PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1988,
edited by ICOT. © ICOT, 1988

A MULTI-TARGET MACHINE TRANSLATION SYSTEM

Michael C. McCord

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

ABSTRACT

The **LMT** system is a Prolog-based machine translation system, originally developed for English-to-German translation. Recently the system has been reorganized and improved so that it contains a large subsystem **LMTX** – an "English-to-X translation shell" – which is essentially independent of the target language, and which can be shared dynamically among versions of **LMT** for different target languages. An **LMT** system consists of the shell **LMTX** plus one or more target-language-specific subsystems ***T*SPEC** for target languages *T*. The shell includes (1) the English grammar (a Modular Logic Grammar), (2) most of the source/transfer morphology system and lexical processing system, (3) the transfer algorithm and rule system, except for low-level, lexical transfer entries, (4) the syntactic generation algorithm, (5) target-independent procedures dealing with morphological generation, and (6) many utility procedures. A target-specific subsystem ***T*SPEC** consists of (a) a source/transfer lexicon, (b) a set of transformations for syntactic generation, and (c) a target morphological system.

1 INTRODUCTION

The **LMT** system[1] (McCord 1986, McCord and Wolff 1987, McCord 1988) was originally developed as an English-to-German MT system. Recently, **LMT** has become a multi-target system, based on an "English-to-X translation shell," **LMTX**, which is essentially target-language-independent and can be shared among versions of **LMT** for different target languages.

Versions of **LMT** for the target languages French (**LMTF**), Danish (**LMTD**), Spanish (**LMTS**), and Portuguese (**LMTP**) have been started up, and the German version **LMTG** is being further improved.[2] These different versions share **LMTX** in the strong sense that they can all be in the Prolog workspace simultaneously, sharing the same code for **LMTX**. One can switch target languages in the same session by typing *e.g.* `+french` or `+german`. Obviously, the idea is to design **LMTX** so that the portion **LMT*T*-LMTX** of a given version **LMT*T*** can be minimized.

One way of trying to develop MT systems efficiently for multiple languages is to exploit symmetry as much as possible by designing grammars that are useable for both analysis and synthesis (see *e.g.* Jin and Simmons 1986). In a stronger version, there is a universal semantic representation language, an interlingua, used as the target of analysis and the source of synthesis (see *e.g.* Carbonell and Tomita (1986)). **LMT** is not symmetric in these ways. The analysis and synthesis grammars are quite different in nature. It seems better to optimize grammars for analysis or for synthesis, especially considering that the hardest part of translation is getting a good analysis of the source language. Furthermore, it seems difficult to develop an interlingua on a large scale.

LMT uses a syntactic transfer method, although the source analysis trees contain information that is often called "semantic". Terminal nodes are *word sense predications*, which consist of word senses together with the arguments they should have in logical form. Deep grammatical relations are shown in these arguments.

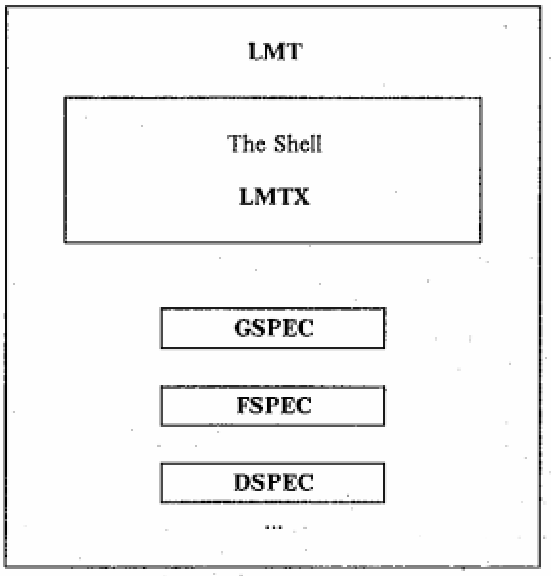
The next section, *Overview of an LMT system*, describes the organization of **LMT** from two points of view. The first is the static organization of the system, in terms of its division into modules, with an emphasis on which parts are target-independent and which are target-dependent. The second is the dynamic organization. A diagram is given, outlining the steps of translation and showing which modules are involved in each step. The remaining sections of the paper constitute an expanded exposition of the steps of translation: *Source/transfer lexical analysis*, *Source syntactic analysis*, *Transfer*, *Syntactic generation*, and *Morphological generation*.

[1] The abbreviation is for "Logic-based (or Logic-programming-based) Machine Translation." The system is written entirely in Prolog.

[2] I wish to thank Susanne Wolff, Arendse Bernth, and Nelson Correa for work on LMTG, LMTD, and LMTS, respectively. LMTF is a collaborative effort with the KALIPSOS team of the IBM Paris Scientific Center (Fargues et al. 1987), with work especially by Eric Bilange. LMTD is being developed in collaboration with IBM European Language Services, Copenhagen. LMTP is being developed by Paula Newman's group at the IBM L.A. Scientific Center.

2 OVERVIEW OF AN LMT SYSTEM

An LMT system consists of the target-language-independent subsystem **LMTX** plus one or more target-language-specific subsystems *T***SPEC** for target languages *T*.



The shared subsystem **LMTX** has three main components: (1) **MODL**, the English analysis grammar with supporting procedures, (2) **MODMOR**, the source/transfer morphological component, and (3) **LMTSYN**, the component dealing with target-language-independent, syntactic aspects of translation. These three modules are used in a runtime **LMT** system. In addition, **LMTX** contains a module **LMTUTIL** of utility procedures dealing mainly with preparation of lexicons.

A target-specific subsystem *T***SPEC** consists of three components: (a) *T***LEX**, the source/transfer lexicon, (b) *T***SYN**, the target syntactic component, and (c) *T***MOR**, the target morphological component. (In the actual names for a specific target language, *T* is replaced appropriately for the target language, as in **DLEX** for Danish.)

The shell **LMTX** contains about 2800 rules (before various compilations, which produce more clauses) and constitutes about 80% of **LMTG-GLEX**, for example.

LMTG was tested on a 500-sentence corpus from a computer manual, and was able to translate 95% of the sentences in an *understandable* way (understandable to a native German speaker without consulting the source). The processing time for this corpus was about 20 milliseconds per word, using VM/Prolog (Gillet) as a Prolog interpreter on an IBM 3081.

2.1 The target-independent subsystem LMTX

The English analysis grammar is written as a Modular Logic Grammar (McCord 1985, 1987), and the **MODL** component includes the MLG rule compiler, as well as this grammar. The grammar is compiled into Prolog when **MODL** is loaded. Aside from these subcomponents dealing with syntax, the **MODL** component contains the following: (1) an interface between the syntax and *T***LEX** lexicons which is called the *lexical compiler* (discussed in Section 3), (2) rule compilers for other rule subsystems of **LMT**, as part of a general procedure `gconsult` that consults Prolog files and compiles special rules (MLG rules and others), (3) a semantic interpretation system, (4) the top-level (driver) procedures for the whole system, (5) the tokenizer, and (6) many utility procedures.

The **MODMOR** component of **LMTX** deals with morphological analysis of English based on a state transition algorithm (McCord and Wolff 1987). **MODMOR** also deals with synthesis of new lexical transfer entries: When an English word is derived morphologically from an English base word, the transfer entries for the base can often be converted correspondingly into transfer entries for the derived word. **MODMOR** handles the target-language-independent aspects of this synthesis (mainly top-level bookkeeping procedures); but low-level, target-specific rules for transfer synthesis exist in the various target morphological components *T***MOR**. **MODMOR** contains the interface to these *T***MOR** procedures; and the proper target language is chosen automatically at runtime on the basis of a control predicate specifying the current target language.

The **LMTSYN** component of **LMTX** contains (1) the top-level procedures for translation, (2) the transfer system, (3) the basic algorithm for syntactic generation of the target language, along with several utility procedures, (4) the top level of inflectional generation, with an interface to target-specific inflectional systems in the components *T***MOR**, (5) procedures for producing suitable output character strings for translated text, (6) procedures for lexical compiling of transfer entries, (7) a rule compiler that creates interface rules between target-independent and target-dependent predicates, and (8) a system for keeping statistics and evaluating translations. Aside from the rule compiling mentioned in (7), there is minor rule compiling done (by `gconsult`) for the transfer rules and the top-level inflectional rules of **LMTSYN** (which have an abbreviated external format). It is interesting that the transfer system for **LMT** (except for specific lexical transfers) is essentially all target-independent (and hence is contained mainly in **LMTSYN**).

2.2 The target-dependent subsystem *T*SPEC

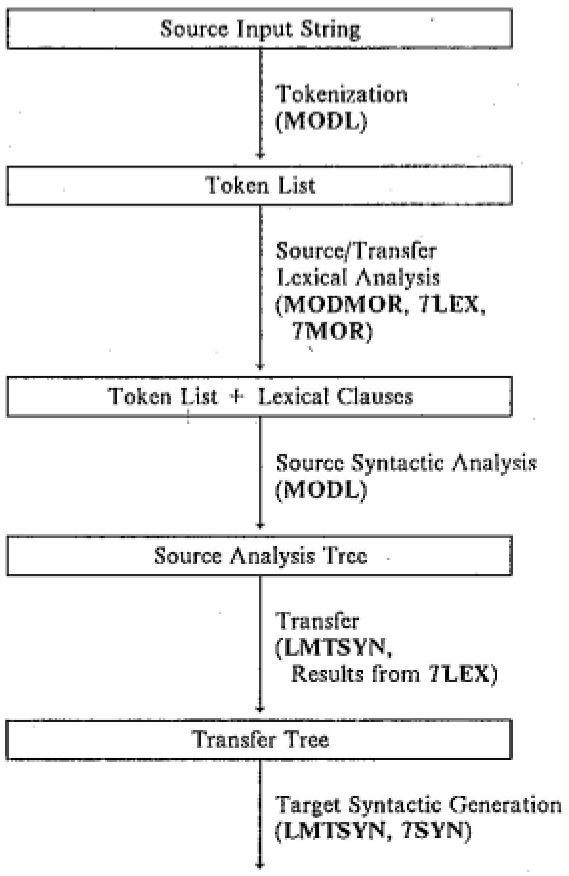
The *T***LEX** component of a *T***SPEC** subsystem, as indicated above, is a *source/transfer* lexicon: It is indexed by source (English) words, and contains (1) monolingual English information, (2) transfer information (for transfer to the target language *T*), and (3) some monolingual target information, stored in transfer elements. All the lexicons *T***LEX** (for different *T*) have the same external format. Lexical entries from different *T***LEX**s can coexist in the runtime system and can be distinguished (as to the proper target language) by means of VM/Prolog predicate prefixing: Lexical clauses, when read in, are prefixed with the name of the appropriate target language. Lexical compiling of *7***LEX** entries is done at runtime, as specific words of the source text are encountered.

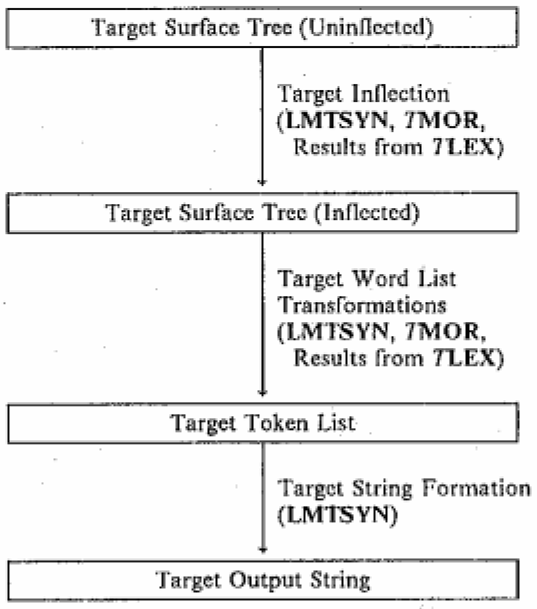
Syntactic target-language generation is done by a transformational grammar. The *7***SYN** component of *7***SPEC** consists of the transformations themselves; these are controlled by the target-independent syntactic generation algorithm from **LMTSYN**. *7***SYN** transformations are expressed in a convenient external format which is compiled (by `gconsult` as *7***SYN** is read in) into Prolog clauses for a procedure `transform`. So that different transformational systems can co-exist, `transform` is prefixed with the name of the target language.

The target morphological component *7***MOR** deals mainly with (1) inflectional generation of target language word forms, and (2) derivational generation of transfer entries. These two subcomponents of *7***MOR** have interfaces from **LMTSYN** and **MODMOR**, respectively, as indicated above.

2.3 Dynamic organization

The following diagram shows the steps of translation in **LMT**. The items in the boxes are the data structures involved. The items beside the arrows show the names of steps and which modules are called.


Source Input String
Tokenization (MODL)
Token List
Source/Transfer Lexical Analysis (MODMOR, 7LEX, 7MOR)
Token List + Lexical Clauses
Source Syntactic Analysis (MODL)
Source Analysis Tree
Transfer (LMTSYN, Results from 7LEX)
Transfer Tree
Target Syntactic Generation (LMTSYN, 7SYN)


Target Surface Tree (Uninflected)
Target Inflection (LMTSYN, 7MOR, Results from 7LEX)
Target Surface Tree (Inflected)
Target Word List Transformations (LMTSYN, 7MOR, Results from 7LEX)
Target Token List
Target String Formation (LMTSYN)
Target Output String

3 SOURCE/TRANSFER LEXICAL ANALYSIS

English text to be translated is first separated into sentences[3] and text formatting commands. A sentence to be processed is tokenized, and its words are looked up in the appropriate *7***LEX** lexicon in combination with morphological analysis. This involves calling the morphological procedures in **MODMOR**, plus transfer synthesis rules from *7***MOR**. Thus each word gets a (possibly derived) lexical analysis (unless the lexical/morphological system does not cover it). Such a lexical analysis, being based on the compact external format of *7***LEX**, is then converted by the lexical compiler (discussed below) into internal-form clauses convenient for the syntactic components. These clauses are asserted into the Prolog workspace, but are retracted after the sentence is processed.

A source/transfer lexicon *7***LEX** is of course dependent on a target language *7*, but nearly all of the procedures that process it, outlined above, are target-independent (in **LMTX**).

Let us look at some details of the format of a lexicon *7***LEX** (for more details, see McCord and Wolff (1987)). The lexicon consists entirely of Prolog unit clauses of the form `W < A`, where `W` is an English word and `A` is the *analysis* of `W`. (There is only one such clause for a given word `W`.)

A sample entry in **GLEX** (the English-German lexicon) for the word *view* might be

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

[3] Sentence fragments are handled also, but we use the term "sentence" — referring to text strings separated by end-of-sentence symbols, etc.

The first line contains two source (English) analysis elements, showing that *view* is a verb with an object slot and is a noun with a complement slot filled by an *of*-PP. The second line contains two transfer elements corresponding to the source elements (in general, a source element could have more than one corresponding transfer element). The `tv` ("target verb") element shows that the verb translates into *betrachten*, and the German complement corresponding to the English object slot gets the accusative case. The `tn` element shows that the noun translates into *Ansicht*, with complement taking the genitive case.

Note that there is target-language morphological information in these transfer elements. The German verb is shown to be an inseparable-prefix verb, and the noun's gender and declension class are specified. Such information is given in a compact form, and its specification here has the advantage that only one lexical look-up is needed for processing *view* and its translations. Also, it is easier for a person creating the lexicon to see all of the relevant information in one place.

The elements of word analyses (such as the `v`, `n`, `tv`, and `tn` elements above) can be of several different types. Eleven different parts of speech are allowed for source elements, with corresponding transfer elements, and there are eleven types for showing irregular inflections of source words.

There are also *multiword* types corresponding to most of the parts of speech. For example, in order to translate *take care of X* into *sich um X kümmern*, the word *take* could contain the multiword elements:

```
take < ... < mv(=.care.of,obj)
      < tmv(pc(um,acc),sich#kümmer).
```

The examples shown so far exhibit no semantic type conditions; but semantic type requirements (given as arbitrary Boolean combinations of simple types) can optionally be associated with complements — both in source elements, for word sense selection, and in transfer elements, for target word selection. For example, the following entry for *eat*

```
eat < v(obj) < tv(nom:human,acc,ess)
    < tv(nom:(animate&¬human),acc,fress).
```

can select *essen* or *fressen* as the translation according as the subject is human or is animate and not human.

When an English word is not found in **7LEX**, it may nevertheless be derived by the morphological procedures in **MODMOR** from a word that can be found. For example, *reuseable* might be derived from *use*, with morphological structure `(re.b(use)).able`. (The `b` indicates the base.) Suppose the target language is German, and the entry for `use` is

```
use < v(obj1) < tv(acc,ver+wend)
    < n(nobj) < tn(gen,verwendung).
```

Then **MODMOR** manages the action of the affixes `re` and `able` on this analysis, to produce a derived analysis of `reuseable`:

```
((re.b(use)).able) + adj
<  ((re.b(use)).able) + tadj('wieder verwendbar').
```

The morphological structure is shown along with each analysis element, since it can vary in general (as with `resaw`).

This morphological derivation illustrates the division of labor between **LMTX** and **7SPEC**. All of the work is in **MODMOR** (⊂ **LMTX**) except for the specific, low-level rules for affix action on transfer elements. This is "dispatched" by **MODMOR** to the proper **7MOR** module in the following way. **MODMOR** contains a general rule for the action of the suffix `able` on a transitive verb transfer element, roughly like the following:

```
suffixop(able,TV,tadj(TargetAdj)) ←
  transitiveTransferElement(TV) &
  lastarg(TV,TargetVerb) &
  tsuffixop(able,tv,TargetVerb,TargetAdj).
```

Thus the problem is reduced to the call to `tsuffixop`, which lets `able` act on a target verb to produce a target adjective.

This still does not mention a specific target language. The "target dispatching" is done by clauses for `tsuffixop` of the following form:

```
tsuffixop(Suffix,Type,Word1,Word2) ←
  target(german) &
  gsuffixop(Suffix,Type,Word1,Word2).
tsuffixop(Suffix,Type,Word1,Word2) ←
  target(french) &
  fsuffixop(Suffix,Type,Word1,Word2).
 ...
```

Here `target` is a "control" predicate, which the user can set to the desired target language. The definition of `gsuffixop` is in **GMOR**, `fsuffixop` is in **FMOR**, etc. For example, the `gsuffixop` clause for `able` is roughly like:

```
gsuffixop(able,tv,TVerb,TVerb+bar).
```

There are many cases of target dispatching clause sets in **LMT**, like those for `tsuffixop`. Such clause sets are created by a rule compiler in **LMTSYN**, which is given the names of the target languages and the predicates to be dispatched (without the initial `t`) and their arities. The rule compiler names the target-specific predicates by prefixing the first letter of the language names, and prefixes the dispatched predicate with a `t`.

Once a (possibly derived) analysis for an input word is found, the analysis is given to the lexical compiler to be converted into internal-form Prolog clauses. These clauses are more efficient for the syntactic components to use because they are put into a standard form obtained by interpreting abbreviations allowed in the external format. For example, a `v` (verb) analysis element in most general form can specify a word sense name, but if this is omitted (as in all the examples above), it is taken to be the index word. Semantic type requirements on verb complements are expanded into combinations of isa goals, which can be executed by Prolog against an isa hierarchy.

Let us look at an example of lexical compilation. The tv element for the *eat* → *fressen* transfer given above is compiled into the clause:

```
tverb(eat(nom:XS:XF,acc:YS:YF),fress)  ←
   isa(XS,animate) & ¬isa(XS,human).
```

Here the first argument of tverb is the word sense predication for eat, which is produced by the lexical compiler from the v element for eat. This will appear in the terminal node for eat in the syntax tree and hence will be available for transfer.

In general, the arguments of a word sense predication for an open-class word are of the form X:Sense:Features where X is the *marker variable* associated with the complement corresponding to this argument,[4] Sense is unified with the word sense name for the head of the complement (like man1), and Features is unified with a term representing the syntactic features of the complement.

The tverb clause for eat above accomplishes two things: (1) It unifies the marker variables of the complements with German syntactic cases, and these are used to determine the cases marked on these complements, and (2) the isa goals make semantic requirements on the subject complement. Marker variables are like pointers into complements, and are in general used to pass information from a head word to its complements. For example, a transfer element can pass a "rule switch" to a clausal complement — a flag that triggers a transformation in *T*SYN on this complement.

4 SOURCE SYNTACTIC ANALYSIS

As indicated earlier, the English analysis grammar is written as a Modular Logic Grammar (MLG). This has been reported on in previous work, (McCord 1982, 1985, 1987, 1988), but the main features of the grammar will be described here briefly. The grammar is written essentially independently of the task of translation,[5] but there are a very few switches in the grammar that do concern the **LMT** application, even referring to target languages or classes of them, and these will be discussed in this section.

MLGs are variants of DCGs (Colmerauer 1978, Pereira and Warren 1980), having several extra ingredients in the syntax rules that allow grammars to be more compact, and having a separate semantic interpretation component of a certain type (dealing with problems of generalized quantifier scoping). The MLG syntax rule compiler takes care of analysis structure building, so that the grammar writer does not have to bother with the bookkeeping for this. Syntactic analysis trees can be built in a first pass, with logical forms being built in a second pass — or optionally logical forms can be built in a single pass by interleaved calls to the semantic interpreter added to syntax rules by the rule compiler. The two-pass option is used in **LMT**, since transfer is done from the syntax trees.

MLG syntax rules are compiled into Prolog clauses in such a way that parsing follows Prolog's native execution scheme and is top-down with backtracking, etc. There is a fairly good treatment of left-recursive constructions in MLGs. An ingredient of the syntactic formalism, called the *shift operator*, is interpreted by the rule compiler to build left-embedded syntactic analyses even though right-recursive rules are used.

The automatically built syntax trees are like derivation trees, but differ from these in three ways. (1) Nodes are not built for the expansion of all non-terminals, but only for those declared as *strong* non-terminals. (2) The shift operator produces a deviation from a derivation tree. (3) The terminal nodes are *logical terminals*, which appear on the righthand sides of syntax rules (in addition to ordinary word-string terminals).

Logical terminals are the building blocks for logical form analyses. They are terms of the form Op-LF where LF is a logical form, usually a word sense predication, and Op is a term called an *operator*, which determines how LF operates on (modifies) other logical terminals in building up a logical form analysis. In **LMT**, the word sense predications produced by the lexical compiler from *T*LEX entries appear in logical terminals in the **MODL** grammar, hence appear in terminal nodes of syntax trees.

The automatic structure-building scheme of MLGs makes it easier to develop useful augmentations of the rule compiler. There is a metarule system for handling coordination and *bracketing* — constructions with paired symbols like parentheses and text-formatting font-change symbols. Also there is a limited tabular parsing facility for parsing input phrases that are not complete sentences.

The **MODL** grammar has a fair amount of lexicalism in its techniques, because of a systematic use of slot-frames that originate in lexical entries. Complements of open-class words are analyzed by general slot-filling rules. Slot-frame manipulations are used, for example, to handle passive constructions. Preference heuristics in the sense of Wilks, Huang, and Fass (1985) for attachment of PPs look at slot-frames and prefer slot-filling over adjunct modification. Semantic type tests (for ambiguity resolution) can be done during parsing, based on semantic type conditions associated with slots (see Section 3 above).

Let us look at the switches in **MODL** that are concerned with the translation task, and in particular with multi-target translation.

An example is the treatment of relative clauses and other noun postmodifiers. When logical forms are being built (and the translation task is not "on"), the marker variable for the relative NP (usually a relative pronoun) of

[4] The marker variable is unified with the main logical variable for the complement used in building logical forms. For example, if the complement is an NP with head man having logical form man1(X), then the marker variable is X.

[5] The MODL semantic interpretation system produces logical forms in the logical form language LFL (McCord 1985, 1987). This facility is being used by Bernth (1988) in a discourse understanding system.

a relative clause should be unified with the marker variable of the modified noun phrase.[6] However, when **LMT** is "on", we do not want these marker variables to be unified, because marker variables are used to control target syntactic features like cases, and the case of the relative NP is independent of the case of the modified NP. There is a predicate `linkmarker(X,Y)` that controls this. Its two arguments are the typed marker variables[7] associated with the modified NP and the relative NP. If **LMT** is off, then `linkmarker` simply unifies `X` and `Y`. If **LMT** is on, it allows the actual marker variables to be independent, but it does unify the remaining portions of `X` and `Y`. (Thus a semantic type requirement by the slot filled virtually by the relative NP will in fact be made on the noun modified by the relative clause.)

Another example of a switch, having to do with classes of target languages, is the treatment of noun compounds. **MODL** has a fairly general treatment of noun compounds, allowing left- and right-branching structures (by use of the shift operator) according to the various ways members of the noun compound modify (or *attach* to) other members. A procedure `attach` controls this; it is given the typed marker variables of the two subcompounds, as well as the slot-frame of the one to the right. One way of attaching is for the first subcompound to fill a slot in the second one, as in *file management*, so `attach` can unify the marker of the first with the marker in a suitable slot of the second, and return the reduced slot-frame. For producing logical forms, exactly this is done. But there is a variant `nlinkmarker` of `linkmarker` that does something different when the target language is a language like German or Danish where English noun compounds are translated into more or less isomorphic conglomerations of nouns. For such target languages, and for noun-noun modification (as opposed to adjective-noun modification), `nlinkmarker` does the same thing as `linkmarker` above, allowing the two marker variables to be independent. (But the slot-frame of the modified compound is still reduced.) The transfer component recognizes the independence of these marker variables, and unifies all those of the premodifying nouns with a special "case" symbol `comb` (for *combining form*), which causes the morphological generation rules to inflect the noun premodifier as a combining form (as in *command name* → *Befehlsname*).

For translation to Romance target languages, English noun compounds must often be unfolded to NP-PP combinations; and in this situation, `nlinkmarker` does unify the marker variables. When the modified noun is transferred, its complement case requirements can include specific prepositional cases, which are unified with the marker variable of the premodifier. Thus the proper prepositional case is marked on the premodifier, and a general "unfolding" transformation in *7SYN* creates the appropriate PP out of the premodifier.

A final example of a switch related to choice of translation target is the treatment of adjective phrase complements of linking verbs. In French and Danish, such an adjective complement must agree in gender and number with the subject of the verb, whereas in German the adjective is uninflected. Such verbs are distinguished in **MODL** by having a slot `predcmp`. The filler rule for `predcmp` calls another linking procedure which (currently), for all targets but German, makes a suitable unification of subcomponents of the markers of the subject and the complement, namely the subcomponents of the feature components that specify number and gender. In the case of German, it is desirable to leave this subcomponent unbound, because the German morphological component leaves an adjective uninflected if its gender feature is unbound.

5 TRANSFER

The transfer system of **LMT** converts an English syntactic analysis tree into a tree which is basically isomorphic but which has contents appropriate for the target language. The terminal nodes are target-language words in base form together with feature structures that determine correct inflections in the target language. The non-terminal nodes are feature structures that have two basic purposes: (1) They serve as vehicles for passing information up and down the tree during transfer. (2) During syntactic generation, transformations (from *7SYN*) do pattern matching on these nodes as part of their tests of applicability.

As mentioned in Section 2.1, most of the transfer algorithm is target-independent — except for the low-level word transfers compiled from *7LEX* entries. There are three kinds of transfer rules, the first two of which are target-independent: (1) *non-terminal transfer rules*, (2) *terminal transfer rules*, and (3) *word transfer rules*. We will describe these first, and then describe the transfer algorithm that uses them.

The simplest form of a non-terminal transfer rule is just

```
SourceLabel ---> TargetLabel.
```

Here the lefthand side is a node label for the English syntactic analysis tree, created automatically by the MLG rule compiler.[8] A rule of this simple type is

```
s(Infl,*,*,*) ---> vp(ind:s,Infl:*,nil).
```

This transfers a source `s` (sentence) node label (only the inflection argument is relevant for transfer) to a target `vp` structure, which is used for any phrase whose head is a

[6] Thus, in the noun phrase *the man that John said Bill saw*, the logical variable associated with *man* eventually gets unified with the object variable of *saw* since the relative pronoun fills the corresponding slot.

[7] These are marker variables together with the sense and feature terms as described at the end of the previous section and also with a term representing a test on the sense and/or feature term.

[8] Such a node label is taken normally from the strong non-terminal on the lefthand side of the MLG syntax rule that is expanded in creating the node. It consists of the non-terminal name, as a functor, together with its (instantiated) *feature* arguments. The feature arguments are the first *n* arguments, where *n* is specified along with the non-terminal in the declaration of strong non-terminals for the grammar.

verb. (We will not give the general form of a target **vp** structure here.)

Non-terminal transfer rules can also look at a kind of context, in order to pass information around the tree, namely at (the label on) the mother node. Such rules are written in the form

```
SourceLabel ---> TargetLabel % MotherLabel.
```

The **MotherLabel** is the *target* feature structure on the mother node; transfer works top-down, so this is available. Most non-terminal transfer rules use the mother context. An example of this sort is needed for transferring the label on a participial clause modifier of a noun phrase (as in *the file created by the user*), where it is important (for some target languages) to link the person-number-gender feature on the mother NP with the inflection of the participle.

Finally, non-terminal transfer rules (of either of the above types) can have a condition **←Cond** added at the end. A simple rule compiler converts any non-terminal transfer rule into a clause for the predicate

```
tranlabel(SourceLabel,TargetLabel,MotherLabel)
```

with a condition, if given.

The second kind of transfer rule is a terminal transfer rule. The purpose of such a rule is to take a terminal node of the source analysis tree, namely an MLG logical terminal, into a terminal of the transfer tree. Terminals of the transfer tree are of the form **BaseWord+Features** where **BaseWord** is a target language word or multiword in base form (or, in a very few cases, symbols that are treated specially by syntactic or morphological generation) and **Features** is a feature structure to be used by morphological generation for inflecting **BaseWord**. Recall that a logical terminal (in the source tree) is of the form **Operator-LogicalForm**. Transfer ignores the operator component. The logical form component is normally a word sense predication; in fact transfer zeroes out all logical terminals for which this is not true. Obviously, a logical terminal (in the source tree) has no features marked on it directly, but the target label for its mother is available to use. (This would be produced by a non-terminal transfer rule acting on the source label of the mother.)

The most general kind of terminal transfer rule is of the form

```
EPred%MotherLabel -->> TargetWord+TargetFeatures
   ← Condition.
```

(Here the rule symbol -->> has higher precedence than ←.) The left operand of -->> is the logical form component **EPred** of a logical terminal, together with its mother label.

An example is

```
EPred%vc(T,Infl,C) -->> TVerb+vc(T,Infl1,C)
  ← cons(*.X.*,EPred) &
     tverb(EPred,TVerb) &
     svagree(Infl,X,Infl1).
```

This is one of the rules for transferring a verb terminal. The mother label (the **vc** structure) is used almost unchanged for the target feature structure, but there is a (possible) adjustment of its inflectional feature (**Infl**) by **svagree** ("subject-verb-agree"), to make sure that it agrees in person and number with the target language subject (which could have features different from those of the English subject). There are other verb terminal transfer rules, dealing for example with the case when the target subject does not correspond to the English subject (as in *like* → *gefallen* in **LMTG**), dealing with passives, etc.

If the target feature is just to be taken as the (target) mother label for the source terminal, then the target feature need not be specified on the righthand side of the -->> rule:

```
EPred%MotherLabel -->> TargetWord
   ← Condition.
```

A simple example is:

```
EPred%pp -->> TPrep
   ← tprep(EPred,TPrep).
```

Also, the condition and/or the mother label may be omitted if they are not needed because of a special form for the terminal itself, as in the transfer of special symbols like numbers.

Again, a simple rule compiler translates all of these forms into a standard form, a clause for

```
tranword(EPred,MotherLabel,
         TargetWord,TargetFeatures).
```

As will be noted from the preceding examples, terminal transfer rules (internally, clauses for **tranword**) generally call transfer rules of the third kind, the target-specific word transfer rules (like those for **tverb**) which are compiled from **7LEX**. But most of the terminal transfer rules have some non-trivial preparatory work to do before calling these low-level rules, and this is done in an essentially target-independent way (in a couple of cases target dispatching is used).

Now that we have described the types of transfer rules, we can specify the transfer algorithm as a whole. It works in a simple top-down, left-to-right manner. The transfer procedure,

```
transfer(SourceTree,MotherLabel,TargetTree),
```

is recursive and keeps track of the target label, **MotherLabel**, for the mother of the current tree (**SourceTree**) being transferred. In the top-level call of **transfer**, this is just a symbol **top**. There are two cases:

1. If **SourceTree** is non-terminal, then **transfer** calls **tranlabel** (*i.e.*, the non-terminal transfer rule system) on the label of **SourceTree**, with **MotherLabel** also as input, getting a target label for **TargetTree** as output. Then **transfer** runs through the list of daughter trees, calling itself on each of them, with the label on **TargetTree** (just obtained) as the mother label, thus getting the list of daughter trees for **TargetTree**. (In the processing of the daughter list of **SourceTree**,

certain terminals not relevant to translation are ignored.)

2. If `SourceTree` is a terminal (a logical terminal containing an English word sense predication), then `transfer` calls `tranword` (*i.e.*, the terminal transfer rule system) with the English word sense predication and the mother label as input, and obtains the target word with its feature structure as output.

The basic idea in making most of the transfer rules target-independent is to include enough information in target feature structures to make them useable for all the target languages currently under consideration. It seems to be possible to do this with reasonably simple structures.

Let us look at an example of target feature information that is useful for a class of target languages, but can be ignored for other target languages. This concerns the treatment of adjective inflections and articles. In both German and Danish, there is a distinction between a strong and a weak declension for adjectives. For premodifying adjectives in a noun phrase, the declension depends on the kind of determiner (and the presence of one) in the noun phrase. Furthermore, in Danish the definite article is enclitic on the head noun if there are no adjectives and no restrictive postmodifiers present.

These interactions between determiners and adjectives in German and Danish are handled by a subcomponent of the noun phrase feature structure — the *premodifier shape* feature.[9] All of the noun premodifiers, including the determiner, share this feature by unification. (For the required unifications, the "mother context" in non-terminal transfer rules is important.)

The premodifier shape feature is a term of the form `St:A`. The subterm `A` is bound to the atom `a` by the non-terminal transfer rules if an adjective premodifier or a restrictive postmodifier is present (otherwise `A` gets bound to `0`). Thus `A` is relevant to the Danish noun phrase; it is ignored for the other languages currently considered, but would be relevant for other Scandinavian languages. The subterm `St` has to do with the strong/weak choice for adjective declensions. Although the unifications of `St` variables during transfer are important, these are not actually bound during transfer, but during morphological generation (by rules in **GMOR** or **DMOR**). In both German and Danish, `St` can get the value `st` or `wk` according as adjective premodifiers should receive the strong or the weak declension. In the case of Danish, `St` can also get the value `clitic` (during morphological generation), when it is appropriate to have an enclitic definite article. More details (for Danish) will be given in Section 7. An interesting point is that the Danish definite article is treated without using transformations (from the syntactic generation component).

[9] There are only three other subcomponents, dealing with noun type (common, proper, etc.), case, and person-number-gender.

6 SYNTACTIC GENERATION

Syntactic generation takes the transfer tree and produces a surface structure tree for the target language by using a transformational grammar. As mentioned above, the transformational algorithm is target-independent. It works recursively on the tree. At each level, the algorithm is first applied recursively to the daughter nodes, giving a tree with a new list of daughter nodes. Then transformations are applied to this tree in a loop: Each time through the loop, the first applicable transformation is used (thus the order of the transformations matters). The loop terminates when no transformations are applicable.

The transformations themselves are of course target-specific, coming from *?*SYN. The call to them by the transformational algorithm is as follows:

```
target(Lang) & Lang:transform(Name,OldTree,NewTree).
```

Thus a specific transformation is given by a clause for the predicate `transform`, which is prefixed by the target language. The `Name` argument is just the name of the transformation, used for tracing. The actual transformations (in *?*SYN) are expressed in a special notation that is convenient for doing pattern matching with sublists (of tree constituents), and the rules in this format are compiled into Prolog clauses for `transform`. The transformation rule compiler is of course target-independent. We will not give details here about the format of the transformation rules and the rule compiler, because there is nothing new for these over what was described in previous papers (McCord 1986, 1988).

We should comment that generally a ?SYN component is relatively small, because a principle of LMT is to get as much right as possible during transfer. Currently GSYN has only 44 transformations.

Some MT systems apply a target-dependent system of transformations to source analysis trees *prior* to transfer, as well as a system of target-dependent transformations after transfer. The LMT scheme, with most of the rules up through the transfer step being target-independent (and with transformations applied only after transfer), seems better suited for a multi-target translation system.

7 MORPHOLOGICAL GENERATION

This last step of translation in LMT takes the output tree from syntactic generation and produces the character string representing the target sentence. There are actually three substeps: (1) the application of inflectional procedures to the terminal nodes of the tree, producing an *inflected tree*, (2) the application of *target word list transformations* to the inflected tree, producing a list of words and special symbols, and (3) the production of the final character string, taking account of punctuation, text formatting symbols, etc.

The first substep is the principal one. There is an outer layer of target-independent rules handling inflections. The main rules of this sort have heads of the form

```
BaseWord+Features ->>> InflectedWord.
```

Ultimately, these rules call target-dependent inflection procedures in *T*MOR; but there is some recursion in ->>> rules for handling compound words (perhaps of more than one part of speech), and there are various "tidying-up" operations for getting the inflectional features in good order for *T*MOR. For example, subject-verb agreement for coordinated subjects is treated here.

The target-dependent procedures called by ->>> rules are mainly just the inflection procedures for specific parts of speech, like

```
tverbf(Verb,Inflection,VerbForm)
```

for verbs. These are "target-dispatched"[10] to procedures like gverbf in **GMOR** and fverbf in **FMOR**.

It could be said that a target inflectional system (in *T*MOR) is the most independent part of an **LMT** system; the task of finding inflected forms from base forms and inflectional features is rather theory-independent. **GMOR** and **DMOR** were written locally, but most of the French inflectional morphology for **LMTF** is that of the **KALIPSOS** system (Fargues et al. 1987).

In Section 5 we mentioned the use of the premodifier shape feature St:A in treating determiner, adjective, and noun inflections for Danish. If a definite article is present and A can be bound to 0 (meaning that no adjective premodifiers or restrictive postmodifiers are present), then ddetf can both bind St to clitic and return 0 as the inflected form of the premodifying article (zeroing it out). When dnounf sees this feature, it adds the clitic to the noun. The other cases for St can be handled appropriately by ddetf and dadjf, depending on the type of determiner present.

The second substep of morphological generation is to convert the inflected tree into a target word list, applying *target word list transformations*, if possible, on each level. The purpose of these transformations is to do any cleaning up of the translation that is appropriate on the word list level. The main example of interest is the treatment of contractions, as in *in dem Haus* → *im Haus*, or *le homme* → *l'homme*.

The main part of the algorithm for this second substep is itself target-independent. At each level, the daughters of the tree node being converted are first converted recursively to lists and concatenated into a single list, Words. At this point, application of a word list transformation is attempted, by calling a procedure

```
tphrase(Category,Words,Words1).
```

The Category is the principal functor of the label on the tree node being converted. This procedure is target-dispatched to target-specific procedures in *T*MOR (like fphrase in **FMOR**). If the call to tphrase succeeds, then the desired word list for the node is taken to be Words1; otherwise it is Words.

[10] Recall that there is a rule compiler in **LMTSYN** that creates the clauses for switching to the target-specific procedures for the appropriate target languages.

Word list transformations can also be used sometimes to effect idiomatic phrasal translations, by letting phrases translate compositionally and then transforming them. The transformations can appear, in a slightly abbreviated format, in a *T*LEX lexicon, and the lexical compiler converts them into tphrase rules.

The third and last substep of morphological generation (producing the final character string, handling punctuation, etc.) will not be described, although it should be mentioned that it is target-independent.

REFERENCES

Bernth, A. (1988) "LODUS — A logic-oriented discourse understanding system," Research Report RC 13676, IBM Research Division, Yorktown Heights, NY 10598.

Carbonell, J. G. and Tomita, M. (1986) "Knowledge-based machine translation, the CMU approach," in Nirenburg, S. (ed.), *Machine Translation: Theoretical and Methodological Issues*, Cambridge University Press.

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

Fargues, J., Bérard-Dugourd, A., Landau, M. C., Nogier, J. F., Catach, L. (1987) "KALIPSOS Project: Conceptual semantics and linguistics," *Proc. of the Conf. on Artificial Intelligence and Natural Language Technology*, IBM European Language Services, Copenhagen.

Jin, W. and Simmons, R. F. (1986) "Symmetric rules for translation of English and Chinese," *Computers and Translation*, vol. 1, pp. 153-167.

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. (1988) "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. 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.

Pereira, F. C. N. and Warren, D. H. D. (1980) "Definite clause grammars for language analysis — a survey of the formalism and a comparison with transition networks," *Artificial Intelligence*, vol. 13, pp. 231-278.

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.

Wilks, Y., Huang, X-M., and Fass, D. (1985) "Syntax, preference and right-attachment," *Proc. 9th International Joint Conference on Artificial Intelligence*, Los Angeles.