An implementation of Japanese Grammar based on HPSG

Yoshiko FUJINAMI

MSc Information Technology: Natural Language Processing
Department of Artificial Intelligence
University of Edinburgh
1996

1 Abstract

In this thesis, I shall show how Japanese Phrase Structure Grammar [Gun87] can be implemented in LexGram [KB95], an amalgam of Categorial grammar [Lam61] and Head Driven Phrase Structure Grammar [PS94]. The thesis presents a parser which covers three characteristic phenomena in Japanese: (1)word order variation, (2) gaps in a sentence, and (3)relativization. To cover word order variation, the argument list for a verb, which specifies its subcategorization requirements, is defined as a set of arguments rather than as a list. The gaps in a sentence are dealt with by giving verbs categories additional to the basic ones. As for relativization, verb adnominal forms are defined in a similar way to adjectives without using traces. The parser was tested against data collected from Japanese Map Task Corpus [AIK$^+$94]. We have shown that the parser can parse sentences in reasonable speed, an average of 271.77 msec. per sentence with an average of 6 words per sentence.

2 Acknowledgements

I wish to thank Dr. Chris Mellish for his helpful comments and guidance on my project, and Dr. Esther König for her advice and support on the grammar and thesis. I also thank Dr. Carl Vogel who gave me lectures on CUF and HPSG, and Kei Yoshimoto for his comments on JPSG.

The Map Task corpora were valuable sources to test the parser on, and I thank people who have worked on them. As for the Japanese version, I am specially grateful to Syun Tutiya and Hanae Koiso, who provided me with the corpus and related documents.

Contents

List of Figures

List of Tables

3 Introduction

It is assumed in Artificial Intelligence (AI) that understanding language should be computable. Linguistics, on the other hand, aims to find theories which adequately describe phenomena in natural language without unnecessarily being concerned with computability. Head Driven Phrase Structure Grammar (HPSG) [PS94] is one proposal which attempts to satisfy the requirements from both sides.

HPSG is a unification based grammar that makes use of sort-resolved, fully well-typed feature structures [Car92]. To implement HPSG on computers, we need a formalism which can support typed feature structures and unification on them. The comprehensive Unification Formalism (CUF) [DD93] is one such tool which can be used to formalise HPSG, but CUF needs a huge search space even to parse simple sentences since it is a general constraint-based logic programming language and uses no domain specific knowledge for resolution.

LexGram [KB95] is developed in the CUF system, intended to be a useful tool for actual grammar writing. By amalgamating Categorial Grammar [Lam61] and HPSG, LexGram constrains the search space; it restricts trees to be binary and allows only two hard coded rules. With the LexGram system, grammar writing becomes just lexicon writing.

Japanese Phrase Structure Grammar(JPSG) [Gun87] is a unification based grammar and is loosely based on advances from Generalised Phrase Structure Grammar(GPSG) [GKPS85] and Head Driven Phrase Structure Grammar(HPSG). A parser [HTH^{+}86] was developed based on JPSG at the Institute for New-Generation Computer Technology(ICOT) in Tokyo.

The aim of this project is to implement a parser based on JPSG using LexGram and to test whether the restrictions provided by LexGram becomes a barrier for the implementation. The parser is evaluated against data collected from Japanese Map Task Corpus [AIK+94].

3.1 Overview

In this thesis, I will explain LexGram (Section 4) and JPSG (Section 5) briefly with examples. Then, I will explain how a parser based on JPSG can be implemented in LexGram (Section 6).

I will analyse collected data from Japanese Map Task Corpus (Section 7) and discuss possible grammars for the phenomena (Section 8). Then, the grammar will be evaluated (Section 9). Finally, I will conclude this thesis (Section 10)

4 LexGram

LexGram [KB95] is a feature-based version of Lambek Categorial Grammars, which can be considered as a lexicalized version of Head Driven Phrase Structure Grammar(HPSG) [PS94]. LexGram has been build on top of the Comprehensive Unification Formalism(CUF) system [DD93], a constraint-based logic programming language. Feature structures in LexGram are deal with by the CUF system.

LexGram is not the only unification-based categorial grammar, and there are others such as Unification Categorial Grammar(UCG) [Zee88] and Categorial Unification Grammar(CUG) [Usz86]. Compared to those that are developed based on PATR-II and Prolog, LexGram benefits from the following CUF properties:

- Typed feature terms and untyped first-order terms are supported.
- CUF sorts allow recursive data structures.
- Goals in sorts can be suspended.

In addition to these, LexGram supports HPSG's principled way of treating nonlocal dependencies.

In the following sections, I explain

- how a grammar for English transitive verbs can be defined in Categorial Grammar,
- what are the built-in feature structures in LexGram,
- how a grammar can be encoded in LexGram with the features,
- what the Fundamental Phrase Structure Schemata are and how they are used in LexGram.

- how the phenomena of subject-verb agreement and nonlocal dependencies in English can be described in LexGram using CUF feature terms.

4.1 Pure Categorial Grammar

Pure Categorial Grammar is weakly equivalent to context-free grammar. I show how a categorial grammar can capture the same syntactic facts, in this case those involving English transitive verbs, as the following context-free rules in (1) do, where S means sentence, NP noun phrase, VP verb phrase, and TV transitive verb.

(1) r1. S $\rightarrow$ NP VP

r2. VP $\rightarrow$ TV NP

l1. TV $\rightarrow$ *'loves'*

l2. NP $\rightarrow$ *'Ken'*

l3. NP $\rightarrow$ *'Naomi'*

In Categorical Grammar, each category is either a basic category or a complex category, which is also called a functor category. A complex category is expressed by concatenating basic categories with slashes (\, /), with a notation defined as follows:

Definition 1 *Notation for categories (Steedman Notation)*
Basic categories are categories.
If X and Y are categories, then X/Y and X\Y, too, are categories.
(X is called the VALUE CATEGORY *and Y is the* ARGUMENT CATEGORY.*)*

In the above notation, the value category appears always to the left of the argument category, and the slashes (\, /) show the direction of arguments.

For example, the category S/NP means that it takes the argument category NP on the right and results in the value category S. On the other hand, the category S\NP means that it takes the argument category NP on the left and results in the value category S. Thus, a functor category can been seen as a function which takes as its argument categories in the direction specified by slashes, and results in a value category.

With the following Function Application Rules (Definition 2), functor categories play the rule of functions described above.

Definition 2 *The Function Application Rules:*

FORWARD APPLICATION	$X/Y \ Y \ \rightarrow X$
BACKWARD APPLICATION	$Y \ X\backslash Y \ \rightarrow X$

FORWARD APPLICATION means that if an expression with a category Y follows an expression with a category X/Y, then those two expressions form an expression of a category X.

BACKWARD APPLICATION means that if an expression with a category Y precedes an expression with a category X/Y, then those two expressions form an expression of a category X.

Let us go back to the phrase structure rules in (1) and rewrite them in Categorial Grammar. In the functional view, the first rule, S $\rightarrow$ NP VP, can be read as *"*VP *takes* NP *on the left and becomes* S*".* Thus, a category VP can be expressed as S\NP in Categorial Grammar.

Reading the second rule in the same way, TV can be expressed as (S\NP)/NP. Thus, the rules are translated to the following definitions in Categorial Grammar:

(2) l1. loves := (S\NP)/NP

l2. Ken := NP

l3. Naomi := NP

The point is that the rules in (1) are all encoded here in the lexicon. Then, the grammar regards a sentence *"Ken loves Naomi"* as the result of the following derivation (Figure 1) with the Function Application Rules. The symbol $>$ in Figure 1 shows the FORWARD APPLICATION and $<$ the BACKWARD APPLICATION. The derivation corresponds to the parse tree (Figure 2) which the phrase structure rules (1) produce.

$$\dfrac{\dfrac{\textit{`Ken'}}{NP} \quad \dfrac{\dfrac{\textit{`loves'}}{(S\backslash NP)/NP} \quad \dfrac{\textit{`Naomi'}}{NP}}{S\backslash NP}\,>}{S}\,<$$

Figure 1: A derivation

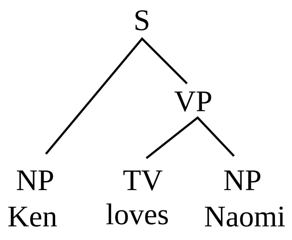


Figure 2: A parse tree

4.2 Feature Structures

In LexGram, a grammar is a mapping from words to syntactic categories. A grammar is defined by the following one place CUF sort:

```
l(afs) -> cat.
```

where `cat` is the CUF type for syntactic categories, and `afs` is a predefined CUF type. The definition for `cat` is shown in (3).

Feature structures in CUF, as the definition(3) shows, have feature and value pairs, with the notation `feature:value` where values are also feature structures. Types in CUF also denote set of feature structures, and are defined as shown in (3). The type `cat` has two features, `root` and `leaves`, and feature `root` also has values which are feature structures denoted by type `root`. `list` is a predefined CUF type.

(3)
```
cat::
  root: root,
  leaves: list.   %list of goal category

root::
  syn: nonterminal,
  sem: sem.
```

The `root` type is predefined as being a pair of a `nonterminal` and a partial semantic structure `sem`. The definitions for `nonterminal` and `sem` are left for grammar writers. The `leaves` value is a list of goal categories, which has the following features in addition to the above `cat` structures.

(4)
```
goal::
  dir: direction,
  slash: list,
  constraints: constraints.

direction = {left, right}.
```

The feature `dir` shows the direction in which the argument category exists, and `slash` shows the traces for nonlocal dependencies. How these features are used is shown in the later sections. [1]

4.3 A Grammar in LexGram

As is mentioned above, a complex category such as (S\NP)/NP is a function that takes the first argument NP on the right and the second argument NP on the left, to result in a category S. On the other hand, a basic category such as NP can also be regarded as a function which takes no argument and becomes the same category.

In LexGram, arguments are expressed as a feature `leaves` and the resulting category as a feature `root:syn`. The order of the feature `leaves` specifies the order of applying arguments to a function. The direction of each argument is specified by the feature `dir`. Thus, the simplest definitions for *'loves'*, *'Ken'*, and *'Naomi'* in LexGram are as follows, where & means conjunction:

(5) a. l(Ken) := (root:syn:np & leaves:[]).
b. l(Naomi) := (root:syn:np & leaves:[]).
c. l(loves) := (root:syn:s & leaves: [(root:syn:np & dir:right & leaves:[]), (root:syn:np & dir:left & leaves:[])])

Notice here that the verb *'loves'* is defined as a function which takes two noun phrases, its object and subject, to result in a category `s`. The value of feature `root:syn` has to be a basic category.

With the Fundamental Phrase Structure Schemata, explained in the next section §4.4, *"Ken loves Naomi"* can be recognized as a sentence.

[1]Although there are built-in `constructor sorts` to denote the above built-in feature structures, the examples are explained without using the sorts. For example, to denote the feature structure `root:syn`, LexGram users have to use the `constructor sort cons_root/2` and so on.

4.4 The Fundamental Phrase Structure Schemata

There are two rules in LexGram: the Fundamental Phrase Structure Schema(1) (Figure 3) which plays the same role in LexGram as the Function Application Rules in Categorial Grammar; the Fundamental Phrase Structure Schema(2) (Figure 4) which corresponds to the Nonlocal Feature Principle in HPSG. Both schemata apply to a pair of phrases at the same time.[2] Let us see the schemata in more detail.

If a string is a functor category, such as (root:R & leaves: [A&dir:D | Leaves] in (Figure 3), and there is a string which can unify with the first element of the feature `leaves`, A' in this case, in the direction specified by the feature `dir`, they result in a category (root:R & leaves: [Leaves]).

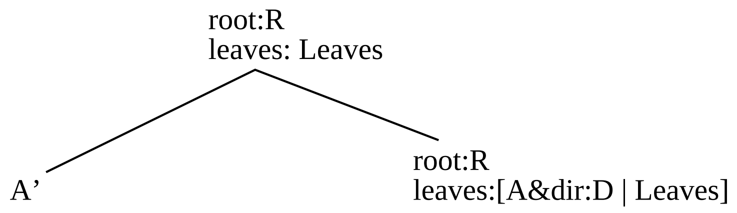


Figure 3: Fundamental Phrase Structure Schema(1)

When traces are specified using the feature `slash`, the Fundamental Phrase Structure Schema(2) (Figure 4) propagates that value to the daughters. Slash values relate to the phenomena such as relativization. In Figure 4, S_i indicates `slash` values, and $S_i \cup S_j$ means the union of the `slash` values S_i and S_j. A and A' are unifiable with the respect to the treatment of the slash value.

With the Fundamental Phrase Structure Schemata and the grammar in example (5), a string *"Ken loves Naomi"* can be recognized as a sentence in

[2]The terminology, schema, is borrowed from HPSG.

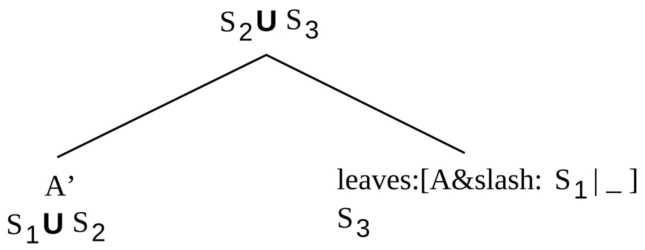


Figure 4: Fundamental Phrase Structure Schema(2)

the following way (Figure5 and Figure6).

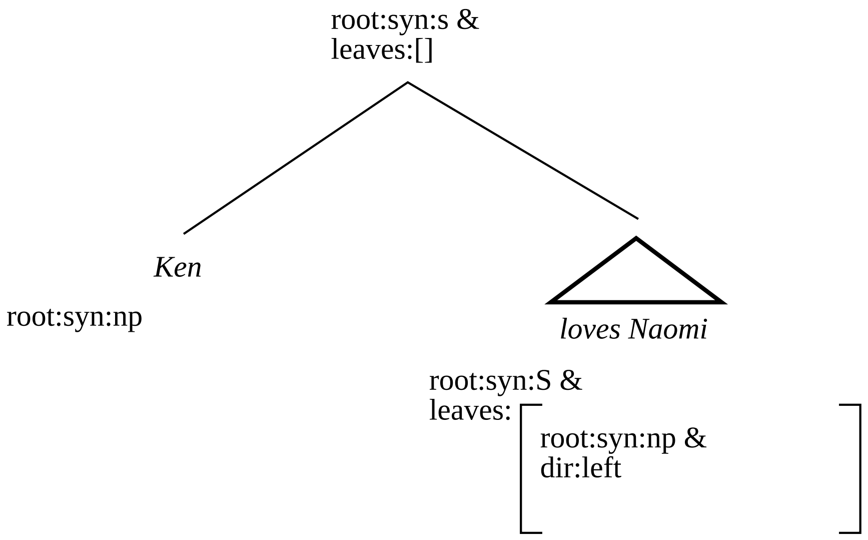


Figure 5: *"Ken loves Naomi"*

4.5 Grammars in LexGram

In this section, I show how subject verb agreement can be encoded using the feature structures explained in §4.2. Then, I rewrite the grammar using CUF sorts and show how CUF feature terms can be utilised to keep modularity in grammars. Finally, I show an example of nonlocal dependencies.

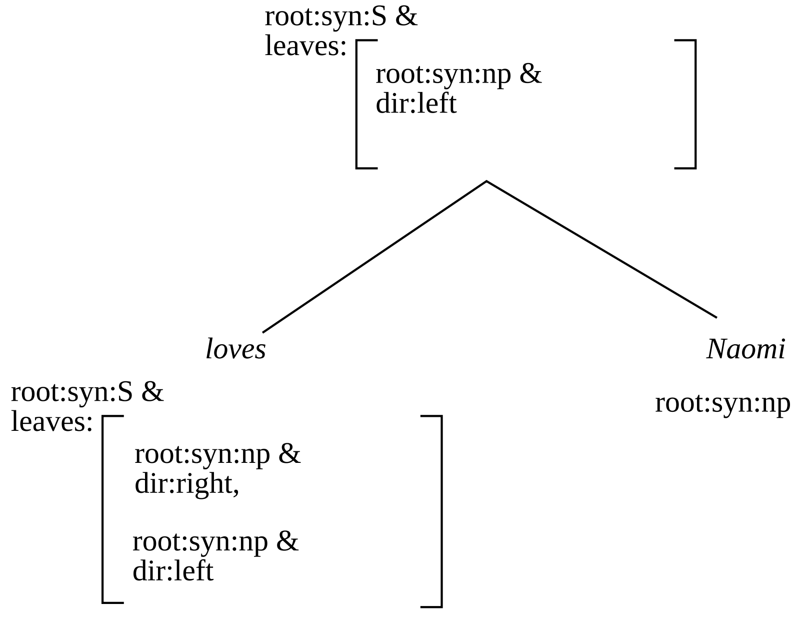


Figure 6: *"loves Naomi"*

4.5.1 Agreement phenomena

In LexGram, agreement can be expressed using features. For example, *'Ken'* can have a feature `root:syn:agr` and *third_sing* as its value. Then, the second argument of *'loves'*, which is the subject for *'loves'*, must also be constrained to have that feature value. Thus, the revised version for *'Ken'*, *'Naomi'* and *'loves'* is as follows:

(6) a. `l('Ken'):= root:syn:(np& agr:third_sing)& leaves:[].`

b. `l('Naomi'):= root:syn:(np& agr:third_sing) & leaves:[].`

c.
```
l(loves) := root:syn:s &
      leaves:[root:syn:np& dir:right& leaves:[],
               root:syn:(np& agr:third_sing)& dir:left& leaves:[]].
```

4.5.2 Using CUF Sorts

CUF sorts denote sets of feature structures, and can express equations and inequations over values of paths in feature structures. Sorts can have variables which denote substructures of feature structures. Sorts are defined with the := operator, and can be used in a similar way to the predicates in Prolog.

The following example(8) defines the same grammar as example(6) using the CUF sorts `noun`, `singular`, and `vt`. For example, `l('Ken')` in (8) denotes the following feature structure using the CUF sorts `noun` and `singular`.

(7)
$$\begin{bmatrix} \text{root:syn} & \begin{bmatrix} \text{np} \\ \text{agr:third\_sing} \end{bmatrix} \\ \text{leaves} & \langle\rangle \end{bmatrix}$$

The feature structure(7) is the same as the one denoted by (6a). With CUF sorts, we can denote feature structures without writing all paths in the structures.

(8) a. `l('Ken'):= noun_p& singular.`
b. `l('Naomi'):= noun_p& singular.`
c. `l(loves):= vt& leaves: [noun_p, noun_p&singular].`

```
noun_p:= root:syn:np & leaves:[].
singular:= root:syn:agr: third_sing.
vt := root:syn:s& leaves:[noun_p&dir:right, noun_p&dir:left].
```

4.5.3 Adjunction

Traditionally, adjuncts are treated as functor categories in CG. For example, an English adjective *'pretty'* can be defined as a functor category NP/NP which takes a noun phrase on the right and results in a noun phrase as shown in Figure 7.

$$\dfrac{\dfrac{\textit{`pretty'}}{NP/NP} \qquad \dfrac{\textit{`Naomi'}}{NP}}{NP}\,>$$

Figure 7: A derivation of adjective

In LexGram *'pretty'* can be defined in the following way. It takes a noun phrase on the right as its argument, to become a noun phrase keeping that noun phrase's feature `leaves`, L, as the remainder of its own feature `leaves`. The definition for *'pretty'* is shown below.

(9) a. l(pretty) := adj.

b. adj:= (root:syn:np &
leaves: [noun_p& dir:right &leaves:L | L]).

4.5.4 Traces

In LexGram, traces are described as the feature values of `slash` of a complement. For example, the English relative pronoun *'whom'* can be defined as a category (NP\NP)/$\bar{\mathrm{S}}$ where $\bar{\mathrm{S}}$ is a category S which includes a trace, ϵ, as shown in Figure 8. *'whom'* takes a complement which has a slash value, [`noun_p`], as shown in example(10).

(10) a. l(whom) := (root:syn:np &
leaves: [s_slash &dir:right, noun_p&dir:left]).

b. s_slash := (root:syn:s &
leaves: [] &
slash: [noun_p]).

$$\cfrac{\cfrac{\textit{‘Ken’}}{NP} \qquad \cfrac{\cfrac{\textit{‘whom’}}{NP\backslash NP/\bar{\mathrm{S}}} \qquad \cfrac{\textit{‘Naomi loves } \epsilon\textit{’}}{\bar{\mathrm{S}}}}{NP\backslash NP}\, >}{NP}\, <$$

Figure 8: *“Ken whom Naomi loves”*

5 Japanese Phrase Structure Grammar

Japanese Phrase Structure Grammar(JPSG) [Gun87] is a Japanese grammar which is loosely based on GPSG and HPSG. It is a revision and extension of Gunji's 1981 thesis: A Phrase Structural Analysis of the Japanese. A comprehensive grammar and its parser was developed based on the thesis at the Institute for New-Generation Computer Technology (ICOT) in Tokyo.

Since in JPSG many items of information about syntactic structures are moved from rules to the lexicon, it has just one phrase structure rule, M → D H (M for mother, D for daughter, and H for head). This means that a head comes at the last position in the binary branch. With its feature principles, such as the Head Feature Principle(HFP) and the Subcat Feature Principle (SFP), given a lexical item to fill in the H, the values of the HEAD features(§5.2) and the SUBCAT features(§5.3) of M and H are determined by HFP and SFP respectively, and the category D is determined by the SUBCAT value of H.

In the following sections, I will outline the basic idea of JPSG, its Phrase Structure Rule, HFP, and SFP, comparing it with traditional context-free grammars. Other principles will be explained in later chapters when they are necessary for explanations.

5.1 Categories in JPSG

Consider the following context-free rules for a fragment of Japanese involving transitive verbs. The word *'aisiteru'* in the example, means *'love'*, *'Ken'* and *'Naomi'* are peoples' names, and *'ga'* and *'wo'* are post positions.

(11) r1. S → PP VP
     r2. VP → PP TV
     r3. PP → NP P

l1. TV → *'aisiteru'*
l2. NP → *'Ken'*
l3. NP → *'Naomi'*
l4. P → *'ga'*
l5. P → *'wo'*

In the above context-free rules, categories are expressed by mnemonics such as S, VP, and so on. In JPSG, the categories are expressed by a set of syntactic features, the `POS`(Part Of Speech) feature (one of the Head features) and the `SUBCAT` (SUBCATegorization) feature. The categories are distinguished by the values of these features. For example, the category S has a value *V* for `POS` and an empty set for `SUBCAT`. The example (12) shows the difference between the traditional and JPSG expressions for categories.

(12) Categories:
S: {POS V; SUBCAT {}}
PP: {POS P; SUBCAT {}}
VP: {POS V; SUBCAT {PP[SBJ]}}
TVP:{POS V; SUBCAT {PP[SBJ], PP[OBJ]}}
NP: {POS N; SUBCAT {}}

As the result, the lexicon is defined in JPSG as follows:

(13) Lexicon:
aisiteru : {POS V; SUBCAT {PP[SBJ], PP[OBJ]}}
ken : {POS N; SUBCAT {}}
Naomi : {POS N; SUBCAT {}}
ga : {POS P; GR SBJ; SUBCAT {NP}}
wo : {POS P; GR OBJ; SUBCAT {NP}}

In the above, PP[SBJ] is an abbreviation for

{POS P; SUBCAT {}; GR SBJ}

and PP[OBJ] is an abbreviation for

{POS P; SUBCAT {}; GR OBJ}.

The feature, `GR`(GRammatical function), is another `HEAD` feature.

5.2 HEAD features and the HEAD Feature Principle

As a matter of fact, many features are passed on from the mother M to the head H in the Phrase Structure Rule, M $\rightarrow$ D H. These features are called `HEAD features`, and the relating principle is called the `Head feature Principle` (Def. 1).

Principle 1 *HEAD Feature Principle (HFP):*
The `HEAD` feature values of the head are identical to the values of the respective `HEAD` features of the mother.

For example, the feature `POS` and `GR` in example (13) are `HEAD` features, and they are passed from mother node *"Ken ga"* to its head daughter *"ga"* as shown in Figure 9.

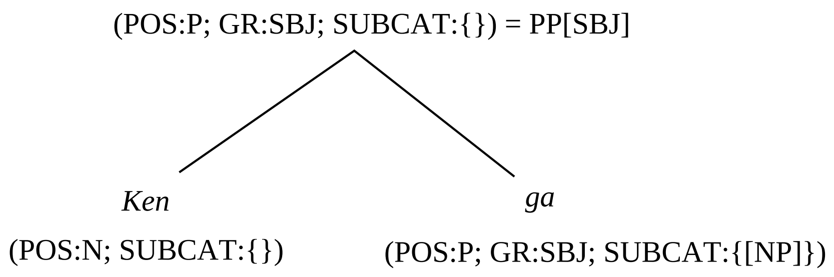

(POS:P; GR:SBJ; SUBCAT:{}) = PP[SBJ]
Ken
ga
(POS:N; SUBCAT:{})
(POS:P; GR:SBJ; SUBCAT:{[NP]})

Figure 9: *"Ken ga"*

5.3 The SUBCAT feature and the SUBCAT Feature Principle

The `SUBCAT` feature takes a set of categories as its value for which the category subcategorizes. For example, a category S subcategorizes for nothing, while another category VP subcategorizes for a subject. Thus, the `SUBCAT` values in VP and TV show a set of complements needed to make up a sentence (cf. example(12)). There are three cases in the SUBCAT Feature Principle, for complementation, adjunction, and coordination, depending on the relationship between the SUBCAT values of the three nodes in the Phrase Structure Rule(§5.4).

Principle 2 *The SUBCAT Feature Principle (SFP):* [3]

Complementation M $\rightarrow$ C H
The value of the ***SUBCAT*** *feature of the head,* H*, unifies with that of the mother,* M*, except for the category in* H*'s* ***SUBCAT*** *that unifies with the complement,* C*.*

Adjunction M $\rightarrow$ A H [4]
The value of the ***SUBCAT*** *feature of the head unifies with that of the mother.*

Coordination M $\rightarrow$ H H
The values of the SUBCAT feature of the heads unify with the mother.[5]

[3]It is not very clear how the three cases, complementation, adjunction, and coordination, are distinguished from [Gun87]

[4]In order to further specify what kind of adjunct can appear in a local tree, the feature ADJUNCT is used.
The ADJUNCT Feature Principle: the value of the feature ADJUNCT of A unifies with H. This is described on in p.26 [Gun87]

[5]Coordination is not treated in [Gun87] and will not be in this thesis.

5.4 The Phrase Structure Rule

JPSG has only one phrase structure rule.

Definition 3 *The Phrase Structure Rule (PSR) in JPSG:*

M $\rightarrow$ D H

Mother has two daughters, D *and* H*, and the last daughter is called the* ***head****.*

With the HFP, SFP, and the Phrase Structure Rule, a sentence *"Ken ga Naomi wo aisiteru"* (Ken loves Naomi) can be parsed in the following way (Figure 10 and Figure 9).

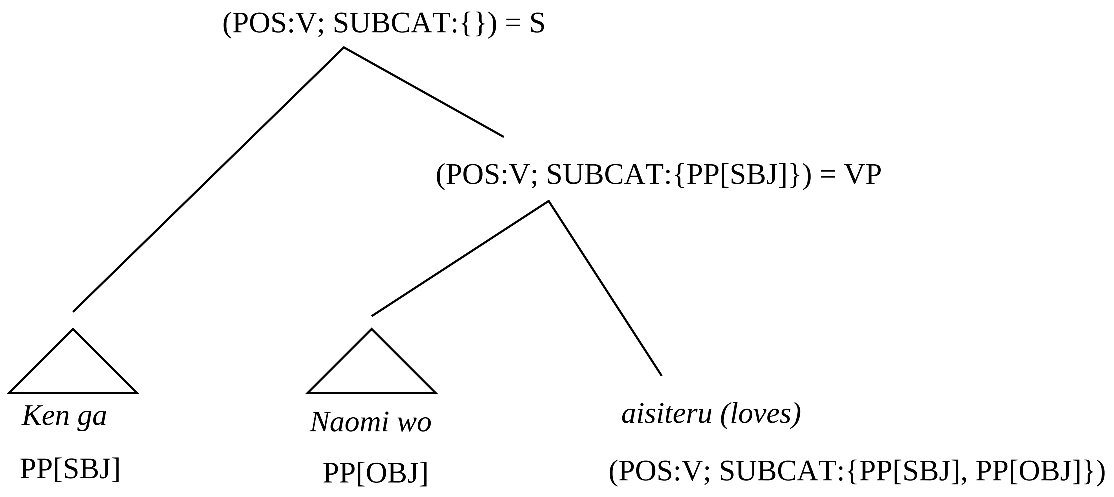
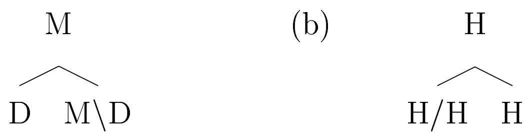


Figure 10: *"Ken ga Naomi wo aisiteru"*

6 JPSG in LexGram

JPSG is loosely based on HPSG, while LexGram can be considered as a lexical version of HPSG. There are some advantages and disadvantages in implementing JPSG in LexGram. One advantage is that the local tree encoded by JPSG is the similar kind of binary tree as the one by LexGram. Another advantage is that the HEAD Feature Principle, SUBCAT Feature Principle, and Phrase Structure Rule (§5.2, §5.3, and §5.4) correspond to LexGram's Fundamental Phrase Structure Schemata (§4.4). When the daughter in the PSR (Definition 3) is a complement, the head corresponds to a functor category in CG which takes that daughter and results in the mother node (cf. (14a)). If the daughter is an adjunct, i.e the SUBCAT value of the mother is identical to that of the head, the adjunct corresponds to a functor category which takes the head to result in the head as its mother (cf. (14b)).

(14) (a) (b)



One disadvantage is that the order of applying arguments is fixed in LexGram, while JPSG does not assume any predetermined order among complements in the `subcat` feature. Thus, it can deal with flexibility in word order. Another disadvantage is that the relativization cannot be encoded in the same way as in (§4.5.4) where the relative pronoun is defined to have a slashed sentence $\bar{\mathrm{S}}$ since Japanese does not have relative pronouns. These problems are discussed in the later chapter.

In this chapter, I will explain the following aspects of the implemented parser based on JPSG in LexGram:

- the head features of the grammar based on those of JPSG,

- the definition of the lexicon (13) in LexGram,
- the derivation of *"Ken ga Naomi wo aisiteru"* with the lexicon, and
- the semantic representations used in the parser.

6.1 Head Features

I redefined head features in the following way using typed feature structures in CUF, while JPSG does not have any types. Some features are added to the originals in JPSG to cover the phenomena discussed in §7.

The type `nonterminal`, head feature(3) in §4.2, is divided into four categories, `v`(verb phrase), `n`(noun phrase), `p`(post positional phrase), and `p2`(particle), and each one can carry its own head features. The implemented grammar does not have the head feature `POS`(Part Of Speech) in JPSG, as this is expressed by CUF types such as v and n.

```
nonterminal = v | n | p | p2.

v ::
  sub   : sub_info.

p ::
  gr    : gr_info,
  pform : pf_info,

n ::
  aform : af_info.

p2 ::
  p2form : p2_info.
```

Verb Verb has a head feature `sub`. The feature `sub` has the information about the elements of the feature `leaves` and is used for subject verb agreement. The verb adnominal form and subject verb agreements are explained in the later sections.

Post Positional Phrase `gr` (grammatical function) shows what case the postpositional particle carries, such as nominative, and `pform` (particle form) keeps the form of that particle. We need that information since the cases depend on verbs and other post positional particles.

Noun The feature `aform` (adnominal particle form) is used when a noun is constructed with adnominal particles such as *'no'* and nouns. The feature `aform` keeps the form of the particle.

Particle There are particles which are neither adnominal particles nor post positional particles, such as *'to'* in §7.3.1. `p2form` keeps the form of the particle.

6.2 Definition in LexGram

With the above head feature definitions, example(13) can be written in LexGram in the following way. The feature `leaves` in LexGram corresponds to the SUBCAT feature in JPSG.

A phrase is recognized as a sentence, when its feature `syn` is `v`, and its feature `leaves` is saturated. The derivation of *"Ken ga Naomi wo aisiteru"* with the definition (15) is shown in Figure 11.

(15) a. l('Ken') := noun.

b. l('Naomi') := noun.

c. l(ga) := pp & nominative & pform(ga).

d. l(wo) := pp & accusative & pform(wo).

e. l(aisiteru) := vt & leaves([accusative, nominative]).

```
noun := root:syn:n & saturated.
pp   := root:syn:p & leaves([noun&dir:left]).
vt   := root:syn:v & leaves([pp& dir:left, pp&dir:left]).
nominative := root:syn:gr: sbj.
accusative := root:syn:gr: obj.
saturated  := leaves([]).
leaves(L)  := leaves: L.
```

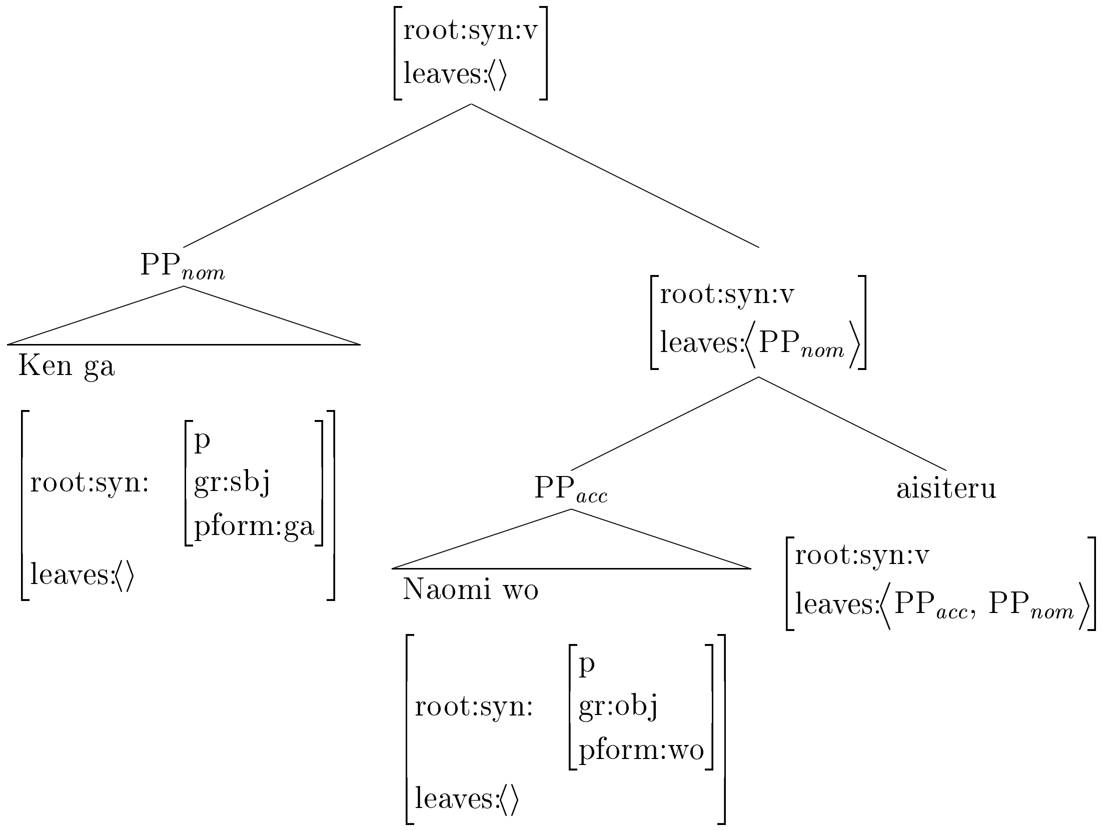


Figure 11: A derivation of *"Ken ga Naomi wo aisiteru"*

6.3 Semantic Representation

The semantic theory adopted in JPSG is based on Montague semantics and JPSG employs λ-expressions as its semantic representation.

Using λ-expressions, the definition(15) added its semantic representation could be something like (16) below, where α and β are indices to denote the identities between the parts of semantics of head categories and that of complements. P, x, and y are lambda variables. In the definition, the CUF sort `root_sem(X)` denotes the feature structure `root:sem:X`.

(16) a. l('Ken'):= noun & root_sem(λP P(k))
b. l('Naomi'):= noun & root_sem(λP P(n)).
c. l(ga):= pp & nominative &pform(ga) &root_sem(α) & leaves([root_sem(α)]).
d. l(wo):= pp & accusative &pform(wo) & root_sem(α) & leaves([root_sem(α)]).
e. l(aisiteru) : = vt & root_sem(β (λyα(λx love(x,y)))) &
leaves([accusative &root_sem(β), nominative & root_sem(α)]).

With this definition, we want to obtain a representation for the formula `love(k, n)` [6] when 'ken ga naomi wo aisiteru' is derived as a sentence successfully. For this purpose, we have to

- implement λ-expressions in feature structures,
- define representations for formulae, and
- implement some procedures for λ-conversion.

For example, λx.λy.love(x, y) [7] should be written in feature structures such as example(17), and `love(k, n)` as example(18):

[6]This is the result of the following λ-conversion :
$\lambda Q.Q(n)(\lambda y.\lambda P.P(k)(\lambda x.love(x,y))) = \lambda Q.Q(n)(\lambda y.love(k,y))$ = love(k, n).

[7]λx.λy.love(x, y) is an abbreviation for λx.λy.love(y)(x)

(17) $$\begin{bmatrix} \text{lambda: X} & \\ \text{formula:} & \begin{bmatrix} \text{lambda: Y} & \\ \text{formula:} & \begin{bmatrix} \text{rel: love} & \\ \text{args:} & \begin{bmatrix} \text{arg1: X} \\ \text{arg2: Y} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

(18) $$\begin{bmatrix} \text{rel: love} & \\ \text{args:} & \begin{bmatrix} \text{arg1: k} \\ \text{arg2: n} \end{bmatrix} \end{bmatrix}$$

However, for technical reasons, I employed various feature names instead of simulating the λ calculus and adopted the following conventions for semantic representations. Unification is also used instead of λ-conversion, although unification is less expressive than λ-conversion. [8]

Using feature names The following feature structure(19) represents the same semantic expression as $\lambda x.\lambda y.love(x, y)$ in example(17), where the feature `subj` and `obj` play the same role as λ. Using feature names, unification becomes much simpler than using λ, because we do not need to remember the whole structures. For example, when we used λ for features such as example(17), we would have had to remember which variable specifies the object to assign a value to it, because it is not clear only from the representation. Also, we would have had to specify the whole path(formula:lambda:Y) to unify the value to it.

(19) $$\begin{bmatrix} \text{subj: X} & \\ \text{obj:Y} & \\ \text{formula:} & \begin{bmatrix} \text{rel: love} & \\ \text{args:} & \begin{bmatrix} \text{arg1: X} \\ \text{arg2: Y} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

[8] For example, unification cannot simulate the conversion (λP.P(a))λX.X .

Convention-1 Proper nouns such as ‘Ken’ are treated in the same way as common nouns. For example, the meaning of ‘Ken’ is expressed by the formula $\lambda x.named(x, ken)$ and represented by example(20), while JPSG defines it as $\lambda P.P(k)$ as was seen in example(16a).

(20)
$$\begin{bmatrix} \text{index: X} & \\ \text{formula:} & \begin{bmatrix} \text{rel: named} & \\ \text{args:} & \begin{bmatrix} \text{arg1: X} \\ \text{arg2: ken} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Convention-2 Japanese does not have strict distinction between nouns and noun phrases as pointed out by Gunji([Gun87] p.202). Since there is no determiner in Japanese which corresponds to English ones, a noun can be interpreted in a number of ways. For example, a Japanese common noun such as ‘`inu`’(*dog*) can correspond not only to *a dog* but also to *the dog* and *dogs*. Thus, nouns can be regarded as being attached hidden determiners and treated as noun phrases. [9]

Because of the above reason, the parser considers every noun to be attached determiners. For example, the original meaning of Ken is expressed as $\lambda x.named(x, ken)$ and represented as example(20), but the example(21), which expresses $\lambda P.\exists x.named(x, ken) \wedge P(x)$, is recorded as the semantic representation for (cf. example(23a)).

The example(21) does not include a representation for the formula P(x) in the feature structure `forms`, but it will be added later to the tail of the list. The feature `vstore`(variable store) is used for storing the variable X to be exported to the function P.

[9]Such a use of common nouns was not considered in [Gun87]. Noun phrases in [Gun87] are limited to the ones composed of quantifiers and common nouns.

(21)
$$\begin{bmatrix} \text{vstore:X} & \\ \text{formula:} & \begin{bmatrix} \text{vars:} & \left\langle \text{X} \right\rangle \\ \text{forms:} & \left\langle \begin{bmatrix} \text{rel: named} & \\ \text{args:} & \begin{bmatrix} \text{arg1: X} \\ \text{arg2: ken} \end{bmatrix} \end{bmatrix} \right\rangle \end{bmatrix} \end{bmatrix}$$

Definition of the feature `sem` To support the above semantic representation, the type `sem`(cf. feature structure(3) in §4.2) is defined in the following way. The value of the feature `formula` has the semantic representation as its value. The feature `subj`, `obj`, `obj2`, and `index` play the same role as the λ operator.

(22)
```
sem ::
  formula : quantified_formula.

sem = verbal_sem | np_sem.

  verbal_sem ::
    subj: afs,   %nominative
    obj : afs,   %accusative
    obj2: afs.   %dative; or locative

  np_sem ::
    index: afs,
    vstore: afs,
    semf:semf.

  semf ::   % semantic information for subcat
    animate: boolean,
    attribute: att_info.

  boolean = {+, -}.
  att_info = { loc, dir }.

quantified_formula::
    vars: list,
    forms: list.

formula= basic_relation.

basic_relation ::
  label: afs,
  rel : afs,
  args: argument_frame.

argument_frame ::
  arg1 : top,
  arg2 : top,
  arg3 : top.
```

Lexicon with semantic representations As a result, the example (16) is redefined as (23). The semantic representation(24) is obtained through a derivation of 'ken ga naomi wo aisiteru', using the definition(23). The structure(24) represents the formula $\exists y.\exists x.named(y, naomi) \wedge named(x, ken) \wedge love(x, y)$ [10].

The CUF sort `n_type('Ken')` in (23) denotes the semantic representation(20), `np_semantics(n_type(ken))` the representation(21), and `vt_type('LOVE')` the representation(19).

(23) a. `l('Ken'):= noun & np_semantics(n_type('KEN')).`

b. `l('Naomi'):= noun & np_semantics(n_type('NAOMI')).`

c.
```
l(ga):=  pp & nominative &pform(ga) &root_sem(S) &
             leaves([root_sem(S)]).
```

d.
```
l(wo):= pp & accusative &pform(wo) & root_sem(S) &
             leaves([sem(S)]).
```

e.
```
l(aisiteru) : =
    vt &
    vt_type('LOVE')&
    subj:S & obj:O & forms(F)&
    root_sem(vars(append(V1, V2))&
              forms(append(F1, append(F2, F)))&
    leaves([accusative&root_sem(vstore:O&vars(V)&forms(F1)),
            nominative&root_sem(vstore:S&vars(V)&forms(F2))]).
```

[10]This is the result of the conversion:
$\lambda Q \exists y.named(y, naomi) \wedge Q(y)(\lambda w.\lambda P \exists x.named(x, ken) \wedge P(x)(\lambda z.love(z, w)))$

(24)
$$\begin{bmatrix} \text{vars: } \left\langle \text{Y,X} \right\rangle & \\ \text{forms:} & \left\langle \begin{array}{l} \begin{bmatrix} \text{rel: named} & \\ \text{args:} & \begin{bmatrix} \text{arg1: Y} \\ \text{arg2: naomi} \end{bmatrix} \end{bmatrix}, \\ \begin{bmatrix} \text{rel: named} & \\ \text{args:} & \begin{bmatrix} \text{arg1: X} \\ \text{arg2: ken} \end{bmatrix} \end{bmatrix}, \\ \begin{bmatrix} \text{rel: love} & \\ \text{args:} & \begin{bmatrix} \text{arg1: X} \\ \text{arg2: Y} \end{bmatrix} \end{bmatrix} \end{array} \right\rangle \end{bmatrix}$$

6.4 Summary

In this section, I explained (1)the head features and (2)semantic representations employed in the implemented parser. For the implementation, I redefined the head features in JPSG using typed feature structures and added some features to cover the phenomena to be discussed later. As for semantic representations, I employed various feature names and unification while λ and λ-conversion are used in JPSG. Also, I employed some conventions as explained in §6.3.

7 Phenomena in the Data

There are two approaches to the study of languages; the competence and performance approaches [Win83]. In the performance approach, real data such as corpora and the results of psychological experiments are analysed to study the mechanisms underlying the phenomena. In the competence approach, the analysis is replaced by the intuitions of native speakers about their language. The target of the study is then not the actual usage of language, but the structure of the language at a certain abstract level.

Since I am more interested in applications, I opted for the performance approach. The reason why I chose the Japanese Map Task Corpus [AIK$^+$94] for the analysis is that it records spontaneous dialogues.

In this section, firstly I explain the Japanese Map Task Corpus (JMTC) and the data I picked up from that corpus. Then I explain typical phenomena in the data:

- Noun Phrases
- Agreement
- Relative Clauses
- Flexibility in Word Order and Arguments

7.1 The Japanese Map Task Corpus

The Japanese Map Task Corpus is a collection of task-oriented dialogues aimed to support the study of spontaneous speech at many levels. The corpus uses the Map Task in which speakers must collaborate verbally. The basic design of the task follows that of the Edinburgh Map Task Corpus [ABB$^+$91].

The Map Task The task involves two participants each of whom is given a map to work with. One speaker plays the role of instruction giver in describing a route on his map to the follower whose map has only the starting point and no route. The goal of this task is that the follower must reproduce the giver's route using information obtained through communication. The maps given to the giver and follower describe the same areas but they are slightly different (See the maps for giver and follower in Appendix A). Although the participants are informed that their maps may not be identical, they do not know how they may differ.

7.2 Data Studied

For this project, I selected from fifteen dialogues in JMTC one hundred sentences which include words expressing directions or locations. I first marked these words in the corpus, and picked up seven to ten sentences from each dialogue. Then I modified this data as explained below. Since the same person tends to use the same pattern of sentences, I collected sentences from different speakers' dialogues. The data are shown in Appendix I, where each item has three lines as below; the first is the Japanese sentence in Roman font, the second is the word to word translation in English of the first, and the third is the translation of the first into English.

(s1) syupatsutiten ga sabaku no yoko ni aru
starting-point NOM *desert* NO *next-to* DAT *there-is*
"There is a starting point next to desert"

Since there are no corresponding words in English as for the Japanese particles, I indicated them by their cases or sounds written in capital letters. For example, the particle 'ga' in (s1) carries nominative case and is specified with NOM, while 'ni' is specified with DAT. The particle 'no' is specified with its sound, NO.

Overview The tense of all sentences is present and there are no determiners in the data. [11] No anaphora appears in the data, either. They are all assertive since I modified the data. The average length of a sentences is six words. The sentences whose verbs are copular such as 'aru' and 'iru'(*'there is'*) have subjects but all others do not have subjects. [12]

Modifications Because the corpus is a collection of dialogues and most sentences are incomplete or grammatically incorrect, I modified some parts of the data while keeping locative expressions and the overall meaning as in the originals. I also changed some forms of verbs and nouns which are not our target. The modifications are listed below:

1. Changing aspectual verb forms to sentence final forms so that the data become sentences

	sogen	no	tokoro	kara	masugu	**itte**,
→	sogen	no	tokoro	kara	masugu	**iku**.
	plain	NO	place	from	straight	go

   (Go straight from the plain.)

2. Adding a missing post positional particle, 'no'

	sinkohoukou		migi
→	sinkohoukou	**no**	migi
	heading direction	relative to	right

[11]Nouns in Japanese do not have different forms for singular and plural nouns and there is no corresponding word to the English word *'a'*. There are some determiners similar to *'the'* but usually native speakers do not use them.

[12]Since the hearers are denoted by the subjects in those sentences and it is obvious both to the speakers and the hearers, the native Japanese speakers omit them.

(the right relative to heading direction)

3. Adding verbs so that the data become sentences

	syupatutiten	ga	sabaku	no	yoko	ni	
→	syupatutiten	ga	sabaku	no	yoko	ni	**aru**
	starting point	NOM	desert	NO	beside	LOC	there is

(There is a starting point beside the desert.)

4. Shortening the names for landmarks

	nagarenohayai	**kawa**
→		**kawa**
	whose stream is rapid	river

(the river)

5. Deleting adjectives and intensifiers

	itiban	migi
→		migi
	most	right

(the right)

6. Changing interrogative sentences to declarative ones

	ginko	ga	migisita	ni	aru	**yone**
→	ginko	ga	migisita	ni	aru	
	bank	NOM	below right	LOC	there is	isn't it?

(There is a bank below right.)

7. Changing the honorific form to the normal form

	o	furo
→		furo
	honorific marking	bath

(bath)

Lexicon The data has nineteen different nouns for locations and directions, six different post positions, and twelve verbs (sentential form) and two verbs (adnominal form).[13] The other words are names for landmarks in the maps. The lexical items are summarised in Appendix B. In the data, particles are observed more frequently than other word classes. The adnominal particle 'no' is the most frequently observed particle followed by 'ni', 'ga', and 'wo'.

As for verbs, 'aru'(*there is*) is most frequently used, and others are verbs expressing actions for movement such as 'tooru'(*pass by*).

7.3 The Phenomena

In the following sections, I explain typical phenomena found in the data:

- Noun Phrases
- Agreement
- Relative Clauses
- Flexibility in Word Order and Arguments

7.3.1 Noun Phrases

Many noun phrases in the data are of the form 'A no B' which contains two nouns A and B with an adnominal particle 'no'. Some noun phrases include more than one 'no' in a phrase such as 'ginko no tokoro no hidari'(*the left of the place at the bank*). In the data, there are two types as to 'B': location and direction. Also, when 'B' is a word 'aida'(*space/between*), 'A' takes the form 'A1 to A2' where 'A1' and 'A2' are nouns and 'to' is another adnominal particle. In the following, I explain the relation between 'A', 'B', and 'no'.

[13]the difference between a sentential form and an adnominal form is explained in §7.3.3

Location The noun phrase 'A no B', where 'B' is a location, can be paraphrased to a relative clause. For example, 'ginko no tokoro' , where 'A' is 'ginko'(*bank*) and 'B' is 'tokoro'(*place*), can be paraphrased to 'ginko ga aru tokoro' (*the place where the bank stands*) with the following rule:

> A 'no' B $\rightarrow$ A 'ga aru' B,
> where 'ga' is a particle and 'aru' is the verb which means *exist*.

Hence, when the paraphrase rule can be applied to, 'A no' plays an adjective role to 'B' and 'B' is restricted by 'A no'.

Direction When 'B' is a direction such as 'ginko no hidari', where A is 'ginko'(*bank*) and B is 'hidari'(*left*), the above paraphrasing is not possible. Thus, what role do 'A', 'no', and 'B' play for constructing the meaning of 'A no B'? There is a common meaning between 'ginko no hidari'(*left of the bank*) and 'hidari'(*left*); Both of them denote the same direction, *left*. The difference is that 'ginko no hidari' is more specific than 'hidari'. Thus, 'A no' restricts B by adding an item of information to B.

B is the word 'aida' This is a special case of the above `Location`. When B is the word 'aida'('space'), 'A' usually takes the form of 'A1 to A2' such as (s12'). There is also a pattern involving an additional particle such as 'to'(*To**) in (s12), but this 'to' can be omitted without changing the meaning. Both (s12) and (s12') denote the same meaning. That is, the combined particle 'to no' plays the same role as 'no'.

(s12)	seitetsujo	to	ki	to	no	aida	wo	tooru
	iron-mill	*To*	*tree*	*To**	*No*	*gap*	*Wo*	*pass-through*

"Pass through the gap between the iron mill and the tree"

(s12') seitetsujo to ki no aida wo tooru
iron-mill To tree No gap Wo pass-through
"Pass through the gap between the iron mill and the tree"

To sum up, when 'B' is a location or direction, the noun phrase 'A no B' inherits the attributes from 'B', and the meaning of 'A no B' is a subset of that of 'B'. For example, the set of entities for 'ginko-no-tokoro', is a subset of entities for 'tokoro', that is ($[\![$'ginko-no-tokoro'$]\!] \subset [\![$'tokoro'$]\!]$). Similarly, ($[\![$'ginko-no-hidari'$]\!] \subset [\![$'hidari'$]\!]$) holds.

7.3.2 Agreement

Japanese does not have the same subject-verb agreement as English does, but there are other types of agreement. One is the relation between the verbs 'aru' in (s1) and 'iru' (s9) and their subjects. Although they have the same meaning, *'exist/there is'*, they need different types of subjects; the subject of 'aru' has to be animate while the subject of 'iru' should not be animate.

(s1) syupatsutiten ga sabaku no yoko ni aru
starting-point NOM *desert* NO *next-to* LOC *there-is*
"There is a starting point next to desert"

(s9) yama no migite ni kamosika ga iru
mountain NO *right* LOC *goat NOM there-is*
"There is a goat on the right of a mountain"

The following sentence(s2) has the verb 'mieru'(can see) and the post positional particle 'wa' which can be both nominative and accusative case. Since people know that only animal can see something and 'tsuirakugenba' (clash-spot) is not animate, 'tsuirakugenba' cannot be interpreted as the subject of (s2).

(s2) tsuirakugenba wa sinkouhoukou no migigawa ni mieru
clash-spot ACC *heading-direction* NO *rightS* LOC *see*
"See the clash spot on your right hand side"

The following example shows that certain verbs require particular post positional particles for case markers. For example, when a verb expresses an ability, such as 'mieru'(*can see*) in (s2'), the case of the particle 'ga' must be accusative although it is usually nominative. Thus, the post positional phrase *"ken ga"* in example(s2') cannot be interpreted as its subject although 'mieru' requires an animate subject as shown above and 'Ken' is animate.

(s2') Ken ga migigawa ni mieru
Ken ACC *right* LOC *can see*
"(You) can see Ken on the right"

7.3.3 Relative Clauses

Relativization in Japanese is different from that of English in the following respects as the corpus (s62) and [Gun87](r1) show:

- Japanese does not have relative pronouns such as *who*.
- Head nouns (or noun phrases) follow relative clauses.
- Some relative clauses are not bound by their head nouns (or noun phrases). (Type II relativization)

In the corpus(s62), 'haka no e ga ϵ owatta'(ϵ indicates a trace) is a relative clause, and is bound by the noun, 'atari'. Notice that traces in relative clauses are post positional phrases, but the postcedents are nouns (or noun phrases) such as 'atari' in (s62).

(s62) haka no e ga ϵ owatta atari wo magaru
graveyard NO *picture* NOM TRACE *end* *somewhere* ACC *turn*
"Turn around at the place where the picture of a graveyard ends"

The forms of main verbs in relative clauses such as 'owatta' in the example(s62) is called *adnominal* since it always precedes nouns (or noun phrases), and is distinguished from *sentential final form* such as 'magaru' in the example(s62)

Type II relativization such as (r1) 'hangagu ga yakeru nioi' has no trace in the relative clause except for a free gap and is not bound by the postcedent 'nioi'. Type II relativization cannot be observed in English.

(r1) hanbagu ga yakeru nioi
hamburger NOM *grill* *smell*
"the smell of grilling hamburger"

7.3.4 Flexibility in Word Order and Arguments

Japanese is flexible in its word order and arguments. Let us see first an example of flexible word ordering. Both sentences (s40) and (s40') end with the same verb, 'aru'(*exist/there is*), and have the same meaning, but the order of post positional phrases are different.

(s40) sogen ga yama no sita ni aru
plain NOM *mountain* NO *foot* LOC *there-is*
"there is a plain on the foot of the mountain"

(s40') yama no sita ni sogen ga aru
mountain NO *foot* LOC *plain* NOM *there-is*
"there is a plain on the foot of the mountain"

The following sentences (s14), (s5), and (s5') show flexibility concerning arguments. (s14) and (s5) end with the same verb 'magaru'(turn), but the arguments are marked with different particles 'ni' and 'wo'. The verb 'magaru' can also take at the same time these two post positional phrases marked by 'ni' and 'ga' as shown in (s5').

(s14)	migigawa	ni	magaru
	right	LOC	*turn-to*

"Turn to the right"

(s5)	atoti	no	kado	wo	magaru
	ruin	NO	*corner*	ACC	*turn-to*

"Turn at the corner of the ruin"

(s5')	atoti	no	kado	wo	migigawa	ni	magaru
	ruin	NO	*corner*	ACC	*right*	LOC	*turn-to*

"Turn at the corner of the ruin to the right"

7.4 Conclusion

The data selected from the corpus had the following phenomena:

- Many noun phrases have the form 'A no B', where 'A' and 'B' are nouns and 'no' is an adnominal particle. When 'B' means a location or direction, the set of entities for 'A no B' is always a subset of the set of entities for 'B'.
- Although Japanese does not have the same subject-verb agreement as English, there are some agreement phenomena between complements and verbs such that the verb 'iru'(*to exist*) always requires an animate subject.
- In Japanese, head nouns (or noun phrases) follow relative clauses. The main verb in the relative clause is in the last of the clause and comes just in front of the head noun. Because of that, the verb is called an adnominal form and is distinguished from the sentential final form.
- Many sentences in the data lack the subjects, and the number of arguments for the same verb is often different. The word order in the data is also very flexible.

Based on the analysis we have done, I will implement the following aspects in addition to the basic part described in §6.

- the adnominal particles
- agreements between complements and verbs
- relative clauses
- flexibility in word orders and arguments

8 Grammar

In this section, firstly, I show alternatives to encode the following phenomena explained in §7.

- Noun Phrases
- Agreements
- Relative Clauses
- Flexibility in Word Order and Arguments

Then, I explain the reasons for the decision I made about each item. In the following sections, I take the head features, lexicons, and semantics defined in §6.2 as the basic definitions and change them to cover the phenomena.

8.1 Noun Phrases

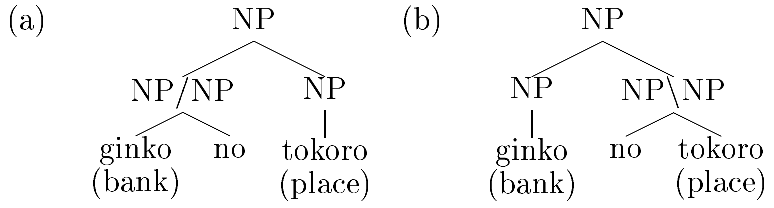
There are two possible categories for 'no' of 'A no B':

a) no := (NP/NP)\NP

b) no := (NP\NP)/NP

Both a) and b) can derive 'ginko no tokoro'(*place at the bank*) as a noun phrase, but the parsing trees are different as below. The tree (25a) is derived with a), and (25b) is derived with b).

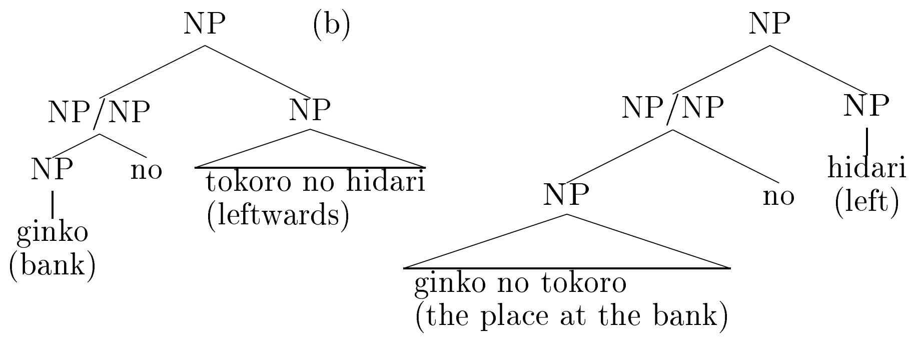
(25)



Let us recall the relation between 'A', 'B', and 'no' discussed in §7.3.1. When 'B' is a location or direction, the phrase 'A no' plays a similar role to adjectives and restricts the meaning of 'B'. Thus 'A no B' should be divided into two parts 'A no' and 'B'. Hence, the tree (25a) reflects the relation between 'A no' and 'B' correctly while (25b) does not.

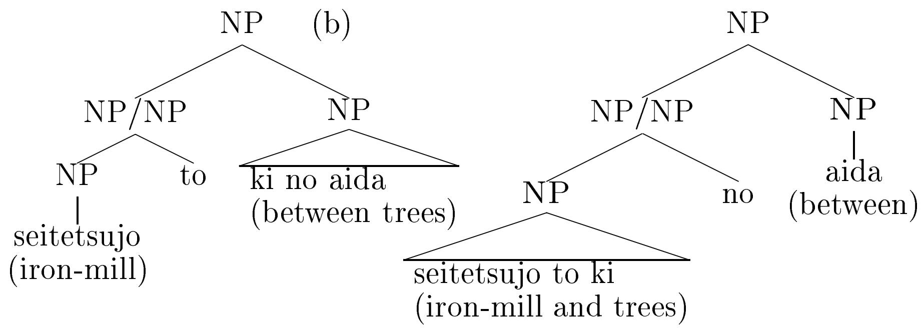
Although the category a) works in the above example, when a noun phrase has more than one 'no' such as 'ginko no tokoro no hidari', the category will derive the phrase as a noun phrase in two different ways as shown in (26). Applying the same analysis to this case, the phrase should be divided into the two, 'ginnko no tokoro ' and 'hidari'. Thus only (26b) reflects the relation.

(26) (a)



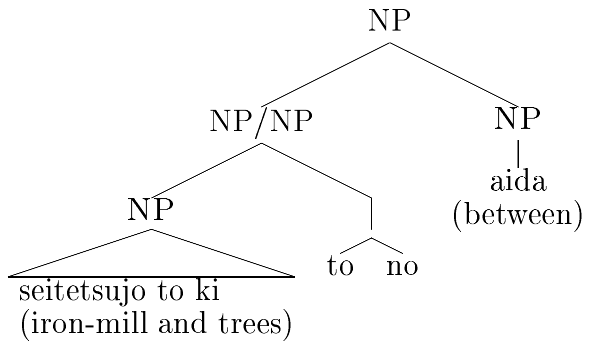
When 'B' is the word 'aida', 'A' usually takes the form of 'A1 to A2' and the same situation happens as shown in (27).(The category for 'to' is defined in the same way as 'no') This case also has to keep the relation 'A no' and 'B', and only (27b) reflects the relation correctly.

(27) (a)



When the adnominal particle 'no' has an additional particle 'to' preceding to it, the combined particle 'to no' plays the same role as 'no' as explained in §7.3.1. Thus, the tree for 'seitetsujo to ki to no aida' should be the one as below.

(28)



The following is a possible encoding for the adnominal particles, 'no' and 'to'. The feature aform is the head feature for noun phrases which records the forms of adnominal particles. The CUF sort goal, apform(˜no), prevents the noun phrase on the right of 'no' from composing the form 'A no B'. Therefore, we can avoid generating the tree(26a) and tree(27a).

The CUF sort ap_semantics/1 generates the semantic representations for the noun phrase 'A no/to B'. In the program, ap means an adnominal particle.

(29)
```
l(no)       :=  ap(no)& ap_semantics(no) & apform(no)&
l(to)       :=  ap(to)& ap_semantics(to) & apform(to);
                root:syn:p2&leaves:[].

%% Category: (NP/NP)\NP
ap(no)      :=  root_syn(n)&
                leaves([np_goal, np_r_goal&apform(~no)]).

%Category: ((NP/NP)\NP)\AP
ap(no)      :=  root_syn(n)&
                leaves([p2_goal, np_goal, np_r_goal&apform(~no)]).

%% Category: (NP/NP)\NP
ap(to)      :=  root_syn(n)&
                leaves([np_goal, np_r_goal&apform(~no)]).
```

```
np_goal     := root:syn:n & dir:left.
np_r_goal   := root:syn:n & dir:right.
apform(X)   := root:syn:aform:X.
p2_goal     := root:syn:p2&dir:left.
```

8.2 Agreement

To deal with the phenomena described in §7.3.2, each noun should have the information on animateness as a feature, but there is a question whether the information is semantic or syntactic. From a philosophical point of view, it would be strange if animateness was treated as part of syntactic information. In the following sections, I discuss the encoding in each approach.

8.2.1 Syntactic approach

The feature `animate` can be added to the head feature structures for a noun phrase in the following way. `animate(+)` shows that the noun is animate and `animate(-)` shows the noun is NOT animate.

(30)
```
n ::
   animate: boolean.

boolean = {+, -}.
```

Then, we need to constrain the element, its subject, of the feature `leaves` of 'aisiteru':

(31)
```
l(aisiteru):= vt &
              leaves([accusative, nominative&animate(+)]).
animate(A):= root:syn:animate:A.
```

However, this is not enough because verbs take post positional phrases as their complements but the head features of nouns are not inherited to post positional phrases. Thus, we need to add the feature `animate` in the head of post positional phrase and also to change the definition of post positional phrase in order to store the information about animateness as below:

(32)
```
p::
   gr:gr_info,
   pform:pf_info,
   animate: boolean.

pp := root:syn:p&
      animate(A)&
      leaves([nun&animate(A)&dir:left]).
```

8.2.2 Semantic approach

The semantic structure can be changed in the following way to have the information about animateness:

(33)
```
np_sem::
      index: afs,
      animate: boolean.
```

The semantics for a post positional phrase is the same as its complement, i.e. the noun phrase, so that the definition of post positional phrases, pp defined in §6.2, does not need to be changed. The subject for 'aisiteru' can be constrained in the same way as in the syntactic approach but the definition of CUF sort `animate` is a little different. It refers to the semantic feature, not to the syntactic feature as below:

(34)
```
l(aisiteru):= vt &
                leaves([accusative, nominative&animate(+)]).
animate(A):= root:sem:animate:A.
```

8.3 Relative clauses

Since Japanese does not have relative pronouns as explained in §7.3.3, relative clauses cannot be expressed in the same way as described in §4.5.4. There are at least two ways to encode Japanese relative clauses; One of them is to treat relative clauses as adjuncts, similar to adjectives described in §4.5.3 (we will call this the adjunct approach). In the adjunct approach, verb adnominal forms are defined as the category which takes fewer arguments than the originals on the left and the head noun on the right. Thus, this approach does not use any traces for relative clauses. For example, the adnominal form of 'aisiteru'(*'love'*) can be defined as the category (NP/NP)\PP (Figure 12). The problem in this approach is that we have to enumerate semantic representation for every case; if there is not the nominative argument in the relative clause, the head noun is the subject. if there is not the accusative argument, the head noun is the object, so on. A problem of this approach is that every verb will have additional lexical entries, which lead to parsing ambiguities.

$$\dfrac{\dfrac{\dfrac{\textit{‘Kenga’}}{PP}\quad\dfrac{\textit{‘aisiteru’}}{(NP/NP)\backslash PP}}{NP/NP}<\quad\dfrac{\textit{‘Naomi’}}{NP}}{NP}>$$

Figure 12: The Adjunct Approach

The other way (we call this the trace approach) is to define noun phrases as a category which takes a relative clause, $\bar{\text{S}}$, and results in a NP as shown in Figure 13, where ϵ shows the trace.

One of the problems of the trace approach is to do with the semantics of nouns. In Montague grammar, a noun is a basic category which does not take any complements. Another problem is that if every noun has the two

$$\dfrac{\dfrac{\textit{‘Ken ga } \epsilon \textit{ aisiteru’}}{\bar{\mathrm{S}}} \quad \dfrac{\textit{‘Naomi’}}{NP\backslash\bar{\mathrm{S}}}}{NP} <$$

Figure 13: The Trace Approach

categories 1) NP and 2) NP/$\bar{\mathrm{S}}$, parsing ambiguity increases more than in the first approach because nouns appear most frequently in sentences.

8.3.1 The Adjunct Approach

The adnominal form of a transitive verb, `vt_adnominal`, can be defined in CUF using `join_trees/2` and `project/2` in the following way, where the symbol ; is a disjunct operator:

(35)
```
l(aisiteru) := vt ; vt_adnominal.
vt_adnominal:= join_trees(adj, project(vt, [_])).
```

The CUF sort `vt` uses the same definition as in example(15) in §6.2, and `adj` can be defined in the same way as in example(9) in §4.5.3. As the result, the CUF sort `vt_adnominal` denotes the following feature structures, where PP denotes the same feature structures by the CUF sort `pp` and NP by `noun` in §6.2:

(36) $\begin{bmatrix} \text{root:syn:} & \text{n} \\ \text{leaves:} & \left\langle \text{PP, NP} \right\rangle \end{bmatrix}$

The CUF sorts, `project/2` and `join_trees/2`, are library functions defined in the following way.

project/2 The CUF sort `project(Tree, CutOffLeaves)` cuts off the leaves `CutOffLeaves` from the arguments of`Tree`(We call this operation `projection` of a category and the resulting category projected form or projected category).

```
     project((root:R&leaves:append(L1, L2)), L1):=
             (root:R&leaves:L2).
```

join_tree/2 The CUF sort `join_trees(Tree1, Tree2)` replaces the lexical head of `Tree1` by `Tree2`.

```
     join_trees((root:R1&leaves:L1), (root:R2&leaves:L2)):=
             (root:R1&leaves:append(L2, L1)).
```

8.3.2 The Trace approach

As I described above, a noun has two categories with this approach: (37b) takes no argument; (37c) takes a sentence which includes a trace, `s_slash`. These are defined in CUF as follows:

(37) a. `l(naomi):= noun.`
 b. `noun:= root:syn:n&leaves([]).`
 c. `noun:= root:syn:n&leaves([s_slash&dir:left]).`

`s_slash:= root:syn:v & leaves:[] & slash:[pp].`

8.4 Flexibility in Word Order and Arguments

Japanese is flexible in its word order and use of arguments as explained in §7.3.4. The following examples (38a) and (38b) have the same meaning, and sentences which lack complements such as (38c) are often observed.

(38) a. Ken-ga Naomi-wo aisiteru

Ken-ga	Naomi-wo	aisiteru
NOM	ACC	love

"Ken loves Naomi"

b. Naomi-wo Ken-ga aisiteru

`NOM`	`ACC`	love

"Ken loves Naomi"

c. Naomi-wo aisiteru

`ACC`	love

"loves Naomi"

There are two choices to deal with the flexibilities:

- To define all arguments of a verb as having adverbial categories, being functions from sentences to sentences as described in [Whi88].
- To define a verb as having categories in addition to the basic ones, and also the argument list for a verb as a set of arguments rather than a list in a similar way to JPSG.

In the following sections, I explain each approach and its possible encodings.

8.4.1 Using Adverbial Category

In this approach, there is no distinction between VP and S, and post positional phrases are defined as adjuncts similar to adverbs. Thus, a post positional particle is defined as a category which takes a noun phrase on the left and a sentence on the right, resulting in a sentence. For example, given the above category for post positional particles 'ni' and 'ga' as shown in (39), both corpus (s40) and (s40') in §7.3.4 can be recognized as S successfully, as shown in Figure 14 and 15.

(39) a. 'ni':= (S/S)\NP

b. 'ga':= (S/S)\NP

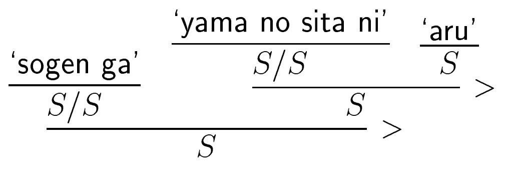


Figure 14: A derivation of 'sogen ga yama no sita ni aru'

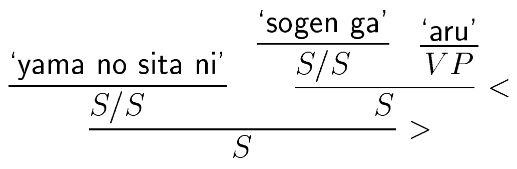


Figure 15: A derivation 'yama no sita ni sogen ga aru'

In the adjunction approach, the lexicon for 'ga', 'wo', and 'aisiteru', can be defined as shown in example(40). In this definition, verb does not have any complements so that a post positional phrase can be defined without copying the arguments of its complement as presented in §4.5.3.

(40)
```
l(ga):= pp(ga, sbj).
l(wo):= pp(wo, obj).
l(aisiteru):= verb.

pp(Pform, GR) :=
   root:syn:(vp & pform:Pform & gr:GR) &
   leaves([noun&dir:left, verb&dir:right]).

verb:= root:syn:vp & leaves([]).
```

Although example(40) can deal with flexibility both in word order and arguments, it is too general to prevent parsing the following ungrammatical sentences:

(41) a. *Ken-ga Naomi-ga aisiteru

NOM	NOM	love

b. *Ken-ga Naomi-wo Naomi-ga aisiteru

NOM	ACC	NOM	love

Example(a) has two complements marked by the same case marker, 'ga' and example(b) has an extra complement, 'Naomi ga'.

We also have to encode the subject-verb agreement shown in §8.2 in this approach. To do that, verbs should have the constraints expressed as their head features. And the post positional phrases should be checked whether the argument verb has a complement marked by the same case of the particle.

In addition, we have to get the same semantic representation such as example(24) in §6.3 both from (38a) and its canonical form(38b). Thus, we have to define semantics for all sentences with different word orders, although the syntax definition can be general.

8.4.2 Changing Subcategorization

The subcategorization approach of LexGram does not make it easy to implement the flexibility in word order and arguments because the order and the number of elements of the feature `leaves` are fixed.

Let us consider the word order first, taking the verb 'aru' (exist) in the corpus (s40) in §7.3.4 as its example. The basic category for 'aru' can be defined as follows:

(42) 'aru' := (S\PP)\PP

In this way, 'aru' is specified to take two post positional phrases from its left to become S, but the definition does not say which PP is subject or locative. Supposing that cases are checked in the semantics, then sentences (s40) and (s40') are recognized as S (Figure 16 and 17).

$$\dfrac{\dfrac{\text{`sogen ga'}}{PP} \qquad \dfrac{\dfrac{\text{`yama no sita ni'}}{PP} \quad \dfrac{\text{`aru'}}{(S\backslash PP)\backslash PP}}{S\backslash PP}<}{S}<$$

Figure 16: A derivation of 'sogen ga yama no sita ni aru'

$$\dfrac{\dfrac{\text{`yama no sita ni'}}{PP} \qquad \dfrac{\dfrac{\text{`sogen ga'}}{PP} \quad \dfrac{\text{`aru'}}{(S\backslash PP)\backslash PP}}{S\backslash PP}<}{S}<$$

Figure 17: A derivation of 'yama no sita ni sogen ga aru'

To deal with optional arguments shown in (s14), (s5) and (s5') in §7.3.4, 'magaru' needs projected categories in addition to the original one.

'magaru' (base form) := (S\PP\PP)\PP

'magaru' (projected form) := S\PP\PP

'magaru' (projected form) := S\PP

With these categories, the sentences (s14) and (s5') are recognized as S as shown in Figure(18) and (19).

With this approach, the definition for verb 'aisiteru' in example(15) in §6.2 cannot deal with the canonical form specified in example(38b). The definition(43) shown below, stopping checking the complements' cases, recognises both

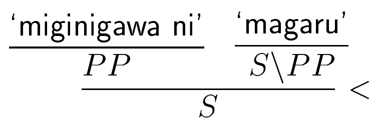


Figure 18: A derivation of 'migigawa ni magaru'

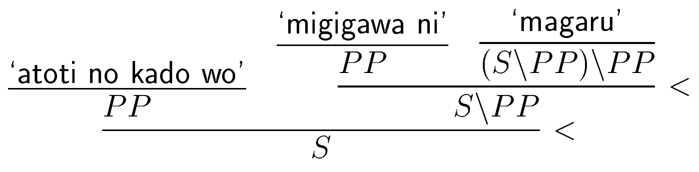


Figure 19: A derivation of 'atoti no kado wo migigawa ni magaru'

(38a) and (38b) as sentences. However, we become unable to reject the ungrammatical sentence(41a).

(43) `l(aisiteru):= vt.`

As mentioned in §8.4.1, we must get the same semantic representation from every canonical sentence. That is, we need to know which argument is nominative and accusative for constructing the semantic representation anyway. Example(43) can be revised as in (44) which checks cases while generating semantic representations. CUF sort `per/1` permutates a list and results in the permutated list, so that the same semantic representation such as example(24) in §6.3 can be obtained both from (38a) and (38b).

(44)
```
l(aisiteru) : =
      vt &
      vt_type('LOVE')&
      subj:S & obj:O & forms(F)&
      root_sem(vars(append(V1, V2))&
               forms(append(F1, append(F2, F)))&
      leaves(per([accusative&root_sem(vstore:O&vars(V)&forms(F1)),
                   nominative&root_sem(vstore:S&vars(V)&forms(F2))])).
```

To deal with flexibility in arguments, we have to enumerate every case. Thus, 'aisiteru' will have additional categories which have only one argument while its basic category has two as seen in (44). Using CUF sort project/2, 'aisiteru' can be defined in the following way, where `vt_p1_semantics` checks the case of an argument and calculates semantic representation. When an argument lacks, the semantic representation for a verb will have an unbounded variable as its argument.

(45)
```
l(aisiteru):= vt&vt_semantics;
              project(vt, [_])& vt_p1_semantics.
```

8.5 The Implemented Grammar

In the above sections, I have shown the alternatives to encode the phenomena discussed in §7. I have chosen the following approaches for each case:

- The category for the noun phrases which have the form 'A no B'

  The category for adnominal particle 'no' is defined as (NP/NP)\NP which reflects the relation between 'A no' and 'B'. Also, 'no' is defined not to have a noun phrase of the form 'A no B' on its right. Another adnominal particle 'to' is defined in a similar way.

- Relative Clauses: the adjunct approach

  I chose the adjunct approach because relative clauses in Japanese play a similar role to adjectives, and it is common to treat adjectives as adjuncts in Categorial Grammar as described in §4.5.3. Another reason is that the trace approach could suffer more parsing ambiguities as I mentioned in §8.3.

- Agreement: the semantic approach

  Since the semantic approach does not need to change the structure of the grammar, I chose this approach.

- Flexibility in Word Order and Arguments: the subcategorization approach

  I decided to choose the subcategorization approach since I cannot be convinced that a grammar which gets the correct semantics for every corpus can be written in the adjunct approach. Although the subcategorization approach seems to be not flexible enough, it is more solid than the adjunct approach and is easy to describe constraints between verbs and complements.

In the data, every verb has a maximum number of complements, and there is no more than one post positional particle in the same sentence which carries the same case. The classification of verbs and particles in the corpus is shown in Appendix C and D.

The implemented parser has 96 lexical items, and is constructed from the following ten files. The numbers in the table shows the size (in bytes) of each file.

file Name	size	contents
gen.cuf	937	general CUF tools
syn.cuf	1556	head feature def.
		type signature for `nonterminal`
cat.cuf	2004	syntactic category definition
sem.cuf	1272	semantic feature def.
		type signature for `sen`
lexsem.cuf	1011	lexical semantics
phrsem.cuf	1009	semantics def except verb semantics
verbsem.cuf	7498	verb semantics
interface.cuf	3237	interface between lexicon and semantics def
lex.cuf	3384	lexicon
psent.cuf	5790	100 test data in LexGram

Table 1: Table of files

9 Evaluation

The purpose of this evaluation is to check the coverage of the grammar described in §8 and to check the efficiency of the parser. The LexGram parser uses a Head-driven chart parsing strategy [vN96], which is particularly useful for lexicalist grammar formalisms such as HPSG and Categorial Grammar. The Head-driven chart parsing is bi-directional in the sense that it adopts both bottom-up and top-down strategies, and also head-driven in the sense that it does not parse a phrase from left to right but starts by locating a potential head of the phrase and then proceeds to parsing its daughters.

To check the coverage of the grammar, the selected data shown in AppendixI were parsed using the program `check/0` (Appendix G). The average run time for parsing one sentence was calculated using programs `timer/0` and `av_time/0` (Appendix G). The programs were run ten times and the average was calculated by hand. Of the one hundred sentences, there was one sentence which could not be parsed with the grammar and two sentences which got two different semantic representations. In this section, I explain the problematic cases and about the run time.

9.1 Problematic Cases

Sentence (s85) could not be parsed with the grammar since the subject of the sentence is animate, but the verb requires an inanimate subject. Because the speaker and the hearer made the conversation while watching maps, both knew that 'gorira' denoted a picture of a gorilla in the map. This is the reason why the verb 'iru' was used. Since this case is outside the scope of what we intend to cover, we can ignore this.[14]

[14]In the run time test, I changed 'gorira' to be inanimate

(s85)

(s85)	kyanpu	to	taisho	no	iti	ni	gorira	ga	aru
	camp	TO	*opposite*	NO	*side*	LOC	*gorilla*	NOM	*there-is*

"There is a gorilla on the opposite side of the camp"

The following sentences (s3) and (s6) are assigned two different semantic representations.

(s3)	atoti	to	iu	tokoro	no	hidarigawa	wo	tooru
	ruin	DAT	*called*	*place*	NO	*left*	ACC	*pass-by*

"Pass by the left of the place called ruin"

(s62)	haka	no	e	ga	owatta	atari	wo	magaru
	graveyard	NO	*picture*	NOM	*end*	*somewhere*	ACC	*turn*

"Turn at around the end of the picture of graveyard"

The noun phrases, 'atoti to iu tokoro no hidarigawa' in(s3) and 'haka no e ga owatta atari' in (s62), can be recognized as noun phrases from the two different derivations shown in Figure 20, 24, 22, and 23. The existence of adnominal particles, 'no', and adnominal verbs, 'iu' and 'owatta' in the phrases causes those different derivations, as the figures show.

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{'atoti to iu'}}{NP\backslash NP} \quad \dfrac{\text{'tokoro'}}{NP}}{NP}> \quad \dfrac{\text{'no'}}{(NP/NP)\backslash NP}}{NP/NP}< \quad \dfrac{\text{'hidarigawa'}}{NP}}{NP}>$$

Figure 20: 'atoti to iu tokoro no hidarigawa'

Let us consider the noun phrase 'atoti to iu tokoro no hidarigawa' in (s3) first. The meaning of (('atoti to iu tokoro' 'no') ('hidarigawa')) in Figure 20 is that *'there is a place called ruin, and it is the left of that place'*. On the

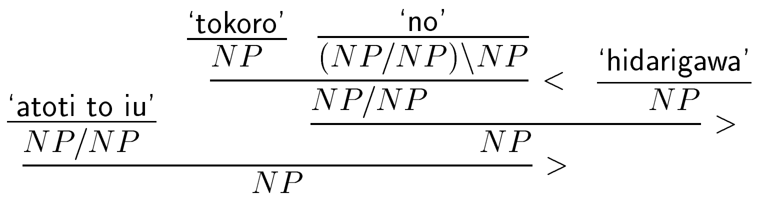


Figure 21: 'atoti to iu tokoro no hidarigawa'*

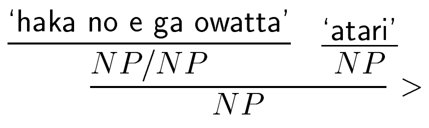


Figure 22: 'haka no e ga owatta atari'2

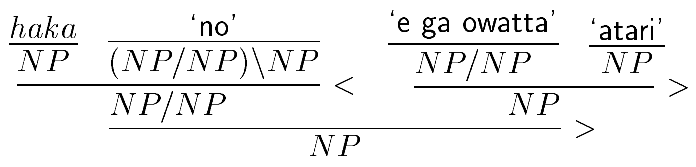


Figure 23: 'haka no e ga owatta atari'*

other hand, the second ((‘atoti to iu’) (‘tokoro no hidarigawa’)) means nothing. Since *‘tokoro’*(*place*), call it X, is an abstract noun and does not specify any particular place, we cannot get a meaning for *‘the left of X’*.

In the case of (s62), the meaning of ((‘haka no e ga owatta’) (‘atari’)) in Figure 22 is that *‘the place where the picture of a graveyard ends’*, while the meaning of the second ((‘haka no’)(‘e ga owatta atari’)) in Figure 23 is that *‘the place where a picture ends and a graveyard exists’*. Since there are a lot of items in a map, it is ambiguous which place the second interpretation specifies. It is also clear that the first reading is the speaker’s intention from the situation where the speaker was looking at a picture of a graveyard.

However, in the following cases, the above false derivation can be correct. (s3’) has the same pattern as ‘atoti to iu tokoro no hidarigawa’ in (s3), and (s62’) as ‘haka no e ga owatta atari’. The correct meanings of the strings are obtained from the derivations in Figure(24) and Figure(25).

(s3’) Naomi to iu Ken no tomodati
Naomi DAT *called Ken* NO *friend*
“Ken’s friend who is called Naomi”

(s62’) Ken no futi ga kaketa sara
Ken NO *edge* NOM *broken dish*
“Ken’s dish whose edge is broken”

Thus, we must derive the strings in two different ways as the parser did, but it will have to be able to select the correct one.

9.2 Run Time Test

The program `timer/0` in Appendix G is used to record the runtime for parsing each sentence. The test was done ten times and the average run time for

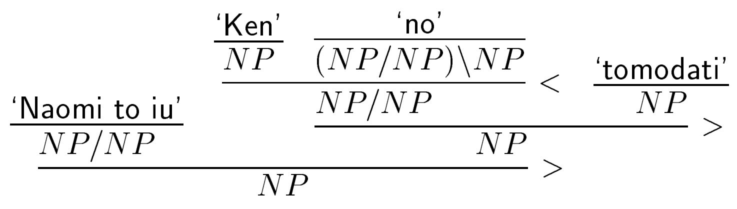


Figure 24: 'Naomi to iu Ken no tomodati'

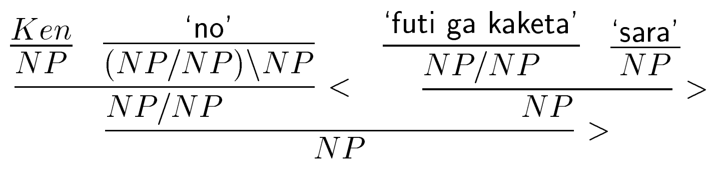


Figure 25: 'Ken no futi ga kaketa sara'

one sentence was 271.77 msec. Sentence (s16) was the worst in every test, which took more than 700msec. The fastest sentence was different each time, but (s26) was most often faster than the others. One of the ten results is presented in the table (Appendix E), and a histogram is in Appendix F.

In the following sections, I observe the results and explain the reasons for the results.

9.2.1 Observation

The items of the table in Appendix E are sorted by their runtime; the first, (s26), took the least time to be parsed, while the 100th took the most. The first line of the table shows the meaning of each column; `SNo` means sentence number, `L` the length of the sentence, `PP` the number of post positional particles in the sentence, `AP` the number of adnominal particles in the sentence, and `V` the verb type of the verb in the sentence (v2: a verb which takes two arguments, v3: a verb which takes three arguments). The histogram shows the distribution of the times, where one star, *, corresponds to ten msec.

In the following, I list some characteristic features of the table.

- The first five sentences, (s26), (s19), (s52), (s35), and (s22), took less than or equal to 100 msec. All of them have the same length, (five words), two post positional particles, no adnominal particles, and the same verb type, v2.

- The first forty four items, from (s26) to (s65), have the same verb type, v2, except the five v3 type sentences (s30), (s96), (s36), (s40), and (s8) marked with a star, *. Most of them are parsed in between 130 and 160 msec. All the exceptions have no adnominal particles and fewer words than the others; the average word number of the five is 3.5 words while the average of the others is 6.82 words.

- The rest of the items in the table, from the 45th to the 100th, have the same verb type v3, except sentence (s23) marked with a star, *. The sentence (s23) is the longest sentence in the data.

- From the 45th to 85th, all sentences have the same pattern; the length of five words, one postpositional particle, one adnominal particle, and v3 type. They range within 100 msec and most of them took 300 or 310 msec to be parsed. From the 90th to the 97th, they have two adnominal particles and they took 200 msec longer to be parsed than the others.

- The two groups of sentences (s42), (s50), (s99) and (s81), (s4), (s2) have the same pattern; the seven words length, one post positional particle, and one adnominal particle. The difference between the two is in the verb type; the first group is of type v2 and the second is of v3. Although the difference is just in the verb type, the second took twice as long as the first to be parsed; the average run time of the first group is 147 msec, while the second is 396 msec.

- Sentences (s62) and (s3) were the 98th and 99th, respectively, although they have just one adnominal particle. All other sentences after the 90th have more than two adnominal particles. The characteristic point of these is that both have relative clauses, while the others do not.

- The worst sentence is (s16). The difference between this and the others is the number of adnominal particles. (s16) has three adnominal particles while no other sentences have that many adnominal particles. The verb of (16) is also of v3 type.

9.2.2 Analysis

There are two main factors for the run time as the observations show; the difference of verb types, and the number of the adnominal particles. Let us consider the verb type first.

The v2 type verb has two and the v3 verb type has three arguments in their basic categories.

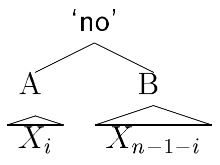
v2 type := (S\PP)\PP (original category)
S\PP (projected category)

v3 type := ((S\PP\)PP)\PP (original category)
(S\PP)\PP ; S\PP (projected category)

In addition to the basic category, each verb type has projected categories in order to allow free gaps in a sentence as discussed in §8.4.2. Also, if a verb has two arguments for its syntactic category, it will have two lexical entries to get the same semantic representation from the different word ordered sentences as discussed in §8.4.2. Thus, if the number of arguments becomes three, the number of the lexical entries becomes six. Considering the projected categories, one v3 type verb has fifteen lexical entries, while v2 has four. Hence, the difference of the verb type affects the run time significantly.

The other factor, adnominal particles such as 'no' and 'to', have the category (NP/NP)\NP. A noun phrase with 'no' has the general form 'A no B' as shown below, where n means the total number of 'no' occurring in the noun phrase 'A no B', X_y means a phrase containing y occurrences of 'no'. $(i \geq 0; i \leq n-1)$

(46)



Using the above notation, the number of possible analyses of a noun phrase with 'no' can be expressed with the following expression.

$$f(X_n) = \sum_{i=0}^{n-1} f(X_i) * f(X_{n-1-i})$$

$$f(X_0) = 1$$

Let us consider the ambiguity of 'no' in detail, taking the worst case (s16) as an example, where the noun phrase 'kyanpu no hidari no koya no hidari' has three 'no's. According to the above expression, it can be divided into three cases as shown below:

f('kyanpu no hidari no koya no hidari') =

f('kyanpu')*f('hidari no koya no hidari') +

f('kyanpu no hidari') * f('koya no hidari') +

f(' kyanpu no hidari no koya') * f('hidari').

Since $f(X_0) = 1$, $f(X_1) = 1(= 1 * 1)$, and $f(X_2) = 2(= 1 + 1)$, the result of the above expression becomes five (=1*2 + 1*1 + 2*1). Thus, the noun phrase has twice as many ambiguities as the one with two 'no's.

10 Conclusion

The parser based on JPSG was implemented in LexGram and was tested against the one hundred sentences collected from the Japanese Map Task Corpus. Although LexGram provides only two built-in rules, the parser succeeded in covering some characteristic phenomena in Japanese: (1)word order variation, (2)gaps in a sentence, and (3)relativization. It parses sentences at a reasonable speed, an average of 271.77 msec. per sentence with an average of 6 words per sentence.

We have also shown that it is easy to write a grammar in LexGram. The core part of the program was implemented in a few weeks. CUF types and sorts turned to be very useful for implementing the parser. There was, however, a problem that debugging was sometimes hard due to lack of good debugging tools.

10.1 Comparison With Related Work

The Adverbial Approach Whitelock [Whi88] proposes to define all arguments of a verb as the functor category S\S, functions from sentences to sentences, and a verb is defined as a sentence S. In his approach, it is easy to cover the word order variation and gaps in a sentence. However, we chose a different approach to cover the same phenomena; To deal with word order variation, the argument list for a verb as a set of arguments rather than as a list. The gaps in a sentence are dealt with by giving verbs categories additional to the basic ones. We also consider semantic representations for sentences, while he did not attempt it due to his approach.

JPSG In the thesis, JPSG [Gun87] is implemented in a Categorial Grammar. Sentences can be parsed only with the forward and backward function application rules. Although JPSG uses traces and foot feature principles to define relative clauses, we have proposed an alternative which does not require traces. However, the implemented grammar requires more lexical entries.

10.2 Future Work

I would like to extend the parser in the following aspects so that it can cover the problems pointed out in §9.1 and also can deal with morphology.

Semantic Representation The semantic representation on the whole has not been studied enough for the data in this thesis. For example, we have not considered semantic representations for adverbs and adjectives, which should be implemented. In addition, as the two problems pointed out in §9.1 suggest, we should check the semantics of noun phrases to eliminate false semantic representations and noun phrases should have more information for the checking. I am also expecting that enriching the semantics for noun phrases allows the parser to cover the sentences where particles are omitted as we observed in §7.2 since the meaning of the particles could be guessed from the meaning of the noun phrases.

Morphology In this thesis, I did not consider morphology. For example, 'aisiteru' is treated as one word. However, the word can actually be seen as composed of 'aisu'(*to love*) and 'iru'(*the present tense*). Honorific information, omitted from the data in §7.2, can also be dealt with when we extend the parser.

References

[ABB$^+$91] Anne H. Anderson, Miles Bader, Ellen G. Bard, Elizabeth H. Boyle, Gwyneth M. Doherty, Simon C. Garrod, Stephen D. Isard, Jacqueline C. Kowtko, Jan M. McAllister, Jim Miller, Catherine F. Sotillo, Henry S. Thompson, and Regina Weinert. The HCRC Map Task Corpus. *Language and Speech*, 34(4):351–366, 1991.

[AIK$^+$94] Motoko Aono, Akira Ichikawa, Hanae Koiso, Shinji Sato, Makiko Naka, Syun Tutiya, Kenji Yagi, Naoya Watanabe, Masato Ishizaki, Michio Okada, Hiroyuki Suzuki, Yukiko Nakano, and Keiko Nonaka. The Japanese Map Task Corpus: an interim report. In *Spoken Language Processing*, volume SLP3(5), pages 25–30. Information Processing Society of Japan, 1994. in Japanese.

[Car92] Bob Carpenter. *The Logic of Typed Feature Structures.* Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, 1992.

[DD93] Jochen Dörre and Michael Dorna. CUF - a formalism for linguistic knowledge representation. Technical report, August 1993. DYANA2 Report R.1.2A.

[GKPS85] Gerald Gazdar, Ewan Klein, G.K. Pullumi, and Ivan Sag. *Generalized Phrase Structure Grammar.* Blackwell, Oxford, UK, 1985.

[Gun87] Takeo Gunji. *Japanese Phrase Structure Grammar.* D.Reidel Publishing Company, Holland, 1987.

[HTH$^+$86] H.Miyoshi, T.Gunji, H.Sirai, K.Hasida, and Y.Harada. A phrase structure grammer for Japanese; JPSG. Technical report, Institute for New Generation Computer Technology(ICOT), Tokyo, Japan, 1986. TR0199.

[KB95] Esther König-Baumer. LexGram - a practical categorial grammar formalism. In *Proceedings of the Workshop on Computational Logic for Natural Language Processing*, Edinburgh, Scotland, April 1995. A Joint COMPULOGNET/ELSNET/EAGLES Workshop.

[Lam61] Joachim Lambek. On the calculus of syntactic types. In R. Jakobson, editor, *Structure of Language and its Mathematical Aspects*, pages 166–178. American Mathematical Society, Providence, Rhode Island, 1961. Proceedings of the Symposium in Applied Mathematics, XII.

[PS94] Carl Pollard and Ivan A. Sag. *Head Driven Phrase Structure Grammar.* The University of Chicago Press and Center for the Study of Language and Information, Chicago, Ill. and Stanford, Ca., 1994.

[Usz86] Hans Uszkoreit. Categorial Unification Grammars. In *Proceedings of the 11th International Conference on Computational Linguistics*, pages 187–194, Bonn, 1986.

[vN96] Gertjan van Noord. An efficient implementation of the Head-corner parser. Manuscript available via http://grid.let.rug.nl/ vannoord/papers/ Netherlands Organization for Scientific Research, Language and Speech Technology - document number 25, 1996.

[Whi88] P. Whitelock. A feature-based categorial morpho-syntax for Japanese. In Uwe Reyle and Christian Rohrer, editors, *(eds.)*

(1988): Natural Language Parsing and Linguistic Theory, pages 230–259. Reidel, 1988.

[Win83] Terry Winograd. *Language as a cognitive process: Syntax.* Addition-Wesley Publishing Company, 1983.

[Zee88] Henk Zeevat. Combining categorial grammar and unification. In Uwe Reyle and Christian Rohrer, editors, *Natural Language Parsing and Linguistic Theories*, pages 202–229. Reidel, 1988.

A Map

the copy of the map will be added

B Lexicon

The number in the table shows the frequncy of each word in the data.

Items: 96words

%%a

agaru	(to go up)	:	1
aida	(space)	:	3
apati	(apachi)	:	1
aru	(to exist)	:	37
atari	(surroundings)	:	3
atoti	(ruin)	:	4

%%b

bokugo	(shelter)	:	1
bokujo	(farm)	:	2
boti	(graveyard)	:	3

%%d

danso	(dislocation)	:	3
denwa	(phone-booth)	:	2

%%e

e	(picture)	:	2

%%f

furo	(bath)	:	3

%%g

ga	(NOM)	:	39
gake	(cliff)	:	4
gareki	(ruin)	:	1
ginko	(bank)	:	9
gorira	(gorira)	:	1

%%h

haikyo	(ruin)	:	6
haji	(side)	:	2
haka	(graveyard)	:	1
hasira	(pole)	:	1
hatake	(farm)	:	1
hayasi	(forest)	:	1
hidari	(left)	:	9
hidarigawa	(left)	:	13
hidarisita	(bottom left)	:	4
hidariue	(top left)	:	6
hou	(side/way)	:	12

%%i

iku	(to go)	:	18
iru	(to exist)	:	3
iti	(position)	:	1
iu	(called)	:	1
iwa	(rock)	:	1

%%k

kado	(corner)	:	1
kaigansen	(coast)	:	1
kamosika	(mountain-goat)	:	1
kanu	(canu)	:	1
kawa	(river)	:	7
ki	(tree)	:	3
kinenhi	(monument)	:	1
kojo	(factory)	:	2
kokuyurin	(forest)	:	1
koya	(cottage)	:	2
kozan	(mine)	:	2
kuru	(to come)	:	5
kyanpu	(camp)	:	9
kyoru	(dinosaur)	:	1

%%m

made	(LOC)	:	11
magaru	(to turn to)	:	5
makiba	(farm)	:	2

mannaka	(middle)	: 4
mawaru	(to turn)	: 2
mieru	(can see)	: 2
mieru	(see)	: 2
migi	(right)	: 4
migigawa	(right)	: 8
migisita	(bottom right)	: 3
migite	(right)	: 2
migiue	(top right)	: 2
minka	(house)	: 2
mitisirube	(guideboad)	: 1
mizutamari	(pond)	: 2
mo	(ACC)	: 1
mon	(gate)	: 1
%%n		
ni	(LOC)	: 58
no	(NO)	: 100
numa	(bog)	: 2
%%o		
owatta	(ended)	: 1
%%s		
sabaku	(desert)	: 4
sakeru	(to avoid)	: 1
seitetsujo	(iron-mill)	: 4
sinkouhoukou	(heading direction)	: 2
sita	(botoom)	: 21
sogen	(plain)	: 4
sotogawa	(outside)	: 1
sunahama	(beach)	: 2
susumu	(to go)	: 2
syupatsutiten	(starting point)	: 9
%%t		
taisho	(opposite)	: 1
tenbodai	(observatory)	: 1
tizu	(map)	: 2
to	(TO, DAT)	: 9
tokoro	(place)	: 8
tooru	(to pass by)	: 22
tsuirakugenba	(clash spot)	: 2
tsukiatari	(end)	: 1
tsuribasi	(bridge)	: 1
%%u		
ue	(upside)	: 10
usi	(cow)	: 1
uti	(house)	: 1
%%w		
wa	(ACC)	: 1
wan	(bay)	: 2
wo	(ACC)	: 34
%%y		
yama	(mountain)	: 5
yoko	(next)	: 4

C Particles

The post positional particles in the corpus carry the following cases. Since I choose the subcategorization approach, described in §8.5, the categories for all post positional particles are the same as shown below.

C.1 Post Positional Particle

Particle	Case	Category
ga	NOM	PP\NP
mo	ACC	PP\NP
wa	ACC	PP\NP
wo	ACC	PP\NP
to	DAT	PP\NP
ni	LOC	PP\NP
made	LOC	PP\NP

C.2 Adnominal Particle

Particle	Category
no	(NP/NP)\NP
no	((NP/NP)\NP)\P2
to	(NP/NP)\NP
to	P2

D Verbs

D.1 Verb Types

Verbs(sentential final form) in the corpus are classified into the following four types. Each type has its basic category and also projected categories.

`v3_ann` is the verb type which takes three arguments(nominative, accusative, and locative); its nominative argument is animate, its accusative argument is not animate, and its locative argument is not animate.

`v3_aoo` is the verb type which takes three arguments(nominative, accusative, and locative); and its nominative argumeent is animate, the others can be anything.

`copl_an` is the verb type which is copular and takes two arguments; its nominative argument is animate, its locative argument is not animate.

`copl_nn` is the verb type which is copular and takes two arguments; its nominative argument is not animate, locative is not animate.

D.2 Verbs

Verb Type	Verbs
v3_ann	agaru, iku, kuru, magaru, mawaru, sakeru, tooru, susumu
v3_aoo	mieru, miru
copl_an	iru
copl_nn	aru

Verb Type	Category	Explanation
v3_ann	$((S\backslash PP_x)\backslash PP_y)\backslash PP_z$ $(S\backslash PP_x)\backslash PP_y$ $S\backslash PP_x$	x, y, z are either nominative & animate(+), accusative & animate(-), or locative & animate(-).
v3_aoo	$((S\backslash PP_x)\backslash PP_y)\backslash PP_z$ $(S\backslash PP_x)\backslash PP_y$ $S\backslash PP_x$	x, y, z are either nominative & animate(+), accusative & animate(Any), or locative & animate(Any).
copl_an	$(S\backslash PP_x)\backslash PP_y$ $S\backslash PP_x$	x, y are either nominative & animate(+),or locative & animate(-).
copl_nn	$(S\backslash PP_x)\backslash PP_y$ $S\backslash PP_x$	x, y are either nominative & animate(-),or locative & animate(-).

E Evaluation

The average runtime: 270.6 (msc.)/ sentence

No	SNo	Time	L	PP	AP	V
1	(s26)	80	5	2	0	v2
2	(s19)	80	5	2	0	v2
3	(s52)	90	5	2	0	v2
4	(s35)	100	5	2	0	v2
5	(s22)	100	5	2	0	v2
6	(s36)	120	7	2	1	v2
7	(s25)	120	5	1	1	v2
8	(s59)	130	6	2	0	v2
9	(s42)	130	7	2	1	v2
10	(s37)	130	7	2	1	v2
11	(s32)	130	7	2	1	v2
12	(s31)	130	7	2	1	v2
13	(s76)	140	7	2	1	v2
14	(s45)	140	7	2	1	v2
15	(s43)	140	7	2	1	v2
16	(s33)	140	7	2	1	v2
17	(s15)	140	7	2	1	v2
18	(s11)	140	7	2	1	v2
19	(s9)	140	7	2	1	v2
20	(s84)	150	7	2	1	v2
21	(s83)	150	7	2	1	v2
22	(s57)	150	6	1	1	v2
23	(s50)	150	7	2	1	v2
24	(s44)	150	7	2	1	v2
∗25	(s30)	150	3	1	0	v3
26	(s18)	150	7	2	1	v2
27	(s6)	150	7	2	1	v2
28	(s99)	160	7	2	1	v2
29	(s98)	160	7	2	1	v2
30	(s97)	160	7	2	1	v2
∗31	(s96)	160	3	1	0	v3
32	(s47)	160	7	2	1	v2
33	(s17)	160	7	2	1	v2
34	(s41)	170	7	2	1	v2
35	(s40)	170	7	2	1	v2
∗36	(s14)	170	3	1	0	v3
37	(s1)	180	7	2	1	v2
38	(s85)	200	8	2	2	v2
39	(s38)	210	7	1	2	v2
∗40	(s100)	240	5	2	0	v3
41	(s58)	250	9	2	2	v2
42	(s51)	250	9	2	2	v2
∗43	(s8)	250	5	2	0	v3
44	(s65)	270	9	2	2	v2
45	(s49)	270	5	1	1	v3
46	(s27)	280	5	1	1	v3
47	(s54)	290	5	1	1	v3
48	(s46)	290	5	1	1	v3
49	(s20)	290	5	1	1	v3
50	(s92)	300	5	1	1	v3
51	(s87)	300	5	1	1	v3
52	(s79)	300	5	1	1	v3
53	(s77)	300	5	1	1	v3
54	(s75)	300	5	1	1	v3
55	(s70)	300	5	1	1	v3
56	(s60)	300	5	1	1	v3
57	(s55)	300	5	1	1	v3
58	(s39)	300	5	1	1	v3
59	(s34)	300	5	1	1	v3
60	(s13)	300	5	1	1	v3
61	(s7)	300	5	1	1	v3
62	(s90)	310	5	1	1	v3
63	(s86)	310	5	1	1	v3
64	(s82)	310	5	1	1	v3
65	(s73)	310	5	1	1	v3
66	(s69)	310	5	1	1	v3
67	(s66)	310	5	1	1	v3
68	(s64)	310	5	1	1	v3
69	(s63)	310	5	1	1	v3
70	(s61)	310	5	1	1	v3
71	(s56)	310	5	1	1	v3
72	(s53)	310	5	1	1	v3
73	(s48)	310	5	1	1	v3
74	(s28)	310	5	1	1	v3
75	(s21)	310	5	1	1	v3
76	(s10)	310	5	1	1	v3
77	(s91)	320	5	1	1	v3
78	(s89)	320	5	1	1	v3
79	(s88)	320	5	1	1	v3
80	(s78)	320	5	1	1	v3
81	(s72)	320	5	1	1	v3
82	(s68)	320	5	1	1	v3
83	(s29)	320	5	1	1	v3
84	(s5)	320	5	1	1	v3
85	(s94)	360	5	1	1	v3
∗86	(s23)	360	10	2	2	v2
87	(s81)	380	7	2	1	v3
88	(s4)	400	7	2	1	v3
89	(s2)	410	7	2	1	v3
90	(s80)	490	7	1	2	v3
91	(s67)	490	7	1	2	v3
92	(s95)	500	7	1	2	v3
93	(s74)	500	7	1	2	v3
94	(s71)	500	7	1	2	v3
95	(s93)	510	7	1	2	v3
96	(s24)	510	7	1	2	v3
97	(s12)	580	8	1	2	v3
98	(s62)	610	8	2	1	v3
99	(s3)	620	8	2	1	v3
100	(s16)	770	9	1	3	v3

F Histogram

```
 (s26)  ********
 (s19)  ********
 (s52)  *********
 (s35)  **********
 (s22)  **********
 (s36)  ************
 (s25)  ************
 (s59)  *************
 (s42)  *************
 (s37)  *************
 (s32)  *************
 (s31)  *************
 (s76)  **************
 (s45)  **************
 (s43)  **************
 (s33)  **************
 (s15)  **************
 (s11)  **************
  (s9)  **************
 (s84)  ***************
 (s83)  ***************
 (s57)  ***************
 (s50)  ***************
 (s44)  ***************
 (s30)  ***************
 (s18)  ***************
  (s6)  ***************
 (s99)  ****************
 (s98)  ****************
 (s97)  ****************
 (s96)  ****************
 (s47)  ****************
 (s17)  ****************
 (s41)  *****************
 (s40)  *****************
 (s14)  *****************
  (s1)  ******************
 (s85)  ********************
 (s38)  *********************
(s100)  ************************
 (s58)  *************************
 (s51)  *************************
  (s8)  *************************
 (s65)  ***************************
 (s49)  ***************************
 (s27)  ****************************
 (s54)  *****************************
 (s46)  *****************************
 (s20)  *****************************
 (s92)  ******************************
 (s87)  ******************************
 (s79)  ******************************
 (s77)  ******************************
 (s75)  ******************************
 (s70)  ******************************
 (s60)  ******************************
 (s55)  ******************************
 (s39)  ******************************
 (s34)  ******************************
 (s13)  ******************************
  (s7)  ******************************
 (s90)  *******************************
 (s86)  *******************************
 (s82)  *******************************
 (s73)  *******************************
 (s69)  *******************************
 (s66)  *******************************
 (s64)  *******************************
 (s63)  *******************************
 (s61)  *******************************
 (s56)  *******************************
 (s53)  *******************************
 (s48)  *******************************
 (s28)  *******************************
 (s21)  *******************************
 (s10)  *******************************
 (s91)  ********************************
 (s89)  ********************************
 (s88)  ********************************
 (s78)  ********************************
 (s72)  ********************************
 (s68)  ********************************
 (s29)  ********************************
  (s5)  ********************************
 (s94)  ************************************
 (s23)  ************************************
 (s81)  **************************************
  (s4)  ****************************************
  (s2)  *****************************************
 (s80)  *************************************************
 (s67)  *************************************************
 (s95)  **************************************************
 (s74)  **************************************************
 (s71)  **************************************************
 (s93)  ***************************************************
 (s24)  ***************************************************
 (s12)  **********************************************************
 (s62)  *************************************************************
  (s3)  **************************************************************
 (s16)  *****************************************************************************
```

G Test Program

```
/* Test Program for coverate */
check:-
        asserta(count(1)),
        asserta(time(0,0)),
        repeat,
          testno(TestNo),
          p0(TestNo, ID),
          show_item(ID), nl, nl, %LexGram command
          show_tree(ID), nl, nl, %LexGram command
        until(100).

/* Program for recording runtime data */
timer:-
        asserta(count(1)),
        asserta(time(0,0)),
        repeat,
          testno(TestNo),
          time0(TestNo, Time),
          assert0(time(TestNo, Time)),
        until(100).

time0(TestNo, Total):-
   statistics(runtime,[StartTime|_]),
   p(TestNo),
   statistics(runtime,[EndTime|_]),
   Total is EndTime - StartTime,!.

until(Max):-
        count(C),
        Max =< C.
until(Max):-
        retract0(count(C)),
        N is C + 1,
        assert0(count(N)),
        fail.

testno(X):-count(X), !.
retract0(X):-   retract(X), !.
assert0(X):-    assert(X), !.
p0(TestNo, ID):- p(TestNp, _, ID), !. %p/3 is LexGram command

av_time:-
        bagof([No, Time], time(No, Time), L),
        sort_list(L, Start, Sorted),
        cal(Sorted, Total),
        A is Total / 100,
        write('The average : '), write(A), nl.

cal([], Total, Total):- !.
cal([[_|Time]|Tail], Total, Result):-
        Tmp is Total + Time,
        cal(Tail, Tmp, Result), !.
```

H The Code for the Parser

```
%% File: gen.cuf
%% Yoshiko

append([], L&list) := L.
append([H|T], L&list) := [H| append(T, L)].


del(H, [H|T]) := T.
del(H, [T1|T2]) := [T1|del(H, T2)].

mem([H|_T]) := H.
mem([_|T]) := mem(T).

per([]) := [].
per(L&list):= [X|per( del(X, L) )].

%%%

%%
%% join_trees(Tree1,Tree2) := CombinedTree.
%%  - join the leaves of Tree1 and Tree2
%%    (Tree2 is inserted at the head position of Tree1)
%%
join_trees(
     category(Root1,Leaves1),        % upper part of tree
     category(_Root2,Leaves2 ) ) :=  % lower part of tree
  category(                          % combined tree
     Root1,
     append(Leaves2,Leaves1) ).
     % leaves of lower part are closer to the head leaf

%%
%% project(Category,Leaves) := Projection
%%  - Leaves : those Leaves which have been cut off to produce Projection
%%
project(
      category(
         Root,
         append(Leaves1,Leaves2) ),
      Leaves1 ) :=
   category(Root,Leaves2).


%% File: syn.cuf
%% Yoshiko Fujinami

grammar_name := japanese.  % added ek96-06-15

nonterminal = v | n | p | p2.

v ::
  cform : cf_info,
  sub   : sub_info.

p ::
  gr    : gr_info,
  pform : pf_info.

n ::
  aform : af_info.

p2 ::
  p2form : p2_info.

sub_info ::
  no: boolean,
  ac: boolean,
  da: boolean,
  lo: boolean.

gr_info  = { sbj, obj, dat, loc }.
pf_info  = { ga, wo, ni, wa, mo, to, made }.
p2_info  = { to }.
cf_info  = { senf, adnm }.
af_info  = { no, to }.
boolean  = { +, -}.

%============================================
% data access
%

'_gr'(GR)   :=  gr: GR.
'_pform'(PF):=  pform: PF.

apform(X)  := root_syn(  aform: X ).

adnominal  := root_syn(  cform: adnm).
sent_final := root_syn(  cform: senf).

nom(X)     := root_syn(  sub: no:X).
acc(X)     := root_syn(  sub: ac:X).
dat(X)     := root_syn(  sub: da:X).
loc(X)     := root_syn(  sub: lo:X).

nominative := root_syn(  gr: sbj).
accusative := root_syn(  gr: obj).
dative     := root_syn(  gr: dat).
locative   := root_syn(  gr: loc).

pform(P)   := root_syn(  pform: P).
gr(G)      := root_syn(  gr: G).

p2form(P)  := root_syn(  p2form: P).

nonterminal(Syn):=  cons_root(Syn, _Sem).
dir(Dir)        :=  goal(Dir, _n, _su, _sl, _c).
root(R)         :=  category(R, _L).
goal(Dir, N)    :=  goal(Dir, N, [], [], _C).

root_syn(S)       :=  category( cons_root(S, _Sem), _G ).
root_sem(S)       :=  category( cons_root(_Syn, S), _G ).

satured           :=  category( _R, [] ).
leaves(L)         :=  category( _R, L  ).
top_leaves(T)     :=  category( _R, [T|_]).

%% File Name: cat.cuf
%% Yoshiko Fujinami

%============================================
%
```

```
%% categories
%
%% Category: NP
np          :=  root_syn(n) &
                satured.

%% Category:  S\NP
v(i)        :=  root_syn(v) &
                leaves([pp_goal]).

%% Category:  (S\NP)\NP
v(t)        :=  root_syn(v) &
                leaves([pp_goal, pp_goal]).


%% added form syn.cuf
%% Category:  NP/NP

v(irel)      := join_trees(adj, project(v(i), [_])).

%% Category:  (NP/NP)\PP
v(trel)      := join_trees(adj, project(v(t), [_])).

%% Category:  ((S\NP)\NP)\NP
v(d)        :=  root_syn(v) &
                leaves([pp_goal, pp_goal, pp_goal]).

%% Category: NP\NP
pp          :=  root_syn(p)&
                leaves([np_goal]).

%% Category: (NP/NP)\NP
ap(no)      :=  root_syn(n)&
                leaves([np_goal, np_r_goal&apform(~no)]).

ap(to)      :=  root_syn(n)&
                leaves([np_goal, np_r_goal&apform(~no)]).

%% Category: ((NP/NP)\NP\AP
ap(no)         :=  root_syn(n)&
                leaves([p2_goal, np_goal, np_r_goal&apform(~no)]).

%% Category: P
part        :=  root_syn(p2)&
                satured.

%% Category: NP\NP
adj       := category(
                  nonterminal(n),
                  [ np &
                    goal(right,
                      nonterminal(n),
                      Leaves,
                      _Slash,
                      _C )
                  | Leaves ]).

%% Category: VP\VP
adv       := category(
                  nonterminal(v),
                  [ verb_category &
                    goal(right,
                     nonterminal(v),
                     Leaves,
                     _Slash,
                     _C )
                 | Leaves ]).

verb_category:= v(i);v(t).

adv2     := category(
                 nonterminal(n),
                 [ v(trel) &
                   goal(right,
                     nonterminal(n),
                     Leaves,
                     _Slash,
                     _C )
                 | Leaves ]).

%============================================
% goals
%

pp_goal     := goal(left, nonterminal(p)).

p2_goal     := goal(left, nonterminal(p2)).

vp_goal     := goal( left, nonterminal(v)).

np_goal     := goal( left, nonterminal(n)).

np_r_goal   := goal( right, nonterminal(n)).


%% FileName: sem.cuf
%% Yoshiko Fujinami


sem ::
   formula : quantified_formula.

sem = verbal_sem | np_sem.

  verbal_sem ::
    subj: afs,   %nominative
    obj : afs,   %accusative
    obj2: afs.   %dative; or locative

  np_sem ::
    index: afs,
    vstore:afs,
    semf:semf.


  semf ::   % semantic information for subcat
    animate: boolean,
    attribute: att_info.

  boolean = {+, -}.
  att_info = { loc, dir }.

quantified_formula::
    vars: list,
    forms: list.

formula= basic_relation.

basic_relation ::
  label: afs,
  rel : afs,
  args: argument_frame.

argument_frame ::
  arg1 : top,
  arg2 : top,
  arg3 : top.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% data access
%

semrepr(Formula) :=
  formula : forms: [Formula].

rel(RelName, Args) :=
  rel : RelName &
  args: Args.

% root_sem is defined in syn.cuf
% root_sem(S)       :=  category( cons_root(_Syn, S), _G ).

sem(Sem) := cons_root(_Syn, Sem).
forms(F):= formula: forms: F.
vars(V) := formula: vars: V.

i(I)      := root_sem(vstore:I).
f(F)      := root_sem(formula: forms: F).
v(V)      := root_sem(formula: vars: V).
attr(A)   := root_sem(semf: attribute: A).
animate(A):= root_sem(semf: animate:A).

subj(S)   := root_sem(subj:S).
obj(O)    := root_sem(obj:O).
obj2(O2)  := root_sem(obj2:O2).


%% File Name: lexsem.cuf
%% Yoshiko Fujinami

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% lexical semantics
%


vi_type(R) :=
    verbal_sem &
    (subj: Actor) &
    semrepr(rel(R, (arg1: Actor) )).

cp_type(R) :=
    verbal_sem &
    (subj: Actor &
     obj2 : Object )  &
    semrepr(rel(R, (arg1: Actor &
            arg2: Object))).

vt_type(R) :=
    verbal_sem &
    (subj: Actor &
     obj : Object )  &
    semrepr(rel(R, (arg1: Actor &
            arg2: Object))).

vd_type(R) :=
    verbal_sem &
    (subj: Actor &
     obj : Object&
     obj2: Object2 )  &
    semrepr(rel(R,
            (arg1: Actor &
             arg2: Object &
             arg3: Object2))).

n_type(Idn) :=
    np_sem &
    index:Index &
    semrepr(rel('named', (arg1:Index &
                  arg2:Idn))).

cn_type(Rel) :=
    np_sem &
    (index:Index)&
    semrepr(rel(Rel, arg1:Index)).

no_noun(X, Y):=
    rel(no,
        (arg1:X & arg2:Y)).

conc(I, X, Y):=
    rel(conc,
        (arg1:I & arg2:X & arg3:Y)).


%% File Name: phrsem.cuf
%% Yoshiko Fujinami


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% phrasal semantics
%

%% exist X: np(X)&P(X)
np_semantics(index:X&forms(F)):=
%    root_sem(vars([X])&index:X&forms(F)).
    root_sem(vars([X])&vstore:X&forms(F)).


%% post propositions
pp_semantics:=
   leaves([root_sem(P)]) &
   root_sem(P).

%% adnominal particle
ap_semantics(no):=
    leaves([ i(Li)&f(Lf), i(Ri)&f(Rf)])&
%    root_sem(index:Ri &
    root_sem(vstore:Ri &
      vars([Ri])&
      forms( append([no_noun(Ri, Li)], append(Rf,Lf)))).

ap_semantics(no):=
    leaves([_, i(Li)&f(Lf), i(Ri)&f(Rf)])&
%    root_sem(index:Ri &
    root_sem(vstore:Ri &
            vars([Ri])&
            forms( append([no_noun(Ri, Li)], append(Rf,Lf)))).

ap_semantics(to):=
   leaves([ i(Li)&f(Lf), i(Ri)&f(Rf)])&
%   root_sem(index:I &
   root_sem(vstore:I &
          vars([Li, Ri])&
          forms( append([conc(I, Ri, Li)], append(Rf,Lf)))).

%-----------------------------------------------
%


adv_semantics:=
    leaves([root_sem(S)| _]) &
    root_sem(S).

adj_semantics:=
    leaves([root_sem(S)| _]) &
    root_sem(S).

%% File Name: verbsem.cuf
%% Yoshiko Fujinami
%%


%==================================================
% Definitions for Grammar
%      Intransitive Verb: sentential final, and adnominal forms
%      Transitive Verb:   sentential final, and adnominal forms
%      Copla Verb:        sentential final form
%      Verb with three complements: sentential final form
%                (nominative, accusative, locative)
%

%==================================================
% Intransitive Verb
%

v1( L, V, F):=
    v1_p0(L, V, F).

%==================================================
% Intransitive Verb without free gap
% Sentential Final Form
%

v1_p0(Label, V, F):=
    v(i) &
    v1_sem(vi_type(Label), V, F).


%--------------------------------------------------
% v1_sem: semantics for intransitive verb
%

v1_sem(subj:S & forms(F),
       V1,
       append(F, F1)) :=
    leaves([NOM]) &
    nom_sem(NOM, S, V1, F1).

%================================================
% Intransitive Verb
% Adnominal Form
%

v1_relative(Label, I, V, F):=
    v(irel) &
    virel_semantics(vi_type(Label), I, V, F).


%------------------------------------------------
% Semantics for intransitive verb
%

virel_semantics(subj:Index & forms(ViSem),
               Index, Var, append(NpSem, ViSem)):=
%               Index, Var, append(ViSem, NpSem)):=
    leaves([RNoun])&
    relativized_noun(RNoun, Index, Var, NpSem).


%================================================
%  Transitive Verb Definition
%

v2(Label, V, F):= v2_p0(Label, V, F).

v2_p0(Label, V, F):=
    v(t) &
    v2_sem(vt_type(Label), V, F).


%------------------------------------------------
%  Semantic Definition
%  Transitive Verb
%

v2_sem(subj:S & obj:O & forms(F),
       append(V1, V2),
       append(append(F, F1), F2)) :=

    leaves(per([NOM, ACC])) &
    nom_sem(NOM, S, V1, F1) &
    acc_sem(ACC, O, V2, F2).


%================================================
%  Transitive Verb
%  Adnominal Form
%

v2_relative(Label, I, V, F):=
    v(trel) &
    vtrel_semantics(vt_type(Label), I, V, F).



%------------------------------------------------
% Semantics Definition
% Transitive verb adnominal form
%

vtrel_semantics(subj: SIndex & obj: OIndex & forms(VtSem),
                OIndex, append(OV, SV),
%                append(SNp, append(ONp, VtSem))):=
                append(VtSem, append(SNp, ONp))):=

    leaves([NOM, RNoun])&
    nom_sem(NOM, SIndex, SV, SNp) &
    relativized_noun(RNoun, OIndex, OV, ONp).


%% transitive verb relative caluse
%% which lacks sbject

vtrel_semantics(subj: SIndex & obj: OIndex & forms(VtSem),
               SIndex,append(OV, SV),
%               append(SNp, append(ONp, VtSem))):=
               append(VtSem, append(SNp, ONp))):=

    leaves([ACC, RNoun])&
    acc_sem(ACC, OIndex, OV, ONp)&
    relativized_noun(RNoun, SIndex, SV, SNp).

%------------------------------------------------
% Copl verb

copl( L, V, F):=
    copl_p0(L, V, F).

copl(L, V, F):=
    copl_p1(L, V, F).

%------------------------------------------------
% Copl Verb
%    p0: without a free gap
%    p1: with one free gap
%

copl_p0(Label, V, F):=
    v(t) &
    copl_sem(cp_type(Label), V, F).

copl_p1(Label, V, F):=
    project(v(t), [_]) &
    copl_p1_sem(cp_type(Label), V, F).


%------------------------------------------------
% copl_p1_sem/4
%

copl_p1_sem(Sem, V, F):=
    p1_sem(n, Sem, V, F).
```

```
copl_p1_sem(Sem, V, F):=
    p1_sem(l, Sem, V, F).


%------------------------------------------------
%

copl_sem(subj:S & obj2:O & forms(F),
       append(V1, V2),
       append(append(F1, F2), F)) :=

    leaves(per([NOM, LOC])) &
    nom(Na)& loc(Lo)&
    nom_sem(NOM&animate(Na), S, V1, F1) &
    loc_sem(LOC&animate(Lo), O, V2, F2).


%------------------------------------------------
% p1_sem/4
%

p1_sem(n, subj:S & forms(F),
          V1,
          append(F1, F)):=
    leaves([NOM])&
    nom(Na)&
    nom_sem(NOM&animate(Na), S, V1, F1).

p1_sem(a, obj:O & forms(F),
          V1,
          append(F1, F)):=
    leaves([ACC])&
    acc(Aa)&
    acc_sem(ACC&animate(Aa), O, V1, F1).


p1_sem(l, obj2:O & forms(F),
          V1,
          append(F1, F)):=
    leaves([LOC])&
    loc(Lo)&
    loc_sem(LOC&animate(Lo), O, V1, F1).

/*
p1_sem(d, obj2:O & forms(F),
          V1,
          append(F1, F)):=
    leaves([DAT])&
    dat(Da)&
    dat_sem(DAT&animate(Da), O, V1, F1).
*/



%------------------------------------------------
%

v3(Label, V, F) :=
    v3_p0(Label, V, F).

v3(Label, V, F) :=
    v3_p2(Label, V, F).
```

```
v3(Label, V, F) :=
    v3_p1(Label, V, F).

%-------------------------------------------------
%

v3_p0(Label, V, F):=
    v(d) &
    v3_sem(vd_type(Label), V, F).

v3_p1(Label, V, F):=
    project(v(d), [_]) &
    v3_p1_sem(vd_type(Label),V,F).

v3_p2(Label, V, F):=
    project(v(d), [_,_]) &
    v3_p2_sem(vd_type(Label),V,F).

%-------------------------------------------------
% v3_sem/2
% semantics for verbs which takes three complements
% ann: subject is animate, objects are not animated
%

v3_sem(subj:S & obj:O & obj2:O2 & forms(F),
       append(V1, append(V2, V3)),
       append(F1, append(F2, append(F3, F))) ):=

    leaves(per([NOM, ACC, DAT]))&
    nom(Na) & acc(Aa) & loc(Da) &
    nom_sem(NOM& animate(Na), S, V1, F1) &
    acc_sem(ACC& animate(Aa), O, V2, F2) &
    loc_sem(DAT& animate(Da), O2,V3, F3).


%-------------------------------------------------
% v3_p1_sem
%


% project dative
v3_p1_sem(subj:S & obj:O & forms(F),
            append(V1, V2),
            append(F1, append(F2, F))):=

    leaves(per([NOM, ACC]))&
    nom(Na) & acc(Aa) &
    nom_sem(NOM& animate(Na), S, V1, F1) &
    acc_sem(ACC& animate(Aa), O, V2, F2).

% project accusative
v3_p1_sem(subj:S & obj2:O & forms(F),
            append(V1, V2),
            append(F1, append(F2, F))):=

    leaves(per([NOM, DAT]))&
    nom(Na) & loc(Da) &
    nom_sem(NOM& animate(Na), S, V1, F1) &
    loc_sem(DAT& animate(Da), O, V2, F2).

% project nominative
```

```
v3_p1_sem(obj:S & obj2:O & forms(F),
            append(V1, V2),
            append(F1, append(F2, F))):=

    leaves(per([ACC, DAT]))&
    acc(Aa) & loc(Da) &
    acc_sem(ACC&animate(Aa), S, V1, F1) &
    loc_sem(DAT&animate(Da), O, V2, F2).

%------------------------------------------------
%

v3_p2_sem(Vsem, V, F):=
    p1_sem(_, Vsem, V, F).



%% the locative is relarivized
v2_nd_relative_sem( subj:S&obj2:O&forms(F),
                   O,
                   V2,
                   append(F2, append(F, F1))):=

    leaves([NOM, LOC]) &
    nom_sem(NOM, S, _V1, F1) &
    relativized_noun(LOC, O, V2, F2).


%------------------------------------------------
% Verb adnominal form

% the acc is relativized
% the nominative phrase is projected
v3_p1_relative_sem(a, obj:O & obj2:D & forms(F),
                   O,
                   V2,
                   append(F2, append(F, F1))):=

    leaves([DAT, ACC]) &
    dat_sem(DAT, D, _V1, F1) &
    relativized_noun(ACC, O, V2, F2).

%------------------------------------------------
% definition for relativized noun
%

relativized_noun(root_sem(vstore:S&vars(V)&forms(F)), S, V, F) := _.


%------------------------------------------------
% definition for complements
%

nom_sem(nominative&root_sem(vstore:S&vars(V)&forms(F)), S, V, F):= _.
acc_sem(accusative&root_sem(vstore:S&vars(V)&forms(F)), S, V, F):= _.
dat_sem(dative&root_sem(vstore:S&vars(V)&forms(F)), S, V, F):= _.
loc_sem(locative&root_sem(vstore:S&vars(V)&forms(F)), S, V, F):= _.

%% File Name: interface.cuf
%% Yoshiko Fujinami

%================================================
% Interface
%
% c/3
% Def. for nouns
% c(+Flag, +Animate, +Label)
% Flag::    cn:common noun, n:noun
% Animate:: a: animate, n: not animate
%


%================================================
% Nouns
%

c(cn, n, Label):=
    np &
    np_semantics(cn_type(Label))&
    animate(-).

c(cn, a, Label):=
    np &
    np_semantics(cn_type(Label))&
    animate(+).

c(n, Label):=
    np &
    np_semantics(n_type(Label))&
    animate(+).


%================================================
% Particle

%------------------------------------------------
% Post Positional Particle
% c(+pp, +Case, +PForm)
% Case:  one of gr_info
% PForm: one of pf_info
%

c(pp, Case, PForm):=
    pp &
    pp_semantics &
    Case &
    pform(PForm).

%------------------------------------------------
% Adnominal Particle
% c(+ap, ApForm)
%

c(ap, AF):=
    ap(AF) &
    ap_semantics(AF) &
    apform(AF).

%------------------------------------------------
% Particle
% whichi is a complement of an adnominal particle
%

c(p2, _ApForm):=
    root_syn(p2)&satured.

%==================================================
% VERBS
% c(+Flag, +Label)
% Label:: the name of the verb
%

%--------------------------------------------------
% Intransitive Verb
%

c(v1_a, Label):=
    nom(+)&
    v1(Label, V, F) &
    root_sem(vars(V)&forms(F)).

c(v1_relative, Label):=
    v1_relative(Label, I, V, F)&
    root_sem(vstore:I& vars(V)& forms(F)).

%--------------------------------------------------
% Transitive Verb Sentence Final Form
%

c(v2_ao, Label):=
    nom(+)& acc(_) &
    v2(Label, V, F) &
    root_sem(vars(V)&forms(F)).

%--------------------------------------------------
% Transitive Verb Adnominal Form
%

c(v2_relative, Label):=
    v2_relative(Label, I, V, F)&
    root_sem(vstore:I& vars(V)& forms(F)).


%--------------------------------------------------
% Coplua verb
%

%--------------------------------------------------
% nominative is not animate
%

c(copl_nn, Label):=
    nom(-)&loc(-) &
    copl(Label, V, F) &
    sent_final &
    root_sem(vars(V)&forms(F)).

%--------------------------------------------------
% nominative is animate;
% dative is Not animate

c(copl_an, Label):=
    nom(+)&loc(-) &
    copl(Label, V, F)&
    sent_final &
    root_sem(vars(V)&forms(F)).
```

```
%-------------------------------------------------
% Ditransitive Verb
%

% nominative is animate;
% accusative and dative are Not
c(v3_ann, Label):=
    nom(+) & acc(-) & loc(-) &
    v3(Label, V, F) &
%    sent_final &
    root_sem(vars(V)&forms(F)).

% nominative is animate;
% others are any
c(v3_aoo, Label):=
    nom(+) & acc(_) & loc(_) &
    v3(Label, V, F)&
    sent_final &
    root_sem(vars(V)&forms(F)).


%=================================================
% VERB adnominal form
%

% example "owatta"
c(cp_rel, Label):=
    join_trees(adj, project(v(t), [_]))&
    v2_nd_relative_sem( cp_type(Label), I, V, F) &
    root_sem(vstore:I& vars(V)& forms(F)).


% example ".. to iu"
c(v3_p1_relative, Label):=
    join_trees(adj, project(v(d), [_, _])) &
    v3_p1_relative_sem(a, vd_type(Label), I, V, F)&
    root_sem(vstore:I& vars(V)& forms(F)).
```

I Data

(s1)

syupatsutiten	ga	sabaku	no	yoko	ni	aru
starting-point	NOM	*desert*	NO	*next-to*	LOC	*there-is*

"There is a starting point next to desert"

(s2)

tsuirakugenba	wa	sinkouhoukou	no	migigawa	ni	mieru
clash-spot	ACC	*heading-direction*	NO	*right*	LOC	*see*

"See the clash spot on your right hand side"

(s3)

atoti	to	iu	tokoro	no	hidarigawa	wo	tooru
ruin	DAT	*called*	*place*	NO	*left*	ACC	*pass-by*

"Pass by the left of place called ruin"

(s4)

sogen	mo	sinkouhoukou	no	migigawa	ni	mieru
plain	ACC	*heading-direction*	NO	*right-side*	LOC	*see*

"See a plain on your right hand side"

(s5)

atoti	no	kado	wo	magaru
ruin	NO	*corner*	ACC	*turn-to*

"Turn at the corner of the ruin"

(s6)

tsukiatari	no	tokoro	ni	sogen	ga	aru
end	NO	*place*	LOC	*plain*	NOM	*there-is*

"There is a plain at the end of the road"

(s7)

furo	no	hidarigawa	wo	tooru
bath	NO	*left*	ACC	*pass-by*

"Pass by the left of the bath"

(s8)

furo	wo	migite	ni	miru
bath	ACC	*right*	LOC	*see*

"See a bath on your right"

(s9)

yama	no	migite	ni	kamosika	ga	iru
mountain	NO	*right*	LOC	*mountain-goat*	NOM	*is*

"A mountain goat is on the right of the mountain"

(s10)

gake	no	hidarigawa	wo	tooru
cliff	NO	*left*	ACC	*pass-by*

"Pass by the left of the cliff"

(s11) gake no sita ni seitetsujo ga aru
cliff NO *bottom* LOC *iron-mill* NOM *there-is*
"There is an iron mill at the bottom of the cliff"

(s12) seitetsujo to ki to no aida wo tooru
iron-mill TO *tree* TO NO *gap* ACC *pass-through*
"Pass through the gap between the iron mill and the tree"

(s13) hayasi no hidarigawa wo susumu
forest NO *left* ACC *go*
"Go by the left of the forest"

(s14) migigawa ni magaru
right LOC *turn-to*
"Turn to the right"

(s15) hatake ga kojo no ue ni aru
farm NOM *factory* NO *upside* LOC *there-is*
"There is a farm on the upside of a factory"

(s16) kyanpu no hidari no koya no hidari wo tooru
camp NO *left* NO *cottage* NO *left* ACC *pass-by*
"Pass by the left of a cottage on the left hand side of the camp"

(s17) mitisirube ga numa no migigawa ni aru
guide-board NOM *bog* NO *right* LOC *there-is*
"There is a guide-board on the right of a bog"

(s18) syupatsutiten ga hidarigawa no ue ni aru
starting-point NOM *left* NO *upside* LOC *there-is*
"There is a starting point on the upside of the left"

(s19) ginko ga migisita ni aru
bank NOM *right-bottom* LOC *there-is*
"There is a bank on the bottom right"

(s20) ginko no hidarisita made iku
ban NO *left-down* LOC *go*
"Go till the bottom left of the bank"

(s21) ginko no sita made iku
bank NO *bottom* LOC *go*
"Go till the bottom of the bank"

(s22) kawa ga migiue ni aru
river NOM *right-upside* LOC *there-is*
"there is a river on the upside of the right"

(s23) boti ga ginko to kawa to no aida ni aru
graveyard NOM *bank* TO *river* TO NO *between* LOC *there-is*
"There is a graveyard between a bank and a river"

(s24) boti to kawa no aida wo tooru
graveyard TO *river* NO *between* ACC *pass*
" Pass between a graveyard and a river "

(s25) kawa no migi ni iru
river NO *right* LOC *are*
"You are on the right of a river"

(s26) hidarigawa ni kawa ga aru
left LOC *river* NOM *there-is*
"There is a river on the left"

(s27) kawa no ue wo iku
river NO *upside* ACC *go*
"Go on the upside of a river"

(s28) sabaku no sita made iku
desert NO *bottom* LOC *go*
"Go till the bottom of a desert"

(s29) e no hidarigawa wo iku
picture NO *left* ACC *go*
"Go on the left of the picture "

(s30) migigawa ni iku
right LOC *go*
"Go to the right"

(s31) syupatsutiten no migisita ni gake ga aru
start-point NO *right-bottom* LOC *cliff* NOM *there-is*
"There is a cliff on the right bottom of the starting point"

(s32) gake no sita ni seitetsujo ga aru
cliff NO *bottom* LOC *iron-mill* NOM *there-is*
"There is an iron mill on the bottom of a cliff"

(s33) seitetsujo no migi ni ki ga aru
iron-mill NO *right* LOC *tree* NOM *there-is*
"There is a tree on the right of an iron mill"

(s34) kojo no hidariue ni iku
factory NO *left-upside* LOC *go*
"Go to the upside left of a factory"

(s35) wan ga sita ni aru
bay NOM *bottom* LOC *there-is*
"There is an bay at the bottom"

(s36) syupatsutiten no sita ni sabaku ga aru
start-point NO *bottom* LOC *desert* NOM *there-is*
"There is a desert on the bottom of the start point"

(s37) bokugo no hidarisita ni tsuirakugenba ga aru
shelter NO *bottom-left* LOC *clash-spot* NOM *there-is*
"There is a clash spot on the bottom left of a shelter"

(s38) sabaku no hidarisita no haji ni iru
desert NO *left-bottom* NO *edge* LOC *are*
"You are at the edge on the left bottom of a desert"

(s39) atoti no ue wo tooru
ruin NO *upside* ACC *pass*
"Pass the upside of the ruin"

(s40) sogen ga yama no sita ni aru
plain NOM *mountain* NO *bottom* LOC *there-is*
"there is a plain on the bottom of the mountain"

(s41) yama ga atoti no migiue ni aru
mountain NOM *ruin* NO *right-upside* LOC *there-is*
"There is a plain on the right upside of a ruin"

(s42) mizutamari no ue ni yama ga aru
pond NO *upside* LOC *mountain* NOM *there-is*
"There is a mountain on the upside of a pond"

(s43) sogen no hidarigawa ni mizutamari ga aru
plain NO *left* LOC *pond* NOM *there-is*
"There is a pond on the left of a plain"

(s44)	furo	ga	yama	no	hidarigawa	ni	aru
	bath	NOM	*mountain*	NO	*left*	LOC	*there-is*

“There is a bath on the left of a mountain”

(s45)	syupatsutiten	no	sita	ni	kyanpu	ga	aru
	star-point	NO	*bottom*	LOC	*camp*	NOM	*there-is*

“There is a camp on the bottom of the start point”

(s46)	koya	no	hidarigawa	ni	iku
	cottage	NO	*left*	LOC	*go*

“Go to the left of a cottage”

(s47)	kokuyurin	ga	haikyo	no	sita	ni	aru
	forest	NOM	*ruin*	NO	*bottom*	LOC	*there-is*

“There is a forest on the bottom of a ruin”

(s48)	haikyo	no	ue	wo	tooru
	ruin	NO	*upside*	ACC	*pass-by*

“Pass by the upside of a ruin”

(s49)	haikyo	no	hidariue	ni	iku
	ruin	NO	*left-upside*	LOC	*go*

“Go to the upside of the left of the ruin”

(s50)	bokujo	ga	kinenhi	no	sita	ni	aru
	farm	NOM	*monument*	NO	*bottom*	LOC	*there-is*

“There is a farm on the bottom of a monument”

(s51)	syupatsutiten	ga	hidari	no	ue	no	hou	ni	aru
	start-point	NOM	*left*	NO	*upside*	NO	*around*	LOC	*there-is*

“There is a start point around the left upside”

(s52)	kozan	ga	sita	ni	aru
	mine	NOM	*bottom*	LOC	*there-is*

“There is a mine on the bottom”

(s53)	kozan	no	sita	made	iku
	mine	NO	*bottom*	LOC	*go*

“Go till the bottom of a mine ”

(s54)	ginko	no	atari	made	iku
	bank	NO	*somewhere*	LOC	*go*

“Go till somewhere a bank exists”

(s55) danso no mannaka wo tooru
dislocation NO *middle* ACC *pass-through*
"Pass through the middle of a dislocation"

(s56) danso no hidariue wo tooru
dislocation NO *left-upside* ACC *pass-by*
"Pass by the left upside of a dislocation"

(s57) boti no sita ni hasira ga aru
graveyard NO *bottom* LOC *pole* *there-is*
"There is a pole on the bottom of a graveyard"

(s58) kanu ga kawa to apati no mannaka ni aru
canoe NOM *river* TO *apache* NO *middle* LOC *there-is*
" There is a canoe in middle of a canoe and apache"

(s59) iwa no sita ni makiba ga aru
rock *bottom* LOC *farm* NOM *there-is*
"There is a pasture on the bottom of a rock"

(s60) ginko no hidarigawa wo tooru
bank NO *left* ACC *pass-by*
"Pass by the left of a bank"

(s61) ginko no sita wo tooru
bank NO *bottom* ACC *pass-by*
"Pass by the bottom of a bank"

(s62) haka no e ga owatta atari wo magaru
graveyard NO *picture* NOM *end* *somewhere* ACC *turn*
"Turn at somewhere the picture of a graveyard ends"

(s63) usi no sita wo tooru
cow NO *bottom* ACC *pass-by*
"Pass by the bottom of the cow"

(s64) makiba no yoko wo tooru
farm NO *side* ACC *pass-by*
"Pass by the side of the farm"

(s65) syupatsutiten ga hidariue no sunahama no tokoro ni aru
start-point NOM *left-upside* NO *beach* NO *place* LOC *there-is*
"There is a start point around the beach on the upside of the left"

(s66) kaigansen no tokoro wo iku
coast NO *place* ACC *iku*
"Go along the coast"

(s67) numa no sita no hou wo iku
swamp NO *bottom* NO *around* ACC *go*
"Go around the bottom of a swamp"

(s68) hidari no hou ni magaru
left NO *toward* LOC *turn*
"Turn toward left"

(s69) wan no tokoro wo sakeru
bay NO *place* ACC *avoid*
"Avoid the bay"

(s70) minka no yoko wo tooru
house NO *side* ACC *pass-by*
"Pass by the side of a house"

(s71) minka no hidari no tokoro wo tooru
house NO *left* NO *place* ACC *pass-by*
"Pass by the left of a house"

(s72) migi no hou ni iku
right NO *toward* LOC *go*
"Go toward right"

(s73) sita no hou ni iku
bottom NO *toward* LOC *go*
"Go toward the bottom"

(s74) gareki no sita no tokoro wo tooru
ruin NO *bottom* NO *place* ACC *pass-by*
"Pass by the bottom of the ruin"

(s75) denwa no sotogawa wo tooru
phone-booth NO *outside* ACC *go*
"Go outside of the phone booth"

(s76) denwa no migigawa wo tooru
phon-booth NO *right* ACC *pass-by*
"Pass by the right of the phon booth"

(s77) tizu no mannaka ni haikyo ga aru
map NO *center* LOC *ruin* NOM *there-is*
"There is a ruin in the center of the map"

(s78) mon no hidarigawa wo tooru
gate NO *left* ACC *pass-by*
"Pass by the left of the gate"

(s79) haikyo no migi made kuru
ruin NO *right* LOC *come*
"Come till the right of the ruin"

(s80) bokujo no ue no atari made kuru
farm NO *upside* NO *somewhere* LOC *come*
"Come till the upside of the farm"

(s81) haikyo no ue wo hidari ni iku
ruin NO *upside* ACC *left* LOC *go*
"Go to left at the upside of the ruin"

(s82) ki no migigawa ni mawaru
tree NO *right* LOC *turn*
"Turn to the right of the tree"

(s83) tenbodai ga hidariue no hou ni aru
observatory NOM *left-upside* NO *around* LOC *there-is*
"There is an observatory around the upside of the left"

(s84) kyanpu ga migisita no hou ni aru
camp NOM *right-bottom* NO *toward* LOC *there-is*
"There is a camp toward the bottom of the right"

(s85) kyanpu to taisho no iti ni gorira ga aru
camp TO *opposite* NO *side* LOC *gorira* NOM*there-is*
"There is a gorira on the opposite side of the camp"

(s86) kyanpu no hidari wo tooru
camp NO *left* ACC *pass-by*
"Pass by the left of the camp "

(s87) hidarisita no hou ni magaru
left-bottom NO *toward* LOC *turn*
"Turn toward the bottom left"

(s88) kyanpu no yoko made kuru
camp NO *next-to* LOC *come*
"Come next to the camp"

(s89) tsuribasi no hou ni iku
bridge NO *toward* LOC *go*
"Go toward the bridge"

(s90) kyanpu no haji made iku
camp NO *end* LOC *go*
"Go till the end of the camp"

(s91) kyanpu no hidari wo tooru
camp NO *left* ACC *pass-by*
"Pass by the left of the camp"

(s92) kyanpu no sita made kuru
camp NO *bottom* LOC *come*
"Come till the bottom of the camp"

(s93) tizu no hidarigawa no hou ni susumu
map NO *left* NO *toward* LOC *go*
"Go toward the left of the map"

(s94) ginko no hidari wo mawaru
bank NO *left* ACC *turn*
"Turn at the left of the bank"

(s95) ginko to danso no mannaka made kuru
bank TO *dislocation* NO *middle* LOC *come*
"Come till the middle between the bank and the dislocation"

(s96) ue ni agaru
up LOC *go*
"Go up"

(s97) syupatsutiten ga sunahama no tokoro ni aru
start-point NOM *beach* NO *place* LOC *there-is*
"There is a start point at the beach"

(s98) syupatsutiten ga hidariue no hou ni aru
start-point NOM *left-upside* NO *around* LOC *there-is*
"There is a start point around the upside left"

(s99)	kyoru	ga	sita	no	hou	ni	aru
	dinosaur	NOM	*bottom*	NO	*around*	LOC	*there-is*

"There is a dinosaur around the bottom"

(s100)	uti	wo	migigawa	ni	miru
	house	ACC	*right*	LOC	*see*

"See a house on the right"