Morphology in MiMo2

Gertjan van Noord

Research Institute for Language and Speech
Trans 10 3512 JK Utrecht
The Netherlands
vannoord@let.ruu.nl

Abstract

This paper gives an informal overview of the orthographical and morphological components of the experimental MiMo2 machine translation system.

1 Introduction

The MiMo2 system is an experimental machine translation system developed by the Utrecht Eurotra center as a sideline of the official Eurotra work. In MiMo2 reversible unification grammars are used to define monolingual relations betweeen strings and logical forms (for parsing and generation) and bilingual relations between language dependent logical forms (for transfer). An overview of the system can be found in [12]. The notion of reversibility is developed in [11].

In this paper I will give an overview of the morphological components of the system. These components are completely reversible; i.e. they are used succesfully in both analysis and generation and define symmetric relations. The relation between the different components can be graphically represented as in figure 1. Together the different morphological modules define the lexical entries of the monolingual unification grammar for syntax and semantics.

The easiest way to think of the relation between the different components is as being used in generation. A morphological unification grammar together with a lexicon of stems defines a set of pairs ⟨ list of morphemes, feature structure ⟩; for example the following pair for the dutch compound 'opbel' (upcall = to call up).

$$\langle [op, bell], \begin{bmatrix} head = \begin{bmatrix} cat = verb \end{bmatrix} \\ pred = opbellen \\ subcat = \ldots \end{bmatrix} \rangle$$

Figure 1: The relation between different modules

```
-------------    ---------------    --------------
|           |    |             |    |            |
| morpholo- |    | paradigmatic |    |  two-level |
|gical rules |    |             |    |            | lexical entries
|___________|<=>|             |<=>|            | =      of
|           |    |             |    |            |  unif. grammar
|  lexicon  |    | inflection  |    | orthography |
| of stems  |    |             |    |            |
|___________|    |_____________|    |____________|
```

Inflection rules define a mapping between such pairs. For example a rule might be applied for third singular present tense, yielding:

$$\langle [op, bell, t], \begin{bmatrix} head = \begin{bmatrix} cat = verb \\ vform = fin \\ tense = present \\ agr = \begin{bmatrix} person = third \\ number = sg \end{bmatrix} \end{bmatrix} \\ pred = opbellen \\ subcat = \ldots \end{bmatrix} \rangle$$

In the third component, the two-level orthography, the list of stems of such pairs will be mapped into a word by applying the orthographical rules. The feature structure will simply be copied over. This may then yield:

$$\langle opbelt, \begin{bmatrix} head = \begin{bmatrix} cat = verb \\ vform = fin \\ tense = present \\ agr = \begin{bmatrix} person = third \\ number = sg \end{bmatrix} \end{bmatrix} \\ pred = opbellen \\ subcat = \ldots \end{bmatrix} \rangle$$

The resulting set of word/feature structure pairs act as the lexical entries of the unification grammar defined for syntax and semantics.

2 Two-level orthography

The two-level orthography component of MiMo2 is developed by Herbert Ruessink. The component is described in a seperate paper [6]. The component is a variant of the two-level formalism such as originally defined by Koskenniemi [3] and revised by Black et al. [1].

Essentially the component defines a (symmetric) relation between lists of characters. However, as the morpheme boundary will be a character on the lexical side, we can think of a pair "# o p b e l t #" ⇔ "# o p + b e l l + t #" as a word / list of morphemes pair "opbelt" ⇔ "op bell t".

Each rule is of the form

```
SurfPat Arrow LexPat (:- Cond) .
```

SurfPat and LexPat are patterns consisting of list of characters (or variables) of the form

```
LeftContext^Change^RightContext
```

and Cond are Prolog conditions (optional). The arrow is one of

```
==>
<==
```

for optional and obligatory rules respectively. The reader is referred to the paper by Ruessink for the exact interpretation of the rules.

The following set of rules is the current set of rules for English:

```
[]^[]^[] ==> []^[+]^[].

[]^[X]^[] ==> []^[X]^[]
    :- \+ X = '+', \+ X = '#'.

[e]^[]^[d,#] <== []^[e]^[d,+].

[]^[]^[] <== []^[e]^[+,i,n,g].

[]^[i]^[] <== [C]^[y]^[+,e,d]
    :- consonant(C).

[]^[i,e]^[s,#] <== [C]^[y]^[+,s]
    :- consonant(C).

[o]^[]^[#] <== [o]^[e]^[].

[C1]^[]^[C2] <== [C1]^[C1]^[+]
    :- consonant(C1), consonant(C2), \+ slf(C1).

[C1]^[]^[#] <== [C1]^[C1]^[+]
```

```
:- consonant(C1), \+ slf(C1).
```

The first two rules can be defined as the minimal set of rules, in case this component is not used.

3 Paradigmatic Inflection

The inflection component resembles the paradigmatic inflection rules of Calder [2]. The rules define triples of the form ⟨ list of morphems, list of morphemes, feature structure ⟩. For example, ⟨ "op bell t", "op bell", [number=sg, cat=v] ⟩.

Notice that the feature structures defined here will be *unified* with the feature structure resulting from the morphological unification grammar. Thinking the other way around, inflection rules can only *further instantiate* feature structures. Also notice that each word *must* be handled by exactly one and only one inflection rule (thus you need to define inflection rules for eg. prepostions as well).

Infl rules look like:

```
LexicalPattern <==> SurfacePattern
  :  FeatureStructure
( :  Exceptions )
( :- Strong )
```

The exceptions part and the strong part are optional. A *pattern* is built from variables, the concatenation sign (/), and morphemes (strings in square brackets). Usually variables in the pattern occur both in the lexical- and the surface part. An example of a pattern is "X/[t]"; this pattern may match with "op bell t" binding X to "op bell". Note that pattern matching here is string unification. Some simple examples:

```
X <==> X :
[ <head cat> = v,
  <subject head agr number> = plur,
  <head tense> = pres,
  <head vform> = fin ].

X <==> Y/[s] :
[ <head cat> = v,
  <subject head agr number> = sing,
  <subject head agr person> = third,
  <head tense> = pres,
  <head vform> = fin ].
```

```
X <==> X:
[ <head cat> = prep ]
```

Exceptions is a list of irregular cases *for this specific rule*, where one case looks like ListOfMorphemes-ListOfMorphemes, for example the exception list of a rule for English plural nouns may look as follows:

```
X <==> X/[s] :
[ <cat> = noun,
  <head agr number> = plural ] :
[ [man] - [men],
  [woman] - [women],
  [sheep] - [sheep],
  [mouse] - [mice] ].
```

The exception thus only is an exception with respect to the strings, but not w.r.t. the feature structure. Note that relation between strings defined in the pattern applies iff no exception applies and moreover after applying the pattern no string occurs for which an exception was defined. This complication is necessary to preserve the symmetry.

The Strong part is the prolog call to strong_pst/3 or strong_prt/3; these are precompiled definitions resulting from the loading of the lexicon. In the lexicon it is assumed that for so called 'strong' verbs the irregular forms are the value of some specified features (eg. the features ¡head strong past¿ and ¡head strong part¿). This will then result in a list of definitions:

```
% strong_pst(LexicalMorpheme,PastMorpheme,Language)
strong_pst(go,went,english).
strong_pst(do,did,english).
strong_pst(see,saw,english).

% strong_prt(LexicalMorpheme,PartMorpheme,Language)
strong_prt(go,gone,english).
strong_prt(do,done,english).
strong_prt(see,seen,english).
```

Variables of the pattern are then usually unified with arguments of these predicates; for example:

```
X <==> Y :
[ <head cat> = v,
  <head strong value> = yes,
  <head vform> = fin,
  <head tense> = past ]
     :- strong_pst(X,Y,english).
```

If the user does not want to use the inflectional component he/she might define the following rule:

```
X <==> X : [].
```

4 Derivations, Compounds and Lexical Rules

Derivations, compounds and lexical rules can be defined by a seperate unification grammar. The lexicon for this grammar is a lexicon of stems. Possible 'sentences' of this grammar (i.e. lists of morphemems) with their associated feature structures will be input to the inflection rules. Or, going the other way around, the output of the inflection rules will be a parsing problem for this unification grammar.

The parser and generator of this grammar are completely similar to the parser and generator defined for the syntax/semantics unification grammar. For the parser see for instance [4, 5]; for the generator [9, 7, 8, 10].

Note that if the user does not intend to use the morphological unification grammar no special action is required.

5 Other remarks

Note that both the inflection and orthographical rules are monotonic in the sense that they can only *add* information to the feature structures defined by the morphological unification grammar. This allows the delay of inflection- and orthographical rules during generation (cf. [8]).

References

[1] Alan Black, Graeme Ritchie, Steve Pulman, and Graham Russell. Formalisms for morphographemic description. In *Third Conference of the European Chapter of the Association for Computational Linguistics*, Copenhagen, 1987.

[2] Jonathan Calder. Paradigmatic morphology. In *Fourth Conference of the European Chapter of the Association for Computational Linguistics*, Manchester, 1989.

[3] Kimmo Koskenniemi. Two-level morphology: a general computational model for word-form recognition and production. Technical Report 11, Department of General Linguistics, University of Helsinki, 1983.

[4] Y. Matsumoto, H. Tanaka, H. Hirakawa, H. Miyoshi, and H. Yasukawa. BUP: a bottom up parser embedded in prolog. *New Generation Computing*, 1(2), 1983.

[5] Fernando C.N. Pereira and Stuart M. Shieber. *Prolog and Natural Language Analysis*. Center for the Study of Language and Information Stanford, 1987.

[6] Herbert Ruessink. Two-level formalisms. *Working Papers in Natural Language Processing, Katholieke Universiteit Leuven, Stichting Taaltechnologie Utrecht*, 5, 1989.

[7] Stuart M. Shieber, Gertjan van Noord, Robert C. Moore, and Fernando C.N. Pereira. A semantic-head-driven generation algorithm for unification based formalisms. In *27th Annual Meeting of the Association for Computational Linguistics*, Vancouver, 1989.

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

[9] Gertjan van Noord. BUG: A directed bottom-up generator for unification based formalisms. *Working Papers in Natural Language Processing, Katholieke Universiteit Leuven, Stichting Taaltechnologie Utrecht*, 4, 1989.

[10] Gertjan van Noord. An overview of head-driven bottom-up generation. In Robert Dale, Chris Mellish, and Michael Zock, editors, *Current Research in Natural Language Generation*. Academic Press, 1990.

[11] Gertjan van Noord. Reversible unification-based machine translation. In *Proceedings of the 13th International Conference on Computational Linguistics*, Helsinki, 1990. to appear.

[12] Gertjan van Noord, Joke Dorrepaal, Pim van der Eijk, Maria Florenza, and Louis des Tombe. The MiMo2 research system. In *Proceedings of the Third International Conference on Theoretical and Methodological issues in Machine Translation of Natural Languages*, University of Texas at Austin, 1990. to appear.