Shalt2 - a Symmetric Machine Translation System with Conceptual Transfer

Koichi TAKEDA Naohiko URAMOTO
Tetsuya NASUKAWA Taijiro TSUTSUMI

Tokyo Research Laboratory, IBM Research
5-19 Sanban-cho, Chiyoda-ku, Tokyo 102, Japan
{takeda,uramoto,nasukawa,tsutsumi}@trl.vnet.ibm.com

ABSTRACT

Shalt2 is a knowledge-based machine translation system with a symmetric architecture. The grammar rules, mapping rules between syntactic and conceptual (semantic) representations, and transfer rules for conceptual paraphrasing are all bi-directional knowledge sources used by both a parser and a generator.

1. Introduction

Shalt2 is a research prototype of a knowledge-based, multi-domain/multi-lingual MT system. It has two predecessors, SHALT[13] (1982-90) and KBMT-89[3] (1987-89), which brought us valuable lessons we reflected in the design and implementation of **Shalt2**. As a result, **Shalt2** has been designed and implemented as a symmetric MT system with frame-based conceptual representation. The advantages of a symmetric architecture coupled with a conceptual representation are obvious. First, it allows us to maintain single knowledge sources for both parsing and generation. We can automatically control the coverage and reversibility of these processes. Second, conceptual structures are a desirable interface for representing the meaning of sentences, machine-generated output (such as diagnosis by an expert system), and expressions in graphical languages. AI-based approaches can provide powerful inference methods for identification of a (semi-)equivalent class of conceptual representations, which corresponds to a paraphrasing capability. Unlike interlingual MT systems, our approach relieves the parser of the burden of generating a unique meaning representation of all equivalent sentences, which becomes harder in proportion to the number of languages the system has to translate.

2. Shalt2 Architecture and Knowledge Sources

Shalt2 has five types of knowledge sources: a set G of *syntactic grammar rules* (including dictionaries), a set C of hierarchically defined conceptual primitives called *concept definitions*, a set M of *mapping rules* between syntactic and conceptual structures, a set P of *conceptual paraphrasing* rules, and a set E of *cases* (a structurally and conceptually disambiguated set of sample sentences). These knowledge sources are shared by three major processes: a *parser*, a *concept mapper*, and a *generator*. G should be provided for each language, whereas the set C should be defined for each application domain. M, P, and E should be provided for each pair of a language and a domain.† Figure 1 shows an overview of the **Shalt2** architecture.

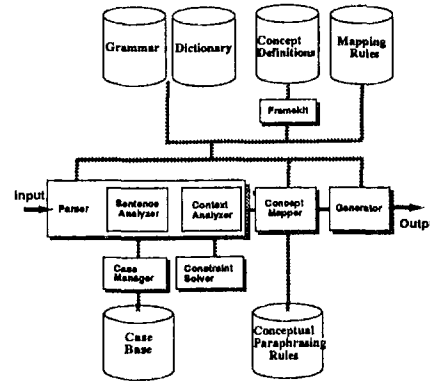


Figure 1: Shalt2 Architecture

2.1 Syntactic Grammar

Shalt2 currently has two English grammars (PEG and a less competent English grammar) and a Japanese grammar. The last two grammars are bi-directional grammars written in an LFG-like formalism called a *pseudo-unification grammar (PUG)*[10], and were originally employed in KBMT-89.†† PEG is not bi-directional, since it has too many destructive operations to build or encode record structures, but it is our primary English grammar for three reasons: (1) broad coverage, (2) ability to encapsulate structural ambiguities in a compact parse tree, and (3) compatibility with other NLP programs that use PEG to analyze English sentences. Our bi-directional English grammar is following up PEG and will be able to replace it. The symmetric architecture of **Shalt2**, however, allows uni-directional knowledge sources and processes to be hooked into the system. Their coverage and ability to parse or generate sentences can be measured in terms of a set of conceptual representations that they can relate to syntactic structures.

Although the syntactic structures of PEG and PUG grammars differ, they are integrated into a single syntactic representation called *Dependency Structures* (DSs)[6]. Roughly speaking, a DS is a tree-like structure with a set of nodes and a set of arcs, which correspond to maximal projections of syntactic constituents and gram-

†Our theory of multi-domain translation aims to compose a set of mapping rules efficiently when several domains are combined. It is expected that two sets of mapping rules for a single language will differ mainly in lexical mapping rules.

††The English version was originally written by Gates et al.[2] but was not bi-directional. The Japanese grammar was originally written by Mitamura and Takeda[2]. The Shalt2 versions of these PUG grammars have been modified considerably to allow them to handle coordinations, comparatives, and so on.

matical relationships, respectively. Some grammatical formalisms such as Constraint Dependency Grammar [4] postulate DSs as syntactic representations to which a constraint satisfaction algorithm [5] can be applied in order to resolve syntactic/semantic ambiguities efficiently. In the following, we show a PEG parse tree and a PUG f-structure, which will have the same DS, for the simple sentence "Insert a diskette into the drive."

```
PEG Parse Tree:
(IMPR (VERB* 'Insert' (insert PS))
      (NP (DET   (ADJ* 'a' (a BS)))
          (NOUN* 'diskette' (diskette SG)))
      (PP (PREP 'into')
          (DET (ADJ* 'the' (the BS)))
          (NOUN 'drive' (drive SG)))
      (PUNC '.'))

F-structure in the PUG English grammar:
((ROOT insert) (CAT v) (SUBCAT trans)
 (FORM inf) (MOOD imp) (TENSE pres)
 (OBJ ((ROOT diskette) (CAT n)
       (DET indef) (NUM sg)))
 (PPADJUNCT (((ROOT drive) (CAT n)
              (DET def) (NUM sg)))))

DS:
insert (CAT v, SUBCAT trans,
        MOOD imp, TENSE pres)
 DOBJECT
   diskette (CAT n, DET indef, NUM sg)
 PPADJUNCT
   drive (CAT n, PREP into, DET def, NUM sg)
```

The reader may notice that the above sentence should really have ambiguities in prepositional phrase attachment, which result in two conflicting dependencies "insert -PPADJUNCT- drive" and "diskette -PPADJUNCT- drive" in a single DS. We will discuss the handling of such ambiguities in Section 3.

2.2 Concept Definitions

A set of conceptual primitives is called a *Natural Language (NL) class system* [11], and is maintained as an object-oriented knowledge base under FrameKit [7]. The NL class system consists of a set of *constant classes*, three *meta-classes*, and *virtual classes*, with an *exclusive* inheritance mechanism discussed below.

Each class represents a concept in the real world. A class has zero or more *slots*, which describe the fillers it may use to represent a compound concept. *NL objects* are particular instances of NL classes. There are *is-a* and *part-of* relationships defined over the NL classes. For example,

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

defines a 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.

A class inherits each slot definition from its superclass(es), that is, the fillers of the is-a slot, unless the slot is redefined. A class can have more than one immediate superclass, but we restrict its inheritance to be *exclusive* rather than general multiple inheritance. That is, an instance of a class can inherit slot definitions from only one of its immediate superclasses. The idea behind exclusive inheritance is to realize certain 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." Such a pair usually shares slot definitions (:agent, :theme, and :goal) and selectional restrictions, except that "insert" inherits tense, aspect, and modality from its "verbal" superclass but not cardinality, ordinal, and definiteness (that is, the quality of being indefinite or definite) from its "nominal" superclass, although these features are inherited by "insertion." The following class definitions

```
(defclass *physical-action
   (is-a (value *predicate)))
(defclass *mental-object
   (is-a (value *object)))
(defclass *action
   (is-a (value *physical-action
                *mental-object)))
```

allow every instance of subclasses of *action to inherit slot definitions from either *physical-action or *mental-object. Exclusive inheritance also contributes to performance improvement of the parser since it allows us to reduce the number of possible superclasses from an exponential number to a linear number.

There are three meta-classes in NL classes – *var, *set, and *fun – to represent concepts that are not included in the normal class hierarchy. The first, *var, is a variable that ranges over a set of NL classes, which are *constants*. Pronouns and question words in natural languages usually carry this kind of incomplete concept. The second, *set, is a set constructor that can represent a coordinated structure in natural languages. The third, *fun, is a function from NL objects to NL objects. It captures the meaning of a so-called semi-function 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.

Since we allow only exclusive inheritance, the NL class system certainly lacks the ability to organize classes from various viewpoints, unlike ordinary multiple inheritance. *Virtual classes* are therefore introduced to compensate for this inability. For example,

```
(defvclass *option
           (def (*math-coprocessor
                 *hard-disk *software)))
(defvclass *male-thing
           (def (equal :sex *male)))
```

shows two types of virtual classe, *option and *male-thing. The *option consists of the classes *math-coprocessor, *hard-disk, and *software. The *male-thing is a class that includes instances of any class with the :sex slot filled with *male. Note that the maintainability of a class hierarchy drastically declines if we allow classes such as *option to be "actual" rather than virtual, as we will have many is-a links from anything that could be an option. The socond type of virtual class helps incorporate so-called *semantic features* into the NL class system. Existing machine-readable dictionaries (for example, LDOCE [8]) often have entries with semantic features such as HUMAN, LIQUID, and VEHICLE that may not fit into a given ontological class hierarchy. A virtual class definition

```
(defvclass *human
          (def (equal :human *true)))
```

with semantic restrictions (:agent (sem *human)) make it possible to integrate such entries into the NL class system.

The NL class system currently includes a few thousand concepts extracted from the personal-computer domain. The word senses in the SHALT dictionary (about 100,000 entries) and the LDOCE (about 55,000 entries) have been gradually incorporated into the NL class system.

2.3 Mapping Rules

Mapping rules define lexical and structural correspondences between syntactic and conceptual representations. A *lexical* mapping rule has the form

```
(emap *insert
   <=l=> insert ((CAT v) (SUBCAT trans))
   (role =sem (*physical-action))
   (:agent =syn (SUBJECT))
   (:theme =syn (DOBJECT))
   (:goal =syn (PPADJUNCT
                ((PREP into) (CAT n)))))
```

where a transitive verb "insert" maps to or from an instance of *insert with its exclusive superclass *physical-action. The three slots for *structural mapping* between concepts (:agent, :theme, and :goal) and grammatical roles (SUBJECT, DOBJECT, and PPADJUNCT) are also defined in this rule. The :agent filler, for example, should be an instance that is mapped from a syntactic SUBJECT of the verb "insert." The :goal filler must be a prepositional phrase consisting of a noun with the preposition "into." The fragments of syntactic feature structures following a lexical word or a grammatical function in a mapping rule specify the minimum structures that subsume feature structures of candidate syntactic constituents. These structural mappings are specific to this instance.

The *structural* mapping rule

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

specifies that the conceptual slots :mood and :time map to or from the grammatical roles MOOD and TENSE, respectively. Unlike the structural mapping 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. Given a dependency structure (DS), mapping rules work as distributed constraints on the nodes and arcs in the DS in such a way that a conceptual representation R is an image of the DS iff R is the minimal representation satisfying all the lexical and structural mappings associated with each node and arc. On the other hand, given a conceptual representation R, mapping rules work inversely as constraints on R to define a minimal DS that can be mapped to R. Thus, assuming that lexical mapping rules are similarly provided for nouns (diskette and drive) and feature values (imp, pres, and so on), we will have the conceptual representation†

```
(*insert-1
 (:mood  (*imp-1))
 (:time  (*pres-1))
 (:theme (*diskette-1 (:def (*indef-1))
                      (:num (*sg-1))))
 (:goal  (*drive-1 (:def (*def-1))
                   (:num (*sg-2)))))
```

for the sample sentence and its DS shown earlier in this section.

2.4 Conceptual Paraphrasing Rules

We assume that a source language sentence and its translation into a target language frequently have slightly different conceptual representations. An adjective in English might be a conjugable verb in translation. These differences result in added/missing information in the corresponding representations. The conceptual paraphrasing rules describe such equivalence and semi-equivalence among conceptual representations. These rules are 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 (*equal (:agent (*X (:num (*V))))
               (:theme (*Y/*person
                           (:def *indef)
                           (:num (*W)))))
       (*Z/*action (:agent (*X) (:num (*V))))
       (such-that (humanization *Z *Y)
                  (sibling *V *W)))
```

where *Y/*person specifies *Y to be an instance of any subclass of *person, *equal is roughly the verb "be," *humanization* is a relation that holds for pairs such as (*singer, *sing) and (*swimmer, *swim), and *sibling* holds for two instances of the same class. Intuitively, this rule specifies an equivalence relationship between sentences such as "Tom is a good singer" and "Tom sings well," as the following bindings hold:

```
(*equal (:mood (*dec)) (:time (*pres))
        (:agent (*tom (:num (*sg))))
        (:theme (*singer (:property (*good))
 (:def (*indef))
 (:num (*sg)))))

(*sing (:mood (*dec)) (:time (*pres))
       (:agent (*tom (:num (*sg))))
       (:property (*good)))

where *X = *tom, *Y = *singer, *Z = *sing,
      *V = *sg, and *W = *sg
```

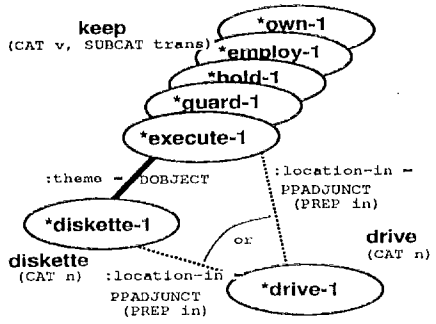
All the instances that have no binding in a rule must remain unchanged as the same slot fillers (e.g., *dec and *pres), while some fillers that have bindings in a rule may be missing from a counterpart instance (e.g., *indef and *W above). Note that *good has lexical mappings to the adjective "good" and the adverb "well."

2.5 Case Base

A case base is a set of DSs with no syntactic/semantic ambiguities. A conceptual representation for a DS can be computed by a top-down algorithm that recursively tries to combine an instance mapped from the root node of a DS with an instance of each of its subtrees. The arc from the node to a subtree determines the conceptual slot name.

We have already built a case base that includes about 30,000 sentences from the IBM Dictionary of

†Conceptual representation of a sentence consists of instances of classes. We use a hyphen and a number following a class name (*insert-1, *imp-1, ...) when it is necessary to show instances explicitly. Otherwise, we identify class names and instance names.



Only relevant information is shown. Mapping constraints (e.g. :theme = DOBJECT) are actually associated for each pair of instances.

Figure 2: Sample DS with Mapping Constraints

Computing[1]. Selected sentences in the LDOCE have also been added to the case base. The sentences in the LDOCE define word senses or show sample usages of each entry. Though composed of a limited vocabulary, they are often syntactically/semantically ambiguous and it is time-consuming for users to build the case base completely manually. Therefore, the **Shalt2** parser is used to develop the case base. Starting with a small, manually crafted "core" case base, each new sentence is analyzed and disambiguated by the parser to generate a DS, which is corrected or modified by the user and then added to the case base. As the size of the case base grows, the proportion of human corrections/modifications decreases, since the output of the parser becomes more and more accurate. The process is called *knowledge bootstrapping* and is discussed by Nagao[6] in more detail. Mapping constraints, however, are associated with only a part of the case base, because the NL class system and the mapping rules are not yet complete.

3. Parser

The **Shalt2** parser first generates a DS with packed structural ambiguities for a given input sentence. It actually calls a PEG parser or Tomita's LR-parser[12] for PUG, and then calls a DS converter to map a PEG parse tree or a PUG feature structure† to a DS. Next, mapping rules are applied to the DS so that lexical and structural mappings are associated with each node and arc in the DS. Figure 2 shows a DS with mapping constraints for the sentence "Keep the diskette in the drive," where the verb "keep" has five word senses: *execute, *guard, *hold, *employ, and *own. It is clear that we will end up with ten distinct conceptual representations if we evaluate all the mapping rules, and in general, combinatorial explosion could easily make the parser impractical. Viewing the mapping rules as *constraints* rather than procedural rules is the key to our parser, and is called *delayed composition* of conceptual representation.

A sentence analyzer called SENA[14] disambiguates the DS by using a constraint solver JAUNT[5] and a case base. JAUNT applies grammatical constraints (for instance, modifier-modifiee links between nodes do not cross one another) and semantic constraints (such as selectional restrictions,†† *functional control*, and other NL object identity constraints detected by the context analyzer) uniformly to a DS that has ambiguities, and calculates pairwise consistency efficiently for each combination of nodes. Finally, the case base provides preferential knowledge to favor one pair of nodes over all other consistent pairs. The disambiguation process can be summarized as follows:

1. For each conflicting arc in the DS, calculate the "distance"[6] between the two nodes in the arc by using a case base.
2. Leave the arc with the minimal distance and eliminate all the other conflicting arcs. Each NL object associated with a matching node in the case base also gives a higher preference to the same class of instance over the other instances in a node.
3. Propagate the deletion of arcs to other nodes and arcs in the DS. Eliminate nodes and arcs that are no longer valid in the DS.
4. Apply the above steps until there are no conflicting arcs in the DS.

The resulting DS has no structural ambiguity. Remaining lexical ambiguities are similarly resolved, because we can also determine which pair of NL objects connected with an arc has the minimal distance in the case base. Our case base for computer manuals would support the "diskette -PPADJUNCT- drive" arc and the *hold-1 instance with diskette as its DOBJECT. Therefore, the parser will eventually return

```
(*hold-1
  (:theme (*diskette
            (:location-in (*drive)))))
```

as a disambiguated result. Nagao[6] discusses more sophisticated techniques such as scheduling sets of arcs to be disambiguated, backtracking, and relaxation of case base matching by means of an is-a hierarchy.

Finally, a context analyzer is called to resolve anaphora, identity of definite nouns, and implicit identity between NL objects. It stores the DS in the working space, where references to preceding instances are represented by the links between instances in the DSs. These inter-sentential links are used to determine the scopes of adverbs such as "also" and "only".

For example, if the phrase "role of an operator" appears in a text, the word "operator" could be a person who operates a machine or a logical operator for computation, but no sufficient information is available to resolve this ambiguity at this moment. In such cases, creating referential links in a forest of DSs could lead us to find evidence for disambiguating these two meanings. The scope of an adverb, such as "also," is determined by identifying repeated NL objects and newly introduced NL objects, where the latter are more likely to fall within the scope of the adverb.

The context analyzer uses a similar method to determine lexical ambiguities that were not resolved by the sentence analyzer when the case base failed to provide enough information.

†Tomita's local ambiguity packing[12] is used to obtain a feature structure with packed structural ambiguities.

††We use about 20 of the semantic features described in the LDOCE. The restrictions imposed by the features are rather "loose," and are used to eliminate only unlikely combinations of word senses.

4. Concept Mapper

Given a conceptual representation, which is an output from the parser, and a target language, the *concept mapper* tries to discover another conceptual representation that has a well-defined mapping to a DS while keeping the semantic content as intact as possible. This process is called *conceptual transfer*. If the given conceptual representation already has a well-defined mapping to a DS, the concept mapper does nothing and **Shalt2** works like an interlingual MT system. It is important that conceptual transfer should be related with the mapping to a DS, because there are generally many conceptual representations with a similar semantic content. The existence of well-defined mapping not only guarantees that the generator can produce a sentence in the target language, but also effectively eliminates unsuccessful paraphrasing.

In addition to the paraphrasing rules mentioned earlier, the concept mapper uses the following general rules for conceptual transfer.† The paraphrasing rules are composed to make a complex mapping.

- Projection: Map an NL object 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 NL object of a class X to an instance of one of the superclasses of X.
- Specialization: Map an NL object of a class X to an instance of one of the subclasses of X.

As an example, a projection rule is frequently used when we translate English nouns into Japanese ones, as in the following examp:

```
diskette          (*diskette (:num (*sg)))
diskettes         (*diskette (:num (*pl)))
a diskette        (*diskette (:num (*sg))
                             (:def (*indef)))
the diskettes (*diskette (:num (*pl))
                             (:def (*def)))
ディスケット   (*diskette)
```

Here, the four English noun phrases above are usually translated by the same Japanese noun phase†† (the fifth one), which does not carry any information on :num and :def. We provide a paraphrasing rule for translation in the opposite direction such that for any instance of the *object can obtain appropriate :num and :def fillers. The parser, however, is responsible for determining these fillers in most cases. In general, the designer of semi-equivalent rules for translation in one direction has to provide 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 rules 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.

†These are semi-equivalent rules. Equivalent rules have higher priority when the rules are to be applied.

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

5. Grammar-Based Sentence Generator

Recent investigation of unification grammars and their bi-directionality [15, 9, 10] has enabled us to design and implement a grammar-based generator. Our generator uses a PUG grammar, which is also used by the parser, to traverse or decompose a feature structure obtained from a DS in order to find a sequence of grammar rule applications, which eventually lead to lexical rules for generating a sentence. The generation algorithm is based primarily on Wedekind's algorithm, but is modified for PUG.

The current implementation of our generator lacks subtle control of word ordering, honorific expressions, and style preference. We are developing a set of *discourse parameters* to associate preferences with grammar rules to be tried, so that specific expressions are favored by the parameter settings.

Acknowledgment

Yutaka Tsutsumi built an initial version of conceptual definitions. Katashi Nagao originally designed and implemented the disambiguation algorighm of the sentence analyzer. His continuous efforts to build a large-scale case base and to improve the disambiguation algorithm have been continued by the **Shalt2** group. Hiroshi Maruyama enhanced his constraint solver for our project, which has led us to a constraint propagation method with delayed composition of conceptual representations. Michael McDonald read through an early draft of this paper and gave us many suggestions on technical writing. We thank them all.

References

1) IBM Corp. *"Dictionary of Computing" (8th Edition).* SC20-1699-07, 1987.

2) D. Gates, K. Takeda, T. Mitamura, L. Levin, and M. Kee. "Analysis and Generation Grammar". *Machine Translation*, 4(1):53–66, March 1989.

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) H. Maruyama. "Structural Disambiguation with Constraint Propagation". In *Proc. of the 28th Annual Meeting of ACL*, volume 3, pages 31–38, June 1990.

5) H. Maruyama. "JAUNT: A Constraint Solver for Disjunctive Feature Structures". Technical Report RT0059, Tokyo Research Lab., IBM Research, 1991.

6) K. Nagao. "Dependency Analyzer: A Knowledge-Based Approach to Structural Disambiguation". In *Proc. of the 13th International Conference on Computational Linguistics*, pages 484–489, Aug. 1990.

7) 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.

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

9) S. M. Shieber, F. C. N. Pereira, G. van Noord, and R. C. Moore. "Semantic-Head-Driven Generation". *Computational Linguistics*, 16(1):30–42, March 1990.

10) K. Takeda. "Bi-Directional Grammars for Machine Translation". In *Proc. of Seoul International Conference on Natural Language Processing*, pages 162–167, Seoul, Korea, November 1990.

11) 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.

12) M. Tomita. *"Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems"*. Kluwer Academic Publishers, Boston, MA, 1985.

13) T. Tsutsumi. "A Prototype English-Japanese Machine Translation System for Translating IBM Computer Manuals". In *Proc. of the 11th International Conference on Computational Liguistics*, August 1986.

14) N. Uramoto. "Lexical and Structural Disambiguation Using an Example-Base". In *Proc. of the 2nd Japan-Australia Joint Symposium on NLP*, pages 150–160, Oct. 1991.

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