KURA:
A Transfer-Based Lexico-Structural Paraphrasing Engine

TAKAHASHI Tetsuro, IWAKURA Tomoya, IIDA Ryu,
FUJITA Atsushi and **INUI Kentaro**
Department of Artificial Intelligence, Kyushu Institute of Technology, JAPAN
{t_taka,t_iwa,r_iida,a_fujita,inui}@pluto.ai.kyutech.ac.jp

Abstract

This paper describes a Japanese lexico-structural paraphrasing engine called KURA, discussing its underlying architectural and implementational issues. KURA has several advantages: (a) it is designed to enable lexico-structural paraphrasing as an application-independent task, (b) it is based on the division of labor between transfer and revision processes, (c) it is designed to represent all paraphrasing knowledge uniformly as a set of declarative lexico-structural transformation rules, (d) it is efficient enough to be used for large-scale experiments, and finally (e) it provides a computational environment that supports process monitoring, error analysis, and rule debugging.

1 Introduction

Automated paraphrasing is a sub-field of NLP that has been receiving increasing attention because of its potential for a wide range of NLP applications: machine translation (Shirai et al., 1998; Yoshimi and Sata, 1999), text generation (Inui et al., 1992; Robin and McKeown, 1996), authoring/revision support (Takahashi and Ushijima, 1991; Takeishi and Hayashi, 1992; Hayashi, 1992), reading assistance (Carroll *et al.*, 1998; Inui, 2001), summarization (Mani *et al.*, 1999; Nanba and Okumura, 2000), etc.

Paraphrasing can be primarily viewed as a special case of translation in the sense that both transform the wording of a source text into another different wording, while preserving its meaning as much as possible. Due to this similarity, one may think that paraphrasing can be done simply by using state-of-the-art transfer-based technologies of cross-lingual machine translation (MT) such as those described in (Wahlster *et al.*, 2000; Richardson *et al.*, 2001). However, the matter is not so simple for three reasons:

- While extensive work has been conducted on cross-lingual translation, little has been done on paraphrasing from the point of view of computational linguistics — the nature of paraphrasing is not yet clear. Paraphrasing may have its own idiosyncracies that need to be taken into account in designing the architecture of a paraphrasing system.
- While there are many parallel and comparable corpora available for cross-lingual translation, it is difficult to collect a large amount of examples for paraphrasing. This makes it difficult to use present corpus-based MT techniques for paraphrasing.
- One promising way to gain insight into the nature of paraphrasing is to conduct empirical experiments using a large-scale corpus. However, there are few, if any, computational tools available that support such experiments.

Motivated by these problems, we have developed a computational environment in which one can effectively conduct empirical experiments with a variety of lexico-structural paraphrasing tasks (paraphrasing, hereafter)[1] in Japanese. This paper gives an overview of this paraphrasing engine called KURA. In Section 2, we first review the already-known idiosyncratic aspects of paraphrasing and then propose a three-layered, revision-based model of paraphrasing. KURA is an instance of the implementation of this model. We describe the internal structure of KURA focusing on the knowledge representation it uses in Section 3 and the currently implemented mechanism for process control in Section 4. We provide information about the installation of the system and the devices for process monitoring and output data management in Section 5, and conclude the paper in Section 6.

[1] By lexico-structural paraphrasing we mean meaning-preserving linguistic transformation, such as lexical/phrasal replacement, verb alternation, topicalization, and sentence aggregation/division, that can be done without using the associated communicative context.

2 Overall architecture of KURA

2.1 Architectural issues in paraphrasing

In contrast to cross-lingual translation, paraphrasing has several idiosyncratic aspects that affected our architectural design:

- The goal-oriented nature of paraphrasing is more salient than that of cross-lingual translation; namely, the output must be adequate according to not only (a) the morpho-syntactic well-formedness and (b) meaning preservation but also (c) the application-specific purposiveness. In reading assistance, for example, it is critical that the paraphrasing of an input sentence or text[2] improve its comprehensibility (Inui and Yamamoto, 2001). The problem here is how to control paraphrasing to achieve a given purpose for a given input.
- In paraphrasing, the morpho-syntactic information of a source sentence should be accessible throughout the transfer process since morpho-syntactic transformation in itself can often be a motivation for or a goal of paraphrasing. Therefore, approaches such as semantic transfer, where morpho-syntactic information is highly abstracted as in (Dorna *et al.*, 1998; Richardson *et al.*, 2001), do not suit this task. Provided that the morpho-syntactic stratum is an optimal level of abstraction for transfer in paraphrasing, one should recall that semantic-transfer approaches such as those cited above were motivated mainly by the need to reduce the complexity of transfer knowledge, which could be unmanageable in morpho-syntactic transfer. We need, therefore, to find a way to reduce the complexity of transfer knowledge while staying at the morpho-syntactic level.
- While the wording of a source sentence changes completely in cross-lingual translation, it is largely preserved in most cases of paraphrasing. In practice, therefore, a paraphrasing system often needs to access either the morpho-syntactic or the semantico-contextual information, and only of a limited part of a source sentence. The problem here is that it is almost always unpredictable which type of information of which part is needed in a particular case.

2.2 A three-layered, revision-based paraphrasing model

The above considerations have led us to design a computational model of paraphrasing that has the following features (see Figure 1):

[2]This paper uses *sentence* and *text* interchangeably. In fact, KURA is designed to receive and produce a sequence of sentences that constitute a text.

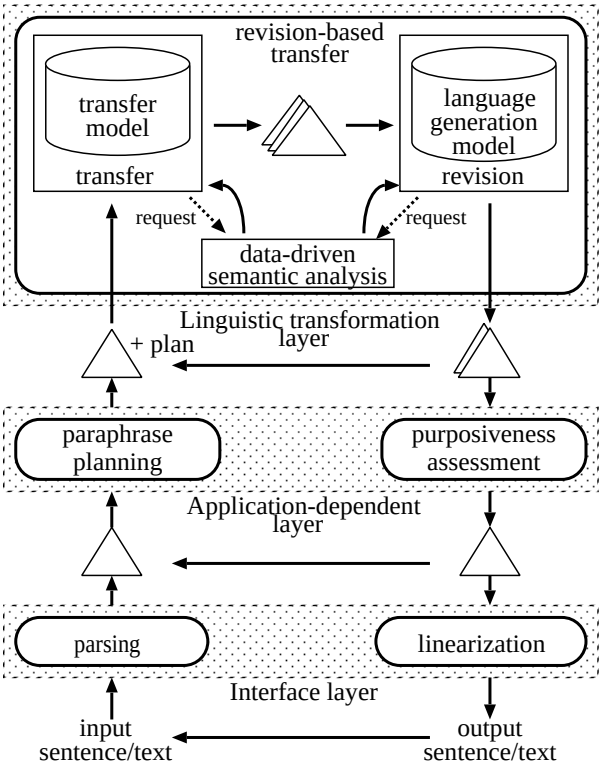

Figure 1: Overall architecture

- *Three-layered division of labor*: It is designed to maximally encapsulate the task of application-neutral linguistic transformation from application-specific tasks including problem identification, paraphrase planning, and purposiveness assessment.
- *Revision-based morpho-syntactic transfer*: It has a revision component to revise or reject ill-formed/inadequate paraphrase candidates produced by the non-deterministic morpho-syntactic transfer component.
- *Data-driven control of semantic analysis*: It has a semantic analysis component that can be activated on demand to carry out various procedural computational tasks for semantic or contextual analysis.

We will elaborate on the first two features below.

2.2.1 Three-layered division of labor

The overall architecture is functionally three-layered:

- *The interface layer* provides an interface between input/output strings and their internal representations. In our architecture, all intermediate structures used by the components are represented uniformly as morpho-syntactic dependency structures which can be annotated with semantic and textual information accordingly.
- *The application-dependent layer* covers subtasks related to application-dependent purposiveness.

At the source side, it produces a paraphrasing plan that specifies what class of linguistic transformation should be applied to which part of the input. At the target side, on the other hand, it assesses the purposiveness of each paraphrase of a given set of paraphrase candidates and selects an optimal one.

- *The linguistic transformation layer* performs linguistic transformation, the core application-independent subtask of paraphrasing, which generates morpho-syntactically well-formed and meaning-preserving paraphrases from a given input coupled with a paraphrase plan. The current implementation of KURA focuses particularly on this layer.

If this division of labor works properly, one should be able to build a paraphrasing engine that is reasonably application-independent and can thus be used for different applications. Although it is unclear if one can fully eliminate application-dependent processes in linguistic transformation, we believe that it is important, at least in the preliminary stage of research, to make a maximum effort to avoid the confusion between linguistic and purposive adequacy of a paraphrase.

2.2.2 Revision-based morpho-syntactic transfer

Our model carries out linguistic transformation at the morpho-syntactic stratum. As noted in Section 2.1, one problem we must address in morpho-syntactic transfer is how to reduce the complexity of transfer knowledge. Let us consider an example:

(1) s. リンゴしか食べない.
ringo (apple) *shika* (except) *tabe* (to eat) *nai* (not)
(I eat nothing but apples.)
t. 食べるのはリンゴだけだ.
taberu (to eat) *no-wa* (thing-TOP) *ringo* (apple) *dake* (only) *da* (COPULA)
(All I eat is apples.)

Generalizing from this example, one can come up with syntactic transfer pattern:

(2) N しか V ない → V のは N だけだ
N *shika* (except) V *nai* (not) → V *no-wa* (thing-TOP) N *dake* (only) *da* (COPULA)

However, this rule does not specify what to do to paraphrase such sentences as (1). For example, it does not take into account verb conjugation. The form of an input verb must changes depending on the context: in (1), "*tabe* (to eat)" must be transformed into "*taberu* (to eat)".

The next example illustrates how easily the situation can become even more complicated:

(3) s. 彼はおいしいリンゴしか食べたくなかった.
kare (he) *wa* (TOP) *oishii* (delicious) *ringo* (apple) *shika* (except) *tabe* (to eat) *taku* (to want) *nakat* (not) *ta* (PAST)
(He wanted to eat nothing but delicious apples.)
t. ∗ 彼 は 食べ たく のはおいしいリンゴだけ だ た.
kare (he) *wa* (TOP) *tabe* (to eat) *taku* (to want) *no-wa* (thing-TOP) *oishii* (delicious) *ringo* (apple) *dake* (only) *da* (COPULA) *ta* (PAST)
r. 彼 が 食べた かった のはおいしいリンゴだけ だっ た.
kare (he) *ga* (NOM) *tabe* (to eat) *takat* (to want) *ta* (past) *no-wa* (thing-TOP) *oishii* (delicious) *ringo* (apple) *dake* (only) *dat* (COPULA) *ta* (PAST)
(All he wanted to eat was a delicious apple.)

A naive application of rule (2) to sentence (3s) produces (3t), which has a topicalization error ("は `wa`"), a tense-related error ("たく `taku`") and two verb-conjugation errors — the desirable output would be (3r). However as one can imagine, incorporating all such factors into a transfer rule would invariably make it very complicated.

Our solution to this problem is to (a) leave the description of each transfer pattern *underspecified* as in (2) and (b) implement the knowledge about linguistic constraints that are independent of a particular transfer pattern separately from the transfer knowledge. There should be a wide range of such transfer-independent linguistic constraints. Constraints on morpheme connectivity, verb conjugation, word collocation, and tense and aspect forms in relative clauses are typical examples of such constraints. We call the description of such linguistic knowledge a *language model* (or a language generation model). If a transfer rule is left underspecified, it is likely to produce morpho-syntactically ill-formed or semantically inadequate paraphrases. To cope with this problem, we introduced a revision component that uses the language model to revise and reject faulty results of transfer.

In contrast to existing transfer-based cross-lingual MT frameworks, our revision-based approach can be seen as an effort to maximally enhance the role of generation and post-editing in order to maximally reduce the load of transfer. The advantages are as follows:

- The incorporation of the revision component will reduce the redundancy and thus the complexity of the transfer knowledge while leaving the level of transfer at the morpho-syntactic stratum. This conclusion is based on our empirical observation that each single piece of the knowl-

edge of a language model (e.g., collocation constraints for a pair of words) is likely to be associated with more than one transfer pattern.

- Our approach can be better integrated with emerging corpus-based approaches to the acquisition of transfer patterns (Barzilay and McKeown, 2001; Sekine, 2001) since the ability to revise transfer results will make the system tolerant of the deficiencies of acquired knowledge.

3 Knowledge representation

3.1 Data structure

In KURA, the internal representations used by all the components have a uniform data structure — a morpheme-based dependency structure (MDS). An MDS is a dependency tree each of whose nodes corresponds to a morpheme (lexical entry) with various morpho-syntactic and semantic annotations. As suggested by previous works on transfer-based MT (Meyers *et al.*, 1996; Lavoie *et al.*, 2000), the dependency-based representation has the advantage of facilitating syntactic transformation operations.

3.2 Uniform representation of transfer and revision knowledge

The linguistic transformation component uses both transfer knowledge (a transfer model) and revision knowledge (a language model). In KURA, both are represented uniformly as a set of MDS rewriting rules.

3.2.1 Transfer rules

Previous works on transfer-based MT systems (Lavoie *et al.*, 2000; Dorna *et al.*, 1998) and alignment-based acquisition of transfer knowledge (Meyers *et al.*, 1996; Richardson *et al.*, 2001) have proven that transfer knowledge can best be represented by declarative structure mapping (rewriting) rules each of which typically consists of a pair of source and target partial structures. We adopt a similar type of representation: we represent transfer knowledge as a set of MDS rewriting rules (SR rules) in the following form:

(4) `sr_rule(`*id*`,`*transfer_class*`,`*name*`,`*s_mds*`,`*t_mds*`).`

which denotes that a partial MDS that matches *s_mds* can be replaced with the instantiated target MDS specified by *t_mds*. An example of an SR rule is presented in Figure 2, which also shows a snapshot of the application of the rule to sentence (3s).

3.2.2 Variably-lengthened morpheme sequences (VMSs)

Recall here that we are trying to achieve linguistic transformation at the morpho-syntactic stratum. This requires devising additional means to enable flexible pattern matching of morpho-syntactic structures. For example, one may want to be able to apply rule `(108)` in Figure 2 equally to each of the following input sentences:

(5) a. リンゴしか食べない.
ringo (apple) *shika* (except) *tabe* (to eat) *nai* (not)
(I eat nothing but apples.)

b. リンゴしか食べ させ ない.
ringo (apple) *shika* (except) *tabe* (to eat) *sase* (to allow) *nai* (not)
(I do/will not allow *someone* to eat anything but apples.)

c. リンゴしか食べ させてい ない.
ringo (apple) *shika* (except) *tabe* (to eat) *sase-te* (to allow) *i* (PRES-PERFECT) *nai* (not)
(I have not allowed *someone* to eat anything but apples.)

This requirement has led us to enhance the expressibility of SR rules by introducing the notion of a *variably-lengthened morpheme sequence* (VMS). An example of a VMS can be found at node `X2` in Figure 2, where POS label `vms_aux_verb` denotes a subclass of VMSs. Each subclass of VMSs can be defined by a rule editor in terms of a regular expression of morpheme sequence patterns as follows:

(6) a.
```
def_vms(vms_aux_verb,
        [(aux_verb,"*")]).
```

b.
```
def_vms(vms_postp,
        [(postp,"*"),(comma,"*")]).
```

3.2.3 Revision rules

Compared with the representation of transfer knowledge, the representation of revision knowledge has a wider range of options. For example, one may develop a blackboard model where revision knowledge will be decomposed and distributionally assigned to procedural revision experts. Alternatively, one may devise a general procedure to consolidate an ill-formed transfer output with a lexico-grammar that specifies linguistic constraints, perhaps in way analogous to robust parsing of ill-formed sentences as in (Mellish, 1989). While understanding the advantages of the approaches (particularly of the second one), we adopted yet another approach: to represent all revision knowledge as a set of SR rules expressed by the same formalism as transfer rules.

To use the SR rule form for revision knowledge, we slightly modified it:

(7) a. `sr_rule(`*id*`,`*revision_class*`,`*name*`,`
s_mds`,`*t_mds*`).`

b. `sr_rule(`*id*`,`*revision_class*`,`*name*`,`
s_mds`,must([`e_mds_1`,...,`e_mds_n`])).`

The rule of form (7a) is used to revise an ill-formed or inadequate transfer output; it denotes that a partial MDS that matches *s_mds* must always be

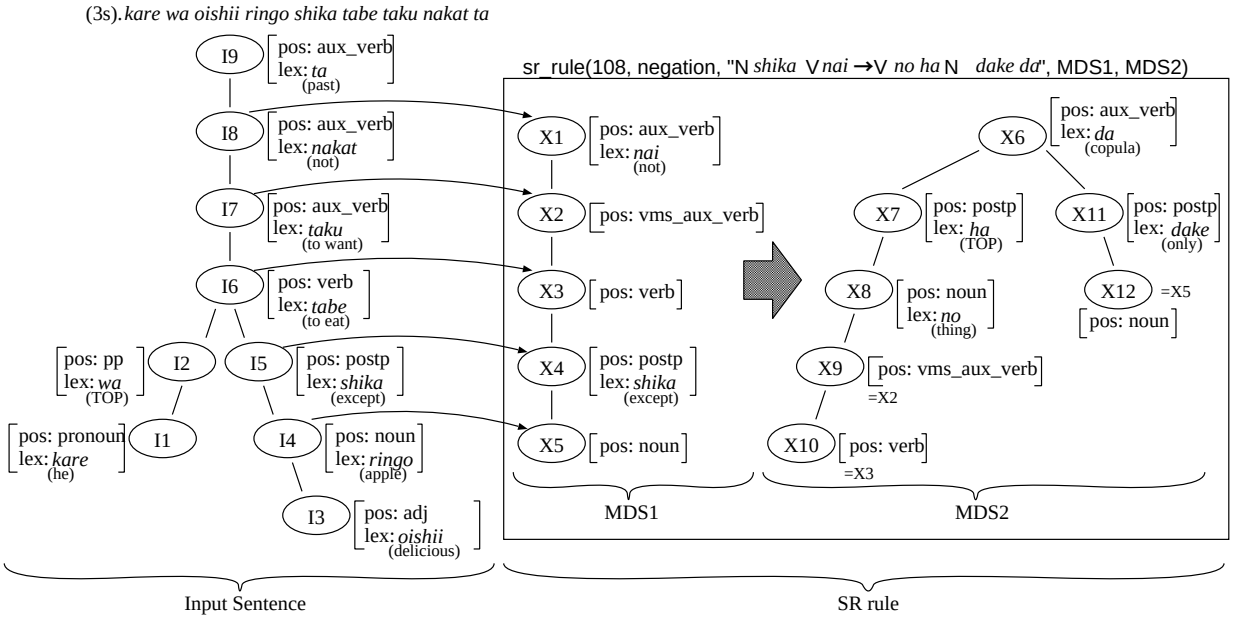

Figure 2: An SR rule

replaced with the instantiated target MDS, t_mds. The rule of form (7b) is, on the other hand, used to reject a fatally ill-formed or inadequate transfer output. It denotes the constraint that if s_mds holds for a given MDS then either e_mds_1, ..., or e_mds_n must also hold; in other words, if

$$s\_mds \wedge (\overline{e\_mds_1 \vee \ldots \vee e\_mds_n})$$

holds for a given MDS then it must be rejected. You will find an example of a revision rule later in Section 3.3.1.

Note that, while the form of revision rules is the same as that of transfer rules, the semantics is slightly different. Each transfer rule corresponds to a different version of paraphrasing; that is, different transfer rules produce different paraphrases. In revision, on the other hand, every revision rule is obligatorily applied whenever applicable, in manner analogous to rule application in forward-chaining production systems. We will describe an effective controlling method for rule application later in Section 4. Revision rules may conflict with one another; that is, the result of a sequence of rule applications may differ depending on the order of application. We will address the problem of conflict resolution as a subject for our future work in Section 6.

This rule-based declarative knowledge representation has an advantage in that it allows us to edit the revision knowledge simply by adding, deleting, or revising individual rules. Given the fact that we know very little about the kinds of knowledge needed in the revision process, we initially need to try to accumulate a sufficient number of prototypical instances of revision knowledge in a error-driven manner. To carry out such a trial-and-error-based exploration, we need to develop mechanisms for knowledge editing, even though this may force us to sacrifice, to a certain extent the expressibility of knowledge representation.

3.3 Layered rule representation

The form of SR rules is so designed as to express different types of transfer and revision knowledge. Its expressibility is, however, not powerful enough to serve as an implementation language for procedural semantic analysis and application-dependent processing. SR rules are not simple enough to be easily handcrafted, either. Motivated by these two deficiencies of the SR rule form, we developed three-layered knowledge representation as shown in Figure 3. To provide an interface between the layers, KURA has (a) a rule translator that automatically translates each simplified SR rule (SSR rules) into the corresponding SR rule and (b) a rule compiler that automatically compiles each SR rule into the corresponding sequence of MDS processing operators (SP operators). By introducing the SSR rule and SP operator layers, one can handcraft transfer and revision rules at a sufficiently abstract level, while being able to implement arbitrary procedural operations for MDSs.

In the rest of this subsection, we will elaborate on each of the newly introduced layers: SSR and SP.

3.3.1 Simplified SR (SSR) rules

The form of SSR rules is designed to facilitate the task of handcoding SR rules. For example, to define SR rule (108) in Figure 2, a rule editor needs only to specify its simplified form as in (8):

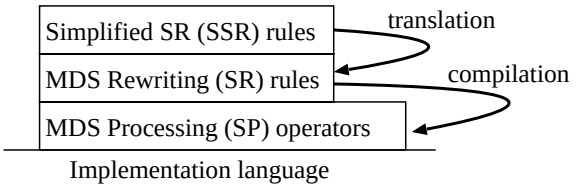

Figure 3: Three-layered knowledge representation

(8) N しか V し-ない -> V のは N だけだ.
N *shika* V *shinai* -> V *no ha* N *dake da*.

The advantages of introducing the SSR rule layer should be obvious. The SSR rule form allows a rule writer to edit rules with an ordinary text editor, which makes the task of rule editing much more efficient than providing her/him with a GUI-based complex tool for editing SR rules directly. It also has the advantage of "readability", which is particularly important in group work.

The idea is that most specifications of an SR rule can usually be abbreviated if the means to automatically complement it are provided. We use a parser and macros to do this: the rule translator complements an SSR rule by macro expansion and parsing to produce the corresponding SR rule specifications. We will demonstrate this through examples.

N and V in (8) are examples of macros. Each macro is defined as in (9), where symbol M denotes a general morpheme that can be annotated with arbitrary features:

(9) a. N -> M(noun).
b. V -> M(verb).

Symbol "-" as found in (8) is also a macro that is expanded into a VMS (see 3.2.2 above) by a set of context-sensitive expansion rules as in (10).

(10) a. noun-* -> vms_postp.
b. verb-* -> vms_aux_verb.
c. *-eos -> vms_aux_verb.

Given an SSR rule, the translator first expands the macros used in it by applying macro definitions. For example, rule (8) is transformed into intermediate representation (11):

(11) M1(noun) しか M2(verb) M3(vms_aux_verb) ない -> M2(verb) M3(vms_aux_verb) のは M1(noun) だけだ.

The translator then parses the source and target parts of the result separately, and finally produces the full specifications of the corresponding SR rule (see Figure 2 for the current example). To parse SSR rules, we use the same parser as that used to parse input sentences in the interface layer (see Figure 1). This also improves the efficiency of rule development because it significantly reduces the burden of maintaining the consistency between the POS-tag set used for parsing input and that used for rule specifications. This is also an important advantage of introducing the SSR rule layer.

In rule translation, the parser can produce a wrong parse without any additional control. To cope with parsing errors, the SSR rule form allows a rule editor to annotate SSR rules to fully control parsing results. An example can be seen in rule (12):

(12) N1 M(cnct_prt; lex:と) [N2 の N3] -> ...

where M(cnct_prt; lex:と) forces "と" to be analyzed as cnct_prt (connective particle), and the square brackets impose constraints on the parse tree so that "N1 と" depends on "N3" (the head of "N2 の N3").

The current implementation of the SSR rule form also allows other types of annotation including the following:

- Semantic condition
  (13) N(sem_class:person) -> ...
- Negation of feature values
  (14) N(sem_class!:person) -> ...
- Negation of the existence of morphemes
  (15) N のみ<ならず> -> N だけ.
- Disjunction
  (16) N1-が N2-を思案する
  -> N1-が N2-{ を, について } 考える.
- Rejection (The rules below are equivalent.)
  (17) a. まったく V する
  -> must(まったく V し-ない).
  b. まったく V し-<ない> -> reject.

3.3.2 MDS processing (SP) operators

In the current implementation of KURA, the internal data structure of MDSs is totally encapsulated, allowing access only through a small fixed set of SP operators. Several frequently used SP operators are listed in Figure 4.

The roles of SP operators can be summarized as follows:

- SP operators provide an interface between the SR rule layer and the implementation layer; each SR rule is automatically compiled into a sequence of SP operators, which is then interpreted by the system. Figure 5, for example, shows the process and the result of compiling the SR rule given in Figure 2. As illustrated in Figure 5, the result of rule compilation is still reasonably "readable", which significantly facilitates process monitoring and knowledge debugging.
- The encapsulation of the low-level internal data structure of MDSs allows a user to stay at an abstract level even in implementing the semantic analysis component.
- The encapsulation also allows the low-level implementation of SP operators to be performed

`match(+MOR1, +MOR2)`: is true iff node `MOR1` is unifiable with another node `MOR2`.

`exist(+MOR, +MDS)`: is true iff a node that is unifiable with `MOR` exists in `MDS`.

`depend(?MOTHER, +VMS, ?DAUGHTER)`: is true iff two nodes corresponding to `MOTHER` and `DAUGHTER` respectively are in the mother-daughter relationship via a VMS corresponding to `VMS`.

`feature(+MOR, +PATH, ?VALUE)`: is true iff the value of a feature specified by `PATH` of node `MOR` is `VALUE`.

`disown(+MOR)`: removes the link between node `MOR` and its mother.

`replace(+OLD_MOR, +NEW_MOR)`: replaces node `OLD_MOR` with node `NEW_MOR`.

`adopt(+MOR, +MOTHER)`: adds node `MOR` as a daughter of mother node `MOTHER`.

`change_feature(+MDS, +PATH, +VALUE)`: destructively substitutes a value `VALUE` into a feature slot specified by `MDS` and `PATH`.

Figure 4: SP operators

independently of the knowledge and algorithms described at higher levels of abstraction.

4 Process control

This section addresses the issues of process control. Because the current version of KURA is not equipped with any devices for application-dependent processing, we are not yet ready to discuss which transfer rule should be applied when and where. Instead, in this section, we first describe three modes of rule application control designed to conduct various experiments (Section 4.1), and then present two simple techniques we have implemented to improve the process efficiency. These techniques are (a) data-driven rule retrieval (Section 4.2) and (b) optimization in rule compilation (Section 4.3).

4.1 Rule application mode

The current version of KURA provides three modes of rule application control:

- *Single*: For each input sentence, the system tries to apply each rule of a given set of transfer rules.
- *Best*: For each input sentence, the system recursively paraphrases it in the best-first order until no transfer rule is applicable to the result. The current version of KURA simply gives priority to transfer rules according to the order of their loading.
- *Multi*: For each input sentence, the system tries to apply all permutations of transfer rules.

In any mode, the revision component checks the resulting sentence every time a transfer rule is applied.

The Single mode is useful for debugging each transfer rule independently, whereas the Best (or Multi) mode can be used for checking the effects of

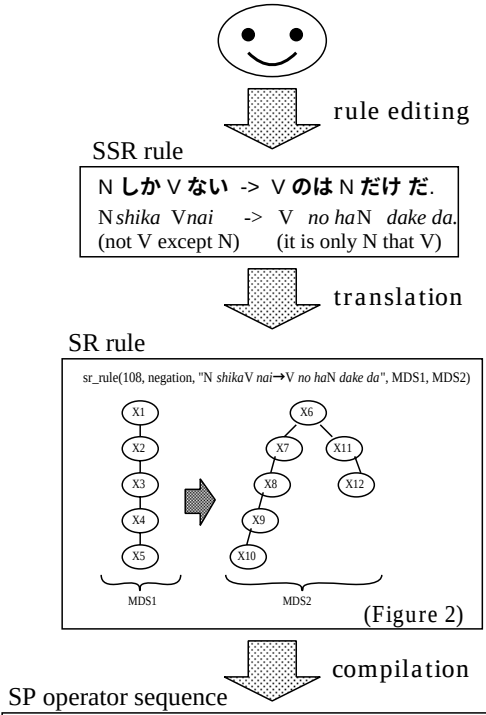

Figure 5: Rule compilation

combining more than one transfer rule. Since KURA is designed to manage all results of rule applications using a relational database as we will mention later, users can easily browse through and analyze these results with an appropriate user interface.

4.2 Data-driven rule application

If you try to apply a huge number of transfer and revision rules to a large-scaled corpus, the computational efficiency will be an issue. To improve the efficiency of rule retrieval, we have so far implemented a simple mechanism of indexing transfer and revision rules as illustrated in Figure 6.

First, for each rule, its index node is chosen to be the most specific (i.e., the most discriminative) morpheme node of the source parts of the rule. Here a lexicalized node is considered more specific than a lexically unbound node. A node anchored with a less frequent lexical entry is considered even more specific. For rule `(108)` given in Figure 2, for example, node $X4$ is chosen as an index node. Then, given a set of tuples

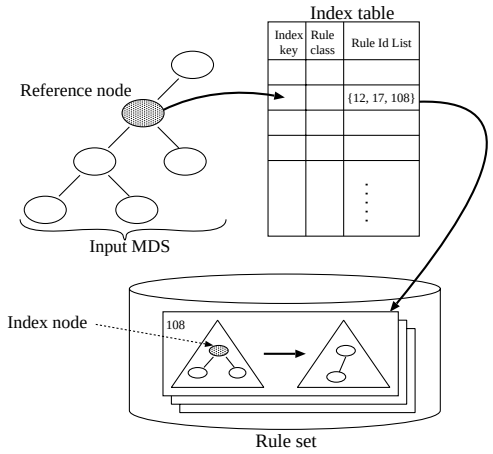

Figure 6: Data-driven rule retrieval

⟨*index_key*,*rule_id*,*rule_class*⟩, where *index_key* is a string deterministically computed from the original index node, one can straightforwardly make an index table as shown in Figure 6. Finally, in rule retrieval, the system traverses a given MDS, and for each node it visits, it refers to the index table to retrieve a set of rules that may match the current node.

This method can be improved in several directions. For example, one may put restrictions on the range of traversal of a given MDS in the revision process, to avoid unnecessary application of revision rules to unchanged subparts of the MDS. One may also improve the rule indexing method by using, for example, node n-gram patterns or decision trees (Utsuro, 1995). Improving the indexing method will be the subject of our future work.

4.3 Optimization in rule compilation

We have also implemented an optimization technique for rule compilation that improves the computational efficiency of rule retrieval in manner analogous to that of rule optimization techniques developed for production systems.

Let us reconsider rule `(108)` given in Figure 2. The rule matching process starts with the index node:

(18) `match(ReferenceNode,`
`X4=[pos:postp,lex:`*shika*`]).`

Then the question is in which order the subsequent matching operations should be carried out. In Figure 2, giving priority to checking the ancestors of `X4` up toward another lexicalized node, `X1`, referred to by the third operator in Figure 5 is obviously a better choice than checking `X5` first, since `X5` will match any noun and is therefore much less discriminative than`X1`.

Likewise, the order of matching operations can be individually optimized for each rule based on the following heuristics:

- Nodes whose mother or daughter has already been checked have priority.
- Operations checking for lexical conditions have priority. In particular, less frequent words should be given priority.
- Operations checking for semantic conditions should be delayed because they may require a time-consuming procedure of semantic analysis.

5 Installation and user interface

The backbone of the current version of KURA is implemented on a Prolog-like typed feature unification system called LiLFeS (Makino *et al.*, 1997); that is, we use LiLFeS in the implementation layer in Figure 3. In this layer, MDSs are represented as a typed feature structure, and each SP operator is implemented as a LiLFeS program. For morpho-syntactic analysis and rule translation, we use Chasen (Asahara and Matsumoto, 2000) and Cabocha (Kudoh and Matsumoto, 2000).

To give the reader an idea about the time efficiency of the current system, we conducted a small experiment. First, we prepared 768 pre-compiled transfer rules associated with a negative expression like the one in rule (8), and 65 pre-compiled revision rules. Next, we collected 38,383 pre-parsed input sentences from the Kyoto corpus (Kurohashi and Nagao, 1997). Then, we put them into the system running on a 400MHz Sparc processor in the Single rule application mode. The system was told to try all the combinations of 38,383 sentences by using 768 transfer rules. As a result, the system produced 8,619 paraphrases. The time consumed for this task was no more than 110 minutes.

KURA also provides a computational environment that facilitates process monitoring, error analysis, and rule debugging. Given input sentences and a rule set, KURA produces a set of XML log files, each of which is then converted to be imported into a relational database (RDB). Figure 7 shows a snapshot of our error analysis using commercial RDB software, where one can see (a) a source, intermediate, and target sentences (on the left of the background window), (b) the list of rules applied in the current paraphrasing instance (in the middle), (c) various tags manually annotated for evaluation and error analysis (on the right), and (d) a list of paraphrasing instances where a certain rule is applied (in the foreground window).

6 Conclusion

We presented our new lexico-structural paraphrasing engine KURA, discussing its underlying architectural and implementational issues. KURA is characterized by its three-layered and revision-based architectural design and rule-based uniform knowledge representation. KURA is designed to sup-

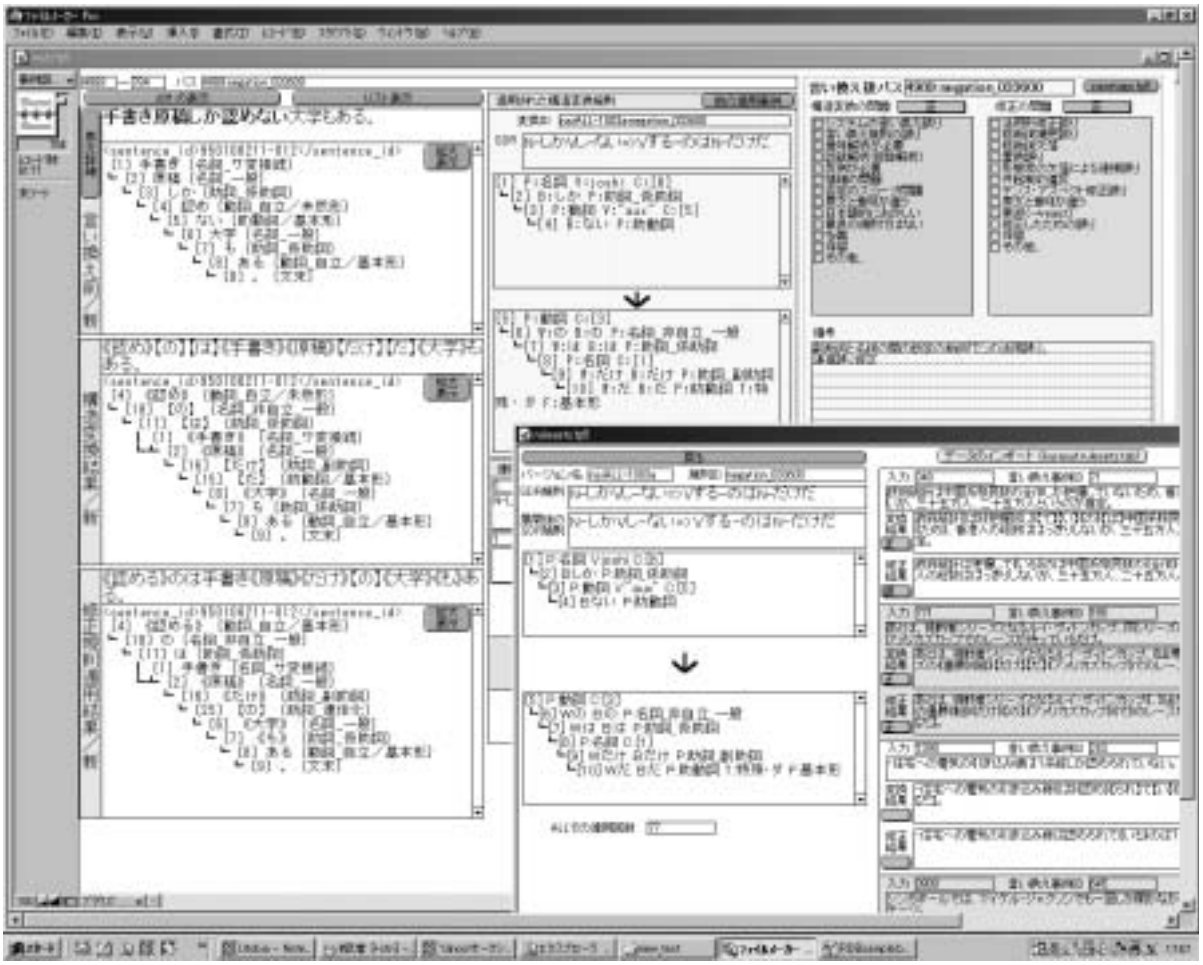

Figure 7: A snapshot of error analysis

port large-scaled experiments on rule-based paraphrasing, which we hope will help researcher explore the nature of paraphrasing.

Our revision-based paraphrasing model is closely related to the SANDGLASS MT model (Yamamoto *et al.*, 2001), which incorporates paraphrasing modules into both the source and target monolingual processors to maximally reduce the burden on the transfer module. Our claim is that paraphrasing in itself is so complicated that we need to devise a method to reduce the load of transfer. Our revision-based model is an attempt to achieve this goal.

KURA is still in the preliminary stage of development; it needs to be made more sophisticated in various directions. First, we need to address the issue of conflict resolution. The constraint-based approach to sentence planning proposed by Beale et al. (1998) may be a good start in this direction. Second, we would like to devise a more efficient method for rule indexing; here we can learn much from the work done by Utsuro (1995) and Bohnet and Wanner (2001). Third, the expressibility of transfer rules also needs to be enhanced to enable handling more complicated combinations of transfer patterns (Inui and Nogami, 2001). Finally, it is also important to try to link the work on the system with the work on the application-dependent layer (Chandrasekar *et al.*, 1996; Inui and Yamamoto, 2001).

The system is available for free at `www.pluto.ai.kyutech.ac.jp/plt/inui-lab/tools`. It can work wherever LiLFeS, Chasen, and Cabocha, which are all available free of charge, work properly.

Acknowledgments

This research has been supported by PREST, Japan Science and Technology Corporation. We would like to thank Takaki Makino and his colleagues (Tokyo University) for allowing us to use LiLFeS and for their helpful advice on LiLFeS programming, Yuji Matsumoto and his colleagues (Nara Advanced Institute of Technology) for allowing us to use ChaSen and CaboCha, and the NTT Communication Science Laboratories for providing us with their case frame dictionary and thesaurus. We also thank the anonymous reviewers for their suggestions and encouraging comments.

References

Asahara, M. and Matsumoto, Y. Extended Models and Tools for High-performance Part-of-Speech Tagger. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING)*, 2000.

Barzilay, R. and McKeown, K. Extracting paraphrases from a parallel corpus. In *Proceedings of the 39th Annual Meeting and the 10th Conference of the European Chapter of Association for Computational Linguistics*, pp. 50–57, 2001.

Beale S. and Nirenburg S. and Viegas E. and Wanner L. De-Constraining Text Generation. In *Proceedings of the 9th International Workshop on Natural Language Generation*, pp. 48–57, 1998.

Bohnet, B. and Wanner, L. On using a parallel greph rewriting formalism in generation. In *Proceedings of the 8th European Workshop on Natural Language Generation*, pp. 47–56, 2001.

Carroll, J., Minnen, G., Canning, Y., Devlin, S. and Tait, J. Practical simplification of English newspaper text to assist aphasic readers. In *Proceedings of AAAI-98 Workshop on Integrating Artificial Intelligence and Assistive Technology*, 1998.

Chandrasekar, R., Coran, C. and Srinivas, B. Motivations and methods for text simplification. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, 1996.

Dorna, M., Frank, A., Genabith, J. and Emele, M. Syntactic and Semantic Transfer with FStructures. In *Proceedings of the Thirty-Sixth Annual Meeting of the ACL*, 1998.

Hayashi, R. A Three-level Revision Model for Improving Japanese Bad-styled Expressions. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING)*, 1992.

Inui, K. A text simplification system for congenitally deaf readers. In *Proceedings of the ANLP-2001 Workshop on Automated Paraphrasing*, 2001. (In Japanese)

Inui, K. and Nogami, M. A paraphrase-based exploration of cohesiveness criteria. In *Proceedings of the eighth European Workshop on Natulan Language Generation*, pp. 101–110, 2001.

Inui, K. and Yamamoto, S. Corpus-based acquisition of sentence readability ranking models for deaf people. In *Proceedings of Natural Language Processing Pacific Rim Symposium*, 2001.

Inui, K., Tokunaga, T. and Tanaka, H. Text revision: a model and its implementation. In R. Dale, E. Hovy, D. Rosner, and O. Stock (eds.), *Aspects of Automated Natural Language Generation*, pp. 215–230, Springer-Verlag, 1992.

Kudoh, T. and Matsumoto, Y. Japanese Dependency Analysis Based on Support Vector Machines. In *Proceedings of the 2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC)*, 2000.

Kurohashi, S. and Nagao, M. Building a Japanese parsed corpus while improving the parsing system. In *Proceedings of Natural Language Processing Pacific Rim Symposium*, pp. 451–456, 1997.

Lavoie, B. Kittredge, R. Korelsky, T. Rambow, O. A framework for MT and multilingual NLG ystems based on uniform lexico-structural processing. *ANLP-NAACL*, 2000.

Makino T., Torisawa, K. and Tsujii, J. LiLFeS: practical programming language for typed feature structures. In *the Proceedings of Natural Language Pacific Rim Symposium (NLPRS)*, 1997.

Mani, I., Gates, B., and Bloedorn, E. Improving Summaries by Revising Them. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 1999.

Mellish., C.S. Some chart-based techniques for parsing ill-formed input. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*,1989.

Meyers, A., Yangarber, R. and Grishman, R. Alignment of shared forests for bilingual corpora. In *Proceeding of the 16th International Conference on Computational Linguistics (COLING)*, pp. 460–465, 1996.

Nanba, H. and Okumura, M. Producing more readable extracts by revising them. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING)*, 2000.

Richardson, S., Dolan, W., Menezes, A. and Corston-Oliver, M. Overcoming the customization bottleneck using example-based MT. In *Proceedings of the 39th Annual Meeting and the 10th Conference of the European Chapter of Association for Computational Linguistics*, pp. 9–16, 2001.

Robin, J. and McKeown, K. Empirically designing and evaluating a new revision-based model for summary generation. *Artificial Intelligence*, Vol. 85, No. 1–2, pp. 135–179, 1996.

Sekine, S. Extracting synonymous expressions from multiple newspaper documents. In *Proceedings of the ANLP-2001 Workshop on Automated Paraphrasing*, 2001. (In Japanese)

Shirai, S., Ikehara, S., Yokoo, A. and Ooyama, Y. Automatic rewriting method for internal expressions in Japanese to English MT and its effects. In *Proceedings of the second International Workshop on Controlled Language Applications (CLAW-98)*, pp. 62–75, 1998.

Takahashi, Z. and Ushijima, K. Measuring clearity of computer manuals, *Journal of Information Processing Sciety of Japan*, Vol. 32, No. 4, pp. 460–469, 1991. (In Japanese)

Takeishi, E. and Hayashi, Y. Dividing Japanese complex sentences base on conjunctive expression analysis. *Journal of Information Processing Sciety of Japan*, Vol. 33, No. 5, 1992. (In Japanese)

Utsuro, T. Optimizing nearest neighbor retrieval by similarity template and retrieval query generation. Information Science Technical Report NAIST-IS-TR95002, Graduate School of Information Science, Nara Institute of Science and Technology, January 1995.

Wahlster, W. (ED.). Verbmobil: Foundations of Speech-to-Speech Translation. Springer, 2000.

Yamamoto, K., Shirai, S., Sakamoto, M. and Zhang, Y. SANDGLASS: twin paraphrasing spoken language translation. In *the 19th International Conference on Computater Processing of Oriental Languages (ICCPOL2001)*, pp. 154–159, 2001.

Yoshimi, T. and Sata, I. Automatic preediting of English newspaper headlines and its effects in an English-to-Japanese MT system. In *Proceedings of Natural Language Processing Pacific-Rim Symposium (NLPRS)*, pp. 275–279, 1999.