). Lexical-functional grammar vs. logic grammar. In *Theoretical Approaches to Natural Language Understanding, a Workshop at Halifax, Nova Scotia*.

Kegelmann, M. (2002). *Continuous Domains in Logical Form*, volume 49 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers B.V.

Levy, Paul Blain (1999). Call-by-push-value: A subsuming paradigm. In *Typed Lambda Calculus and Applications*, pp. 228–242.

Marcial-Romero, J. R. (2004). *Semantics of a sequential language for exact real-number computation*. Ph.D. thesis, University of Birmingham.

Milner, R. (1977). Fully abstract models of typed lambda-calculi. *Theoretical Computer Science*, **4**:1–22.

Moggi, Eugenio (1991). Notions of computation and monads. *Information and Computation*, **93**(1):55–92.

Moss, L. and E. Keenan (1984). Generalized quantifiers and the logical expressive power of natural language. In et al M. Cobler, ed., *Third West Coast Conference on Formal Linguistics*, pp. 149–157. Stanford Linguistics Association.

Moss, L. and E. Keenan (1985). Generalized quantifiers and the expressive power of natural language. In J. van Benthem and A. ter Meulen, eds., *Groningen-Amsterdam Series in Semantics No. 4*, pp. 73–124. Foris.

Plotkin, G. D. (1977). LCF considered as a programming language. *Theoretical Computer Science*, **5**:223–255.

Plotkin, G. D. (2004). A Structural Approach to Operational Semantics. **61–62**:17–139. Original version: University of Aarhus Technical Report DAIMI FN-19, 1981.

Pollard, Carl J. and Ivan A. Sag (1987). *Information-based Syntax and Semantics, Vol. 1*. Number 13 in CSLI Lecture Notes. CSLI Publications, Stanford University. Distributed by University of Chicago Press.

Pollard, Carl J. and Ivan A. Sag (1988). An information-based theory of agreement. In D. Brentari and al, eds., *Papers on the Parasession on Agreement from the 24th Regional Meeting of the Chicago Linguistic Society*, volume 24. CLS, CLS, Chicago, Illinois. Also published in slightly revised form as CSLI Report 88-132. Stanford: Center for the Study of Language and Information.

Pollard, Carl J. and Ivan A. Sag (1992). Hpsg: Binding theory. In Byung-Soo Park, ed., *Linguistic Studies on Natural Language*, volume 1 of *Kyunghee Language Institute Monograph*. Hanshin, Seoul, Korea.

Smyth, M. B. (1983). Powerdomains and predicate transformers: a topological view. In J. Diaz, ed., *Automata, Languages and Programming*, volume 154 of *Lecture Notes in Computer Science*, pp. 662–675. Springer Verlag.

Stone, M. H. (1936). The theory of representations for Boolean algebras. *Trans. American Math. Soc.*, **40**:37–111.

Stone, M. H. (1937). Applications of the theory of Boolean rings to general topology. *Trans. American Math. Soc.*, **41**:375–481.

Chapter 9

An Automata-Theoretic Approach to Minimalism

Gregory M. Kobele, Christian Retoré and Sylvain Salvati
INRIA Futurs, LaBRI (Université de Bordeaux – CNRS)
Bordeaux, France
kobele|retore|salvati@labri.fr

Vijay-Shanker et al. (1987) note that many interesting linguistic formalisms can be thought of as having essentially context-free structure, but operating over objects richer than simple strings (sequences of strings, trees, or graphs). They introduce linear context-free rewriting systems (LCFRS's, see also Weir (1988)) as a unifying framework for superficially different such formalisms (like (multi component) tree adjoining grammars, head grammars, and categorial grammars). Later work (Michaelis, 1998) has added minimalist grammars (MGs, see (Stabler, 1997)) to this list. Recently, Fülöp et al. (2004) have introduced multiple bottom-up tree transducers (mbutt), which can be thought of as offering a transductive perspective on LCFRSs. The transductive perspective allows us to view a grammar in one of these grammar formalisms as defining both a set of well-formed derivations, and functions which interpret these derivations as the derived structures (trees, strings, or meanings) they are derivations of. Being explicit about the structure of the derivation, and divorcing it from the construction of the object so derived has two main advantages. First, we may entertain and study the effects of modifications to the structure of the derivational process, such as insisting that a particular operation apply only in case there is an isomorphic subderivation somewhere in the same derivation (for example, in deletion under identity with an antecedent), or other non-local filters on well-formed derivations, without worrying about the kinds of data structures that would be required to support such operations in real-time (as in parsers, for example). Secondly, viewing derivational grammar formalisms in this way makes particularly salient two loci of language theoretic complexity:

1. the set of well-formed derivation structures
2. the transformation from derivation structures to derived structures

Taking this latter perspective, Shieber (2006) shows that TAGs are exactly characterized in terms of monadic macro tree transducers simple in both the input and the parameters (1-MTT$_{si,sp}$) (Engelfriet and Vogler, 1985) acting on a regular tree language (see also Mönnich (1997)).

Minimalist grammars offer a formal perspective on some of the core ideas in Chomsky's minimalist program (Chomsky, 1995) (various extensions to the core formalism have been proposed and investigated; a variant with copying was introduced and studied in (Kobele, 2006)). We show in this paper how, given a minimalist grammar G, to construct a simple, regular, characterization of its well formed derivations. Furthermore, given the close connection between LCFRSs and mbutts, it is straightforward to construct a linear deterministic mbutt which maps derivation trees to the structures they are derivations of. Deterministic mbutts were proven in Fülöp et al. (2004) to be equivalent to deterministic top-down tree transducers with regular look-ahead (dTT^R), and it was conjectured that adding linearity to the mbutt corresponded to restricting the dTT^R to be finite copying. We prove half of this conjecture in the appendix: linear deterministic mbutts (ldmbutt) can be simulated by finite copying deterministic top-down tree transducers with regular look-ahead (dTT^R_{fc}).[1] We obtain thus both a bottom-up and a top-down characterization of the function from minimalist derivations to derived trees. The same construction extends to minimalist grammars with copying simply by removing the finite copying restriction (dTT^R). In other words, the structure languages generated by minimalist grammars with (without) copying are contained in the output languages of (finite copying) tree homomorphisms.

We can immediately conclude that, although the string languages generated by minimalist grammars properly include those generated by TAGs,[2] the same is not true of

[1] Michaelis et al. (2001) have provided a different characterization of the derived trees definable by minimalist grammars (see also Morawietz (2003)). Given a minimalist grammar, they define a regular tree grammar which encodes the operations of an equivalent LCFRS as operation symbols in a lifted signature. From there, they show that one can obtain the desired trees using a monadic second order logic transduction, a MTT simple in the input and the parameters, or a deterministic tree walking automaton. As we think the derivation tree is an interesting object in its own right (as per our introductory comments), we prefer to start from there. Our obtained transducer class is different in non-trivial ways as well, with MSO and simple MTTs able to define transductions which $dTT^R_{(fc)}$s cannot.

[2] MGs were proven to be equivalent to multiple context-free grammars (Seki et al., 1991) in (Michaelis, 1998; Harkema, 2001; Michaelis, 2001). The variant with copying is equivalent to parallel multiple context-free grammars (Seki et al., 1991), see (Kobele, 2006). TAGs

their respective structure languages, as the output languages of deterministic (finite copying) tree transducers are incomparable to those of 1-MTT$_{si,sp}$s (Engelfriet and Maneth, 2000). An example of a derived TAG tree language that is not also generable by an MG is $\{a^n(b^n(e)) : n \geq 1\}$ (as monadic languages which are the output of a regular tree transducer are all recognizable).

Tree transducers can also be used to characterize transformations of trees into non-tree-like structures, such as graphs, or even arbitrary algebras (Bauderon and Courcelle, 1987; Engelfriet, 1994). The idea is to encode elements of the algebra as trees, and to 'decode' the tree $\tau(t)$, for input tree t and transducer τ, into the algebraic object it represents (this is the idea behind the common 'tree-to-string' mappings). For instance, we might interpret the derived objects not as strings, but rather as partially ordered multi-sets, as proposed in Pan (2007), which allows for an elegant statement of otherwise quite difficult to describe (Bobaljik, 1999) word order regularities in languages like Norwegian. Compositionality, the principle that the meaning of an object is determined by the meanings of its immediate parts and their mode of combination, is naturally formulated as a transduction mapping derivation trees to (terms denoting) semantic values. The compositional semantics for minimalist grammars introduced in Kobele (2006) is naturally expressed in terms of a transduction of the same type as that mapping derivations to derived trees (a $dTT^R_{(fc)}$). We present a general method of synchronizing (in the sense of Shieber (1994)) multiple transductions over the same derivation, showing as a result that the form-meaning relations definable by MGs interpreted as per Kobele (2006) can be described as bimorphisms of type $\mathbf{B}(\mathbf{M},\mathbf{M})$ (in the terminology of Shieber (2006)).

The rest of this paper is structured as follows. After some mathematical preliminaries, we introduce minimalist grammars. We then define the derivation tree language of a minimalist grammar, and prove that it is regular. We then introduce multi bottom-up tree transducers, and show that one can therewith transform a minimalist derivation tree into the derived structure it represents a derivation of. Finally, bimorphisms are introduced, and the form-meaning relations generable by minimalist grammars are shown to be contained within the bimorphism class **B**(**M**,**M**) (**M** is the set of unrestricted homomophisms). In the appendix, the linear deterministic multi bottom-up tree transducers used in this paper to establish the above results are shown to be included in the top-down tree transducers with regular look-ahead and finite copying, as conjectured by Fülöp et al. (2005). At the end we include a picture which contains some of the wealth of information on tree languages generated by various grammars and devices.

are equivalent to a proper subclass of multiple context-free grammars (Seki et al., 1991).

9.1 Preliminaries

The set of natural numbers will be denoted by $\mathbb{N}$ and $[n]$ will denote the set $\{1,\ldots,n\}$ with the convention that $[0]$ represents the empty set.

Σ^* is the set of all finite sequences of elements of Σ. Σ^+ is the set of all non-empty such sequences, and ε is the empty sequence. The length of a sequence w is denoted $|w|$, and $|\varepsilon| = 0$. For a non-empty sequence aw, $a \in \Sigma$ is its head and w its tail.

A ranked alphabet Ω is a finite set (also denoted Ω) together with a function $\mathbf{rank} : \Omega \to \mathbb{N}$ assigning to each $\omega \in \Omega$ its rank. The notation $\Omega^{(n)}$, for $n \in \mathbb{N}$, denotes the set $\{\omega \in \Omega : \mathbf{rank}(\omega) = n\}$ of symbols of rank n. Given $\omega \in \Omega^{(n)}$, we sometimes write $\omega^{(n)}$ to remind us that $\mathbf{rank}(\omega) = n$. The set of trees built on a ranked alphabet Ω, noted T_Ω, is the smallest set that contains $\Omega^{(0)}$ and $\omega(t_1,\ldots,t_n)$ iff for all $i \in [n]$, $t_i \in T_\Omega$.

9.2 Minimalist Grammars

An idea common to many grammar formalisms is that natural languages are resource sensitive, in the sense that grammatical operations consume resources when applied. Minimalist grammars implement this idea in terms of *features*, which are deleted or *checked* as operations are applied. Syntactic features come in two varieties: **licensing** features and **selection** features, which are relevant for the grammatical operations of **move** and **merge** respectively. Each feature type has a positive and a negative polarity. The set of licensing features is **lic**, and for $x \in$ **lic**, +x is the positive, and -x the negative polarity feature of type x. The set of selection features is **sel**, and for $x \in$ **sel**, =x is the positive, and x the negative polarity feature of type x. We assume without loss of generality that **lic** and **sel** are disjoint. $\mathbb{F} = \{+x, -x, =y, y : x \in \mathbf{lic}, y \in \mathbf{sel}\}$ is the set of all positive and negative polarity features of all types. An expression $\phi = \phi_0, \phi_1, \ldots, \phi_n$ is a finite sequence of pairs $\phi_i = \langle t_i, l_i \rangle$ of trees t and sequences of features l.[3] The intuition is that the grammatical operations combine trees in various ways based on the features that are associated with these trees. Given an alphabet Σ and a symbol $\varepsilon \notin \Sigma$ we will interpret as the empty string (the set $\Sigma \cup \{\varepsilon\}$ is denoted Σ_ε), the tree components of a minimalist expression have internal nodes labelled with either < or > (indicating that the head of the tree as a whole is the head of the left or right subtree respectively), and leaves labelled with either t (a 'trace') or elements of Σ_ε. These labels form a ranked alphabet $\Omega = \{<^{(2)}, >^{(2)}, t^{(0)}\} \cup \Sigma_\varepsilon$, where each $\sigma \in \Sigma_\varepsilon$ has rank 0. In lexicalized grammar formalisms like MGs, the grammatical operations are held constant across grammars, the locus of variation being confined to different choices of lexical items. A *lexicon* is a finite set

[3] This is the 'chain-based' presentation of MGs (Stabler and Keenan, 2003), but with trees, and not strings, as the derived objects. The possibility of such a representation was first noticed by Michaelis (1998), who used it to prove the containment of the minimalist languages in the MCFGs.

$Lex \subset \Sigma_\varepsilon \times \mathbb{F}^+$. The grammatical operations **move** and **merge**, common to all minimalist grammars, are defined as per the following. The unary operation **move** is defined on an expression $\phi_0, \ldots, \phi_n$ just in case the head of the feature sequence of ϕ_0 is a positive polarity licensing feature type $+\mathtt{x}$, and there is exactly one ϕ_i the head of whose sequence is the corresponding negative feature $-\mathtt{x}$ (the requirement that ϕ_i be unique is called the SMC, and results in an upper bound of $|\mathbf{lic}|+1$ on the length of useful expressions). The definition of **move** is given in two cases, as per whether the moving element has exactly one (**move1**), or more than one (**move2**) feature in its feature sequence. The tree denoted by $f(t_1,t_2)$ is $<(t_1,t_2)$ if $t_1 \in \Sigma_\varepsilon$, and is $>(t_2,t_1)$ otherwise. For l_i non-empty,

$$\begin{aligned}\mathbf{move1}(\langle t_0, +\mathtt{x} l_0\rangle, \ldots, \langle t_i, -\mathtt{x}\rangle, \ldots, \phi_n) \\ = \langle f(t_0,t_i), l_0\rangle, \ldots, \phi_n\end{aligned}$$

$$\begin{aligned}\mathbf{move2}(\langle t_0, +\mathtt{x} l_0\rangle, \ldots, \langle t_i, -\mathtt{x} l_i\rangle, \ldots, \phi_n) \\ = \langle f(t_0,\mathtt{t}), l_0\rangle, \ldots, \langle t_i, l_i\rangle, \ldots, \phi_n\end{aligned}$$

The binary operation **merge** is defined on expressions ϕ and ψ just in case the head of the feature sequence of ϕ_0 is a positive polarity selection feature $=\mathtt{x}$, and the head of the feature sequence of ψ_0 is the corresponding negative feature $\mathtt{x}$. As before, we split the definition of **merge** into two cases, based on whether the feature sequence of ψ_0 contains exactly one (**merge1**) or more than one (**merge2**) feature. For l'_0 non-empty,

$$\begin{aligned}\mathbf{merge1}(\langle t_0, =\mathtt{x} l_0\rangle, \phi_1, \ldots, \phi_m;\ \langle t'_0, \mathtt{x}\rangle, \psi_1, \ldots, \psi_n) \\ = \langle f(t_0,t'_0), l_0\rangle, \phi_1, \ldots, \phi_m, \psi_1, \ldots, \psi_n\end{aligned}$$

$$\begin{aligned}\mathbf{merge2}(\langle t_0, =\mathtt{x} l_0\rangle, \phi_1, \ldots, \phi_m;\ \langle t'_0, \mathtt{x} l'_0\rangle, \psi_1, \ldots, \psi_n) \\ = \langle f(t_0,\mathtt{t}), l_0\rangle, \phi_1, \ldots, \phi_m, \langle t'_0, l'_0\rangle, \psi_1, \ldots, \psi_n\end{aligned}$$

Given an alphabet Σ, a minimalist grammar G over Σ is given by its set of features $\mathbb{F}$, a lexicon *Lex*, and a designated feature $\mathtt{c} \in \mathbb{F}$ (the type of sentences). The expressions generated by a minimalist grammar $G = \langle \mathbb{F}, Lex, \mathtt{c}\rangle$ are those in $CL(Lex) = \bigcup_{n\in\mathbb{N}} CL_n(Lex)$, where[4]

$$\begin{aligned}CL_0(Lex) &= Lex \\ CL_{n+1}(Lex) &= CL_n(Lex) \\ &\quad \cup \{\mathbf{move}(\phi) : \phi \in CL_n(Lex)\} \\ &\quad \cup \{\mathbf{merge}(\phi,\psi) : \phi,\psi \in CL_n(Lex)\}\end{aligned}$$

An expression $\phi = \phi_0, \ldots, \phi_n$ is complete iff $n = 0$. The structure language $S(G) = \{t : \langle t, \mathtt{c}\rangle \in CL(Lex)\}$ generated by G is the set of tree components of complete expressions whose feature sequence component is the designated feature $\mathtt{c}$.

[4] It is implicitly assumed that the arguments presented to the generating functions are restricted to those in their domains.

9.3 Derivations as trees

Given a minimalist grammar over Σ, $G = \langle \mathbb{F}, Lex, \mathtt{c}\rangle$, its derivation trees are defined to be the terms over the ranked alphabet $\Gamma = \{\mathbf{mrg}^{(2)}, \mathbf{mv}^{(1)}\} \cup Lex$, where the elements of *Lex* have rank 0. A derivation tree $t \in T_\Gamma$ is *a derivation of* an expression ϕ just in case $\phi = h(t)$, where h maps lexical items to themselves, and $h(\mathbf{mv}(t)) = \mathbf{move}(h(t))$ and $h(\mathbf{mrg}(t_1,t_2)) = \mathbf{merge}(h(t_1),h(t_2))$. As the functions **merge** and **move** are partial, so is h. We can identify the set of *convergent* (well-formed) derivation trees with the domain of h.

The first question we ask is as to the language theoretic complexity of the set of well-formed derivation trees of complete expressions. We will show (by exhibiting the automaton) that this set is the language accepted by a bottom-up tree automaton; in other words, a regular tree language. A bottom-up tree automaton (BA) is a structure $A = \langle Q, Q_f, \Sigma, \delta\rangle$, where Q is a finite set of states, $Q_f \subseteq Q$ the set of final states, Σ is a ranked alphabet, and $\delta = (\delta_\sigma)_{\sigma\in\Sigma}$ is a family of partial functions $\delta_\sigma : Q^{\mathbf{rank}(\sigma)} \to 2^Q$ from $\mathbf{rank}(\sigma)$-tuples of states to sets of states. If for every $\sigma^{(n)} \in \Sigma$, and for every $q_1, \ldots, q_n \in Q$, $|\delta_\sigma(q_1,\ldots,q_n)| \leq 1$ then A is deterministic, and we write $\delta_\sigma(q_1,\ldots,q_n) = q$ for $\delta_\sigma(q_1,\ldots,q_n) = \{q\}$. For a term $\sigma^{(n)}(t_1,\ldots,t_n) \in T_\Sigma$, $\delta(\sigma(t_1,\ldots,t_n)) = \bigcup\{\delta_\sigma(q_1,\ldots,q_n) : q_i \in \delta(t_i)\}$. A term $t \in T_\Sigma$ is accepted by A just in case $\delta(t) \cap Q_f$ is non-empty.

Theorem 9.3.1. *For $G = \langle \mathbb{F}, Lex, \mathtt{c}\rangle$ a minimalist grammar over an alphabet Σ, and for $l \in \mathbb{F}^+$, the set of convergent derivation trees of complete expressions of type l is a regular tree language.*

Proof. We construct a deterministic bottom up tree automaton $A_G = \langle Q, Q_f, \Gamma, \delta\rangle$ which recognizes just the convergent derivations in T_Γ of complete expressions of type l. Any set accepted by such an automaton is regular, whence the conclusion. The states of our automaton will keep track of the featural components of the expression $h(t)$ that the derivation tree t is a derivation of. To bring out the logical structure of the feature calculus (and thereby simplify the statement of the transition function), instead of working with arbitrary sequences of feature sequences (the right projection of minimalist expressions) we represent the features had by an expression ϕ as an $n+1$-ary sequence of feature sequences, with $n = |\mathbf{lic}|$ (recall that the SMC condition on **move** ensures that no expression that is part of a convergent derivation of a complete expression has more than one subpart ϕ_i with feature sequence beginning $-\mathtt{x}$, for any $x \in \mathbf{lic}$). Moreover, an arbitrary but fixed enumeration of **lic** allows us to denote licensing feature types with positive integers (thus $+1$ denotes a positive polarity feature of the first licensing feature type), and we require that the i^{th} component of our states, if non-empty, contain a feature sequence beginning with $-\mathtt{i}$. Formally, for $\mathtt{suf}(Lex) := \{\beta : \langle \sigma, \alpha\beta\rangle \in Lex\}$ the set of suffixes of lexical feature sequences, we define

our set of states such that

$$Q := \{\langle s_0,\ldots,s_n\rangle : s_0,\ldots,s_n \in \mathtt{suf}(Lex) \text{ and for } 1 \le i \le n \text{ either } s_i = \varepsilon \text{ or } s_i = \mathtt{-i}\alpha\}$$

The set of final states Q_f is the singleton $\{\langle l,\varepsilon,\ldots,\varepsilon\rangle\}$ It remains to describe the action of the transition function on states. To make the description of the results of these functions easier, we define the partial binary operation over feature sequences $\oplus$ ('sum') which is defined just in case at least one of its arguments is ε, and returns its non-empty argument if one exists, and ε otherwise. We extend $\oplus$ to a function which takes a state $q = \langle s_0,\ldots,s_i,\ldots\rangle$ and a feature sequence s and returns q if $s = \varepsilon$ and $\langle s_0,\ldots,(s_i \oplus s),\ldots\rangle$ if $s = \mathtt{-i}s'$ (otherwise, $\oplus$ is undefined). The transition function $\delta_{\mathbf{mv}}$ is defined on a state $q = \langle s_0,\ldots,s_n\rangle$ just in case the head of the sequence of features in the initial position of is a positive polarity licensing feature (`+i`), the head of the feature sequence in the i^{th} position is the corresponding negative polarity licensing feature (`-i`), and if the tail of the feature sequence in the i^{th} position is non-empty and begins with `-j`, then the j^{th} position is empty. If defined, the result is identical to q, except that the matching i^{th} licensing features are deleted, and the remainder of the feature sequence in the i^{th} array position is moved to the j^{th} array position if it begins with `-j`. Formally,

$$\delta_{\mathbf{mv}}(\langle \mathtt{+i}s_0,\ldots,\mathtt{-i}s_i,\ldots\rangle) = \langle s_0,\ldots,\varepsilon,\ldots\rangle \oplus s_i$$

The transition function $\delta_{\mathbf{mrg}}$ applies to a pair of states just in case the following three conditions are met. First, the heads of the initial feature sequence of the two states must be positive and negative polarity features of the same selection feature type, respectively. Second, whenever a non-initial feature sequence of the first state is non-empty, the corresponding feature sequence in the second state must be empty. Finally, if the tail of the initial feature sequence of the second state begins with `-j`, then the j^{th} position of both states must be empty. If defined, $\delta_{\mathbf{mrg}}(q_1,q_2)$ is the state whose initial component is the tail of the initial component of q_1, whose j^{th} component is the sum of the tail of the initial component of q_2 with the j^{th} components of both input states, and whose non-initial components are the sum of the corresponding non-initial components in q_1 and q_2. Formally,

$$\delta_{\mathbf{mrg}}(\langle \mathtt{=f}s_0,\ldots,s_i,\ldots\rangle, \langle \mathtt{f}s'_0,\ldots,s'_i,\ldots\rangle) = \langle s_0,\ldots,(s_i \oplus s'_i),\ldots\rangle \oplus s'_0$$

Finally, for each lexical item $\langle\sigma,l\rangle$, $\delta_{\langle\sigma,l\rangle}$ is the constant function that outputs the state with initial component l, and all other components ε. Formally,

$$\delta_{\langle\sigma,l\rangle}(\langle\sigma,l\rangle) = \langle l,\varepsilon,\ldots,\varepsilon\rangle$$

A simple induction on derivation trees proves the correctness of the automaton. The only slightly tricky bit stems from the fact that the automaton in effect enforces the SMC at each step, whereas the minimalist grammars 'wait' until a **move** step. This is harmless as once an expression is generated which has more than one component with the same initial `-i` feature it can never be 'rescued' and turned into a complete expression. □

As a special case we obtain

Corollary 9.3.2. *For any MG $G = \langle \mathbb{F}, Lex, \mathtt{c}\rangle$, the set of derivation trees of sentences (complete expressions of category* `c`*) is regular.*

9.4 Interpreting derivations

The picture of minimalist grammars with which we began conflates the structure of the feature calculus with the process of tree assembly. We have seen that by factoring out the tree-building operations from the syntactic feature manipulation, we are left with a simple and elegant system, and that the structure of the feature calculus is underlyingly regular. We can think of the syntactic calculus as delivering blueprints for building trees. We now know that these blueprints themselves have a simple regular structure, but what is left to determine is the complexity of building trees from blueprints.

We will extend the bottom-up automaton from the previous section (which manipulated sequences of feature sequences) so as to allow it to build trees. In minimalist expressions $\phi = \phi_0,\ldots,\phi_m$, each tree t_i is paired with its syntactic features s_i directly. This makes the order of occurrence of the ϕ_is irrelevant. In contrast, in our automata, features are used in the description of states, and thus are dissociated from their trees. Accordingly, we make the objects derived during a derivation $n+1$-ary sequences of trees over the ranked alphabet $\Omega = \{<^{(2)}, >^{(2)}, \mathtt{t}^{(0)}\} \cup \Sigma_\varepsilon$. The connection between a tree and its feature sequence is established by the invariant that the i^{th} component of a state represents the features of the i^{th} tree in a sequence. There are n^2+2n+1 basic operations on $n+1$-ary sequences of trees: $\mathbf{m}^{(2)}$, $\mathbf{m_j}^{(2)}$, $\mathbf{v_i}^{(1)}$, and $\mathbf{v_{i,j}}^{(1)}$, for $1 \le i,j \le n$. These operations form an algebra $\mathcal{S}$ over the carrier set $S = \{\langle t_0,\ldots,t_n\rangle : t_0,\ldots,t_n \in T_\Omega\}$ of $n+1$-ary tree sequences. Intuitively, the operations on tree sequences are indexed to particular cases of the $\delta_{\mathbf{mv}}$ and $\delta_{\mathbf{mrg}}$ functions, and derivations in the syntactic calculus then control tree sequence assembly (as shown in figure 9.1). The operations are defined as per the following, where $t_1 \oplus t_2$ is defined iff at least one of t_1 and t_2 is $\mathtt{t}$, in which case it returns the other one.

$$\mathbf{v_i}(\langle t_0,\ldots,t_i,\ldots\rangle) = \langle f(t_0,t_i),\ldots,\mathtt{t},\ldots\rangle$$

$$\mathbf{v_{i,j}}(\langle t_0,\ldots,t_i,\ldots,t_j,\ldots\rangle) = \langle f(t_0,\mathtt{t}),\ldots,\mathtt{t},\ldots,(t_j \oplus t_i),\ldots\rangle$$

$$\mathbf{m}(\langle t_0,\ldots,t_i,\ldots\rangle, \langle t'_0,\ldots,t'_i,\ldots\rangle) = \langle f(t_0,t'_0),\ldots,(t_i \oplus t'_i),\ldots\rangle$$

operation	case of $\delta_{\mathbf{mv}}/\delta_{\mathbf{mrg}}$
$\mathbf{v_i}$	$s_i = \varepsilon$
$\mathbf{v_{i,j}}$	$s_i = \texttt{-j}l$
$\mathbf{m}$	$s'_0 = \varepsilon$
$\mathbf{m_j}$	$s'_0 = \texttt{-j}l$

Figure 9.1: Operations on tree sequences and the syntactic operations they are associated with

$$\mathbf{m_j}(\langle t_0,\ldots,t_j,\ldots\rangle, \langle t'_0,\ldots,t'_j,\ldots\rangle) = \langle f(t_0,\mathtt{t}),\ldots,(t_j \oplus t'_j)\oplus t'_0,\ldots\rangle$$

Each state $q = \langle s_0,\ldots,s_n\rangle$ is associated with an $n+1$-tuple of trees $\langle t_0,\ldots,t_n\rangle$. We would like the states to be put together in accord with the transition function δ from the proof of theorem 9.3.1, and the tuples of trees in accord with the operations in figure 9.1. Thus, we would like to map $\mathbf{mrg}(u,v)$, where u is mapped to $q(t_0,\ldots,t_n)$ and v to $q'(t'_0,\ldots,t'_n)$, to $\delta_{\mathbf{mrg}}(q,q')(\mathbf{m}(\langle t_0,\ldots,t_n\rangle, \langle t'_0,\ldots,t'_n\rangle))$ if the first component of q' is of length one, and to $\delta_{\mathbf{mrg}}(q,q')(\mathbf{m_j}(\langle t_0,\ldots,t_n\rangle, \langle t'_0,\ldots,t'_n\rangle))$ if the tail of the first component of q' begins with $\texttt{-j}$. This intuitive picture is very close to the multi bottom-up tree transducer model introduced in (Fülöp et al., 2004). A multi bottom-up tree transducer is a tuple $M = (Q,\Sigma,\Delta,root,f,R)$ where:

1. Q is a ranked alphabet, *the states*, with $Q^{(0)} = \emptyset$
2. Σ and Δ are ranked alphabets, respectively the *input alphabet* and the *output alphabet*
3. *root* is a unary symbol called the *root*
4. f is the final 'state'. Q, $\Sigma\cup\Delta$, $\{root\}$, and $\{f\}$ are pairwise disjoint sets
5. R is the set of *rules* which are of one of the two forms below, for $\sigma\in\Sigma^{(n)}$, $q_k\in Q^{(r_k)}$, and $t_l \in T_\Delta(\{y_{1,1},\ldots,y_{1,r_1},\ldots,y_{n,1},\ldots,y_{n,r_n}\})$ and $q\in Q^{(n)}$ and $t\in T_\Delta(X_n)$

$$\sigma(q_1(y_{1,1},\ldots,y_{1,r_1}),\ldots,q_n(y_{n,1},\ldots,y_{n,r_n})) \to q_0(t_1,\ldots,t_{r_0})$$

$$root(q(x_1,\ldots,x_n)) \to f(t)$$

An mbutt is *linear* just in case each of the variables occur *at most once* in *at most one* of the output trees. It is *deterministic* just in case there are no two productions with the same left hand sides.

Theorem 9.4.1. *For every minimalist grammar $G = \langle\mathbb{F},Lex,\mathtt{c}\rangle$, there is a linear deterministic multi-bottom up tree transducer M_G such that for $L(A_G)$ the set of derivations of complete expressions of category* c*, $M_G(L(A_G)) = S(G)$.*

Proof. The states of M_G are triples of states from our bottom-up tree automaton, our tree sequence algebra operations, and a boolean value which encodes the result of the test for the function f (whether or not the first tree is a symbol from Σ_ε). Each state has the same arity, $|\mathbf{lic}|+1$. Our rules R include[5]

1. for $q = \delta_{\mathbf{mv}}(q_1)$, $\rho\in\{\mathbf{v_i},\mathbf{v_{i,j}}\}$ such that the condition in figure 9.1 is satisfied, and $\langle t_0,\ldots,t_n\rangle = \rho(\langle y_{1,1},\ldots,y_{1,n+1}\rangle)$,

$$\mathbf{mv}(\langle q_1,\rho_1,b_1\rangle(y_{1,1},\ldots,y_{1,n+1})) \to \langle q,\rho,0\rangle(t_0,\ldots,t_n)$$

2. for $q = \delta_{\mathbf{mrg}}(q_1,q_2)$, $\rho\in\{\mathbf{m},\mathbf{m_i}\}$ such that the condition in figure 9.1 is satisfied, and $\langle t_0,\ldots,t_n\rangle = \rho(\langle y_{1,1},\ldots,y_{1,n+1}\rangle,\langle y_{2,1},\ldots,y_{2,n+1}\rangle)$,

$$\mathbf{mrg}(\langle q_1,\rho_1,b_1\rangle(y_{1,1},\ldots,y_{1,n+1}), \langle q_2,\rho_2,b_2\rangle(y_{2,1},\ldots,y_{2,n+1})) \to \langle q,\rho,0\rangle(t_0,\ldots,t_n)$$

3. $\langle\sigma,l\rangle \to \langle q,\rho,1\rangle(\sigma,\mathtt{t},\ldots,\mathtt{t})$ just in case $q = \delta_{\langle\sigma,l\rangle}(\langle\sigma,l\rangle)$
4. $root(\langle q,\rho,b\rangle(x_1,\ldots,x_{n+1})) \to f(x_1)$ just in case $q = \langle\mathtt{c},\varepsilon,\ldots,\varepsilon\rangle$

Again, a simple induction on derivation trees suffices to establish the correctness of the construction. □

Given the construction of the ldmbutt in theorem 9.4.1, it is clear that we could just as well have chosen different operations over different $n+1$ tuples of objects (Kobele (2006) provides an example of such). Additionally, we can use the very same feature calculus to simultaneously control different operations over different algebras. A synchronization of two dmbutts M and M' is a triple $\langle M,M',C\rangle$ where $C\subseteq R\times R'$ is the control set, which serves to specify which transitions in M are allowed to be used with which productions in M'. The relation defined by such an object is

$$\{\langle u,v\rangle : \exists t\in T_\Sigma.\exists c\in C^*.\ t\vdash_M^{\pi_1(c)} u \wedge t\vdash_{M'}^{\pi_2(c)} v\}$$

where π_i is the i^{th} projection function extended over sequences in the obvious way, $u\vdash_M^{aw} v$ just in case $u\vdash_M^a v'$ and $v'\vdash_M^w v$, and $u\vdash_M^a v$ just in case a is a production in M, u rewrites to v in a single step using a, and a is applied to the left-most rewritable node in u.

[5] The resolution of the operation $\oplus$ in the definition of ρ must be done by the states q_1 and q_2. As an empty component in a state is shadowed by a trace in a tree sequence, this is merely a notational inconvenience.

This is (a restriction to a particular transducer type of) a generalization of the model of synchronous tree adjoining grammars, as developed by Shieber and Schabes (1990), and thus can be used, for example, to model the syntax-semantics interface (Nesson and Shieber, 2006). Shieber (2006) investigates the complexity of the form-meaning relationships definable by synchronous TAGs by situating them within the context of bimorphisms. A bimorphism is a triple $B = \langle \Phi, L, \Psi \rangle$, where L is a recognizable tree language and Φ and Ψ are homomorphisms; the relation it defines is $L(B) = \{\langle \Phi(t), \Psi(t) \rangle : t \in L\}$. Given classes $\mathbf{H_1}$ and $\mathbf{H_2}$ of homomorphisms, $\mathbf{B}(\mathbf{H_1}, \mathbf{H_2})$ denotes the class of bimorphisms $\langle h_1, L, h_2 \rangle$ where $h_i \in \mathbf{H_i}$ and L is recognizable. Shieber proves that synchronous TAGs define exactly those relations definable by bimorphisms where the homomorphisms are one state monadic macro tree transducers simple in the input and parameters.

The following theorem is an easy consequence of a result in Fülöp et al. (2004).

Theorem 9.4.2. *The relation defined by a synchronization $\langle M, M', C \rangle$ of dmbutts M and M' is in the bimorphism class $\mathbf{B}(\mathbf{M}, \mathbf{M})$, where $\mathbf{M}$ is the class of unrestricted homomorphisms. It is in the class $\mathbf{B}(\mathbf{FC}, \mathbf{FC})$, where $\mathbf{FC}$ is the class of finite copying homomorphisms, if M and M' are linear.*

Proof. By moving to the expanded alphabet $\Sigma \times R \times R'$, we can find new dmbutts $M_\star$ and $M'_\star$ such that the set $\{\langle M_\star(t), M'_\star(t) \rangle : t \in T_{\Sigma \times R \times R'}\}$ is the very same relation as defined by $\langle M, M', C \rangle$ (we essentially encode the control information into the vocabulary itself). By theorem 4.4 in Fülöp et al. (2004), we can find an equivalent dTT^R for any dmbutt. It is well-known that regular look-ahead and states can be encoded into a regular set of trees (Engelfriet, 1977), and therefore for any dTT^R T and regular language L we can find a homomorphism h and regular language L_h such that $T(L) = h(L_h)$. Thus, from M and M' over T_Σ, we move to $M_\star$ and $M'_\star$ over $T_{\Sigma \times R \times R'}$, and from there we obtain $T_\star$ and $T'_\star$, whence we finally arrive at homomorphisms $h_\star$ and $h'_\star$. By the result in the appendix, $h_\star$ ($h'_\star$) is finite copying if M (M') is linear. □

9.5 Conclusion

In light of our transductive characterization of minimalist grammars, what seems the core of the minimalist grammar framework is the underlying feature calculus, and the n-tuples of terms that are therewith naturally controllable. The cases of the generating functions (**merge1**,**merge2**,...) that were introduced at the beginning are now revealed to be gerrymanderings of the feature calculus to support the particular mode of manipulating expressions qua minimalist trees. Different modes of expression manipulation, or different choices of expressions to manipulate, might well have drawn different lines in the sand. This perspective allows us to consider the *relations* that the minimalist feature calculus makes definable. Situating natural langauge formalisms in the context of bimorphisms provides an elegant and principled way of measuring and comparing their 'strong generative capacity'—the kinds of form-meaning relations the formalism can define. We have seen that all of the relations definable by synchronous minimalist grammars are naturally expressible as bimorphisms where the component maps are simple tree-to-tree homomorphisms. Our characterization is still loose. We must leave it for future work to determine a tighter description of the relations naturally definable by minimalist grammars.

Appendix

In this appendix we show the inclusion of the relations definable by linear deterministic multi bottom-up tree transducers in those definable by single use deterministic top-down tree transducers with regular look-ahead (dTT^R_{su}) which are known to be equivalent to deterministic top-down tree transducers with regular look-ahead with finite copying (dTT^R_{fc}) (Engelfriet and Maneth, 1999). First, some definitions.

Given Σ and Δ two ranked alphabets, we define $\Sigma \cup \Delta$ to be the ranked alphabet such that $(\Sigma \cup \Delta)^{(n)} = \Sigma^{(n)} \cup \Delta^{(n)}$. A set A is made into a ranked alphabet $R(A)$ such that $R(A)^{(0)} = A$ and $R(A)^{(k)} = \emptyset$ when $k > 0$. In particular we write $T_\Omega(A)$ for $T_{\Omega \cup R(A)}$.

We describe tree substitution with a set of indexed input variables $X = \{x_k : k \in \mathbb{N} \wedge k > 0\}$ and also a set of double indexed input variables $Y = \{y_{i,j} : i \in \mathbb{N} \wedge i > 0 \wedge j \in \mathbb{N} \wedge j > 0\}$. The set X_n will denote $\{x_k : k \in [n]\}$ and the set $Y_{n,\langle r_1,\ldots,r_n \rangle}$ will denote $\{y_{k,l} : k \in [n] \wedge l \in [r_k]\}$. Given $t \in T_\Sigma(X_n)$ (resp $T_\Sigma(Y_{n,\langle r_1,\ldots,r_n \rangle})$) and for $k \in [n]$ $t_k \in T_\Sigma$ (resp $k \in [n]$, $l \in [r_k]$ and $t_{k,l} \in T_\Sigma$), we write $t[t_1, \ldots, t_n]$ (resp $t[t_{1,[r_1]}, \ldots, t_{n,[r_n]}]$) for the result of replacing every occurrence of x_k (resp $y_{k,l}$) in t by t_k (resp $t_{k,l}$) for all $k \in [n]$ (resp $k \in [n]$ and $l \in [r_k]$). Given $\sigma \in \Sigma^{(r_k)}$ and a family $(t_{k,l})_{l \in [r_k]}$, we abbreviate $\sigma(t_{k,1}, \ldots, t_{k,r_k})$ to $\sigma(t_{k,[r_k]})$ (a particular case is when $t_{k,l} = y_{k,l}$). We also assume z to be a variable that is neither in X nor in Y and we use to define contexts. A context of T_Σ is an element C of $T_\Sigma(\{z\})$ such that z occurs exactly once in C. Given a t we write $C[t]$ the tree obtained by replacing the occurrence of z in C by t. Contexts will alway be written using capital C with indicies, exponents, primes, ...

Definition 9.5.1. *A top-down tree transducer with regular look-ahead (TT^R for short) is a tuple $M = (Q, \Sigma, \Delta, q_0, R, P, \delta)$ where*

1. *Q is a ranked alphabet of* states *such that $Q^k = \emptyset$ for every $k \in \mathbb{N} \backslash \{1\}$.*
2. *Σ and Δ are ranked alphabets, repectively, the* input alphabet *and the* output alphabet. *Q and $\Sigma \cup \Delta$ are disjoint.*

3. q_0 *is an element of* Q*, the* initial state.
4. (P,Σ,δ) *is a deterministic BA (the final states are unnecessary and will be suppressed).*
5. R *a subset of* $\bigcup_{\Sigma^{(n)}\neq\emptyset} Q\times\Sigma^{(n)}\times T_\Delta(\{q(x_i):q\in Q\wedge x_i\in[n]\})\times P^n$*, the* rules.

As usual, the rule $(q,\sigma^{(n)},t,(p_1,\ldots,p_n))$ of a $tdTT^R$ will be written

$$\langle q(\sigma(x_1,\ldots,x_n))\to t,p_1,\ldots,p_n\rangle.$$

A $tdTT^R$, $M=(Q,\Sigma,\Delta,q_0,R,P,\delta)$, is said to be deterministic whenever given rules

1. $\langle q(\sigma(x_1,\ldots,x_n))\to t,p_1,\ldots,p_n\rangle$ and
2. $\langle q(\sigma(x_1,\ldots,x_n))\to t',p_1,\ldots,p_n\rangle$

in R then $t=t'$. The class of dTT^R that are deterministic will be written dTT^R.

A TT^R, $M=(Q,\Sigma,\Delta,q_0,R,P,\delta)$, defines a relation on $T_{\Sigma\cup\Delta\cup Q}$. Given $t,t'\in T_{\Sigma\cup\Delta\cup Q}$, we write $t\to_M t'$ if there is C, a context of $T_{\Sigma\cup\Delta\cup Q}$, $\langle q(\sigma(x_1,\ldots,x_n))\to v,p_1,\ldots,p_n\rangle\in R$ and $(t_k)_{k\in[n]}$ verifying:

1. $t=C[q(\sigma(t_1,\ldots,t_n))]$
2. for all $k\in[n]$, $t_k\in\mathcal{L}(P,p_k)$
3. $t'=C[v[t_1,\ldots,t_n]]$.

The reflexive and transitive closure of $\to_M$ is written $\Rightarrow_M$, and the relation that M defines between T_Σ and T_Δ is

$$\mathcal{R}_M=\{(t,t'):t\in T_\Sigma\wedge t'\in T_\Delta\wedge q_0(t)\Rightarrow_M t'\}.$$

We now introduce the notion of strongly single use and single use deterministic top-down transduction that has been introduced in Engelfriet and Maneth (1999).

Definition 9.5.2. *Let* $M=(Q,\Sigma,\Delta,q_0,R,P,\delta)$ *be a* dTT^R *and* $\overline{Q}$ *be a nonempty subset of* Q*. M is said* strongly single use with respect to $\overline{Q}$*, if for all* $q,q'\in\overline{Q}$ *and all two rules of R :*

1. $\langle q(\sigma(x_1,\ldots,x_n))\to v,p_1,\ldots,p_n\rangle$
2. $\langle q'(\sigma(x_1,\ldots,x_n))\to w,p_1,\ldots,p_n\rangle$

the existence of contexts C *and* C'*,* $q''\in Q$ *and* $j\in[n]$ *such that* $v=C[q''(x_j)]$ *and* $w=C'[q''(x_j)]$ *implies* $q=q'$ *and* $C=C'$.

If M is strongly single use with respect to Q the M is said strongly single use.

Definition 9.5.3. *Let* $M=(Q,\Sigma,\Delta,q_0,R,P,\delta)$ *be a* dTT^R*. M is said* single use *if there is a partition* Π *of* Q *and a collection of mappings* $(\mathcal{T}_{\sigma,\langle p_1,\ldots,p_n\rangle}:\Pi\times[n]\to\Pi:\sigma\in\Sigma^{(n)},p_1,\ldots,p_n\in P)$ *such that:*

1. *for all* $\overline{Q}\in\Pi$*, M is strongly single use with respect to* $\overline{Q}$ *and*
2. *for all* $\langle q(\sigma(x_1,\ldots,x_n))\to v,p_1,\ldots,p_n\rangle\in R$ *with* $q\in\overline{Q}\in\Pi$*, if there is an occurrence of* $q'(x_i)$ *in* v *then* $q'\in\mathcal{T}_{\sigma,\langle p_1,\ldots,p_n\rangle}(\overline{Q},i)$.

The partition Π *is called a* su partition *for M and* $\mathcal{T}$ *is called a* collection of su mapping for M*. We will write* dTT^R_{su} *to denote the class of* dTT^R *that are single use.*

We now define the relation computed by multi bottom-up tree transduction.

A mbutt $M=(Q,\Sigma,\Delta,root,q_f,R)$ defines a relation $\to_M$ on the trees of $T_{Q\cup\Sigma\cup\Delta\cup\{root,q_f\}}$. Given $t,t'\in T_{Q\cup\Sigma\cup\Delta\cup\{root,q_f\}}$, we have $t\to_M t'$ if there is C a context of $T_{Q\cup\Sigma\cup\Delta\cup\{root,q_f\}}$ verifying one of the two following properties:

1. $t=C[\sigma(q_1(t_{1,[r_1]}),\ldots,q_n(t_{n,[r_n]}))]$, $t'=C[q_0(t_1,\ldots,t_{r_0})[t_{1,[r_1]},\ldots,t_{n,[r_n]}]]$ and
$$\sigma(q_1(y_{1,[r_1]}),\ldots,q_n(y_{n,[r_n]}))\to q_0(t_1,\ldots,t_{r_0})\in R$$
2. $t=C[root(q(t_1,\ldots,t_n))]$, $t'=C[q_f(v[t_1,\ldots,t_n])]$ and
$$root(q(x_1,\ldots,x_n))\to q_f(v)\in R$$

The reflexive and transitive closure of $\to_M$ is denoted by $\Rightarrow_M$. M defines a relation between T_Σ and T_Δ, $\mathcal{R}_M=\{(t,t')\in T_\Sigma\times T_\Delta:root(t)\Rightarrow_M q_f(t')\}$.

A mbutt, $M=(Q,\Sigma,\Delta,root,q_f,R)$, is called *deterministic* whenever

1. $\sigma(q_1(y_{1,[r_1]}),\ldots,q_n(y_{n,[r_n]}))\to q_0(t_1,\ldots,t_{r_0})\in R$ and
2. $\sigma(q_1(y_{1,[r_1]}),\ldots,q_n(y_{n,[r_n]}))\to q'_0(t'_1,\ldots,t'_{r'_0})\in R$

imply $q_0=q'_0$ and for all $k\in[r_0]$, $t_k=t'_k$.

Now that we have defined all the necessary notions, we prove that the classes of transduction realized by dTT^R_{su} include those defined by ldmbutts. In Fülöp et al. (2004), it is shown that dTT^R and dmbutts define the same class of transduction. We here prove the transductions defined by ldmbutts can be defined by dTT^R_{su}; this proof uses the same construction as in lemma 4.1 in Fülöp et al. (2004) and we thus only have to prove that when this construction is used on a ldmbutt it ouputs a dTT^R_{su}.

Let $M=(Q,\Sigma,\Delta,root,q_f,R)$ be a ldmbutt and $A=(Q,\Sigma,\delta)$ be the *dBA* underlying M. We construct the dTT^R $T=(Q',\Sigma,\Delta,p_0,R',Q,\delta)$ as follows:

1. $Q'=\{p_0\}\cup\{\langle q,j\rangle:q\in Q^{(n)}\wedge j\in[n]\}$
2. R' is the smallest set of rules verifying:
   (a) if $\sigma(q_1(y_{1,[r_1]}),\ldots,q_n(y_{n,[r_n]}))\to q_0(t_1,\ldots,t_{r_0})\in R$ then for all $j\in[r_0]$,
   $$\langle\langle q,j\rangle(\sigma(x_1,\ldots,x_n))\to t_j[t_{1,[r_1]},\ldots,t_{n,[r_n]}],q_1,\ldots,q_k\rangle\in R'$$
   with for $k\in[n]$ and $l\in[r_k]$, $t_{k,l}=\langle q_k,l\rangle(x_k)$.

(b) if $root(q(x_1,\ldots,x_n) \to q_f(t)) \in R$ and for all $k \in [n]$ we have that there is, in R' a rule of the form

$$\langle\langle q,k\rangle(\sigma(x_1,\ldots,x_k))t'_k, q_1,\ldots,q_n\rangle$$

then $\langle p_0(\sigma(x_1,\ldots,x_n)) \to t[t'_1,\ldots,t'_n], q_1,\ldots q_n\rangle \in R'$

Fülöp et al. (2004) proves that $\mathcal{R}_T = \mathcal{R}_m$ and that T is indeed a dTT^R. We just have to prove that from the fact that M is linear, T effects a single use transduction.

Although T is not itself single use (the start state p_0 does not satisfy the definition), we will prove that the transducer T' naturally obtained from T by suppressing p_0 **is**. Since p_0 is used only once at the very beginning of any transduction performed by T, it follows that T is finite copying, and can thus be turned into a single use transducer (Engelfriet and Maneth, 1999). To prove that T' is single use we need to find an *su* partition Π for T'. We define Π to be $(\Pi_q)_{q\in Q}$ with $\Pi_q = \{\langle q,i\rangle | i \in [rank(q)]\}$. An element of Π corresponds to the set of states of Q' that are defined from a unique state of Q.

We now first prove that for a given $q \in Q$, T' is strictly single use with respect to Π_q. Suppose that the rules $\langle\langle q,i\rangle(\sigma(x_1,\ldots,x_n)) \to v, q_1,\ldots,q_n\rangle$ and $\langle\langle q,j\rangle(\sigma(x_1,\ldots,x_n)) \to w, q_1,\ldots,q_n\rangle$ are in R', because M is deterministic there is in R a unique rule of the form

$$\sigma(q_1(y_{1,1},\ldots,y_{1,r_1}),\ldots,q_n(y_{n,1},\ldots,y_{n,r_n})) \to q_0(t_1,\ldots,t_{r_0})$$

thus, by definition of R', we must have:

1. $q = q_0$,
2. $v = t_i[t_{1,[r_1]},\ldots,t_{n,[r_n]}]$,
3. $w = t_j[t_{1,[r_1]},\ldots,t_{n,[r_n]}]$ with,
4. for $k \in [n]$ and $l \in [r_k]$, $t_{k,l} = \langle q_k,l\rangle(x_k)$.

Suppose that both v and w contain an occurrence of $\langle q_k,l\rangle(x_k)$, then both t_i and t_j contain an occurrence of $x_{k,l}$ and since M is linear we have $i = j$ which finaly entails that the two rules are the same, and the occurrences $\langle q_k,l\rangle(x_k)$considered in v and w are in fact a unique occurrence; therefore M is strictly single use with respect to Π_q.

To complete the proof that T' is single use, we now define a collection of *su* mapping of T'.

Given $\sigma \in \Sigma^{(n)}$ and $q_0, \ldots, q_n \in Q$, we define the function $\mathcal{L}_{\sigma,\langle q_1,\ldots,q_n\rangle} : \Pi \times [n] \to \Pi$ to associate Π_{q_i} to (Π_{q_0}, i) if $\sigma(q_1(y_{1,[r_1]}),\ldots,q_n(y_{n,[r_n]})) \to q_0(t_1,\ldots,t_{r_0})$ is in R. The determinism of M trivially implies that $\mathcal{L}_{\sigma,\langle q_1,\ldots,q_n\rangle}$ is actually a function. Now for $\langle\langle q_0,i\rangle(\sigma(x_1,\ldots,x_n)) \to v, q_1,\ldots,q_n\rangle \in R'$, whenever $\langle q_k,l\rangle(x_k)$ occurs in v, by construction, we have that $\langle q_k,l\rangle \in \mathcal{L}_{\sigma,\langle q_1,\ldots,q_n\rangle}(\Pi_{q_0},k) = \Pi_k$. This finally shows that T' is single use (and therefore, as per the remark above, that T realizes a single use transduction).

Bibliography

Bauderon, Michel and Bruno Courcelle (1987). Graph expressions and graph rewritings. *Mathematical Systems Theory*, **20**(2-3):83–127.

Bobaljik, Jonathan David (1999). Adverbs: The hierarchy paradox. *Glot International*, **4**(9-10):27–28.

Chomsky, Noam (1995). *The Minimalist Program*. MIT Press, Cambridge, Massachusetts.

de Groote, Philippe, Glyn F. Morrill, and Christian Retoré, eds. (2001). *Logical Aspects of Computational Linguistics*, volume 2099 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, Berlin.

Engelfriet, Joost (1977). Top-down tree transducers with regular look-ahead. *Mathematical Systems Theory*, **10**:289–303.

Engelfriet, Joost (1994). Graph grammars and tree transducers. In Sophie Tison, ed., *Trees in Algebra and Programming – CAAP'94*, volume 787 of *Lecture Notes in Computer Science*, pp. 15–36. Springer.

Engelfriet, Joost and Linda Heyker (1992). Context-free hypergraph grammars have the same term-generating power as attribute grammars. *Acta Informatica*, **29**(2):161–210.

Engelfriet, Joost and Sebastian Maneth (1999). Macro tree transducers, attribute grammars, and MSO definable tree translations. *Information and Computation*, **154**:34–91.

Engelfriet, Joost and Sebastian Maneth (2000). Tree languages generated by context-free graph grammars. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, eds., *Selected papers from the 6th International Workshop on Theory and Application of Graph Transformations*, volume 1764 of *Lecture Notes in Computer Science*, pp. 15–29. Springer Verlag.

Engelfriet, Joost and Heiko Vogler (1985). Macro tree transducers. *Journal of Computer and System Sciences*, **31**:71–146.

Fülöp, Zoltán, Armin Kühnemann, and Heiko Vogler (2004). A bottom-up characterization of deterministic top-down tree transducers with regular look-ahead. *Information Processing Letters*, **91**:57–67.

Fülöp, Zoltán, Armin Kühnemann, and Heiko Vogler (2005). Linear deterministic multi bottom-up tree transducers. *Theoretical Computer Science*, **347**:276–287.

Harkema, Henk (2001). A characterization of minimalist grammars. In de Groote et al. (2001).

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

Michaelis, Jens (1998). Derivational minimalism is mildly context-sensitive. In M. Moortgat, ed., *Logical Aspects of Computational Linguistics, (LACL '98)*, volume 2014 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, Berlin, Heidelberg, Germany.

Michaelis, Jens (2001). Transforming linear context-free rewriting systems into minimalist grammars. In de Groote et al. (2001).

Michaelis, Jens (2005). An additional observation on strict derivational minimalism. In Gerhard Jäger, Paola Monachesi, Gerald Penn, James Rogers, and Shuly Wintner, eds., *Proceedings of the 10th conference on Formal Grammar and the 9th Meeting on Mathematics of Language*. Edinburgh.

Michaelis, Jens, Uwe Mönnich, and Frank Morawietz (2001). On minimalist attribute grammars and macro tree transducers. In Christian Rohrer, Antje Rossdeutscher, and Hans Kamp, eds., *Linguistic Form and its Computation*, pp. 51–91. CSLI Publications.

Mönnich, Uwe (1997). Adjunction as substitution. In Geert-Jan M. Kruijff, Glyn Morrill, and Richard T. Oehrle, eds., *Formal Grammar 1997: Proceedings of the Conference*, pp. 169–178. Aix-en-Provence.

Morawietz, Frank (2003). *Two-Step Approaches to Natural Language Formalisms*, volume 64 of *Studies in Generative Grammar*. Mouton de Gruyter.

Nesson, Rebecca and Stuart Shieber (2006). Simpler TAG semantics through synchronization. In Paola Monachesi, Gerald Penn, Giorgio Satta, and Shuly Wintner, eds., *Proceedings of the 11th conference on Formal Grammar*, pp. 103–117. CSLI.

Pan, Michael J. (2007). Pomset mcfgs. In *Proceedings of the 10th International Conference on Parsing Technology (IWPT)*.

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

Shieber, Stuart M. (1994). Restricting the weak-generative capacity of synchronous tree-adjoining grammars. *Computational Intelligence*, **10**(4):371–385.

Shieber, Stuart M. (2006). Unifying synchronous tree-adjoining grammars and tree transducers via bimorphisms. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-2006)*, pp. 377–384. Trento.

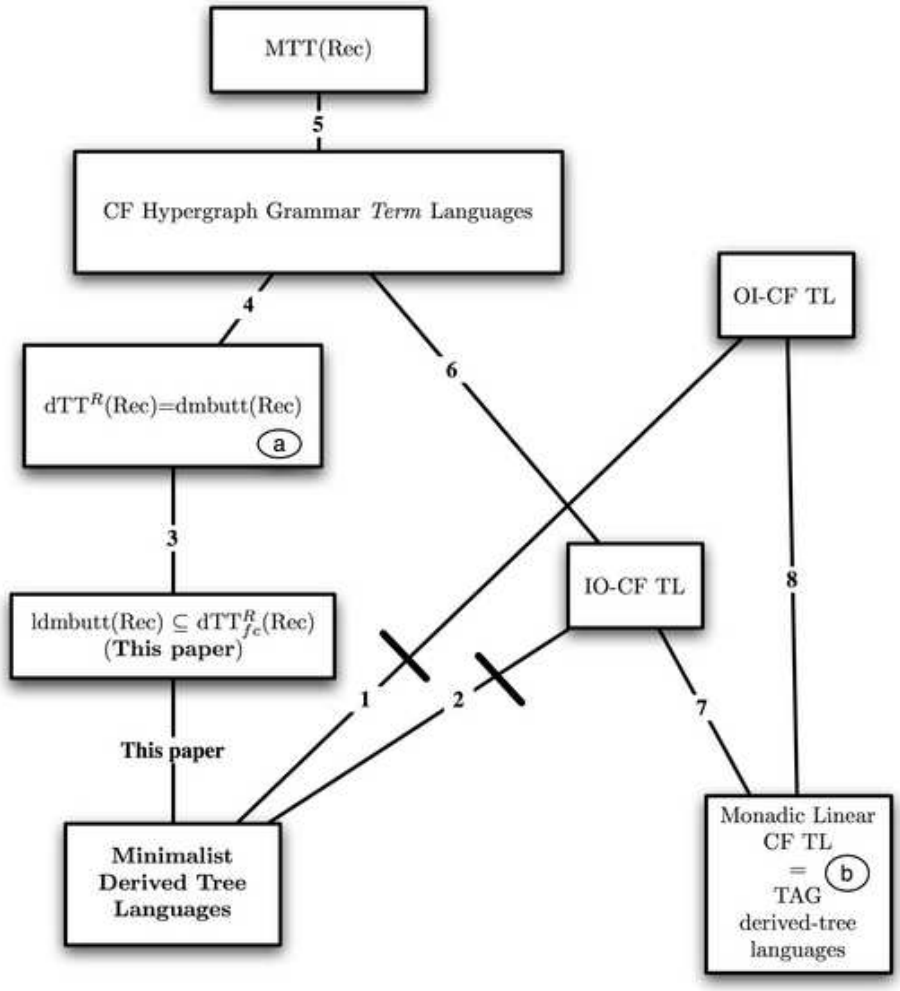
Shieber, Stuart M. and Yves Schabes (1990). Synchronous tree-adjoining grammars. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING)*, volume 3, pp. 253–258. Helsinki,Finland.

Stabler, Edward P. (1997). Derivational minimalism. In Christian Retoré, ed., *Logical Aspects of Computational Linguistics*, volume 1328 of *Lecture Notes in Computer Science*, pp. 68–95. Springer-Verlag, Berlin.

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

Vijay-Shanker, K., David Weir, and Aravind Joshi (1987). Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th Meeting of the Association for Computational Linguistics*, pp. 104–111.

Weir, David J. (1988). *Characterizing Mildly Context-Sensitive Grammar Formalisms*. Ph.D. thesis, University of Pennsylvania.



TL Tree Languages **Rec** Regular Tree Languages **CF** Context Free **OI** Outside-In **IO** Inside Out **TAG** Tree Adjoining Grammars	**MTT** Macro Tree Transducer **dTT**$^R_{fc}$ Deterministic Top Down Tree Transducer with Regular Look-Ahead (Finite Copy) — see this paper. **(l)dmbutt** (Linear) Deterministic Multi-Bottom-Up Tree Transducer — see this paper.	**a** was proved in Fülöp et al. (2004) **b** was proved in Mönnich (1997) **1, 2** these non inclusions were proved in Michaelis (2005) by looking at the string languages **4,5,6** are presented in Engelfriet and Heyker (1992) **7,8** are obvious.

Figure 9.2: Minimalist derived trees and friends

Chapter 10

Minimalist Syntax, Multiple Regular Tree Grammars and Direction Preserving Tree Transductions[1]

Uwe Mönnich
Linguistics Department, University of Tübingen
Tübingen, Germany
um@sfs.uni-tuebingen.de

10.1 Introduction

Model-theoretic syntax deals with the logical characterization of complexity classes. The first results in this area were obtained in the early and late Sixties of the last century. In these results it was established that languages recognised by finite string and tree automata are definable by means of monadic second-order logic (MSO).

To be slightly more precise, the classical results, just mentioned, can be seen as providing translation procedures that relate logical specifications with finite automata equivalent with respect to the defined language classes. Along this way, Büchi (1960) and Elgot (1961) have shown that regular string languages represented through finite (string) automata can be expressed by sentences in the (weak) MSO logic with one successor. For tree languages an analogous result is well known: a tree language is definable in weak MSO logic with multiple successors if and only if it is recognizable by a finite tree automaton (Thatcher and Wright 1968; Doner 1970).

All these approaches suffer from a lack of expressive power in that the family of regular tree languages properly includes all other language families that are captured by the logical formalisms that have been considered in model-theoretic syntax. It is due to this lack of expressive power that grammatical phenomena like cross-serial dependencies in languages like Swiss German or Bambara are beyond the reach of the kind of logical apparatus currently applied to natural language syntax.

We shall therefore propose a hybrid solution to the problem of how to account for mildly context-sensitive phenomena with the help of tree logic. The limited expressive power of this logic in its original set-up makes it impossible to formulate the solution in a way that would deal directly with the problematic phenomena, but we can give these phenomena a slightly different appearance whereby they do become regular and as such definable in tree logic.

The paper is structured as follows. Section 10.2 recalls basic notions from logic, tree grammars and tree translation we need for our discussion in the rest of the paper. In Section 3 we prove that minimalist syntax is equivalent to direction preserving MSO transductions. We conclude with some general remarks and point out some open problems.

10.2 Preliminaries

The introductory section has tried to motivate the method of semantic interpretation and to explain its application to the theory of natural syntax. This section defines familiar notions from the theory of syntax-directed semantics together with its model-theoretic counterpart, the theory of monadic second-order transductions. We assume that the reader has seen an exposition of the basic concepts of universal algebra.

10.2.1 Syntax-Directed Semantics

Macro tree transducers (MTT) are a model of tree transformation that transduces in a recursive top-down fashion an input tree into an output tree, handling context information in an implicit way. The elements of context information do not have explicit names, but are passed along as parameters of the states in this kind of translation device.

Definition 10.2.1. *A* macro tree transducer *is a tuple* $M = (Q, \Sigma, \Omega, q_0, R)$, *where* Q *is a ranked alphabet of* states, Σ *and* Ω *are ranked alphabets of* input *and* output *symbols, respectively,* $q_0 \in Q_0$ *is the* initial *state and* R *is a set of* rules *of the following form:*

$$(q, \sigma(x_1, \ldots, x_m))(y_1, \ldots, y_n) \rightarrow \xi$$

where $q \in Q^{(n)}$, $\sigma \in \Sigma^{(m)}$ *and* $\xi \in T_{\langle Q, X_m \rangle \cup \Omega}(Y_n)$.

Remark 10.2.2. If every state in Q has rank zero, then M is a *top-down transducer (TOP)*. Macro tree transducers

[1]We are grateful to a number of people who have helped us understand the connections between syntax-directed semantics and logic based transductions, including: Tom Cornell, Hap Kolb, Stephan Kepser, Jens Michaelis and Frank Morawietz. This work was partially supported by the German Research Council through a grant to the Collaborative Research Center 441 at Tübingen University.

can therefore be regarded as a context-sensitive extension of top-down transducers.

Macro tree transducers can realize translations that are of double exponential size increase. A subtree of an input tree s can be processed arbitrarily many times by a macro tree transducer M depending on the number of occurrences of an input variable x_i in the right-hand side of a rule of M that rewrites the mother of the particular subtree in a particular state. Restricting the contribution to an output tree that is provided by this copying power of macro tree transducers leads to the notion of *finite-copying* macro tree transducers.

Definition 10.2.3. *Let M be a macro tree transducers with input alphabet Σ that is simple in the parameters. If there is a number $k \in \mathbb{N}$ such that for every input $s \in T_\Sigma$ and node u of s the length of the state sequence of s at node u $|sts_M(s,u)| \leq k$, then M is finite-copying (fc).*

Definition 10.2.4. *Disregarding the input of a macro tree transducer one obtains a context-free tree (CFT) grammar. A CFT grammar is a tuple $G = (\mathcal{F}, \Omega, S, P)$ where $\mathcal{F}$ and Ω are ranked alphabets of nonterminals and terminals, respectively, $S \in \mathcal{F}_0$ is the start symbol and P is a finite set of productions of the form*

$$F(y_1, \ldots, y_m) \rightarrow \xi$$

where $F \in \mathcal{F}$ and ξ is a tree over $\mathcal{F}$, Ω and Y_m.

The family of tree languages which is generated by context-free tree grammars which are simple in their parameters is designated as CFT_{sp}.

A grammar $G = \mathcal{F}, \Omega, S, P)$ is called a *regular tree (REGT)* grammar if $\mathcal{F} = \mathcal{F}^{(0)}$, i.e., if all nonterminals are of rank 0.

A further grammatical formalism is defined for the generation of tree tuples. This is an extension of the notion of regular tree grammar, i.e., all the nonterminals are of rank 0, but they range over tuples of trees instead of single trees only.

Definition 10.2.5. *A grammar $G = (\mathcal{F}, \Omega, S, P)$ is called a* multiple regular tree *(MREGT) grammar if $\mathcal{F} = \mathcal{F}^{(0)}$, i.e., if all nonterminals are of rank 0, each nonterminal has assigned a tuple index ≥ 1, the start symbol S has tuple index 1 and the productions are of the form*

$$F \rightarrow (\xi_1, \ldots, \xi_n)$$

where n is the tuple index of F, $\xi_i (1 \leq i \leq n)$ is a tree over $\mathcal{F} \times \{1, \ldots, m\}$ and Ω. It is assumed that m is the maximal tuple index of a nonterminal in $\mathcal{F}$ and that each component $\langle F, k \rangle$ of a nonterminal leaf label occurs exactly once in the ξ_i.

The special case in which the right-hand sides of the productions are tuples of words over $\mathcal{F}$ and Ω is referred to a s a *multiple context-free* (MCF) grammar.

10.2.2 Semantic Interpretations

Declarative tree transductions are inspired by the model-theoretic technique of semantic interpretation (Rabin, 1965). The idea is to define a relational structure inside another structure in terms of monadic second-order formulas. Both the input and the output structures are finite trees regarded as finite models. The definitional power of monadic second-order tree transducers is highly restricted. The output string languages of these tree transducers defined over regular tree families are mildly context-sensitive (Engelfriet and Heyker, 1991), as will be discussed in the next section.

The language to be used for the specification of properties and relations satisfied by finite tree structures is a straightforward extension of first-order logic: monadic second-order logic (*MSO*). The language of this logic contains variables that range over subsets of the universe of discourse and quantifiers that bind these (monadic) predicate variables.

Given a ranked signature Σ the monadic second-order language over trees in T_Σ uses atomic formulas $lab_\sigma(x)$ $(\sigma \in \Sigma)$, $child_i(x,y)$, $x = y$ and $x \in X$ to convey the idea that node x has label σ, that node y is the i-th child of node x, that x and y are the same node and that node x is a member of the set of nodes X.

Definition 10.2.6. *Given two ranked alphabets Σ and Ω and a finite set C of copy names, a* monadic second-order definable *tree transducer T from T_Σ to T_Ω is specified by the following formulas of the monadic second-order language over Σ:*

(i) a closed formula φ, the domain *formula*

(ii) formulas $\nu_c(x)$ with $c \in C$, the node *formulas*

(iii) formulas $\psi_{\delta,c}(x)$ with $c \in C$ and $\delta \in \Omega$, the labelling *formulas*

(iv) formulas $\chi_{i,c,d}(x,y)$ with $c,d \in C$ and $i \leq$ maximal arity of symbols in Ω, the edge *formulas*

In sharp contrast with the syntax-directed transformation devices a logic based tree transducer T does not translate its input trees in a recursive top-down manner. The translation τ_T realized by such a declarative transducer has to be defined in terms of the familiar ingredients of a relational structure.

Definition 10.2.7. *The* tree translation τ_T *realized by a monadic second-order definable tree transducer T from T_Σ to T_Ω is a partial function $\tau_T : T_\Sigma \rightarrow T_\Omega$ defined as follows. The domain of τ_T is $\{s \in T_\Sigma \mid s \models \varphi\}$. For every $s \in T_\Sigma$ in the domain of τ_T $\tau_T(s)$ is the tree structure $t \in T_\Omega$ such that:*

$D_t = \{(c,x) \in C \times D_s \mid s \models \nu_c(x)\}$
is the tree domain of t,
$E_t = \left\{ \begin{array}{l} ((c,x),i,(d,y)) \in D_t \times ar(\Omega) \times D_t \mid \\ s \models \chi_{i,c,d}(x,y) \end{array} \right\}$
is the edge relation of t,
where $ar(\Omega)$ *denotes the arity of* Ω,
$L_t = \{((c,x),\delta) \in D_t \times \Omega \mid s \models \psi_{c,\delta}(x)\}$
is the labelling function of t.

10.3 Minimalist Syntax

The syntactic theory in question is the version of minimalism as presented by Stabler (1997). Minimalist grammars reflect the transition within the transformational tradition from an approach which relies on an explicit formulation of well-formedness conditions on derivations and representations to a procedural specification of the derivational process itself. In this procedural specification the structure building operations are assumed to be determined by the syntactic features of the structures that form the arguments of these operations. Given this operational orientation of feature-driven minimalist syntax it may already be surprising that it allows for a logical specification by means of a particular form of semantic interpretation.

In this section we will show that minimalist syntax can be described in terms multiple regular grammars and, based on this description, sketch its relation with the subclass of direction preserving semantic interpretation where we rely on previously established results on the close connection between multiple regular grammars, top-down tree transducers and direction preserving logic based tree translations. For reasons of space we suppress a formal exposition of Minimalist Grammar (MG) along the lines of Michaelis et al. (2001).

The main insight behind the construction by Michaelis (2001a) of an equivalent multiple context-free string grammar for a given minimalist grammar consists in the realization that it is possible to code the tree structure relevant for the application of the operations of *merge* and *move* by means of nonterminal symbols of the resulting target grammar. These symbols range over yields of tuples of the relevant subtrees. Our proof omits the yield step and retains the tuples of subtrees. What remains to be done is the verification that the action of *merge* and *move* can be simulated by appropriate tree rearrangements that are permitted within the framework of multiple regular tree grammars.

The particular details of the relevant structural aspects that go into the coding of the nonterminals of the multiple regular tree grammar to be constructed are a direct consequence of the definition of the two operations *merge* and *move*. Occurrences of selection features and of their corresponding syntactic features can only be deleted by an application of the operation *merge* if they form the start feature of the head-label of some expression. Besides this structural information the nonterminal has to comprise the additional information as to whether the head with the starting selection feature is part of a complex tree or not, because this has a direct effect on the ordering of the expression resulting from an application of *merge*. In order to be a candidate for the operation *move* an expression has to display a head-label whose starting feature is a licensor. In addition, the expression has to contain a subtree which is a maximal projection and whose head-label starts with a matching licensee feature. This subtree of the expression has to be the only subtree fulfilling the condition of being a maximal projection with a particular licensee feature as the start category of its head-label.

Summarizing the conditions on a nonterminal of the resulting multiple regular tree grammar that collectively succeed in coding all structural aspects decisive for the application of one of the operations *merge* or *move* we are led to the following specifications. Assume that the given minimalist grammar G contains a set of n licensees $(-l_i)_{l \leq i < n}$. Each nonterminal of the resulting multiple tree grammar $G' = (N, \Sigma, P, F, S)$ is then represented by an $n+2$-tuple, where the first component is a suffix of one of the lexically given strings of syntactic categories, except those suffixes that start with one of the licensees $-l_i$ which form the next n components, the last component consisting of the feature *simple* or *complex*. This set of nonterminals N is certainly finite since it is constructed as a finite product of finite sets.

Theorem 10.3.1. *For every minimalist grammar G, there exists a strongly equivalent multiple regular grammar G'.*

In the last paragraphs we have indicated that minimalist grammars are captured by a slight extension of regular tree grammars. This extension employs productions in which the right-hand sides are a "leaf-linked" forest, i.e., finite tuples of trees with some of the leaves connected by means of secondary relations. This type of productions is one of the two ways considered in formal language theory of defining special subclasses of rules in context-free hyperedge replacement grammars with the purpose of limiting the generated graph languages to families of trees. The other subclass is characterized by the restriction that the right-hand sides of these particular expansion rules have to be just trees.

In the remaining part of this section we will try to fulfil our promises regarding minimalist syntax. Fortunately, the characterization of minimalist grammars in terms of multiple regular tree grammars furnishes the missing link in the chain of known equivalences leading from natural language syntax to model-theoretic interpretation via a link provided by automata-theoretic translation. We will first recall the relationship between multiple regular tree grammars and top-down tree transducers and then conclude this part of our discussion by providing a declarative definition of tree translations achieved through top-down transducers.

Theorem 10.3.2 (Raoult, 1997). *Multiple regular tree languages are the same as the output of finite-copying top-down tree transducers.*

Raoult (1997) presents a detailed verification that the construction illustrated by means of our current example provides for any given multiple regular tree grammar a strongly equivalent finite-copying top-down tree transducer. He shows furthermore that an analogous result holds in the other direction. It is possible to specify an equivalent multiple regular tree grammar for any given finite-copying top-down tree transducer. Since our principal objective is to give a purely model-theoretic account of a grammatical framework that derives its inspiration from the basic idea that constituents move we will omit a discussion of this result.

Up to this moment we have merely moved from one system of formal language theory to the next one. It is now that we finally proceed to the logical model of tree transductions that we have to face the question what sort of logical definitions are to account for the tree transformations performed by the structure building operations *merge* and *move*. Implicitly, we have already answered this question with the move to the model of top-down tree transducers, as we will explain.

It has been known for some time that top-down tree transducers coincide with attributed tree transducers with synthesized attributes only (Courcelle and Franchi-Zannettacci, 1982a,b). This correspondence still holds if one considers the restricted case of the output of finite-copying top-down tree transducers and single use attributed tree transducers with synthesized attributes only, respectively, when applied to the family of regular tree languages (Engelfriet and Maneth, 1999). This last restricted family of single use attributed tree transducers with synthesized attributes only, in its turn, is equivalent to the family of direction preserving monadic second-order tree transducers when applied to the family of the regular tree languages (Bloem and Engelfriet, 2000).

Theorem 10.3.3. *For every minimalist grammar G, there exists a strongly equivalent direction preserving tree transducer T definable in monadic second-order logic.*

10.4 Conclusion

The last section has closed the last gap in the chain of equivalences leading from minimalist grammars to a restricted notion of grammar morphism. An important rôle was played by the notion of finite-copying top-down tree transducers connecting multiple regular tree grammars and direction preserving logical tree translations. This special device of syntax-directed interpretation transforms input trees into output trees in a strict recursive top-down manner without regard for any context information. That such a context-free account was possible for a grammatical framework firmly entrenched in the transformational tradition is due to the special form of minimal link condition embodied in the definition of the *move* operation by which it was required that there is exactly one maximal projection of licensee feature $-x$. This formulation provides the basis for the decomposition of minimalist expression trees into tuples of trees that are the appropriate input for the kind of rearrangements performed by multiple regular tree grammars. A similar analysis is not possible for the second-order operations of tree substitution permitted by the framework of tree adjoining grammars. Tree translations equivalent to this model of natural language syntax cannot be defined solely in terms of subtrees. Elements of context information have to be passed along either implicitly in terms of state parameters or explicitly in terms of inherited attributes.

Bibliography

Bloem, Roderick and Joost Engelfriet (2000). A Comparison of Tree Transductions Defined by Monadic Second-Order Logic and by Attribute Grammars. *J. Comp. System Sci.*, **61**:1–50.

Büchi, J. Richard (1960). Weak Second-order Arithmetic and Finite Automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, **6**:66–92.

Courcelle, Bruno and Paul Franchi-Zannettacci (1982a). Attribute grammars and recursive program schemes I. *Theor. Comput. Sci.*, **17**:163–191.

Courcelle, Bruno and Paul Franchi-Zannettacci (1982b). Attribute grammars and recursive program schemes II. *Theor. Comput. Sci.*, **17**:235–257.

Devlin, Keith (1991). *Logic and Information*. Cambridge University Press.

Doner, John (1970). Tree Acceptors and Some of Their Applications. *Journal of Computer and System Sciences*, **4**:406–451.

Elgot, Calvin C. (1961). Decision Problems of Finite Automata Design and Related Arithmetics. *Trans. Amer. Math. Soc.*, **98**:21–51.

Engelfriet, Joost and Linda Heyker (1991). The string-generating power of context-free graph grammars. *Journal of Computing Systems Science*, **43**:328–360.

Engelfriet, Joost and Sebastian Maneth (1999). Macro Tree Transducers, Attribute Grammars, and MSO Definable Tree Translations. *Information and Computation*, **154**:34–91.

Engelfriet, Joost and Sven Skyum (1976). Copying Theorems. *Information Processing Letters*, **4**:157–161.

Michaelis, Jens (2001a). *On Formal Properties of Minimalist Grammar*, volume 13 of *Linguistics in Potsdam*. Universität Potsdam.

Michaelis, Jens (2001b). Transforming linear context-free rewriting systems into minimalist grammars. In P. de Groote, G.F. Morrill, and C. Retoré, eds., *Logical Aspects of Computational Linguistics (LACL 2001)*, volume 2099 of *LNAI*, pp. 228–244. Berlin, Springer.

Michaelis, Jens, Uwe Mönnich, and Frank Morawietz (2001). On minimalist attribute grammars and macro tree transducers. In Christian Rohrer, Antje Roßdeutscher, and Hans Kamp, eds., *Linguistic Form and its Computation*, pp. 287–326. CSLI.

Rabin, Michael (1965). A simple method for undecidability proofs and some applications. In Y. Bar-Hillel, ed., *Logic Methodology and Philosophy of Science II*, pp. 58–68. North-Holland, Amsterdam.

Rabin, Michael (1977). Decidable theories. In Jon Barwise, ed., *Handbook of Mathematical Logic*, pp. 595–629. North-Holland.

Raoult, Jean-Claude (1997). Rational Tree Relations. *Bull. Belg. Math. Soc.*, **4**:149–176.

Stabler, Edward P. (1997). Derivational minimalism. In C. Retoré, ed., *Logical Aspects of Computational Linguistics (LACL '96)*, volume 1328 of *LNAI*. Springer, Berlin, Heidelberg.

Thatcher, James W. and Jesse B. Wright (1968). Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, **2**(1):57–81.

Chapter 11

Locality Conditions and the Complexity of Minimalist Grammars: A Preliminary Survey

Hans-Martin Gärtner
ZAS
Berlin, Germany
gaertner@zas.gwz-berlin.de
and
Jens Michaelis
Universität Osnabrück
Osnabrück, Germany
jmichael@uos.de

11.1 Introduction

Among the well-established variety of formal grammar types providing a *mildly context-sensitive grammar (MCSG)* formalism in the sense of Joshi (1985), Stabler's *minimalist grammars (MGs)* (Stabler 1997, 1999) come closest to modeling the tools used in the Chomskyan branch of generative syntax known as "minimalism" (Chomsky 1995, 2000, 2001). Interestingly, without there being a rise in (at least weak) generative power, (extensions and variants of) MGs accommodate a wide variety of (arguably) "odd" items from the syntactician's toolbox, such as *head movement* (Stabler 1997, 2001), *affix hopping* (Stabler 2001), *(strict) remnant movement* (Stabler 1997, 1999), and (to some extent) *adjunction* and *scrambling* (Frey and Gärtner 2002; Gärtner and Michaelis 2003). As a descendant of *transformational grammar (TG)*, minimalism carries over the division of labor between a phrase-structural and a transformational component. These find their way into MGs as operations *merge* and *move*, respectively. As is well-known, the *Aspects*-vintage of TG (Chomsky 1965) was shown to be Turing equivalent by Peters and Ritchie 1973. This led to intensive investigation into *locality conditions (LCs)* (Chomsky 1973, 1977, 1986; Rizzi 1990; Cinque 1990; Manzini 1992; among others) in an attempt to restrict the power of transformations. However, complexity results for these grammars with LC-add-ons are largely absent.[1] This picture has changed with MGs, as a formalized version of minimalism, which were shown to belong among the MCSG-formalisms in Michaelis 2001a. On the basis of this result it was possible to begin an investigation into how the complexity of MGs is affected by the presence or absence of various LCs. Here we are going to review this work and explore some directions for further research.

In particular we are going to look at the behavior and interaction of the *shortest move condition (SMC)*, the *specifier island condition (SPIC)* and the *adjunct island condition (AIC)*. It will turn out that different LCs have different effects on complexity. The original complexity result has been shown to hold for standard MGs incorporating the SMC. Adding the SPIC to standard MGs has non-monotonic consequences: whether complexity goes up or down depends on the absence or co-presence of the SMC, respectively (Section 11.3.1).[2]

For the AIC, the picture is more complicated. First of all, the AIC only makes sense if base-adjunction and adjunction by scrambling/extraposition is added to MGs (as suggested in Frey and Gärtner 2002; Gärtner and Michaelis 2003). Even more specifically, the AIC seems to make a difference if adjunction is allowed to occur countercyclically or *late*, i.e. if it is allowed to target a non-root constituent. Under these conditions, adding the AIC together with the SMC guarantees that the resulting grammars stay within the class of MCSGs. Without the AIC there are configurations that appear to go beyond. In MGs without the SMC, on the other hand, it is plausible to assume that the AIC does not change complexity at all, i.e. it is void (Section 11.3.2).

[1] One notable exception is the work of Rogers (1998), who proves the (strong) context-freeness of a restricted *government and binding (GB)* formalism, which he develops in terms of a (monadic second order) logical approach. In the connection of a logical formalization of GB(-like) structures also Kracht 1995b and 1995a (as well as Kracht's follow-up work) deserve attention. Some further relevant discussion can be found in the literature on *constraint-* or *principle-based parsing* such as, e.g., Cornell 1992, Stabler 1992, or Berwick 1991.

[2] A more general picture of the MCSG landscape is given in Figure 11.11 at the end of the paper (Page 96) where, in particular, we have the following abbreviations: TAG = tree adjoining grammars, LIG = linear indexed grammars, CCG = combinatory categorial grammars, HG = head grammars, LCFRS = linear context-free rewriting systems, MCTAG = (set local) multi-component tree adjoining grammars, IG = indexed grammars (cf. Joshi et al. 1991). An arrow always points to a class which is less powerful in generative capacity. If there is a double-arrow between two classes their generative capacity is equal.

Before we go into these particulars about LCs, we will provide a formal introduction to (the relevant variants of) MGs in Section 11.2. In a further outlook (Section 11.4), we sketch an MG-analysis of multiple wh-constructions and conclude with some general remarks about future research.

11.2 Minimalist Grammars

Throughout we let $\neg Syn$ and Syn be a finite set of *non-syntactic features* and a finite set of *syntactic features*, respectively, in accordance with (F1)–(F3) below. We take *Feat* to be the set $\neg Syn \cup Syn$.

(F1) $\neg Syn$ is disjoint from *Syn* and partitioned into the sets *Phon* and *Sem*, a set of *phonetic features* and a set *semantic features*, respectively.

(F2) *Syn* is partitioned into six sets:[3]

$$\begin{aligned} &Base,\\ &M\text{-}Select &&= \{\, =\mathsf{x} \mid \mathsf{x} \in Base \,\},\\ &A\text{-}Select &&= \{\, \approx\mathsf{x} \mid \mathsf{x} \in Base \,\},\\ &M\text{-}Licensors &&= \{\, +\mathsf{x} \mid \mathsf{x} \in Base \,\},\\ &M\text{-}Licensees &&= \{\, -\mathsf{x} \mid \mathsf{x} \in Base \,\}, \text{ and}\\ &S\text{-}Licensees &&= \{\, \sim\mathsf{x} \mid \mathsf{x} \in Base \,\}, \end{aligned}$$

the sets of *(basic) categories*, *m(erge)-selectors*, *a(djoin)-selectors*, *m(ove)-licensors*, *m(ove)-licensees*, and *s(cramble)-licensees*, respectively.

(F3) *Base* includes at least the category c.

We use *Licensees* as a shorthand denoting the set $M\text{-}Licensees \cup S\text{-}Licensees$.

Definition 11.2.1. An *expression (over Feat)*, also referred to as a *minimalist tree (over Feat)*, is a five-tuple $\langle N_\tau, \lhd_\tau^*, \prec_\tau, <_\tau, label_\tau\rangle$ obeying (E1)–(E3).

(E1) $\langle N_\tau, \lhd_\tau^*, \prec_\tau\rangle$ is a finite, binary (ordered) tree defined in the usual sense: N_τ is the finite, non-empty set of *nodes*, and $\lhd_\tau^*$ and $\prec_\tau$ are the respective binary relations of *dominance* and *precedence* on N_τ.[4]

(E2) $<_\tau \subseteq N_\tau \times N_\tau$ is the asymmetric relation of *(immediate) projection* that holds for any two siblings, i.e., for each $x \in N_\tau$ different from the root of $\langle N_\tau, \lhd_\tau^*, \prec_\tau\rangle$ either $x <_\tau sibling_\tau(x)$ or $sibling_\tau(x) <_\tau x$ holds.[5]

(E3) $label_\tau$ is the *leaf-labeling function* from the set of leaves of $\langle N_\tau, \lhd_\tau^*, \prec_\tau\rangle$ into $Syn^*\{\#\}Syn^*Phon^*Sem^*$.[6]

[3] Elements from *Syn* will usually be typeset in `typewriter font`.

[4] Thus, $\lhd_\tau^*$ denotes the reflexive-transitive closure of $\lhd_\tau$, the binary relation of *immediate dominance* on N_τ.

[5] For each $x \in N_\tau$ different from the root of $\langle N_\tau, \lhd_\tau^*, \prec_\tau\rangle$, $sibling_\tau(x)$ denotes the (unique) sibling. If $x <_\tau y$ for some $x, y \in N_\tau$ then x is said to *(immediately) project over* y.

[6] For each set M, M^* is the Kleene closure of M, including ε, the empty string. For any two sets of strings, M and N, MN is the product of M and N w.r.t. string concatenation. Further, $\#$ denotes a new symbol not appearing in *Feat*.

We take $Exp(Feat)$ to denote the class of all expressions over *Feat*.

Let $\tau = \langle N_\tau, \lhd_\tau^*, \prec_\tau, <_\tau, label_\tau\rangle \in Exp(Feat)$.

For each $x \in N_\tau$, the *head of x (in τ)*, denoted by $head_\tau(x)$, is the (unique) leaf of τ with $x \lhd_\tau^* head_\tau(x)$ such that each $y \in N_\tau$ on the path from x to $head_\tau(x)$ with $y \neq x$ projects over its sibling, i.e. $y <_\tau sibling_\tau(y)$. The *head of* τ is the head of τ's root. τ is said to be a *head* (or *simple*) if N_τ consists of exactly one node, otherwise τ is said to be a *non-head* (or *complex*).

A given expression $\phi = \langle N_\phi, \lhd_\phi^*, \prec_\phi, <_\phi, label_\phi\rangle$ belonging to $Exp(Feat)$ is a *subexpression of* τ in case $\langle N_\phi, \lhd_\phi^*, \prec_\phi\rangle$ is a subtree of $\langle N_\tau, \lhd_\tau^*, \prec_\tau\rangle$, $<_\phi = <_\tau \restriction_{N_\phi \times N_\phi}$, and $label_\phi = label_\tau \restriction_{N_\phi}$. Such a subexpression ϕ is a *maximal projection (in τ)* if its root is a node $x \in N_\tau$ such that x is the root of τ, or such that $sibling_\tau(x) <_\tau x$. $MaxProj(\tau)$ is the set of maximal projections in τ.

$comp_\tau \subseteq MaxProj(\tau) \times MaxProj(\tau)$ is the binary relation defined such that for all $\phi, \chi \in MaxProj(\tau)$ it holds that $\phi\, comp_\tau\, \chi$ iff $head_\tau(r_\phi) <_\tau r_\chi$, where r_ϕ and r_χ are the roots of ϕ and χ, respectively. If $\phi\, comp_\tau\, \chi$ holds for some $\phi, \chi \in MaxProj(\tau)$ then χ is a *complement of* ϕ *(in τ)*. $comp_\tau^+$ is the transitive closure of $comp_\tau$. $Comp^+(\tau)$ is the set $\{\phi \mid \tau\, comp_\tau^+\, \phi\}$.

$spec_\tau \subseteq MaxProj(\tau) \times MaxProj(\tau)$ is the binary relation defined such that for all $\phi, \chi \in MaxProj(\tau)$ it holds that $\phi\, spec_\tau\, \chi$ iff both $r_\chi = sibling_\tau(x)$ and $x <_\tau r_\chi$ for some $x \in N_\tau$ with $r_\phi \lhd_\tau^+ x \lhd_\tau^+ head_\tau(r_\phi)$, where r_ϕ and r_χ are the roots of ϕ and χ, respectively. If $\phi\, spec_\tau\, \chi$ for some $\phi, \chi \in MaxProj(\tau)$ then χ is a *specifier of* ϕ *(in τ)*. $Spec(\tau)$ is the set $\{\phi \mid \tau\, spec_\tau\, \phi\}$. Note that, if $Spec(\tau) \neq \emptyset$ then $Spec(\tau)$ is not necessarily a singleton set, but there is a unique specifier υ of τ, which we will refer to as the *highest specifier* of τ, such that the root of υ is immediately dominated by the root of τ.[7]

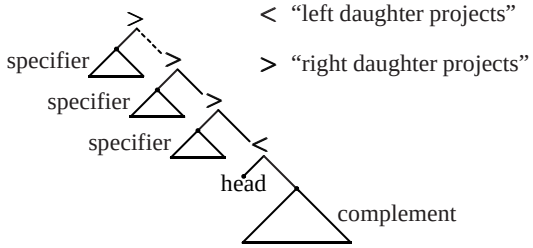


Figure 11.1: A typical minimalist tree.

A $\phi \in MaxProj(\tau)$ is said to *have*, or *display*, *(open) feature* f if the label assigned to ϕ's head by $label_\tau$ is of the form $\beta \# f \beta'$ for some $f \in Feat$ and some $\beta, \beta' \in Feat^*$.[8]

τ is *complete* if its head-label is in $Syn^*\{\#\}\{\mathsf{c}\}Phon^*Sem^*$, and each of its other leaf-

[7] Note that the leaf-labeling function $label_\tau$ can easily be extended to a total labeling function ℓ_τ from N_τ into $Feat^*\{\#\}Feat^* \cup \{<, >\}$, where $<$ and $>$ are two new distinct symbols: to each non-leaf $x \in N_\tau$ we can assign a label from $\{<, >\}$ by ℓ_τ such that $\ell_\tau(x) = <$ iff $y <_\tau z$ for $y, z \in N_\tau$ with $x \lhd_\tau y, z$, and $y \prec_\tau z$. In this sense a concrete $\tau \in Exp(Feat)$ is

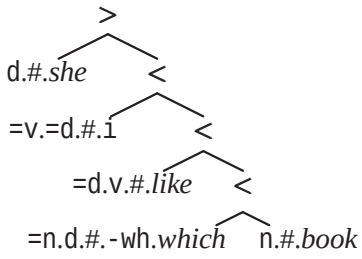


Figure 11.2: Example of a minimalist tree.

labels is in $Syn^*\{\#\}Phon^*Sem^*$. Hence, a complete expression over *Feat* is an expression that has category c, and this instance of c is the only instance of a syntactic feature which is preceded by an instance of # within its local leaf-label, i.e. the leaf-label it appears in.

The *phonetic yield of* τ, denoted by $Y_{Phon}(\tau)$, is the string which results from concatenating in "left-to-right-manner" the labels assigned via $label_\tau$ to the leaves of $\langle N_\tau, \lhd_\tau^*, \prec_\tau\rangle$, and replacing all instances of non-phonetic features with the empty string, afterwards.[9]

For two expressions $\phi,\chi \in Exp(Feat)$, $[_<\phi,\chi]$ (respectively, $[_>\phi,\chi]$) denotes the complex expression $\psi = \langle N_\psi, \lhd_\psi^*, \prec_\psi, <_\psi, label_\psi\rangle \in Exp(Feat)$ for which ϕ and χ are those two subexpressions such that $r_\psi \lhd_\psi r_\phi$, $r_\psi \lhd_\psi r_\chi$ and $r_\phi \prec_\psi r_\chi$, and such that $r_\phi <_\psi r_\chi$ (respectively $r_\chi <_\psi r_\phi$), where r_ϕ, r_χ and r_ψ are the roots of ϕ, χ and ψ, respectively.

For any $\phi,\chi,\psi \in Exp(Feat)$ such that χ is a subexpression of ϕ, $\phi\{\chi/\psi\}$ is the expression which results from substituting ψ for χ in ϕ.

As before we use *MG* as a shorthand for *minimalist grammar*.

Definition 11.2.2. An *MG without both SMC and SPIC (MG$^{/-,-/}$)* is a 5-tuple of the form $\langle\neg Syn, Syn, Lex, \Omega, c\rangle$ where Ω is the operator set consisting of the structure building functions $merge^{/-/}$ and $move^{/-,-/}$ defined as in (me-SPIC) and (mo-SMC,-SPIC) below, respectively, and where *Lex* is a *lexicon (over Feat)*, a finite set of simple expressions over *Feat*, with each lexical item $\tau \in Lex$ being of the form $\langle\{r_\tau\}, \lhd_\tau^*, \prec_\tau, <_\tau, label_\tau\rangle$ such that $label_\tau(r_\tau)$ is an element in $\{\#\}(M\text{-}Select \cup M\text{-}Licensors)^*Base\,M\text{-}Licensees^*Phon^*Sem^*$.

The operators from Ω build larger structure from given expressions by successively checking "from left to right" the instances of syntactic features appearing within the leaf-labels of the expressions involved. The symbol # serves to mark which feature instances have already been checked by the application of some structure building operation.

(me-SPIC) $merge^{/-/}$ is a partial mapping from $Exp(Feat) \times Exp(Feat)$ into $Exp(Feat)$. For any $\phi,\chi \in Exp(Feat)$, $\langle\phi,\chi\rangle$ is in $Dom(merge^{/-/})$ if for some category $x \in Base$ and $\alpha,\alpha',\beta,\beta' \in Feat^*$, conditions (me.i) and (me.ii) are fulfilled:[10]

(me.i) the head-label of ϕ is $\alpha\#{=}x\alpha'$ (i.e. ϕ displays m-selector =x),

(me.ii) the head-label of χ is $\beta\#x\beta'$ (i.e. χ displays category x).

Then,

(me.1) $merge^{/-/}(\phi,\chi) = [_<\phi',\chi']$ if ϕ is simple,

(me.2) $merge^{/-/}(\phi,\chi) = [_>\chi',\phi']$ if ϕ is complex,

where ϕ' and χ' result from ϕ and χ, respectively, just by interchanging the instance of # and the instance of the feature directly following the instance of # within the respective head-label (cf. Figure 11.3).

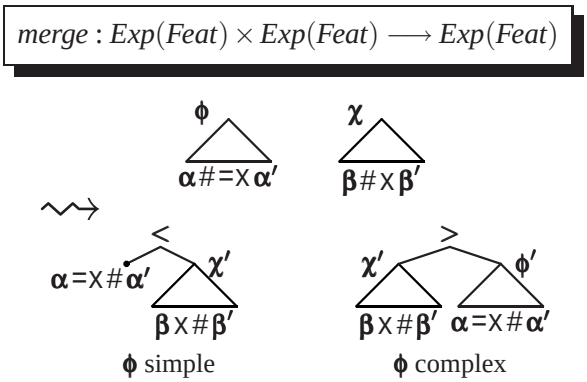


Figure 11.3: The merge-operator.

(mo-SMC,-SPIC) $move^{/-,-/}$ is a partial mapping from $Exp(Feat)$ to $\mathcal{P}_{\text{fin}}(Exp(Feat))$.[11] An expression $\phi \in Exp(Feat)$ is in $Dom(move^{/-,-/})$ if -x $\in$ *M-Licensees* and $\alpha,\alpha' \in Feat^*$ exist such that (mo.i) and (mo.ii) are true:

(mo.i) the head-label of ϕ is $\alpha\#{+}x\alpha'$ (i.e. ϕ displays licensor +x),

(mo.ii) there is a $\chi \in MaxProj(\phi)$ with head-label $\beta\#{-}x\beta'$ for some $\beta,\beta' \in Feat^*$ (i.e. $\chi \in MaxProj(\phi)$ exists displaying feature -x).

Then,

depictable in the way indicated in Figure 11.1.

[8] Thus, e.g., the expression depicted in Figure 11.2 has feature i, while there is a maximal projection which has feature -wh. For the sake of simplicity, we assume *she*, *like*, *which*, and *book* to be strings of phonetic features.

[9] Tree in Figure 11.2 has phonetic yield *she like which book*.

[10] For a partial function f from a class A into a class B, $Dom(f)$ is the domain of f, i.e., the class of all $x \in A$ for which $f(x)$ is defined.

[11] $\mathcal{P}_{\text{fin}}(Exp(Feat))$ is the class of all finite subsets of $Exp(Feat)$.

$$move^{/-,-/}(\phi) = \left\{ [_{>}\chi', \phi'] \;\middle|\; \begin{array}{l} \chi \in MaxProj(\phi) \text{ with} \\ \text{head-label } \beta\#\text{-}x\beta' \text{ for} \\ \text{some } \beta, \beta' \in Feat^* \end{array} \right\},$$

where ϕ' results from ϕ by interchanging the instance of # and the instance of +x directly following it within the head-label of ϕ, while the subtree χ is replaced by a single node labeled ε. χ' arises from χ by interchanging the instance of # and the instance of -x immediately to its right within the head-label of χ (cf. Figure 11.4).

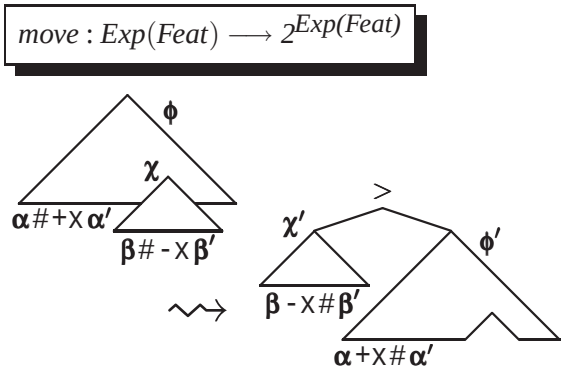


Figure 11.4: The move-operator.

Definition 11.2.3. An *MG without SMC, but with SPIC ($MG^{/-,+/}$)* is a five-tuple of the form $\langle \neg Syn, Syn, Lex, \Omega, c \rangle$ where Ω is the operator set consisting of the structure building functions $merge^{/+/}$ and $move^{/-,+/}$ defined as in (me^{+SPIC}) and ($mo^{-SMC,+SPIC}$) below, respectively, and where *Lex* is a lexicon over *Feat* defined as in Definition 11.2.2.

(me^{+SPIC}) $merge^{/+/}$ is a partial mapping from $Exp(Feat) \times Exp(Feat)$ into $Exp(Feat)$. For any $\phi, \chi \in Exp(Feat)$, $\langle \phi, \chi \rangle$ is in $Dom(merge^{/+/})$ if for some category $x \in Base$ and $\alpha, \alpha', \beta, \beta' \in Feat^*$, conditions (me.i) and (me.ii) above and (me.spic) are fulfilled:

(me.spic) if ϕ is complex then there is no $\psi \in MaxProj(\chi)$ with head-label $\gamma\#y\gamma'$ for some $y \in Licensees$ and $\gamma, \gamma' \in Feat^*$ (i.e. the selected specifier does not properly contain a maximal projection with an unchecked licensee feature instance).

Then, $merge^{/+/}(\phi, \chi) = merge^{/-/}(\phi, \chi)$.

($mo^{-SMC,+SPIC}$) The operator $move^{/-,+/}$ is a partial mapping from $Exp(Feat)$ to $\mathcal{P}_{fin}(Exp(Feat))$. A $\phi \in Exp(Feat)$ is in $Dom(move^{/-,+/})$ if for some -x $\in$ *M-Licensees* and $\alpha, \alpha' \in Feat^*$, (mo.i) and (mo.ii) given above and (mo.spic) are true:

(mo.spic) there is no $\psi \in MaxProj(\chi)$ different from χ, and with head-label $\gamma\#y\gamma'$ for some $y \in Licensees$ and $\gamma, \gamma' \in Feat^*$ (i.e. the maximal projection moved to the specifier does not itself properly contain itself a maximal projection displaying an unchecked syntactic feature instance).

Then, $move^{/-,+/}(\phi) = move^{/-,-/}(\phi)$.

The formulation of the SPIC as presented here, could be seen as an "active" variant, preventing the creation of expressions which include specifiers from which proper extraction could potentially take place. The MG-version presented in Stabler 1999 allows derivation of such expressions, but prevents these expressions to enter a convergent derivation by explicitly stating a "passive" formulation of the SPIC, demanding that the maximal projection $\chi \in MaxProj(\phi)$ which has feature -x can only move in order to check the licensee, if there exists a $\psi \in Comp^+(\phi)$ with $\chi = \psi$ or $\chi \in Spec(\psi)$.

Definition 11.2.4. An *MG with SMC, but without SPIC ($MG^{/+,-/}$)* is a five-tuple of the form $\langle \neg Syn, Syn, Lex, \Omega, c \rangle$ where Ω is the operator set consisting of the structure building functions $merge^{/-/}$ and $move^{/+,-/}$ defined as in (me^{-SPIC}) above and ($mo^{+SMC,-SPIC}$) below, respectively, and where *Lex* is a lexicon over *Feat* defined as in Definition 11.2.2.

($mo^{+SMC,-SPIC}$) The operator $move^{/+,-/}$ is a partial mapping from $Exp(Feat)$ to $\mathcal{P}_{fin}(Exp(Feat))$. A $\phi \in Exp(Feat)$ is in $Dom(move^{/+,-/})$ if for some -x $\in$ *M-Licensees* and $\alpha, \alpha' \in Feat^*$, (mo.i) and (mo.ii) above and (mo.smc) are true:

(mo.smc) exactly one $\chi \in MaxProj(\phi)$ exists with head-label $\gamma\#\text{-}x\gamma'$ for some $\gamma, \gamma' \in Feat^*$ (i.e. exactly one $\chi \in MaxProj(\phi)$ displays -x).[12]

Then, $move^{/+,-/}(\phi) = move^{/-,-/}(\phi)$.

Definition 11.2.5. An *MG with both SMC and SPIC ($MG^{/+,+/}$)* is a five-tuple of the form $\langle \neg Syn, Syn, Lex, \Omega, c \rangle$ where Ω is the operator set consisting of the structure building functions $merge^{/+/}$ and $move^{/+,+/}$ defined as in (me^{+SPIC}) above and ($mo^{+SMC,+SPIC}$) below, respectively, and where *Lex* is a lexicon over *Feat* defined as in Definition 11.2.2.

($mo^{+SMC,+SPIC}$) The operator $move^{/+,+/}$ is a partial mapping from $Exp(Feat)$ to $\mathcal{P}_{fin}(Exp(Feat))$. A $\phi \in Exp(Feat)$ is in $Dom(move^{/+,+/})$ if for some -x $\in$ *M-Licensees* and $\alpha, \alpha' \in Feat^*$, (mo.i), (mo.ii), (mo.spic) and (mo.smc) above are true.

Then, $move^{/+,+/}(\phi) = move^{/-,-/}(\phi)$.[13]

[12] Note that condition (mo.smc) implies (mo.ii).

[13] Note that the sets $move^{/+,-/}(\phi)$ and $move^{/+,+/}(\phi)$ in ($mo^{+SMC,-SPIC}$) and ($mo^{+SMC,+SPIC}$), respectively, both are singleton sets because of (mo.smc). Thus, the corresponding functions can easily be identified with one from $Exp(Feat)$ to $Exp(Feat)$.

Let $G = \langle \neg Syn, Syn, Lex, \Omega, c \rangle$ be an $MG^{/-,-/}$, $MG^{/-,+/}$, $MG^{/+,-/}$, respectively $MG^{/+,+/}$. For the sake of convenience, we refer to the corresponding merge- and move-operator in Ω by *merge* and *move*, respectively. Then the *closure of G*, $CL(G)$, is the set $\bigcup_{k\in\mathbb{N}} CL^k(G)$, where $CL^0(G) = Lex$, and for $k \in \mathbb{N}$,[14] $CL^{k+1}(G) \subseteq Exp(Feat)$ is recursively defined as the set

$$CL^k(G)$$
$$\cup \{merge(\phi,\chi) \mid \langle \phi,\chi\rangle \in Dom(merge) \cap CL^k(G) \times CL^k(G)\}$$
$$\cup \bigcup_{\phi \in Dom(move)\cap CL^k(G)} move(\phi).$$

The set $\{\tau \mid \tau \in CL(G) \text{ and } \tau \text{ complete}\}$, denoted $T(G)$, is the *minimalist tree language derivable by G*. The set $\{Y_{Phon}(\tau) \mid \tau \in T(G)\}$, denoted $L(G)$, is the *minimalist (string) language derivable by G*.

In the following we will use the notation $MG_{adj,ext}$ as a shorthand for *minimalist grammar with generalized adjunction and extraposition*.

Definition 11.2.6. An $MG_{adj,ext}$ *without both SMC and AIC* ($MG^{/-,-/}_{adj,ext}$) is a 5-tuple $G = \langle \neg Syn, Syn, Lex, \Omega, c \rangle$ where Ω is the operator set consisting of the functions $merge^{/-,-/}$, $move^{/-,-/}$, $adjoin^{/-/}$ and $scramble^{/-,-/}$ defined as in (me-SPIC) and (mo-SMC,-SPIC) above, and (ad-AIC) and (sc-SMC,-AIC) below, respectively, and where *Lex* is a *lexicon (over Feat)*, a finite set of simple expressions over *Feat*, and each lexical item $\tau \in Lex$ is of the form $\langle \{r_\tau\}, \lhd^*_\tau, \prec_\tau, <_\tau, label_\tau\rangle$ such that $label_\tau(r_\tau)$ belongs to $\{\#\}(M\text{-}Select \cup M\text{-}Licensors)^*(Base \cup A\text{-}Select)Licensees^* Phon^* Sem^*$.

(ad-AIC) $adjoin^{/-/}$ is a partial mapping from $Exp(Feat) \times Exp(Feat)$ into the class $\mathcal{P}_{fin}(Exp(Feat))$. A pair $\langle \phi,\chi\rangle$ with $\phi,\chi \in Exp(Feat)$ belongs to $Dom(adjoin^{/-/})$ if for some category $x \in Base$ and $\alpha,\alpha' \in Feat^*$, conditions (ad.i) and (ad.ii) are fulfilled:

(ad.i) the head-label of ϕ is $\alpha\#\approx x\alpha'$ (i.e. ϕ displays a-selector $\approx x$), and

(ad.ii) there exists some $\psi \in MaxProj(\chi)$ with head-label of the form $\beta\#x\beta'$ or $\beta x\beta'\#\beta''$ for some $\beta,\beta',\beta'' \in Feat^*$

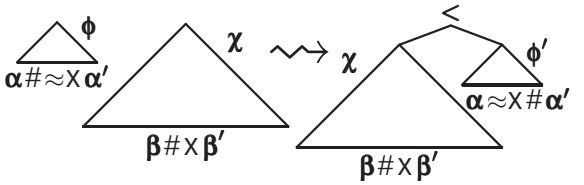
Then,

$$adjoin^{/-/}(\phi,\chi) = \left\{ \chi\{\psi/[_<\psi,\phi']\} \;\middle|\; \begin{array}{l}\psi \in MaxProj(\chi) \\ \text{with head-label} \\ \beta\#x\beta' \text{ or } \beta x\beta'\#\beta'' \\ \text{for some} \\ \beta,\beta',\beta'' \in Feat^*\end{array} \right\},$$

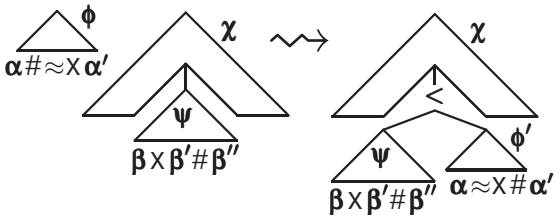
where ϕ' results from ϕ by interchanging the instances of # and $\approx x$, the latter directly following the former in the head-label of ϕ (cf. Figure 11.5).

[14] $\mathbb{N}$ is the set of all non-negative integers.

$$adjoin : Exp(Feat) \times Exp(Feat) \longrightarrow 2^{Exp(Feat)}$$



cyclic adjunction (Frey and Gärtner 2002)



acyclic/late adjunction (Gärtner and Michaelis 2003)

Figure 11.5: The adjoin-operator.

(sc-SMC,-AIC) The function $scramble^{/-,-/}$ maps partially from $Exp(Feat)$ into the class $\mathcal{P}_{fin}(Exp(Feat))$. A $\phi \in Exp(Feat)$ is in $Dom(scramble^{/-,-/})$ if for some $x \in Base$ and $\alpha,\alpha' \in Feat^*$, (sc.i) and (sc.ii) are true:

(sc.i) the head-label of ϕ is $\alpha\#x\alpha'$ (i.e. ϕ displays category x), and

(sc.ii) there is a $\chi \in MaxProj(\phi)$ with head-label $\beta\#\sim x\beta'$ for some $\beta,\beta' \in Feat^*$ (i.e. there is some $\chi \in MaxProj(\phi)$ displaying $\sim x$).

Then,

$$scramble^{/-,-/}(\phi) = \left\{ [_<\phi',\chi'] \;\middle|\; \begin{array}{l}\chi \in MaxProj(\phi) \text{ with} \\ \text{head-label } \beta\#\sim x\beta' \text{ for} \\ \text{some } \beta,\beta' \in Feat^*\end{array} \right\},$$

where $\phi' \in Exp(Feat)$ is identical to ϕ except for the fact that the subtree χ is replaced by a single node labeled ε. $\chi' \in Exp(Feat)$ arises from χ by interchanging the instance of # and the instance of $\sim x$ immediately to its right within the head-label of χ (cf. Figure 11.6).

Definition 11.2.7. An $MG_{adj,ext}$ *without SMC, but with AIC* ($MG^{/-,+/}_{adj,ext}$) is a five-tuple of the form $\langle \neg Syn, Syn, Lex, \Omega, c \rangle$ where Ω is the operator set consisting of the structure building functions $merge^{/-/}$, $move^{/-,-/}$, $adjoin^{/+/}$ and $scramble^{/-,+/}$ defined as in (me-SPIC) and (mo-SMC,-SPIC) above, and (ad+AIC) and (sc-SMC,+AIC) below, respectively, and where *Lex* is a lexicon over *Feat* defined as in Definition 11.2.6.

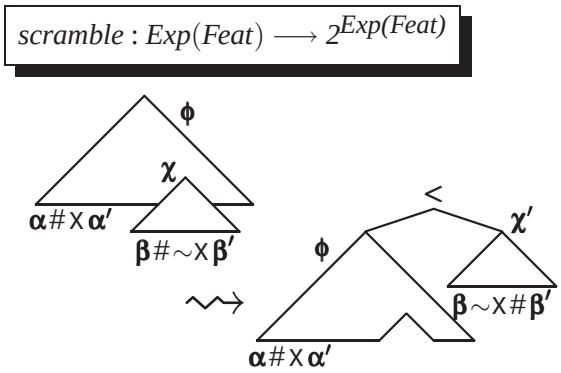


Figure 11.6: The scramble-operator.

(ad$^{+\text{AIC}}$) *adjoin*$^{/+/}$ is a partial mapping from $Exp(Feat) \times Exp(Feat)$ into the class $\mathcal{P}_{\text{fin}}(Exp(Feat))$. A pair $\langle \phi, \chi \rangle$ with $\phi, \chi \in Exp(Feat)$ belongs to $Dom(adjoin^{/+/})$ if for some category $x \in Base$ and $\alpha, \alpha' \in Feat^*$, conditions (ad.i) and (ad.ii) above and (ad.aic) are fulfilled:

(ad.aic) there is no $\psi \in MaxProj(\phi)$ with head-label $\gamma \# y \gamma'$ for some $y \in Licensees$ and $\gamma, \gamma' \in Feat^*$ (i.e. the adjunct does not properly contain a maximal projection with an unchecked syntactic feature instance).

Then, $adjoin^{/+/}(\phi, \chi) = adjoin^{/-/}(\phi, \chi)$.

(sc$^{\text{-SMC,+AIC}}$) The function *scramble*$^{/-,+/}$ maps partially from $Exp(Feat)$ into the class $\mathcal{P}_{\text{fin}}(Exp(Feat))$. A $\phi \in Exp(Feat)$ is in $Dom(scramble^{/-,+/})$ if for some $x \in Base$ and $\alpha, \alpha' \in Feat^*$, (sc.i) and (sc.ii) above and (sc.aic) are true:

(sc.aic) there is no $\psi \in MaxProj(\chi)$ different from χ, and with head-label $\gamma \# y \gamma'$ for some $y \in Licensees$ and $\gamma, \gamma' \in Feat^*$ (i.e. the maximal projection scrambled/extraposed to an adjunct position does not itself properly contain a maximal projection displaying an unchecked syntactic feature instance).

Then, $scramble^{/-,+/}(\phi) = scramble^{/-,-/}(\phi)$.

Definition 11.2.8. An $MG_{adj,ext}$ *with SMC, but without AIC* ($MG^{/+,-/}_{adj,ext}$) is a five-tuple of the form $\langle \neg Syn, Syn, Lex, \Omega, c \rangle$ where Ω is the operator set consisting of the structure building functions *merge*$^{/-/}$, *move*$^{/+,-/}$, *adjoin*$^{/-/}$ and *scramble*$^{/+,-/}$ defined as in (me$^{\text{-SPIC}}$), (mo$^{\text{+SMC,-SPIC}}$) and (ad$^{\text{-AIC}}$) above and (sc$^{\text{+SMC,-AIC}}$) below, respectively, and where *Lex* is a lexicon over *Feat* defined as in Definition 11.2.6.

(sc$^{\text{+SMC,-AIC}}$) *scramble*$^{/+,-/}$ is a partial mapping from $Exp(Feat)$ to $\mathcal{P}_{\text{fin}}(Exp(Feat))$. A tree $\phi \in Exp(Feat)$ is in $Dom(scramble^{/+,-/})$ if for some $x \in Base$ and $\alpha, \alpha' \in Feat^*$, (sc.i) and (sc.ii) above and (sc.smc) are true:

(sc.smc) exactly one $\chi \in MaxProj(\phi)$ exists with head-label $\gamma \# \sim x \gamma'$ for some $\gamma, \gamma' \in Feat^*$ (i.e. exactly one $\chi \in MaxProj(\phi)$ displays $\sim x$).[15]

Then, $scramble^{/+,-/}(\phi) = scramble^{/-,-/}(\phi)$.

Definition 11.2.9. An $MG_{adj,ext}$ *with both SMC and AIC* ($MG^{/+,+/}_{adj,ext}$) is a five-tuple of the form $\langle \neg Syn, Syn, Lex, \Omega, c \rangle$ where Ω is the operator set consisting of the structure building functions *merge*$^{/-/}$, *move*$^{/+,-/}$, *adjoin*$^{/+/}$ and *scramble*$^{/+,+/}$ defined as in (me$^{\text{-SPIC}}$), (mo$^{\text{+SMC,-SPIC}}$) and (ad$^{\text{+AIC}}$) above and (sc$^{\text{+SMC,+AIC}}$) below, respectively, and where *Lex* is a lexicon over *Feat* defined as in Definition 11.2.6.

(sc$^{\text{+SMC,+AIC}}$) *scramble*$^{/+,+/}$ is a partial mapping from $Exp(Feat)$ to $\mathcal{P}_{\text{fin}}(Exp(Feat))$. A tree $\phi \in Exp(Feat)$ is in $Dom(scramble^{/+,+/})$ if for some $x \in Base$ and $\alpha, \alpha' \in Feat^*$, (sc.i), (sc.ii), (sc.aic) and (sc.smc) above are true.

Then, $scramble^{/+,+/}(\phi) = scramble^{/-,-/}(\phi)$.[16]

Let $G = \langle \neg Syn, Syn, Lex, \Omega, c \rangle$ be an $MG^{/-,-/}_{adj,ext}$, $MG^{/-,+/}_{adj,ext}$, $MG^{/+,-/}_{adj,ext}$, respectively $MG^{/+,+/}_{adj,ext}$. For the sake of convenience, we refer to the corresponding merge-, move-, adjoin- and scramble-operator in Ω by *merge*, *move*, *adjoin* and *scramble*, respectively. Let $CL^0(G) = Lex$, and for each $k \in \mathbb{N}$, let $CL^{k+1}(G) \subseteq Exp(Feat)$ be recursively defined as

$$
\begin{aligned}
& CL^k(G) \\
\cup\ & \{ merge(\phi, \chi) \mid \\
& \qquad \langle \phi, \chi \rangle \in Dom(merge) \cap CL^k(G) \times CL^k(G) \} \\
\cup\ & \bigcup\nolimits_{\phi \in Dom(move) \cap CL^k(G)} move(\phi) \\
\cup\ & \bigcup\nolimits_{\langle \phi, \chi \rangle \in Dom(adjoin) \cap CL^k(G) \times CL^k(G)} adjoin(\phi, \chi) \\
\cup\ & \bigcup\nolimits_{\phi \in Dom(scramble) \cap CL^k(G)} scramble(\phi)
\end{aligned}
$$

Then, $\bigcup_{k \in \mathbb{N}} CL^k(G)$ is the *closure of G*, denoted $CL(G)$. The set $\{\tau \mid \tau \in CL(G) \text{ and } \tau \text{ complete}\}$, denoted $T(G)$, is the *minimalist tree language derivable by G*. The set $\{Y_{Phon}(\tau) \mid \tau \in T(G)\}$, denoted $L(G)$, is the *minimalist (string) language derivable by G*.

[15] Note that condition (sc.smc) implies (sc.ii).

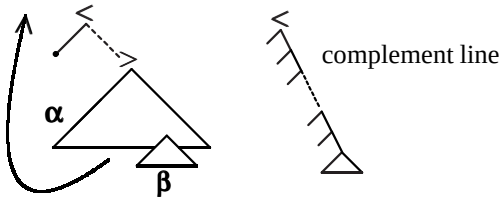
[16] $scramble^{/+,-/}(\phi)$ and $scramble^{/+,+/}(\phi)$ in (sc$^{\text{+SMC,-AIC}}$) and (sc$^{\text{+SMC,+AIC}}$), respectively, both are singleton sets because of (sc.smc). Thus, the corresponding functions can easily be identified with one from $Exp(Feat)$ to $Exp(Feat)$.

11.3 Locality Conditions and Complexity Results

11.3.1 The Specifier Island Condition

Figure 11.7 presents an example of a non-mildly context-sensitive MG not fulfilling the SMC but the SPIC, and deriving a language without constant growth property, namely, $\{ a^{2^n} \mid n \geq 0 \}$. The central column shows the lexical items as they are drawn from the lexicon, i.e., with all features unchecked. Arrows show the possible orders of interaction among lexical items and resulting constituents in terms of *merge*. Intermediate steps of *move* are left implicit.

As shown by Kobele and Michaelis (2005), not only this language, but in fact every language of type 0 can be derived by some MG not fulfilling the SMC but the SPIC for essentially two reasons: a) because of the SPIC,



movement of a constituent α into a specifier position freezes every proper subconstituent β within α, and b) without the SMC, therefore, the complement line of a tree (in terms of the successively embedded complements) can technically be employed as a queue. As is well-known, systems able to simulate queues are able to generate arbitrary type 0-languages.

Starting the "outer" cycle of our example in Figure 11.7, the currently derived tree shows 2^n+1 successively embedded complements on the complement line, all with an unchecked instance of `-l`, except for the lowest one, which displays `-m`. (*n* equals the number of cycles already completed.) The initializing selecting head `#.=v.-l` introduces an additional licensee `-l` to create string *a* on a cycleless derivation. Going through the cycle provides a successive bottom-to-top "roll-up" of those complements in order to check the displayed features. Thereby, 2^{n+1}+1 successively embedded complements on the complement line are created, again all displaying feature `-l` except for the lowest, which displays feature `-m`. Leaving the cycle procedure after a cycle has been completed leads to a final checking of the displayed licensees, where for each checked `-l` an *a* is introduced in the structure. This is the only way to create a convergent derivation.[17]

Figure 11.12 (Page 98) summarizes what we know about the interaction of SMC and SPIC,[18] where $\mathcal{L}_1 \searrow \mathcal{L}_2$, respectively $\mathcal{L}_2 \swarrow \mathcal{L}_1$, means "language class $\mathcal{L}_2$ is lower in generative capacity than language class $\mathcal{L}_1$" while $\mathcal{L}_1 \nearrow \mathcal{L}_2$, respectively $\mathcal{L}_2 \nwarrow \mathcal{L}_1$, means "language class $\mathcal{L}_2$ is higher in generative capacity than language class $\mathcal{L}_1$." Crucially, adding the SPIC can either properly reduce complexity (lower left side) or properly increase complexity (upper right side). What the SPIC does depends on the presence or absence of SMC. Its behavior is thus non-monotonic.

[17] For further details see Gärtner and Michaelis 2005.

[18] The MIX language is the language of all finite strings consisting of an equal number of *a*'s, *b*'s, and *c*'s appearing in arbitrary order.

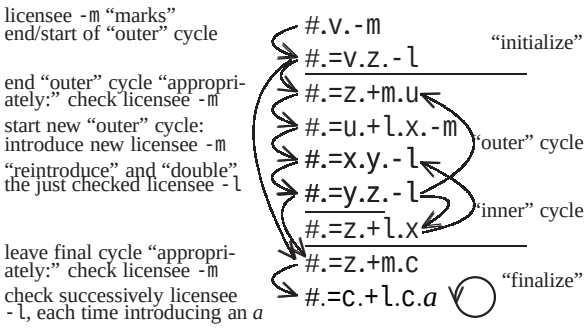


Figure 11.7: MG-example.

The SPIC, in fact, strictly reduces the generative capacity, when the SMC is present. Michaelis 2005 presents a string language which is derivable by an MGs obeying the SMC, but falls outside the class of string languages derivable by MGs obeying both the SMC and SPIC.[19]

11.3.2 The Adjunct Island Condition

In this section we look at MGs with *(late) adjunction* and *scrambling/extraposition* and the effects of imposing the AIC in a situation where the SMC alone appears to be too weak to guarantee mild context-sensitivity. Figure 11.8 gives a schematic illustration of countercyclic or late adjunction, i.e. adjunction to a non-root position.

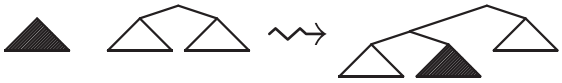


Figure 11.8: Countercyclic/late adjunction.

For the complexity issue we are interested in here it is important to note that late adjunction is capable of circumventing the SMC (cf. Gärtner and Michaelis 2003). (1) presents a case where this is actually welcome.

(1) [[[[Only those papers t_i]$_k$ did
[everyone t_j] read t_k]
[who was on the committee]$_j$]
[that deal with adjunction]$_i$]

We assume for simplicity that both of the relative clauses in (1) are extraposed by an application of right-

[19] More concretely, Michaelis 2005 proves the latter class to be (strictly) subsumed by the class of *indexed languages (ILs)* in the sense of Aho 1968, and the corresponding language presented as a case in point does, as shown in Staudacher 1993, not even belong to the class of ILs.

ward scrambling and are adjoined to CP. This is very roughly sketched in (2).

(2) * $[_{CP}\ CP_2\quad CP_1\ ]$ with α … α

For standard bottom-up derivations, (2) violates the SMC, given the simultaneous presence of alpha (i.e. $\sim$c) on both CPs. However, as sketched in (3), a derivational sequence of (first) extraposition, late adjunction and (second) extraposition voids this problem.

(3)	$[_{CP}\quad CP_1^{\alpha}\ ]$	start here
	$[_{CP}\quad -\ ]\ CP_1^{\not\alpha}$	move CP_1, check α
	$[_{CP}CP_2^{\alpha}\quad -\ ]\ CP_1^{\not\alpha}$	late adjoin CP_2
	$[_{CP}\ --\quad -\ ]\ CP_1^{\not\alpha}\ CP_2^{\not\alpha}$	move CP_2, check α

The proof that MGs without late adjunction, but obeying the SMC are mildly context-sensitive rests on the technical possibility of removing checked features from the structures. Formally, late adjunction creates a situation where in order to locate the individual adjunction sites, an a priori not bounded amount of (categorial) information has to be stored during a derivation, i.e., adjunction sites have to be kept accessible. Therefore it is unclear whether, in general, MGs allowing late adjunction still belong to the same complexity class. If, however, the AIC is imposed, we can apply a specific reduction method in proving that for the resulting MGs the old complexity result holds. Under this reduction, however, late adjunction can only be simulated if the adjunct does not properly contain constituents bearing unchecked licensees. But, this is exactly the situation where the AIC comes in. From a linguistic point of view it is rather natural to exclude extraction from adjuncts as Huang (1982) argued. This means that the weak generative capacity of MGs with late adjunction and extraposition can be kept within the bounds of standard MGs, i.e. mild context-sensitivity, if the AIC is imposed in addition to the SMC. Figure 11.13 (Page 98) summarizes our results for SMC/AIC-interaction.

11.4 Further Outlook

11.4.1 Multiple Wh-Constructions and the SMC

One phenomenon appearing to challenge the SMC adopted here is multiple wh-fronting in Slavic languages. Take, e.g., (4) from Bulgarian (Richards 2001, p. 249).

(4) *Koj$_i$ kogo$_j$ kakvo$_k$* t$_i$ *e pital* t$_j$ t$_k$
Who whom what AUX ask
'Who asked whom what?'

On standard assumptions, (4) requires three licensee instances of type -wh, which are successively checked in the C-domain. The required pre-movement representation, (5), is ruled out by (the strictest version of) the SMC.

(5) $[_{IP}$ -wh.koj e $[_{VP}$ pital -wh.kogo -wh.kakvo]]

A corresponding SMC-violation can be circumvented, however, if we adopt the wh-cluster hypothesis as argued for by Sabel (1998; 2001) and Grewendorf (2001) going back to Rudin (1988). Under this perspective, wh-expressions undergo successive cluster-formation before the resulting cluster takes a single wh-movement step, in compliance with the SMC. For this we have to add the feature type of *c(luster)-licensees* and *-licensors* to MGs.

c(luster)-licensees: $^{\triangle}x, {}^{\triangle}y, {}^{\triangle}z, \ldots$
c(luster)-licensors: $^{\triangledown}x, {}^{\triangledown}y, {}^{\triangledown}z, \ldots$

In Figure 11.9 we show a derivation with two wh-phrases. For cases with three or more such phrases the intermediate ones have to be of type d.$^{\triangledown}$wh.$^{\triangle}$wh.

Note that additional word order variation can be found in Bulgarian, as shown in (6) (Richards 2001, p. 249).

(6) *Koj kakvo kogo e pital*

This can be derived if cluster-formation is preceded by a scrambling-step of *kakvo* across *kogo* to VP, which requires it to be of type d.$\sim$v.$^{\triangledown}$wh. See Sabel (1998) for more discussion of wh- and focus-driven movements in multiple wh-configurations. A formal definition of the cluster-operator is given now.[20]

(cl$^{+\mathrm{SMC}}$) The operator *cluster* is a partial mapping from *Exp*(*Feat*) to *Exp*(*Feat*). An expression $\phi \in$ *Exp*(*Feat*) is in *Dom*(*cluster*) if there are a c-licensee $^{\triangle}x$ and $\alpha, \alpha' \in$ *Feat** such that (cl.i), (cl.ii) and (cl.smc) are true:

(cl.i) there is a $\chi \in$ *MaxProj*(ϕ) such that χ is the highest specifier of ϕ, and the head-label of χ is $\alpha\#^{\triangledown}x\alpha'$ (i.e. ϕ displays the corresponding c-licensor $^{\triangledown}x$),

(cl.ii) there is a $\psi \in$ *MaxProj*(ϕ) with head-label $\beta\#^{\triangle}x\beta'$ for some $\beta, \beta' \in$ *Feat** (i.e. $\psi \in$ *MaxProj*(ϕ) exists displaying $^{\triangle}x$).

(cl.smc) the existing $\psi \in$ *MaxProj*(ϕ) from (cl.ii) is unique (i.e. there is exactly one $\psi \in$ *MaxProj*(ϕ) displaying $^{\triangle}x$).

Then, $cluster(\phi) = \phi'\{\chi/[_{<}\chi', \psi']\}$,

[20] Given the "specifier condition" (cl.i), it is clear that—in order to potentially end up with a convergent derivation—within a lexical item an instance of a c-licensor must be immediately preceded by an instance of a basic category, a-selector, m-licensee, or s-licensee, i.e., in particular an instance of a c-licensor cannot be preceded by an instance of a c-licensee.

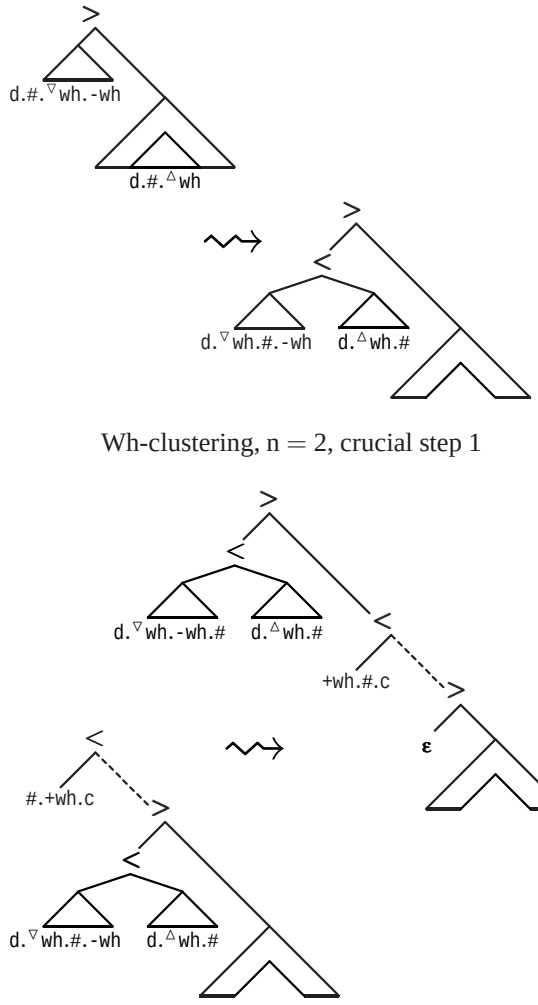


Figure 11.9: Wh-clustering involving c-licensors and c-licensees.

where ϕ' results from ϕ by replacing the subtree ψ by a single node labeled ε. χ' results from χ by interchanging the instances of # and $^{\triangledown}$x, the latter directly following the former in the head-label of χ, while ψ' results from ψ by interchanging the instances of # and $^{\triangle}$x, the latter directly following the former in the head-label of ψ (cf. Figure 11.10).[21]

Semantically, wh-cluster-formation can be interpreted as quantifier composition, a.k.a. "absorption" (Higginbotham and May 1981).

11.4.2 Future Research

There are two directions for future research that we consider of immediate relevance. First, it is necessary to develop a more systematic and complete combinatorics of LCs within and their complexity impact on MGs. Second, it is important to analyze the exact role LCs are playing in the other MCSG-frameworks (for TAGs see especially Frank 2002, for CCGs see Steedman 1996, for LIG-extensions see Wartena 1999), and try to establish the LCs' impact on complexity there. From the study of LCs within GB we already know that boundedness of chain overlap is crucial for $L^2_{K,P}$-definability of locality (Rogers 1998, p. 182, cf. the result on Scandinavian extraction in Miller 1991). This comes very close to the essence of what the SMC does within MGs. We also strongly suspect that it is the addition of *remnant movement (RM)* that puts MGs beyond context-freeness. A proof of the non-$L^2_{K,P}$-definability of recursively applicable RM would thus be a particularly interesting way of confirming this.

[21] As long as the SMC is obeyed, a proof showing that at least the weak generative capacity is unaffected seems to be straightforward by employing the "usual" reduction methods (cf. Michaelis 2001a).

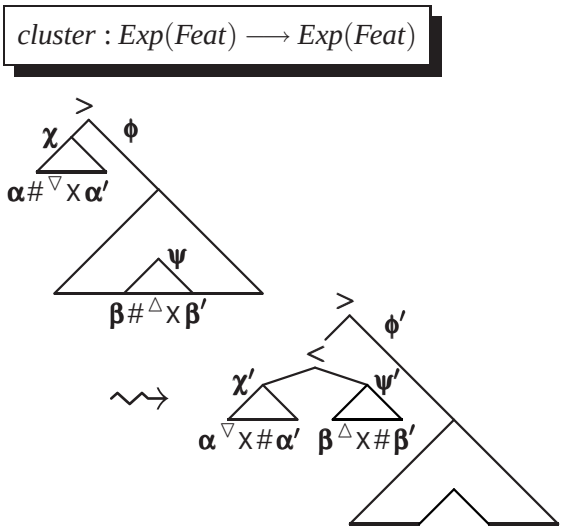


Figure 11.10: The cluster-operator.

Bibliography

Aho, Alfred V. (1968). Indexed grammars—An extension of context-free grammars. *Journal of the Association for Computing Machinery*, **15**:647–671.

Berwick, Robert C. (1991). Principle-based parsing. In Sells et al. 1991, pp. 115–226.

Chomsky, Noam (1965). *Aspects of the Theory of Syntax*. MIT Press, Cambridge, MA.

Chomsky, Noam (1973). Conditions on transformations. In S. Anderson and P. Kiparsky, eds., *A Festschrift for Morris Halle*, pp. 232–286. Holt, Rinehart and Winston, New York, NY.

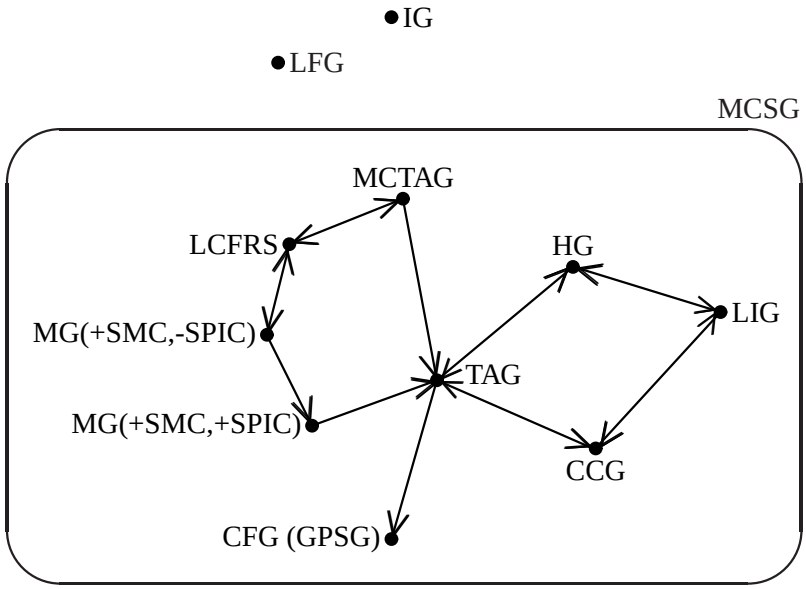


Figure 11.11: MCSG landscape.

Chomsky, Noam (1977). On wh-movement. In P. Culicover, T. Wasow, and A. Akmajian, eds., *Formal Syntax*, pp. 71–132. Academic Press, New York, NY.

Chomsky, Noam (1986). *Barriers*. MIT Press, Cambridge, MA.

Chomsky, Noam (1995). *The Minimalist Program*. MIT Press, Cambridge, MA.

Chomsky, Noam (2000). Minimalist inquiries. The framework. In R. Martin, D. Michaels, and J. Uriagereka, eds., *Step by Step. Essays on Minimalist Syntax in Honor of Howard Lasnik*, pp. 89–155. MIT Press, Cambridge, MA.

Chomsky, Noam (2001). Derivation by phase. In Michael Kenstowicz, ed., *Ken Hale. A Life in Language*, pp. 1–52. MIT Press, Cambridge, MA.

Cinque, Guglielmo (1990). *Types of A'-Dependencies*. MIT Press, Cambridge, MA.

Cornell, Thomas L. (1992). *Description Theory, Licensing Theory, and Principle-Based Grammars and Parsers*. Ph.D. thesis, University of California, Los Angeles, CA.

de Groote, Philippe, Glyn Morrill, and Christian Retoré, eds. (2001). *Logical Aspects of Computational Linguistics (LACL '01)*, LNAI Vol. 2099. Springer, Berlin, Heidelberg.

Frank, Robert (2002). *Phrase Structure Composition and Syntactic Dependencies*. MIT Press, Cambridge, MA.

Frey, Werner and Hans-Martin Gärtner (2002). On the treatment of scrambling and adjunction in minimalist grammars. In *Proceedings of the Conference on Formal Grammar (FGTrento)*, Trento, pp. 41–52.

Gärtner, Hans-Martin and Jens Michaelis (2003). A note on countercyclicity and minimalist grammars. In *Proceedings of the Conference on Formal Grammar (FGVienna)*, Vienna, pp. 103–114.

Gärtner, Hans-Martin and Jens Michaelis (2005). A note on the complexity of constraint interaction. Locality conditions and minimalist grammars. In P. Blache, E. Stabler, J. Busquets, and R. Moot, eds., *Logical Aspects of Computational Linguistics (LACL '05)*, LNAI Vol. 3492, pp. 114–130. Springer, Berlin, Heidelberg.

Grewendorf, Günther (2001). Multiple wh-fronting. *Linguistic Inquiry*, **32**:87–122.

Harkema, Henk (2001). A characterization of minimalist languages. In de Groote et al. 2001, pp. 193–211.

Higginbotham, James and Robert May (1981). Questions, quantifiers, and crossing. *The Linguistic Review*, **1**:41–79.

Huang, James C.-T. (1982). *Logical Relations in Chinese and the Theory of Grammar*. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA.

Joshi, Aravind K. (1985). Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions? In D. R. Dowty, L. Karttunen, and A. M. Zwicky, eds., *Natural Language Parsing. Psychological, Computational, and Theoretical Perspectives*, pp. 206–250. Cambridge University Press, New York, NY.

Joshi, Aravind K. K. Vijay-Shanker, and David J. Weir (1991). The convergence of mildly context-sensitive grammar formalisms. In Sells et al. 1991, pp. 31–81.

Kobele, Gregory M. and Jens Michaelis (2005). Two type-0 variants of minimalist grammars. In *FG-MoL 2005. The 10th conference on Formal Grammar and The 9th Meeting on Mathematics of Language*, Edinburgh, pp. 83–93.

Kracht, Marcus (1995a). Is there a genuine perspective on feature structures? *Linguistics and Philosophy*, **18**:401–458.

Kracht, Marcus (1995b). Syntactic codes and grammar refinement. *Journal of Logic, Language and Information*, **4**:41–60.

Manzini, Rita (1992). *Locality*. MIT Press, Cambridge, MA.

Michaelis, Jens (2001a). Derivational minimalism is mildly context-sensitive. In M. Moortgat, ed., *Logical Aspects of Computational Linguistics (LACL '98)*, LNAI Vol. 2014, pp. 179–198. Springer, Berlin, Heidelberg.

Michaelis, Jens (2001b). Transforming linear context-free rewriting systems into minimalist grammars. In de Groote et al. 2001, pp. 228–244.

Michaelis, Jens (2005). An additional observation on strict derivational minimalism. In *FG-MoL 2005. The 10th conference on Formal Grammar and The 9th Meeting on Mathematics of Language*, Edinburgh, pp. 103–113.

Miller, Philip H. (1991). Scandinavian extraction phenomena revisted. Weak and strong generative capacity. *Linguistics and Philosophy*, **14**:101–113.

Peters, Stanley and Graeme Ritchie (1973). On the generative power of transformational grammars. *Information Sciences*, **6**:49–84.

Richards, Norvin (2001). *Movement in Language. Interactions and Architectures*. Oxford University Press, Oxford.

Rizzi, Luigi (1990). *Relativized Minimality*. MIT Press, Cambridge, MA.

Rogers, James (1998). *A Descriptive Approach to Language-Theoretic Complexity*. Studies in Logic, Language and Information. CSLI Publications, Stanford, CA.

Rudin, Catherine (1988). On multiple questions and multiple wh-fronting. *Natural Language and Linguistic Theory*, **6**:445–501.

Sabel, Joachim (1998). Principles and parameters of wh-movement. Habilitationsschrift, Universität Frankfurt.

Sabel, Joachim (2001). Deriving multiple head and phrasal movement. The cluster hypothesis. *Linguistic Inquiry*, **32**:532–547.

Sells, Peter, Stuart M. Shieber, and Thomas Wasow, eds. (1991). *Foundational Issues in Natural Language Processing*. MIT Press, Cambridge, MA.

Stabler, Edward P. (1992). *The Logical Approach to Syntax*. MIT Press, Cambridge, MA.

Stabler, Edward P. (1997). Derivational minimalism. In C. Retoré, ed., *Logical Aspects of Computational Linguistics (LACL '96)*, LNAI Vol. 1328, pp. 68–95. Springer, Berlin, Heidelberg.

Stabler, Edward P. (1999). Remnant movement and complexity. In G. Bouma, G.-J. M. Kruijff, E. Hinrichs, and R. T. Oehrle, eds., *Constraints and Resources in Natural Language Syntax and Semantics*, pp. 299–326. CSLI Publications, Stanford, CA.

Stabler, Edward P. (2001). Recognizing head movement. In de Groote et al. 2001, pp. 245–260.

Staudacher, Peter (1993). New frontiers beyond context-freeness: DI-grammars and DI-automata. In *Sixth Conference of the European Chapter of the Association for Computational Linguistics (EACL '93)*, Utrecht, pp. 358–367. ACL.

Steedman, Mark (1996). *Surface Structure and Interpretation*. MIT Press, Cambridge, MA.

Wartena, Christian (1999). *Storage Structures and Conditions on Movement in Natural Language Syntax*. Ph.D. thesis, Potsdam University, Potsdam.

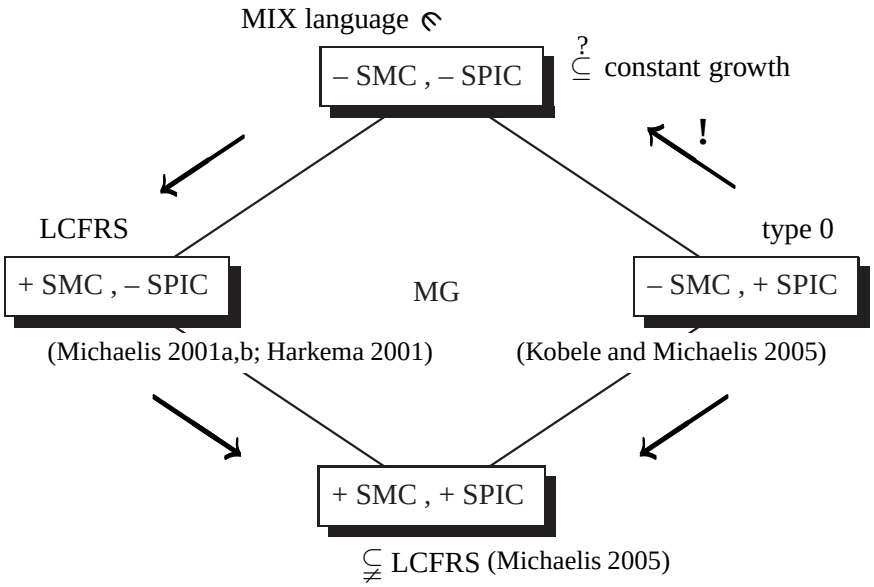


Figure 11.12: MG-diamond — Shortest move (SMC) and specifier islands (SPIC).

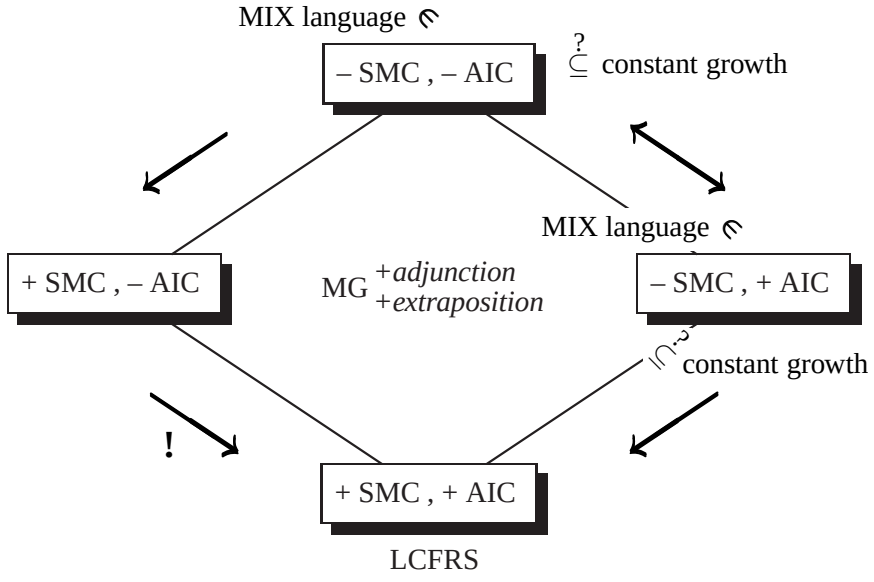


Figure 11.13: MG-diamond — Shortest Move (SMC) and Adjunct Islands (AIC).

Chapter 12

Closer to the Truth: A New Model Theory for HPSG

Frank Richter
Eberhard Karls Universität Tübingen
Tübingen, Germany
fr@sfs.uni-tuebingen.de

12.1 Introduction

HPSG is a model theoretic grammar framework in which a grammar is formulated as a pair consisting of (a) a signature which generates a space of possible structures and (b) a set of grammar principles which single out the well-formed structures among them. There are three proposals of how to precisely define the denotation of grammars within this general setting. Each proposal is accompanied by its own meta-theory of the ontological nature of the structures in the denotation of the grammar and their relationship to empirically observable phenomena. I will show that all three model theories face serious, if not fatal, problems: One of them makes very idiosyncratic fundamental assumptions about the nature of linguistic theories which many linguists might not share; the other two fail to capture the concepts they were designed to make mathematically precise. I will propose an alternative model theory which takes into account the shape of actual grammars and fixes the shortcomings of its predecessors.

12.2 The Plot

HPSG is an attractive candidate for studying a model theoretic linguistic framework. It has a history of over 20 years, many HPSG grammars of different languages have been written, and there are mathematically precise proposals about the denotation of HPSG grammars. Thus it is possible to take actual grammar specifications written by linguists and investigate the classes of structures the grammars denote according to the different model theories.

Here I want to take advantage of this fortunate situation to address the following questions:

1. Do the models of HPSG grammars meet the apparent intentions of the linguists who write them? And if they do not, how can we repair the problem(s) as conservatively as possible?

2. Are the structures in the denotation of the grammars actually compatible with the meta-theories of the meaning of grammars formulated within the HPSG framework?

The paper proceeds in three steps. Section 12.3 reviews problems with models of typical grammars (irrespective of the choice of meta-theory) and suggests universal restrictions on the form of HPSG grammars to amend them. Section 12.4 presupposes these amendments and investigates the models which the existing three meta-theories postulate. In response to the shortcomings we find, Section 12.5 proposes a new definition of the meaning of HPSG grammars, together with a meta-theory of the relationship between the set of structures denoted by an HPSG grammar and empirical linguistic phenomena. In the final section I conclude with a few remarks on the relationship of the new proposal to its predecessors.

For space reasons, I will concentrate on a rather informal discussion of the problems and their solutions. The presentation of the mathematical details is left for a different occasion.

12.3 Imprecise Grammars

Instead of taking a realistic grammar of a natural language as my object of study, I approach the questions of Section 12.2 with a very simple toy grammar which is built in such a way that it reflects crucial properties which all actual HPSG grammars in the literature share. This simplification helps to keep our modeling structures at a manageable (i.e., readable) size. Crucially, for our toy grammar below it will be obvious which structures form its intended denotation, and we can easily investigate whether the logical formalism supports the apparent expectations of the linguist.

12.3.1 An Example

An HPSG grammar consists of (a) a signature, Σ, declaring a sort hierarchy, attribute appropriateness conditions, and a set of relations and their arity, and (b) a set of logical statements, θ,usually called the *principles of grammar*. The grammar $\langle \Sigma_1, \theta_1 \rangle$ in (7) and (8) is a particularly simple example which, however, is structured like a typical linguistic grammar.

A most general sort, *top*, is the supersort of all other sort symbols in the sort hierarchy. The attributes PHON

(for phonology) and CAT (syntactic category) are appropriate to all signs, with values *list* and *cat*, respectively. Attribute appropriateness is inherited by more specific sorts, in this case *word* and *phrase*, with the possibility of subsorts adding further appropriate attributes. Here the sort *phrase* also bears the attributes H_DTR (head daughter) and NH_DTR (non-head daughter) for the syntactic tree structure. Another important feature of the present signature is the attribute SUBCAT, appropriate to *cat*. SUBCAT will be used for the selection of syntactic arguments. Finally, the signature introduces a relation symbol for a ternary relation, append.

(7) The signature Σ_1:

top
- *sign* — PHON *list*; CAT *cat*
  - *phrase* — H_DTR *sign*; NH_DTR *sign*
  - *word*
- *list*
  - *nelist* — FIRST *top*; REST *list*
  - *elist*
- *cat* — HEAD *head*; SUBCAT *list*
- *head*
  - *verb*
  - *noun*
- *phonstring*
  - *uther*
  - *walks*

append/3

The signature Σ_1 together with the theory θ_1 predicts exactly three well-formed signs: The words *Uther* and *walks* and the phrase *Uther walks*. The idea is that *Uther* and *walks* are not only words in our grammar, they may also occur as complete independent utterances, e.g. in exclamations and elliptical statements. θ_1 incorporates important HPSG principles: A WORD PRINCIPLE specifies the well-formed words, a (trivial) IMMEDIATE DOMINANCE (ID) PRINCIPLE specifies admissible phrase structures, a HEAD FEATURE PRINCIPLE makes category information travel up syntactic head projections, and a CONSTITUENT ORDER PRINCIPLE regulates word order. The last principle fixes the intended meaning of the relation symbol append.

(8) The theory θ_1:

a. WORD PRINCIPLE:

$$[\mathit{word}] \rightarrow \left(\begin{bmatrix} \text{PHON} & \langle \mathit{uther} \rangle \\ \text{CAT} & \begin{bmatrix} \text{HEAD} & \mathit{noun} \\ \text{SUBCAT} & \mathit{elist} \end{bmatrix} \end{bmatrix} \vee \begin{bmatrix} \text{PHON} & \langle \mathit{walks} \rangle \\ \text{CAT} & \begin{bmatrix} \text{HEAD} & \mathit{verb} \\ \text{SUBCAT} & \left\langle \begin{bmatrix} \text{HEAD} & \mathit{noun} \\ \text{SUBCAT} & \mathit{elist} \end{bmatrix} \right\rangle \end{bmatrix} \end{bmatrix} \right)$$

b. ID PRINCIPLE:

$$[\mathit{phrase}] \rightarrow \begin{bmatrix} \text{CAT SUBCAT} & \mathit{elist} \\ \text{H-DTR CAT SUBCAT} & \langle \boxed{1} \rangle \\ \text{NH-DTR CAT} & \boxed{1} \end{bmatrix}$$

c. HEAD FEATURE PRINCIPLE:

$$[\mathit{phrase}] \rightarrow \begin{bmatrix} \text{CAT HEAD} & \boxed{1} \\ \text{H\_DTR CAT HEAD} & \boxed{1} \end{bmatrix}$$

d. CONSTITUENT ORDER PRINCIPLE:

$$[\mathit{phrase}] \rightarrow \left(\begin{bmatrix} \text{PHON} & \boxed{3} \\ \text{H\_DTR PHON} & \boxed{2} \\ \text{NH\_DTR PHON} & \boxed{1} \end{bmatrix} \wedge \mathtt{append}(\boxed{1},\boxed{2},\boxed{3}) \right)$$

e. APPEND PRINCIPLE:

$$\forall\boxed{1}\forall\boxed{2}\forall\boxed{3} \left(\mathtt{append}(\boxed{1},\boxed{2},\boxed{3}) \leftrightarrow \left(\begin{array}{l} \left(\boxed{1}[\mathit{elist}] \wedge \boxed{2}[\mathit{list}] \wedge \boxed{2} = \boxed{3} \right) \vee \\ \exists\boxed{4}\exists\boxed{5}\exists\boxed{6} \left(\begin{array}{l} \boxed{1}\langle \boxed{4} \mid \boxed{5} \rangle \wedge \boxed{3}\langle \boxed{4} \mid \boxed{6} \rangle \\ \wedge\ \mathtt{append}(\boxed{5},\boxed{2},\boxed{6}) \end{array} \right) \end{array} \right) \right)$$

Models only contain objects labeled with maximally specific sorts (sorts without any proper subsorts in the sort hierarchy). For each appropriate attribute, there is one outgoing arc which points to an object labeled with an appropriate maximally specific sort. Informally, HPSG grammars denote a class of structures comprising all structures licensed by the signature such that all nodes in these structures also obey the well-formedness requirements imposed by the theory. In other words, the denotation of the grammar comprises at least one copy of each possible well-formed structure. Such 'complete' models are called *exhaustive models*.

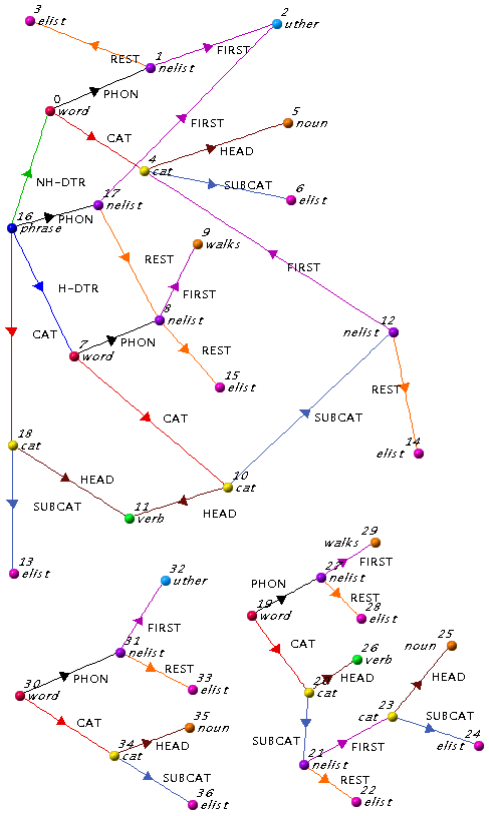
Which structures do linguists expect to find in the denotation of grammar $\langle \Sigma_1, \theta_1 \rangle$? Fig. 12.1 shows the most likely candidate (omitting the relation). The configuration with the phrasal root node 16 represents the sentence *Uther walks*; the configurations with root nodes 30 and 19 represent the words *Uther* and *walks*.

Upon reflection it is not difficult to see that these are by far not the only configurations licensed by our grammar. Three kinds of problems can be readily distinguished, which I will call the *intensionality of lists*, *twin structures*, and *stranded structures*.

The *intensionality of lists* is a side effect of the particular feature logical encoding of lists standardly adopted in HPSG. Consider the structure for the word *walks* under node 19 above. It contains three distinct *elist* objects (22, 24, 28) at the end of the PHON and SUBCAT lists of the verb and at the end of the SUBCAT list of its selected argument. Nothing in the grammar prevents any two or even all three *elist* objects from being the same object. This way we get five possible configurations for the word *walks* which the linguist presumably never intended to distinguish. We should clearly treat this ambiguity as an accident of encoding and get rid of it.

Twin structures are structures with more than one root node. For example, nothing would prevent the HEAD arc originating at the subcategorized object 23 in the word *walks* from pointing to the object 35 of the word *Uther* instead of to the object 25. The *noun* object 35 would then belong to the word *walks* and to the word *Uther*. No restrictions of the grammar would be violated, but what em-

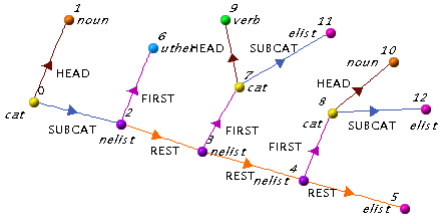
Figure 12.1: The intended $\langle \Sigma_1, \theta_1 \rangle$ model



pirical phenomenon should correspond to linguistic structure belonging to two (or even more) independent utterances? It seems obvious to me that this kind of configuration is not intended by linguists, and it should not occur in the intended models. In this paper I will not elaborate on the causes of the problem and on the full range of possible solutions. It will disappear as a side effect of the solution to the third problem of our grammar, stranded structures.

Stranded structures constitute the most serious one of the three types of problems with the grammar $\langle \Sigma_1, \theta_1 \rangle$. Stranded structures are typically structures which are 'smaller' than utterances. As an immediate consequence, they tend to be inaccessible to empirical observation. A trivial example is a configuration which looks just like the configuration under the *cat* object 34 of *Uther* in Fig. 12.1, the only difference being that there is no arc pointing to the *cat* object: It is stranded and inaccessible to empirical observation, since it is not connected to a phonological value. While some of the stranded structures in the denotation of grammars are isomorphic to structures which occur in observable linguistic signs (such as the one just described), *stranded monster structures* are of a shape which prevents them from being possible substructures of well-formed linguistic signs. Fig. 12.2 shows such a monster in the denotation of $\langle \Sigma_1, \theta_1 \rangle$.

Figure 12.2: A stranded monster structure in a $\langle \Sigma_1, \theta_1 \rangle$ model



$$\text{append} = \left\{ \begin{array}{l} \langle 2,5,2\rangle, \langle 3,5,3\rangle, \langle 4,5,4\rangle, \langle 5,2,2\rangle, \langle 5,3,3\rangle, \\ \langle 5,4,4\rangle, \langle 5,5,5\rangle, \langle 5,11,11\rangle, \langle 5,12,12\rangle, \\ \langle 11,2,2\rangle, \langle 11,3,3\rangle, \langle 11,4,4\rangle, \langle 11,5,5\rangle, \\ \langle 11,11,11\rangle, \langle 11,12,12\rangle, \langle 12,2,2\rangle, \langle 12,3,3\rangle, \\ \langle 12,4,4\rangle, \langle 12,5,5\rangle, \langle 12,11,11\rangle, \langle 12,12,12\rangle \end{array} \right\}$$

The monster in Fig. 12.2 is a nominal *cat* object whose SUBCAT list contains the phonetic string *Uther* and selects a verb and a noun. Although no such category exists in a word in the denotation of our grammar, it exists as a stranded structure because the constraints that prevent its existence in words all operate at the sign level. It is immediately clear that our grammar denotes infinitely many stranded monster structures. Even worse, the architecture of signs in HPSG and the standard grammar principles guarantee the existence of infinite classes of stranded monster structures in realistic grammars.

Contrary to first appearances, there is no simple remedy for this problem. Consider a brute force restriction which states that only configurations with root nodes of sort *word* and *phrase* may populate the linguistically relevant models, configurations which are empirically accessible through their phonology. However, there are phrases which require a licensing environment. In HPSG this environment may in fact contribute crucial structural restrictions, and its absence leads to absurd phrasal structures. Slashed constituents – phrases which contain an extraction site for a constituent without their corresponding filler – are a straightforward example. Their semantics will partly depend on the extracted constituent as recorded in the SLASH set. According to HPSG signatures, configurations in SLASH are smaller than signs (they are of sort *local*). Moreover, there are hardly any well-formedness restrictions on these *local* configurations as long as the extracted constituent is not realized as a sign in the syntactic tree. Therefore the configurations under *local* objects in the SLASH set of a slashed constituent without its complete licensing environment are usually not configurations which may actually occur in signs according to the grammar principles. A slashed constituent without its embedding matrix environment might thus have an arbitrary and even impossible semantics, due to the unrestricted *local* configuration in SLASH and its contribution to the meaning of the con-

stituent. This means that monster structures are back, and this time they even have a phonology and make empirically false predictions.

The grammars in the HPSG literature are not precise enough for their models to match the intentions of linguists. Independent of the choice of model theory they denote structures that their authors do not intend to predict. As the considerations about slashed constituents show, this is not a problem of the model theories. It is preferable to solve it by amending the grammars.

12.3.2 Normal Form Grammars

What we saw in the previous section was a weakness of the linguistic theory rather than of the logical formalism. Stranded structures are often inaccessible to empirical observation and should not be predicted. In grammars with interesting coverage stranded structures also materialize as phrasal stranded monster structures. These have a phonology, which means that they should be observable, but their internal structure prevents them from occurring as part of an actual utterance.

Appropriate extensions of the linguistic theory eliminate the spurious structures and can simply be added to most HPSG grammars. The extensions consist of general assumptions about the signature and of a number of logical statements to be included among the grammar principles.

The first move is to single out utterances from other types of signs as the only ones that are immediately empirically accessible. Every kind of linguistic structure is ultimately part of an utterance. Since no linguistic structure can simultaneously belong to two utterances, twin structures are ruled out. A minor technical amendment concerns lists: For their encoding we fix a unique structure that excludes spurious ambiguities that stem from multiple *elist* objects. In sum, I add to each HPSG grammar

(9) a. a sort hierarchy of signs which distinguishes unembedded signs from embedded signs,
b. an attribute, appropriate to each sort, which articulates the insight that each entity in the linguistic universe has the property of belonging to an unembedded sign,
c. a principle which requires that each entity be a component of an unembedded sign,
d. a principle which requires the uniqueness of unembedded sign entities in connected configurations of entities, and, finally,
e. a principle which formulates the weak extensionality of *elist* entities.

A grammar which incorporates these restrictions will be called a *normal form grammar*. The signature of the normal form grammar derived from the grammar $\langle \Sigma_1, \theta_1 \rangle$ is shown in (10). The hierarchy of signs distinguishes between unembedded signs (*u_sign*) and embedded signs (*e_sign*), a distinction which is inherited by words and phrases. Syntactic daughters are always embedded signs. The specification in the signature of the EMBEDDED value *u_sign* for each object ensures that every object in an interpretation is tied to an unembedded sign. The dots under *list* stand for all declarations under *list* in (7), including append.

(10) Normal form extension Σ_2 of signature Σ_1:

top EMBEDDED *u_sign*
sign PHON *list*
CAT *cat*
e_sign
e_word
e_phrase
u_sign
u_word
u_phrase
word
e_word
u_word
phrase H_DTR *e_sign*
NH_DTR *e_sign*
e_phrase
u_phrase
list
...
component/2

(11) shows the logical statements which must be added to the theory θ_1 in (8) to obtain the corresponding normal form grammar $\langle \Sigma_2, \theta_2 \rangle$. The new theory, θ_2, incorporates all principles from θ_1 in (8), adding four new restrictions on admissible models. For each of the new principles the corresponding formulation in (9) is indicated. The relation component is defined with respect to all attributes $\mathcal{A}$ in the signature. (11i) states that each pair of nodes *x* and *y* in a configuration is in the component relation iff a sequence of attributes leads from *y* to *x*.

(11) Normal form extension θ_2 of theory θ_1:[1]

f. (3c) U-SIGN COMPONENT CONDITION:
$\forall \boxed{1} \left(\boxed{1}[\mathit{top}] \rightarrow \exists \boxed{2} \mathtt{component}\left(\boxed{1}, \boxed{2}[\mathit{u\_sign}] \right) \right)$

g. (3d) UNIQUE U-SIGN CONDITION:
$\forall \boxed{1} \forall \boxed{2} \left(\left(\boxed{1}[\mathit{u\text{-}sign}] \wedge \boxed{2}[\mathit{u\text{-}sign}] \right) \rightarrow \boxed{1} = \boxed{2} \right)$

h. (3e) UNIQUE EMPTY LIST CONDITION:
$\forall \boxed{1} \forall \boxed{2} \left(\left(\boxed{1}[\mathit{elist}] \wedge \boxed{2}[\mathit{elist}] \right) \rightarrow \boxed{1} = \boxed{2} \right)$

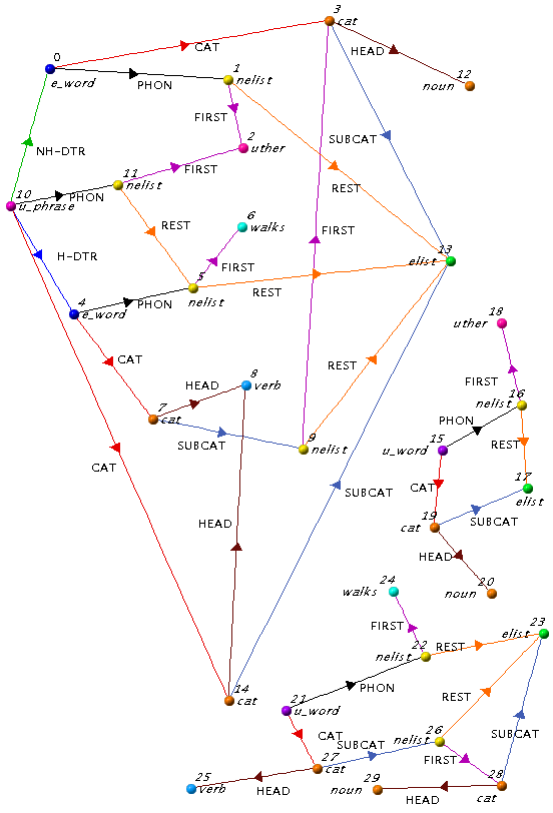
i. COMPONENT PRINCIPLE:
$$\forall \boxed{1} \forall \boxed{2} \left(\mathtt{component}(\boxed{1}, \boxed{2}) \leftrightarrow \left(\boxed{1} = \boxed{2} \vee \bigvee_{\alpha \in \mathcal{A}} \exists \boxed{3} \left(\boxed{2}[\alpha\ \boxed{3}] \wedge \mathtt{component}(\boxed{1}, \boxed{3}) \right) \right) \right)$$

The effect of normalizing the grammar $\langle \Sigma_1, \theta_1 \rangle$ can be inspected in Fig. 12.3. For readability I systematically omit the attribute EMBEDDED, which points from each node to the unique *u_sign* node to which the node belongs. For example, each node in the configuration with

[1] The logical expressions are RSRL descriptions (Richter, 2004). '∀' is not the first order universal quantifier.

the *u_phrase* 10 – representing the sentence *Uther walks* – has an outgoing EMBEDDED arc pointing to 10. The reader may want to verify that there are no other possible configurations in the denotation of the grammar. It should also be noted that the independent words *Uther* (under *u_word* node 15) and *walks* (under *u_word* node 21) are no longer isomorphic to the occurrences of these words in the sentence, because they are now marked as unembedded.

Figure 12.3: An exhaustive $\langle \Sigma_2, \theta_2 \rangle$ model, systematically omitting the attribute EMBEDDED for readability (see the explanation in the text)



$$\texttt{append} = \left\{ \begin{array}{l} \langle 1,13,1\rangle, \langle 1,5,11\rangle, \langle 13,1,1\rangle, \langle 13,13,13\rangle, \langle 13,5,5\rangle, \\ \langle 13,9,9\rangle, \langle 13,11,11\rangle, \langle 5,13,5\rangle, \langle 9,13,9\rangle, \\ \langle 11,13,11\rangle, \langle 16,17,16\rangle, \langle 17,16,16\rangle, \langle 17,17,17\rangle, \\ \langle 22,23,22\rangle, \langle 23,22,22\rangle, \langle 23,23,23\rangle, \langle 23,26,26\rangle, \\ \langle 26,23,26\rangle \end{array} \right\}$$

$$\begin{array}{rl} \texttt{component} = & \{0,1,\ldots,14\} \times \{0,1,\ldots,14\} \cup \\ & \{15,16,\ldots,20\} \times \{15,16,\ldots,20\} \cup \\ & \{21,22,\ldots,29\} \times \{21,22,\ldots,29\} \end{array}$$

12.4 Problems in Previous Model Theories

On the basis of the notion of *normal form HPSG grammars* I can now investigate the previous mathematical characterizations of the meaning of HPSG grammars. These are (1) Pollard and Sag's original theory of linguistic utterance types modeled by abstract feature structures (Pollard and Sag 1994), (2) Pollard's theory of mathematical idealizations of utterance tokens (Pollard 1999), and (3) King's theory of exhaustive models containing sets of possible utterance tokens (King 1999). In order to make sure that all three logical formalisms can easily be compared and are comprehensive enough for a full formalization of HPSG grammars of the kind introduced by Pollard and Sag (1994), I use them in their variants defined in (Richter, 2004), which expresses them in terms of Relational Speciate Re-entrant Language (RSRL).

12.4.1 Informal Overview

The formalization of the model theory of (1) and (2) fails to produce models that agree with their respective meta-theories of the structures in their grammar models. In essence, the problem is that both (1) and (2) intend to capture the idea that for each isomorphism class of well-formed utterances in a language, we find exactly one structure in the denotation of the grammar which models the members of the isomorphism class. For example, take a realization of the utterance *I am sitting in a 370 year old house in Engadin.* The intention of the model theory of (1) is to have exactly one abstract feature structure in the denotation a grammar of English which models – or stands for the utterance type of – the utterance token. Similarly, the intention of the model theory of (2) is to have exactly one mathematical idealization of the isomorphism class of tokens of the given sentence in the denotation of the grammar. However, this intention is not borne out in either formalism. Their models are defined in such a way that we necessarily find a large number of modeling structures for the given sentence in the denotation of a correct grammar of English. Subsection 12.4.2 sketches the properties of the formalisms which are responsible for this result.

The problem with (3) is not of a technical nature, it comes from the meta-theory itself. King postulates that the intended model of a grammar is an *exhaustive model* like the one shown in Fig. 12.3 for the grammar $\langle \Sigma_2, \theta_2 \rangle$. According to King, the exhaustive model of a language that the linguist aims for does not contain utterance types or mathematical idealizations of utterance tokens. Instead it contains the utterance tokens of the language themselves. Since we cannot know how many tokens of a given utterance there have been and will be in the world, we never know how many isomorphic copies of each utterance token the intended model contains. The definition of exhaustive models permits an arbitrary number of isomorphic copies of each possible configuration, all that is required is the presence of at least one representative of each. From the definition we only know that the class of exhaustive models of a grammar comprises, among many others, the particular exhaustive model which, for each utterance, contains the right number of tokens (if the grammar is correct). However, since there will be grammatical utterances of a language which have never occurred and will never occur, this is not yet the full

story. As exhaustive models (by definition) contain at least one copy of each potential grammatical utterance in the language, the intended exhaustive model must also comprise *possible* (as opposed to actual) utterance tokens, at least for those well-formed utterances of a language which never occur. This means that the configurations in exhaustive models are *potential utterance tokens*. These potential utterance tokens are a dubious concept if tokens are supposed to be actual occurrences of a linguistic form. In light of this problem, King's model theory has been unacceptable to some linguists.

12.4.2 Details

In this section I substantiate my claim that the model theories based on abstract feature structures by Pollard and Sag (1994) and on mathematical idealizations of linguistic utterance tokens by Pollard (1999) do not achieve what their meta-theories call for. Henceforth I refer to these two theories as AFS and MI, respectively.

Let us first consider AFS. The underlying idea is that the denotation of a grammar is a set of *relational abstract feature structures* as determined by an *admission relation*. Each abstract feature structure in the set of relational abstract feature structures admitted by a grammar is a unique representative of exactly one utterance type of the natural language which the grammar is supposed to capture. This means that there is a one-to-one correspondence between the utterance types of the natural language and the abstract feature structures which the grammar admits. A grammar can then be falsified by showing either that there is no feature structure admitted by the grammar which corresponds to a particular utterance type of the language or that the grammar admits an abstract feature structure which does not correspond to any grammatical utterance type in the language.

Relational abstract feature structures consist of four sets: A *basis set*, β, which provides the basic syntactic material; a *re-entrancy relation*, ρ, which is an equivalence relation that can be understood as an abstract representation of the nodes in connected configurations; a *label function*, λ, which assigns species to the abstract nodes; and a *relation extension*, symbolized below as ξ, which represents the tuples of abstract nodes which are in the relations of a grammar.

How these four components of a relational abstract feature structure conspire to produce a representation of the utterance type *Uther* from Fig. 12.3 can be seen in (12).[2] The symbol ε stands for the empty path, i.e., an empty sequence of attributes. The basis set, β_U, contains all attribute paths which can be created by following sequences of arcs from 15. The re-entrancy relation, ρ_U, enumerates all possibilities of getting to the same node by a pair of attribute paths; and the label function, λ_U, assigns the correct species to each attribute path.

[2]For expository purposes I pretend that the attribute EMBEDDED is not in the grammar. See footnote 3 for further remarks on this simplification.

(12) $\mathbb{A}_{\text{Uther}} = \langle \beta_U, \rho_U, \lambda_U, \xi_U \rangle$ with

$$\beta_U = \left\{ \begin{array}{l} \varepsilon, \text{PHON}, \text{PHON REST}, \text{PHON FIRST}, \\ \text{CAT}, \text{CAT SUBCAT}, \text{CAT HEAD} \end{array} \right\},$$

$$\rho_U = \left\{ \begin{array}{l} \langle \varepsilon, \varepsilon \rangle, \langle \text{PHON}, \text{PHON} \rangle, \langle \text{CAT}, \text{CAT} \rangle, \\ \langle \text{PHON FIRST}, \text{PHON FIRST} \rangle, \\ \langle \text{PHON REST}, \text{PHON REST} \rangle, \\ \langle \text{PHON REST}, \text{CAT SUBCAT} \rangle, \\ \langle \text{CAT SUBCAT}, \text{PHON REST} \rangle, \\ \langle \text{CAT SUBCAT}, \text{CAT SUBCAT} \rangle, \\ \langle \text{CAT HEAD}, \text{CAT HEAD} \rangle \end{array} \right\},$$

$$\lambda_U = \left\{ \begin{array}{l} \langle \varepsilon, \textit{u\_word} \rangle, \langle \text{PHON}, \textit{nelist} \rangle, \\ \langle \text{PHON REST}, \textit{elist} \rangle, \\ \langle \text{CAT SUBCAT}, \textit{elist} \rangle, \\ \langle \text{PHON FIRST}, \textit{uther} \rangle, \langle \text{CAT}, \textit{cat} \rangle, \\ \langle \text{CAT HEAD}, \textit{noun} \rangle \end{array} \right\},$$

$$\xi_U = \left\{ \begin{array}{l} \langle \texttt{append}, \text{PHON}, \text{PHON REST}, \text{PHON} \rangle, \\ \langle \texttt{append}, \text{PHON REST}, \text{PHON}, \text{PHON} \rangle, \\ \langle \texttt{append}, \text{PHON}, \text{CAT SUBCAT}, \text{PHON} \rangle, \\ \langle \texttt{append}, \text{CAT SUBCAT}, \text{PHON}, \text{PHON} \rangle \end{array} \right\}$$

$$\cup \left\{ \langle \texttt{append}, \pi_1, \pi_2, \pi_3 \rangle \mid \begin{array}{l} \pi_1, \pi_2, \pi_3 \in \\ \left\{ \begin{array}{l} \text{PHON REST}, \\ \text{CAT SUBCAT} \end{array} \right\} \end{array} \right\}$$

$$\cup \left\{ \langle \texttt{component}, \pi_1, \pi_2 \rangle \mid \begin{array}{l} \pi_1 \in \beta_U, \pi_2 \in \beta_U, \& \\ \pi_1 = \pi_2 \text{ or} \\ \pi_2 \text{is a prefix of } \pi_1 \end{array} \right\}$$

Note that the set theoretical definition of abstract feature structures guarantees that every abstract feature structure isomorphic to another one is identical with it.

Figure 12.4: The utterance type *Uther* and its reducts, without relations and the EMBEDDED attribute

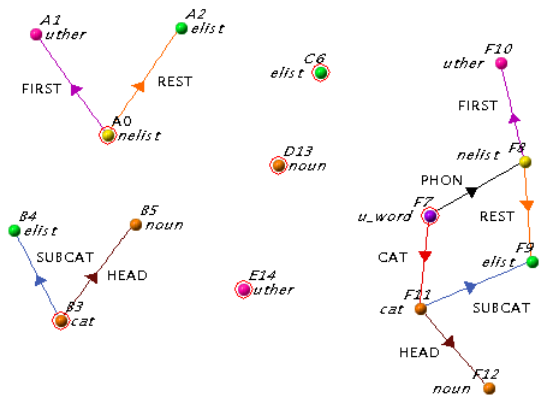


Fig. 12.4 repeats the *Uther* configuration from Fig. 12.3 and adds a few more configurations. They are all rooted at a distinguished node (marked by a circle). The significance of the new configurations is the fact that the set of abstract feature structures admitted by our grammar does not only contain the abstract feature structure corresponding to the *Uther* configuration under *F7* (beside the two corresponding to *walks* and *Uther walks*).

Since the abstract feature structure for *Uther* is in the set, it also contains abstract feature structures corresponding to the configurations under $A0$, $B3$, $C6$, $D13$ and $E14$.

The reason for this is to be found in the definition of relational abstract feature structures and the ensuing admission relation based on the traditional satisfaction relation for feature structures, and it is an artifact of the construction. Intuitively, this is what happens: Abstract feature structures lack an internal recursive structure. Since the admission relation must ensure that the entire abstract feature structure including all of its abstract nodes satisfies the set of principles of a grammar, an auxiliary notion of reducts provides the necessary recursion. The idea is that a relational abstract feature structure is admitted by a theory if and only if the feature structure itself and all its reducts satisfy the theory. But that means that not only the relational abstract feature structure but also all of its reducts are in the set of abstract feature structures admitted by the theory.

The definition of reducts is straightforward. Any attribute path in the basis set may be followed to get to an abstract node in the feature structure. At the end of each path we find a new abstract root node of a reduct. This can best be seen by considering the corresponding pictures of configurations in Fig. 12.4 again. The configuration under $A0$ corresponds to the PHON reduct of the *Uther* configuration; the configuration under $B3$ corresponds to the CAT reduct of the *Uther* configuration; $C6$ to the PHON REST and CAT SUBCAT reduct; and analogously for the two remaining atomic configurations. (13) contains an example of the reducts depicted in Fig. 12.4, an abstract feature structure corresponding to the configuration with root node $E14$. The reducts can be obtained either by abstraction from the configurations in Fig. 12.4 or directly from $\mathbb{A}_{\text{Uther}}$ by a reduct formation operation. In contrast to the depictions of the corresponding graphical configuration in Fig. 12.4, the PHON FIRST reduct of *Uther* in (13) contains the relation(s).

(13) The PHON FIRST reduct of $\mathbb{A}_{\text{Uther}}$:
$\beta_{\text{PF}} = \{\varepsilon\}$,
$\rho_{\text{PF}} = \{\langle \varepsilon, \varepsilon \rangle\}$,
$\lambda_{\text{PF}} = \{\langle \varepsilon, \textit{uther} \rangle\}$, and
$\xi_{\text{PF}} = \{\langle \texttt{component}, \varepsilon, \varepsilon \rangle\}$.

The scientific purpose of relational abstract feature structures in linguistic theory is their use as conveniently structured mathematical entities which correspond to types of linguistic entities. The relational abstract feature structures admitted by a grammar are meant to constitute the predictions of the grammar (Pollard and Sag, 1994, p. 8).

In the context of our example, we are talking about one empirical prediction of the grammar $\langle \Sigma_2, \theta_2 \rangle$, the prediction that the described language contains the utterance *Uther*. The exhaustive models mirror this prediction by containing (potentially multiple but isomorphic) *Uther* configurations. There is nothing else in the exhaustive models which has to do with this particular prediction of the grammar. The abstract feature structures admitted by the grammar predict six different types for this single expression. The six types are distinct, and they are unavoidable by construction if the grammar predicts the relational abstract feature structure which is an abstraction of a *Uther* configuration. The fundamental problem of the construction is that the well-formedness of $\mathbb{A}_{\text{Uther}}$ is only guaranteed by the well-formedness of all of its reducts. Hence we do not get a one-to-one correspondence between the types predicted by the grammar and the empirically observable expressions. Rather, it is the case that the abstract feature structures admitted by a grammar necessarily introduce a version of stranded structures, although there are no stranded monster structures among them as long as the grammar is a normal form grammar.[3]

I conclude that AFS fails to behave in the intended way. Even if one is willing to accept types of linguistic expressions as an appropriate target for linguistic theory, relational abstract feature structures are not adequate to make this approach to the theory of grammatical meaning technically precise.

Let us now turn to the second theory, MI. Pollard (1999) postulates that a formal grammar as a scientific theory should predict the grammatical utterance tokens of a natural language by specifying a set of structures which contains an idealized mathematical structure for each utterance token (and for nothing else). For two utterance tokens of the same expression there should only be one mathematical structure in the set. Moreover, the idealized mathematical structure should be structurally isomorphic to the utterance tokens it represents. This last condition is in fact much stronger than what (Pollard and Sag, 1994) asks from its linguistic types. Pollard and Sag's linguistic types merely stand in a relationship of conventional correspondence to utterance tokens. The conventional correspondence must be intuited by linguists without any further guidance with respect to the correctness of these intuitions from the meta-theory of linguistic meaning.

The most significant technical difference compared to AFS resides in how Pollard sets out to construct the mathematical idealizations of utterance tokens. Pollard's construction eschews relational abstract feature structures and consequently does not need the specialized feature structure satisfaction and admission relations of strictly feature structure based grammar formalisms. Instead, Pollard starts from the conventional grammar models of King (1999). From these standard models he proceeds to define *singly generated models* and then canonical representatives of singly generated models as mathematical idealizations of utterance tokens.

A singly generated model is a connected configuration under an entity which is actually a model of a grammar.

[3] Nothing substantial changes when we include the structure generated by the attribute EMBEDDED in the relational abstract feature structures. All four component sets of $\mathbb{A}_{\text{Uther}}$ as well as those of its five reducts become infinite, but the six feature structures remain distinct mathematical entities seemingly representing six different linguistic types.

In other words, a singly generated model has a topmost entity such that all other entities in the model are components of it. However, this is not yet the whole picture. Pollard defines the structures of interest as models together with their distinguished topmost entity. They are pairs, $\langle u, \langle \mathsf{U}_u, \mathsf{S}_u, \mathsf{A}_u, \mathsf{R}_u \rangle \rangle$, usually simply written as $\langle u, \mathsf{I}_u \rangle$.[4] The subscripts indicate that all entities in the universe U are components of u. We could say that I is a connected configuration under u which happens to be a model of a given grammar. Pollard then uses the distinguished entity in the configuration to define the canonical representative for each $\langle u, \mathsf{I}_u \rangle$ of the grammar. In essence, the entities in the canonical representatives are defined as equivalence classes of terms relative to the distinguished root entity. Not all details are relevant here,[5] the only important thing to note is that the standard model-theoretic technique of using terms of the logical language in the construction of a canonical model guarantees the uniqueness of each $\langle u, \langle \mathsf{U}_u, \mathsf{S}_u, \mathsf{A}_u, \mathsf{R}_u \rangle \rangle$ by the extensionality of the set-theoretic entities which serve as the elements of the universe U_u. As a result, Pollard manages to fix the canonical structure which stands for all isomorphically configured structures or utterance tokens. In order to have a name for them, I will henceforth call them *canonical representatives*. The collection of all canonical representatives of a grammar is the prediction of a grammar.

As in the investigation of AFS, I will focus on one prediction of $\langle \Sigma_2, \theta_2 \rangle$, the prediction that the utterance *Uther* will be judged grammatical. Although the structures of MI are defined quite differently from the set of relational abstract feature structures admitted by it, we will see immediately that AFS and MI share closely related problematic aspects.

Assume that we apply Pollard's method of constructing the canonical universes of Σ_2 interpretations as equivalence classes of Σ_2 terms. (14) shows schematically which canonical representatives Pollard's construction yields for the *Uther* configuration when it is applied to our exhaustive model. The subscripts indicate which entity of the exhaustive model of Fig. 12.3 is turned into the root entity of each of the six canonical representatives. By construction, each of the canonical representatives in (14) is a different set-theoretic entity. In brackets I mention the species of each root entity.

(14) a. $\langle u_{15}, \langle \mathsf{U}_{15}, \mathsf{S}_{15}, \mathsf{A}_{15}, \mathsf{R}_{15} \rangle \rangle$ (*u_word*)
b. $\langle u_{16}, \langle \mathsf{U}_{16}, \mathsf{S}_{16}, \mathsf{A}_{16}, \mathsf{R}_{16} \rangle \rangle$ (*nelist*)
c. $\langle u_{17}, \langle \mathsf{U}_{17}, \mathsf{S}_{17}, \mathsf{A}_{17}, \mathsf{R}_{17} \rangle \rangle$ (*elist*)
d. $\langle u_{18}, \langle \mathsf{U}_{18}, \mathsf{S}_{18}, \mathsf{A}_{18}, \mathsf{R}_{18} \rangle \rangle$ (*uther*)
e. $\langle u_{19}, \langle \mathsf{U}_{19}, \mathsf{S}_{19}, \mathsf{A}_{19}, \mathsf{R}_{19} \rangle \rangle$ (*cat*)
f. $\langle u_{20}, \langle \mathsf{U}_{20}, \mathsf{S}_{20}, \mathsf{A}_{20}, \mathsf{R}_{20} \rangle \rangle$ (*noun*)

It is immediately obvious that we observe here the same effect which we saw before with Pollard and Sag's utterance types. Since the *Uther* configuration contains six entities there are six distinct canonical representatives for it, although I assume that they would constitute one single prediction in Pollard's sense. The intended prediction seems to be that utterance tokens isomorphic to the *Uther* configuration are grammatical. In fact, for each n with $15 \leq n \leq 20$, all $\langle \mathsf{U}_n, \mathsf{S}_n, \mathsf{A}_n, \mathsf{R}_n \rangle$ in (14) are isomorphic, but this is not relevant in the construction. MI distinguishes between the corresponding entities in the universes because they are made of different equivalence classes of terms. Intuitively, the problem is that the entities are in different locations relative to their root entity, which entails that they are in a different equivalence class of terms defined on the root entity.[6]

I conclude that Pollard's construction fails to behave in the intended way. Pollard suggests that an HPSG grammar should be interpreted as specifying a set of canonical representatives such that no two members of the set are isomorphic, and utterance tokens of the language which are judged grammatical are isomorphic to one of the canonical representatives. Even if one is prepared to share Pollard's view of the goal of linguistics as a scientific theory, the particular construction proposed in (Pollard, 1999) is not suited to realize this conception without serious problems. For normal form grammars it introduces exactly the multiplicity of canonical representatives which it was designed to eliminate.

To sum up the preceding discussion, AFS and MI clearly fall short of the goals their proponents set for themselves. Neither Pollard and Sag's set of structures corresponding to linguistic utterance types nor Pollard's set of canonical representatives isomorphic to grammatical utterance tokens meets the intentions of their respective authors.

12.5 Minimal Exhaustive Models

I will now present an extension of King's theory of exhaustive models which avoids his problematic ontological commitment to possible utterance tokens, while retaining all other aspects of his model theory. At the same time, I also avoid the commitments to the ontological reality of utterance types or to the mathematical nature of the grammar models, which are characteristic of the metatheories (1) and (2). My starting point are the structural assumptions of normal form HPSG grammars, which I take to be independently motivated by the arguments in Section 12.3. For normal form grammars I define unique models which contain exactly one structure which is isomorphic to each utterance of a language considered well-formed by an ideal speaker of the language. This is, of course, what (1) and (2) essentially wanted to do, except

[4] The notation is explained in some detail in Section 12.5.

[5] They can be found in (Pollard, 1999, pp. 294–295) and even more explicitly in (Richter, 2004, pp. 208–210).

[6] It should be pointed out that the six interpretations in (14) are only isomorphic because we assume normal form grammars with an attribute EMBEDDED. However, without the EMBEDDED attribute we would run into the problems discussed in Section 12.3. In particular we would have stranded monster structures, and they would occur as canonical representatives which should correspond to possible utterance tokens, contrary to fact.

that I define *minimal exhaustive models* in such a way that I am not forced to make any commitments to the ontological nature of the structures in them. Given the philosophical intricacies of such commitments, I take this to be a highly desirable property of my proposal.

The goal is to characterize the meaning of grammars in terms of a set of structures, $\mathcal{M}$, which should have at least the following three properties: Each structure in $\mathcal{M}$ should have empirical consequences, i.e., there must be empirical facts which can falsify the predictions embodied by the structure; there should not be isomorphic copies of any empirically significant structure in the set of structures $\mathcal{M}$ assigned to each grammar; and finally, in accordance with one of Pollard's criteria, actual utterance tokens which are judged grammatical must be isomorphic to precisely one element in $\mathcal{M}$.

At first this small collection of desirable properties of $\mathcal{M}$ might seem arbitrary, even if every one of them can be individually justified. However, there is a way of integrating them with King's well-motivated theory of exhaustive models.

King's theory of grammatical truth conceives of language as a system of possible linguistic tokens. It claims that the system of possible tokens can be described as an exhaustive model of a grammar. The controversial aspect of this theory concerns the idea that language is a system of possible (i.e., actual and non-actual) tokens. Assume that we give up this aspect of King's theory. Instead we take an agnostic view toward language and say that we do not really know what it consists of. In our grammars we only make predictions about the discernible shapes of the empirical manifestations of language. We can operationalize this conception as follows: We want to write grammars such that whenever we encounter an actual utterance token, it will be judged grammatical if and only if there is an isomorphically structured connected configuration in an exhaustive model of the grammar. The connected configurations of interest will turn out to be the familiar connected configurations under unembedded signs. The choice of exhaustive model will not matter, since we are only concerned with the shape of the configurations, and we know that all shapes are present in any exhaustive model (by definition). However, since we are no longer after a system of possible tokens with an unknown number of isomorphic copies of configurations, we can be more precise about our choice of exhaustive model. It suffices to choose one which contains just one copy of each relevant connected configuration.

The theory of meaning we obtain from these considerations is a weakened form of King's theory. King says that a grammar is true of a natural language only if the language can be construed as a system of possible tokens, and the system of possible tokens forms an exhaustive model of the grammar. The theory proposed here as an alternative refrains from making such strong claims about the nature of language. It says that a grammar is true of a natural language only if each actual utterance token which is judged grammatical by an ideal speaker of the language is isomorphic to a maximal connected configuration in a minimal exhaustive model. The definitions of *maximal connected configurations* and *minimal exhaustive models* will be supplied directly below. Note that this condition endorses all arguments which King adduced to motivate exhaustive models, except for the ontological claim that the intended model is a system of possible (actual and non-actual) tokens.

Connected configurations in interpretations have been a leading intuitive concept since the first examples above. Their definition is straightforward. It presupposes the familiar RSRL signatures with a sort hierarchy $\langle \mathcal{G}, \sqsubseteq \rangle$, a distinguished set of maximally specific sorts $\mathcal{S}$, a set of attributes $\mathcal{A}$, an appropriateness function $\mathcal{F}$, and a set of relation symbols $\mathcal{R}$ whose arity is determined by a function $\mathcal{AR}$. Interpretations consist of a universe of objects U, a sort assignment function S which associates a symbol from $\mathcal{S}$ with each object in U, an attribute interpretation function A which treats each attribute symbol as the name of a partial function from U to U, and a relation interpretation function R which interprets each relation symbol as a set of tuples of the appropriate arity. $\mathsf{Co}_{\mathsf{I}}^{u}$ is the set of those objects in U which can be reached from u by following a (possibly empty) sequence of attributes.

Definition 12.5.1. *For each signature* $\Sigma = \langle \mathcal{G}, \sqsubseteq, \mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{R}, \mathcal{AR} \rangle$*, for each* Σ *interpretation* $\mathsf{I} = \langle \mathsf{U}, \mathsf{S}, \mathsf{A}, \mathsf{R} \rangle$*,*
$\langle \mathsf{U}', \mathsf{S}', \mathsf{A}', \mathsf{R}' \rangle$ *is a* connected configuration in I *iff*

1. $\mathsf{U}' \subseteq \mathsf{U}$,
2. *for some* $u' \in \mathsf{U}'$, $\mathsf{Co}_{\mathsf{I}}^{u'} = \mathsf{U}'$,
3. $\mathsf{S}' = \mathsf{S} \cap (\mathsf{U}' \times \mathcal{S})$,
4. $\mathsf{A}' = \mathsf{A} \cap (\mathcal{A} \times \{\mathsf{U}' \times \mathsf{U}'\})$,
5. $\mathsf{R}' = \mathsf{R} \cap \left(\mathcal{R} \times \mathit{Pow}\left(\bigcup_{n \in \mathbf{N}} (\overline{\mathsf{U}'})^n \right) \right)$.

Certain connected configurations in interpretations are of special interest to us. These are connected configurations which are not properly contained within other connected configurations in their interpretation. I will call them *maximal*:

Definition 12.5.2. *For each signature* Σ*, for each* Σ *interpretation* $\mathsf{I} = \langle \mathsf{U}, \mathsf{S}, \mathsf{A}, \mathsf{R} \rangle$*,*
$\langle \mathsf{U}', \mathsf{S}', \mathsf{A}', \mathsf{R}' \rangle$ *is a* maximal connected configuration in I *iff*
$\langle \mathsf{U}', \mathsf{S}', \mathsf{A}', \mathsf{R}' \rangle$ *is a connected configuration in* I*,*
and for some $u' \in \mathsf{U}'$*:*
$\mathsf{Co}_{\mathsf{I}}^{u'} = \mathsf{U}'$*, and for every* $u'' \in \mathsf{U}$, $\mathsf{Co}_{\mathsf{I}}^{u'} \not\subset \mathsf{Co}_{\mathsf{I}}^{u''}$.

There are three maximal connected configurations in the interpretation of Fig. 12.1. Their topmost elements are the *phrase* entity 16, which is the topmost entity in the connected configuration with the phonology *Uther walks*; the *word* entity 30, which is the topmost entity in the connected configuration with the phonology *Uther*; and the *word* entity 19, which is the topmost entity in the connected configuration with the phonology *walks*.

We can prove important properties of maximal connected configurations in models of normal form grammars: No two of them overlap. Each of them contains exactly one *u_sign* entity, which guarantees that they are empirical structures. Each entity in a model actually belongs to a maximal connected configuration, which ensures the empiricity of all entities. Every *u_sign* entity is contained in a maximal connected configuration, which guarantees that maximal connected configurations indeed capture all empirically relevant predictions without missing any. From now on I refer to maximal connected configurations in models of normal form grammars as *u-sign configurations*. The u-sign configurations in models of our grammars constitute the empirical predictions of the grammars.

I define *minimal exhaustive grammar models* as exhaustive models which contain exactly one copy of each possible u-sign configuration.

Definition 12.5.3. *For each signature* Σ*, for each* Σ*-theory* θ*, for each exhaustive* $\langle\Sigma,\theta\rangle$ *model* $\mathfrak{I}$*,*

$\mathfrak{I}$ *is a* minimal exhaustive $\langle\Sigma,\theta\rangle$ model *iff*

for each maximal connected configuration $\mathfrak{I}_1$ *in* $\mathfrak{I}$*, for each maximal connected configuration* $\mathfrak{I}_2$ *in* $\mathfrak{I}$*:*

if $\mathfrak{I}_1$ *and* $\mathfrak{I}_2$ *are isomorphic then* $\mathfrak{I}_1 = \mathfrak{I}_2$.

The exhaustive $\langle\Sigma_2,\theta_2\rangle$ model of Fig. 12.3 is an example of a minimal exhaustive grammar model. It contains exactly one copy of each u-sign configuration predicted by the grammar $\langle\Sigma_2,\theta_2\rangle$.

According to the properties of u-sign configurations, a minimal exhaustive model of a normal form grammar is partitioned into separate u-sign configurations. Each pair of u-sign configurations in it is structurally distinct and thus constitutes a different prediction of the grammar. Since all connected configurations in these models are u-sign configurations, they do not contain anything which is empirically vacuous.

With my construction I have not made any ontological commitments. I have claimed that the internal structure of actual utterance tokens can be discovered, and that this structure is mirrored precisely in u-sign configurations in minimal exhaustive grammar models. This did not presuppose saying anything about the ontology of linguistic objects. It was not even necessary to say what kinds of entities populate the minimal exhaustive models.

12.6 Concluding Remarks

Should there be any concern about the undetermined nature of the entities in minimal exhaustive models, or a preference for mathematical models, it is possible to pick out one mathematical model and fix it as the intended minimal exhaustive model of a given normal form grammar. The architecture of minimal exhaustive models of normal form grammars suggests strongly how to do this. Since the minimal exhaustive models are populated by a collection of u-sign configurations, and since the unique *u_sign* entity in each u-sign configuration contains all other elements of the configuration as its components, it is quite natural to define the entities in the u-sign configurations as equivalence classes of paths which lead to them from their individual *u_sign*. This of course is essentially Pollard's construction of canonical representatives, except that I avoid the multiplicity of representatives for one and the same prediction because my mathematical idealizations do not consist of pairs of entities and configurations. Instead, I exploit the special properties of the models of normal form grammars and am thus able to make do with bare u-sign configurations.

But although the construction of minimal exhaustive models from mathematical entities is simple, I am not aware of any convincing argument for them. In my opinion, DEFINITION 12.5.3 completes the explanation of the meaning of normal form HPSG grammars.

Bibliography

King, Paul J. (1999). Towards Truth in Head-driven Phrase Structure Grammar. In Valia Kordoni, ed., *Tübingen Studies in Head-Driven Phrase Structure Grammar*, Arbeitspapiere des SFB 340, Nr. 132, Volume 2, pp. 301–352. Eberhard Karls Universität Tübingen.

Pollard, Carl and Ivan A. Sag (1994). *Head-Driven Phrase Structure Grammar*. University of Chicago Press.

Pollard, Carl J. (1999). Strong generative capacity in HPSG. In Gert Webelhuth, Jean-Pierre Koenig, and Andreas Kathol, eds., *Lexical and Constructional Aspects of Linguistic Explanation*, pp. 281–297. CSLI Publications.

Richter, Frank (2004). *A Mathematical Formalism for Linguistic Theories with an Application in Head-Driven Phrase Structure Grammar*. Phil. dissertation (2000), Eberhard Karls Universität Tübingen.

Chapter 13

Coalgebras, Trees, and Grammars

Lawrence S. Moss
Department of Mathematics
Indiana University
Bloomington, IN 47405 USA
lsm@cs.indiana.edu

13.1 Introduction

This paper is a contribution to discussions of the mathematical background of MTS, including formalizations of all the basic notions in the subject. The main point is that work in coalgebra in the past five years has developed a body of results which already has been applied to automata theory and formal language theory and which also might be fruitfully applied to MTS. This paper is a note primarily addressed to people interested in the formal underpinnings of MTS, and also in implementations in functional programming languages. So it consists of summaries of several papers, and discussions of ideas at a fairly high level of abstraction. It is more of a summary of my talk at the workshop and a plan of future action, and less of a paper with new results.

There are at least two reasons why someone interested in MTS would be interested in this project. First, MTS works with mathematical tools such as higher-dimensional trees which were coined especially for use in the subject. That is, the tools were developed *ad hoc*. I do not say this as a criticism: many times things done ad hoc are exactly the ones needed. At the same time, one may wonder where the tools come from; that is, how they relate mathematically to other notions. My sense is that coalgebra and related fields will be a help here, because they are dealing with subjects at a level of abstraction which is high enough to show interesting and unexpected relationships, and yet not so high that the results apply to absolutely everything.

The other reason is perhaps more prosaic, but for some it will perhaps be more pressing. If one wants to write programs to work with representations coming from MTS, then it will be useful to have presentations which are much "lighter" and more abstract than the original presentations due to Rogers (see (Rogers, 2003) and other papers). Similarly, if one wants a smooth generalization of the ideas of MTS to probabilistic settings (to name just one example), then it would be good to see a more general development of the overall ideas.

13.1.1 Sources

The sources of this talk are (a) the coalgebraic treatment of automata initiated by (Rutten, 1999) and then elaborated in work such as (Jacobs, 2005); (b) coalgebraic automata theory, developed extensively by Venema and colleagues: see (Kupke and Venema, 2005); (c) coalgebraic recursion theory, especially treatments of second-order substitution and recursive program schemes (Milius and Moss, 2006); and (d) very recent work (Ghani and Kurz, 2007) that aims at algebraic and coalgebraic renderings of multidimensional trees.

algebra	coalgebra
initial algebra	final coalgebra
least fixed point	greatest fixed point
congruence relation	bisimulation equivalence
recursion: map out of an initial algebra	corecursion: map into a final coalgebra
equational logic	modal logic
construct	observe
bottom-up	top-down
useful in syntax	useful in semantics

Figure 13.1: The conceptual comparison

13.2 General conceptual comparison

Figure 13.1 is part of a broader conceptual comparison between *algebraic* and *coalgebraic* ideas in several guises. (It comes from (Moss, 2007), a source aiming more towards set theory.) The entries towards the top of the chart are actual mathematical dualities. The duality of equational logic and modal logic is more of a programmatic goal, and here there is a body of work which substantiates the duality claim. And the rows at the bottom are the most informal of all. However, they highlight the interest of this kind of comparison for mathematical linguistics in general and MTS in particular. Actually, the interest for most people will come in the following way: despite the greater familiarity of most people with the left-hand column in the chart, the right-hand one is at least as important for mathematical linguistics. The point

will be made in a stronger way below when we see the connection of formal languages with modal logic.

13.3 Coalgebra

Many of the definitions in the rest of this note lean on the basics of category theory, especially those pertaining to *algebras*, *coalgebras*, *monads*, and *completely iterative algebras*. As with all the work here, this note will be much too succinct to convey everything that is needed to read the papers.

Let F be an endofunctor on a category C. A *coalgebra for F* is a pair (c, f), where c is an object of C, and $f : c \to Fc$ is a morphism called the *structure* (map).

For example, every graph "is" a coalgebra of $\mathcal{P}$ on Set. What this means more concretely is that there is a correspondence between graphs in the form $(G, \to)$, that is a set with a relation on it, and coalgebras $(G, e : G \to \mathcal{P}(G))$ given by taking $\to$ to the function $x \mapsto \{y : x \to y\}$. In words, we trade in the edge relation for the function assigning to each node in the graph its set of children.

Let (c, f) and (d, g) be coalgebras for the same functor. A *morphism of coalgebras from (c, f) to (d, g)* is a morphism $\alpha : c \to d$ in the category C so that the diagram below commutes:

$$\begin{array}{ccc} c & \xrightarrow{f} & Fc \\ {\scriptstyle\alpha}\downarrow & & \downarrow{\scriptstyle F\alpha} \\ d & \xrightarrow[g]{} & Fd \end{array}$$

A coalgebra (c, f) is a *final* (or *terminal*) coalgebra if for every coalgebra (d, g), there is a *unique* morphism of coalgebras $\alpha : (d, g) \to (c, f)$.

Although we have mentioned coalgebras first, one could just as well mention algebras. These are morphisms the other way, from Fa to a. Morphisms between algebras are morphism in the category between the carriers which again make a diagram commute. As our chart above suggests, some of the main algebras of interest in the area will be *initial algebras* of functors; these are the ones which have the property that every algebra is the target of a unique morphism. Initial algebras usually have what it takes to define functions by recursion *out of them*, and in the examples of interest they look like *term algebras*.

Let Σ be a signature, a set together of *function symbols* with an assignment of natural number *arities* to the elements of Σ. Σ gives rise to an endofunctor H_Σ on sets, taking a set A to the set of height-one Σ-trees with leaves labeled in A. Algebras for H_Σ are like operation tables; they are the natural semantic models that one would use in subjects like universal algebra. For each set X, we also have a functor $H_\Sigma(\_) + X$ which adds the elements of X as fresh constants. The collection $T^0_\Sigma X$ of finite Σ-trees with variables from X is an initial algebra of the functor $H_\Sigma(\_) + X$. The collection $T_\Sigma X$ of finite *and infinite* Σ-trees with variables from X is a final coalgebra of the functor $H_\Sigma(\_) + X$. In the algebra result, the structure is tree tupling, and in the coalgebra result it is the inverse.

A *monad* on a category C is a triple (T, μ, η) consisting of a functor $T : C \to C$, and natural transformations $\mu : TT \to T$, and $\eta : Id \to T$, satisfying the *unit laws* $\mu \circ T\eta = \mu \circ \eta T = id$, and the *associative law* $\mu \circ T\mu = \mu \circ \mu T$:

$$\begin{array}{ccccc} T & \xrightarrow{T\eta} & TT & \xleftarrow{\eta T} & T \\ & \searrow & \downarrow{\scriptstyle\mu} & \swarrow & \\ & & T & & \end{array} \qquad \begin{array}{ccc} TTT & \xrightarrow{T\mu} & TT \\ {\scriptstyle\mu T}\downarrow & & \downarrow{\scriptstyle\mu} \\ TT & \xrightarrow[\mu]{} & T \end{array}$$

The notion of a monad is a very short and elegant presentation of *syntax*; I realize that this is not at all apparent from the definition, and it does take some extra observations to make it precise. The reason is that all of the basic properties of *substitution* can be seen to follow from the monad properties (and conversely).

13.4 Automata from an Algebraic/Coalgebraic Perspective

Let A be a set of *input letters*, and B a set of output letters. Then a *deterministic automaton* with input A and output B is a set S of states together with functions $\delta : S \to S^A$ and $\varepsilon : S \to B$. Putting these functions together gives $\delta \times \varepsilon : S \to S^A \times B$, so we have a coalgebra for the functor $FS = S^A \times B$. It is also important that coalgebra morphisms for this functor are exactly the morphisms of automata that one would define from scratch, and also that the coalgebraic notion of bisimulation (which we did not spell out) again specializes to the natural notion of equivalence. The final coalgebra of F has as carrier the set B^{A^*}; the map of an automaton a into it takes a word w over A to the result in B of running w in a.

Incidentally, the notion of a *weighted automaton* also comes out as an example, and this topic also makes use of connections to monoids (related to the monads we already mentioned).

As with all our summaries in this note, what we have mentioned is the tip of the tip of the iceberg. One should see (Jacobs, 2005) for much more, including connections to regular expressions and bialgebras (the latter are structures which combine algebras and coalgebras, using a *distributive law*.

Connections to probability are not immediately relevant for this project, but perhaps in the future one would like to see such connections. Here we note that a finite state Markov process is essentially a coalgebra of the functor taking a set to the set of its discrete probability distributions.

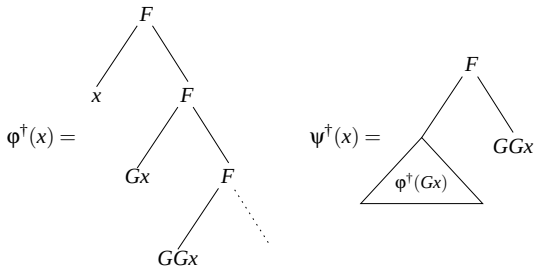
13.5 Recursive Program Schemes

Recursive program schemes are systems of equations used in defining functions or languages. The topic is at

the heart of the semantics of programming languages (or at least it once had been there) but it is not so prominent in mathematical linguistics. One takes a system of equations such as

$$\begin{array}{rcl} \varphi(x) & \approx & F(x,\varphi(Gx)) \\ \psi(x) & \approx & F(\varphi(Gx),GGx)) \end{array} \tag{13.1}$$

where F and G are *given* or *base* functions and where φ and ψ are defined in terms of them by (13.1). One then asks for an *uninterpreted solution* – this would be a pair of infinite trees – or an *interpreted solution* in some algebra for the endofunctor on sets associated to the two given symbols. In our case, the uninterpreted solution is



This solution can in general be obtained as follows: the given scheme expands to a system of equations with variables for each term t over $\Sigma+\Phi$. To each variable $\underline{t}$ the right-hand side of the system is given by replacing all appearances of symbols of Φ by their right-hand sides in the given scheme. For example from the rps above we have the equations

$$\begin{array}{rcl} \underline{x} & \approx & x \\ \underline{\varphi(Gx)} & \approx & F(\underline{Gx},\underline{\varphi(GGx)}) \\ \underline{\varphi(x)} & \approx & F(\underline{x},\underline{\varphi(Gx)}) \\ \underline{F(x,\varphi(Gx))} & \approx & F(\underline{x},\underline{F(Gx,\varphi(GGx))}) \\ \underline{\psi(x)} & \approx & F(\underline{\varphi(Gx)},\underline{GGx}) \\ \underline{GGx} & \approx & G(\underline{Gx}) \\ \underline{\varphi(\psi(x))} & \approx & \\ F(\underline{F(\varphi(Gx,GGx))},\underline{\varphi(G(F(\varphi(Gx,GGx))))}) & & \\ \vdots & & \end{array}$$

and so on. Notice that each term on the right-hand side is a term over Σ which is either just a syntactic variable or *flat*, i. e., one operation symbol from Σ with variables. The solution of φ is now just the tree unfolding of the variable $\underline{\varphi(x)}$ and similarly for ψ.

Now the notion of second-order substitution is exactly what one finds in *tree adjunction*, and this is our reason for bringing up the topic in this work.

It so happens that the entire theory of second-order substitution generalizes to categories with coproduct with the property that for each object a, the endofunctor $b \mapsto F(b)+a$ has a final coalgebra. This is a fairly mild condition, and essentially all functors of practical interest satisfy it. There is then an extensive theory of recursion for such endofunctors. Some of the high points are the theory of first-order and second-order substitution, (see (Aczel et al., 2003)), the theory of recursive program schemes, and the theory of *completely iterative algebras*. All of these have some relevance to formal language theory primarily because the way that grammars define languages may be modeled as solutions to *interpreted* recursive program schemes.

Although it is not of direct interest in this talk, the theory here also encompasses fractal sets of real numbers and also recursive functions as usually defined in mathematics. It is the only approach I know of with this kind of scope.

13.6 The Algebraic Treatment of Higher Dimensional Trees

We have saved for last the only source so far which has directly looked at Rogers' work from a point of view related to the one we advocate, (Ghani and Kurz, 2007). Our purpose up until now is to *begin to* make it and papers to come more accessible to people in the area. And as usual, we only state the opening points of their work. Consider the following sequences of functors on sets:

$$\begin{array}{rcl} R_1X & = & \emptyset \\ T_nX & = & 1+R_nX \\ R_{n+1}X & = & \mu Y.X\times T_nY \end{array}$$

Here $n \geq -1$. And in the last point what we have is the least fixed point of the operation taking a set Y to $X \times T_nY$ (this is an initial algebra construction).

What does one get from such definitions? It happens that $R_{n+1}X$ is the set of non-empty $n+1$-dimensional tree domains on X, and $T_{n+1}X$ is the set of empty or non-empty $n+1$-dimensional tree domains on X.

This observation is important because it gives a "domain-free" characterization of the higher dimensional tree domains. Further, it is connected to observations made in terms of monads (each R_n is one) and also *comonads* (formal duals of monads) that lead to a clean definition in Haskell of the higher-dimensional domains.

Furthermore, (Ghani and Kurz, 2007) go on to propose coalgebraic versions of higher-dimensional automata and language acceptance, generalizing what we have seen for classical treatment.

13.7 Future work

I have tried to make the case that enough groundwork is already present to get a mathematical treatment of the notions behind MTS which is abstract and elegant, partly because it expresses higher-dimensional notions in terms of the universal constructions that come from category theory; also the theory should be useful in the sense that it leads to generic algorithms and programs, and also that it would generalize to still wider settings. The main task now is to put the existing ingredients together.

Bibliography

P. Aczel, J. Adámek, S. Milius and J. Velebil, Infinite Trees and Completely Iterative Theories: A Coalgebraic View, *Theoret. Comput. Sci.*, 300 (2003), 1–45.

Ghani, Neil and Alexander Kurz. Higher Dimensional Trees, Algebraically. to appear in the proceedings of *CALCO'07*.

Jacobs, Bart, A Bialgebraic Review of Deterministic Automata, Regular Expressions and Languages. In: K. Futatsugi, et al (Eds.), Algebra, Meaning and Computation: Essays dedicated to Joseph A. Goguen on the Occasion of his 65th Birthday. Springer LNCS 4060, pp .375–404, 2006

Kupke, Clemens and Yde Venema. Closure Properties of Coalgebra Automata. Proceedings of the Twentieth IEEE Symposium on Logic in Computer Science (LICS 2005), 199-208.

Milius, Stefan and Lawrence S. Moss. The Category Theoretic Solution of Recursive Program Schemes. *Theoretical Computer Science*, Volume 366, Issues 1–2, November 2006, pp. 3–59.

Moss, Lawrence S. Set theory and circularity. to appear in the *Stanford Encyclopedia of Philosophy*.

Rogers, James. Syntactic structures as multidimensional trees. Research on Language and Computation 1 (3–4), 265–305, 2003.

Rutten, Jan. Automata, power series, and coinduction: taking input derivatives seriously. In J. Wiedermann et al (eds.), proceedings of ICALP 1999, Springer LNCS 1644, pp. 645–654, 1999.