The MSO Logic–Automaton Connection in Linguistics

Frank Morawietz and Tom Cornell

Universität Tübingen, Seminar für Sprachwissenschaft, Kleine Wilhelmstraße 113, 72074 Tübingen, Germany
Email: {frank,cornell}@sfs.nphil.uni-tuebingen.de

Abstract. In this paper we discuss possible applications of a system which uses automata-based theorem-proving techniques drawing on the decidability proof for weak monadic second-order (MSO) logic on trees to implement linguistic processing and theory verification. Despite a staggering complexity bound, the success of and the continuing work on these techniques in computer science promises a usable tool to test formalizations of grammars. The advantages are readily apparent. The direct use of a succinct and flexible description language together with an environment to test the formalizations with the resulting finite, deterministic tree automata offers a way of combining the needs of both formalization and processing. The aim of this paper is threefold. Firstly we show how to use this technique for the verification of separate modules of a Principles-and-Parameters (P&P) grammar and secondly for the approximation of an entire P&P theory. And thirdly, we extend the language of the MSO tree logic to overcome remaining engineering problems.

1 Introduction

There is continuing interest in computational linguistics in both model-theoretic syntax and finite-state techniques. The decidability proof for the (weak) monadic second-order logic (MSO) of two successor functions (WS2S) furnishes a technique for deriving (bottom-up) tree automata for recognizable relations from the MSO constraints effectively (Thatcher and Wright 1968, Doner 1970).[1] The connection between MSO constraints and tree automata is based on techniques for coding variable-assignment functions as labeled trees; then tree automata can be "compiled" from the source formula in such a way that they accept exactly the trees which represent satisfying assignments to the source formula. While these results have been known for some time, the suitability of this formalism as a description language for Principles-and-Parameters (P&P) based linguistic theories has only recently been shown by Rogers (To appear) in a large scale formalization of most of the ideas from Rizzi (1990).

In general, on an abstract level, we address two issues in this paper: firstly, how we can use this technique in (computational) linguistics and secondly how

[1] Generalizing a similar result by Büchi (1960) from S1S and finite state automata to WS2S and tree automata.

useful it is. Naturally the first issue is more theoretically and the second one is more practically oriented. The paper proceeds by sketching the technique we want to make use of by presenting a short introduction into its background and an example of an MSO constraint and the correspondingly generated tree automaton. The following sections deal with possible applications of the technique to natural language processing (NLP). In particular, we discuss the separate compilation and verification of modules of a P&P theory and how to approximate P&P grammars within MSO tree logic despite its limited generative capacity. We illustrate the usefulness of the technique with a section on our experiences with a software tool. We conclude by applying a constraint logic programming (CLP) extension—resulting from the embedding of the compiler as its constraint solver—to remaining engineering problems.

We assume some basic familiarity with both the decidability proof for WS2S (an introductory presentation is in Gurevich 1985) and tree automata (an introduction can be found in Gécseg and Steinby 1997).

2 From Logic to Automata

MSO logic is a straightforward extension of first-order logic to include variables that range over sets (i.e., monadic predicates) and quantifiers over these variables. Let $\mathcal{N}_n$ denote the structure of $1 \leq n \leq \omega$ successor functions. $\mathcal{N}_n$ is the infinite, uniformly n-branching tree with the ith successor function leading to the ith daughter of a node. By SnS we denote the MSO theory of n successor functions, and by WSnS we denote the *weak* MSO theory of n successor functions, meaning that set variables are constrained to range only over *finite* sets. We will overload the term WSnS in the sense that we use it also to denote the MSO language over (W)SnS. In this paper we will focus in particular on WS2S: the weak monadic second-order theory of $\mathcal{N}_2$, the structure of two successor functions.

A (maximally) binary branching tree is a rooted, dominance connected subset T of the domain of $\mathcal{N}_2$. A labeled tree is a tuple $\langle T, F_1, \ldots, F_k \rangle$ of subsets of this domain, with the sets F_i being the features labeling the nodes of tree T. A grammar is an axiomatic definition of a $k+1$-ary relation in $\mathcal{N}_2$ picking out all and only the well-formed labeled trees.

There is a close correspondence between formulas in WSnS and finite automata. More precisely, any relation definable in WS2S can also be defined by a tree automaton, which encodes the satisfying assignments to the formula in the labels on the nodes of the trees that it accepts. The coding is simple: assignment functions map first-order variables to nodes in $\mathcal{N}_2$ and second-order variables to sets of nodes in $\mathcal{N}_2$. The labeling functions on the trees the automaton accepts do the converse: they map nodes to the set of variables to which that node is assigned. Concretely, we can think of nodes as being labeled with bit-strings such that bit i is on at node n if and only if n is to be assigned to variable X_i. Because all sets are required to be finite, we can restrict our attention to automata over finite trees. This makes the development of software tools possible, since an efficient minimization algorithm exists for automata over finite sets.[2]

The proof of the logic-automaton correspondence is by induction. In the base step one need only exhibit automata recognizing the atomic formulas of the logic, given the coding of assignment functions sketched above. In the inductive step one need only show that that operators in the logic correspond to constructions on automata: conjunction corresponds to a cross-product construction on automata, negation to the inversion of final states of the automaton and existential quantification to a projection construction which converts an automaton over a k bit alphabet to a (nondeterministic) automaton over a $k-1$ bit alphabet. It follows that a sentence in the language of WSnS which is not in the theory of $\mathcal{N}_n$, having no satisfying assignments, will correspond to an automaton which accepts no trees. Since the emptiness problem for tree automata is decidable, so is WSnS. For the special case of S1S this was shown in Büchi (1960), for WSnS in Thatcher and Wright (1968) and independently in Doner (1970) and finally for SnS in Rabin (1969).

There are, however, some very daunting engineering problems which have prevented the implementation of these results in practical theorem provers. In particular, the negation construction on automata only works for deterministic automata, and the projection construction, which implements quantification, outputs nondeterministic automata. It is always possible to determinize a (bottom-up) tree automaton, using essentially the same subset construction as is used for finite-state automata. Using the subset construction can in the worst case exponentially increase the number of states in the input automaton. Since the subset construction must be used every time a negation has an existential quantifier in its scope, the number of states can grow according to a function with a stack of exponents whose height is determined by the number of $\forall-\exists$ quantifier alternations. In spite of this, some very encouraging recent results from the areas of computer hardware and system verification (e.g., Kelb et al. 1997; Basin and Klarlund (To Appear); Møller and Klarlund 1998) suggest that these techniques are in fact efficient; it is the problem space expressible in MSO tree logics that is hard.

Since the decidability proof for WS2S is inductive on the structure of MSO formulas, we can choose our particular tree description language rather freely, knowing that the resulting logic will be decidable and that the translation to automata will go through as long as the atomic formulas of the language represent relations which can be translated (by hand if necessary) to tree automata.[3] For example, Niehren and Podelski (1992) and Ayari et al. (1998) have investigated the usefulness of these techniques in dealing with feature trees which unfold feature structures; there the attributes of an attribute-value term are translated to distinct successor functions. On the other hand, Rogers (To appear) has de-

[2] There exists a class of automata working on infinite sets. But for these so called Büchi or Rabin automata there is no efficient minimization routine, which prevents a reasonable implementation of a tool for (strong) SnS.

[3] It requires further proof that such a language has the full power of WS2S, though.

veloped a language with long-distance relations (dominance and precedence) which is more appropriate for work in Government-Binding (GB) theory. One can imagine other possibilities as well: the automaton for Kayne-style asymmetric, precedence-restricted c-command (Kayne 1994) is very compact, and makes a suitable primitive for a description language along the lines suggested by Frank and Vijay-Shanker (1998).

We will illustrate these techniques with a simple example. We define a property Path which holds of a dominance-connected set P just in case for every pair of nodes in P one dominates the other.[4] For this and all other formulas in the paper, we use the language of Rogers (To appear), which is interdefinable with (W)S2S. Notational conventions will be the following: Uppercase letters denote second-order variables, lowercase letters denote first-order ones. The binary relation symbol $\triangleleft$ denotes parenthood, i.e., immediate dominance, $\triangleleft^*$ and $\triangleleft^+$ denote reflexive and proper domination, respectively, and $\prec$ denotes the precedence relation. Sans-serif labels stand for defined predicates (with their arguments). We include comments in the formulas for better understanding.

(1)
$$\mathsf{Connected}(P) \overset{def}{\iff}$$
% If an element falls between any two other elements
% wrt domination it has to be in P.
$$(\forall x, y, z)[(x \in P \wedge y \in P \wedge x \triangleleft^* z \wedge z \triangleleft^* y) \Rightarrow z \in P]$$

(2)
$$\mathsf{Path}(P) \overset{def}{\iff}$$
% P is connected and linearly ordered.
$$\mathsf{Connected}(P) \wedge (\forall x, y \in P)[x \triangleleft^* y \vee y \triangleleft^* x]$$

The tree automaton corresponding to the formula for Path in (2) resulting from the compilation procedure is shown in Fig. 1. For readability, we denote the alphabet of node labels as tuples indicating for each free variable whether a node is assigned to it or not. In this case we have only one free variable, P, so the alphabet just consists of the tuples $\langle P \rangle$ and $\langle \neg P \rangle$.

The initial state is a_0. We remain in the initial state until we encounter a node labeled for membership in P, at which point we transit to state a_1. As long as we are in a_1 on exactly one daughter and the mother node is also assigned to P, we remain in a_1. Note that if we find ourselves in a_1 on *both* daughters, then we enter a_3, the sink state. If we are in a_1 on one daughter and the mother is not in P, then we enter state a_2, indicating that we have completed our search for the path P. Any further P-nodes force us into state a_3: this tree cannot represent an assignment function satisfying the constraints on Path-hood. All of

[4] Observe that this is a property of sets, and so, apparently, a third-order variable. However, we can freely use any relations which are over sets or non-monadic provided that we explicitly define them in MSO—as we do here in (1) and (2)—such that they are reducible via syntactic substitution to an MSO signature, i.e., $r(\boldsymbol{X}) \overset{def}{\iff} \phi(\boldsymbol{X})$. We cannot quantify over them, of course.

$$\begin{aligned} \mathfrak{A} &= \langle A, \Sigma, a_0, F, \alpha \rangle \\ A &= \{a_0, a_1, a_2, a_3\} \\ \Sigma &= \{\langle \neg P \rangle, \langle P \rangle\} \\ F &= \{a_0, a_1, a_2\}, \end{aligned}$$

$$\begin{array}{ll} \alpha(a_0, a_0, \langle \neg P \rangle) = a_0 & \alpha(a_0, a_0, \langle \ \ P \rangle) = a_1 \\ \alpha(a_0, a_1, \langle \neg P \rangle) = a_2 & \alpha(a_0, a_1, \langle \ \ P \rangle) = a_1 \\ \alpha(a_1, a_0, \langle \neg P \rangle) = a_2 & \alpha(a_1, a_0, \langle \ \ P \rangle) = a_1 \\ \alpha(a_0, a_2, \langle \neg P \rangle) = a_2 & \alpha(a_2, a_0, \langle \neg P \rangle) = a_2 \\ \multicolumn{2}{c}{\text{all other transitions are to } a_3} \end{array}$$

Fig. 1. The tree automaton for $\mathsf{Path}(P)$

states a_0, a_1 and a_2 are final states: P may be the empty set (a_0), it may include the root (a_1) or not (a_2).

Obviously, there are many other ways one can define Path. However, any other equivalent definition would still yield an automaton isomorphic to the one in Fig. 1. That is, minimized automata are normal forms. Since they allow for the direct and efficient recovery of particular solutions, they make appropriate solved forms for use in constraint programming, a point we will take up and develop in Sect. 6.

There are two perspectives one can take on the equivalence of MSO logic to tree automata as ways of defining sets of syntax trees. First, one can think of the decidability proof as suggesting a way of doing (MSO) theorem proving. This is the way things are done in computer science applications such as hardware and system verification. However, one can also think of things the other way around, and take MSO logic as a specification language for automata. Tree automata have the advantages of finite-state machines presented in Roche and Schabes (1997): They provide a compact and uniform framework, i.e., they are space and time efficient and suitable for many aspects of NLP such as morphology, lexicon, syntax and input descriptions and they provide decidable tests for a number of important questions.[5] So we can make use of the logic-automaton correspondence either as a way of proving theorems using automata or as a way of manufacturing automata to MSO specifications.

Furthermore, we have a descriptive complexity result for MSO logic on trees. Since tree automata accept *recognizable* sets of trees, which are sets of trees which yield (all and only) the context-free languages, we have that we can only define in MSO logic principle-based grammars which are weakly context-free. The extensive formalizations of GB theories in MSO logic presented in Rogers represent the first descriptive complexity results of this kind for principle-based

[5] For example: $w \in \mathcal{L}$, $\mathcal{L} = \emptyset$, $\mathcal{L} = \Sigma^*$, $\mathcal{L}_1 \subset \mathcal{L}_2$, $\mathcal{L}_1 = \mathcal{L}_2$.

grammars, which had been thought to have considerably more expressive power and to be in a format for which any kind of descriptive complexity analysis would be impossible.

Consider, from these perspectives, the parsing problem and the recognition problem for a principle-based grammar. If the grammar (a collection of "principles" and "parameters") can be formalized in MSO logic, then it is logically possible to compile it into a single (huge) tree automaton, which accepts all and only the well-formed parse trees strongly generated by the grammar. In this setting, the parsing problem is—as standardly for satisfiability based grammars—the conjunction of a formula characterizing the input (as the set of all trees yielding the input string proper) with a grammar formula. In automata theoretic terms, we are considering two tree automata, and the parsing problem reduces to the problem of forming the cross-product automaton recognizing the intersection of the two input tree sets, i.e., the parse forest for the input. The recognition problem just reduces to the problem of determining the emptiness of this intersection. These are both decidable, in fact polynomial problems. Technically they are polynomial in the size of the input, since the grammar is a constant. Practically speaking, however, this "grammar constant" can be overwhelming, since the compilation from a formula to an automaton can be so explosive. Note that using minimal automata offers the easiest way of deciding emptiness. The language recognized by a minimal automaton is empty iff it has only one state which is initial but not final.

This problem of explosiveness is a problem in practice, but there is also a problem in principle, namely that the recognizable forests only yield context free string languages. It is generally believed that at least some languages (e.g., Swiss German) are not even weakly context-free, and the theoretical analysis in the P&P framework of some constructions like complex verb cluster formation in Dutch definitely define unrecognizable forests of syntax trees. In the remainder of this paper we consider three ways to deal with these limitations in practice and in principle.

First one can entertain more modest goals than the compilation of a complete grammar: there certainly are many grammatical properties besides the property of global well-formedness that can be fully formalized in MSO logic, and if such properties can be compiled on the computers of the present day then we have at once computationally efficient, automata-based implementations of those properties and also a means to perform extensive verification and debugging of the underlying theory of that module.

The second approach actually follows from the first. If one looks for modules of the grammar that drive it beyond the context free boundary, it appears that there may in fact be a single culprit. Rogers shows that the notion of Free Indexing, that is, that constituents can be assigned numerical indices freely, with the distribution later constrained by the grammar, is inexpressible in MSO logic.[6]

[6] More formally, he shows that the addition of co-indexation to the logic makes it undecidable and, since SnS is decidable, there cannot possibly be any definition of co-indexation in SnS.

Simplifying a good deal, indices are features, and features must be implemented as free set variables; obviously, there can only be a bounded number of variables appearing in a formula. From the automaton perspective, indices must be encoded in the alphabet, and the alphabet must be fixed before an automaton can be generated, so an automaton can be defined only given a bounded number of indices. We address the problem of working with a bounded number of indices in Sect. 4.

The third approach is based on the observation that minimal automata are normal forms for formulas and that concrete solutions, i.e., acceptable trees, can be extracted very efficiently. Therefore minimal automata make appropriate solved forms for a constraint solver. In Sect. 6 we consider embedding an MSO-compiler in a CLP scheme. This extends the power of the formalism (indeed, we lose decidability), and also adds a degree of control to the compilation process by using constraint logic programs.

3 Compiling Grammatical Principles to Tree Automata

In this section, we consider our most modest proposal, namely to do automaton construction and theory verification on subparts of the grammar. The question of which grammatical properties can be compiled in a reasonable time is largely empirical. The high-quality software tool MONA for the compilation of MSO formulas into both FSAs and tree automata has been made available recently (Klarlund and Møller 1998, URL: `http://www.brics.dk/~mona/`), so we can now investigate this question.[7]

As an example of the application of this tool, we compiled an automaton from the X-Bar Theory presented in Rogers monograph. Note that this is a non-trivial part of the overall P&P theory formalized there.

In many variants of P&P theories this module of the grammar is essentially a set of simple phrase structure rules defining possible d-structures, over which transformations will later operate. Rogers' formalization is of a monostratal theory, however, in which well-formedness conditions on chains take the place of movement transformations. As a result, the X-Bar theory Rogers presents is far from trivial, since it must describe the kinds of structures that arise after transformations have applied. As a result it amounts to some five pages of MSO formulas which serve as input to MONA, and we cannot hope to present it in detail here.

Additional complications arise in the MSO specification of X-Bar theory because, in the underlying grammatical theory, there are three layers of structure: nodes of the tree are "segments" which are grouped into "categories" (here referring to a purely structural notion having little or nothing to do with morphosyntactic categories like noun, verb, preposition, etc.). Then categories are assigned "bar-level" features. So bar-levels, being features, are sets of categories, which

[7] An older version of MONA (Henriksen et al. 1995), restricted to strings and finite state automata, has been in use for some time in hardware and system verification, see references above.

in turn are sets of segments. However, it is easy to avoid this third-order notion by associating features with the segments in such a way as to assure that every segment of a category has features identical to those which, in the grammatical theory, are to be assigned to the category. Furthermore we have to keep track of which nodes are *trace* positions in their chains, which nodes by contrast are still in their *base* positions, and which phrases are *adjuncts*, i.e., attached to a segment of a non-trivial category.

The main content of X-Bar theory is a set of conditions on the distribution of the bar-level features, and a principle which states that every category has a "head", another category immediately below it with which it shares essential features like grammatical category (noun, verb, etc.). A node is allowed to be labeled with *Bar0* if it has exactly one lexical child category whose features it inherits. It is labeled *Bar1* if its category immediately dominates a node (its head) which is labeled *Bar0* and all its other (non-head) children are labeled *Bar2*. Finally, a node is labeled *Bar2* if it is a trace (i.e., is assigned to the set $Trace$) or its category immediately dominates some corresponding category belonging to *Bar1* and all its non-head children are in *Bar2*. The X-Bar module encodes these three definitions and principles limiting the distribution of adjuncts (i.e., of the *Adj* feature) such that no node is marked as adjoined unless it is the immediate child of a non-minimal node of a non-trivial category and that we cannot adjoin to any arbitrary node, e.g., not to traces.

One sees immediately that such an X-Bar theory becomes quite complicated indeed, and its correctness correspondingly hard to verify by hand! In fact, we did discover a minor bug in Rogers' formalization. We attempted to prove the assertion that Rogers' X-Bar theory implies the disjointness of the three bar-levels, i.e., that no node can belong to more than one bar-level. That is, we coded the implication as an MSO formula, and attempted to prove it in MONA. The attempt failed, with MONA reporting a counterexample.[8] In particular, Rogers' X-Bar theory does not imply that the *Bar0*-feature is disjoint from the *Bar1* and *Bar2* features.

The problem arises because of the fact that the distribution of features is such that they can appear anywhere they are not explicitly forbidden. In our case this means that the constraints allow for example a unary branching subtree with two nodes; a node which is labeled with both *Bar0* and *Bar1* and whose (unique) daughter is at the same time lexical and *Bar0*. In such a tree all constraints are satisfied. The mother node fulfills the *Bar1* constraint since it has a *Bar0* daughter and no other children and the *Bar0* constraint since it has a unique daughter which is lexical. This second node is allowed since Rogers does not place a constraint on the sets denoting lexical elements. Positively expressed this means that they are allowed to appear on internal nodes of a parse tree. To avoid this we have to add conditions forcing the lexical variables to only

[8] Without reference to special properties of a particular software tool, in general if $P \Rightarrow Q$ is unsatisfiable then any MSO logic-to-automaton compiler will construct the empty automaton. In that case $P \wedge \neg Q$ will be satisfiable and the resulting automaton can be used to generate counterexamples to the original query.

appear on the frontier of our parse tree. Furthermore, both to avoid errors of this type and to improve efficiency, we added a constraint which makes all our features appear only on nodes in the parse tree, i.e., we ensured that all our features are subsets of the set encoding our parse tree. The variable we use to denote the parse tree is initialized as a connected, rooted set for convenience. That constraint is not necessary, but makes the results much more readable.

The resulting corrected XBar-predicate has 11 free variables and its description consists of the aforementioned five pages of MONA formulas. We only assumed absolutely necessary features and a minimum of lexical entries (in fact only two lexical entries and 8 features).[9] Nevertheless it represents a full module of a large scale formalization of a P&P theory, it could be compiled in less than 5 min, see Table 1, and it could be used in further verification tasks. This shows that—while we still do not know whether it is feasible to compile a grammar automaton—we can indeed handle interesting linguistic properties with these techniques. And, as a further advantage of the presented technique, we can verify even immensely complicated theories automatically and generate counterexamples which help in debugging.

4 Approximating P&P Grammars

As stated, the set of well-formed trees defined by a P&P grammar is not in general recognizable. Therefore we cannot hope in principle to construct a grammar formula which defines exactly the well-formed parse trees strongly generated by a P&P grammar using just the MSO logic–tree automaton correspondence. One question which arises immediately in a more practical and less theoretical setting is how well we can approximate a strictly context sensitive language with a context free grammar. There are two features of P&P grammars which help us here. First, P&P grammars are *principle-based*, so they are readily formalized in logic. The logic of WS2S is insufficient, but it should be easy to embed it in a more powerful logic which is sufficient. Then the techniques of general model theory become available to address the question of how good an approximation we can achieve. Secondly, P&P grammars are *modular*, so it may be possible to isolate those parts of the grammar which exceed the expressive capacity of WS2S. In that case we can observe quite directly the compromises that would have to be made to stay within an MSO logic for natural language. This is indeed the case here: the power of a P&P grammar seems to come rather directly from the assumption of an unbounded supply of referential indices, which can be used to keep track of arbitrarily many simultaneously occurring long-distance dependencies. Tree automata, on the other hand, can only keep track of a bounded number of distinct objects via their (finite) set of states. Hence, no formalization of a grammar in MSO logic is possible without the assumption of a bounded number of indices.

So the problem of approximating a full P&P grammar becomes the problem of estimating in advance how many indices will be enough. Clearly, any given

[9] We come back to the problem of lexical entries in Sect. 6.

sentence will require a bounded number of indices, but we cannot realistically expect to recompile our entire grammar automaton for each input sentence.[10] We can, however, precompile a number of different versions of a grammar offline, each with an index theory of a different size, and then select which grammar automaton to use given the input sentence. So it makes sense to consider how one could formalize the parts of the grammar that rely on indices in such a way that they can be parameterized with the number of indices and compiled in a variety of sizes.

Furthermore there are a number of ways in which one can "engineer" a theory so that it uses a minimal number of indices. For example, any one index can be used to keep track of any number of distinct chains, as long as those chains do not overlap. Also, there is no need to make each notionally distinct index correspond to a single feature: if we have n "index variables" I_i then we can use them as bits of a complex index, meaning that we have in fact 2^n possible distinct indices. We can extend this idea by making use of other features besides special purpose index (or index-bit) features: using bar-level features for example we can distinguish between the twelfth *Bar2* chain and the twelfth *Bar0* chain. Pursuing this strategy we end up with the approach Rogers actually employs. He uses no special purpose index variables but rather only combinations of those features which are independently required in the grammar. This reduces the number of possible alphabet symbols that an automaton must deal with.

Given such a bounded set of indices which we can use on a particular parse, we proceed to formalize the necessary conditions in our MSO tree logic. As noted above, what we use for our "indices" is formally relatively unimportant. What matters is what it means for two nodes to be co-indexed, see (3). That is the definition which gives the theory its "memory resources". Note that in MSO logic free variables must be represented in both the head of the formula and the resulting automaton to preserve their satisfying assignments.[11] Where readability is more of an issue, one can leave these "global" variables out under the assumption that they are implicit. However, since they cannot be ignored in the alphabet of the automata, we try to be more exact. Therefore, $\boldsymbol{I}_n$ stands for the n indices we have to create depending on the input, and the "definitions" we now present are really definition schemata.

(3)
$$\mathsf{Co\_Idx}(x, y, \boldsymbol{I}_n) \stackrel{def}{\Longleftrightarrow} (x \in I_1 \Leftrightarrow y \in I_1) \wedge \ldots \wedge (x \in I_n \Leftrightarrow y \in I_n)$$

After the instantiation of the schematic representation, we can compile an automaton for the corresponding memory limited grammar formula. All the predicates which depend on any predicate scheme at any point will themselves

[10] Note that the problem of estimating the number of indices a sentence will require is considerably simpler than the problem of estimating the number of traces it may contain. Roughly, every movable expression (overt or covert) is either headed by an overt lexical item or else licensed by an overt lexical item's selectional properties.

[11] If one is not interested in the information provided by a particular variable, it can be bound existentially at the top level of the formula.

have to be schematized, of course, and realized as families of definitions which vary in the number of their arguments. As an example of these we present a formula scheme encoding a simple version of a "trace binding condition" (a simplified version of the empty category principle which uses c-command instead of local government), which simply requires that all traces have a c-commanding antecedent with which they are co-indexed. TBind will have $n+2$ arguments; n depending on the number of indices currently allowed and the chosen coding. The needed c-command definition simply says that all nodes z which properly dominate x also have to dominate y and that x must not reflexively dominate y.

(4)
$$\mathsf{C\text{-}Com}(x,y) \overset{def}{\Longleftrightarrow} (\forall z)[z \lhd^{+} x \Rightarrow z \lhd^{+} y] \wedge \neg(x \lhd^{*} y)$$

(5)
$$\begin{aligned}
\mathsf{TBind}(P, Trace, \boldsymbol{I}_n) \overset{def}{\Longleftrightarrow}& \\
&\quad \%\ \text{for all traces in the parse tree} \\
&(\forall x \in P)[x \in Trace \Rightarrow \\
&\quad \%\ \text{there exists a proper antecedent.} \\
&\quad (\exists y \in P)[\mathsf{C\text{-}Com}(y,x) \wedge \mathsf{Co_Idx}(x,y,\boldsymbol{I}_n)]]
\end{aligned}$$

In this definition scheme we use a set called $Trace$ to identify traces. Naturally this presupposes that we formulate more appropriate constraints on the distribution of this label in the resulting parse tree in our grammar.

In conclusion, we note that the logic-automaton connection may be fruitful for doing principle-based parsing, even in spite of the principled limitation to weakly context-free languages. In particular, it seems unlikely that in any given corpus of input sentences there are any which require large numbers of overlapping chains: the same drain on memory which this causes for tree automata seems to affect human language users as well (Stabler 1994). The fact that we can adjust to such memory limitations without substantially affecting the underlying grammar—all we require is a special definition of what it means to be co-indexed—is especially welcome. However, for doing theory verification these limitations remain serious: given n indices, the claim that there are only n indices is a theorem, but not a theorem of the underlying grammar. On the other hand, if there is a sentence in the language which actually does have $n+1$ overlapping chains it will be rejected by a limited grammar. So we have neither soundness nor completeness for doing theorem proving.

5 Some Experimental Results

Even supposing, as seems reasonable, that real world parsers can function with a limited supply of indices, we still have to face the extreme explosive potential of the formula-to-automaton compilation procedure. It is still an open question which formulas can be compiled on present day computers. We used MONA

Table 1. Statistics for various predicates

Predicate/N	**Total Time**	$\|\mathbf{A}\|$	$\|\alpha\|$	$\|\mathbf{BDD}\|$
Connected	00:00:00.090	4	25	7
Path	00:00:00.130	4	21	9
C-Com	00:00:00.170	5	40	19
TBind/2	00:00:00.230	12	832	52
TBind/4	00:00:01.610	80	76288	422
TBind/8	00:55:15.000	2304	–	13881
TBind/16	core dump after approx. 10 min			
TBind′/4	00:00:07.480	87	145963	759
TBind′/6	00:04:58.830	457	–	4901
TBind′/7	core dump after approx. 30 min			
XBar	00:04:51.080	66	18133	982

(Klarlund and Møller 1998) to provide some answers to the usability of these ideas in practice.

The main innovation in MONA, which seems indeed to be a real technical breakthrough, is the use of binary decision diagrams (BDDs) to compress the transition tables of the output automata.[12] Since an alphabet of k features contains 2^k symbols, alphabet size, and hence transition table size, is a problem which, while only elementarily complex, can actually dominate in practice the non-elementary complexity of the actual compilation procedure. The use of this technique significantly improves performance over our own prototype so that we can advance considerably further with compilations than what we reported in Morawietz and Cornell (1997). Now we can actually implement significant modules of a large scale P&P theory and verify them (recall the discussion of X-Bar theory in Sect. 3). But we also encountered limits on the number of indices in the compilation of various TBind predicates. In this section we summarize some of our results so far.

Timings where done on a SUN Sparc 20 with 225 MB of RAM using a beta release of version 1.2 of MONA. Our results are displayed in Table 1. The first column of the table contains the predicate identifier and the number of indices (not the index bits). The second column contains the time needed to compile the automaton without the printing of the actual output automaton. The third column contains the number of states in the output automaton, the fourth column the number of transitions. The number of transitions has been compacted in the

[12] An introduction to BDDs can for example be found in Bryant (1992).

sense that it uses a don't care symbol, i.e., if transitions on q_1 and q_2 and both $\langle x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n \rangle$ and $\langle x_1, \ldots, x_{i-1}, \neg x_i, x_{i+1}, \ldots, x_n \rangle$ lead to the same state q, we have only a single transition on $\langle x_1, \ldots, x_{i-1}, \perp, x_{i+1}, \ldots, x_n \rangle$. Also note that for very large automata we cannot present the exact number of transitions any more since printing the result takes too much time and space and unfortunately MONA does not have an option to display (only) the number of transitions without printing the full automaton. The last column contains the size of the BDD used to represent the transition table. One readily observes the enormous gains this compression scheme confers by comparing this to the transition table size.

Consider the tricky "grammar engineering" question of what kind of coding to use for indices, as discussed above. In TBind, we used features to encode index bits and in TBind′ we naïvly used features to encode (disjoint) indices. The second approach turned out to consume considerable amounts of memory: we were only able to compile an automaton with six indices, though on a machine with 800 megabytes of RAM it is possible to compile TBind′ with at least seven indices (Nils Klarlund, p.c.). Using the index-bit encoding we could compile a version of TBind with eight indices (i.e., three index bit features) on our own machines; however, adding another bit again ran us out of memory resources.[13] The steepness of the relevant curves leads us to suspect that no present day machine will be able to compile TBind/16. The use of index bits cannot in principle reduce the memory load which long distance dependencies place on automata. This shows that MSO logic is exponentially more compact in coding the facts than tree automata.

An interesting question is whether a large machine can compile TBind′/9. More generally, the discreet encoding allows one to better approximate the capacity of available machines. Adding a bit increases the number of chains that can be represented too greatly, even though in general this encoding leads to more compact automata. The compactness gain is more than offset by the memory requirements that even one more chain places on a tree automaton's state space.

Another interesting point to observe is that the memory demands in these compilations mainly came from intermediate automata. Considering Table 1, one notes that the BDD size and number of states in the automata which are actually output are not overwhelmingly large. So while it requires rather large and powerful computers to manufacture automata from MSO formulas, the resulting automata can potentially be put to use on much smaller machines.

6 MSO Logic and CLP

The automaton for a grammar formula is presumably quite a lot larger than a parse-forest automaton, that is, the automaton for the grammar conjoined with an input description. Considering the problems we might have in extending the

[13] This happened even on the machines of the MONA crew. (Nils Klarlund, p.c.).

compilation of separate modules to an entire grammar automaton it makes sense to search for ways to incrementally construct the parse-forest automaton for a particular input. For this, we propose the embedding of the MSO constraint language into a CLP scheme. Intuitively, the constraint base is an automaton which represents the incremental accumulation of knowledge about the possible valuations of variables constructed by applying and solving clauses with their associated constraints. We can start with a description of the input and add further constraints incrementally rather then having to represent all constraints at once and then limiting them to an input. That is, we actually use the compiler *on line* as the constraint solver. Some obvious advantages include that we can still use our succinct and flexible constraint language with negation and MSO quantification. In fact, we gain even more power, since we can include inductive definitions of relations and have a way of guiding the compilation process under the control of a program.

We define a relational extension $\mathcal{R}(\mathrm{WS2S})$ of our constraint language following Höhfeld and Smolka (1988). Our constraint language WS2S is extended with a set of relations $\mathcal{R}$ such that we can formulate recursive definitions. From the scheme we get a sound and complete, but now only semi-decidable, operational interpretation of a definite clause based derivation process (for details see Morawietz (To appear)). The resulting structure of interpretation $\mathcal{R}(\mathrm{WS2S})$ is an extension of the underlying constraint structure for WS2S with the interpretation of the new relations in $\mathcal{R}$ constructed via the least fixpoints of their defining clauses. For simplicity we assume a standard left-to-right, depth-first interpreter for the execution of the programs. The solution to a search branch of a program is a satisfiable constraint, represented in solved form as an automaton.

To demonstrate how we split the work between compiler and CLP interpreter, we present as a simple example the following naïve specification of a lexicon:

$$
\begin{aligned}
&\mathsf{Lexicon}(x, Sees, V, John, N, \dots) \overset{def}{\Longleftrightarrow} \\
&\qquad (x \in Sees \wedge x \in V \wedge \dots) \\
&\qquad \vee (x \in John \wedge x \in N \wedge \dots) \\
&\qquad \vee \dots
\end{aligned} \tag{6}
$$

Remember that, naïvely at least, every feature we use (e.g., *Sees*, V), has to be coded into the alphabet Σ of the automaton. So, $|\Sigma|$ with an encoding as bitstrings will be 2^n (n the number of free variables). It is immediately clear that the compilation of such an automaton—even if we use the discussed approach of using lexicon-bits—is extremely unattractive, if at all feasible.[14] We have several

[14] Lexical entries seem to be rather special variables. Usually each of them appears only once on the frontier labeling a single node. They are disjoint from each other (we do not want a node to be simultaneously *John* and *Bill*). Maybe this can be used to define a more efficient compilation process. But considering the number of lexical entries needed in an NLP application, it seems unlikely that even the representation using BDDs could be able to reduce the number of necessary alphabet symbols sufficiently.

choices here. Either we find some special higher-level language built on top of WS2S to enable more efficient compilation of the variables denoting the lexical items (e.g., LISA, Ayari et al. 1998), or one could separate the lexicon from the pure MSO specification. In fact this would mean that the MSO constraints apply only to the pre-terminals of which there are considerably less distinct ones. Alternatively, we propose to avoid having to compile the entire lexicon as one automaton by having the obvious clauses for each lexical entry in the CLP extension. Note that we write R(WS2S) clauses lowercase whereas MSO constraints are written uppercase.

$$\mathsf{lexicon}(x, Sees, V, \dots) \longleftarrow \{x \in Sees \wedge x \in V \wedge \dots\} \tag{7}$$

$$\mathsf{lexicon}(x, John, N, \dots) \longleftarrow \{x \in John \wedge x \in N \wedge \dots\} \tag{8}$$

$$\vdots$$

Clearly this shifts the burden of handling disjunctions to the interpreter. But it is no longer the case that every constraint in the grammar—and therefore every occuring free variable—has to be expressed in one single tree automaton. Constraint resolution will reject all clauses not bearing a satisfiable constraint on the node labelings such that we only compile those features/free variables from clauses into the constraint store whose lexical entries are given in the input description.

Another benefit of the CLP extension is that we can use offline compiled modules in a $\mathcal{R}$(WS2S) parsing program. As a source of examples we use the definition of P&P parsing from Johnson (1995), see (9)–(13). We have to define the subgoals in the body either via precompiled automata (so they are essentially treated as facts), or else with more standard definite clause definitions. In (9) and in the following clauses we simplify notation to achieve better readability. Instead of specifying all free variables which appear in the clauses explicitly, we focus on the relevant ones. We use $\boldsymbol{V}$ to stand for all free variables which are global but not relevant for the clause we are looking at. In practical terms this means that we have to use the automata theoretic operation of cylindrification whenever we have to add newly appearing variables to the constraint store, i.e., the tree automaton representing it.

$$\begin{aligned} \mathsf{parse}(Words, Parse, \boldsymbol{V}) \longleftarrow & \\ & \{\mathsf{Tree}(Parse)\}\ \& \\ & \mathsf{yield}(Words, Parse, \boldsymbol{V})\ \& \\ & \mathsf{xbar}(Parse, \boldsymbol{V})\ \& \\ & \mathsf{ecp}(Parse, \boldsymbol{V}) \end{aligned} \tag{9}$$

We use two MSO variables to identify the input and the parse tree. *Words* denotes a set of nodes labeled according to the input description. Our initial con-

$$\mathsf{Tree}(P) \stackrel{def}{\Longleftrightarrow}$$

$$\begin{array}{l}
\quad\% \text{ A tree is a connected rooted set.}\\
\mathsf{Connected}(P) \wedge (\exists x \in P)[\mathsf{Root}(P, x)]
\end{array}$$

$$\mathsf{Root}(X, x) \stackrel{def}{\Longleftrightarrow}$$

$$\begin{array}{l}
\quad\% \ x \text{ dominates or equals all nodes in } X.\\
x \in X \wedge (\forall y \in X)[x \triangleleft^* y]
\end{array}$$

$$\mathsf{Yield}(W, P, \mathit{Empty}) \stackrel{def}{\Longleftrightarrow}$$

$$\begin{array}{l}
\quad\% \text{ All nodes of } P \text{ which have no daughter}\\
\quad\% \text{ are either in } W \text{ or phonologically empty}\\
(\forall x \in P)[((\neg(\exists y \in P)[x \triangleleft y]) \Leftrightarrow (x \in W \vee x \in \mathit{Empty})) \wedge\\
\quad\% \text{ but not both}\\
\qquad \neg(x \in W \wedge x \in \mathit{Empty})] \wedge\\
\quad\% \text{ and no other nodes are in } W.\\
W \subseteq P
\end{array}$$

Fig. 2. Some auxiliary formulas for the parse clause

straint base—which can be automatically generated from a list of input words—is the corresponding tree automaton. Its alphabet contains apart from the variables $Words$ and $Parse$ also the variables labeling the input string and the relevant precedence information. Now constraint solving and derivation refine information on variables which are already present, as is the case for the associated constraint Tree of the parse clause (for its full definition see Fig. 2). It is easily compilable and serves as the initialization for our parse tree. Or, in the case that we encounter new variables, the store is cylindrified accordingly. The Yield predicate serves as an example. It can be explicitly defined, compiled and treated as a fact (again, see Fig. 2), but it also introduces the additional MSO variable $Empty$ into the constraint base which labels phonologically empty nodes. Naturally, we need more constraints to achieve a correct distribution of this feature.

Since the ecp is dependent upon the actual number of indices, it has to be schematically expressed. But it can be realized as either a fact as a variant of TBind in (5) or a number of recursive rules which traverse the structure of the parse tree as in Johnson's proposal (Chap. 3, pp 43–48). Rather than going into more detail here, we illustrate the rule-based approach to the definition of a relation with the xbar predicate in the next example. It is a disjunctive specification of licensing conditions depending on different configurations of the features in the trees, which can be more intuitively expressed as two separate rules as given in (10) and (11). In fact, since we want the lexicon to be represented as clauses (and xbar anchors the projections in the lexicon), we cannot have xbar as a pure MSO constraint.

(10)
$$\begin{aligned}
&\mathsf{xbar}(X, \boldsymbol{V}) \leftarrow \\
&\qquad \{\mathsf{Sing}(X)\}\ \& \\
&\qquad \mathsf{lexicon}(X, \boldsymbol{V})
\end{aligned}$$

(11)
$$\begin{aligned}
&\mathsf{xbar}(X, \boldsymbol{V}) \leftarrow \\
&\qquad \{\mathsf{Subtrees}(X, x, Y, y, Z, z)\}\ \& \\
&\qquad \mathsf{local_xbar}(x, y, z, \boldsymbol{V})\ \& \\
&\qquad \mathsf{xbar}(Y, \boldsymbol{V})\ \& \\
&\qquad \mathsf{xbar}(Z, \boldsymbol{V})
\end{aligned}$$

The xbar clauses need some predicates defined in the associated constraints. They can be found in Fig. 3. The first xbar clause just licenses lexical elements. Note that this base case of xbar does not have a special clause for traces—we assume traces to be lexical entries. The second one allows binary forks in case there is a corresponding local rule and the subtrees obey the X-Bar theory. So we need further clauses specifying which local forks are valid. For examples of such clauses, see the following (12) and (13).

(12)
$$\begin{aligned}
&\mathsf{local_xbar}(x, y, z, \boldsymbol{V}) \leftarrow \\
&\qquad \{x \in CP \wedge y \in Wh \wedge z \in C1 \wedge y \prec z\}
\end{aligned}$$

(13)
$$\begin{aligned}
&\mathsf{local_xbar}(x, y, z, \boldsymbol{V}) \leftarrow \\
&\qquad \{x \in CP \wedge y \in C \wedge z \in IP \wedge y \prec z\}
\end{aligned}$$

$$\vdots$$

These clauses are taken straight out of Johnson's course notes. They do not represent a proper P&P account, but rather resemble phrase structure rules. Note that they are implemented simply by checking precedence constraints and membership in the right kind of feature set. For dominance we rely on the Subtrees constraint in the third clause of the X-Bar predicate.

For completeness, we also present a corresponding table of statistics for the auxiliary predicates we need for the clauses. But unfortunately we cannot present comparable timings for the CLP extension. So far, MONA cannot conveniently be used inside other programs. Therefore our very simple implementation of the basic interpreter has to use our own prototype of a compiler which is not sufficient for the compilation of (some of) the examples we give in the paper.[15] This is due to the two facts that we do not use BDDs to represent the transition tables such that the automata involved simply get too big for our constraint solver and that our implementation of the necessary cylindrification for simplicity uses assertion and retraction from files and databases which is too slow.

[15] For comparison, the compilation of Component which is a part of Roger' X-Bar theory takes on a comparable machine more than 330 seconds using our implementation, but less than 5 seconds using MONA.

$$
\begin{aligned}
\mathsf{Sing}(X) &\overset{def}{\iff} \\
&\qquad \%\text{ All subsets of } X \text{ are equal to } X \text{ or the empty set} \\
&(\forall Y, Z)[Y \subseteq X \Rightarrow (X \subseteq Y \vee Y \subseteq Z)] \wedge \\
&\qquad \%\text{ and } X \text{ is not the empty set.} \\
&(\exists Y)[X \not\subseteq Y] \\
\mathsf{Dtrs}(x, y, z) &\overset{def}{\iff} \\
&x \lhd y \wedge x \lhd z \wedge y \prec z \\
\mathsf{Subtree}(X, y, Y) &\overset{def}{\iff} \\
&\qquad \%\ Y \text{ is the subtree of } X \text{ rooted in } y. \\
&(\forall x \in X)[y \lhd^* x \Leftrightarrow x \in Y] \\
\mathsf{Subtrees}(X, x, Y, y, Z, z) &\overset{def}{\iff} \\
&\qquad \%\ x \text{ is the root of } X \text{ with daughters } y \text{ and } z \\
&\mathsf{Root}(X, x) \wedge \mathsf{Dtrs}(x, y, z) \wedge \\
&\qquad \%\text{ with subtrees } Y \text{ and } Z. \\
&\mathsf{Subtree}(X, y, Y) \wedge \mathsf{Subtree}(X, z, Z)
\end{aligned}
$$

Fig. 3. Some auxiliary formulas for the xbar clauses

7 Summary

Summing up, we can say that while it is theoretically not possible to write a formula covering a non-context-free theory, we can still use the independent parts of the formalization and families of definitions with respect to a particular input to answer questions of grammaticality of sentences. On the practical side, the advent of MONA has enabled us to advance significantly with the experiments on the compilation of P&P based grammatical theories. Although a definitive answer on the question whether an entire grammatical theory can be compiled into one automaton is still not possible, we know that we can compile non-trivial modules. The CLP extension we propose would allow us to combine these modules in a principled way and interleaves the intersection of the grammar and the input automata so that only the minimal amount of information needed to determine the parse is incrementally stored in the constraint base. It offers an even more powerful language which allows a clear separation of processing and specification issues while retaining the flexibility of the original.

But there are still many problems left. The form of the formulas has a large effect on the time required by the compiler and it is important to figure out which sort of formulas can be compiled more efficiently than others. Furthermore, writing grammar formulas in WS2S is an experience akin to assembler programming, i.e., error prone and time consuming. Therefore it remains to be seen how much

Table 2. More statistics for predicates

Predicate	Total Time	$\|\mathbf{A}\|$	$\|\alpha\|$	$\|\mathbf{BDD}\|$
Sing	00:00:00.050	3	12	5
Tree	00:00:00.130	4	23	8
Root	00:00:00.070	4	29	10
Yield	00:00:00.130	3	27	10
Dtrs	00:00:00.170	5	38	13
Subtree	00:00:00.340	5	67	19
Subtrees	00:00:00.290	7	168	49

impact higher-level languages such as FIDO (Klarlund and Schwartzbach 1997) and LISA have on the time required to formalize and compile P&P theories. The added power of the relational extension might also be seen as a source of problems. A closer investigation into which relations are definable inductively in MSO logic itself, into which stronger ones can be added without losing decidability and what effects are caused by them with respect to the generative capacity is necessary. Further work has to be done in the development of the CLP interpreter. Especially promising seems to be the incorporation of existential quantification into the relational extension which can be included into the Höhfeld and Smolka scheme. This theoretical background can be used to implement an explicit garbage collection procedure by marking certain variables for projection which are no longer needed thereby reducing the alphabet space of the constraint store, i.e., the size of the underlying automaton.

Acknowledgments. This work has been supported by the project A8 of the SFB 340 of the Deutsche Forschungsgemeinschaft. We wish especially to thank Uwe Mönnich and Jim Rogers for discussions and advice and three anonymous reviewers for comments on an abstract for this paper; Nils Klarlund and Anders Møller for valuable help with the installation and usage of MONA and Projects B4/B8 of the SFB 340 for letting us use their machine (especially Frank Richter). Needless to say, any errors and infelicities which remain are ours alone.

References

Ayari, A., Basin, D. and Podelski, A. (1998). LISA: A specification language based on WS2S, *in* M. Nielsen and W. Thomas (eds), *Computer Science Logic, 11th International Workshop, CSL'97*, LNCS 1414, Springer.

Basin, D. and Klarlund, N. (To Appear). Automata based symbolic reasoning in hardware verification, *The Journal of Formal Methods in Systems Design*.

Bryant, R. E. (1992). Symbolic boolean manipulation with ordered binary-decision diagrams, *ACM Computing Surveys* **24**(3): 293–318.

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

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

Frank, R. and Vijay-Shanker, K. (1998). TAG Derivation as Monotonic C-Command, *Proceedings of the Fourth Workshop on Tree-Adjoining Grammars and Related Frameworks TAG+98*, Philadelphia, PA. (Extended abstract)

Gécseg, F. and Steinby, M. (1997). Tree languages, *in* G. Rozenberg and A. Salomaa (eds), *Handbook of Formal Languages: Beyond Words*, Vol. 3, Springer, Berlin.

Gurevich, Y. (1985). Monadic second-order theories, *in* J. Barwise and S. Feferman (eds), *Model-Theoretic Logics*, Springer, Heidelberg, pp. 479–506.

Henriksen, J. G., Jensen, J., Jørgensen, M., Klarlund, N., Paige, R., Rauhe, T. and Sandhol, A. (1995). MONA: Monadic second-order logic in practice, *in* Brinksma et al. (eds), *International Workshop TACAS '95*, LNCS 1019, Springer, pp. 89–110.

Höhfeld, M. and Smolka, G. (1988). Definite relations over constraint languages, *LILOG Report 53*, IBM Deutschland, Stuttgart, Germany.

Johnson, M. (1995). Constraint-based natural language parsing, European Summer School of Logic, Language and Information (ESSLLI '95), Barcelona, Course notes.

Kayne, R. S. (1994). *The Antisymmetry of Syntax*, Vol. 25 of *Linguistic Inquiry Monographs*, MIT Press, Cambridge, Mass. and London, England.

Kelb, P., Margaria, T., Mendler, M. and Gsottberger, C. (1997). Mosel: A flexible toolset for monadic second-order logic, *in* E. Brinksma (ed), *International Workshop TACAS '97*, LNCS 1019, Springer, pp. 183–202.

Klarlund, N. and Møller, A. (1998). *MONA Version 1.2 User Manual*, BRICS Notes Series NS-98-3, Department of Computer Science, University of Aarhus.

Klarlund, N. and Schwartzbach, M. I. (1997). A domain-specific language for regular sets of strings and trees, *Proceedings of the Conference on Domain-Specific Languages*, USENIX, Santa Barbara, Ca.

Morawietz, F. (To appear). Monadic second order logic, tree automata and constraint logic programming, *in* H.-P. Kolb and U. Mönnich (eds), *The Mathematics of Syntactic Structure*, Mouton de Gruyter.

Morawietz, F. and Cornell, T. L. (1997). Representing constraints with automata, *Proceedings of the 35th Annual Meeting of the ACL and the 8th Conference of the EACL*, Madrid, Spain, pp. 468–475.

Niehren, J. and Podelski, A. (1992). Feature automata and recognizable sets of feature trees, *in* M.-C. Gaudel and J.-P. Jouannaud (eds), *Proceedings of the 4th International Joint Conference on Theory and Practice of Software Development*, Springer, LNCS 668, Orsay, France, pp. 356–375.

Rabin, M. O. (1969). Decidability of second-order theories and automata on infinite trees, *Transactions of the American Mathematical Society* **141**: 1–35.

Rizzi, L. (1990). *Relativized Minimality*, MIT Press.

Roche, E. and Schabes, Y. (1997). Introduction to finite-state devices in natural language processing, *in* E. Roche and Y. Schabes (eds), *Finite-State Language Processing*, Language, Speech, and Communication Series, MIT Press.

Rogers, J. (To appear). *A Descriptive Approach to Language-Theoretic Complexity*, Studies in Logic, Language and Information, CSLI Publications.

Stabler, E. (1994). The finite connectivity of linguistic structure, *in* C. Clifton et al. (eds), *Perspectives on Sentence Processing*, Lawrence Erlbaum, New Jersey.

Thatcher, J. W. and Wright, J. B. (1968). Generalized finite automata theory with an application to a decision problem of second-order logic, *Mathematical Systems Theory* **2**(1): 57–81.