An Object-Oriented Implementation of Machine Translation Systems

Koichi Takeda

Tokyo Research Laboratory, IBM Research
1623-14 Shimotsuruma, Yamato, Kanagawa 242, Japan
e-mail: takeda@trl.vnet.ibm.com

In this paper, we describe an implementation of object-oriented knowledge sources and functions for knowledge-based machine translation, where the meanings of sentences are represented by conceptual "objects." While parsing and generation are viewed as mappings between syntactic and conceptual representations, such functions as paraphrasing, abstraction, and information filtering are implemented as "methods" in conceptual objects. By incorporating set constructors, variables, and functions as "meta" objects, we show that broad-coverage conceptual representations can be compositionally defined for natural language sentences. Specifications for mapping between syntactic and conceptual representations are used to control the paraphrasing process. Our approach is compatible with bidirectional machine translation and massively parallel processing.

1. Introduction

Since the syntactic analysis of natural languages has made considerable progress in the last two decades, the success of machine translation systems appears to hinge on adequate semantic processing for practical domains, which is the key to optimizing the selection of syntactic representations (i.e., to choosing the most acceptable sentences) while preserving the meaning. Knowledge-based machine translation (KBMT) [10, 20] with grammar-based sentence generation is a promising approach, since it satisfies the above requirement as well as having other important properties, including bidirectionality.

Two major problems, however, have to be solved in order to realize practical KBMT systems. First, we have to provide a class of conceptual (semantic) representations whose coverage matches that of syntactic representations, and to create a well-defined mapping between these two kinds of representation. Second, we have to identify the subtle differences among conceptual representations of source and target language sentences, since a conceptual representation of a source language sentence may not be appropriate for generating a target language sentence. We often find that such a conceptual representation either includes overly specific information that cannot be explicitly represented in the target language, or else lacks information critical for selecting correct words and phrases.

The Electronic Dictionary Research (EDR) project[2] in Japan provides lexical coverage for a huge number of words, but not enough mapping specifications for phrasal or sentential representations. The interlingual approach(e.g., Muraki [9]) avoids the problem of inconsistency in source and target conceptual representations, but the parser and generator inevitably suffer from overheads for

creating and interpreting language-independent representations. Interaction with the user [3] and controlled language [13] might solve the inconsistency problem in some cases, but this is unlikely in the translation of bulky, everyday documents, which are the primary target of current machine translation systems. The generative lexicon [15] has excellent descriptive power and logical semantic representation, but paraphrasing (or equivalence checking) for translation might be intractable.

Our approach is to extend the conventional frame-based representation scheme into an object-oriented representation scheme with special "meta" objects, secondary objects, and an exclusive inheritance mechanism. Each concept is represented by an *object* (an instance of a class). By incorporating the semantic classifications of the Longman Dictionary of Contemporary English (LDOCE) [14] into our class hierarchy, we can provide lexical coverage of more than 50,000 words, and our mapping specifications cover a broad range of sentential representations.

A paraphrasing function is implemented as a *method* in objects. When the "paraphrase method" is activated in an object, which may have one or more modifying objects, it creates another object (with recursively paraphrased modifiers) that represents an equivalent or semi-equivalent meaning. This method utilizes only knowledge on domains and target languages, and is not affected by which source language is used to represent the input sentences.

2. Natural Language Class System

A *concept* is a primitive used to represent a specific meaning in the real world. It has a unique name and zero or more *slots* for holding modifiers used to represent a complex meaning. Some slots are used to define is-a and part-of relationships among concepts. Concepts correspond to the *classes* of object-oriented knowledge bases (e.g., Cattell [1]) and to *categories* of the CYC project [6]. We use the two terms *concepts* and *classes* interchangeably. For example, one of the classes expressed by the verb "insert" and the noun "insertion" would be defined as follows:

```
(defconcept  *insert
   (is-a  (value  *action))
   (:agent  (sem *human  *system))
   (:theme  (sem  *physical-object))
   (:goal  (sem *location  *physical-object)))
```

This is the definition of the class *insert with an is-a link to a class *action, and three slots - :agent, :theme, and :goal. The (value . . .) facet shows an actual filler of the slot, while the (sem ...) facet shows the *selectional restrictions* on the slot. An *instance* of a class is a specific representation of the meaning of a word or a phrase in a sentence.†A conventional definition of a plain ontology, however, turns out to be insufficient for establishing a framework for conceptual mapping. *Exclusive inheritance, meta classes,* and *secondary classes,* described below, are thus introduced to the set of concept definitions, and the extended set of concept definitions is called the Natural Language Class System (NLCS). [18]

2.1 Exclusive Inheritance

A class inherits each slot definition from its *ancestors* (superordinate classes), that is, from the transitive closure of the fillers of the is-a slot, unless the slot is redefined. When a class has

† We use a hyphen and a number following a class name (*insert-l, *diskette-l, . . .) when it is necessary to show instances explicitly. We often identify class names and instance names, and simply say "*X" instead of "an instance of a class *X" when it is not confusing.

more than one immediate ancestor, we can restrict its inheritance to be *exclusive* rather than the usual *multiple* inheritance, which has been employed in many object-oriented systems. Exclusive inheritance allows a class to inherit slot definitions from only one of its immediate ancestors. The idea behind exclusive inheritance is to realize identity of verbal and nominal word senses without mixing the slot definitions of both. For example, most verbs have nominal counterparts in a natural language, such as "insert" and "insertion," and such a pair usually shares slot definitions (:agent, :theme, and :goal) and selectional restrictions. There are some exceptions; for example, "insert" inherits tense, aspect, and modality from its "verbal" ancestor but, unlike "insertion," it does not inherit cardinality, ordinal, and definiteness (that is, the quality of being indefinite or definite) from its "nominal" ancestor.††

2.2 Meta Classes

There are three rneta classes in NLCS - *var, *set, and *fun - to represent classes not included in the normal ontological hierarchy.

The first, *var, is a variable that ranges over a set of NLCS classes, which are *constants*. Pronouns and question words in natural languages usually carry this kind of incomplete concept. A schematic definition of *var is

```
(defconcept  *var  (domain)  (range)  (referent))
```

where *domain* specifies an initial set of classes that an individual instance of ^{x}var ranges over, *range* specifies a set of (possibly complex) instances that satisfy inter/intra-sentential constraints on *var, and *referent* specifies an instance that *var actually refers to in the context. Neither selectional restrictions on the above slot fillers nor the is-a slot can be predefined for *var.

The second, *set, is a set constructor that can represent a coordinated structure in natural languages. A schematic definition of *var is

```
(defconcept  *set  (type)  (member))
```

whose slot definitions are obvious.

The third, *fun, is a function from NLCS instances to NLCS instances. It captures the meaning of a so-called *semifunctional word.* For example, in some usages, the verb "take" does not really indicate any specific action until it gets an argument such as "a walk," "a rest," "a look." It is therefore well characterized as a function. A schematic definition of *fun is similar to that of an ordinary class except that a "head" class and arguments to the *fun must be explicitly specified (see Section 3.2 for more details.)

2.3 Secondary Classes

Since we heavily use exclusive inheritance, NLCS may lack the ability to organize an ontological hierarchy from various viewpoints, unlike ordinary multiple inheritance. *Secondary classes* are therefore introduced to compensate for this inability. A secondary class only defines a collection of other classes. For example,

---

†† One may claim that the "verbal" and "nominal" distinction of concepts is a syntactically biased definition and that there really should be one *action sub-hierarchy for both verbs and their nominal counterparts. We observed, however, that the expression "an insertion of the diskette" can be identified with "inserting the diskette," but the expression "the insertion of the diskette" should not be. Exclusive inheritance easily settles this kind of problem. Interestingly, "adjective (adverbial)" and "nominal" inheritances (e.g., "thick" and "thickness") also need to be exclusive. The above observation holds for Japanese as well as English.

```
(defvconcept *option
   (def    *math-coprocessor
           *hard-disk *software)))
```

and

```
(defvconcept  *movable-thing
   (def  (include  :property  *movable)))
```

show two types of secondary concept, *option and *movable-thing. The *option is a collection of the classes *math-coprocessor, *hard-disk, and *software. The *movable-thing is a class that includes any instance in which the :property fillers is *movable. Secondary classes differ from *primary classes* (classes defined in the ontological hierarchy) in the sense that they have neither is-a ancestors nor inheritance paths. Each member of a secondary class determines both dynamically.

3. Conceptual Mapping Rules

Conceptual mapping rules define lexical and structural correspondences between syntactic and conceptual representations.† We can view a mapping rule as an *instance creation rule* for a parser as well as a *feature structure creation rule* for a generator.

A *lexical* mapping rule has the form

```
(emap  *install  <=1=>  install  ((CAT  v)  (SUBCAT  trans))
   (:agent  =syn  (SUBJ))
   (:theme  =syn  (OBJ))
   (:goal  =syn  (PPADJUNCT  ((PREP  (*or*  in  into))))))
```

where a transitive verb "install" †† maps to or from an instance of *install. The *structural mapping (=syn)* between three slots (:agent, :theme, and :goal) and grammatical roles (SUBJ, OBJ, and PPADJUNCT) are also defined in this rule. The :agent filler, for example, should be an instance that is mapped from a syntactic subject, SUBJ, of the verb "install." The :goal filler must be a prepositional phrase consisting of a noun with the preposition "in" or "into." The fragments of syntactic feature structures following a lexical word or a grammatical function in a mapping rule specify the minimum structures that must *subsume* feature structures of candidate syntactic constituents. These structural mappings are specific to this instance. A sample grammatical rule that triggers this mapping rule would be:

V → install
(V CAT) = v
(V SUBCAT) = trans
(V FORM) = finite
(V SUBJ AGR NUM) = pl,

where the first two equations are minimally required. Certain syntactic feature values such as *sg (singular)* and *imp (imperative)* are not lexical entries, but they are relevant to conceptual representation. A variant of a lexical mapping rule, such as

```
(emap *singular  <=v=>  sg)
```

† By syntactic structures we mean feature structures of unification grammars such as LFG [4] and PATR-II [16]. Throughout the paper, we use PATR-II notation to describe grammar rules.
†† "Emap" stands for mapping rules for English.

is used to define the mapping between feature values and instances of classes.

The *structural* mapping rule

```
(emap *physical-action  <=s=>
   (:mood  =syn  (MOOD))
   (:time  =syn  (TENSE)))
```

specifies that the slots :mood and :time map to or from the grammatical roles MOOD and TENSE, respectively. Unlike the slot mappings in a lexical mapping rule, these slot mappings can be inherited by any instance of a subclass of *physical-action. The *insert instance defined above, for example, can inherit these :mood and :time mappings.

We are now ready to see the expressive power of meta class mapping and how it can be used to handle various syntactic expressions.

3.1 Pronouns, WH-Words, and Gaps

The pronoun "one" in the sentence "Flip down the red one." will be mapped to the instance

```
(*var-1
   (domain (value *physical-object-1))
   (range (value *lever-1))
   (referent (value *power-switch-1)))
```

where selectional restriction exerted by the verb "flip down" on its :theme slot requires the filler to be *lever (which is-a *physical-object), and the pronoun actually refers to a power switch that appeared in the context. The mapping rule for this pronoun is

```
(emap  *var  <=1=>  one  ((CAT pro))
   (domain  =sem  (*physical-object))
   (:definiteness  =syn  (DET))
   (:property  =syn  (APMOD)))
```

The slot mappings (:definiteness = syn (DET)) for the determiner "the" and (:property = syn (APMOD)) for the adjective phrase "red" are applied to each *range filler,* and we get

```
(*lever-1
   (:definiteness  (value  *definite-1))
   (:property  (value  *red-1)))
```

as a more constrained range filler. Contextual constraints (or an *anaphora resolver)* determine the *referent* filler of *var-1.

Note that *var-1 is so informative that it allows a generator to produce at least three different syntactic representations. The first, "the red one," is just the inverse of the above conceptual mapping. That is, an instance of *var with (domain (value *physical-object)) maps to "one," and the :property and :definiteness modifiers of the range filler map to "the red." The second, "the red lever," is recovered from the range filler alone. The third, "the red power switch" is recovered from the referent filler and the :property and :definiteness modifiers of the range filler. The second and third ones are paraphrases of the first one. We can specify the latter two paraphrases in terms of paraphrasing rules for *var instances.

Mapping rules for *var can be also defined for WH-words and gaps. The WH-word "why" can be mapped to *var by

```
(emap  *var  <=1=>  why   ((CAT  wh))
   (domain  =sem  (*reason)))
```

A gap in a relative clause can be mapped to *var by

```
(emap  *var  <=1=>  @gap)
```

where a referent filler is immediately obtained from the antecedent noun of the relative clause.† It is not possible, however, to get paraphrases from these instances of *var.

Other interesting usages of *var include the handling of unknown words. An unknown word triggers the lexical mapping rule

```
(emap  *var  <=1=>  @unknown
   (string  =syn   (string)))
   (domain  =sem  (*object))
```

which means that the unknown word could be any object whose string slot is filled by the character string of the word. If we have to assume that the unknown word is a verb, an adjective, and so on, we can define similar lexical mapping rules. Inheritance of slot mappings from potential domain fillers such as *object, *physical-action, or *attribute would help us build conceptual representations including instances of such unknown words.

3.2 Semifunctional Words and Idioms

An example of a conceptual mapping rule for "take a look" is

```
(emap  *take-fun  <=f=>  take  ((CAT v)  (SUBCAT  trans))
   (head  =filler  (:theme))
   (:theme  =arg (OBJ ((root look) (cat  n) (num  sg) (det a)))))
```

where the verb "take" must have a singular OBJ "a look" since it is specified as a required argument (=arg equation) in the rule, and the :theme filler should be the conceptual head of this *fun instance. Assuming that the phrase "look at Joe" is represented as (*look-l (:goal (*Joe-l))), we have the conceptual representation

```
(*take-fun-l
 (:mood   (*imperative-l))
 (:theme   (*look-l   (:goal   (*Joe-l)))))
```

which is paraphrasable to

```
(*look-2
 (mood  (*imperative-l))
 (:goal  (*Joe-l)))
```

since the head of the *take-fun-l is the :theme filler, *look-l, and we can evaluate *fun-l to obtain *look-2 by composing all other slot fillers (i.e., the :mood filler) with the conceptual head.†† The following mapping rule maps the expression "help *oneself* to ... (food)" into the object

```
(*eat-l   (:manner   (*free-l))    (:theme   (...))).
```

---

† An empty symbol of a gap is encoded as @gap.

†† To be precise, we also have to apply a *role-changing* rule to the *mental-object "*look-l" to obtain the *physical-action "*look-2."

```
(emap *help-fun <=f=> help ((CAT v) (SUBCAT trans) (prep to))
   (head =sem (*eat (:theme :goal)))
   (:theme =darg (OBJ ((root self) (cat pro))))
   (:manner =sem (*free))
   ((:goal (sem *food)) =arg (ppadjunct ((prep to)))))
```

The *help-fun takes two arguments, :theme and :goal, to be complete. The first argument is a reflexive pronoun, which must agree with the subject of the verb "help,"† but the :theme filler is redundant, and is not mapped to the image *eat ("=darg" stands for a dummy argument.) The :goal filler, which is-a *food, is mapped to the :theme filler of the *eat concept. This is a filler-role-changing rule specified in (*eat (:theme :goal)). Finally, the :manner filler "*free" is inserted when the *eat object is instantiated.

3.3 Coordinated Structures

Coordinated structures are so commonly used in documents that we found as many as 27% of all the sentences in a PC manual included some form of coordinated structure. Syntactic accounts for coordinated structures have been proposed by Kaplan and Maxwell [5], who use a notion of a *set* off-structures briefly mentioned by Kaplan and Bresnan[4]. Since they defined an f-structure for a coordinated structure as a pure set of constituent f-structures, it was not possible to represent, for example, that a coordinated noun phrase has (person 3) (number pl) features as a whole. Therefore, we define a feature structure of a coordinated structure to be a pair $\langle f, \{f_1,\ldots, f_k\}\rangle$, where f is a feature structure that is peculiar to the coordinated structure, including a feature structure of a conjunction, and $f_1,\ldots, f_k$ are feature structures of coordinated constituents. We denote *first(F)* = f and *rest(F)* = $\{f_1,\ldots f_k\}$ for a complex feature structure F = $\langle f \{f_1, \ldots,f_k\}\rangle$. For any simple feature structure F, first(F) = rest(F) = F. For example,

$NP \rightarrow NP_1$ CONJ NP_2

‹NP CONJ› =s ‹CONJ›

‹NP NUM› =s pl

‹NP› ə ‹NP_1›

‹NP› ə ‹NP_2›

generates the feature structure

(((CONJ ((SIGN and) (CAT conj))) (NUM pl)), $\{f_1,f_2\}$)

for NP when $NP_1 = f_1$, $NP_2 = f_2$, and CONJ = ((SIGN and) (CAT conj)). The equation "‹NP NUM› =s pl" defines a complex feature structure F for NP such that NUM of first(F) unifies with *pl*. ††

The structural mapping rule for this coordinated structure is

```
(emap   *set   <=s=>
   (type =syn first(CONJ))
   (member =syn rest()))
```

† This is a grammatical constraint. Imperative sentences are assumed to have "you" as an implicit subject.

†† To be precise, the equations have to be extended in order to specify the unification of a complex feature structure and a simple/complex feature structure. Takeda defined a new equation =s for LFG in addition to = and ə equations[19], where F_1 =s F_2 unifies first(F_1) with F_2 , and $F_1 = F_2$ unifies rest(F_1) with F_2, to allow the same grammar rules to be used for both simple and complex syntactic structures as far as possible.

where *set is always mapped from the complex feature structure F, and first(CONJ) and rest() denote the CONJ value of first(F) and rest(F), respectively. Likewise, we can build instances of *set for the phrases "either A or B," "A, B, then C," "A as well as B," and so on.

Note the difference between coordinated structures and prepositional phrases (adjuncts), although the latter are formulated in terms of complex feature structures in Kaplan and Bresnan [4] as follows:

VP --> V NP NP PP*
(↑ *OBJ*) ∋↓ (↑ *OBJ2*) ∋↓ ↓∋ (↑ *ADJUNCTS)*

However, the complex feature structures in ADJUNCTS † should not be mapped to a *set instance because a possible instance

```
(*set-1 (member (value *tuesday-1 *tokyo-1)))
```

for two adjacent PP's "on Tuesday in Tokyo" clearly confuses the location and :date roles. For this reason, we have another kind of equation > for non-set-forming constituents.†† That is,

VP1 → VP_2 PP
‹VP_1› = ‹VP_2›
‹VP_1 *PPADJUNCT*› > (PP)
where each PP fills a specific role of the head VP.

3.4 Integers, Character Strings, and Derivative Words

Integers and character strings can be arbitrarily large and long. Therefore, two generic lexical mapping rules are defined for these lexical entities. Any integer *X* is mapped to an instance of *integer with the :value slot filled with the integer X. Similarly, any character string "Y" embedded in a sentence is mapped to an instance of *string with the string slot filled with "Y."

Generic mapping rules are also necessary to define consistent mapping rules among derivative words. An intuitive example is a verb and its infinitive and present participle forms, as illustrated below:

```
(emap *insert <=1=> insert ((CAT v) (SUBCAT trans) (FORM infinite))
   (role =sem (*physical-action))
   (:agent =syn (PPADJUNCT ((PREP for) (CAT (*or* n pro)) (PREMOD +))))
   (:theme =syn (OBJ))
   (:goal =syn (PPADJUNCT ((PREP into)))))

(emap *insert <=1=> inserting ((CAT v) (SUBCAT trans) (FORM prsprt))
   (role =sem (*physical-action))
   (:agent =syn (PREMOD ((CAT (*or n pro)) (FORM genitive))))
   (:theme =syn (OBJ))
   (:goal =syn (PPADJUNCT ((PREP into)))))
```

By generalizing this regularity, we can define generic mapping rules for derivative words.

† PP* is a regular expression used to denote any number of PP adjuncts in the right hand side of the rule.
†† This equation is called an "append" operation in KBMT-89, and is inappropriately used for both adjuncts and coordinated structures.

4. Paraphrasing, Abstraction, and Information Extraction

An adjective modifying a noun in English might be a conjugable verb in translation, and the differences between adjective and verb might result in added or missing information in their conceptual representations. This fact implies that we have to provide a set of *conceptual paraphrasing rules* to describe a set of equivalent and semi-equivalent conceptual representations. Practically, these rules should be sensitive to the target language, but not to the source language, since the definition of equivalence among conceptual representations depends on the cultural and pragmatic background of the language in which a translation has to be expressed. An example of a paraphrasing rule is

```
(equiv  (*be-identity
          (:agent  (*X))
        (:theme  (*Y/*person  (:definiteness  (*indefinite))
        (:number  (*V))
        (:property  (*W)))))
        (*Z/*action
        (:agent  (*X))
        (:manner  (*W)))
        (such-that  (humanization  *Z *Y)))
```

where *X, *Y, ... are variables that denote fillers of value facets, *Y/*person specifies *Y to be an instance of any descendant of *person, *be-identity is roughly the verb "be," *humanization* is a relation that holds for pairs such as (*singer, *sing) and (*swimmer, *swim). Intuitively, this rule specifies an equivalence relationship between sentences such as "Tom is a good singer" and "Torn sings well." Obviously, some of the paraphrasing rules are so idiosyncratic that they should be localized to each class (e.g., *be-identity) since no other class needs to know the paraphrasing rules, while general paraphrasing rules (see below) should be defined in the top-level class and inherited by its subclasses.

In addition to the paraphrasing rules mentioned earlier, the following general rules are available for each class. The paraphrasing rules are composed to make various paraphrases of conceptual representations.

- Projection: Map an instance with a filled slot *s* to an instance of the same class with the unfilled slot *s*. Projection corresponds to deletion of a slot s.
- Generalization: Map an instance of a class *X to an instance of one of the ancestors of *X.
- Specialization: Map an instance of a class *X to an instance of one of the descendants of *X.

A projection rule is frequently used in translating English nouns into Japanese ones, as in the following example:

```
diskette     (*diskette (:number (*singular)))
diskettes    (*diskette (:number (*plural)))
a diskette   (*diskette (:number (*singular))
                        (:definiteness (*indefinite)))
the diskettes (*diskette (:number (*plural))
                        (:definiteness   (*definite)))
ディスケット  (*diskette)
```

Here, the four English noun phrases above are usually translated by the same Japanese noun phrase ! (the fifth one), which does not carry any information on :number and :definiteness. Naturally, we have to provide a paraphrasing rule for translation in the opposite direction, so that for any instance of the *object we can obtain appropriate :number and :definiteness fillers.

In general, a bidirectional machine translation system requires that semi-equivalent rules for translation in one direction should be coupled with a way of inferring missing information for translation in the opposite direction. Generalization and specialization rules are complementary and can be paired to become equivalent when a specialization rule for any instance of a class x is unambiguous. That is, without losing any fillers, one can always choose an instance of a subclass y to which x can be uniquely mapped. A generalization from each y to x provides the opposite mapping.

Using these paraphrasing rules, each instance can invoke a *paraphrase method* for interpreting the rules to find instances with (semi-)equivalent meanings. Abstraction and information filtering can also be defined as *methods* for obtaining specific information from given instances. *Abstraction* consists of the following steps:

1. If the caller (instance) of the abstraction method is *important,* then the entire instance, including all the fillers, is *important.*
2. If one of the fillers of the caller is *important,* then the caller and the filler are *important.*
3. Quantifiers, including the instances for "not" and "no," are always *important*
4. Abstraction of an instance is a minimal subset of the instance that includes all the important instances and fillers, and a well-defined mapping to a syntactic feature structure.

The importance is given as a list of pairs of classes $(a_1, b_1),\ldots, (a_n, b_n)$ such that an instance I of a class C is important iff C is-a a_i and b_i is-a C for some i ($1 < i < n$). Similarly, the extraction method, or information filtering, is defined as follows:

1. If the caller of the extraction method is *complete,* then only the relevant portion of the instance is *complete.*
2. If one of the filler of the caller is *complete,* then the caller and the filler are *complete.*
3. Quantifiers, including "not" and "no", are always *complete.*
4. Extraction of an instance is a minimal subset of the instance which includes all the complete instance and fillers, and a well-defined mapping to a syntactic feature structure.

The completeness is given as a list of pairs of a class and slot names $(c_1, s_1), \ldots, (c_n, s_n)$ such that an instance I of a class C is complete iff C is-a c_i and I has non-empty fillers for all slot s (possibly empty) in s_i for some i ($1 < i < n$). Intuitively speaking, abstraction captures a collection of interesting sentences, while extraction captures fragments of specific facts. For example, given importance = {(*dollar, *dollar)}, completeness = {(*soar, (:agent))}, and the sentence "The dollar dropped sharply yesterday, and the yen and mark soared in Tokyo," abstraction returns "The dollar dropped," and extraction returns "The yen and mark soared." If we allow paraphrasing of instances, these two methods can return noun phrases such as "fall of Dollar" since *tense* and *mood* information can be omitted.

†One exception is that deictic noun phrases are translated when the Japanese counterpart "その" is used for the determiner "the."

The object-oriented implementation of these methods will be more advantageous in an massively parallel architecture. Combination of these methods will lead us to new applications of machine translation such as real-time translation of extracts from bulky foreign marketing reports.

4.1 Degree of Paraphrasing and Sentence Generation

An algorithm for paraphrasing is a top-down, recursive call of paraphrase methods to find a conceptual representation that has a well-defined mapping to a feature structure of a target language grammar. Let us view a conceptual representation as a tree and its fillers as subtrees.† The algorithm then consists of the following steps:

1. If a root has no lexical mapping, or there is a slot which has no structural mapping, find a paraphrasing rule to map the root to some other instance. Use generalization or specialization when there is no appropriate paraphrasing rule.
2. Apply step 1 to each subtree. That is, call the paraphrase method of each root of the subtrees. If there is a subtree with no well-defined mapping, paraphrase the subtree to generate another instance.
3. Compose the mapping rules for the subtrees to make an entire feature structure. If this fails, paraphrase the root into some other instance. Start from step 1.

If the algorithm successfully terminates, we have a conceptual representation and its corresponding feature structure. Once the feature structures for the subtrees have been computed, it is not necessary to recompute them, even when the root is mapped to another instance, as long as the filler is unchanged. Similarly, only one of the duplicated instances needs to be traversed.

The degree of paraphrasing is measured in terms of the number N of semi-equivalent rules used in the steps above. If N is 0, the obtained paraphrase is equivalent to the original one, and as N increases, we consider that the quality of paraphrasing becomes worse.

By defining a constant K (possibly comparable to the number of instances in the input representation) as an allowable degree of paraphrasing, and preferring equivalent rules to semi-equivalent rules, we can make the algorithm terminate in order to produce an appropriate feature structure, although we may have to sacrifice some of the original semantic content. A sample result of the paraphrasing for English to Japanese translation is given below. (Two projection rules are applied. Hence, K = 2)

```
Input  Conceptual  Representation

(*install
 (:mood   (*imperative))
 (:theme   (*operating-system
              (:number   (*singular))
             (:definiteness (*definite)))))

Paraphrased Conceptual Representation (K = 2) and Its Feature Structure

(*install
 (:mood (*imperative))
```

† A general graph structure can be converted into a tree by duplicating each node that has more than one parent.

```
 (:theme   (*operating-system)))

((ROOT 導入) (CAT v) (SUBCAT trans) (VTYPE sahen)
 (MOOD imp)
 (OBJ ((ROOT オペレーティングシステム) (CAT n))))
```

4.2 Conceptual Paraphrasing and Sentence Generation

As explained in the previous subsection, paraphrasing is successful only if a well-defined feature structure is recovered from the conceptual representation by the use of mapping rules. This is the advantage of our paraphrasing approach, since it is not necessary to run the grammar in order to test the validity of a conceptual representation. Often the worst case for grammar-based generation is when an input is not a valid conceptual representation, and the large size of a practical grammar forces the generator to take a lot of time merely to fail.

Our generator is based on Wedekind's algorithm [21] modified for PATR-II-type grammars, but the paraphrase method can be used for any other grammar-based generation algorithms, [17,7] since these algorithms can be run with such an input that a partially instantiated feature structure is paired with a conceptual representation (or their logical forms). One important aspect of our generation grammar is that we can handle semantically "non-monotonic" [17] expressions by introducing functional classes. For example, recall the idiom "help *oneself to* ...(food)" in Section 2. If a conceptual representation

```
(*eat-1  (:manner  (*free-1))  (:theme  (*banana-1)))
```

is given, the feature structure input to the generator is either

```
((ROOT  eat)  (CAT v)  (SUBCAT  trans)
 (OBJ  ((ROOT  banana)  (CAT n)))
 (ADVADJUNCT  ((ROOT freely)  (CAT  adv))))
```

or

```
((ROOT  help)  (CAT  v)  (SUBCAT  trans)  (PREP   to)
 (OBJ  ((ROOT  self)  (CAT pro)))
 (PPADJUNCT  ((PREP  to)  (ROOT  banana)  (CAT  n))))
```

Hence non-monotonicity is excluded from the grammar.

5. Prototype Implementation

The NLCS and the paraphrase method are currently implemented by using the FrameKit [11], and are part of a prototype machine translation system called **Shalt2** [20]. Shalt2 has about 11,000 concepts related mainly to a computer-manual domain, and integrates the LDOCE lexicons by incorporating ten of their major Boxcodes, not including MALE, FEMALE, and MOVABLE, into the class hierarchy. Although we have not yet fully tested the accuracy and efficiency of the NLCS and the paraphrase method, preliminary study showed a reasonable coverage for 2,000 sample sentences. The following is a sample run of the paraphrase method with the input sentence "The documents are collected in the libraries until it is convenient for you to print them."

```
*** Conceptual Representation ***
(*COLLECT
  (ROLE (*-PREDICATE-))
  (:DESCRIPTOR (*-ASPECT-MODAL- (:TENSE (*-PRESENT-)) (:PASSIVE (*-PLUS-))))
  (:MOOD (*-DECLARATIVE-))
  (:THEME (*PRINTED-DOCUMENT (:NUMBER (*-PLURAL-)) (:DEFINITENESS (*-DEFINITE-))))
  (:LOCATION (*PROGRAM-LIBRARY (:NUMBER (*-PLURAL-)) (:DEFINITENESS (*-DEFINITE-))))
  (:PROPOSITION-LINK
   (*BE-STATE
    (:CONNECTOR (*ENDING-LINK))
    (ROLE (*-PREDICATE-))
    (:DESCRIPTOR (*-ASPECT-MODAL- (:TENSE (*-PRESENT-))))
    (:PROPERTY (*CONVENIENT))
    (:EXPERIENCER (*-VAR-
                   (DOMAIN (*-OBJECT-))
                   (RANGE (*-OBJECT- (:HUMAN (*-PLUS-)) (:NUMBER (*-SINGULAR-))))
                   (REFERENT (*LISTENER))))
    (:AGENT
     (*PRINT
      (ROLE (*-PREDICATE-))
      (:DESCRIPTOR (*-ASPECT-MODAL- (:TENSE (*-PRESENT-))))
      (:THEME (*-VAR- (DOMAIN (*-OBJECT-))
                      (RANGE (*-OBJECT- (:HUMAN (*-MINUS-))
                                        (:NUMBER (*-PLURAL-))
                                        (:GENDER (*-NEUTER-))))))))))))

*** Paraphrase Method ***

(*COLLECT
 (:DESCRIPTOR (*-ASPECT-MODAL- (:TENSE (*-PRESENT-)) (:PASSIVE (*-PLUS-))))
 (:MOOD (*-DECLARATIVE-))
 (:THEME (*PRINTED-DOCUMENT))
 (:LOCATION (*PROGRAM-LIBRARY))
 (:PROPOSITION-LINK
  (*CHANGE-STATE
   (:CONNECTOR (*ENDING-LINK))
   (:DESCRIPTOR (*-ASPECT-MODAL- (:TENSE (*-PRESENT-)) (:PASSIVE (*-MINUS-))))
   (:GOAL (*CONVENIENT))
   (:EXPERIENCER
    (*-VAR- (RANGE (*-OBJECT-)) (DOMAIN (*-OBJECT-))
            (:HUMAN (*-PLUS-)) (:NUMBER (*-SINGULAR-)) (REFERENT (*LISTENER))))
   (:AGENT
    (*PRINT
     (:DESCRIPTOR (*-ASPECT-MODAL- (:TENSE (*-PRESENT-)) (:PASSIVE (*-MINUS-))))
     (:THEME
      (*-VAR- (RANGE (*-OBJECT-)) (DOMAIN (*-OBJECT-))
              (:HUMAN (*-MINUS-)) (:NUMBER (*-PLURAL-)) (:GENDER (*-NEUTER-)))))))))

*** Generator ***

それらを印刷することがあなたにとって便利になるまで 文書はライブラリに集められる
```

Figure 1: Sample Run of Paraphrasing Method

Acknowledgements

The author is indebted to thank Michael McDonald for his reviewing an early draft of this paper and giving many suggestions on technical writing.

References

1) R. G. G. Cattell. "Introduction" in special section for Next-Generation Database Systems. *Communications of the ACM,* 34(10):31-33, Oct. 1991.

2) EDR. "Proc. of International Workshop on Electronic Dictionaries". Technical Report TR-031, Japan Electronic Dictionary Research Institute, Ltd., November 1990.

3) K. Goodman and S. Nirenburg, editors. *"The KBMT Project: A Case Study in Knowledge-Based Machine Translation".* Morgan Kaufmann Publishers, San Mateo, California, 1991.

4) R. Kaplan and J. Bresnan. "Lexical-Functional Grammar: A Formal System for Generalized Grammatical Representation". In J. Bresnan, editor, *"Mental Representation of Grammatical Relations",* pages 173-281. MIT Press, Cambridge, Mass., 1982.

5) R. M. Kaplan and J. T. Maxwell III. "Constituent Coordination in Lexical-Functional Grammar". In *Proc. of the 12th International Conference on Computational Linguistics,* pages 303-305, Aug. 1988.

6) D. B. Lenat, R. V. Guha, K. Pittman, D. Pratt, and M. Shepherd. "CYC: Toward Programs with Common Sense". *Communications of the ACM,* 33(8):30-49, Aug. 1990.

7) M. Martinovic and T. Strzalkowski. "Comparing Two Grammar-Based Generation Algorithms: A Case Study". In *Proc. of the 30th Annual Meeting of ACL,* pages 81-88, June 1992.

8) M. Minsky. "A Framework for Representing Knowledge". In P. Winston, editor, *The Psychology of Computer Vision,* pages 211-277. McGraw-Hill, 1975.

9) K. Muraki. *"VENUS: Two-phase Machine Translation System",* pages 117-119. 1986.

10) S. Nirenburg, J. Carbonell, M. Tomita, and K. Goodman, editors. *"Machine Translation: A Knowledge-Based Approach".* Morgan Kaufmann Publishers, San Mateo, California, 1992.

11) E. Nyberg. "The FrameKit User's Guide Version 2.0". Technical Report CMU-CMT-88-MEMO, Center for Machine Translation, Carnegie Mellon University, March 1988.

12) E. H. Nyberg. *"A User's Guide to the FrameKit Knowledge Representation System",* July 1987. Preliminary Draft.

13) E. H. Nyberg, III and T. Mitamura. "The KANT System: Fast, Accurate, High-Quality Translation in Practical Domains". In *Proc. of the 14th International Conference on Computational Linguistics,* pages 1069-1073, July 1992.

14) Procter P. *Longman Dictionary of Contemporary English.* Longman Group Limited, Harlow and London, England, 1978.

15) J. Pustejovsky. "The Generative Lexicon". *Computational Linguistics,* 17(4):409-441, December 1991.

16) S. M. Shieber. *"An Introduction to Unification-Based Approaches to Grammar".* CSLI Lecture Notes, Number 4, Stanford, CA, 1986.

17) S. M. Shieber, F. C. N. Pereira, G. van Noord, and R. C. Moore. "Semantic-Head-Driven Generation". *Computational Linguistics,* 16(l):30-42, March 1990.

18) K. Takeda. "Designing Natural Language Objects". In *Proc. of 2nd International Symposium on Database Systems for Advanced Applications,* pages 444-448, Tokyo, Japan, April 1991.

19) K. Takeda. "Unification Grammars for Processing Coordinated Structures". Unpublished manuscript, 1992.

20) K. Takeda, N. Uramoto, T. Nasukawa, and T. Tsutsumi. "Shalt2 - A Symmetric Machine Translation System with Conceptual Transfer". In *Proc. of the 14th International Conference on Computational Linguistics,* pages 1034-1038, July 1992.

21) J. Wedekind. "Generation as Structure Driven Derivation". In *Proc. of the 12th International Conference on Computational Linguistics,* pages 732-737, August 1988.