LMT

Michael MCCORD

IBM Thomas J. Watson Research Center
P. O. Box 704
Yorktown Heights, NY 10598

1 INTRODUCTION

The main characteristics of **LMT** are the following: (a) It is formulated in a logic programming framework (whence the name, "**L**ogic-programming-based **M**achine **T**ranslation"). **LMT** is implemented entirely in Prolog. (b) Source analysis is based on Slot Grammar (McCord 1980, 1989). This grammatical system has a high degree of lexicalism, involving a systematic use of *slot frames* derived from lexical entries. (c) A large portion of the system is language-independent, forming a kind of "X-to-Y translation shell."

LMT is a transfer-based system and currently deals with only one sentence (or input phrase) at a time. There are five steps (passes) in translating a sentence: (1) source/transfer lexical processing, (2) source syntactic analysis (with Slot Grammar), (3) transfer, (4) target syntactic generation, and (5) target morphological generation. The output of source analysis (and input to transfer) is basically a syntax tree (a dependency tree), but it shows deep grammatical relations through unification of logical variables associated with slots and their fillers – including remote dependencies, logical relations in passive constructions, and implicit subjects in non-finite verb phrases.

LMT began as an English-German system (McCord 1986, 1988a, McCord and Wolff 1988), but the shell has facilitated the development of prototype versions for other language pairs, in co-operation with other groups and individuals in IBM, namely: English-Danish and Danish-English (Arendse Bernth and IBM European Language Services), English-French (the KALIPSOS team of the IBM Paris Scientific Center, with work especially by Eric Bilangc), German-English (Ulrike Schwall, IBM Germany Scientific Center), and English-Spanish (Nelson Correa).

In the following sections we will discuss briefly the treatment of the five steps of translation in **LMT**, except for morphological generation, which is adequately discussed in previous papers cited.

2 SOURCE/TRANSFER LEXICAL PROCESSING

In the current design of **LMT**, there is only one lexicon for each language pair, the *source/transfer lexicon.* It is indexed by citation forms of source language words, and each entry contains *source elements* and *transfer elements.* The source elements arc readily separable (and could form the basis of the lexicon for a different target language), but the transfer elements refer to information in the various source elements for a source word.

An example (simplified) entry in an English-German lexicon for the word *view* might be

```
view < v(obj) < n(nobj)
    < tv(acc,be+tracht) < tn(gen,ansicht.f.n).
```

There are two source elements on the first line showing *view* as a verb and a noun, and associated transfer elements on the second line. In general, there can be several transfer elements for each source element, and they can contain semantic types for use in selecting target words.

The ingredients of lexical entries most consequential for the rest of the system are slot lists shown in source elements and corresponding case lists shown in associated transfer elements. Above, **`view`** as a verb is given a direct object slot **`obj`**; the **`subj`** slot for verbs is added by the system. Such slots are *syntactic* slots, representing *complements,* and are important input to the Slot Grammar. *Adjunct* slots are declared in the grammar itself.

The lexical processing step actually involves two substeps. For each word in the input (possibly inflected or derived), the first step is to use morphological rules to produce a *derived* lexical entry for the word. Such an entry is produced from the entry of the base word by applying operations to its elements, both source and transfer, associated with the affixes of the derived word.

The second substep is to translate the derived lexical analysis elements into Prolog clauses useable by source analysis and transfer. Source analysis elements for a word **`Word`** are translated into clauses for the predicate

```
wordframe(Word,Sense,Features,Slot Frame).
```

Mere **`Sense`** is normally the citation form of **`Word`**, but can be a particular sense name (like **`give1`**) if such is specified in the source element, or can show morphological structure in the case of a derived word. **Features** is a term representing the morphosyntactic feature structure of the word, showing mainly part of speech and inflectional features. And **`SlotFrame`** is the list of complement slots in *internal form. Ea*ch internal-form slot is of the form **`slot(Slot,Ob,X)`**, where **`Slot`** is the slot name (like **`obj`**, appearing in the external lexical analysis), **`Ob`** indicates whether the slot is obligatory or not, and **`X`** is the *marker* of the slot. When the slot is filled by a phrase, the slot marker **`X`** is unified with the term **`e(X0)`** where **`X0`** is the *marker* of the filler phrase. The marker of a phrase is a logical variable that serves as a kind of identifier of the phrase.

The main part of the treatment of passive verb constructions occurs during the formation of **`wordframe`** clauses. A passive form of a verb, such as an English past participle or a Danish finite passive form, leads to one or more word frames in which the slot frame is altered to show the appropriate passive slots. For more details, see (McCord 1989), or for an earlier treatment, (McCord 1982).

The internal form of transfer elements is described below in Section 4.

The **LMT** lexical format allows for multiword entries for all of the parts of speech. In the case of multiwords, the internal Prolog predicate for source elements is slightly different from the above, showing the boundaries of the multiword in the input word string.

3 SLOT GRAMMAR FOR SOURCE ANALYSIS

The original Slot Grammar system (McCord 1980) was developed in 1976-1978 without consideration of logic programming (and was implemented in Lisp). The Modular Logic Grammar system (McCord 1982, 198.5, 1987), used in the earlier design of **LMT**, represents a *combination* of the original Slot Grammar techniques with the (augmented) phrase structure grammar techniques common in logic programming (Colmerauer 1978), and employs top-down parsing. In this combined approach, Slot Grammar rules, expressed in terms of phrase structure rules and Prolog clauses, are used systematically for postmodification of open-class words; but elsewhere in the grammar more standard phrase structure rules are used.

Recently (McCord 1989) the original ("pure") Slot Grammar system was redone in a logic programming framework, with several improvements, and the rest of the **LMT** system was adapted to this method of source analysis. As with the original Slot Grammar system, a bottom-up chart parser is employed. For comparison of Slot Grammar with other grammatical systems, see (McCord 1989).

A Slot Grammar uses no phrase structure rules, and instead consists (modularly) of the following: (1) A declaration *of adjunct slots* for each part of speech, (2) *slot filler rules,* (3) *slot ordering rules,* and (4) *obligatory dot rules.* In addition, there are certain language-specific data (expressed mainly as unit clauses) for treating extraposition, coordination, and punctuation. Much of the treatment of these constructions is in the language-independent parser module (which is part of the shell).

Let us look briefly at the four main ingredients listed above. There are special rule formalisms and rule compilers for rules of types 2, 3, and 4.

Adjunct slots are declared simply by unit clauses

```
adjuncts(POS,Adjuncts)
```

where **POS** is a part of speech (like **verb**) and **Adjuncts** is the list of possible adjunct slots for all words of that part of speech. Unlike a complement slot, which may be obligatory and can be filled at most once, an adjunct slot is optional and can be filled any number of times.

Slot filler rules are the core of a Slot Grammar, constituting the main rules for modification of one phrase by another. More specifically, given a phrase **P**, one chooses an *available* slot **Slot** for **P**, *i.e.* either an unfilled (complement) slot in the slot frame of **P** (arising from the **wordframe** of its head word), or an adjunct slot associated with the part of speech of the head word. A phrase **M** adjacent to **P** will be a *filler* of **Slot** (and a modifier of **P**) if there is *a filler rule.*

```
Slot ==> Body
```

for which **Body** holds. The condition **Body** has the same form as the body (antecedent) of a Prolog clause, but it can contain *special goals* which refer to the characteristics of the higher phrase **P** or the modifier phrase **M**, which are implicit in the use of the rule. The rule compiler converts a filler rule to a Prolog clause in which the filler phrase and the higher phrase are mentioned explicitly.

An example of a filler rule for the subject slot in English might be

```
subj ==> f(noun(*,nom,Pers,Num)) & hf(verb(fin(Pers,Num,*,*))).
```

Here the special goal **f(F)** (or **hf(F)**) requires that **F** is the feature structure of the filler phrase (or the higher phrase).

There are two types of slot ordering rules, (1) *head/slot ordering rules,* expressing ordering of slots (their fillers actually) with respect to the head word, and (2) *slot/slot ordering rules,* expressing ordering of slots with respect to other slots.

Head/slot ordering rules are of either of the forms:

```
lslot (Slot)   ← Body.
rslot(Slot)  ← Body.
```

These rules say respectively that **Slot** is a *left slot* (or *right slot),* under the conditions of **Body**. The condition **Body** (which may be omitted, with the arrow) may, like the body of a filler rule, contain special goals referring to the modifier phrase or the higher phrase. For example, the rule

```
rslot(subj) ← hf(verb(fin(*,*,*,ind:q:*))) & hsense(Verb) & finaux(Verb).
```

says that **subj** can be a right slot in a question sentence (feature **q**) if the verb is a finite auxiliary.

A slot/slot ordering rule is of the form:

```
LSlot<<RSlot ← Body
```

where the **Body** may be omitted. This means that every filler for **LSlot** must precede every different filler for **RSlot** under the conditions of **Body**. Again, the body can contain special goals referring to the higher phrase or to the filler phrases for **LSlot** or **RSlot**.

Examples of slot/slot ordering rules are the following ones, expressing relative ordering of the direct and indirect objects of verbs:

```
iobj<<obj ← If(noun(*,*,*)).
obj<<iobj ← rf(prep(*,*,*)).
```

These say that the indirect object precedes or follows the direct object according as the indirect object is a noun phrase or a prepositional phrase.

For a phrase to become a modifier of another phrase, all of its *obligatory* slots (after possible extraposition of one of its slots) must be filled. A complement slot may be specified in the lexicon to be obligatory, or one may specify this by a general rule in the grammar. Such rules are of either of the forms:

```
obl(Slot) ← Body.
obl(Slot,Slot1) ← Body.
```

The first rule says that **Slot** is obligatory under the conditions of **Body**, and the second says that **Slot** must be filled if **Slot1** is filled and **Body** holds. The body again can contain special goals. Examples:

```
obl(subj) ← hf(verb(fin(*,*,*,*))).
obl(obj,iobj) ← f(noun(*,*,*)).
```

The second rule says that the direct object must be filled if the indirect object is filled by a noun phrase.

Left extraposition is handled as follows. One declares in the grammar that certain slots allow extraposition of other slots out of their fillers. The parser then takes care of the actual extraposition of slots, storing them in the *extraposed slot list* which is part of a phrase data structure. In turn, the grammar can contain filler rules for such extraposed slots, of the form:

```
ext(Slot,Level)  ==> Body.
```

Here **Level** is **ext** if Slot is actually extraposed, or **norm** if **Slot** is a normal slot. (The latter is needed for examples like the relative clause *who Mary saw.)* The parser again takes care of applying such rules appropriately.

As indicated above, most of the treatment of coordination is in the parser module. The method involves "factoring out" common (or closely similar) slots in elliptical coordinated phrases and filling them on a higher level. This method was outlined in (McCord 1980); for details, see (McCord 1989).

The Slot Grammar parser includes a parse evaluation scheme used for pruning away unlikely analyses during parsing as well as for ranking final analyses. The parse evaluator expresses weighted preferences for close attachment, for complement modification over adjunct modification, and for parallelism in coordination. Parse space pruning may be turned on or off optionally. When it is on, it is fairly common to get only one parse which is correct modulo attachment of postmodifying adjuncts. When it is off, one gets all the parses allowed by the grammar (but these are still ranked).

4 TRANSFER

The main, recursive procedure

```
tran(SourceSlotFiller,Mother-Features,TargetSlotFiller)
```

for transfer takes a source slot/filler pair and the target features corresponding to the mother of this pair, and produces a target slot/filler pair. The steps (basically) in the definition of tran are the following three:

(1) Transfer the features of the source to the features of the target. In doing this one can refer (as input) also to the source slot, the source frame, the source marker, and the mother target features.

(2) Transfer the source head word to the target head word, by a procedure

```
tranword(TargetFeas,SourceWord,X.SourceFrame,TargetWord).
```

Here the term **X** is the marker of the source phrase.

(3) Recursively call **tran** on the modifier slot/filler pairs, using the current target features as the middle argument.

All of the rules for steps (1) and (3), and the top-level rules for step (2), are in the shell. There are some non-trivial things to do in step (1) (feature transfer), getting features in a form suitable for generation and managing the communication up and down the tree through the "mother features" argument. The language-independent top level of word transfer (**tranword**) is also non-trivial, involving rules for handling passives, subject verb agreement, etc.

The top level of **tranword** also includes the interface to the language-specific lexical transfer rules. The latter are rules for a procedure

```
twordframe(POS,SourceWord,Args,TargetWord)
```

(we give a slightly simplified description) which are derived on the fly by the lexical processing step from transfer elements in the lexical analysis of (a derived form of) the source word. As an example, an English-German transfer element for *give* might give rise to a unit clause

```
twordframe(verb, give, nom.acc.dat.nil, geb).
```

In case of a verb, the members of the **Args** list (the third argument of **twordframe**) get unified by **tranword** with the markers of the corresponding complement phrases via the slot frame in the third argument of **tranword**. Hence in step (1) above (feature transfer) such symbols can mark cases in noun phrases or name prepositions in PP complements, or even serve as rule switches for VP complements.

We have omitted the description of semantic type checking in lexical transfer, but this also is handled through the communication of slots to fillers via markers.

5 SYNTACTIC GENERATION

Syntactic generation applies a system of tree transformations to the transfer tree in order to get a target tree in the correct target surface form. (Actually, transformations apply to slot/filler pairs.) The algorithm for applying the transformations, as well as some supporting procedures for writing transformations, are language-independent, but the transformations themselves can depend both on the source and target languages. However, it is possible to share transformations for different language pairs.

There is a special formalism which allows one to write transformations using an extension of Prolog pattern matching involving sublist variables. A rule compiler converts transformation rules to Prolog clauses (McCord 1986, 1988a).

Transformations are specified in two ordered lists. The first list, of *b-tramforms,* consists of transformations that can apply only to non-coordinated phrases. The second list, of *c-transforms,* consists of transformations that can apply to any phrases (possibly coordinated). Generally, there are many more b-transforms than c-transforms.

The algorithm for applying transformations is as follows: In transforming a slot/filler pair, one first recursively transforms all the modifiers of the filler phrase. Then if the phrase is not coordinated, one runs through the b-transform list, applying each one *once* if possible. Then one runs through the c-transform list, applying each one once if possible.

For more discussion of transformations, see (McCord 1986, 1988a). The basic techniques are similar to those in this earlier version of **LMT**, but new ingredients are (1) the separation of transformations into b-transforms and c-transforms, and (2) the treatment of ordering. In the earlier version, Prolog clause ordering was used, and a given transformation could apply more than once on a given level. Also in the new version it is quite convenient to be able to refer to slots as well as features.

6 REFERENCES

Colmerauer, A. (1978) "Metamorphosis Grammars," in L. Bolc (Ed.), *Natural Language Communication with Computers,* Springer-Verlag.

McCord, M. C. (1980) "Slot Grammars," *Computational Linguistics,* vol. 6, pp. 31-43.

McCord, M. C. (1982) "Using Slots and Modifiers in Logic Grammars for Natural Language," *Artificial Intelligence,* vol. 18, pp. 327-367.

McCord, M. C. (1985) "Modular Logic Grammars," *Proc. 23rd Annual Meeting of the Association for Computational Linguistics,* pp. 104-117, Chicago.

McCord, M. C. (1986) "Design of a Prolog-based Machine Translation System," *Proc. of the Third International Logic Programming Conference,* pp. 350-374, Springer-Verlag, Berlin.

McCord, M. C. (1987) "Natural Language Processing in Prolog," in Walker et al. (1987).

McCord, M. C. (1988a) "Design of LMT: A Prolog-based Machine Translation System," Research Report RC 13536, IBM Research Division, Yorktown Heights, NY 10598. To appear in *Computational Linguistics.*

McCord, M. C. (1988b) "A Multi-Target Machine Translation System," *Proc. of the International Conference on Fifth Generation Computer Systems 1988,* pp. 1141-1149, Institute for New Generation Computer Technology, Tokyo, Japan.

McCord, M. C. (1989) "A New Version of Slot Grammar," Research Report RC 14506, IBM Research Division, Yorktown Heights, NY 10598.

McCord, M. C. and Wolff, S. (1988) "The Lexicon and Morphology for LMT, a Prolog-based MT system," Research Report RC 13403, IBM Research Division, Yorktown Heights, NY 10598.

Walker, A. (Ed.), McCord, M., Sowa, J.F., and Wilson, W. G. (1987) *Knowledge Systems and Prolog: A Logical Approach to Expert Systems and Natural Language Processing,* Addison-Wesley, Reading, Mass.