An HPSG-based Parser for Automatic Knowledge Acquisition

Kentaro Torisawa† Jun'ichi Tsujii‡
†‡Department of Information Science
University of Tokyo
Hongo 7-3-1, Bunkyo-ku Tokyo, 113, Japan
{torisawa,tsujii}@is.s.u-tokyo.ac.jp

‡CCL, UMIST
PO BOX 88, Manchester M60 1QD U.K.
tsujii@ccl.umist.ac.uk

1 Introduction

Our aim is to build an HPSG [Pollard, 1993] based parser that can be used as a component of a knowledge acquisition(KA) system from unrestricted text[Horiguchi, 1995]. KA proceeds by using ***underspecified lexical entry templates*** given to each part of speech for words. By filling out the underspecified parts of them through unification, knowledge is acquired.

Our contention is that we cannot give an exhaustive set of specific *CFG skeletons* to the parser prior KA, in order to obtain a wide coverage required for handling corpora. In our parser, rules with *CFG skeletons*, which are widely used in HPSG implementations such as [Carpenter, 1994], are replaced with a few ***rule schemata*** and principles, whose examples are shown in Fig. 1 and 2. They do not specify particular syntactic categories and can cover most of the linguistic constructions in corpora by relying on lexicalization and augmentation with definite clause programs. However, this replacement prevents us from using optimization techniques for conventional unification-based parsers.

Our parser adopts a two-phased architecture. Phase 1 is a bottom-up parsing with compiled object-oriented code realizing only part of constraints in a full grammar. A full grammar is applied to completed parse trees in Phase 2.

Applicaton of rule schemata and their principles is monotone because of monotonicity of unification. (i.e. for any feature structures F_0,F_1, F'_0, F'_1, if $F_0 \sqsubseteq F_1$, $F'_0 \sqsubseteq F'_1$ $F_0 \sqcup F_1 \sqsubseteq F'_0 \sqcup F'_1$.). If a sign S subsumes a sign S' and the application of principles or rule schemata to S' succeeds, the application to S also succeeds and the results reserves their daughters' subsumption relation. Our basic idea is that we can systematically ***weaken*** lexical entries and other grammar components by eliminating certain constraints in them so that they are compiled to simple objects and cheaper procedures without losing the ability of a full grammar. Although the compiled ***grammar*** overgenerates, illegitimate signs are removed in Phase 2.

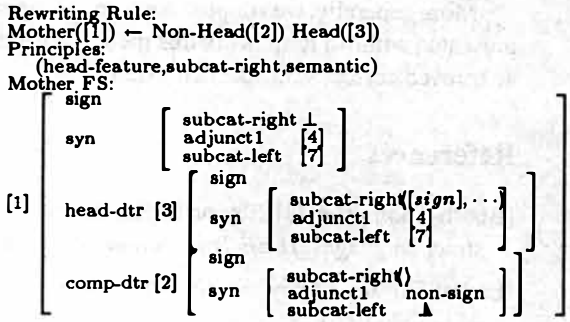

Figure 1: An example of a rule schema.

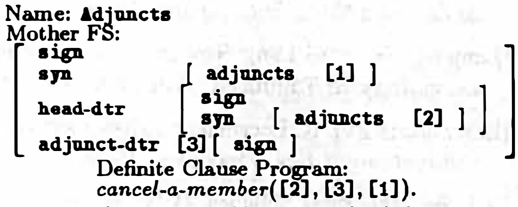

Figure 2: Adjuncts Principle

2 Compilation

Our compiler produces two items, ***Sign Objects***, which are objects corresponding to signs, and ***Rule Methods***, which play roles of rule schemata and principles. Both items are directly executed in Common Lisp Object System. Rule methods take sign objects representing daughter signs as input and produce sign objects corresponding to mothers. Sign objects have slots corresponding to only part of feature structures. This ***reduction*** is justified by monotonicity of unification.

Each slot of a sign object contains a fragment of the feature structure or other Lisp objects converted from the feature structure, such as symbols representing types. Fig. 3

```
(SLOT-NAME    SLOT-VALUE)
(SUBCAT-RIGHT ((E-LIST NIL #(FT @ #x921a82))))
(SUBCAT-LEFT      ((NIL      NON-OBLIGATORY-
SIGN #(FT @ #x921a9a))))
(ADJUNCTS2 (SIGN
        #(FT @ #x92170a)
        ((ADJUNCT1 NON-SIGN :SINGLETON))
        (SELECTING-FEATURE-SHARING
              SUBCAT-LEFT
              SUBCAT-LEFT)))
```

Figure 3: A compiled lexical entry

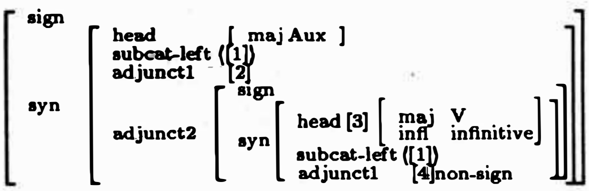

Figure 4: An original lexical entry

shows the sign object compiled from the lexical entry in Fig. 4. In this sign object, the **ADJUNCTS2** slot contains the type **sign** and a head-feature **#(FT @ #x92170a)**, which represents the feature structure denoted by the tag **[3]** in Fig. 4. The third element represents the constraint that the **ADJUNCTS2** value of a selected sign must be **non-sign**. This also corresponds to the feature structure tagged as **[4]** in Fig. 4. The fourth element is a command to transfer the **subcat-left** value of a selected sign to the mother's same slot. This transfer is represented by the structure sharing **[1]** in the original lexical entry.

A rule method contains only part of the constraints realized in principles and rule schemata. This reduction of constraints corresponds to the elimination of feature structures, structure sharings and part of a definite clause program in a rule method or its principles. The soundness of this reduction can be proven by monotonicity of unification. Furthermore, some unification evoked by structure sharing is replaced by simple assignements of slot values. For example, most feature *raising* is performed by assignments. This replacement does not affect the soundness of our compilation because any two feature structures always subsume their unified one. If a sign object is created by the code containing assignement instead of unification, it subsumes the sign object created as the result of unification a unification. Thus, a sign object with rule methods always subsume the sign to be created by the original grammar.

Rule schemata with their principles are categorized as 1) rule schemata to *use* selecting features, such as **SUBCAT**, which are feature structures to be unified with another sign. and 2) rule schemata to transfer selecting features such as a rule schema augmented with a trace principle. The first category is divided further into three according to the type of the selecting features (singleton, list or set)in a rule schema.

The compiler generates rule methods by filling out a template which is prepared for each type of rule schemata with references to a rule schema and its principles. The differences among code templates reflect the differences of the definite clause programs to be evoked in application of each type of rule schemata. For example, **Cancel-a-member** in Fig. 2 is the program for *using* selecting features of set type. The *behaviour* of such important parts of the definite clause programs are reflected directly in the templates. The following is the template for a rule schema for the list type selecting features.

```
(lambda (selector selectee)
  (if (and (selecting-feature-unifiable?
             (<selecting-feature>
                           selector)
             selectee)
           <other-unifiability-checking>)
      (let ((mother-sign (create-sign)))
        <feature-raising>
        <evaluate-structure-sharing-commands>
        mother-sign)
    nil))
```

The rule method takes daughter sign objects, which are bound to the variables **selector** and **selectee** in the argument list, and produces their mother sign object bound to the variable **mother-sign**.

3 Conclusion

For the rule schemata presented in Fig. 1, the compiled code is about 43 times as fast as the application of the rule schemata and its principles. For a 25 word sentence, bottom-up parsing with the compiled code followed by the applications of the original grammar to the completed parse trees was 3.1 times as fast as the parsing with only the original grammar. The required storage was 370% less than that of the original.

Linguistically well-defined grammar formalisms such as HPSG have been regarded as inappropriate for dealing with unrestricted real-world text. In order to build feasible systems, researchers have relied on more procedural grammar whose well-defined-ness is difficult to show. However, by using our compilation technique, we will be able to develop a robust and efficient HPSG-based parser which can be a component of a practical system.

References

[Carpenter, 1994] Carpenter, B. and Penn, G. (1994). *The Attibute Logic Engine User's Guide*. Carnegie Mellon University.

[Horiguchi, 1995] Horiguchi, K., Torisawa, K., and Tsujii, J. (1995). Automatic acquisition of content words using an HPSG-based parser. to be submitted.

[Pollard, 1993] Pollard, C. and Sag, I. A. (1993). *Head-Driven Phrase Structure Grammar*. CSLI Publications.