Efficient Processing with Constraint-Logic Grammars Using Grammar Compilation

Guido Minnen
(Motorola Labs)

Stanford: CSLI Publications (Stanford monographs in linguistics), 2001, viii+255 pp; distributed by University of Chicago Press; hardbound, ISBN 1-57586-305-7, $55.00, £35.00; paperbound, ISBN 0-57586-306-5, $20.00, £13.00

Reviewed by
Suresh Manandhar
University of York

1. Book Content

This monograph should be of interest to researchers working on building practical and efficient methods for processing highly abstract declarative constraint-based grammars, primarily head-driven phrase structure grammar (HPSG) (Pollard and Sag 1994). The work should also be of interest to researchers in the logic-programming community. The monograph is accessible to anyone with a background in logic programming. Background in grammar formalisms or HPSG is not essential to follow the monograph. Minnen has done a commendable job in making the monograph relatively easy to follow by using numerous well-explained examples. A number of typographic errors in the example programs, however, and a crucial missing figure (Figure 4.17) make the reading somewhat more difficult than it need have been.

Declarative constraint-based grammars are notorious for being highly inefficient from a processing point of view. Minnen's techniques, several of which are explored in the monograph, automatically transform the input grammar into a more specialized grammar that efficiently realizes the user's goal. The techniques can be viewed as performing equivalence transformations on the input logic program to derive a logic program that is more efficient with respect to the input goal. These transformations range from simple strategies such as *literal rearrangement* to more complex ones such as *building recursion reversal* and *magic templates*. Although most of these techniques are closely related to work by other researchers (Strzalkowski 1994, Ramakrishnan 1991), Minnen has extended them to make them suitable for dealing with feature-based grammars.

The monograph is divided into three parts: top-down control, bottom-up control, and lexical rules. Whereas the top-down and bottom-up control chapters primarily deal with grammar rule compilation, the lexical-rules chapter deals with methods for processing lexical rules.

Central to the extraction of control information are two notions:

- *the adornment of a literal*, which identifies those arguments in a literal that are bound.
- *the degree of nondeterminism* (DoN) of a literal, which is the number of alternatives or choice points available when evaluating the literal.

Given a user-specified goal and a user-specified global DoN, Minnen's grammar compilation procedures attempt to transform the grammar nondeterministically into an equivalent grammar until the transformed grammar meets the global DoN requirement.

Minnen's method can be understood generally in terms of two mutually recursive methods:

- Adornment of a goal is used to perform a static abstract interpretation of the program to determine the DoN of each literal.
- The information gained from abstract interpretation is employed to perform a program transformation.

Minnen couples program transformation with relatively sophisticated tabulation techniques to store partial solutions and hence minimize the cost of processing recursive goals with shared subgoals. His tabulation method stores failed goals, successful goals, and currently "opened" goals.

Chapter 3 describes literal rearrangement, in which the literals of a clause are rearranged to make the goal more deterministic. Literal rearrangement employs adornment information specified in the user goal to determine the best literal rearrangement to achieve the specified DoN. Literal rearrangement, although fairly simple in principle, requires a recursive search through the clauses of each literal and hence can be costly. Tabulation becomes essential here. In addition, Minnen describes a local heuristic that chooses the literal with the most arguments (or feature paths, in the case of constraint-based grammars) instantiated to minimize the cost of this search. A simple iterative deepening search over the literal rearrangement procedure starting with DoN = 1 finds the literal rearrangement program transformation with the smallest possible DoN.

When information from two literals is simultaneously required to successfully constrain the processing task, such as in certain treatments of German partial verb phrase topicalization (Example (1)), then literal rearrangement alone is insufficient. In fact, in such cases, Minnen states that *coroutining* or parallel processing of literals is required. He then goes on to show that coroutining can be simulated by *unfolding* the literals and applying literal transformation to the resultant clause.

(1) [Anna lieben]$_i$ wird $\_\__i$ Karl.

The second problem that literal rearrangement cannot solve is left recursion, since in a grammar such as the one shown in Figure 1, described by Minnen (p. 72), it is not the literal order that is problematic but that the base case needs to be processed as early as possible. Chapter 4 explores techniques for automatically detecting when such a *building recursion reversal* (BRR) transformation can be applied and describes methods for implementing it correctly. This type of transformation is more involved than the simpler literal rearrangement transformation of the previous chapter, as it involves analyzing argument instantiation patterns called *argument sequences*. For example, in the program in Figure 1, one possible argument sequence (if we trace the second argument of `vp/4`) is the sequence `<Comps,[Comp|Comps]>` (given in simplified form here). Roughly speaking, a building recursion reversal changes the order of argument sequences from a sequence such as `<Comps,[Comp|Comps]>`, which builds structure, to the reversed sequence `<[Comp|Comps],Comps>`, which consumes structure.

Part 2 covers bottom-up processing using *magic transformations*, which transform the original program by adding an additional literal, known as a *magic rule*, to the

```
vp(Subj, Comps, VSem, P0, P):-
       vp(Subj, [Comp|Comps], VSem, P0, P1),
       np(Comp, P1, P0).

vp(Subj, Comps, VSem, P0, P):-
       v(Subj, Comps, VSem, P0, P).

v(Subj, [Obj,IObj], bring(Subj,IObj,Obj),[bring|P],P).

np(john, [john|P], P).
np(flowers, [flowers|P], P).
np(mary, [mary|P], P).
```

Figure 1
Example program from Minnen (p. 86).

start of the right-hand side of each clause. In the example, magic transformation of the predicate vp/4 of the program shown in Figure 1 with respect to the goal shown in Figure 2 would result in the clauses shown in Figure 3. The magic rule `magic_vp/4` acts as a guard by instantiating variables and filters out subgoals that cannot be part of the main goal. In a naive bottom-up strategy, all facts are used to deduce the goal. Naive bottom-up processing is expensive in terms of both space and time. The magic rule provides a top-down filtering component and hence makes an otherwise purely data-driven control more goal driven. Minnen explores both the Earley deduction strategy and an improved seminaive bottom-up processing strategy and concludes that the two are very similar, with the semi-naive strategy being slightly better in terms of space requirements. He concedes, however, that to ensure termination, both a subsumption check and an abstraction function are necessary. Subsumption checks are computationally expensive, and abstraction functions have to be user specified, which is a big disadvantage.

The final part of the monograph deals with the treatment of HPSG lexical rules. This work builds upon work reported by Meurers and Minnen (1997). The nice part is that, in Minnen's setup, lexical rules can be viewed as definite clauses, so that techniques from the previous chapters directly apply to lexical rules. As in Meurers and Minnen (1997), lexical rule interaction (through which the output of a lexical rule can be the input of another lexical rule) is modeled by means of a finite-state automaton, computed off-line, that precompiles all the possible interactions between lexical rules. Minnen shows that nondeterminism in lexical rule expansion can be minimized by combining *partial unfolding* with lexical rule interaction. In this way, a large number of choice points that lead to failure can be eliminated at compile time.

2. Final Analysis

Minnen's monograph provides a refreshing entry point for someone wanting to pursue a research program in efficient implementation of constraint-based grammars. Minnen's work complements work on compilation techniques for typed-feature hi-

```
vp(Subj,Comps,bring(john,flowers,mary),P0,P1)
```

Figure 2
User goal.

```
vp(Subj, Comps, VSem, P0, P):-
       magic_vp(Subj,Comps,VSem, P0, P1),
       vp(Subj, [Comp|Comps], VSem, P0, P1),
       np(Comp, P1, P0).

vp(Subj, Comps, VSem, P0, P):-
       magic_vp(Subj,Comps,VSem, P0, P1),
       v(Subj, Comps, VSem, P0, P).

magic_vp(Subj,Comps,bring(john,flowers,mary), P0, P1).
```

Figure 3
Application of magic transformation to vp/4 from Figure 1.

erarchies (cf. Fall 1996, Wintner and Francez 1995). Coupling Minnen's techniques with compilation methods for typed-feature hierarchies should provide the necessary mechanisms for efficient implementation of large HPSG grammars.

A fair amount of work still remains: For an automated grammar-compilation system, it is essential that as much of the control information be extracted automatically as possible. Minnen's work, however, falls short of achieving this objective. His top-down processing strategy employing literal transformation and BRR transformation comes close to being a fully automated strategy, but I suspect that space requirements from tabulation becomes a factor, and hence (heuristic) techniques for making tabulation decisions need to be explored before the approach can be made practical. Somewhat surprisingly, Minnen does not explore the behavior of other variations on the basic top-down strategy, such as *deterministic closure*. It was not clear to me why the BRR transformation could not be used along with top-down control.

To ensure termination, Minnen's bottom-up control requires additional user-supplied control information in the form of parse types and delay patterns, which is not very desirable. Either automated generation of such control information or methods to eliminate it are needed. Although it is clear that Minnen's transformation techniques apply to typed-feature-structure grammars, there are hardly any examples of this in the monograph. Results of evaluation on a realistic grammar are only glossed over or missing, making it impossible to assess performance on large HPSG grammars such as the LinGo grammar (Copestake and Flickinger 2000).

References

Copestake, Ann and Dan Flickinger. 2000. An open-source grammar development environment and broad-coverage English grammar using HPSG. In *Proceedings of the Second Linguistic Resources and Evaluation Conference*, Athens, Greece, pages 591–600.

Fall, Andrew. 1996. *Reasoning with Taxonomies*. Ph.D. dissertation, Department of Computer Science, Simon Fraser University, July 1996.

Meurers, Detmar and Guido Minnen. 1997. A computational treatment of lexical rules in HPSG as covariation in lexical entries. *Computational Linguistics*, 23(4):543–568.

Pollard, Carl and Ivan Andrew Sag. 1994. *Head-Driven Phrase Structure Grammar*. Chicago: University of Chicago Press and Stanford: CSLI Publications.

Ramakrishnan, Raghu. 1991. Magic templates: A spellbinding approach to logic programs. *Journal of Logic Programming*, 11:189–216.

Strzalkowski, Tomek, editor. 1994. *Reversible Grammar in Natural Language Processing*. Kluwer Academic.

Wintner, Shuly and Nissim Francez. 1995. An abstract machine for typed feature structures. In *Proceedings of the Fifth Workshop on Natural Language Understanding and Logic Programming*, Lisbon, pages 205–220.

Suresh Manandhar is a lecturer in Computer Science at the University of York. He has worked on the implementation and formalization of constraint-based grammars. His published work includes papers on constraint logics and efficient compilation methods for constraint-based grammars, unsupervised learning of categorial grammars, learning of WordNet relations, and applications of inductive logic programming to natural language processing. Manandhar's address is: Department of Computer Science, University of York, Heslington YO1 5DD, York, U.K.; e-mail: suresh@cs.york.ac.uk.