In Proceedings of the 9th Conference of the EACL, Bergen, Norway, June 1999.

Selective Magic HPSG Parsing

Guido Minnen*
Cognitive and Computing Sciences, University of Sussex
Falmer, Brighton BN1 9QH
United Kingdom
Guido.Minnen@cogs.susx.ac.uk
www.cogs.susx.ac.uk/lab/nlp/minnen/minnen.html

Abstract

We propose a parser for constraint-logic grammars implementing HPSG that combines the advantages of dynamic bottom-up and advanced top-down control. The parser allows the user to apply magic compilation to specific constraints in a grammar which as a result can be processed dynamically in a bottom-up and goal-directed fashion. State of the art top-down processing techniques are used to deal with the remaining constraints. We discuss various aspects concerning the implementation of the parser as part of a grammar development system.

1 Introduction

In case space requirements of dynamic parsing often outweigh the benefit of not duplicating sub-computations. We propose a parser that avoids this drawback through combining the advantages of dynamic bottom-up and advanced top-down control.[1] The underlying idea is to achieve faster parsing by avoiding tabling on sub-computations which are not expensive. The so-called *selective magic parser* allows the user to apply magic compilation to specific constraints in a grammar which as a result can be processed dynamically in a bottom-up and goal-directed fashion. State of the art top-down processing techniques are used to deal with the remaining constraints.

Magic is a compilation technique originally developed for goal-directed bottom-up processing of logic programs. See, among others, (Ramakrishnan et al. 1992). As shown in (Minnen, 1996) magic is an interesting technique with respect to natural language processing as it incorporates filtering into the logic underlying the grammar and enables elegant control independent filtering improvements. In this paper we investigate the selective application of magic to *typed feature grammars* a type of constraint-logic grammar based on Typed Feature Logic ($\mathcal{TFL}$; Götz, 1995). Typed feature grammars can be used as the basis for implementations of Head-driven Phrase Structure Grammar (HPSG; Pollard and Sag, 1994) as discussed in (Götz and Meurers, 1997a) and (Meurers and Minnen, 1997). Typed feature grammar constraints that are inexpensive to resolve are dealt with using the top-down interpreter of the ConTroll grammar development system (Götz and Meurers, 1997b) which uses an advanced search function, an advanced selection function and incorporates a coroutining mechanism which supports delayed interpretation.

The proposed parser is related to the so-called *Lemma Table* deduction system (Johnson and Dörre, 1995) which allows the user to specify whether top-down sub-computations are to be tabled. In contrast to Johnson and Dörre's deduction system, though, the selective magic parsing approach combines top-down and bottom-up control strategies. As such it resembles the parser of the grammar development system Attribute Language Engine (ALE) of (Carpenter and Penn, 1994). Unlike the ALE parser, though, the selective magic parser does not presuppose a phrase structure backbone and is more flexible as to which sub-computations are tabled/filtered.

2 Bottom-up Interpretation of Magic-compiled Typed Feature Grammars

We describe typed feature grammars and discuss their use in implementing HPSG grammars. Subsequently we present magic compilation of typed fea-

*The presented research was carried out at the University of Tübingen, Germany, as part of the Sonderforschungsbereich 340.

[1] A more detailed discussion of various aspects of the proposed parser can be found in (Minnen, 1998).

ture grammars on the basis of an example and introduce a dynamic bottom-up interpreter that can be used for goal-directed interpretation of magic-compiled typed feature grammars.

2.1 Typed Feature Grammars

A typed feature grammar consists of a signature and a set of definite clauses over the constraint language of equations of $\mathcal{TFL}$ (Götz, 1995) terms (Höhfeld and Smolka, 1988) which we will refer to as $\mathcal{TFL}$ definite clauses. Equations over $\mathcal{TFL}$ terms can be solved using (graph) unification provided they are in normal form. (Götz, 1994) describes a normal form for $\mathcal{TFL}$ terms, where typed feature structures are interpreted as satisfiable normal form $\mathcal{TFL}$ terms.[2] The signature consists of a type hierarchy and a set of appropriateness conditions.

Example 1 The signature specified in figure 1 and 2 and the $\mathcal{TFL}$ definite clauses in figure 3 constitute an example of a typed feature grammar. We write $\mathcal{TFL}$ terms in normal form, i. e., as typed feature structures. In addition, uninformative feature specifications are ignored and typing is left implicit when immaterial to the example at hand. Equations between typed feature structures are removed by simple substitution or tags indicating structure sharing. Notice that we also use non-numerical tags such as $\boxed{\text{Xs}}$ and $\boxed{\text{XsYs}}$. In general all boxed items indicate structure sharing. For expository reasons we represent the ARGn features of the `append` relation as separate arguments.

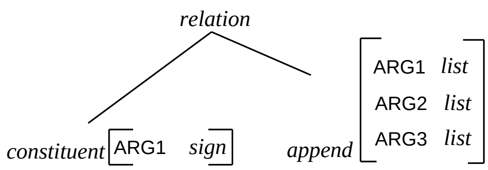

Figure 2: *Example of a typed feature grammar signature (part 2)*

Typed feature grammars can be used as the basis for implementations of Head-driven Phrase Structure Grammar (Pollard and Sag, 1994).[3] (Meurers and Minnen, 1997) propose a compilation of lexical rules into $\mathcal{TFL}$ definite clauses which are used to restrict lexical entries. (Götz and Meurers, 1997b) describe a method for compiling implicational constraints into typed feature grammars and interleaving them with relational constraints.[4] Because of space limitations we have to refrain from an example. The ConTroll grammar development system as described in (Götz and Meurers, 1997b) implements the above mentioned techniques for compiling an HPSG theory into typed feature grammars.

$$
\begin{array}{l}
(1)\ \text{constituent}(\begin{bmatrix}\text{CAT} & s\\ \text{PHON} & \boxed{1}\\ \text{SEM} & \boxed{5}\end{bmatrix})\text{:-}\\
\qquad \text{constituent}(\begin{bmatrix}\text{CAT} & np\\ \text{PHON} & \boxed{2}\\ \text{AGR} & \boxed{4}\\ \text{SEM} & \boxed{6}\end{bmatrix}),\\
\qquad \text{constituent}(\begin{bmatrix}\text{CAT} & v\\ \text{PHON} & \boxed{3}\\ \text{AGR} & \boxed{4}\\ \text{SEM} & \boxed{5}\begin{bmatrix}\text{SUBJ} & \boxed{6}\end{bmatrix}\end{bmatrix}),\\
\qquad \text{append}(\boxed{2},\boxed{3},\boxed{1}).\\
(2)\ \text{constituent}(\begin{bmatrix}\text{CAT} & np\\ \text{PHON} & \langle\ mary\ \rangle\\ \text{AGR} & third\text{-}sing\\ \text{SEM} & mary\_lf\end{bmatrix}).\\
(3)\ \text{constituent}(\begin{bmatrix}\text{CAT} & v\\ \text{PHON} & \langle sleeps\rangle\\ \text{AGR} & third\text{-}sing\\ \text{SEM} & sleep\end{bmatrix}).\\
(4)\ \text{append}(\langle\rangle, \boxed{\text{Ys}}, \boxed{\text{Ys}}).\\
(5)\ \text{append}(\langle\boxed{\text{X}} \mid \boxed{\text{Xs}}\rangle, \boxed{\text{Ys}}, \langle\boxed{\text{X}} \mid \boxed{\text{XsYs}}\rangle)\text{:-}\\
\qquad \text{append}(\boxed{\text{Xs}}, \boxed{\text{Ys}}, \boxed{\text{XsYs}}).
\end{array}
$$

Figure 3: *Example of a set of $\mathcal{TFL}$ definite clauses*

2.2 Magic Compilation

Magic is a compilation technique for goal-directed bottom-up processing of logic programs. See, among others, (Ramakrishnan et al. 1992). Because magic compilation does not refer to the specific constraint language adopted, its application is not limited to logic programs/grammars: It can be applied to relational extensions of other constraint languages such as typed feature grammars without further adaptions.

Due to space limitations we discuss magic compilation by example only. The interested reader is referred to (Nilsson and Małuszynski, 1995) for an introduction.

Example 2 We illustrate magic compilation of typed feature grammars with respect to definite clause 1 in figure 3. Consider the $\mathcal{TFL}$ definite

[2]This view of typed feature structures differs from the perspective on typed feature structures as modeling partial information as in (Carpenter, 1992). Typed feature structures as normal form $\mathcal{TFL}$ terms are merely syntactic objects.

[3]See (King, 1994) for a discussion of the appropriateness of $\mathcal{TFL}$ for HPSG and a comparison with other feature logic approaches designed for HPSG.

[4](Götz, 1995) proves that this compilation method is sound in the general case and defines the large class of type constraints for which it is complete.

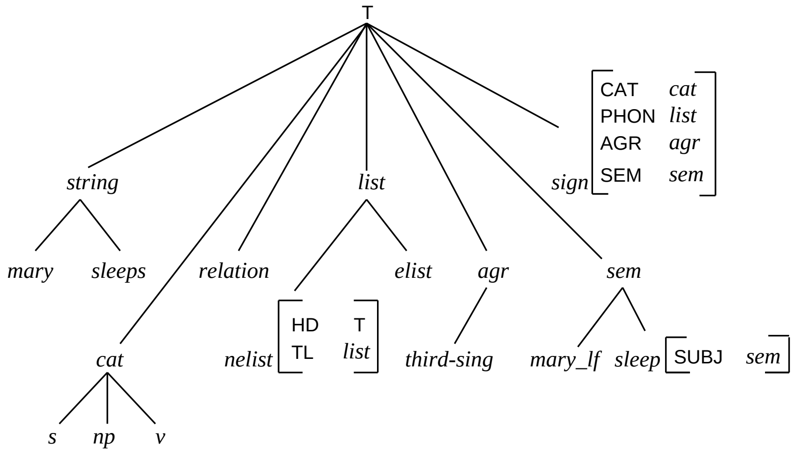

Figure 1: *Example of a typed feature grammar signature (part 1)*

clause in figure 4. As a result of magic compilation

$$
\begin{array}{l}
\text{constituent}(\boxed{\text{magic}}\begin{bmatrix}\text{CAT} & s\\ \text{PHON} & \boxed{1}\\ \text{SEM} & \boxed{5}\end{bmatrix}):\text{-}\\
\quad \text{magic\_constituent}(\boxed{\text{magic}}),\\
\quad \text{constituent}(\begin{bmatrix}\text{CAT} & np\\ \text{PHON} & \boxed{2}\\ \text{AGR} & \boxed{4}\\ \text{SEM} & \boxed{6}\end{bmatrix}),\\
\quad \text{constituent}(\begin{bmatrix}\text{CAT} & v\\ \text{PHON} & \boxed{3}\\ \text{AGR} & \boxed{4}\\ \text{SEM} & \boxed{5}\begin{bmatrix}\text{SUBJ} & \boxed{6}\end{bmatrix}\end{bmatrix}),\\
\quad \text{append}(\boxed{2},\boxed{3},\boxed{1}).
\end{array}
$$

Figure 4: *Magic variant of definite clause 1 in figure 3*

a magic literal is added to the right-hand side of the original definite clause. Intuitively understood, this magic literal "guards" the application of the definite clause. The clause is applied only when there exists a fact that unifies with this magic literal.[5] The resulting definite clause is also referred to as the *magic variant* of the original definite clause.

The definite clause in figure 5 is the so-called *seed* which is used to make the bindings as provided by the initial goal available for bottom-up processing. In this case the seed corresponds to the initial goal of parsing the string 'mary sleeps'. Intuitively understood, the seed makes available the bindings of the initial goal to the magic variants of the definite clauses defining a particular initial goal; in this case the magic variant of the definite clause defining a constituent of category 's'. Only when their magic literal unifies with the seed are these clauses applied.[6]

$$
\text{magic\_constituent}(\begin{bmatrix}\text{CAT} & s\\ \text{PHON} & \langle mary, sleeps\rangle\\ \text{SEM} & sem\end{bmatrix}).
$$

Figure 5: *Seed corresponding to the initial goal of parsing the string 'mary sleeps'*

The so-called *magic rules* in figure 6 are derived in order to be able to use the bindings provided by the seed to derive new facts that provide the bindings which allow for a goal-directed application of the definite clauses in the grammar not directly defining the initial goal. Definite clause 3, for example, can be used to derive a `magic_append` fact which percolates the relevant bindings of the seed/initial goal to restrict the application of the magic variant of definite clauses 4 and 5 in figure 3 (which are not displayed).

2.3 Semi-naive Bottom-up Interpretation

Magic-compiled logic programs/grammars can be interpreted in a bottom-up fashion without losing any of the goal-directedness normally associated with top-down interpretation using a so-called *semi-naive bottom-up* interpreter: A dynamic interpreter that tables only complete intermediate results, i. e., facts or passive edges, and uses an agenda to avoid redundant sub-computations. The Prolog predicates

[5]A fact can be a unit clause, i. e., a $\mathcal{TFL}$ definite clause without right-hand side literals, from the grammar or derived using the rules in the grammar. In the latter case one also speaks of a passive edge.

[6]The creation of the seed can be postponed until run time, such that the grammar does not need to be compiled for every possible initial goal.

$$\text{(1) magic\_constituent(}\begin{bmatrix}\text{CAT} & \textit{np}\\ \text{PHON} & \textit{list}\\ \text{AGR} & \textit{agr}\\ \text{SEM} & \textit{sem}\end{bmatrix}\text{):-}$$

$$\text{magic\_constituent(}\begin{bmatrix}\text{CAT} & \textit{s}\\ \text{PHON} & \textit{list}\\ \text{SEM} & \textit{sem}\end{bmatrix}\text{).}$$

$$\text{(2) magic\_constituent(}\begin{bmatrix}\text{CAT} & \textit{v}\\ \text{PHON} & \textit{list}\\ \text{AGR} & \boxed{4}\\ \text{SEM} & \boxed{5}\begin{bmatrix}\text{SUBJ} & \boxed{6}\end{bmatrix}\end{bmatrix}\text{):-}$$

$$\text{magic\_constituent(}\begin{bmatrix}\text{CAT} & \textit{s}\\ \text{PHON} & \textit{list}\\ \text{SEM} & \boxed{5}\end{bmatrix}\text{),}$$

$$\text{constituent(}\begin{bmatrix}\text{CAT} & \textit{np}\\ \text{PHON} & \textit{list}\\ \text{AGR} & \boxed{4}\\ \text{SEM} & \boxed{6}\end{bmatrix}\text{).}$$

$$\text{(3) magic\_append(}\boxed{2}\text{,}\boxed{3}\text{,}\boxed{1}\text{):-}$$

$$\text{magic\_constituent(}\begin{bmatrix}\text{CAT} & \textit{s}\\ \text{PHON} & \boxed{1}\\ \text{SEM} & \boxed{5}\end{bmatrix}\text{),}$$

$$\text{constituent(}\begin{bmatrix}\text{CAT} & \textit{np}\\ \text{PHON} & \boxed{2}\\ \text{AGR} & \boxed{4}\\ \text{SEM} & \boxed{6}\end{bmatrix}\text{),}$$

$$\text{constituent(}\begin{bmatrix}\text{CAT} & \textit{v}\\ \text{PHON} & \boxed{3}\\ \text{AGR} & \boxed{4}\\ \text{SEM} & \boxed{5}\begin{bmatrix}\text{SUBJ} & \boxed{6}\end{bmatrix}\end{bmatrix}\text{).}$$

Figure 6: *Magic rules resulting from applying magic compilation to definite clause 1 in figure 3*

in figure 7 implement a semi-naive bottom-up interpreter.[7] In this interpreter both the table and the agenda are represented using lists.[8] The agenda keeps track of the facts that have not yet been used to update the table. It is important to notice that in order to use the interpreter for typed feature grammars it has to be adapted to perform graph unification.[9] We refrain from making the necessary adaptions to the code for expository reasons.

The table is initialized with the facts from the grammar. Facts are combined using a operation called *match*. The match operation unifies all but one of the right-hand side literals of a definite clause in the grammar with facts in the table. The remaining right-hand side literal is unified with a newly derived fact, i. e., a fact from the agenda. By doing this, repeated derivation of facts from the same earlier derived facts is avoided.

```
semi_naive_interpret(Goal):-
    initialization(Agenda,Table0),
    update_table(Agenda,Table0,Table),
    member(edge(Goal,[]),Table).
update_table([],Table,Table).
update_table([Edge|Agenda0],Table0,Table):-
    update_table_w_edge(Edge,Edges,
                         Table0,Table1),
    append(Edges,Agenda0,Agenda),
    update_table(Agenda,Table1,Table).
update_table_w_edge(Edge,Edges,Table0,Table):-
    findall( NewEdge,
             match(Edge,NewEdge,Table0),
             Edges),
    store(Edges,Table0,Table).
store([],Table,Table):-
store([Edge|Edges],Table0,Table):-
   member(GenEdge,Table0),
   \+ subsumes(GenEdge,Edge),
   store(Edges,[Edge|Table0],Table).
store([_|Edges],Table0,Table):-
   store(Edges,Table0,Table).
initialization(Edges,Edges):-
    findall( edge(Head,[]),
             definite_clause((Head:- [])),
             Edges).
completion(Edge,edge(Goal,[]),Table):-
    definite_clause((Goal :- Body)),
    Edge = edge(F,[]),
    select(F,Body,R),
    edges(R,Table).
edges([],_).
edges([Lit|Lits],Table):-
    member(edge(Lit,[]),Table),
    edges(Lits,Table).
```

Figure 7: *Semi-naive bottom-up interpreter*

3 Selective Magic HPSG Parsing

In case of large grammars the huge space requirements of dynamic processing often nullify the benefit of tabling intermediate results. By combining control strategies and allowing the user to specify how to process particular constraints in the grammar the selective magic parser avoids this problem. This solution is based on the observation that there are sub-computations that are relatively cheap and as a result do not need tabling (Johnson and Dörre, 1995; van Noord, 1997).

[7] Definite clauses serving as data are encoded using the predicate `definite_clause/1`: `definite_clause((Lhs :- Rhs)).`, where `Rhs` is a (possibly empty) list of literals.

[8] There are various other—more efficient—ways to implement a dynamic control strategy in Prolog. See, for example, (Shieber et al., 1995).

[9] A term encoding of typed feature structures would enable the use of term unification instead. See, for example, (Gerdemann, 1995).

3.1 Parse Type Specification

Combining control strategies depends on a way to differentiate between types of constraints. For example, the ALE parser (Carpenter and Penn, 1994) presupposes a phrase structure backbone which can be used to determine whether a constraint is to be interpreted bottom-up or top-down. In the case of selective magic parsing we use so-called *parse types* which allow the user to specify how constraints in the grammar are to be interpreted. A literal (goal) is considered a *parse type literal (goal)* if it has as its single argument a typed feature structure of a type specified as a parse type.[10]

All types in the type hierarchy can be used as parse types. This way parse type specification supports a flexible filtering component which allows us to experiment with the role of filtering. However, in the remainder we will concentrate on a specific class of parse types: We assume the specification of type *sign* and its sub-types as parse types.[11] This choice is based on the observation that the constraints on type *sign* and its sub-types play an important guiding role in the parsing process and are best interpreted bottom-up given the lexical orientation of HPSG. The parsing process corresponding to such a parse type specification is represented schematically in figure 8. Starting from the lexical entries,

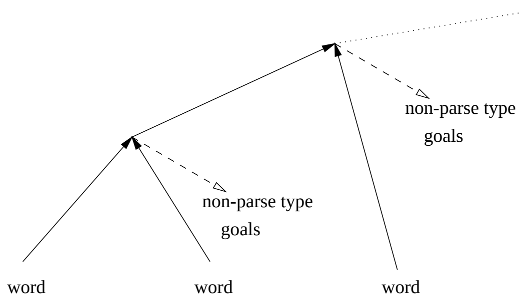

Figure 8: *Schematic representation of the selective magic parsing process*

i. e., the $\mathcal{TFL}$ definite clauses that specify the word objects in the grammar, phrases are built bottom-up by matching the parse type literals of the definite clauses in the grammar against the edges in the table. The non-parse type literals are processed according to the top-down control strategy described in section 3.3.

3.2 Selective Magic Compilation

In order to process parse type goals according to a semi-naive magic control strategy, we apply magic compilation selectively. Only the $\mathcal{TFL}$ definite clauses in a typed feature grammar which define parse type goals are subject to magic compilation. The compilation applied to these clauses is identical to the magic compilation illustrated in section 2.1 except that we derive magic rules only for the right-hand side literals in a clause which are of a parse type. The definite clauses in the grammar defining non-parse type goals are not compiled as they will be processed using the top-down interpreter described in the next section.

3.3 Advanced Top-down Control

Non-parse type goals are interpreted using the standard interpreter of the ConTroll grammar development system (Götz and Meurers, 1997b) as developed and implemented by Thilo Götz. This advanced top-down interpreter uses a search function that allows the user to specify the information on which the definite clauses in the grammar are indexed. An important advantage of deep multiple indexing is that the linguist does not have to take into account of processing criteria with respect to the organization of her/his data as is the case with a standard Prolog search function which indexes on the functor of the first argument.

Another important feature of the top-down interpreter is its use of a selection function that interprets deterministic goals, i. e., goals which unify with the left-hand side literal of exactly one definite clause in the grammar, prior to non-deterministic goals. This is often referred to as incorporating *deterministic closure* (Dörre, 1993). Deterministic closure accomplishes a reduction of the number of choice points that need to be set during processing to a minimum. Furthermore, it leads to earlier failure detection.

Finally, the used top-down interpreter implements a powerful coroutining mechanism:[12] At run time the processing of a goal is postponed in case it is insufficiently instantiated. Whether or not a goal is sufficiently instantiated is determined on the basis of so-called *delay patterns*.[13] These are specifications

[10] The notion of a parse type literal is closely related to that of a *memo literal* as in (Johnson and Dörre, 1995).

[11] When a type is specified as a parse type, all its sub-types are considered as parse types as well. This is necessary as otherwise there may exist magic variants of definite clauses defining a parse type goal for which no magic facts can be derived which means that the magic literal of these clauses can be interpreted neither top-down nor bottom-up.

[12] Coroutining appears under many different guises, like for example, *suspension*, *residuation*, *(goal) freezing*, and *blocking*. See also (Colmerauer, 1982; Naish, 1986).

[13] In the literature delay patterns are sometimes also referred to as *wait declarations* or *block statements*.

provided by the user that indicate which restricting information has to be available before a goal is processed.

3.4 Adapted Semi-naive Bottom-up Interpretation

The definite clauses resulting from selective magic transformation are interpreted using a semi-naive bottom-up interpreter that is adapted in two respects. It ensures that non-parse type goals are interpreted using the advanced top-down interpreter, and it allows non-parse type goals that remain delayed locally to be passed in and out of sub-computations in a similar fashion as proposed by (Johnson and Dörre, 1995). In order to accommodate these changes the adapted semi-naive interpreter enables the use of edges which specify delayed goals.

Figure 9 illustrates the adapted match operation. The first defining clause of `match/3` passes delayed

```
match(Edge,edge(Goal,Delayed),Table):-
   definite_clause((Goal :- Body)),
   select(Lit,Body,Lits),
   parse_type(Lit),
   Edge = edge(Lit,Delayed0),
   edges(Lit,Table,Delayed0,TopDown),
   advanced_td_interpret(TopDown,Delayed).
match(Edge,edge(Goal,Delayed),Table):-
   definite_clause((Goal :- TopDown)),
   advanced_td_interpret(TopDown,Delayed).
```

Figure 9: *Adapted definition of* `match/3`

and non-parse type goals of the definite clause under consideration to the advanced top-down interpreter via the call to `advanced_td_interpret/2` as the list of goals `TopDown`.[14] The second defining clause of `match/3` is added to ensure all right-hand side literals are directly passed to the advanced top-down interpreter if none of them are of a parse type.

Allowing edges which specify delayed goals necessitates the adaption of the definition of `edges/3`. When a parse type literal is matched against an edge in the table, the delayed goals specified by that edge need to be passed to the top-down interpreter. Consider the definition of the predicate `edges` in figure 11. The third argument of the definition of `edges/4` is used to collect delayed goals. When there are no more parse type literals in the right-hand side of the definite clause under consideration, the second

```
edges([Lit|Lits],Table,Delayed0,TopDown):-
   parse_type(Lit),
   member(edge(Lit,Delayed1),Table),
   append(Delayed0,Delayed1,Delayed).
   edges(Lit,Table,Delayed,TopDown).
edges([],_,Delayed,TopDown):-
   append(Delayed,Lit,TopDown).
```

Figure 11: *Adapted definition of* `edges/4`

defining clause of `edges/4` appends the collected delayed goals to the remaining non-parse type literals. Subsequently, the resulting list of literals is passed up again for advanced top-down interpretation.

4 Implementation

The described parser was implemented as part of the ConTroll grammar development system (Götz and Meurers, 1997b). Figure 10 shows the overall setup of the ConTroll magic component. The Controll magic component presupposes a parse type specification and a set of delay patterns to determine when non-parse type constraints are to be interpreted. At run-time the goal-directedness of the selective magic parser is further increased by means of using the phonology of the natural language expression to be parsed as specified by the initial goal to restrict the number of facts that are added to the table during initialization. Only those facts in the grammar corresponding to lexical entries that have a value for their phonology feature that appears as part of the input string are used to initialize the table.

The ConTroll magic component was tested with a larger (> 5000 lines) HPSG grammar of a sizeable fragment of German. This grammar provides an analysis for simple and complex verb-second, verb-first and verb-last sentences with scrambling in the mittelfeld, extraposition phenomena, wh-movement and topicalization, integrated verb-first parentheticals, and an interface to an illocution theory, as well as the three kinds of infinitive constructions, nominal phrases, and adverbials (Hinrichs et al., 1997).

As the test grammar combines sub-strings in a non-concatenative fashion, a preprocessor is used that chunks the input string into linearization domains. This way the standard ConTroll interpreter (as described in section 3.3) achieves parsing times of around 1-5 seconds for 5 word sentences and 10–60 seconds for 12 word sentences.[15] The use of magic compilation on all grammar constraints, i.e.,

[14] The definition of `match/3` assumes that there exists a strict ordering of the right-hand side literals in the definite clauses in the grammar, i. e., parse type literals always preced e non-parse type literals.

[15] Parsing with such a grammar is difficult in any system as it does neither have nor allow the extraction of a phrase structure backbone.

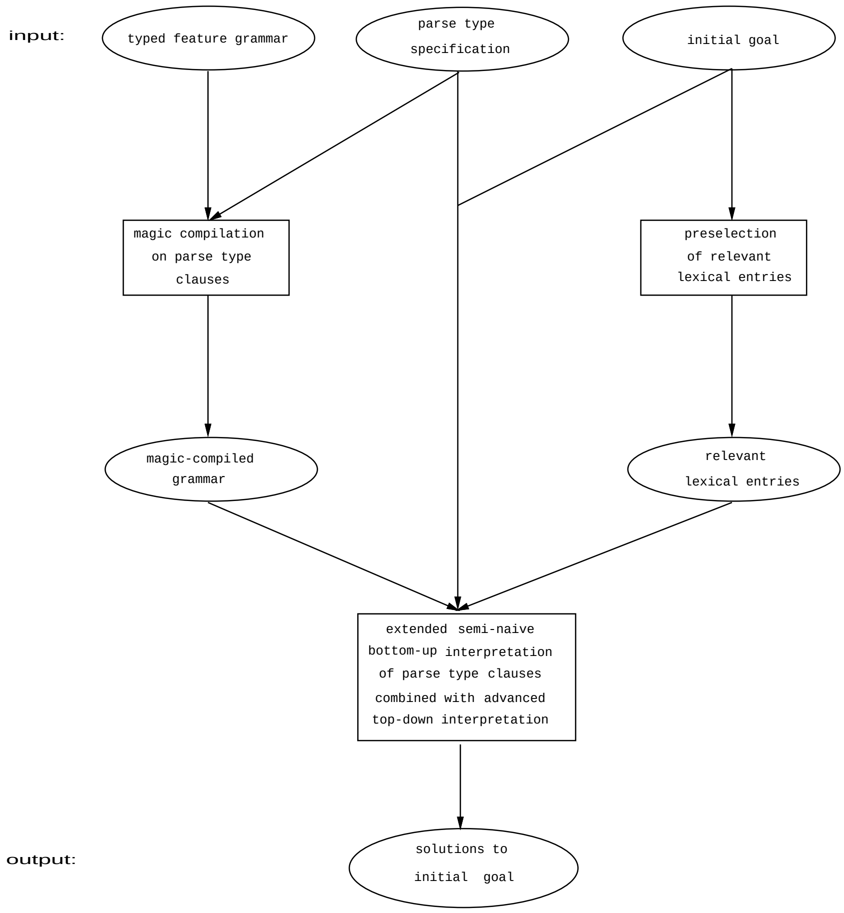

Figure 10: *Setup of the ConTroll magic component*

tabling of all sub-computations, leads to an vast increase of parsing times. The selective magic HPSG parser, however, exhibits a significant speedup in many cases. For example, parsing with the module of the grammar implementing the analysis of nominal phrases is up to nine times faster. At the same time though selective magic HPSG parsing is sometimes significantly slower. For example, parsing of particular sentences exhibiting adverbial subordinate clauses and long extraction is sometimes more than nine times slower. We conjecture that these ambiguous results are due to the use of coroutining: As the test grammar was implemented using the standard ConTroll interpreter, the delay patterns used presuppose a data-flow corresponding to advanced top-down control and are not fine-tuned with respect to the data-flow corresponding to the selective magic parser.

Coroutining is a flexible and powerful facility used in many grammar development systems and it will probably remain indispensable in dealing with many control problems despite its various disadvantages.[16] The test results discussed above indicate that the comparison of parsing strategies can be seriously hampered by fine-tuning parsing using delay patterns. We believe therefore that further research into the systematics underlying coroutining would be desirable.

5 Concluding Remarks

We described a selective magic parser for typed feature grammars implementing HPSG that combines the advantages of dynamic bottom-up and advanced top-down control. As a result the parser avoids the efficiency problems resulting from the huge space requirements of storing intermediate results in parsing

[16] Coroutining has a significant run-time overhead caused by the necessity to check the instantiation status of a literal/goal. In addition, it demands the procedural annotation of an otherwise declarative grammar. Finally, coroutining presupposes that a grammar writer possesses substantial processing expertise.

with large grammars. The parser allows the user to apply magic compilation to specific constraints in a grammar which as a result can be processed dynamically in a bottom-up and goal-directed fashion. State of the art top-down processing techniques are used to deal with the remaining constraints. We discussed various aspects concerning the implementation of the parser which was developed as part of the grammar development system ConTroll.

Acknowledgments

The author gratefully acknowledges the support of the SFB 340 project B4 "From Constraints to Rules: Efficient Compilation of HPSG" funded by the German Science Foundation, and the project "PSET: Practical Simplification of English Text", a three-year project funded by the UK Engineering and Physical Sciences Research Council (GR/L53175), and Apple Computer Inc.. The author wishes to thank Dale Gerdemann and Erhard Hinrichs and the anonymous reviewers for comments and discussion. Of course, the author is responsible for all remaining errors.

References

Bob Carpenter and Gerald Penn. 1994. ALE – The Attribute Logic Engine, User's guide, version 2.0.2. Technical report, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA.

Bob Carpenter. 1992. *The Logic of Typed Feature Structures - With Applications to Unification Grammars, Logic Programs and Constraint Resolution.* Cambridge University Press, New York, USA.

Alain Colmerauer. 1982. PrologII: Manuel de référence et modèle théorique. Technical report, Groupe d'Intelligence Artificielle, Faculté de Sciences de Luminy, Marseille, France.

Jochen Dörre. 1993. Generalizing Earley Deduction for Constraint-based Grammars. In Jochen Dörre and Michael Dorna (eds.), 1993. *Computational Aspects of Constraint-Based Linguistic Description I.* DYANA-2, Deliverable R1.2.A.

Dale Gerdemann. 1995. Term Encoding of Typed Feature Structures. In *Proceedings of the Fourth International Workshop on Parsing Technologies*, Prague, Czech Republic.

Thilo Götz and Detmar Meurers. 1997a. Interleaving Universal Principles and Relational Constraints over Typed Feature Logic. In *ACL/EACL Proceedings*, Madrid, Spain.

Thilo Götz and Detmar Meurers. 1997b. The ConTroll System as Large Grammar Development Platform. In *Proceedings of the ACL Workshop on Computational Environments for Grammar Development and Linguistic Engineering*, Madrid, Spain.

Thilo Götz. 1994. A Normal Form for Typed Feature Structures. Technical report SFB 340 nr. 40, University of Tübingen, Germany.

Thilo Götz. 1995. Compiling HPSG Constraint Grammars into Logic Programs. In *Proceedings of the Workshop on Computational Logic for Natural Language Processing*, Edinburgh, UK.

Erhard Hinrichs, Detmar Meurers, Frank Richter, Manfred Sailer, and Heike Winhart. 1997. Ein HPSG-fragment des Deutschen, Teil 1: Theorie. Technical report SFB 340 95, University of Tübingen, Germany.

Markus Höhfeld and Gert Smolka. 1988. Definite Relations over Constraint Languages. Technical Report 53, IBM, Germany.

Mark Johnson and Jochen Dörre. 1995. Memoization of Coroutined Constraints. In *ACL Proceedings*, Cambridge, Massachusetts, USA.

Paul King. 1994. Typed Feature Structures as Descriptions. In *Proceedings of of the 15th Conference on Computational Linguistics*, Kyoto, Japan.

Detmar Meurers and Guido Minnen. 1997. A Computational Treatment of Lexical Rules in HPSG as Covariation in Lexical Entries. *Computational Linguistics*, 23(4).

Guido Minnen. 1996. Magic for Filter Optimization in Dynamic Bottom-up Processing. In *ACL Proceedings*, Santa Cruz, California, USA.

Guido Minnen. 1998. *Off-line Compilation for Efficient Processing with Constraint-logic Grammars.* Ph.D. thesis, University of Tübingen, Germany. Technical report SFB 340 nr. 130.

Lee Naish. 1986. *Negation and Control in Prolog.* Springer-Verlag, Berlin, Germany.

Ulf Nilsson and Jan Małuszynski. 1995. *Logic, Programming and Prolog.* John Wiley & Sons, Chichester, UK, 2nd edition.

Carl Pollard and Ivan Sag. 1994. *Head-Driven Phrase Structure Grammar.* University of Chicago Press, Chicago, Illinois, USA.

Raghu Ramakrishnan, Divesh Srivastava, and S. Sudarshan. 1992. Efficient Bottom-up Evaluation of Logic Programs. In Joos Vandewalle (ed.), 1992. *The State of the Art in Computer Systems and Software Engineering.* Kluwer Academic Publishers.

Stuart Shieber, Yves Schabes, and Fernando Pereira. 1995. Principles and Implementation of Deductive Parsing. *Journal of Logic Programming*, 24(1-2).

Gertjan van Noord. 1997. An Efficient Implementation of the Head-corner Parser. *Computational Linguistics*, 23(3).