Reversible delayed lexical choice in a bidirectional framework

Graham Wilcock*
Centre for Computational Linguistics
University of Manchester Institute
of Science and Technology
PO Box 88, Manchester M60 1QD
United Kingdom
graham@ccl.umist.ac.uk

Yuji Matsumoto
Graduate School of Information Science
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-01
Japan
matsu@is.aist-nara.ac.jp

Abstract

We describe a bidirectional framework for natural language parsing and generation, using a typed feature formalism and an HPSG-based grammar with a parser and generator derived from parallel processing algorithms. We present an approach to delayed lexical choice in generation, based on subsumption within the sort hierarchy, using a lexicon of under-instantiated signs which are derived from the normal lexicon by lexical rules. We then show how delayed lexical choice can be used in parsing, so that some types of ill-formed inputs can be parsed, but well-formed outputs are generated, using the same shared linguistic information.

1 A bidirectional framework

In our bidirectional framework for a natural language engine, the linguistic descriptions in the grammar and lexicon are shared resources which are used, in appropriately compiled forms, for both parsing and generation. Like the Core Language Engine (CLE) (Alshawi, 1992), we use a unification-based phrase structure grammar, a logical form representation, a bottom-up chart parser and a semantic head-driven generation algorithm. However, we differ from CLE by exploiting recent developments in processing algorithms and in linguistic representation formalisms. We will clarify the similarities and differences at the relevant points.

1.1 Processing algorithms

The SAX parser is a concurrent SICStus Prolog implementation of the PAX parallel parsing algorithm (Matsumoto and Sugimura, 1987; Matsumoto et al., 1994). Like the earlier BUP parser to which the CLE parser was closely related, SAX uses partial execution to compile the grammar for efficient bottom-up parsing. However, instead of building the chart by asserting edges into the Prolog database, SAX implements the chart by creating concurrent *processes*. For compiled rather than interpreted Prolog systems, this is a highly efficient form of chart parsing, even on sequential machines. The terminal and non-terminal symbols of the grammar are realized as processes which communicate via streams to build larger structures. A meta-process monitors the streams and controls the whole parsing process.

The SGX generator is a concurrent SICStus Prolog implementation of the BCG parallel bidirectional chart generation algorithm (Haruno et al., 1993; Den, 1994). SGX uses partial execution to compile the grammar for efficient generation. Like the CLE generator, SGX follows the semantic head-driven (SHD) generation algorithm (Shieber et al., 1990) to ensure efficient ordering of the generation process. However, SGX also implements the suggestion of Shieber et.al. that backtracking and recomputation of results should be avoided by using a chart, as in chart parsing. Like SAX, SGX implements the chart by concurrent processes and communication streams monitored by a meta-process.

SAX and SGX accept definite clause grammars, with specific requirements to eliminate nondeterminism. Prolog code can be added to DCG rules as extra conditions, but the extra conditions must be deterministic. Empty categories are not supported in normal processing,[1] and SGX does not allow non-chain rules with uninstantiated logical forms.

1.2 Grammar formalism

Large DCG-based grammars typically have many rules, many categories, and many arguments per category. Such grammars could be efficiently processed by SAX and SGX, but are difficult to de-

*Visiting researcher of Information Systems Product Development Laboratories, Sharp Corporation.

[1] If empty categories are really necessary, they can be handled in the concurrent processing system via a meta-process. This approach is described in (Imaichi and Matsumoto, 1995) for ill-formed inputs. However, we eliminate traces by lexical rule, and welcome the proposals of (Sag, 1995) for eliminating all empty categories from HPSG.

velop and debug. CLE addressed this problem by adopting GPSG grammatical theory and expressing linguistic descriptions as feature structures, but the CLE grammar still had many arguments per category and many rules. We adopt HPSG grammatical theory (Pollard and Sag, 1994) and express linguistic descriptions in a typed feature formalism, but we still implement the grammar in DCG form.

Since HPSG collects all features into a structured *sign*, the many DCG arguments are replaced by a single HPSG sign. As HPSG generalizes from category-based rules (for S, NP, etc.) to schemas for phrasal signs, the many DCG rules are replaced by a few generalized rules. We specify a separate logical form (LF) for generation, as in (Shieber et al., 1990). Our DCG categories therefore have the format **word(Sign)/LF** and **phrase(Sign)/LF**.[2]

```
phrase(synsem!loc!(
    cat!(head!HF &
         subcat!@list1(SubjSynsem)) &
    cont!Cont))/lf(Cont)
-->
#   word(synsem!loc!(
        cat!(head!HF &
             subcat!@list1(SubjSynsem)) &
        cont!Cont))/lf(Cont).
```

Figure 1: The source form of a grammar rule

Figure 1 shows the source form of a simplified version of HPSG Schema 2 with zero complement daughters. `@list1` is a template which expands to a list with one member. The # symbol is required by SGX to identify the semantic head of a chain rule for SHD generation.

1.3 Grammar compilation

The ProFIT system (Erbach, 1995) is an extension of Prolog which supports a typed feature formalism with multiple inheritance. CLE used a sort hierarchy only for semantic selectional restrictions. HPSG uses a sort hierarchy also for syntactic restrictions, and exploits multiple inheritance for lexicon organization.

[2]In this paper we follow (Shieber et al., 1990) and (Pollard and Sag, 1994) in equating logical form with semantic content. A separate logical form is therefore redundant, as the **content** feature could be used to control SHD generation. However, logical form may need to include other information, such as unscoped quantifiers (HPSG **qstore**), presuppositions (HPSG **context**), pragmatic and discourse factors (in fact Shieber et al. include mood operators). An important consequence of this is that a semantic head in the sense of the HPSG Content Principle is not necessarily a semantic head in the sense of the SHD generation algorithm.

ProFIT compiles the typed feature formalism into Prolog terms, which can be used with any appropriate parser or generator. We therefore use ProFIT in order to combine HPSG grammar with the SAX parser and the SGX generator, by compiling the grammar in two separate stages.

In the first stage, the typed feature structures in the DCG rules are compiled by ProFIT into efficient Prolog terms. Like CLE, ProFIT compiles feature structures into terms, so that relatively slow unification of feature structures is replaced by relatively fast unification of terms. Also like CLE, ProFIT uses the technique of (Mellish, 1988) for compiling finite domains such as index agreement into boolean vectors for fast unification.

In the second stage, the resulting DCG containing only Prolog terms is compiled separately by the SAX translator for parsing and by the SGX translator for generation. Grammar rules can be labelled to be compiled only by SAX or only by SGX, so that parsing could for example use some rules with wider coverage than others used in generation, while sharing most of the grammar.

Like the earlier BUP parser, the SAX translator uses partial execution to produce efficient code for bottom-up chart parsing. The SGX translator compiles tables of chain rules and also uses partial execution to produce efficient code for SHD chart generation.

1.4 Lexicon compilation

We do not compile the lexicon off-line into a static list of signs. Instead, the existence of a lexical sign is *proved* on-line by lexical inference rules. We specify a *morphological lexicon* interface

morph_lex(Form, Cat, [LF, Sign])

where **Form** is a specific morphological form, and **Sign** is a typed feature structure. A lexical inference rule is shown in simplified form in Figure 2. In ProFIT, sorts are written as `<sort`, and features as `feature!value`.

```
morph_lex(Vbse, word, [lf(Cont),
    synsem!loc!(
      cat!(head!(vform!<bse &
                 aux!<n & inv!<n) &
           subcat!@list1(loc!(
               cat!(head!<noun &
                    subcat!<elist) &
               cont!(Subj &
                     index!<ref)))) &
      cont!(Cont & <psoa &
            quants!<elist &
            nucleus!(reln!Reln &
                     Role!Subj)))])
:-
        verb(Vbse, Reln, [np/Role]).
```

Figure 2: A morph_lex rule for a verb base form

We use lexical inference rules to derive full HPSG lexical signs from a database of simple Prolog clauses. Such rules can partially re-use available non-HPSG lexical information. The example assumes a lexical entry such as

```
verb(walk, walk1, [np/agent])
```

specifying a verb with base form *walk* and sense *walk1*, which subcategorizes for a noun phrase subject assigned to a thematic role *agent*.

We also use rules like normal HPSG lexical rules, to derive new signs from other lexical signs for morphological derivations, complement extraction and so on. We have no automatic defaults, so these rules must be written carefully. The simplified example in Figure 3 instantiates nominative and 3rd singular in the first subcat item, and copies the rest of subcat by unification.

```
morph_lex(V3sg, word, [lf(Cont),
    synsem!loc!(
      cat!(head!(vform!<fin &
                  aux!<n & inv!<n) &
           subcat!(first!loc!(
               cat!(head!case!<nom &
                    subcat!<elist) &
               cont!(Subject &
                     index!agr!(3&sg))) &
                  rest!Rest)) &
      cont!(Cont & nucleus!reln!Reln))])
:-
  morph_lex(Vbse, word, [lf(Cont),
    synsem!loc!(
      cat!(head!(vform!<bse &
                  aux!<n & inv!<n) &
           subcat!(first!loc!(
               cat!(head!<noun &
                    subcat!<elist) &
               cont!Subject) &
                  rest!Rest)) &
        cont!Cont)]),
  morph_infl(verb_3sg, Vbse, Reln, V3sg).
```

Figure 3: Lexical rule for 3rd singular verb form

The typed feature structures in the lexical rules are compiled by ProFIT into Prolog terms. The resulting rules are then compiled by SICStus Prolog, together with the database of simple lexical entries.

2 Delayed lexical choice

Delayed lexical choice is an established technique in natural language generation. When a backtracking algorithm is combined with a lexicon of morphological forms, there is considerable nondeterminism during syntactic generation, because features required for a deterministic choice of morphological form are not yet instantiated. With delayed lexical choice, a lexicon of stems is used during syntactic generation, and the choice of morphological form is delayed to a postprocess. Instead of producing a string of word forms, syntactic generation produces a string of lexical items. The morphological postprocess converts the lexical items to final lexical forms, when all required syntactic features have become instantiated.

2.1 Monotonicity

Describing the implementation of delayed lexical choice in the MiMo2 system, Shieber et al. (1990) pointed out that only *monotonic* rules (which further instantiate the feature structure of a lexical item but do not change it) can be delayed. For example, the choice of singular or plural verb form can be delayed until after the subject has been generated, by performing syntactic generation with a lexical item based on the verb stem, which does not specify singular or plural. By contrast, a lexical rule for passivization which changes the order of items on the subcat list is nonmonotonic. Both the active and the passive variants must be made available as distinct lexical items during syntactic generation.[3]

In an inheritance-based typed feature formalism, monotonicity is built into the subsumption relation in the sort hierarchy. A sort subsumes its subsorts, which may further instantiate its features, but cannot change them. We exploit the monotonicity of subsumption in the sort hierarchy in our implementation of delayed lexical choice.

2.2 Syntactic-semantic lexicon

In place of the MiMo2 lexicon of stems, we specify a *syntactic-semantic lexicon* interface

synsem_lex(Lex, Cat, [LF, Sign])

where **Lex** has no significance for generation. Entries in the syntactic-semantic lexicon are derived by a small number of lexical rules from entries in the morphological lexicon. Like the morph_lex rules, the synsem_lex rules are compiled first by ProFIT and then by SICStus Prolog.

To implement delayed lexical choice, we use the synsem_lex interface during syntactic generation, and then use the morph_lex interface in the morphological postprocess. We must therefore ensure that the delayed morph_lex rules will be monotonic. We do that by ensuring that the synsem_lex entries subsume the morph_lex entries from which they are derived.

Figure 4 shows a simplified form of a synsem_lex rule for verbs. The rule derives the synsem_lex entry from the morph_lex base form entry, in which `vform` has a value of sort `<bse`. The `subcat` of the synsem_lex entry is unified with the `subcat` of the

[3]We currently make such variants available via alternative forms of the lexicon access rule (Section 2.3). This could be improved by using constraints to enable subcat-changing rules to be delayed (van Noord and Bouma, 1994; Meurers and Minnen, 1995).

morph_lex entry, so that the synsem_lex entry subcategorizes for the appropriate syntactic complements. The morph_lex base form entry is used so that the agreement features of the subject will not be restricted. The `content` values are also unified, so that the synsem_lex entry includes the appropriate semantic roles. However, the `head` features are *not* unified. The synsem_lex `vform` has a value of sort `<vform`, which is the immediate supersort of the morph_lex `vform` sort `<bse`. Instead of full unification, the synsem_lex head features subsume those of the morph_lex entry.

```
synsem_lex(Lex, word, [lf(Cont & <psoa),
    synsem!loc!(
        cat!(head!(vform!<vform &
                   aux!<n & inv!<n) &
             subcat!Subcat) &
        cont!Cont)])
:-
  morph_lex(Lex, word, [lf(Cont),
    synsem!loc!(
        cat!(head!(vform!<bse &
                   aux!<n & inv!<n) &
             subcat!Subcat) &
        cont!Cont)]).
```

Figure 4: A synsem_lex rule for verbs

2.3 Grammar-lexicon interface

In DCG-based systems, the interface between the grammar and the lexicon can often be specified by a DCG rule which accesses the lexicon by means of an extra condition. In our framework, such a rule might be:

```
word(Sign)/LF -->
    [Word],
    {morph_lex(Word, word, [LF, Sign])}.
```

However, since our concurrent processing algorithms require extra conditions to be deterministic, such a rule would find only one lexical entry (the first unifiable one), which would depend on the order of lexical rules and lexical entries.

For parsing this is not a problem because, like CLE, we use a morphological preprocess which performs lexicon access, building a morpheme lattice to be passed to the syntactic parsing process. Lexicon access is therefore separated from the parsing algorithm, and is not required to be deterministic.

For generation with delayed lexical choice, we use a modified form of the rule:

```
sgx word(Sign)/LF -->
    [Sign],
    {synsem_lex(Word, word, [LF, Sign])}.
```

The label 'sgx' shows that the rule is to be compiled only by SGX, not by SAX. It differs from the previous rule not only by accessing the syntactic-semantic lexicon instead of the morphological lexicon, but also by specifying that the lexical item is [Sign] instead of [Word]. That is, the output of syntactic generation is a string of HPSG signs.

2.4 Semantic head-driven generation

When syntactic generation begins, the SHD algorithm uses chain rules (like the rule in Figure 1) to identify the pivot, the semantic head of the sentence. The synsem_lex entry for the pivot is then accessed by the extra condition in the DCG rule above.

Since the synsem_lex entry for verbs (Figure 4) does not specify subject agreement or vform subtype, but does specify subcategorization and semantic roles, it can be used equally well as the semantic head to drive syntactic generation of, say, a 3rd-singular finite clause or an infinitival complement. Since a single entry can be used in this way, the extra condition can be deterministic, as required.

If the verb is the head of an infinitival complement, its vform becomes instantiated to `<bse` from subcategorization by the auxiliary *to*. If the verb is the head of the main clause, its vform becomes instantiated to `<fin` (finite) by a rule for grammatical units in our grammar.[4]

After syntactic generation, the string of HPSG signs is converted to a string of word forms by a morphological postprocess, which unifies the signs with entries in the morphological lexicon. As the signs are fully instantiated during syntactic generation, this postprocess is also deterministic.

3 Reversible delayed lexical choice

Most forms of robust parsing are based on constraint relaxation. Our approach to delayed lexical choice is based on using *less instantiated* signs from the syntactic-semantic lexicon, rather than the more instantiated signs from the morphological lexicon. This can be viewed as equivalent to constraint relaxation. It therefore seems reasonable to consider *reversing* the approach, using delayed lexical choice for parsing.

Constraint relaxation in parsing typically has a two-pass approach. Strict parsing is attempted with normal grammar rules and the normal parsing algorithm. If strict parsing fails to produce a parse, relaxed parsing is attempted, using a modified algorithm or modified grammar rules. With a lexicalist grammar like HPSG it seems more appropriate to use modified lexical rules, as in our syntactic-semantic lexicon.

[4] In addition to the HPSG categories `word` and `phrase`, we have a category `gram_unit` for grammatical units which can be uttered independently. These include *finite* sentences, *accusative* NPs, and others. The rule for `gram_unit` thus instantiates certain features which are required for well-formed generation.

However, in our approach to delayed lexical choice we do not start with strict constraints and then relax them. On the contrary, we start with relaxed constraints from less instantiated signs and then further instantiate the signs as other constraints become available. Our approach is therefore *incremental description refinement* (Mellish, 1988) rather than constraint relaxation.

3.1 Parsing and generation with delay

When the syntactic-semantic lexicon is used for generation, the logical form is the retrieval key, and the name of the lexeme is irrelevant. In the interface **synsem_lex(Lex, word, [LF, Sign])**, the variable **Lex** does not need to be unified with the name of the morpheme in the morph_lex entry, and could be given another value, such as "verb". However, if we use the syntactic-semantic lexicon for parsing, the value of this variable will be the retrieval key. If the value is taken directly from the words of the input string, it will not necessarily unify with the name of the morpheme in the morph_lex entry.

In the case of verbs (Figure 4), where the input word may be an inflected form but the synsem_lex entry uses the morph_lex entry for the base form, we must first use the morphological preprocess to obtain the "root" form of the word, which is the same as the base form. We then use the root form instead of the input form as the retrieval key. In the case of pronouns, which take different forms according to case and reflexivity but have no natural root form, the input form is used directly as the retrieval key (Section 3.2).

Since the synsem_lex entry for verbs in Figure 4 does not restrict subject agreement, an ill-formed input with incorrect subject-verb agreement is parsed in exactly the same way as a well-formed input. The subject agreement in the verb's sign remains uninstantiated until the subject and the verb phrase are combined by Schema 1, when the agreement features are instantiated to those of the subject. So "she swim" is accepted, but only "she swims" is generated in a finite clause. The synsem_lex entry in Figure 4 also does not restrict `vform`, which remains uninstantiated until the verb phrase is combined into a larger phrase. So "she can swimming" is accepted, but only "she can swim" is generated, since "can" subcategorizes for a VP with vform of sort `<bse`.

3.2 Experimenting with delay

Of course, different specifications in the rules for the syntactic-semantic lexicon produce different effects. In the synsem_lex entry for pronouns in Figure 5, instead of unifying the head feature `case` with the morph_lex entry, the head is specified only as sort `<noun`, leaving the case unspecified. There are distinct morph_lex entries for nominative and accusative forms of personal pronouns, but it is irrelevant which one happens to be found when the rule is executed, because the rule does not unify the head features which include case. So the synsem_lex entry can be used deterministically for syntactic generation, leaving the case to be instantiated from subcategorization by a verb or preposition.

In parsing, the effect of this form of the rule is that the case of an input pronoun is ignored. Whether this is good or bad depends on both the language and the level of relaxation desired. This form of the rule would clearly be unsuitable for free word order languages, but seems useful for English, accepting "for you and I" but generating "for you and me".

```
synsem_lex(Lex, word, [lf(SynsemCont),
    synsem!loc!(
        cat!(head!<noun &
             subcat!<elist) &
        cont!(SynsemCont & <pron &
              index!Index &
              restr!<elist))])
:-
  morph_lex(Lex, word, [lf(MorphCont),
    synsem!loc!(
        cat!(head!<noun &
             subcat!<elist) &
        cont!(MorphCont & <pron &
              index!Index &
              restr!<elist))]).
```

Figure 5: A synsem_lex rule for pronouns

In Figure 5, the synsem_lex `content` value is not unified with the morph_lex `content` value. Only the `index` values are unified, including the gender, number and person features essential for pronouns (the `restr` values are empty lists). The `content` values are constrained only to be of sort `<pron` (pronominal). In the sort hierarchy, `<pron` has subsorts `<ana` (anaphoric) and `<ppro` (personal-pronominal), and `<ana` has its own subsorts `<refl` (reflexive) and `<recp` (reciprocal). HPSG binding theory is based on these sortal distinctions, which are part of the `content` value.

Again, there are distinct morph_lex entries for reflexive and personal-pronominal forms, but it is irrelevant which one happens to be found when the rule is executed, because the rule does not unify the `content` values. Therefore the synsem_lex entry can be used deterministically for syntactic generation before the sort becomes instantiated to `<ana` or `<ppro` by the binding principles.

The effect of this form of the rule is to relax the binding constraints in parsing, accepting "I saw me" but generating "I saw myself". Of course the distinction between "They saw themselves" (co-indexed) and "They saw them" (contra-indexed) is also lost in parsing with this version. The bind-

ing constraints can be re-instated simply by unifying the `content` values in the rule, but the above version is not necessarily bad, for example in parsing non-native English. The rule could be improved by having alternative forms which distinguish 3rd and non-3rd person.

4 Conclusion

Our framework seeks to combine the elegance of a typed feature formalism and HPSG syntactic theory with efficient processing. The ProFIT system gives an efficient encoding of typed feature structures. The SAX and SGX systems use an efficient chart implementation, and their concurrent processing algorithms give further motivation for eliminating empty categories and reducing nondeterminism.

We have addressed the issue of bidirectional use of shared linguistic descriptions, rather than robust parsing. However, the idea of using delayed lexical choice in reverse makes it possible, without modifying the parsing and generation algorithms, to parse certain types of ill-formed inputs and to generate corresponding well-formed outputs, using the same shared linguistic descriptions.

Acknowledgements

The first author would like to thank Mr Hitoshi Suzuki (Sharp Corporation) and Prof Jun-ichi Tsujii (UMIST) for making this work possible. We also thank Dr Kristiina Jokinen (NAIST) and the anonymous reviewers for valuable comments.

References

Hiyan Alshawi, editor. 1992. *The Core Language Engine*. The MIT Press.

Yasuharu Den. 1994. Generalized chart algorithm: an efficient procedure for cost-based abduction. In *32nd Annual Meeting of the Association for Computational Linguistics*, pages 218–225. Association for Computational Linguistics.

Gregor Erbach. 1995. ProFIT: Prolog with Features, Inheritance, and Templates. In *Seventh Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics.

Masahiko Haruno, Yasuharu Den, Yuji Matsumoto, and Makoto Nagao. 1993. Bidirectional chart generation of natural language texts. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 350–356. AAAI Press/The MIT Press.

Osamu Imaichi and Yuji Matsumoto. 1995. Integration of syntactic, semantic and contextual information in processing grammatically ill-formed inputs. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1435–40.

Yuji Matsumoto and Ryoichi Sugimura. 1987. A parsing system based on logic programming. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, volume 2, pages 671–4.

Yuji Matsumoto, Yasuharu Den, and Takehito Utsuro, 1994. *Koubun kaiseki shisutemu SAX, shiyou setsumeisho (Parsing system SAX Manual) version 2.1*. Nara Institute of Science and Technology.

Christopher S. Mellish. 1988. Implementing systemic classification by unification. *Computational Linguistic*, 14(1):40–51.

W. Detmar Meurers and Guido Minnen. 1995. A computational treatment of HPSG lexical rules as covariation in lexical entries. In *Proceedings of the Fifth International Workshop on Natural Language Understanding and Logic Programming*, Lisbon, Portugal.

Carl Pollard and Ivan Sag. 1994. *Head-driven Phrase Structure Grammar*. University Press, Chicago.

Ivan Sag. 1995. English relative clause constructions. Unpublished manuscript.

Stuart M. Shieber, Gertjan van Noord, Fernando C.N. Pereira, and Robert C. Moore. 1990. Semantic head-driven generation. *Computational Linguistics*, 16(1):30–42.

Gertjan van Noord and Gosse Bouma. 1994. Adjuncts and the processing of lexical rules. In *15th International Conference on Computational Linguistics*. Association for Computational Linguistics.