Using MONA for Querying Linguistic Treebanks

Stephan Kepser*
Collaborative Research Centre 441
University of Tübingen
Tübingen, Germany
kepser@sfs.uni-tuebingen.de

Abstract

MONA is an automata toolkit providing a compiler for compiling formulae of monadic second order logic on strings or trees into string automata or tree automata. In this paper, we evaluate the option of using MONA as a treebank query tool. Unfortunately, we find that MONA is not an option. There are several reasons why the main being unsustainable query answer times. If the treebank contains larger trees with more than 100 nodes, then even the processing of simple queries may take hours.

1 Introduction

In recent years large amounts of electronic texts have become available providing a new base for empirical studies in linguistics and offering a chance to linguists to compare their theories with large amounts of utterances from "the real world". While tagging with morphosyntactic categories has become a standard for almost all corpora, more and more of them are nowadays annotated with refined syntactic information. Examples are the Penn Treebank (Marcus et al., 1993) for American English annotated at the University of Pennsylvania, the French treebank (Abeillé and Clément, 1999) developed in Paris, the TIGER Corpus (Brants et al., 2002) for German annotated at the Universities of Saarbrücken and Stuttgart, and the Tübingen Treebanks (Hinrichs et al., 2000) for Japanese, German and English from the University of Tübingen. To make these rich syntactic annotations accessible for linguists, the development of powerful query tools is an obvious need and has become an important task in computational linguistics.

Consequently, a number of treebank query tools have been developed. Probably amongst the most important ones are CorpusSearch (Randall, 2000), ICECUP III (Wallis and Nelson, 2000), fsq (Kepser, 2003), TGrep2 (Rohde, 2001), and TIGERSearch (König and Lezius, 2000). A common feature of these tools is the relatively low expressive power of their query languages. Explicit or implicit references to nodes in a tree are mostly interpreted existentially. The notable exception is fsq, which employs full first order logic as its query language.

The importance of the expressive power of the query language is a consequence of the sizes of the available treebanks, which can contain several tenthousand trees. It is clearly impossible to browse these treebanks manually searching for linguistic phenomena. But a query tool that does not permit the user to specify the sought linguistic phenomenon quite precisely is not too helpful, either. If the user can only approximate the phenomenon he seeks answer sets will be very big, often containing several hundred to thousand trees. Weeding through answer sets of this size is cumbersome and not really fruitful. If the task is to gain small answer sets, then query languages must be powerful. The reason why the above mentioned query tools still offer query languages of limited expressive power is the fear that

* This research was funded by a German Science Foundation grant (DFG SFB441-6).

Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP), pages 555–562, Vancouver, October 2005.

there may be a price to be paid for offering a powerful query language, namely longer query answer times due to more complex query evaluation algorithms.

At least on a theoretical level, this fear is not necessarily justified. As was recently shown by Kepser (2004), there exists a powerful query language with a query evaluation algorithm of low complexity. The query language is monadic second-order logic (MSO henceforth), an extension of first-order logic that additionally allows for the quantification over *sets* of tree nodes. The fact that makes this language so appealing beyond its expressive power is that the evaluation time of an MSO query on a tree is only linear in the size of the tree. The query evaluation algorithm proceeds in two steps. In the first step, a query is compiled into an equivalent tree automaton. In the second, the automaton is run on each tree of the treebank. Since a run of an automaton on a tree is linear in the size of the tree, the evaluation of an MSO query is linear in the size of a tree.

There has sometimes been the question whether the expressive power of MSO is really needed. Beyond the statements above about retrieving small answer sets there is an important argument concerning the expressive power of the grammars underlying the annotation of the treebanks. A standard assumption in the description of the syntax of natural languages is that at least context-free string grammars are required. On the level of trees, these correspond to regular tree grammars (Gécseg and Steinby, 1997). It is natural to demand that the expressive power of the query language matches the expressive power of the underlying grammar. Otherwise there can be linguistic phenomena annotated in the treebank for which a user cannot directly query. The query language which exactly matches the expressive power of regular tree grammars is MSO. In other words, a set of trees is definable by a regular tree grammar iff there is an MSO formula that defines this set of trees (Gécseg and Steinby, 1997). Hence MSO is a natural choice of query language under the given assumption that the syntax of natural language is (at least) context-free on the string or token level.

Since the use of MSO as a query language for treebanks is – at least on a theoretical level – quite appealing, it is worth trying to develop a query system that brings these theoretical concepts to practise. The largest and most demanding subpart of this enterprise is the development of a tree automata toolkit, a toolkit that compiles formulae into tree automata and performs standard operations on tree automata such as union, intersection, negation, and determination. Since this task is very demanding, it makes sense to investigate whether one could use existing tree automata toolkits before starting to develop a new one. To the authors' knowledge, there exists only one of-the-shelf usable tree automata toolkit, and that is MONA (Klarlund, 1998). It is the aim of this paper to give an evaluation of using MONA for querying linguistic treebanks.

2 The Tree Automata Toolkit MONA

Tree automata are generalisations of finite state automata to trees. For a general introduction to tree automata, we refer the reader to (Gécseg and Steinby, 1997). There exists a strong connection between tree automata and MSO. A set of trees is definable by an MSO formula if and only if there exists a tree automaton accepting this set. This equivalence is constructive, there is an algorithm that constructs an automaton from a given MSO formula.

MONA is an implementation of this relationship. It is being developed by Nils Klarlund, Anders Møller, and Michael Schwartzbach. Its intended main uses are hardware and program verification. MONA is actually an implementation of the compilation of monadic second order logic on *strings and* trees into finite state automata or tree automata, respectively. But we focus exclusively on the tree part here. As we will see later, MONA was not developed with linguistic applications in mind.

2.1 The Language of MONA

The language of MONA is pure monadic second order logic of two successors. We will only mention the part of the language that is needed for describing trees. There are first-order and second-order terms. A first-order variable is a first-order term. The constant *root* is a first-order term denoting the root node of a tree. If t is a first-order term and s is a non-empty sequence of 0's and 1's, then $t.s$ is a first-order term. 0 denotes the left daughter, and 1 the

right daughter of a node. A sequence of 0's and 1's denotes a path in the tree. The term *root*.011, e.g., denotes the node that is reached from the root by first going to the left daughter and then going twice down to the right daughter. A set variable is a second-order term. If $t_1, \ldots, t_k$ are first-order terms then $\{t_1, \ldots, t_k\}$ is a second-order term. We consider the following formulae. Let t, t' be first-order terms and T, T' be second order terms. Atomic formulae are

- $t = t'$ – Equality of nodes,
- $T = T'$ – Equality of node sets,
- t `in` T – t is a member of set T,
- `empty`(T) – Set T is empty.

Formulae are constructed from atomic formulae using the boolean connectives and quantification. Let ϕ and ψ be formulae. Then we define complex formulae as

- $\sim \phi$ – Negation of ϕ,
- ϕ & ψ – Conjunction of ϕ and ψ,
- $\phi \mid \psi$ – Disjunction of ϕ and ψ,
- ϕ `=>` ψ – Implication of ϕ and ψ,
- `ex1` $x : \phi$ – First-order existential quantification of x in ϕ,
- `all1` $x : \phi$ – First-order universal quantification of x in ϕ,
- `ex2` $X : \phi$ – Existential quantification of set X in ϕ,
- `all2` $X : \phi$ – Universal quantification of set X in ϕ.

We note that there is no way to extend this language. This has three important consequences. Firstly, we are restricted to using *binary* trees only. And secondly, we cannot accommodate linguistic labels in a direct way. We have to find some coding. Finally, and this is a significant drawback that may exclude the use of MONA for many applications, we cannot code the tokens, the word sequence at the leaves of a tree. Hence we can neither query for particular words or sequences of words. We can only query the structure of a tree – including the labels.

2.2 The MONA Compiler

The main program of MONA is a compiler that compiles formulae in the above described language into tree automata. The input is a file containing the formulae. The output is an analysis of the automaton that is constructed. In particular, it is stated whether the formula is satisfiable at all, i.e., whether an automaton can be constructed.

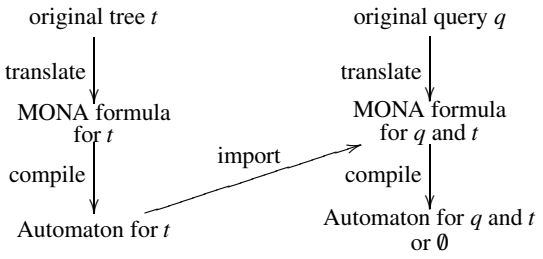

Figure 1: Method of using MONA for querying.

MONA does not provide a method to execute an automaton on a tree. But if a formula can be compiled into an automaton, this automaton can be output to file. And a file containing an automaton can be imported into a file containing a formula. We therefore use the following strategy to query treebanks using MONA. Each tree from the treebank is translated into a formula in the MONA language. How this can be done, will be described later. The formula representing the tree is then compiled into an automaton and written to file. Now the treebank exists as a set of automata files. A query to the original treebank will also be translated into a MONA formula. For each tree of the treebank, this formula is extended by an import statement for the automaton representing the tree. If and only if the extended formula representing query and tree can be compiled into an automaton, then the tree is a match for the original query. This way we can use MONA to query the treebank. The method is depicted in Figure 1.

3 The Tübingen Treebanks

In order to evaluate the usability of MONA as a query tool we had to chose some treebank to do our evaluation on. We opted for the Tübingen Treebank of spoken German. The Tübingen Treebanks, annotated at the University of Tübingen, comprise a German, an English and a Japanese treebank consisting of spoken dialogs restricted to the domain of arranging business appointments. For our evaluation, we focus on the German treebank (TüBa-D/S) (Stegmann et al., 2000; Hinrichs et al., 2000) that contains approximately 38.000 trees.

The treebank is part-of-speech tagged using the Stuttgart-Tübingen tag set (STTS) developed by

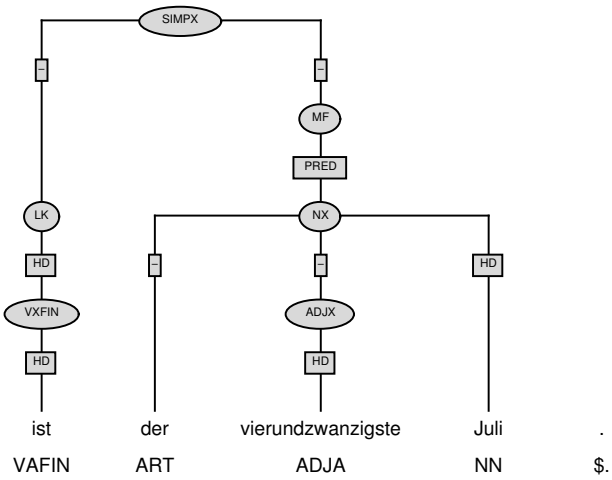

Figure 2: An example tree from TüBa-D/S.

Schiller et al. (1995). One of the design decisions for the development of the treebank was the commitment to reusability. As a consequence, the choice of the syntactic annotation scheme should not reflect a particular syntactic theory but rather be as theory-neutral as possible. Therefore a surface-oriented scheme was adopted to structure German sentences that uses the notion of topological fields in a sense similar to that of Höhle (1985). The verbal elements have the categories `LK` (*linke Klammer*) and `VC` (*verbal complex*); roughly everything preceeding the `LK` forms the "Vorfeld" `VF`, everything between `LK` and `VC` forms the "Mittelfeld" `MF`, and the material following the `VC` forms the "Nachfeld" `NF`.

The treebank is annotated with syntactic categories as node labels, grammatical functions as edge labels and dependency relations. The syntactic categories follow traditional phrase structure and the theory of topological fields. An example of a tree can be found in Figure 2. To cope with the characteristics of spontaneous speech, the data structures in the Tübingen Treebanks are of a more general form than trees. For example, an entry may consist of several tree structures. It may also contain completely disconnected nodes. In contrast to TIGER or the Penn Treebank, there are neither crossing branches nor empty categories.

There is no particular reason why we chose this treebank. Many others could have been used as well for testing the applicability of MONA.

4 Converting Trees into Automata

4.1 Translating Trees into Tree Descriptions

When translating trees from the treebank into MONA formulae describing these trees we consider proper trees only. Many treebanks, including TüBa-D/S, contain more complex structures than proper trees. For the evaluation purpose here we simplify these structures as follows. We ignore the secondary relations. And we introduce a new super root. All disconnected subparts are connected to this super root. Note that we employ this restriction for the evaluation purpose only. The general method does not require these restrictions, because even more complex tree-like structures can be recoded into proper binary trees, as is shown in (Kepser, 2004).

As stated above, the translation of trees into formulae has to perform two tasks. The trees, which are arbitrarily branching, must be transformed into binary trees. And the linguistic labels, i.e., the node categories and grammatical functions, have to be coded. For the transformation into binary trees, we employ the First-Daughter-Next-Sibling encoding, a rather standard technique. Consider an arbitrary node x in the tree. If x has any daughters, its leftmost daughter will become the left daughter of x in the transformed tree. If x has any sisters, then its leftmost sister will become the right daughter of x in the transformed tree. This transformation is applied recursively to all nodes in the tree. For example, the tree in Figure 2 is transformed into the binary tree in Figure 3.

Note how the disconnected punctuation node at the lower right corner in Figure 2 becomes the right daughter of the SIMPX node in Figure 3. Note also that we have both category and grammatical function as node labels for those nodes that have a grammatical function.

Such a binary tree is now described by a several formulae. The first formula, called *Carcase*, collects the addresses of all nodes in the tree to describe the tree structure without any labels. For our example tree, the formula would be

$$\begin{aligned}\text{Carcase} = \{&root, root.0, root.00, root.000, root.0000,\\ &root.01, root.001, root.0010, root.00100,\\ &root.001001, root.0010010, root.0010011\}.\end{aligned}$$

A syntactic category or grammatical function is coded as the set of nodes in the tree that are labelled

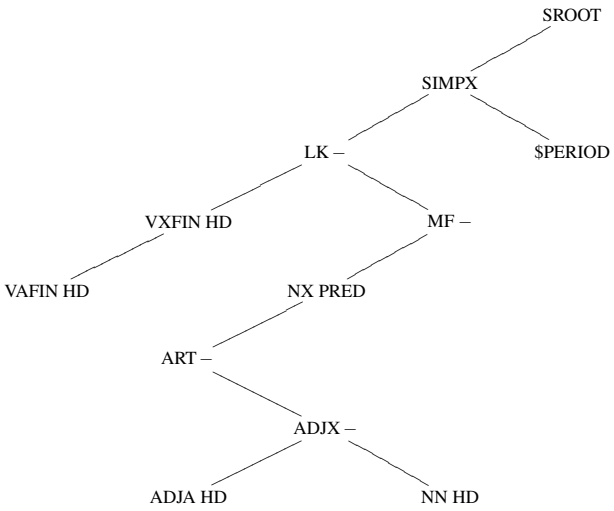

Figure 3: Binary recoding of the tree in Figure 2.

with this category or function. This is the way to circumvent the problem that we cannot extend the language of MONA. Here are some formulae for some labels of the example tree.
LK $= \{root.00\}$, ART $= \{root.00100\}$, HD $= \{root.000, root.0000, root.0010010, root.0010011\}$.

For all category or function labels that are not present at a particular tree, but part of the label set of the treebank, we state that the corresponding sets are empty. For example, the description of the example tree contains the formula `empty`(VC).

We implemented a program that actually performs this translation step. The input is a fraction of the TüBa-D/S treebank in NEGRA export format (Brants, 1997). The output is a file for each tree containing the MONA formulae coding this tree. In this way, we get a set of MONA formulae describing each tree.

4.2 Compiling Tree Descriptions into Automata

As mentioned above, the next step consists in compiling each tree description into an equivalent automaton. This is the first part of the evaluation. We tested whether MONA can actually perform this compilation. Astonishingly, the answer is not as simple as one might expect. It turns out that the computing power required to perform the compilation is quite high. To start, we chose a very small subset of the TüBa-D/S, just 1000 trees. Some of these trees contain more than 100 nodes, one more than 200 nodes. Processing descriptions of these large trees actually requires a lot of computing power.

It seems it is *not* possible to perform this compilation step on a desktop machine. We used an AMD 2200 machine with 2GB Ram for a try, but aborted the compilation of the 1000 trees after 15 hours. At that time, only 230 trees had been compiled.

To actually get through the compilation of the treebank we transfered the task to a cluster computer. On this cluster we used 4 nodes each equipped with two AMD Opteron 146 (2GHz, 4GB Ram) in parallel. Parallelisation is simple since each tree description can be compiled independently of all the others. The parallelisation was done by hand. Using this equipment we could compile 999 trees in about 4 hours. These 4 hours are the time needed to complete the whole task, not pure processing time. The tree containing more than 200 nodes could still not be compiled. Its compilation terminated unsuccessfully after 6 hours. We decided to drop this tree from the sample.

It is obvious that this is a major obstacle for using MONA. It is difficult to believe that many linguists will have access to a cluster computers and sufficient knowledge to use it. And we expect on the base of our experiences that a compilation on an ordinary desktop machine can take several days, provided the machine is equipped with large amounts of memory. Otherwise it will fail. One still has to consider that 1000 trees are not much. The TIGER corpus and the TüBa-D/S have each about 40.000 trees. Thus one may argue that this fact alone makes MONA unsuitable for use by linguists. But the compilation step has to be performed only once. The files containing the resulting automata are machine independent. Hence a corpus provider could at least in theory provide his corpus as a collection of MONA automata. This labour would be worth trying, if the resulting automata could be used for efficient querying.

5 Querying the Treebank

In order to query the treebank we designed a query language that has MSO as its core but contains features desirable for treebank querying. Naturally the language is designed to query the original trees, not their codings. It is therefore necessary to translate a query into an equivalent MONA formula that re-

spects the translation of the trees.

5.1 The Query Language

The query language is defined as follows. The language has a LISP-like syntax. First-order variables ($x, y, \ldots$) range over nodes, set variables ($X, Y, \ldots$) range over sets of nodes. The atomic formulae are

- `(cat x NX)` – Node x is of category `NX`,
- `(fct x HD)` – Node x is of grammatical function `HD`,
- `(> x y)` – Node x is the mother of node y,
- `(>+ x y)` – Node x properly dominates y,
- `(. x y)` – Node x is immediately to the left of y,
- `(.. x y)` – Node x is to the left of y,
- `(= x y)` – Node x and y are identical,
- `(= X Y)` – Node sets X and Y are identical,
- `(in x X)` – Node x is a member of set X.

Complex formulae are constructed by boolean connectives and quantification. Let x be a node variable, X a set variable, and ϕ and ψ formulae. Then we have

- `(! φ)` – Negation of ϕ,
- `(& φ ψ)` – Conjunction of ϕ and ψ,
- `(| φ ψ)` – Disjunction of ϕ and ψ,
- `(-> φ ψ)` – Implication of ϕ and ψ,
- `(E x φ)` – Existential quantification of x in ϕ,
- `(A x φ)` – Universal quantification of x in ϕ,
- `(E2 X φ)` – Existential quantification of set variable X in ϕ,
- `(A2 X φ)` – Universal quantification of set variable X in ϕ.

5.2 Translating the Query Language

The next step consists of translating queries in this language into MONA formulae. As is simple to see, the translation of the complex formulae is straight forward, because they are essentially the same in both languages. The more demanding task is connected with the translation of formulae on category and function labels and the tree structure, i.e., dominance and precedence.

As described above, categories and functions are coded as sets. Hence a query for a category or function is translated into a formula expressing set membership in the relevant set. For example, the query `(cat x SIMPX)` is translated into `(`x `in SIMPX)`.

The translations of dominance and precedence are the most complicated ones, because we transformed the treebank trees into binary trees. Now we have to reconstruct the original tree structures out of these binary trees. In the first step we have to define dominance on coded binary trees. The MONA language contains formulae for the left and right daughter of a node, but there is no formula for dominance, the transitive closure of the daughter relation. That we can define dominance at all is a consequence of the expressive power of MSO. As was shown by Courcelle (1990), the transitive closure of any MSO-definable binary relation is also MSO-definable. Let R be an MSO-definable binary relation. Then

$$\forall X \ (\forall z, w(z \in X \wedge R(z,w) \rightarrow w \in X) \wedge \forall z(R(x,z) \rightarrow z \in X)) \rightarrow y \in X$$

is a formula with free variables x and y that defines the transitive closure of R. If we now take $R(x,y)$ in the above formula to be $(x.0 = y \mid x.1 = y)$ we define dominance (dom). In a similar fashion we can define that y is on the rightmost branch of x ($rb(x,y)$) by taking $R(x,y)$ to be $(x.1 = y)$.

Now for immediate dominance, if node x is the mother of y in the original tree, we have to distinguish to cases. In the simpler case, y is the leftmost daughter of x, so after transformation, y is the left daughter of x. Or y is not the leftmost daughter of x, in that case it is a sister of the leftmost daughter z of x. All sisters of z are found on the rightmost branch of z in the transformed trees. Hence `(> x y)` is translated into $(x.0 = y \mid \texttt{ex1}\ z : x.0 = z \,\&\, rb(z,y))$.

Proper dominance is treated similarly. If we iterate the above argument that the daughters of a node x in the original tree become the left daughter z of x and the rightmost successors of z, we can see that z and all the nodes dominated by z in the translated tree are actually all the nodes dominated by x in the original tree. Hence `(>+ x y)` is translated into $(x.0 = y \mid \texttt{ex1}\ z : x.0 = z \,\&\, dom(z,y))$.

For precedence, consider a node x in a coded binary tree. By definition the left daughter of x and all her successors are nodes that preceed the right daughter of x and her successors in the original tree. Thus `(.. x y)` is translated into

$$(x.1 = y \mid (\ \texttt{ex1}\ z,w,v : z.0 = w \,\&\, z.1 = v \,\&\, (w = x \mid dom(w,x)) \,\&\, (v = y \mid dom(v,y))))).$$

Immediate precedence can be expressed using precedence. Node x immediately precedes y if x precedes y there is no node z that is preceeded by x and precedes y.

There is a small issue in the translation of quantified formulae. In the translation of a first-order quantification (existential or universal) of a variable x we have to make sure that x actually ranges over the nodes in a particular tree. Otherwise MONA may construct an automaton that contains the coded tree as a substructure, but is more general. In such a case we could no longer be certain that a solution found by MONA actually represents a proper match of the original query on the original tree. To solve this problem, we add (x `in Carcase`) to the translation of `(E x` ϕ`)` or `(A x` ϕ`)`. E.g., `(E x` ϕ`)` translates to (`ex1` $x : x$ `in Carcase` & ϕ') where ϕ' is the translation of ϕ. The same holds – mutatis mutandis – for set variable quantification.

5.3 Performing a Query

We implemented a small program that performs the above described translation of queries. It actually does a little bit more. It adds the defining formulae for *dom* and *rb*. Furthermore, as mentioned above, MONA allows to include a precompiled automaton into a set of MONA formulae via a special import declaration. Such an import declaration is used to include the automata representing the (coded) trees from the treebank. Thus the set of MONA formulae to evaluate a query consist of the translation of the query, the formulae for *dom* and *rb*, and an import declaration for one tree from the treebank. This set of MONA formulae can now be fed into MONA to try to compile it into an automaton. If the compilation is successful, there exists an automaton that at the same time represents the translation of the query and the translation of the given tree. Hence the tree is a match for the query. If there is no automaton, the tree is no match for the query. To perform the query on the whole treebank there is a loop that stepwise imports every tree and calls MONA to check if an automaton can be compiled. The result is the set of tree IDs that identify the trees that match the query.

We tested this method on our small treebank of 999 trees from TüBa-D/S. Unfortunately it turned out that the reloading of large precompiled automata (representing large trees) also requires enormous computational resources. We experimented with a very simple query: $\exists x\,\mathrm{NX}(x)$ (or `(E x (cat x NX))`). On our desktop machine (AMD 2200, 2GB Ram), it took 6 hours and 9 minutes to process this query. If we pose the same query on the whole treebank TüBa-D/S (with about 38.000 trees) using established query tools like TIGERSearch or fsq, processing time is about 5 seconds. Hence the method of using MONA is clearly *not appropriate* for desktop computers.

Even access to larger computing power does not solve the problem. We processed the same query on one processor (AMD Opteron 146, 2GHz, 4GB Ram) of the cluster computer mentioned above. There it took 1 minute and 30 seconds. About the same query answer time was required for a second, more complex query that asked for two different NX nodes and a third SIMPX node. These query answer times are still too long, because we queried only about one fortieth of the whole treebank. Since each tree is queried separately, we can expect a linear time increase in the query time in the number of trees. In other words, evaluating the query on the whole treebank would probably take about 1 hour. And that on a computer with such massive computing power. TIGERSearch and fsq are 720 times faster, and they run on desktop computers.

6 Conclusions

Despite the many reported successful applications of MONA in other areas, we have to state that MONA is clearly *not* a choice for querying linguistic treebanks. Firstly, we cannot use MONA to query for tokens (or words). Secondly, the compilation of a treebank into a set of automata is extremely difficult and resources consuming, if not impossible. And finally, practical query answer times are way too long. Apparently, reloading precompiled automata representing large trees takes too much time, because the automata representing these large trees are themselves huge.

We note that this is unfortunately not the first negative experience of trying to apply MONA to computational linguistics tasks. Morawietz and Cornell (1999), who try to use MONA to compile logical formalisations of GB-theory, also report that automata get too large to work with.

The general problem behind these two unsuccessful applications of MONA to problems in computational linguistics seems to be that MONA does not allow users to define their own signatures. Hence linguistic labels have to be coded in an indirect fashion. Though this coding works in theory, the resulting automata can become huge. The reason for this explosion in automata size, though, remains mysterious.

The negative experience we made with MONA does on the other hand not mean that the whole enterprise of using tree automata for querying treebanks is deemed to fail. It seems that it is rather this particular deficit of MONA of providing no direct way to cope with labelled trees that causes the negative result. It could therefore well be worth trying to implement tree automata for labelled trees and use these for treebank querying.

References

Anne Abeillé and Lionel Clément. 1999. A tagged reference corpus for French. In *Proceedings of EACL-LINC*.

Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. 2002. The TIGER Treebank. In Kiril Simov, editor, *Proceedings of the Workshop on Treebanks and Linguistic Theories*, Sozopol.

Thorsten Brants. 1997. The NEGRA export format. CLAUS Report 98, Universität des Saarlandes, Computerlinguistik, Saarbrücken, Germany.

Bruno Courcelle. 1990. Graph rewriting: An algebraic and logic approach. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 5, pages 193–242. Elsevier.

Ferenc Gécseg and Magnus Steinby. 1997. Tree languages. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Vol 3: Beyond Words*, pages 1–68. Springer-Verlag.

Erhard Hinrichs, Julia Bartels, Yasuhiro Kawata, Valia Kordoni, and Heike Telljohann. 2000. The VERBMOBIL treebanks. In *Proceedings of KONVENS 2000*.

Tilman Höhle. 1985. Der Begriff 'Mittelfeld'. Anmerkungen über die Theorie der topologischen Felder. In A. Schöne, editor, *Kontroversen, alte und neue. Akten des 7. Internationalen Germanistenkongresses*, pages 329–340.

Stephan Kepser. 2003. Finite Structure Query: A tool for querying syntactically annotated corpora. In Ann Copestake and Jan Hajič, editors, *Proceedings EACL 2003*, pages 179–186.

Stephan Kepser. 2004. Querying linguistic treebanks with monadic second-order logic in linear time. *Journal of Logic, Language, and Information*, 13:457–470.

Nils Klarlund. 1998. Mona & Fido: The logic-automaton connection in practice. In *Computer Science Logic, CSL '97*, LNCS 1414, pages 311–326. Springer.

Esther König and Wolfgang Lezius. 2000. A description language for syntactically annotated corpora. In *Proceedings of the COLING Conference*, pages 1056–1060.

Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Frank Morawietz and Tom Cornell. 1999. The MSO logic-automaton connection in linguistics. In Alain Lecomte, François Lamarche, and Guy Perrier, editors, *Logical Aspects of Computational Linguistics*, LNCS 1582, pages 112–131. Springer.

Beth Randall. 2000. CorpusSearch user's manual. Technical report, University of Pennsylvania. `http://www.ling.upenn.edu/mideng/ppcme2dir/`.

Douglas Rohde. 2001. Tgrep2. Technical report, Carnegie Mellon University.

Anne Schiller, Simone Teufel, and Christine Thielen. 1995. Guidelines für das Tagging deutscher Textcorpora mit STTS. Manuscript, Universities of Stuttgart and Tübingen.

Rosmary Stegmann, Heike Telljohann, and Erhard Hinrichs. 2000. Stylebook for the German treebank in VERBMOBIL. Technical Report 239, SfS, University of Tübingen.

Sean Wallis and Gerald Nelson. 2000. Exploiting fuzzy tree fragment queries in the investigation of parsed corpora. *Literary and Linguistic Computing*, 15(3):339–361.