TR-I-0346

A Grammar for Japanese Generation in the TUG Framework

Junko HOSAKA

March 1993

Abstract

This is a report on a Japanese generation grammar developed for the C-Star joint project. The grammar has been constructed in the framework of TUG and covers all 262 target sentences. I describe the grammar development environment, introduce the format, explain the work done and finally discuss further work that should be done.

ATR自動翻訳電話研究所
ATR Interpreting Telephony Research Laboratories

Contents

Chapter 1

Introduction

This is a report on a Japanese grammar developed for sentence generation in the framework of Trace and Unification Grammar[1][2]. It is intended for those who are going to refine the grammar. The work was done during my eight-month stay (April 06, 1992-September 10, 1992 and October 23, 1992-January 29, 1993) at Siemens in Munich for the C-Star speech translation joint project[1]. The project dealt with information about an international conference; the basic corpus includes 12 dialogues between the conference organizer and applicants.

The grammar covers all 262 sentences in the 12 dialogues. The processing time varies. Sentences such as *moshimoshi* and *iie* are processed in zero seconds and the sentence *kyoutoeki kara takushii de kaigizyou made iku niwa ikura gurai kakari masu ka* needed 200.14 seconds until the first result was generated. The mean processing time for the first result was 8.21 seconds on a SUN4 with 48 Megabytes of memory.

The grammar was used in the C-Star international joint experiment toward interpreting telephony, undertaken on January 28, 1993[3], without any modification.

[1] C-Star stands for Consortium for Speech Translation Advanced Research operated by, in an alphabetical order, ATR Interpreting Telephony in Japan, Carnegie Mellon University in the USA and Siemens AG in Germany.

Chapter 2

Development Conditions

2.1 Machines and Programming Languages

In developing the grammar, I used the parser compiler, parser, generator compiler and generator for TUG. In the first few months, the four programs were all written in ifprolog. Later, the parser compiler, generator compiler and generator were written in SNI-prolog; SNI-prolog is a prolog based on ifprolog developed by Siemens NIxdorf. The parser was written in quintus prolog.

The machine used in the first few months was a SUN3 (3/60) equipped with 8 Megabytes of memory. This was followed by a SUN4 (Sparc 2) equipped with 48 Megabytes of memory.

2.2 Input for Processing

It was determined that the input for Japanese generation would be the German analysis results. More precisely, the input was to be the **Q**uasi **L**ogical **F**orm (QLF)[4] translated from the actual German analysis results, the **P**seudo **L**ogical **F**orm (PLF)[1]. However, the PLF for German was not fully prepared. Therefore, I took the Japanese QLF directly from a Japanese analysis, which I developed to fill this temporary need. I developed a Japanese analysis grammar, took the Japanese analysis results as input for Japanese generation, and developed the Japanese generation grammar. This explains why I needed four different programs as mentioned in section 2.1. Transfer rules from German PLF to Japanese QLF were written for the purpose of machine translation[2].

The following subsections 2.2.1 and 2.2.2 introduce the formats for manual input[3].

2.2.1 Input for Japanese Analysis

Japanese analysis basically requires a sentence and an additional `utt(_,_,_,Phon,Tree,Sem)`[4], as shown in format 1. The sentence should be segmented into words and each word should be separated by a comma. The additional information starting with `utt` contains `Phon`, `Tree` and `Sem`, as Japanese grammar is set to analyse and generate sentences phonologically, syntactically and semantically[5],

[1] PLF was developed by R.Hunze primarily for database query.
[2] The transfer rules: German PLF → German QLF → Japanese QLF, were written by M.Gelurke.
[3] See section2.3.2 for automatic input.
[4] Three "_" in `utt` do not have any grammatical meaning.
[5] Strings with capital letters, such as WORD1 and WORD2, are variables.

FORMAT 1

```
jparse([WORD 1, WORD 2, ..., WORD n],utt(_,_,_,Phon,Tree,Sem)).
```

Consider example 1[6]:

Ex. 1 用紙を送って下さい。
youshi o okut te kudasai.

For example 1, the input for Japanese analysis is:

```
jparse([youshi, o, okut,te, kudasai],utt(_,_,_,Phon,Tree,Sem)).
```

2.2.2 Input for Japanese Generation

A Japanese generator requires a semantic form obtained by sentence analysis[7] and an additional `gen(utt(Phon,Tree, Sem), Output)`, as shown in format 2:

FORMAT 2

```
Sem = SEM OF ANALYSIS RESULT,
  gen(utt(Phon,Tree, Sem), Output).
```

For example 1, the input for Japanese generation is[8]:

```
Sem =
 give_favor(_40984,okuru
   (quant(nil,nonpolite,ex,_10844,
          restr(nil,[],[]),_10852),
    quant(syn(pro),ellipsis,_49540,
          restr(personal(_49540),[],[]),_49548),
    quant(syn(pro),ellipsis,_47632,
          restr(personal(_47632),[],[]),_47640),
    quant(syn(bare,nonpolite,nil),_4464,_3936,
          restr(youshi(_2744),[],[]),_4476))),
 gen(utt(Phon,Tree, Sem), Output).
```

2.3 Tools

To facilitate the grammar development, two kinds of tools were programmed: a recognizer of conjugational endings and a batch input reader. Unfortunately, there was no debugger, which is definitely indispensable for developing any grammar exceeding a toy grammar. Errors were found by eliminating or adding rules, in other words by trial and error.

[6] Examples in this report are adapted from the basic corpus; some are whole sentences and some are parts of a sentence. Therefore, I do not add any translation, except example 14.

[7] Analysis results of sentences for dialogues A and B are in Appendix A.

[8] Indentation was done manually by the author.

2.3.1 Recognizer of Conjugational Endings

TUG does not support morphology. Therefore, there exists a special program for expanding German verbal stems to different forms. Since Japanese is very rich in morphology, the manual expansion of all verbals is time consuming. I expanded some Japanese verbals such as auxiliary verbs (e.g. reru, da), subsidiary verbs (e.g. itasu, iru), irregular verbs (e.g. kuru, suru) and short verbs (e.g. aru). But for most of the verbals, I treated the morphology with unification rules .

A conjugational ending recognizer *japlexersun4* has been programmed, so that the analyser can recognize the combination of a stem and its ending and treat it as a word. To illustrate, the verb *okut* in example 1 consists of the stem "*oku*" and the conjugational ending "*t.*" By means of the conjugational ending recognizer, we can write `okut` as a word in `jparse`. The recognizer needs a list of endings *japsuffix*[9], namely, the compiled endings *japsuffixcom*. Some endings that can be combined with *oku* are:

```
ra.
re.
ri.
ro.
ru.
t.
```

2.3.2 Batch Input Reader

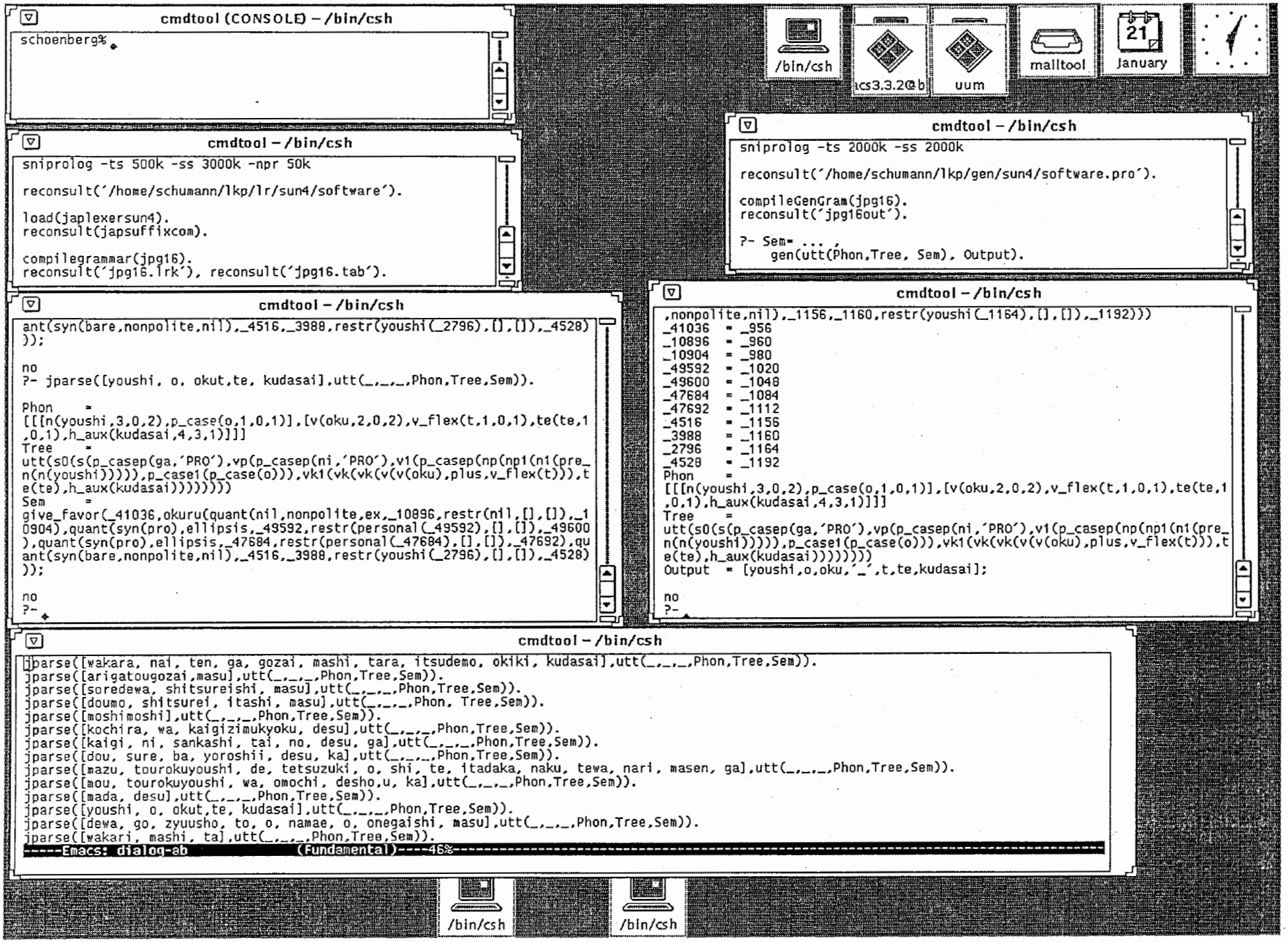
Typing in or cutting and pasting is effective for preparing input in the format described in section 2.2, if there are only a few sentences to be tested. However, to test more than 200 sentences, the input should be processed automatically. For convenience, a batch input reader for analyser *parseall* and a batch input reader for generator *generateall* have been programmed. The output of *parseall* is prepared as the input of *generateall*, and cannot be used for manual generation[10].

2.4 Work Space

The following page shows the work space for parsing and generation. The upper left window shows what one should load for parsing and how one can compile a grammar, and the middle left window shows the result of parsing example 1. The upper right window shows what one should load for generation, how one can compile a grammar and what the input should look like. The bottom window shows the input sentences for parsing.

[9] See Appendix B for the complete list of endings.

[10] See Appendix A and Appendix D for automatic input and output of generation.


cmdtool (CONSOLE) – /bin/csh
schoenberg%
cmdtool – /bin/csh
sniprolog -ts 500k -ss 3000k -npr 50k
reconsult('/home/schumann/lkp/lr/sun4/software').
load(japlexersun4).
reconsult(japsuffixcom).
compilegrammar(jpg16).
reconsult('jpg16.lrk'), reconsult('jpg16.tab').
cmdtool – /bin/csh
ant(syn(bare,nonpolite,nil),_4516,_3988,restr(youshi(_2796),[],[]),_4528)
));
no
?- jparse([youshi, o, okut,te, kudasai],utt(_,_,_,Phon,Tree,Sem)).
Phon =
[[[n(youshi,3,0,2),p_case(o,1,0,1)],[v(oku,2,0,2),v_flex(t,1,0,1),te(te,1
,0,1),h_aux(kudasai,4,3,1)]]]
Tree =
utt(s0(s(p_casep(ga,'PRO'),vp(p_casep(ni,'PRO'),v1(p_casep(np(np1(ni(pre_
n(n(youshi))))),p_case1(p_case(o))),vk1(vk(vk(v(v(oku),plus,v_flex(t))),t
e(te),h_aux(kudasai)))))))
Sem =
give_favor(_41036,okuru(quant(nil,nonpolite,ex,_10896,restr(nil,[],[]),_1
0904),quant(syn(pro),ellipsis,_49592,restr(personal(_49592),[],[]),_49600
),quant(syn(pro),ellipsis,_47684,restr(personal(_47684),[],[]),_47692),qu
ant(syn(bare,nonpolite,nil),_4516,_3988,restr(youshi(_2796),[],[]),_4528)
));
no
?-
cmdtool – /bin/csh
sniprolog -ts 2000k -ss 2000k
reconsult('/home/schumann/lkp/gen/sun4/software.pro').
compileGenGram(jpg16).
reconsult('jpg16out').
?- Sem= ... ,
gen(utt(Phon,Tree, Sem), Output).
cmdtool – /bin/csh
,nonpolite,nil),_1156,_1160,restr(youshi(_1164),[],[]),_1192)))
_41036 = _956
_10896 = _960
_10904 = _980
_49592 = _1020
_49600 = _1048
_47684 = _1084
_47692 = _1112
_4516 = _1156
_3988 = _1160
_2796 = _1164
_4528 = _1192
Phon =
[[[n(youshi,3,0,2),p_case(o,1,0,1)],[v(oku,2,0,2),v_flex(t,1,0,1),te(te,1
,0,1),h_aux(kudasai,4,3,1)]]]
Tree =
utt(s0(s(p_casep(ga,'PRO'),vp(p_casep(ni,'PRO'),v1(p_casep(np(np1(ni(pre_
n(n(youshi))))),p_case1(p_case(o))),vk1(vk(vk(v(v(oku),plus,v_flex(t))),t
e(te),h_aux(kudasai)))))))
Output = [youshi,o,oku,'_',t,te,kudasai];
no
?-
cmdtool – /bin/csh
jparse([wakara, nai, ten, ga, gozai, mashi, tara, itsudemo, okiki, kudasai],utt(_,_,_,Phon,Tree,Sem)).
jparse([arigatougozai,masu],utt(_,_,_,Phon,Tree,Sem)).
jparse([soredewa, shitsureishi, masu],utt(_,_,_,Phon,Tree,Sem)).
jparse([doumo, shitsurei, itashi, masu],utt(_,_,_,Phon, Tree,Sem)).
jparse([moshimoshi],utt(_,_,_,Phon,Tree,Sem)).
jparse([kochira, wa, kaigizimukyoku, desu],utt(_,_,_,Phon,Tree,Sem)).
jparse([kaigi, ni, sankashi, tai, no, desu, ga],utt(_,_,_,Phon,Tree,Sem)).
jparse([dou, sure, ba, yoroshii, desu, ka],utt(_,_,_,Phon,Tree,Sem)).
jparse([mazu, tourokuyoushi, de, tetsuzuki, o, shi, te, itadaka, naku, tewa, nari, masen, ga],utt(_,_,_,Phon,Tree,Sem)).
jparse([mou, tourokuyoushi, wa, omochi, desho,u, ka],utt(_,_,_,Phon,Tree,Sem)).
jparse([mada, desu],utt(_,_,_,Phon,Tree,Sem)).
jparse([youshi, o, okut,te, kudasai],utt(_,_,_,Phon,Tree,Sem)).
jparse([dewa, go, zyuusho, to, o, namae, o, onegaishi, masu],utt(_,_,_,Phon,Tree,Sem)).
jparse([wakari, mashi, ta],utt(_,_,_,Phon,Tree,Sem)).
-----Emacs: dialog-ab (Fundamental)----46%
/bin/csh
mailtool
January
uum
/bin/csh
/bin/csh

Chapter 3

Grammar

3.1 Grammar Format

The rules in TUG basically have a context-free part and an augmentation part. The augmentation is mostly made by a feature equation and can contain a mixture of an attribute-value-pair and Prolog-term unification. A feature path is strictly proved by a feature-type declaration[1], and a Prolog-term is indicated by a backquote. A lexicon can have the same kind of augmentation as the rules. Both rules and lexicons allow the usage of disjunctions and macros.

Further, TUG provides two kinds of writing comments, '%' for each line and '/*' and '*/' for more than one line. The '%' can be put at the beginning of a comment line, and a comment part can be surrounded by '/*' and '*/,' as in examples 2 and 3, respectively:

Ex. 2
```
% semantics
```

Ex. 3
```
/*
quant(<SynFeatures>,<QuantifierHead>,<Variable>,
  restr(<HeadRestriction>,<Modifiers>,<Genitives>),
  <Scope>)
  */
```

In the following subsections 3.1.1 through 3.1.4, I will introduce a way of defining types, macros, rules and lexicons with examples[2].

3.1.1 Declaration Format

There are two kinds of declarations: a type declaration for all categories and a semantic declaration exclusively for the purpose of generation. Types must be declared in order to prove the feature path correctness. Both declarations must be made prior to the macros, the rules and the lexicons.

The type declaration has format 3:

FORMAT 3

```
CATEGORY => f(ATTRIBUTE-NAME 1:TYPE 1, .....
                ATTRIBUTE-NAME n:TYPE n).
```

[1] Unfortunately, an error message that uses the type declaration is not very informative. Localizing the error in the grammar is therefore not straightforward.

[2] Unfortunately, there is no documentation for developing a grammar in the TUG framework.

The number of types for a category is arbitrary; types need only be separated by a comma.

Some examples of type declarations are:

```
utt => f(phon,tree,sem).
s   => f(phon:phon,unary:yesno,syn:v_syn,tree,sem:v_sem).
```

utt is the top category in the Japanese grammar. As phon, tree and sem are declared for utt, their information can be extracted in the processing result, as displayed on page 7.

All attributes must be further defined as follows:

```
phon          => f(p1,p2).
v_syn         => f(st:stem_type,flex:flex,te,ad:adjacent_cond,
                   sc:v_syn_sc,pre:pre_v,topic,polite).
sem           => f(expr,var).
v_sem         => f(expr,var,sc:v_sem_sc).
v_sem_sc      => f(arg1,arg2,arg3).
```

To remember a value used in the grammar, the value can also be noted, as in format 4:

FORMAT 4

```
TYPE => {VALUE 1,VALUE 2, ....., VALUE n}.
```

In the current TUG version, however, the value declaration is used neither for checking nor for processing.

Some examples of the value declaration are:

```
yesno         => {yes, no}.
stem_type     => {godan_k,godan_k_iku,godan_g,godan_s,godan_t,
                  godan_n,godan_b,
                  godan_m,godan_r,godan_w,itidan,kahen,sahen,
                  pre_v_sahen,
                  v_sahen,keiyosi,aux_keiyosi,excep}.
pre_v         => {jp,pre_jp,sino_jp}.
```

Semantic declaration is required for generation. The information is written in format 5:

FORMAT 5

```
semantics(CATEGORY,CATEGORY:TYPE).
```

Only the declared semantic types are taken into account for processing generation. The following example shows that utt considers all types of sem, while s and aux consider only expr of sem.

```
semantics(utt,utt:sem).
semantics(s,s:sem:expr).
semantics(aux,aux:sem:expr).
```

3.1.2 Macro Format

A macro is defined using short_for, as shown in format 6:

FORMAT 6

```
TERM(ARG1,ARG2,ARG3) short_for
   ATTRIBUTE 1 = VALUE 1,
   ATTRIBUTE 2 = VALUE 2,
               :
   ATTRIBUTE n  = VALUE n.
```

Macros can have the same kind of augmentation as rules and lexicons, namely a feature equation and a mixture of an attribute-value-pair and Prolog-term unification. Macros can be used both in rules and in lexicons. An example of a macro is as follows:

```
phon_struc(M,D1,D2) short_for
        M:phon:p1 = '[D1:phon:p1|M:phon:p2],
        D1:phon:p2 = D2:phon:p1,
        D2:phon:p2 = [].
```

The above macro produces square brackets to build a certain constituent. This is written because the Japanese speech synthesis system requires a slightly different structure than the Japanese generation.

3.1.3 Rule Format

A rule starts with a context-free rule. As format 7 shows, the expansion is indicated by an arrow --->. Each category is separated by a comma. The completion of a context-free rule is indicated by a vertical bar |. Features follow mostly in the form of attribute-value-pairs. Each feature is separated by a comma and the rule ends with a period.

FORMAT 7

```
CATEGORY 1 ---> CATEGORY 2, CATEGORY 3 |
       ATTRIBUTE 1 = VALUE 1,
       ATTRIBUTE 2 = VALUE 2,
                   :
       ATTRIBUTE n = VALUE n.
```

An example of a sentence rule is as follows:

```
s3 ---> s, s_prt |
        phon_liste(s3,s,s_prt),
        s3:sem:expr = s_prt:sem:expr,
        s:sem:expr = s_prt:sem:expr2,
        s3:sem:var = s:sem:var,
        s3:syn = s_prt:syn,
        s:syn:flex = s_prt:syn:inh:flex,
        s3:tree = 's3(s:tree,s_prt:tree).
```

As described in section 3.1.2, the Japanese speech synthesis system requires a special structure. `phon_liste`, used in rule `s3`, is a macro, which does not produce any square brackets. After that, semantic and syntactic feature equations follow. The last line contains a Prolog term and using this rule a tree structure is constructed.

Furthermore, TUG allows recursion and disjunction. Disjunction can be used in the lexicons as well.

Recursion

A recursive rule can be written by using an indicator; this prevents self-recursion. In the following example `s:v` is of this kind.

```
s --->  pp, s:v |
        phon_struc(s,pp,v),
        v:unary = no,
        s:syn = v:syn,
        s:sem:expr = 'and(pp:sem:expr,v:sem:expr),
        pp:sem:var = v:sem:var,
        s:sem:var = v:sem:var,
        s:tree = 's(pp:tree,v:tree).
```

The above rule says, `s` goes to `pp` and `s`. Both the `s` on the left-hand side and that on the right-hand side have the same features, but `s` on the right-hand side has another indicator, `v`. In the feature equation `v` is used for the `s` on the right-hand side.

Disjunction

There are two kinds of disjunction: value disjunction as in format 8 and condition disjunction as in format 9.

FORMAT 8

FEATURE PATH in {VALUE 1, VALUE 2, ..., VALUE n}

FORMAT 9

```
(ATTRIBUTE-VALUE-PAIRS OF CONDITION 1
 ;ATTRIBUTE-VALUE-PAIRS OF CONDITION 2
          :
 ;ATTRIBUTE-VALUE-PAIRS OF CONDITION n)
```

The following is an example of a postposition drop, in which the disjunction of values is used:

```
p_case1 ---> [] |
        p_case1:phon:p2 = p_case1:phon:p1,
        p_case1:syn:topic = nil,
        p_case1:syn:case in {ga,o,ni},
        p_case1:syn:empty = yes,
        p_case1:tree = 'p_case1(p_case1:syn:case,'DROP').
```

The fourth line from the top of the rule indicates that the dropped postposition can be either `ga`, `o` or `ni`.

The following is an example of a nominal phrase consisting of a noun phrase and a postposition, in which the disjunction of conditions is used:

```
p_casep ---> np, p_case1 |
        phon_struc(p_casep,np,p_case1),
        p_casep:empty = no,
        p_casep:syn = p_case1:syn,
        np:syn:topic = p_case1:syn:topic,
        (p_case1:syn:empty = yes,
         np:syn:s_use = yes
         ;p_case1:syn:empty = no),
        p_casep:sem = np:sem,
        p_casep:tree = 'p_casep(np:tree, p_case1:tree).
```

The sixth through eighth lines from the top indicate that if `p_case1:syn:empty` is `yes` then `np:syn:s_use` must be `yes`; otherwise, `p_case1:syn:empty` is `no` and does not have any further restriction.

3.1.4 Lexicon Format

A lexicon starts with `lexicon` followed by a word, a comma and the category of the word in round brackets. The declaration of a word and its category is completed by a vertical bar |. Features of the word follow in the form of attribute-value-pairs. Each feature is separated by a comma. The whole lexicon ends with a period. The features can also be determined with a Prolog term, as in rules.

FORMAT 10

```
lexicon(WORD, CATEGORY) |
        ATTRIBUTE 1 = VALUE 1,
        ATTRIBUTE 2 = VALUE 2,
                    :
        ATTRIBUTE n = VALUE n,
```

The lexicon example for the interjection *moshimoshi* is as follows:

```
lexicon(moshimoshi,interj) |
        phon_lex(interj,interj(moshimoshi,4,1,2)),
        interj:sem:expr = 'tel_phatic(interj:sem:var),
        interj:tree = interj(moshimoshi).
```

The second line of the lexicon `phon_lex` is written for the speech synthesis system.

For Japanese Speech Synthesis

In the framework of the C-Star project, the result of Japanese generation is given to the Japanese speech synthesis system, ATR ν-Talk[3][5]. Besides the tree structure, for which the first argument of `phon_lex` is used in TUG, the values of the second argument are defined exclusively for ATR ν-Talk. In the lexicon example of `moshimoshi` the values have the following meanings[4]:

[3] ATR stands for Automatic Tuning of Rules for prosody control.

[4] For details see [6].

moshimoshi	phonemic transcription
4	Number of Mora
1	accent position
2	accent attribute

The phonemic transcription is taken from the Hepburn system. However, two phonemes, an unvoiced vowel and a syllabic nasal, are treated differently:

```
lexicon(desu, cop) |
        phon_lex(cop,cop('desu#',2,0,1)),
             :
             :       .
lexicon(ten, n) |
        phon_lex(n,n(teN,2,0,2)),
             :
             :       .
```

An unvoiced vowel such as u in desu is followed by a #. In this case the whole transcription is surrounded by single quotations. A syllabic nasal is written as N.

3.2 Current Status

The grammar in question has 584 lexicon entries with 57 terminal symbols, and 158 rules with 60 nonterminals. Macros have been prepared for subcategorization, conjugation types and phonemic information[5].

Roughly speaking, the grammar treats declarative, interrogative and imperative sentences, which are expressed by verbal conjugation forms and sentence final postpositions. Further, it treats negation by auxiliary verbs, as well as coordination using either postpositions or a certain verbal conjugation form. Nominal phrases can have different kinds of adnominals, such as genitives and relative sentences.

In the following sections: 3.2.1 through 3.2.6, I will introduce the outline of the current grammar.

3.2.1 Sentence Types

Four different sentence types are defined: a sentence finishing with a sentence particle s3, a sentence consisting of more than one main verbal phrase s2, a sentence containing only one main verbal phrase s0 and a sentence consisting of an interjection interj.

```
utt ---> s3
utt ---> s2
utt ---> s0
utt ---> interj
```

s3 ends with a sentence particle s_prt. s3 can have one main verbal phrase s and more than one main verbal phrase s2. The conjugation form of the proceeding verbal phrase is determined by each s_prt.

[5]The macros were written by Dr. U.Block.

```
s3 ---> s2, s_prt|
          :
        s2:syn:flex = s_prt:syn:inh:flex,
          :    .
s3 ---> s, s_prt|
          :
        s:syn:flex = s_prt:syn:inh:flex,
          :    .
```

s2 can end either with s finishing with a verbal complex or s1 finishing with a conjunctive postposition. When a sentence ends with a verbal complex, then it must use a certain conjugation form which is determined by the feature flex. Conjunctive postpositions are typically used in the middle of a sentence, but in some cases they can be used sentence-finally[7]. The location of conjunctive postpositions is controlled by the feature locate. Some sentence-final postpositions are used in the middle of a sentence as well. This is controlled by the feature quote. Sentence coordination can be realized by using a certain conjugation form. The last rule of s2 is of this type, and s0 takes only certain conjugation forms to complete a sentence[6].

```
s2 ---> s3, s|
          :
        s3:syn:quote = direct,
        s:syn:flex in {syusi,meirei,meirei_yo,meirei_pol},
          :    .
s2 ---> s3, s1 |
          :
        s3:syn:quote = direct,
        s1:syn:locate in {fin,mid_fin},
          :    .
s2 ---> s1, s |
          :
        s:syn:flex in {syusi,meirei,meirei_yo,meirei_pol},
          :    .
s2 ---> s1|
          :
        s1:syn:locate in {fin,mid_fin},
          :    .
s2 ---> s, s0 |
          :
        s:syn:flex = renyo,
          :    .
```

3.2.2 Free Phrase Order

Word/phrase order is quite free in Japanese. This is true not only for postpositional arguments consisting of a nominal and a case marker, but also for adverbial phrases and postpositional phrases in which postpositions do not have a case marker function.

An adverbial phrase *sudeni* in example 4 can be used before the postpositional phrase *touroku-youshi wa* as in example 5[7]

Ex. 4 登録用紙はすでにお持ちでしょうか。
touroku-youshi wa sudeni omochi desho u ka.

Ex. 5 すでに登録用紙はお持ちでしょうか。
sudeni touroku-youshi wa omochi desho u ka.

[6] s0 can be replaced by s with an indicator.

[7] The adverbial phrase can be used after the verbal complex *omochi desho u ka* as well. However, we did not treat this placement, since it did not appear in our basic corpus.

This kind of free phrase order is treated in context-free rules by using recursion[8]. Context-free rules with an adverbial phrase adv_p are:

```
s ---> adv_p, s:v |
vp ---> adv_p, vp:v |
v1 ---> adv_p, v1:v |
vk1 ---> adv_p, vk1:v |
```

Postpositional phrases are treated in parallel with the rules for adverbial phrases[9].

For example 4, two sentences are generated. Syntactic trees tree for the sentences are as follows:

```
GENERATION
tree =
utt(s3(s(p_casep(ga,PRO),
         vp(adv_p(adv(sudeni)),
            vp(p_casep(np(np1(n1(pre_n(n(tourokuyoushi))))),
                       p_case1(p_case(wa))),
               v1(vk1(vk(vk(vk(v(v(omochi))),
                         aux(desho)),aux(u))))))),s_prt(ka))).
```

```
GENERATION
tree =
utt(s3(s(p_casep(ga,PRO),
         vp(p_casep(np(np1(n1(pre_n(n(tourokuyoushi))))),
                    p_case1(p_case(wa))),
            v1(vk1(adv_p(adv(sudeni)),
                   vk1(vk(vk(vk(v(v(omochi))),
                             aux(desho)),aux(u))))))),s_prt(ka))).
```

3.2.3 Nominal Phrase and its Adnominal Phrase

Defining a nominal phrase as a phrase that can be a subject when used with a case marker 'ga[8],' there are two kinds of nominal phrases np: a nominal phrase consisting of verbal complex vk and a nominal phrase consisting of just a nominal phrase np1.

```
np ---> vk |
        np:phon = vk:phon,
        vk:syn:st = v_sahen,
        vk:syn:flex = renyo_noun,
        vk:syn:pre = sino_jp,
        np:syn:adv_use = no,
        np:syn:topic = vk:syn:topic,
        np:syn:s_use = yes,
        np:sem:expr = vk:sem:expr,
        np:tree = 'np(vk:tree).
```

[8] See subsection 3.1.3 for recursion notation.

[9] See section 3.3.1 for free postpositional phrase order.

```
np ---> np1 |
        np:phon = np1:phon,
        np:syn:polite = np1:syn:polite,
        np:syn:adv_use = np1:syn:adv_use,
        np:syn:topic = np1:syn:topic,
        np:syn:s_use = np1:syn:s_use,
        np:syn:harm = np1:syn:harm,
        np:sem = np1:sem,
        np:tree = 'np(np1:tree).
```

Generally speaking, a whole sentence can be a nominal phrase. However this type of rule generates a huge number of sentences. To avoid excessive generation, I allowed only a verbal complex vk as a nominal phrase. Further, as a possible vk I allowed only a certain type sino of a certain verb v_sahen and a certain conjugation form renyo_noun.

sanka in example 6 is treated as verbal complex vk. This is the conjugated form renyo_noun of a v_sahen verb *sanka suru*. The whole vk is treated as an np. In the current version, I allowed postposition drops for this nominal phrase. Therefore, we get two different trees and the same sem.

Ex. 6 まだ参加可能です。
mada sanka kanou desu.

GENERATION

```
tree =
p_casep(np(vk(v(v(sanka)))),
        p_case1(p_case(ga)))

sem =
identical(quant(syn(polite),ex,_1024,
           restr(nil,[],[]),_1068),
          sanka_suru(quant(nil,_1096,ex,_1100,
                     restr(nil,[],[]),_1120),_1152,_1156),
          quant(syn(nonpolite,_1176),_1192,_1196,
                possible(_1196),_1208))
```

GENERATION

```
tree =
p_casep(np(vk(v(v(sanka)))),
        p_case1(ga,DROP))

sem =
identical(quant(syn(polite),ex,_1024,
           restr(nil,[],[]),_1068),
          sanka_suru(quant(nil,_1096,ex,_1100,
                     restr(nil,[],[]),_1120),_1152,_1156),
          quant(syn(nonpolite,_1176),_1192,_1196,
                possible(_1196),_1208))
```

A nominal phrase np1 can be just a nominal phrase n1 or can have adnominal phrases as well. German grammar in the framework of TUG has two kinds of adnominals: genitive, and other modifiers such as a relative sentence. In the same way, n1 distinguishes genitive modifiers n1:sem:gen and other kinds of modifiers n1:sem:mod.

```
np1 ---> n1 |
        np1:phon = n1:phon,
        np1:syn = n1:syn,
        np1:sem:expr = 'quant(syn(bare,n1:syn:polite,n1:syn:topic),
                              _,n1:sem:var,n1:sem:expr,_),
        n1:sem:gen = [],
        n1:sem:mod = [],
        np1:tree = 'np1(n1:tree).
```

Four types of adnominal phrases have been prepared: genitive pp_gen, relative clause rel_s, adverb adv and attribute att.

Genitive

A genitive phrase is marked with a postposition *no*.

```
n1 ---> pp_gen, n1:h |
        phon_liste(n1,pp_gen,h),
        n1:syn = h:syn,
        n1:syn:topic = h:syn:topic,
        h:sem:expr = n1:sem:expr,
        h:sem:gen = '[pp_gen:sem:expr|n1:sem:gen],
        n1:sem:mod = h:sem:mod,
        n1:tree = 'n1(pp_gen:tree, h:tree).
```

In example 7, *siemens no* is the adnominal pp_gen and modifies the name *susanne kaiser*.

Ex. 7 シーメンスのスザンネ＝カイザーです。
siemens no susanne kaiser desu.

tree and sem of the generated np1 for example 7 are as follows:

GENERATION

```
tree =
np1(n1(pp_gen(np(np1(n1(pre_n(n(siemens))))),
              p_gen(no)),
       n1(name(name_germ
                 (name1_germ1(name1_germ(susanne)),
                  name2_germ2(name2_germ(kaiser)))))))

sem =
restr(name_germ(susanne,kaiser),
      [genitive(no,quant(syn(bare,nonpolite,_1160),_1180,_1184,
                         restr(siemens(_1188),[],[]),_1216))],[])
```

Relative Sentence

A relative sentence which modifies a nominal phrase must have a certain conjugation form rentai[10]. This is defined in the rel_s rule.

```
n1 ---> rel_s, n1:h |
        phon_struc(n1,rel_s,h),
        n1:syn = h:syn,
        rel_s:topic:sem:expr = h:sem:var,
        h:sem:expr = n1:sem:expr,
        h:sem:mod = '[rel_s:sem:expr|n1:sem:mod],
        n1:sem:gen = h:sem:gen,
        n1:tree = 'n1(rel_s:tree, h:tree).
```

[10]Note that Japanese does not have relative pronouns such as 'which' and 'who.'

In example 8, *wakara nai* is the adnominal `rel_s` and modifies the noun *ten*.

Ex. 8 分からない点がございましたらいつでもお聞き下さい。
wakara nai ten ga gozai mashi tara itsudemo okiki kudasai.

The syntactic tree and semantic of the generated `np1` for example 8 are as follows:

GENERATION

```
tree =
np1(n1(rel_s(rel_phrase(ga,ANTE),
             s(p_casep(ga,PRO),
               vp(p_casep(ga,TRACE),
                  v1(vk1(vk(vk(v(v(waka),plus,
                                 v_flex(ra))),aux(nai)))))))),
            n1(pre_n(n(ten)))))

sem =
restr(ten(_1348),[],
      [negate(_1360,wakaru(quant(_1364,_1368,ex,_1372,
                                 restr(nil,[],[]),_1392),
                           quant(syn(pro),ellipsis,_1432,
                             restr(personal(_1432),[],[]),__1460)
                             ,_1488))])
```

Adverb

Adverbs usually modify verbals. But some adverbs can function as adnominals for some rare nouns, e.g. those expressing numbers and shapes (e.g. circle, triangle).

```
n1 ---> adv, n1:h |
        phon_struc(n1,adv,h),
        n1:syn = h:syn,
        n1:syn:topic = h:syn:topic,
        adv:syn:n_mod = yes,
        h:syn:n_mod = yes,
        h:sem:expr = n1:sem:expr,
        h:sem:mod = '[adv:sem:expr|n1:sem:mod],
        n1:sem:gen = h:sem:gen,
        n1:tree = 'n1(adv:tree, h:tree).
```

In example 9, an adverb *oyoso* modifies the price *roku sen en*.

Ex. 9 およそ六千円かかります。
oyoso roku sen en kakari masu.

To control the adnominal usage of `adv`, the attribute `n_mod` is introduced. As defined in the rule of `n1`, the value of `n_mod` must be `yes`. The lexicon of *oyoso* is as follows and has the value `yes`.

```
lexicon(oyoso, adv) |
        phon_lex(adv,adv(oyoso,3,0,2)),
        adv:syn:bare = bare,
        adv:syn:n_mod = yes,
        adv:sem:expr =  'degr(quant(syn(adv:syn:polite)),
                                   oyoso,adv:sem:var),
        adv:tree = adv(oyoso).
```

A nominal phrase that has been modified must have the value `yes` for `n_mod` as well. This is determined in the rule for `price`:

```
pre_n ---> price |
        pre_n:phon = price:phon,
        pre_n:syn:suf:wh_ka = no,
        pre_n:syn:n_mod = yes,
        pre_n:sem:expr = price:sem:expr,
        pre_n:tree = 'pre_n(price:tree).
```

tree and sem of the generated np1 for example 9 are as follows:

GENERATION

```
tree =
np1(n1(adv(oyoso),
       n1(pre_n(price(num_unit(sen_unit(num1_9(roku),
                                        sen_unit(sen))),
                      currency_jp(en))))))

sem =
restr(jp_en(num_unit(6,000)),[],
            [degr(quant(syn(_1116)),oyoso,_1136)])
```

Attribute

Deictic attributes such as *sono* and *kono* function as an adnominal.

```
n1 ---> att, n1:h |
        phon_struc(n1,att,h),
        n1:syn = h:syn,
        n1:syn:topic = h:syn:topic,
        h:sem:expr = n1:sem:expr,
        h:sem:mod = '[att:sem:expr|n1:sem:mod],
        n1:sem:gen = h:sem:gen,
        n1:tree = 'n1(att:tree, h:tree).
```

In example 10 *annaisho* is modified by the deictic attribute *sono*:

Ex. 10 その案内書を送って下さい。
sono annaisho o okut te kudasai.

tree and sem of the generated np1 for example 10 are as follows:

GENERATION

```
tree =
np1(n1(att(sono),
       n1(pre_n(n(annaisho)))))

sem =
restr(annaisho(_1320),[],[att(far,_1332)])
```

3.2.4 Postposition Drops

In natural conversation, postpositions are often unspoken. The postposition *wa* in example 11 can be dropped as in example 12[11].

Ex. 11 そちらは会議事務局ですか。
sochira wa kaigizimukyoku desu ka.

[11] Example 12 is the only sentence in which the postposition is dropped in 262 sentences.

Ex. 12 そちら会議事務局ですか。
sochira kaigizimukyoku desu ka.

Possible explanations for postposition drops are that the sentences are uttered in informal environments and quite fast. However, current German analysis cannot provide this type of information for controlling Japanese postposition drops. To allow postposition drops for all nominal phrases will just lead to generating too many sentences.

A study based on a large dialogue database suggests relations between syntactical categories and postposition drops[9]. In the grammar in question, I introduced postposition drops for case markers. And in line with the study, postposition drops are allowed only if case markers are used with a pronoun or an adverbial noun such as nouns indicating day, month and money.

The following three rules say that a postpositional phrase p_casep can have a case marker p_case1 with p_case1:syn:empty = no but it can also be empty p_case1:syn:empty = yes. If a p_casep does not have a case marker, then the attribute np:syn:s_use must be yes.

```
p_casep ---> np, p_case1 |
        phon_struc(p_casep,np,p_case1),
        p_casep:empty = no,
        p_casep:syn = p_case1:syn,
        np:syn:topic = p_case1:syn:topic,
        (p_case1:syn:empty = yes, np:syn:s_use = yes
         ;p_case1:syn:empty = no),
        p_casep:sem = np:sem,
        p_casep:tree = 'p_casep(np:tree, p_case1:tree).

p_case1 ---> p_case |
        p_case1:phon = p_case:phon,
        p_case1:syn:topic = p_case:syn:topic,
        p_case1:syn:case = p_case:syn:case,
        p_case1:syn:empty = no,
        p_case1:tree = 'p_case1(p_case:tree).

p_case1 ---> [] |
        p_case1:phon:p2 = p_case1:phon:p1,
        p_case1:syn:topic = nil,
        p_case1:syn:case in {ga,o,ni},
        p_case1:syn:empty = yes,
        p_case1:tree = 'p_case1(p_case1:syn:case,'DROP').
```

Pronouns and adverbial nouns have a value yes for s_use and the case marker can be dropped. On the other hand, common nouns and some other nouns have a value no for s_use and so the postposition drops are not allowed.

The generated syntactic trees of example 12 are as follows:

GENERATION

```
tree =
utt(s3(s(p_casep(np(np1(n1(pre_n(pron(sochira))))),
                 p_case1(p_case(ga))),
         cop_p(np(np1(n1(pre_n(n(kaigizimukyoku))))),
               copk(cop(desu)))),s_prt(ka))).
```

```
tree =
utt(s3(s(p_casep(np(np1(n1(pre_n(pron(sochira))))),
                 p_case1(ga,DROP)),
         cop_p(np(np1(n1(pre_n(n(kaigizimukyoku))))),
               copk(cop(desu)))),s_prt(ka))).
```

3.2.5 Name, Address and Telephone Number

Name `name`, address `adresse` and telephone number `tel_num` are constructed exclusively to treat the sentences used in the basic corpus. If `adresse` and `name` are used with a copula complex `copk`, then they can only be used in this order. They can be used as a nominal phrase as well. On the other hand `tel_num` is used only with `copk` and cannot be used as a nominal phrase. Context-free rules for the restriction are as follows:

```
cop_p ---> adresse, name, copk |
cop_p ---> tel_num, copk |
n1 ---> name |
n1 ---> adresse |
```

In the current rules, German names, German addresses and Japanese telephone numbers are treated. Further, extension to other languages has been taken into consideration, as the following context-free rules indicate:

```
name ---> name_germ |
adresse ---> adresse_germ |
tel_num ---> tel_num_jp |
```

In `adresse_germ` the constituents are used in the German order, but the numerals used are pronounced in the Japanese way. Example 13 is not pronounced 'vierzig desu' but 'yon zyuu desu.'

Ex. 13 四十です。
40 *desu.*

Numerals are constructed from 0 to 99,999 by rules. Variants are written in separate lexicons and controlled by syntactic feature `harm`. Below are three variants for the numeral eight:

```
lexicon(hachi, num1_9) |
        phon_lex(num1_9,num1_9(hachi,2,0,1)),
        num1_9:syn:harm:neutral = yes,
        num1_9:syn:harm:voiceless = nil,
        num1_9:syn:harm:hpb = nil,
        num1_9:syn:harm:gatu = yes,
        num1_9:syn:harm:niti = n,
        num1_9:syn:harm:zyu = yes,
        num1_9:sem:expr = 8,
        num1_9:tree = 'num1_9(hachi).
```

```
lexicon(has, num1_9) |
        phon_lex(num1_9,num1_9(has,1,0,1)),
        num1_9:syn:harm:neutral = no,
        num1_9:syn:harm:voiceless = s,
        num1_9:syn:harm:hpb = nil,
        num1_9:syn:harm:gatu = no,
        num1_9:syn:harm:niti = nil,
        num1_9:syn:harm:zyu = no,
        num1_9:sem:expr = 8,
        num1_9:tree = 'num1_9(has).

lexicon(hap, num1_9) |
        phon_lex(num1_9,num1_9(hap,1,0,1)),
        num1_9:syn:harm:neutral = no,
        num1_9:syn:harm:voiceless = nil,
        num1_9:syn:harm:hpb in p,p_edit,
        num1_9:syn:harm:gatu = no,
        num1_9:syn:harm:niti = nil,
        num1_9:syn:harm:zyu = no,
        num1_9:sem:expr = 8,
        num1_9:tree = 'num1_9(hap).
```

3.2.6 Frozen Expression

Some expressions consisting of more than one word are used only in a certain order and only with certain words. Lexicon entry of such expressions will reduce the processing time. However, to make the rules general, I limited the frozen expressions to the following two:

```
lexicon(douitashimashite, interj)
lexicon(moushiwakearimasenga, idiom)
```

3.3 Future Work

The current version of the Japanese generation grammar has been built as part of a speech translation system. The input is the result of German-to-Japanese transfer and the generated sentences are further given to the Japanese speech synthesis system. Therefore, there were interface issues both with transfer and synthesis, in order to treat the basic corpus.

There are some phenomena in Japanese which I could not fully implement, such as relatively free argument order and preferred word order in a verbal complex. In the following sections: 3.3.1 through 3.3.4, I will describe the problems.

3.3.1 Argument Phrase Order

Arguments marked with a case marker p_case can be scrambled rather freely. The order might be controlled by the focus. But German analysis cannot provide adequate information. In the C-Star project the grammar should generate a minimal number of sentences so as to achieve fast processing. Therefore, I did not treat the scrambling, but only the canonical order.

The verb *miru* has the ga-o order, as written in the macro sc[12] in the following lexicon:

[12] It is impossible to define obligatory arguments in Japanese. I determined obligatory case markers in sc only for practical reasons.

```
lexicon(mi, v_stem:v) |
        phon_lex(v,v(mi,1,0,2)),
        sc(v,ga,o,nil),
        v:syn:st = itidan,
        v:syn:pre = jp,
        v:syn:te = te,
        v:syn:polite = nonpolite,
        v:sem:expr = 'miru(quant(v:syn:topic,v:syn:polite,ex,
                                 v:sem:var,
                                 restr(nil,[],[]),_),
                                 v:sem:sc:arg1,v:sem:sc:arg2),
        v:tree = v(mi).
```

Example 14[13] has the canonical order of *miru* and can be analyzed and generated correctly.

Ex. 14 私が案内書を見ます。
watashi ga annaisho o mi masu.

Example 14 is syntactically generated as follows:

GENERATION

```
tree =
utt(s0(s(p_casep(np(np1(n1(pre_n(pron(watashi))))),
                 p_case1(p_case(ga))),
         vp(p_casep(np(np1(n1(pre_n(n(annaisho))))),
                    p_case1(p_case(o))),
            v1(vk1(vk(vk(v(v(mi))),aux(masu))))))).
```

GENERATION

```
tree =
utt(s0(s(p_casep(np(np1(n1(pre_n(pron(watashi))))),
                 p_case1(ga,DROP)),
         vp(p_casep(np(np1(n1(pre_n(n(annaisho))))),
                    p_case1(p_case(o))),
            v1(vk1(vk(vk(v(v(mi))),aux(masu))))))).
```

As a case marker can be dropped freely when combined with a pronoun[14], example 14 yields two results.

Example 15 is a scrambled variant of example 14, as can be seen from the reversed o-ga order:

Ex. 15 案内書を私が見ます。
annaisho o watashi ga mi masu.

A scrambled variant such as example 15 fails in analysis and no sentence can be generated.

In the 12 basic dialogues we have one sentence with non-canonical order as seen in example 16:

Ex. 16 案内書に論文の題目が載っております。
annaisho ni ronbun no daimoku ga not te ori masu.

[13] This example is not treated in the basic corpus. It means that I am going to check the announcement.
[14] See section 3.2.4.

The canonical order is defined as **ga-ni** in the macro sc:

```
lexicon(no, v_stem:v) |
        phon_lex(v,v(no,1,0,2)),
        sc(v,ga,ni,nil),
        v:syn:st = godan_r,
        v:syn:pre = jp,
        v:syn:te = te,
        v:sem:expr = 'noru(quant(v:syn:topic,_,ex,v:sem:var,
                                 restr(nil,[],[]),_),
                                 v:sem:sc:arg1,v:sem:sc:arg2),
        v:tree = v(no).
```

Unlike example 15, this sentence can be accepted. The phrase with a postposition **ni** is analyzed and generated as an adjunct with postp, and not as an argument with p_case. A phrase with a p_case **ni** is treated as zero pronoun. As a phrase with postp can be used in any position of a sentence[15], we get two results for example 16:

GENERATION

```
tree =
utt(s0(s(pp(np(np1(n1(pre_n(n(annaisho))))),postp(ni)),
         s(p_casep(np(np1(n1(pp_gen(np(np1(n1(pre_n(n(ronbun))))),
                                    p_gen(no)),
                                 n1(pre_n(n(daimoku))))))),
                   p_case1(p_case(ga))),
           vp(p_casep(ni,PRO),
              v1(vk1(vk(vk(vk(v(v(no),plus,v_flex(t))),
                              te(te),
                              h_aux(ori)),aux(masu))))))))).
```

GENERATION

```
tree =
utt(s0(s(p_casep(np(np1(n1(pp_gen(np(np1(n1(pre_n(n(ronbun))))),
                                  p_gen(no)),
                               n1(pre_n(n(daimoku))))))),
                 p_case1(p_case(ga))),
         vp(p_casep(ni,PRO),
            v1(vk1(pp(np(np1(n1(pre_n(n(annaisho))))),postp(ni)),
                   vk1(vk(vk(vk(v(v(no),plus,v_flex(t))),
                                te(te),
                                h_aux(ori)),aux(masu))))))))).
```

Out of 262 sentences this was the only one generated erroneously[16]. This suggests the appropriateness of introducing canonical order for practical natural language processing.

To treat the argument phrase order properly, simply allowing all possible phrase orders with a case marker leads only to producing overly many sentences, and is thus not the best solution. To find an effective control of phrase order, collaborative study with analysis and transfer is indispensable.

[15] See section 3.2.2.

[16] The erroneous generation did not effect the Japanese speech synthesis system ATR ν-Talk.

3.3.2 Word Order in Verbal Complex

A Japanese verbal complex is rich in verbals such as auxiliary verbs. In the current version, the grammar controls the conjugation types, conjugation forms and some adjacent conditions, such as the usage after the causative and the particles *te* and *no* and the usage of negation.

Conjugation types and forms are determined in the feature `inh:st` and `inh:flex`, respectively, and the adjacent conditions in the feature `ad`. An example of the auxiliary verb *nai* is as follows:

```
lexicon(nai, aux) |
        phon_lex(aux,aux(nai,2,1,1)),
        (godan(aux:syn:inh:st)
         ;aux:syn:inh:st in {itidan,kahen,sahen,keiyosi}),
        aux:syn:inh:flex = mizen,
        aux:syn:st = aux_keiyosi,
        aux:syn:flex in {syusi, rentai},
        aux:syn:ad:att = no,
        aux:syn:ad:neg = nai,
        aux:syn:ad:no_desu = no,
        aux:sem:expr = 'negate(aux:syn:topic,aux:sem:expr2),
        aux:tree = aux(nai).
```

Following is the rule for the verbal complex `vk`, in which `inh` and `ad` are treated. The rule is recursively usable :

```
vk ---> vk:a, aux |
        phon_liste(vk,a,aux),
        vk:syn:sc = a:syn:sc,
        vk:syn:st = aux:syn:st,
        vk:syn:flex = aux:syn:flex,
        vk:syn:topic = aux:syn:topic,
        a:syn = aux:syn:inh,
        vk:syn:ad = aux:syn:ad,
        a:syn:ad:neg = aux:syn:ad:neg,
             :          .
```

When we started grammar construction, we decided that the German-to-Japanese transfer rules should determine the order of verbals. However, it might also help to consider some preferred orders[10] to control the order in the Japanese grammar. It became clear that if we want to go beyond the basic corpus, the transfer rules cannot decide the order based on the German analysis result.

3.3.3 Relative Sentence

Relative sentences generally have a clear case relation. In example 8 in section 3.2.3, the postcedent *ten* is the object of the verbal complex *wakara nai*. However, in some cases such as example 17, the relation between *baai* and *sankasuru* is not straightforward.

Ex. 17 代理人が参加する場合はあらかじめこちらまでお知らせ下さい。
dairinin ga sankasuru baai wa
arakazime kochira made oshirase kudasai

The generation parallels that of example 8. `tree` and `sem` of `np1` for example 17:

GENERATION

```
tree =
np1(n1(rel_s(rel_phrase(ni,ANTE),
             s(p_casep(np(np1(n1(pre_n(n(dairinin))))),
                       p_case1(p_case(ga))),
               vp(p_casep(ni,TRACE),
                  v1(vk1(vk(v(v(sanka),plus,v_flex(suru))))))))),
       n1(pre_n(n(baai)))))

sem =
restr(baai(_1304),[],
     [sanka_suru(quant(_1316,nonpolite,ex,_1320,
                       restr(nil,[],[]),_1340),
                 quant(syn(bare,nonpolite,nil),_1388,_1392,
                       restr(dairinin(_1396),[],[]),_1424),_1452)])
```

The noun *baai* does not fill the *ni*-argument of the verb *sankasuru*. Therefore, the above syntactic tree does not reflect the actual relative construction of example 17, and neither does the corresponding sem value.

It was not clear to me how this type of construction could be treated in the TUG framework.

3.3.4 Passive

The Japanese passive is realized by using the auxiliary verb *reru*. In example 18, the passive auxiliary verb is attached to the verb *kaisaisuru*, conjugated as *re* and *kaisaisa* respectively.

Ex. 18 会議は京都国際会議場で開催されます。
kaigi wa kyoutokokusaikaigizyou de kaisaisa re masu.

The verb *kaisaisuru* in example 18 takes the *ga*-argument and the *o*-argument. Therefore, the nominal phrase *kaigi wa* can be syntactically analyzed both as subject and object, since the topic marker *wa* can be interpreted in both ways. The following are parts of the generated tree and sem for example 18, when the phrase *kaigi wa* is interpreted as object[17]:

GENERATION

```
tree =
s0(s(p_casep(ga,PRO),
     vp(p_casep(np(np1(n1(pre_n(n(kaigi))))),
                p_case1(p_case(wa))),
        v1(vk1(vk(vk(vk(v(v(kaisai),plus,v_flex(sa))),
                        aux(re)),aux(masu)))))))

sem =
polite(_1104,
       passive(_1108,
           kaisai_suru(quant(_1112,nonpolite,ex,_1116,
                             restr(nil,[],[]),_1136),
                       quant(syn(pro),ellipsis,_1176,
                             restr(personal(_1176),[],[]),_1204),
                       quant(syn(bare,nonpolite,wa),_1248,_1252,
                             restr(kaigi(_1256),[],[]),_1284))))
```

[17] The adjunctive phrase *kyoutokokusaikaigizyou de* is omitted.

As shown in the result, the passive auxiliary verb *reru* does not subcategorize. In the current version, main verbs and adjectives subcategorize, and the subcategorization is valid in the entire verbal complex. The passive auxiliary verb should subcategorize as well. However, it was not clear how to map the subcategorization of auxiliary verbs onto that of verbs in the TUG framework.

Acknowledgement

I would like to express my thanks to Dr.Hans-Ulrich Block for introducing TUG, to Christine Zuenkler for explaining the syntactic part of TUG and to Rudolf Hunze for explaining the semantic part. I also thank Manfred Gehrke for collaborative work in his transfer rule construction. Without the appropriate output of German to Japanese transfer, Japanese generation would not have been possible. Ludwig Schmid was helpful when there was machine trouble. Yasunari Harada (Waseda U.) confirmed the naturalness of the generated Japanese sentences and gave me valuable advice for rule construction.

Special thanks go to Hartmut Raffler, Dr.Peter Kleinschmidt and Dr.Akira Kurematsu, who gave me the opportunity to work with the researchers at Siemens AG.

Bibliography

[1] Block,H.U.(1991):"Compiling Trace & Unification Grammar for Parsing and Generation," Proc. of the reversible grammar workshop, ACL.

[2] Block,H.U., Schachtl,S.(1992):"Trace & Unification Grammar," Proc. of COLING-92, Vol.1, pp.87-93.

[3] Yato,H., Takezawa,T., Sagayama,S., Uratani,N., Morimoto,T., Kurematsu,A.(1993): Execution Report of International Joint Experiment toward Interpreting Telephony, TR-I-0338 (in Japanese).

[4] Alshawi,H.(1990):"Resolving Quasi Logical Forms," Computational Linguistics Vol.16, No.3, pp.133-144.

[5] Sagisaka,Y., Kaiki,N.,Iwahashi,N., Mimura,K.(1992):"ATR ν-Talk Speech Synthesis System," Proc. of ICSLP-92, Vol.1, pp.483-486.

[6] Sagisaka,Y., Sato,H.(1984):"Accentuation Rules for Japanese Text-to-Speech Conversion," REVIEW of the Electrical Communication Laboratories, Vol.32, No.2, pp.188-199.

[7] Hosaka,J., Takezawa,T.(1992):"Construction of Corpus-Based Syntactic Rules for Accurate Speech Recognition," Proc of COLING-92, Vol.2, pp.806-812.

[8] Shinozaki,N., Mizuno,Y., Ogura,K., Yoshimoto,K.(1989): User's Manual for Morpheme Information, TR-I-0077 (in Japanese).

[9] Hosaka,J., Takezawa,T., Uratani,N.(1992):"Analyzing Postposition Drops in Spoken Japanese," Proc. of ICSLP-92, Vol.2, pp.1251-1254.

[10] Yoshimoto,K., Kogure,K.(1988): Japanese Sentence Analysis by means of Phrase Structure Grammar, TR-I-0049 (in Japanese).

Appendix A

Input for Japanese Generation

This is a list of sem information extracted from analysis results of dialogues A and B, when using the batch input reader introduced in subsection 2.3.2. This is part of the input for generating Japanese sentences automatically. For manual generation, a slightly different format is required[1].

```
sem = tel_phatic(_1356).

sem = request_inf(identical(quant(syn(polite),ex,_10124,restr(nil,[],[]),_10232),qu
ant(syn(bare,_3456,wa),_5312,_4784,restr(quant(syn(prs2(deix_far),hearer,_3540,wa),
perspron,_3516,restr(personal(_3516),[],[]),_3524),[],[]),_5324),quant(syn(bare,non
polite,_10808),_11844,_11316,restr(kaigizimukyoku(_8608),[],[]),_11856))).

sem = assertive_yes(_1356).

sem = mann(quant(syn(polite)),sou,quant(syn(pro),ellipsis,_4760,restr(personal(_476
0),[],[]),_4768)).

sem = moderate(_144764,polt_aux(_143232,desire(_20512,moushikomu(quant(_15876,nonpo
lite,ex,_15956,restr(nil,[],[]),_15964),quant(syn(pro),ellipsis,_156852,restr(perso
nal(_156852),[],[]),_156860),quant(syn(bare,nonpolite,nil),_6704,_6176,restr(kaigi(
_4752),[],[]),_6716))))).

sem = request_inf(and(tloc(quant(syn(_9336)),sudeni,_9376),guess(_28820,polt_aux(_1
2264,motsu(quant(_10860,polite,ex,_9376,restr(nil,[],[]),_10948),quant(syn(pro),ell
ipsis,_67320,restr(personal(_67320),[],[]),_67328),quant(syn(bare,nonpolite,wa),_61
08,_5580,restr(tourokuyoushi(_4412),[],[]),_6120))))).

sem = assertive_no(_1356).

sem = tloc(quant(syn(polite)),mada,quant(syn(pro),ellipsis,_4760,restr(personal(_47
60),[],[]),_4768)).

sem = perfective(_11740,polite(_3860,wakaru(quant(_2552,_2624,ex,_2632,restr(nil,[]
,[]),_2640),quant(syn(pro),ellipsis,_32452,restr(personal(_32452),[],[]),_32460),qu
ant(syn(pro),ellipsis,_30576,restr(personal(_30576),[],[]),_30584)))).

sem = and(conect(quant(syn(_2660)),soredewa,_2700),polite(_27764,condescend(_19852,
okuru(quant(nil,polite,ex,_2700,restr(nil,[],[]),_17704),quant(syn(pro),ellipsis,_4
9484,restr(personal(_49484),[],[]),_49492),quant(syn(pro),ellipsis,_45868,restr(per
sonal(_45868),[],[]),_45876),quant(syn(bare,nonpolite,nil),_7284,_5060,restr(tourok
uyoushi(_3024),[],[]),_7296))))).

sem = polite(_58092,negau(quant(_46288,polite,ex,_46368,restr(nil,[],[]),_46376),qu
ant(syn(pro),ellipsis,_79228,restr(personal(_79228),[],[]),_79236),coord(and(to),qu
ant(syn(bare,polite,_9448),_10560,_10032,restr(zyuusho(_4796),[],[]),_10572),quant(
syn(bare,polite,nil),_30148,_29268,restr(namae(_27604),[],[]),_30160)))).

sem = identical(quant(syn(polite),ex,_44972,restr(nil,[],[]),_45080),quant(syn(bare
,nonpolite,wa),_6340,_5812,restr(zyuusho(_4644),[],[]),_6352),quant(syn(bare,nonpol
```

[1] Compare with format 2 in subsection 2.2.2.

```
ite,nil),_49756,_49204,restr(np_sem(adresse_germ(adr_strasse(schellingstrasse,zahl_
germ(num_unit(1,2))),adr_ort(zahl_plz(num_unit(8,'000')),muenchen,zahl_pzb(num_unit
(4,'0'))))),[],[]),_49768)).

sem = identical(quant(syn(polite),ex,_9784,restr(nil,[],[]),_9892),quant(syn(bare,n
onpolite,wa),_4300,_3772,restr(namae(_2604),[],[]),_4312),quant(syn(bare,nonpolite,
nil),_12520,_11996,restr(name_germ(berta,stangl),[],[]),_12532)).

sem = perfective(_11740,polite(_3860,wakaru(quant(_2552,_2624,ex,_2632,restr(nil,[]
,[]),_2640),quant(syn(pro),ellipsis,_32452,restr(personal(_32452),[],[]),_32460),qu
ant(syn(pro),ellipsis,_30576,restr(personal(_30576),[],[]),_30584)))).

sem = and(mann(quant(syn(_8804)),shikyuu,_8844),polite(_93112,receive_favor(_91508,
cause(nil,okuru(quant(_10328,nonpolite,ex,_8844,restr(nil,[],[]),_10420),quant(syn(
pro),ellipsis,_118988,restr(personal(_118988),[],[]),_118996),quant(syn(pro),ellips
is,_117032,restr(personal(_117032),[],[]),_117040),quant(syn(bare,nonpolite,wa),_55
76,_5048,restr(tourokuyoushi(_3880),[],[]),_5588))))).

sem = and(and(tloc(quant(syn(_92256)),itsudemo,_70308),give_favor(_112188,kiku(quan
t(nil,polite,ex,_70308,restr(nil,[],[]),_111028),quant(syn(pro),ellipsis,_135576,re
str(personal(_135576),[],[]),_135584),quant(syn(pro),ellipsis,_133572,restr(persona
l(_133572),[],[]),_133580)))),cond_tara(_70308,polite(_62496,aru_polite(quant(_5518
0,polite,ex,_55256,restr(nil,[],[]),_55264),quant(syn(bare,nonpolite,nil),_45372,_4
3948,restr(ten(_23092),[],[negate(_6456,wakaru(quant(_5112,_5184,ex,_5192,restr(nil
,[],[]),_5200),quant(syn(pro),ellipsis,_29108,restr(personal(_29108),[],[]),_29116)
,_5172))]),_45384))))).

sem = polite(_2124,thanking(quant(_1736,polite,ex,_1808,restr(nil,[],[]),_1816))).

sem = and(conect(quant(syn(_2316)),soredewa,_2356),polite(_20532,shitsurei_suru(qua
nt(_2788,nonpolite,ex,_2356,restr(nil,[],[]),_2872),quant(syn(pro),ellipsis,_43900,
restr(personal(_43900),[],[]),_43908)))).

sem = and(state(quant(syn(_2304)),doumo,_2344),polite(_19624,condescend(_4708,shits
urei_suru(quant(nil,_2780,ex,_2344,restr(nil,[],[]),_2860),quant(syn(pro),ellipsis,
_42356,restr(personal(_42356),[],[]),_42364))))).

sem = tel_phatic(_1356).

sem = identical(quant(syn(polite),ex,_9228,restr(nil,[],[]),_9336),quant(syn(bare,_
2560,wa),_4416,_3888,restr(quant(syn(prs1(deix_near),speaker,_2644,wa),perspron,_26
20,restr(personal(_2620),[],[]),_2628),[],[]),_4428),quant(syn(bare,nonpolite,_9912
),_10948,_10420,restr(kaigizimukyoku(_7712),[],[]),_10960)).

sem = moderate(_167120,polt_aux(_165588,desire(_40540,sanka_suru(quant(_15916,nonpo
lite,ex,_15996,restr(nil,[],[]),_16004),quant(syn(pro),ellipsis,_179448,restr(perso
nal(_179448),[],[]),_179456),quant(syn(bare,nonpolite,nil),_6616,_6088,restr(kaigi(
_4664),[],[]),_6628))))).

sem = request_inf(and(polt_aux(_25596,yoroshii(quant(_21660,_21732,ex,_5760,restr(n
il,[],[]),_21748),quant(syn(pro),ellipsis,_41616,restr(personal(_41616),[],[]),_416
24),_21720)),cond_ba(_5760,and(mann(quant(syn(_4272)),dou,_4312),suru(quant(_4624,n
onpolite,ex,_4312,restr(nil,[],[]),_4712),quant(syn(pro),ellipsis,_12052,restr(pers
onal(_12052),[],[]),_12060),quant(syn(pro),ellipsis,_10248,restr(personal(_10248),[
],[]),_10256))))).

sem = moderate(_158144,and(tloc(quant(syn(_6320)),mazu,_6360),and(postp_optn(de,_63
60,quant(syn(bare,nonpolite,nil),_11928,_9188,restr(tourokuyoushi(_6684),[],[]),_11
940)),negate(polite(_141556,change(_141164,negate(wa,receive_favor(_89796,suru(quan
t(nil,nonpolite,ex,_6360,restr(nil,[],[]),_47156),quant(syn(pro),ellipsis,_182344,r
estr(personal(_182344),[],[]),_182352),quant(syn(bare,nonpolite,nil),_39000,_38296,
restr(tetsuzuki(_24772),[],[]),_39012))))))))).

sem = request_inf(and(tloc(quant(syn(_4616)),mou,_4656),guess(_26976,polt_aux(_1722
4,motsu(quant(_14648,polite,ex,_4656,restr(nil,[],[]),_14736),quant(syn(pro),ellips
is,_53848,restr(personal(_53848),[],[]),_53856),quant(syn(bare,nonpolite,wa),_10048
,_7824,restr(tourokuyoushi(_5400),[],[]),_10060))))).

sem = tloc(quant(syn(polite)),mada,quant(syn(pro),ellipsis,_4760,restr(personal(_47
60),[],[]),_4768)).

sem = give_favor(_40984,okuru(quant(nil,nonpolite,ex,_10844,restr(nil,[],[]),_10852
```

),quant(syn(pro),ellipsis,_49540,restr(personal(_49540),[],[]),_49548),quant(syn(pro),ellipsis,_47632,restr(personal(_47632),[],[]),_47640),quant(syn(bare,nonpolite,nil),_4464,_3936,restr(youshi(_2744),[],[]),_4476))).

sem = and(conect(quant(syn(_4460)),dewa,_4500),polite(_68740,negau(quant(_56644,polite,ex,_4500,restr(nil,[],[]),_56732),quant(syn(pro),ellipsis,_96344,restr(personal(_96344),[],[]),_96352),coord(and(to),quant(syn(bare,polite,_14476),_17468,_15244,restr(zyuusho(_7316),[],[]),_17480),quant(syn(bare,polite,nil),_38116,_37412,restr(namae(_35932),[],[]),_38128))))).

sem = identical(quant(syn(polite),ex,_59452,restr(nil,[],[]),_59560),quant(syn(bare,nonpolite,wa),_6540,_6012,restr(zyuusho(_4844),[],[]),_6552),quant(syn(bare,nonpolite,nil),_64332,_63748,restr(np_sem(adresse_germ(adr_strasse(ludwigstrasse,zahl_germ(num_unit(3,2))),adr_ort(zahl_plz(num_unit(8,'000')),muenchen,zahl_pzb(num_unit(4,'0'))))),[],[]),_64344)).

sem = identical(quant(syn(polite),ex,_9784,restr(nil,[],[]),_9892),quant(syn(bare,nonpolite,wa),_4300,_3772,restr(namae(_2604),[],[]),_4312),quant(syn(bare,nonpolite,nil),_12520,_11996,restr(name_germ(hartmut,raffler),[],[]),_12532)).

sem = perfective(_11740,polite(_3860,wakaru(quant(_2552,_2624,ex,_2632,restr(nil,[],[]),_2640),quant(syn(pro),ellipsis,_32452,restr(personal(_32452),[],[]),_32460),quant(syn(pro),ellipsis,_30576,restr(personal(_30576),[],[]),_30584)))).

sem = request_inf(guess(_50440,polt_aux(_48712,iru(quant(_12012,_12080,ex,_12088,restr(nil,[],[]),_12096),quant(syn(bare,nonpolite,wa),_8204,_7676,restr(sankaryou(_6508),[],[]),_8216))))).

sem = assertive_yes(_1356).

sem = and(postp_optn(toshite,_4876,quant(syn(bare,nonpolite,nil),_6112,_5584,restr(tourokuhi(_4516),[],[]),_6124)),and(quant(syn(bare,polite,_12536),_13648,_13120,restr(hitori(_11952),[],[]),_13660),identical(quant(syn(polite),ex,_4876,restr(nil,[],[]),_37388),quant(syn(bare,_25168,nil),_26644,_25900,restr(jp_en(num_unit(3,5,'000')),[],[]),_26656),quant(syn(nonpolite,_37964),_38012,_35564,necessary(_35564),_38024)))).

sem = request_inf(mann(quant(syn(polite)),sou,quant(syn(pro),ellipsis,_6556,restr(personal(_6556),[],[]),_6564))).

sem = and(state(quant(syn(_2560)),doumo,_2600),perfective(_18140,polite(_3968,thanking(quant(_2912,polite,ex,_2600,restr(nil,[],[]),_2992))))).

sem = polite(_11204,condescend(_2860,shitsurei_suru(quant(nil,_2268,ex,_2340,restr(nil,[],[]),_2348),quant(syn(pro),ellipsis,_24260,restr(personal(_24260),[],[]),_24268)))).

Appendix B

Conjugational Endings

```
ba.
be.
bi.
bo.
bu.
chi.
e.
ga.
ge.
gi.
go.
gu.
i.
ka.
kat.
ke.
kere.
ki.
ko.
ku.
ma.
me.
mi.
mo.
mu.
n.
na.
ne.
ni.
no.
nu.
o.
ra.
re.
ri.
ro.
ru.
sa.
se.
shi.
shiro.
shiyo.
so.
su.
sure.
suru.
t.
ta.
te.
to.
tsu.
u.
wa.
yo.
nasara.
nasai.
nasaru.
nasare.
```

Appendix C

Japanese Generation Grammar

C.1 Declaration

```
utt             => f(phon,tree,sem).
s4              => f(phon:phon,syn:v_syn,tree,sem:v_sem).
s3              => f(phon:phon,syn:s_prt_syn,tree,sem:v_sem).
s2              => f(phon:phon,syn:v_syn,tree,sem:v_sem).
s1              => f(phon:phon,syn:v_conj_syn,tree,sem:v_sem).
s0              => f(phon:phon,syn:v_syn,tree,sem:v_sem).
s               => f(phon:phon,unary:yesno,syn:v_syn,tree,sem:v_sem).
rel_s           => f(phon:phon,unary:yesno,topic:rel_phrase,syn:v_syn,tree,
                   sem:v_sem).
pp              => f(phon:phon,syn:pre_n_syn,tree,sem:sem).
np_coord1       => f(phon:phon,syn:pre_n_syn,tree,sem:sem).
p_casep         => f(phon:phon,empty:yesno,syn:p_case_syn,tree,sem:np_sem).
pp_gen          => f(phon:phon,tree,sem:np_sem).
rel_phrase      => f(phon:phon,empty:yesno,syn:p_case_syn,tree,sem:np_sem).
adv_p           => f(phon:phon,syn:adv_syn,tree,sem:sem).
cop_p           => f(phon:phon,unary:yesno,syn:v_syn,tree,sem:v_sem).
copk            => f(phon:phon,syn:v_syn,tree,sem:v_sem).
vp              => f(phon:phon,unary:yesno,syn:v_syn,tree,sem:v_sem).
v1              => f(phon:phon,unary:yesno,syn:v_syn,tree,sem:v_sem).
vk1             => f(phon:phon,unary:yesno,syn:v_syn,tree,sem:v_sem).
vk              => f(phon:phon,syn:v_syn,tree,sem:v_sem).
v               => f(phon:phon,syn:v_syn,tree,sem:v_sem).
adresse         => f(phon:phon,tree,sem:np_sem).
adresse_germ    => f(phon:phon,tree,sem:np_sem).
name            => f(phon:phon,tree,sem:sem).
name_germ       => f(phon:phon,tree,sem:sem).
tel_num         => f(phon:phon,tree,sem:sem).
tel_num_jp      => f(phon:phon,tree,sem:sem).
price           => f(phon:phon,tree,sem:np_sem).
letter          => f(phon:phon,tree,sem:np_sem).
num_unit        => f(phon:phon,tree,sem:sem).
man_unit        => f(phon:phon,tree,sem:sem).
sen_unit        => f(phon:phon,tree,sem:sem).
hyaku_unit      => f(phon:phon,tree,sem:sem).
zyu_unit        => f(phon:phon,tree,sem:sem).
month_day       => f(phon:phon,tree,sem:np_sem).
month           => f(phon:phon,tree,sem:np_sem).
day             => f(phon:phon,tree,sem:np_sem).
day1_10         => f(phon:phon,syn:num_syn,tree,sem:np_sem).
day11_31        => f(phon:phon,syn:num_syn,tree,sem:np_sem).
edition         => f(phon:phon,tree,sem:np_sem).
np              => f(phon:phon,syn:pre_n_syn,tree,sem:np_sem).
np_coord        => f(phon:phon,syn:pre_n_syn,tree,sem:np_sem).
np_coord2       => f(phon:phon,syn:pre_n_syn,tree,sem:np_sem).
np1             => f(phon:phon,syn:pre_n_syn,tree,sem:np_sem).
n1              => f(phon:phon,syn:pre_n_syn,tree,sem:n1_sem).
v_stem          => f(phon:phon,syn:v_stem_syn,tree,sem:v_sem).
v_adj_stem      => f(phon:phon,syn:v_stem_syn,tree,sem:v_sem).
```

```
v_adj_ii        => f(phon:phon,syn:v_adj_ii_syn,tree,sem:v_sem).
v_sahen_stem    => f(phon:phon,syn:v_stem_syn,tree,sem:v_sem).
v_flex          => f(phon:phon,syn:v_flex_syn,tree).
v_adj_flex      => f(phon:phon,syn:v_flex_syn,tree).
v_ka_sahen      => f(phon:phon,syn:v_syn,tree,sem:v_sem).
v_neg           => f(phon:phon,syn:v_syn,tree,sem:v_sem).
v_gozaru        => f(phon:phon,syn:v_syn,tree,sem:v_sem).
v_aru           => f(phon:phon,syn:v_syn,tree,sem:v_sem).
aux             => f(phon:phon,syn:aux_syn,tree,sem:aux_sem).
att_p           => f(phon:phon,syn:aux_syn,tree,sem:aux_sem).
te              => f(phon:phon,syn:te_syn,tree).
h_aux           => f(phon:phon,syn:h_aux_syn,tree,sem:aux_sem).
h_adj           => f(phon:phon,syn:h_aux_syn,tree,sem:aux_sem).
h_adj_stem      => f(phon:phon,syn:h_aux_syn,tree,sem:aux_sem).
cop             => f(phon:phon,syn:v_syn,tree,sem:v_sem).
adv             => f(phon:phon,syn:adv_syn,tree,sem:sem).
conj            => f(phon:phon,syn:conj_syn,tree,sem:conj_sem).
no              => f(phon:phon,tree).
koord           => f(phon:phon,tree,sem:koord_sem).
pre             => f(phon:phon,syn:pre_syn,tree).
pre_num         => f(phon:phon,syn:pre_syn,tree).
s_prt           => f(phon:phon,syn:s_prt_syn,tree,sem:aux_sem).
n               => f(phon:phon,syn:n_syn,tree,sem:sem).
pre_n           => f(phon:phon,syn:pre_n_syn,tree,sem:sem).
suf_noun        => f(phon:phon,syn:suf_syn,tree,sem:aux_sem).
suf_wh          => f(phon:phon,syn:suf_syn,tree,sem:aux_sem).
suf_name        => f(phon:phon,syn:suf_syn,tree,sem:aux_sem).
adj_n           => f(phon:phon,syn:n_syn,tree,sem:sem).
pre_adj_n       => f(phon:phon,syn:pre_n_syn,tree,sem:sem).
pron            => f(phon:phon,syn:pre_n_syn,tree,sem:np_sem).
wh_pron         => f(phon:phon,syn:pre_n_syn,tree,sem:sem).
att             => f(phon:phon,tree,sem:sem).
quote_p         => f(phon:phon,syn:postp_syn,tree,sem:conj_sem).
postp           => f(phon:phon,syn:postp_syn,tree,sem:conj_sem).
p_case          => f(phon:phon,syn:p_case_syn,tree).
p_case1         => f(phon:phon,syn:p_case_syn,tree).
p_gen           => f(phon:phon,tree,sem:sem).
adr1            => f(phon:phon,tree,sem:sem).
adr2            => f(phon:phon,tree,sem:sem).
adr3            => f(phon:phon,tree,sem:sem).
adr_strasse     => f(phon:phon,tree,sem:sem).
adr_ort         => f(phon:phon,tree,sem:sem).
adr1_germ       => f(phon:phon,tree,sem:sem).
adr2_germ       => f(phon:phon,tree,sem:sem).
zahl            => f(phon:phon,tree,sem:sem).
zahl_germ       => f(phon:phon,tree,sem:sem).
zahl_plz        => f(phon:phon,tree,sem:sem).
zahl_pzb        => f(phon:phon,tree,sem:sem).
name1           => f(phon:phon,tree,sem:sem).
name2           => f(phon:phon,tree,sem:sem).
name1_germ1     => f(phon:phon,tree,sem:sem).
name2_germ2     => f(phon:phon,tree,sem:sem).
name1_germ      => f(phon:phon,tree,sem:sem).
name2_germ      => f(phon:phon,tree,sem:sem).
tel_num3_gen    => f(phon:phon,tree,sem:sem).
tel_num3        => f(phon:phon,tree,sem:sem).
tel_num4        => f(phon:phon,tree,sem:sem).
currency_jp     => f(phon:phon,tree,sem:aux_sem).
currency_dt     => f(phon:phon,tree,sem:aux_sem).
letter_zi       => f(phon:phon,tree,sem:aux_sem).
man_unit1       => f(phon:phon,syn:num_syn,tree,sem:sem).
sen_unit1       => f(phon:phon,syn:num_syn,tree,sem:sem).
hyaku_unit1     => f(phon:phon,syn:num_syn,tree,sem:sem).
zyu_unit1       => f(phon:phon,syn:num_syn,tree,sem:sem).
suf_mon         => f(phon:phon,tree,sem:aux_sem).
suf_day         => f(phon:phon,syn:num_syn,tree,sem:aux_sem).
suf_edit        => f(phon:phon,syn:num_syn,tree,sem:aux_sem).
num0_9          => f(phon:phon,syn:num_syn,tree,sem:sem).
```

```
num1_9          => f(phon:phon,syn:num_syn,tree,sem:sem).
num0            => f(phon:phon,syn:num_syn,tree,sem:sem).
num10_12        => f(phon:phon,syn:num_syn,tree,sem:sem).
num11_31        => f(phon:phon,syn:num_syn,tree,sem:sem).
num10           => f(phon:phon,syn:num_syn,tree,sem:sem).
num20           => f(phon:phon,syn:num_syn,tree,sem:sem).
num30_31        => f(phon:phon,syn:num_syn,tree,sem:sem).
interj          => f(phon:phon,tree,sem:sem).
idiom           => f(phon:phon,syn:adv_syn,tree,sem:sem).
plus            => f.
phon            => f(p1,p2).

v_syn           => f(st:stem_type,flex:flex,te,ad:adjacent_cond,sc:v_syn_sc,
                     pre:pre_v,topic,polite).
v_stem_syn      => f(st:stem_type,te,sc:v_syn_sc,pre:pre_v,topic,polite).
v_adj_ii_syn    => f(st:stem_type,flex:flex,te,sc:v_syn_sc,topic).
v_flex_syn      => f(st:stem_type,flex:flex,pre:pre_v,polite).
aux_syn         => f(st:stem_type,flex:flex,te,ad:adjacent_cond,inh:v_syn,
                     topic).
h_aux_syn       => f(st:stem_type,flex:flex,te,ad:adjacent_cond,inh:v_syn,topic).
conj_syn        => f(inh:v_syn,locate).
pre_syn         => f(form).
n_syn           => f(pre_form,adv_use,suf:suf_inh).
p_case_syn      => f(case,topic,empty).
postp_syn       => f(topic,copula,flex:flex,ad:adjacent_cond,adv_use,coord_use).
v_syn_sc        => f(arg1,arg2,arg3).
pre_n_syn       => f(polite,adv_use,suf:suf_inh,topic,copula,s_use,harm:harmony,
                     n_mod).
adjacent_cond   => f(te:te_iru,se:se_caus,att,neg,no_desu,h_verb,verb).
num_syn         => f(harm:harmony).
harmony         => f(neutral:man,voiceless:sen_zen,hpb:hyaku,gatu:month,niti:day,
                     zyu,nani).
suf_syn         => f(polite,adv_use,inh:suf_inh).
suf_inh         => f(full:suf_ippai,after:suf_igo,wh_ka:suf_ka).
te_syn          => f(te,topic).
adv_syn         => f(bare,topic,polite,n_mod).
s_prt_syn       => f(quote,inh:v_syn).
v_conj_syn      => f(locate).

sem             => f(expr,var).
v_sem           => f(expr,var,sc:v_sem_sc).
koord_sem       => f(expr).
np_sem          => f(expr).
aux_sem         => f(expr, expr2).
conj_sem        => f(expr,expr2,var).
v_sem_sc        => f(arg1,arg2,arg3).
n1_sem          => f(expr,var,gen,mod).

%Atomare Wertstrukturen
yesno           => {yes, no}.
form            => {o, go, nil}.
pre_form        => {o, go, nil}.
adv_use         => {yes, no}.
n_mod           => {yes, no}.
flex            => {mizen,mizen_o,mizen_caus,renyo,renyo_i,renyo_noun,renyo_adv,
                    syusi,rentai,katei,meirei,meirei_yo,meirei_pol}.
stem_type       => {godan_k,godan_k_iku,godan_g,godan_s,godan_t,godan_n,godan_b,
                    godan_m,godan_r,godan_w,itidan,kahen,sahen,pre_v_sahen,
                    v_sahen,keiyosi,aux_keiyosi,excep}.
te_iru          => {yes,no}.
se_caus         => {yes,no}.
pre_v           => {jp,pre_jp,sino_jp}.
bare            => {conj,bare,no_bare}.
te              => {te,de}.
man             => {yes,no}.
sen_zen         => {s,z,nil}.
hyaku           => {h,p,b,p_edit,nil}.
month           => {yes,no}.
day             => {n,k,nil}.
att             => {yes,no}.
neg             => {nai,masen}.
```

```
suf_ippai       => {yes,no}.
suf_igo         => {yes,no}.
suf_ka          => {yes,no}.
topic           => {wa,mo,nil}.
polite          => {polite,nonpolite,pejorativ}.
no_desu         => {yes,no}.
zyu             => {yes,no}.
nani            => {nani,nan}.
copula          => {yes,no}.
quote           => {direct,nondirect}.
locate          => {mid,fin,mid_fin}.
verb            => {yes,no}.
coord_use       => {yes,no}.
s_use           => {yes,no}.

% semantics
semantics(utt,utt:sem).
semantics(s4,s4:sem:expr).
semantics(s3,s3:sem:expr).
semantics(s2,s2:sem:expr).
semantics(s1,s1:sem:expr).
semantics(s0,s0:sem:expr).
semantics(s,s:sem:expr).
semantics(rel_s,rel_s:sem:expr).
semantics(pp,pp:sem:expr).
semantics(np_coord1,np_coord1:sem:expr).
semantics(quote_p,quote_p:sem:expr).
semantics(postp,postp:sem:expr).
semantics(p_casep,p_casep:sem:expr).
semantics(pp_gen,pp_gen:sem:expr).
semantics(p_gen,p_gen:sem:expr).
semantics(rel_phrase,rel_phrase:sem:expr).
semantics(adv_p,adv_p:sem:expr).
semantics(cop_p,cop_p:sem:expr).
semantics(copk,copk:sem:expr).
semantics(vp,vp:sem:expr).
semantics(v1,v1:sem:expr).
semantics(vk1,vk1:sem:expr).
semantics(vk,vk:sem:expr).
semantics(v,v:sem:expr).
semantics(adresse,adresse:sem:expr).
semantics(adresse_germ,adresse_germ:sem:expr).
semantics(name,name:sem:expr).
semantics(name_germ,name_germ:sem:expr).
semantics(tel_num ,tel_num:sem).
semantics(tel_num_jp,tel_num_jp:sem).
semantics(price,price:sem:expr).
semantics(letter,letter:sem:expr).
semantics(num_unit,num_unit:sem:expr).
semantics(man_unit,man_unit:sem:expr).
semantics(sen_unit,sen_unit:sem:expr).
semantics(hyaku_unit,hyaku_unit:sem:expr).
semantics(zyu_unit,zyu_unit:sem:expr).
semantics(month_day,month_day:sem:expr).
semantics(month,month:sem:expr).
semantics(day,day:sem:expr).
semantics(day1_10,day1_10:sem:expr).
semantics(day11_31,day11_31:sem:expr).
semantics(edition,edition:sem:expr).
semantics(np,np:sem:expr).
semantics(np_coord,np_coord:sem:expr).
semantics(np_coord2,np_coord2:sem:expr).
semantics(np1,np1:sem:expr).
semantics(n1,n1:sem:expr).
semantics(v_stem,v_stem:sem:expr).
semantics(v_adj_stem,v_adj_stem:sem:expr).
semantics(v_adj_ii,v_adj_ii:sem:expr).
semantics(v_sahen_stem,v_sahen_stem:sem:expr).
semantics(v_ka_sahen,v_ka_sahen:sem:expr).
```

```
semantics(v_neg,v_neg:sem:expr).
semantics(v_gozaru,v_gozaru:sem:expr).
semantics(v_aru,v_aru:sem:expr).
semantics(aux,aux:sem:expr).
semantics(att_p,att_p:sem:expr).
semantics(h_aux,h_aux:sem:expr).
semantics(h_adj,h_adj:sem:expr).
semantics(h_adj_stem,h_adj_stem:sem:expr).
semantics(cop,cop:sem:expr).
semantics(adv,adv:sem:expr).
semantics(conj,conj:sem:expr).
semantice(koord,koord:sem:expr).
semantics(s_prt,s_prt:sem:expr).
semantics(n,n:sem:expr).
semantics(pre_n,pre_n:sem:expr).
semantics(suf_noun,suf_noun:sem:expr).
semantics(suf_wh,suf_wh:sem:expr).
semantics(suf_name,suf_name:sem:expr).
semantics(adj_n,adj_n:sem:expr).
semantics(pre_adj_n,pre_adj_n:sem:expr).
semantics(pron,pron:sem:expr).
semantics(wh_pron,wh_pron:sem:expr).
semantics(att,att:sem:expr).
semantics(adr1,adr1:sem:expr).
semantics(adr2,adr2:sem:expr).
semantics(adr3,adr3:sem:expr).
semantics(adr_strasse,adr_strasse:sem:expr).
semantics(adr_ort,adr_ort:sem:expr).
semantics(adr1_germ,adr1_germ:sem:expr).
semantics(adr2_germ,adr2_germ:sem:expr).
semantics(zahl,zahl:sem:expr).
semantics(zahl_germ,zahl_germ:sem:expr).
semantics(zahl_plz,zahl_plz:sem:expr).
semantics(zahl_pzb,zahl_pzb:sem:expr).
semantics(name1,name1:sem:expr).
semantics(name2,name2:sem:expr).
semantics(name1_germ1,name1_germ1:sem).
semantics(name2_germ2,name2_germ2:sem).
semantics(name1_germ,name1_germ:sem:expr).
semantics(name2_germ,name2_germ:sem:expr).
semantics(tel_num3_gen,tel_num3_gen:sem:expr).
semantics(tel_num3,tel_num4:sem:expr).
semantics(tel_num4,tel_num4:sem:expr).
semantics(currency_jp,currency_jp:sem:expr).
semantics(currency_dt,currency_dt:sem:expr).
semantics(letter_zi,letter_zi:sem:expr).
semantics(suf_mon,suf_mon:sem:expr).
semantics(suf_day,suf_day:sem:expr).
semantics(suf_edit,suf_edit:sem:expr).
semantics(num0_9,num0_9:sem:expr).
semantics(num1_9,num1_9:sem:expr).
semantics(num0,num0:sem:expr).
semantics(num10_12,num10_12:sem:expr).
semantics(num11_31,num11_31:sem:expr).
semantics(num10,num10:sem:expr).
semantics(num20,num20:sem:expr).
semantics(num30_31,num30_31:sem:expr).
semantics(interj,interj:sem:expr).
semantics(idiom,idiom:sem:expr).
permutable(and).
```

C.2 Macro

```
sc(V,A1,A2,A3) short_for
        V:syn:sc:arg1 = A1,
   V:syn:sc:arg2 = A2,
   V:syn:sc:arg3 = A3.
```

```
godan(X) short_for
        X in {godan_k,godan_k_iku, godan_g,godan_s,godan_t,godan_n,godan_b,
              godan_m,godan_r,godan_w}.

phon_struc(M,D1,D2) short_for
        M:phon:p1 = '[D1:phon:p1|M:phon:p2],
        D1:phon:p2 = D2:phon:p1,
        D2:phon:p2 = [].

phon_struc(M,D1,D2,D3) short_for
        M:phon:p1 = '[D1:phon:p1|M:phon:p2],
        D1:phon:p2 = D2:phon:p1,
        D2:phon:p2 = D3:phon:p1,
        D3:phon:p2 = [].

phon_liste(M,D1,D2) short_for
        M:phon:p1 = D1:phon:p1,
        D1:phon:p2 = D2:phon:p1,
        M:phon:p2 = D2:phon:p2.

phon_liste(M,D1,D2,D3) short_for
        M:phon:p1 = D1:phon:p1,
        D1:phon:p2 = D2:phon:p1,
        D2:phon:p2 = D3:phon:p1,
        M:phon:p2 = D3:phon:p2.

phon_lex(C,V) short_for
        C:phon:p1 = '[V|C:phon:p2].
```

C.3 Rule

```
utt ---> s3 |
         utt:phon = s3:phon:p1,
         s3:phon:p2 = [],
         utt:sem = s3:sem:expr,
         utt:tree = 'utt(s3:tree).

utt ---> s2 |
         utt:phon = s2:phon:p1,
         s2:phon:p2 = [],
         utt:sem = s2:sem:expr,
         utt:tree = 'utt(s2:tree).

utt ---> s0 |
         utt:phon = s0:phon:p1,
         s0:phon:p2 = [],
         utt:sem = s0:sem:expr,
         utt:tree = 'utt(s0:tree).

utt ---> interj |
         utt:phon = interj:phon:p1,
         interj:phon:p2 = [],
         utt:sem = interj:sem:expr,
         utt:tree = 'utt(interj:tree).

s4 ---> s1, s:v |
        phon_liste(s4,s1,v),
        s4:sem:expr = 'and(v:sem:expr,s1:sem:expr),
        s1:sem:var = v:sem:var,
        s4:syn:flex = v:syn:flex,
        s4:syn:te = v:syn:te,
        s4:tree = 's4(s1:tree,v:tree).

s3 ---> s2, s_prt |
        phon_liste(s3,s2,s_prt),
        s3:sem:expr = s_prt:sem:expr,
        s2:sem:expr = s_prt:sem:expr2,
        s3:sem:var = s2:sem:var,
        s3:syn = s_prt:syn,
        s2:syn:flex = s_prt:syn:inh:flex,
        s3:tree = 's3(s2:tree,s_prt:tree).

s3 ---> s, s_prt |
        phon_liste(s3,s,s_prt),
        s3:sem:expr = s_prt:sem:expr,
        s:sem:expr = s_prt:sem:expr2,
        s3:sem:var = s:sem:var,
        s3:syn = s_prt:syn,
```

```
        s:syn:flex = s_prt:syn:inh:flex,
        s3:tree = 's3(s:tree,s_prt:tree).
s2 ---> s3, s |
        phon_liste(s2,s3,s),
        s2:sem:expr = 'and(s:sem:expr,s3:sem:expr),
        s3:sem:var = s:sem:var,
        s3:syn:quote = direct,
        s2:syn:flex = s:syn:flex,
        s:syn:flex in {syusi,meirei,meirei_yo,meirei_pol},
        s2:tree = 's2(s3:tree,s:tree).
s2 ---> s3, s1 |
        phon_liste(s2,s3,s1),
        s2:sem:expr = 'and(s1:sem:expr,s3:sem:expr),
        s3:sem:var = s1:sem:var,
        s3:syn:quote = direct,
        s1:syn:locate in {fin,mid_fin},
        s2:tree = 's2(s3:tree,s1:tree).
s2 ---> s1, s |
        phon_liste(s2,s1,s),
        s2:sem:expr = 'and(s:sem:expr,s1:sem:expr),
        s1:sem:var = s:sem:var,
        s2:syn:flex = s:syn:flex,
        s:syn:flex in {syusi,meirei,meirei_yo,meirei_pol},
        s2:tree = 's2(s1:tree,s:tree).
s2 ---> s1|
        s2:phon = s1:phon,
        s2:sem:expr = s1:sem:expr,
        s2:sem:var = s1:sem:var,
        s1:syn:locate in {fin,mid_fin},
        s2:tree = 's2(s1:tree).
s2 ---> s, s0 |
        phon_liste(s2,s,s0),
        s2:sem:expr = 'and(s:sem:expr,s0:sem:expr),
        s:sem:var = s0:sem:var,
        s2:syn:flex = s0:syn:flex,
        s:syn:flex = renyo,
        s2:tree = 's2(s:tree,s0:tree).
s1 ---> s, conj |
        phon_struc(s1,s,conj),
        s1:sem:var = conj:sem:var,
        s1:sem:expr = conj:sem:expr,
        conj:sem:expr2 = s:sem:expr,
        s1:syn:locate = conj:syn:locate,
        s:syn:flex = conj:syn:inh:flex,
        s:syn:te = conj:syn:inh:te,
        s1:tree = 's1(s:tree,conj:tree).
s1 ---> s4, conj |
        phon_struc(s1,s4,conj),
        s1:sem:var = conj:sem:var,
        s1:sem:expr = conj:sem:expr,
        conj:sem:expr2 = s4:sem:expr,
        s1:syn:locate = conj:syn:locate,
        s4:syn:flex = conj:syn:inh:flex,
        s4:syn:te = conj:syn:inh:te,
        s1:tree = 's1(s4:tree,conj:tree).
s1 ---> s, quote_p |
        phon_struc(s1,s,quote_p),
        s1:sem:var = quote_p:sem:var,
        s1:sem:expr = quote_p:sem:expr,
        quote_p:sem:expr2 = s:sem:expr,
        s:syn:flex = syusi,
        s:syn:topic = quote_p:syn:topic,
        s1:tree = 's1(s:tree,quote_p:tree).
s1 ---> np, quote_p |
        phon_struc(s1,np,quote_p),
        s1:sem:var = quote_p:sem:var,
        s1:sem:expr = quote_p:sem:expr,
        quote_p:sem:expr2 = np:sem:expr,
        s1:tree = 's1(np:tree,quote_p:tree).
```

```
s0 ---> s|
        s0:phon = s:phon,
        s0:sem:expr = s:sem:expr,
        s0:sem:var = s:sem:var,
        s0:syn = s:syn,
        s:syn:flex in {syusi,meirei,meirei_yo,meirei_pol},
        s0:tree = 's0(s:tree).

rel_s ---> rel_phrase<trace(ana,p_casep), s |
       phon_struc(rel_s,rel_phrase,s),
       rel_s:unary = rel_phrase:empty,
       rel_s:sem = s:sem,
       s:syn:flex = rentai,
       rel_s:topic:sem = rel_phrase:sem,
       rel_s:topic:syn = rel_phrase:syn,
       rel_phrase:syn = p_casep:syn,
       rel_phrase:sem = p_casep:sem,
       rel_s:tree = 'rel_s(rel_phrase:tree, s:tree).

rel_phrase ---> [] |
        rel_phrase:phon:p2 = '[rel_phrase:phon:p1],
        rel_phrase:empty = yes,
        rel_phrase:tree = 'rel_phrase(rel_phrase:syn:case,'ANTE').

s ---> p_casep, vp |
        phon_struc(s,p_casep,vp),
        s:unary = p_casep:empty,
        s:syn = vp:syn,
        s:syn:topic = vp:syn:topic,
        s:syn:flex = vp:syn:flex,
        vp:syn:sc:arg1 = p_casep:syn:case,
        s:sem = vp:sem,
        vp:sem:sc:arg1 = p_casep:sem:expr,
        s:tree = 's(p_casep:tree, vp:tree).

s ---> vp |
        s:phon = vp:phon,
        s:unary = yes,
        s:syn = vp:syn,
        s:syn:flex = vp:syn:flex,
        s:syn:sc:arg1 = nil,
        s:sem = vp:sem,
        s:tree = 's(vp:tree).

s ---> p_casep, cop_p |
        phon_struc(s,p_casep,cop_p),
        s:unary = p_casep:empty,
        s:syn:flex = cop_p:syn:flex,
        cop_p:syn:sc:arg1 = p_casep:syn:case,
        cop_p:sem:sc:arg1 = p_casep:sem:expr,
        s:sem = cop_p:sem,
        s:tree = 's(p_casep:tree, cop_p:tree).

s ---> cop_p |
        s:phon = cop_p:phon,
        s:unary = yes,
        s:syn = cop_p:syn,
        s:syn:flex = cop_p:syn:flex,
        s:syn:sc:arg1 = nil,
        s:sem = cop_p:sem,
        s:tree = 's(cop_p:tree).

s ---> adv_p, s:v |
        phon_struc(s,adv_p,v),
        v:unary = no,
        s:syn = v:syn,
        adv_p:syn:polite = v:syn:polite,
        s:sem:expr = 'and(adv_p:sem:expr,v:sem:expr),
        adv_p:sem:var = v:sem:var,
        s:sem:var = v:sem:var,
        s:tree = 's(adv_p:tree,v:tree).

s ---> pp, s:v |
        phon_struc(s,pp,v),
        v:unary = no,
        s:syn = v:syn,
        s:sem:expr = 'and(pp:sem:expr,v:sem:expr),
        pp:sem:var = v:sem:var,
        s:sem:var = v:sem:var,
        s:tree = 's(pp:tree,v:tree).

cop_p ---> adv_p, cop_p:v |
```

```
        phon_struc(cop_p,adv_p,v),
        v:unary = no,
        cop_p:syn = v:syn,
        adv_p:syn:polite = v:syn:polite,
        cop_p:sem:sc = v:sem:sc,
        cop_p:sem:expr = 'and(adv_p:sem:expr,v:sem:expr),
        adv_p:sem:var = v:sem:var,
        cop_p:sem:var = v:sem:var,
        cop_p:tree = 'cop_p(adv_p:tree,v:tree).

cop_p ---> pp, cop_p:v |
        phon_struc(cop_p,pp,v),
        v:unary = no,
        cop_p:syn = v:syn,
        cop_p:sem:sc = v:sem:sc,
        cop_p:sem:expr = 'and(pp:sem:expr,v:sem:expr),
        pp:sem:var = v:sem:var,
        cop_p:sem:var = v:sem:var,
        cop_p:tree = 'cop_p(pp:tree,v:tree).

cop_p ---> adv, copk |
        phon_struc(cop_p,adv,copk),
        cop_p:syn = copk:syn,
        copk:syn:sc:arg2 = adv,
        adv:syn:bare in {bare,no_bare},
        adv:syn:polite = copk:syn:polite,
        cop_p:sem = copk:sem,
        adv:sem:var = cop_p:sem:sc:arg1,
        cop_p:sem:sc:arg2 = adv:sem:expr,
        cop_p:tree = 'cop_p(adv:tree, copk:tree).

cop_p ---> np, copk |
        phon_struc(cop_p,np,copk),
        cop_p:syn = copk:syn,
        copk:syn:sc:arg2 = np,
        np:syn:harm:nani = nan,
        cop_p:sem = copk:sem,
        cop_p:sem:sc:arg2 = np:sem:expr,
        cop_p:tree = 'cop_p(np:tree, copk:tree).

cop_p ---> pp, copk |
        phon_struc(cop_p,pp,copk),
        cop_p:syn = copk:syn,
        pp:syn:copula = yes,
        copk:syn:sc:arg2 = np,
        cop_p:sem = copk:sem,
        cop_p:sem:sc:arg2 = pp:sem:expr,
        cop_p:tree = 'cop_p(pp:tree, copk:tree).

cop_p ---> pre_adj_n, copk |
        phon_struc(cop_p,pre_adj_n,copk),
        cop_p:syn = copk:syn,
        copk:syn:sc:arg2 = np,
        cop_p:sem = copk:sem,
        cop_p:sem:sc:arg2 = pre_adj_n:sem:expr,
        cop_p:tree = 'cop_p(pre_adj_n:tree, copk:tree).

cop_p ---> adresse, name, copk |
        phon_struc(cop_p,adresse,name,copk),
        cop_p:syn = copk:syn,
        copk:syn:sc:arg2 = np,
        cop_p:sem = copk:sem,
        cop_p:sem:sc:arg2 = 'cop_p(adresse:sem,name:sem),
        cop_p:tree = 'cop_p(adresse:tree, name:tree, copk:tree).

cop_p ---> tel_num, copk |
        phon_struc(cop_p,tel_num,copk),
        cop_p:syn = copk:syn,
        copk:syn:sc:arg2 = np,
        cop_p:sem = copk:sem,
        cop_p:sem:sc:arg2 = tel_num:sem,
        cop_p:tree = 'cop_p(tel_num:tree, copk:tree).

copk ---> cop |
        copk:phon = cop:phon,
        copk:syn = cop:syn,
        copk:sem = cop:sem,
        copk:tree = 'copk(cop:tree).

copk ---> copk:a, aux |
```

```
        phon_liste(copk,a,aux),
        copk:sem:expr = aux:sem:expr,
        a:sem:expr = aux:sem:expr2,
        copk:sem:var = a:sem:var,
        copk:sem:sc = a:sem:sc,
        copk:syn:sc = a:syn:sc,
        copk:syn:st = aux:syn:st,
        copk:syn:flex = aux:syn:flex,
        a:syn = aux:syn:inh,
        copk:tree = 'copk(a:tree, aux:tree).
copk ---> copk:a, h_aux |
        phon_liste(copk,a,h_aux),
        copk:sem:expr = h_aux:sem:expr,
        a:sem:expr = h_aux:sem:expr2,
        copk:sem:var = a:sem:var,
        copk:sem:sc = a:sem:sc,
        copk:syn:sc = a:syn:sc,
        a:syn:flex = h_aux:syn:inh:flex,
        copk:tree = 'copk(a:tree, h_aux:tree).
copk ---> copk:a, no, aux |
        phon_liste(copk,a,no,aux),
        copk:syn:sc = a:syn:sc,
        copk:syn:st = aux:syn:st,
        copk:syn:flex = aux:syn:flex,
        aux:syn:ad:no_desu = yes,
        aux:syn:inh:st = excep,
        a:syn:flex = rentai,
        copk:sem:expr = aux:sem:expr,
        a:sem:expr = aux:sem:expr2,
        copk:sem:var = a:sem:var,
        copk:sem:sc = a:sem:sc,
        copk:tree = 'copk(a:tree, no:tree, aux:tree).
adv_p ---> adv |
        adv_p:phon:p1 = '[adv:phon:p1|adv_p:phon:p2],
        adv:phon:p2 = [],
        adv:syn:bare in {conj,bare},
        adv_p:syn:polite = adv:syn:polite,
        adv_p:sem = adv:sem,
        adv_p:tree = 'adv_p(adv:tree).
adv_p ---> np |
        adv_p:phon:p1 = '[np:phon:p1|adv_p:phon:p2],
        np:phon:p2 = [],
        np:syn:adv_use = yes,
        adv_p:sem:expr = np:sem:expr,
        adv_p:tree = 'adv_p(np:tree).
adv_p ---> pre_adj_n, postp |
        phon_struc(adv_p,pre_adj_n,postp),
        postp:syn:adv_use = yes,
        pre_adj_n:syn:adv_use = yes,
        pre_adj_n:syn:topic = postp:syn:topic,
        adv_p:sem:expr = postp:sem:expr,
        adv_p:sem:var = postp:sem:var,
        postp:sem:expr2 = pre_adj_n:sem:expr,
        adv_p:tree = 'adv_p(pre_adj_n:tree,postp:tree).
adv_p ---> att_p |
        adv_p:phon:p1 = '[att_p:phon:p1|adv_p:phon:p2],
        att_p:phon:p2 = [],
        att_p:syn:flex = renyo_adv,
        adv_p:sem:expr = att_p:sem:expr,
        adv_p:tree = 'adv_p(att_p:tree).
adv_p ---> idiom |
        adv_p:phon:p1 = '[idiom:phon:p1|adv_p:phon:p2],
        idiom:phon:p2 = [],
        adv_p:sem:expr = idiom:sem:expr,
        adv_p:tree = 'adv_p(idiom:tree).
vp ---> p_casep, v1 |
        phon_struc(vp,p_casep,v1),
        vp:unary = p_casep:empty,
        vp:syn = v1:syn,
```

```
        v1:syn:sc:arg2 = p_casep:syn:case,
        vp:sem = v1:sem,
        vp:sem:sc:arg2 = p_casep:sem:expr,
        vp:tree = 'vp(p_casep:tree, v1:tree).
vp ---> v1 |
        vp:phon = v1:phon,
        vp:unary = yes,
        vp:syn = v1:syn,
        v1:syn:sc:arg2 = nil,
        vp:sem = v1:sem,
        vp:tree = 'vp(v1:tree).
vp ---> adv_p, vp:v |
        phon_struc(vp,adv_p,v),
        v:unary = no,
        vp:syn = v:syn,
        vp:sem:sc = v:sem:sc,
        vp:sem:expr = 'and(adv_p:sem:expr,v:sem:expr),
        adv_p:sem:var = v:sem:var,
        vp:sem:var = v:sem:var,
        vp:tree = 'vp(adv_p:tree,v:tree).
vp ---> pp, vp:v |
        phon_struc(vp,pp,v),
        v:unary = no,
        vp:syn = v:syn,
        vp:sem:sc = v:sem:sc,
        vp:sem:expr = 'and(pp:sem:expr,v:sem:expr),
        pp:sem:var = v:sem:var,
        vp:sem:var = v:sem:var,
        vp:tree = 'vp(pp:tree,v:tree).
v1 ---> p_casep, vk1 |
        phon_struc(v1,p_casep,vk1),
        v1:unary = p_casep:empty,
        v1:syn = vk1:syn,
        vk1:syn:sc:arg3 = p_casep:syn:case,
        v1:sem = vk1:sem,
        v1:sem:sc:arg3 = p_casep:sem:expr,
        v1:tree = 'v1(p_casep:tree, vk1:tree).
v1 ---> vk1 |
        v1:phon = vk1:phon,
        v1:unary = yes,
        v1:syn = vk1:syn,
        vk1:syn:sc:arg3 = nil,
        v1:sem = vk1:sem,
        v1:tree = 'v1(vk1:tree).
v1 ---> adv_p, v1:v |
        phon_struc(v1,adv_p,v),
        v:unary = no,
        v1:syn = v:syn,
        v1:sem:sc = v:sem:sc,
        v1:sem:expr = 'and(adv_p:sem:expr,v:sem:expr),
        adv_p:sem:var = v:sem:var,
        v1:sem:var = v:sem:var,
        v1:tree = 'v1(adv_p:tree,v:tree).
v1 ---> pp, v1:v |
        phon_struc(v1,pp,v),
        v:unary = no,
        v1:syn = v:syn,
        v1:sem:sc = v:sem:sc,
        v1:sem:expr = 'and(pp:sem:expr,v:sem:expr),
        pp:sem:var = v:sem:var,
        v1:sem:var = v:sem:var,
        v1:tree = 'v1(pp:tree,v:tree).
vk1 ---> vk |
        vk1:phon:p1 = '[vk:phon:p1|vk1:phon:p2],
        vk:phon:p2 = [],
        vk1:syn = vk:syn,
        vk1:sem = vk:sem,
        vk1:tree = 'vk1(vk:tree).
vk1 ---> adv_p, vk1:v |
        phon_struc(vk1,adv_p,v),
        vk1:syn = v:syn,
        vk1:sem:sc = v:sem:sc,
```

```
        vk1:sem:expr = 'and(adv_p:sem:expr,v:sem:expr),
        adv_p:sem:var = v:sem:var,
        vk1:sem:var = v:sem:var,
        vk1:tree = 'vk1(adv_p:tree,v:tree).

vk1 ---> pp, vk1:v |
        phon_struc(vk1,pp,v),
        vk1:syn = v:syn,
        vk1:sem:sc = v:sem:sc,
        vk1:sem:expr = 'and(pp:sem:expr,v:sem:expr),
        vk1:tree = 'vk1(pp:tree,v:tree).

vk1 ---> adv, vk1:v |
        phon_struc(vk1,adv,v),
        vk1:syn:st = v:syn:st,
        vk1:syn:flex = v:syn:flex,
        vk1:syn:sc = v:syn:sc,
        vk1:syn:ad = v:syn:ad,
        vk1:syn:te = v:syn:te,
        adv:syn:bare = no_bare,
        vk1:sem:sc = v:sem:sc,
        vk1:sem:expr = 'and(adv:sem:expr,v:sem:expr),
        adv:sem:var = v:sem:var,
        vk1:sem:var = v:sem:var,
        vk1:tree = 'vk1(adv:tree,v:tree).

vk ---> v |
        vk:phon = v:phon,
        vk:syn = v:syn,
        vk:syn:topic = v:syn:topic,
        vk:syn:te = v:syn:te,
        vk:sem = v:sem,
        vk:tree = 'vk(v:tree).

vk ---> vk:a, aux |
        phon_liste(vk,a,aux),
        vk:syn:sc = a:syn:sc,
        vk:syn:st = aux:syn:st,
        vk:syn:flex = aux:syn:flex,
        vk:syn:topic = aux:syn:topic,
        a:syn = aux:syn:inh,
        vk:syn:ad = aux:syn:ad,
        a:syn:ad:neg = aux:syn:ad:neg,
        vk:syn:te = aux:syn:te,
        vk:sem:expr = aux:sem:expr,
        a:sem:expr = aux:sem:expr2,
        vk:sem:var = a:sem:var,
        vk:sem:sc = a:sem:sc,
        vk:tree = 'vk(a:tree, aux:tree).

vk ---> vk:a, no, aux |
        phon_liste(vk,a,no,aux),
        vk:syn:sc = a:syn:sc,
        vk:syn:st = aux:syn:st,
        vk:syn:flex = aux:syn:flex,
        vk:syn:topic = aux:syn:topic,
        vk:syn:te = aux:syn:te,
        aux:syn:ad:no_desu = yes,
        aux:syn:inh:st = nil,
        a:syn:st in {godan_k,godan_k_iku,godan_g,godan_s,godan_t,godan_n,
                     godan_b,godan_m,godan_r,godan_w,itidan,kahen,sahen,
                     v_sahen,keiyosi,aux_keiyosi,excep},
        a:syn:flex = rentai,
        vk:sem:expr = aux:sem:expr,
        a:sem:expr = aux:sem:expr2,
        vk:sem:var = a:sem:var,
        vk:sem:sc = a:sem:sc,
        vk:tree = 'vk(a:tree, no:tree, aux:tree).

vk ---> vk:a, h_aux |
        phon_liste(vk,a,h_aux),
        (a:syn:st = pre_v_sahen
         ;a:syn:st = v_sahen, a:syn:pre = sino_jp),
        a:syn:flex = renyo_noun,
        vk:syn:sc = a:syn:sc,
        h_aux:syn:ad:h_verb = yes,
        vk:syn:st = h_aux:syn:st,
        vk:syn:flex = h_aux:syn:flex,
```

```
        a:syn:topic = nil,
        vk:syn:topic = h_aux:syn:topic,
        a:syn:pre = h_aux:syn:inh:pre,
        a:syn:ad:se = h_aux:syn:ad:se,
        vk:syn:ad = h_aux:syn:ad,
        vk:syn:te = h_aux:syn:te,
        vk:sem:expr = h_aux:sem:expr,
        a:sem:expr = h_aux:sem:expr2,
        vk:sem:var = a:sem:var,
        vk:sem:sc = a:sem:sc,
        vk:tree = 'vk(a:tree, h_aux:tree).
vk ---> vk:a, te, h_aux |
        phon_liste(vk,a,te,h_aux),
        h_aux:syn:ad:te = yes,
        a:syn:topic = te:syn:topic,
        vk:syn:topic = h_aux:syn:topic,
        vk:sem:expr = h_aux:sem:expr,
        a:sem:expr = h_aux:sem:expr2,
        vk:sem:var = a:sem:var,
        vk:sem:sc = a:sem:sc,
        (a:syn:st in {keiyosi,aux_keiyosi}, a:syn:flex = mizen
         ;godan(a:syn:st), a:syn:flex = renyo_i
         ;a:syn:st in {itidan,kahen,sahen,v_sahen,excep}, a:syn:flex = renyo_i
         ;a:syn:st = pre_v_sahen, h_aux:syn:ad:h_verb = no, a:syn:flex = renyo
         ;a:syn:flex = nil),
        a:syn:te = te:syn:te,
        vk:syn:sc = a:syn:sc,
        vk:syn:st = h_aux:syn:st,
        vk:syn:flex = h_aux:syn:flex,
        vk:syn:te = h_aux:syn:te,
        vk:tree = 'vk(a:tree, te:tree, h_aux:tree).
vk ---> vk:a, te, h_adj |
        phon_liste(vk,a,te,h_adj),
        a:syn:topic = te:syn:topic,
        vk:syn:topic = h_adj:syn:topic,
        vk:sem:expr = h_adj:sem:expr,
        a:sem:expr = h_adj:sem:expr2,
        vk:sem:var = a:sem:var,
        vk:sem:sc = a:sem:sc,
        (a:syn:st in {keiyosi,aux_keiyosi}, a:syn:flex = mizen
         ;godan(a:syn:st), a:syn:flex = renyo_i
         ;a:syn:st in {itidan,kahen,sahen,v_sahen,excep},a:syn:flex = renyo_i),
        a:syn:te = te:syn:te,
        vk:syn:sc = a:syn:sc,
        vk:syn:st = h_adj:syn:st,
        vk:syn:flex = h_adj:syn:flex,
        vk:syn:te = h_adj:syn:te,
        vk:tree = 'vk(a:tree, te:tree, h_adj:tree).
p_casep ---> np, p_case1 |
        phon_struc(p_casep,np,p_case1),
        p_casep:empty = no,
        p_casep:syn = p_case1:syn,
        np:syn:topic = p_case1:syn:topic,
        (p_case1:syn:empty = yes, np:syn:s_use = yes
         ;p_case1:syn:empty = no),
        p_casep:sem = np:sem,
        p_casep:tree = 'p_casep(np:tree, p_case1:tree).
p_casep ---> [] |
        p_casep:phon:p2 = '[p_casep:phon:p1],
        p_casep:empty = yes,
        p_casep:syn:topic = nil,
        p_casep:syn:case in {ga,ni,o},
        p_casep:sem:expr = 'quant(syn(pro),ellipsis,X,
                                  restr(personal(X),[],[]),_),
        p_casep:tree = 'p_casep(p_casep:syn:case,'PRO').
p_case1 ---> p_case |
        p_case1:phon = p_case:phon,
        p_case1:syn:topic = p_case:syn:topic,
        p_case1:syn:case = p_case:syn:case,
        p_case1:syn:empty = no,
        p_case1:tree = 'p_case1(p_case:tree).
```

```
p_case1 ---> [] |
        p_case1:phon:p2 = p_case1:phon:p1,
        p_case1:syn:topic = nil,
        p_case1:syn:case in {ga,o,ni},
        p_case1:syn:empty = yes,
        p_case1:tree = 'p_case1(p_case1:syn:case,'DROP').
pp ---> np, postp |
        phon_struc(pp,np,postp),
        pp:syn = np:syn,
        np:syn:topic = postp:syn:topic,
        np:syn:copula = postp:syn:copula,
        pp:sem:expr = postp:sem:expr,
        pp:sem:var = postp:sem:var,
        postp:sem:expr2 = np:sem:expr,
        pp:tree = 'pp(np:tree, postp:tree).
pp ---> s, postp |
        phon_struc(pp,s,postp),
        pp:syn:polite = s:syn:polite,
        s:syn:topic = postp:syn:topic,
        postp:syn:ad:verb = yes,
        s:syn:flex = postp:syn:flex,
        pp:sem:expr = postp:sem:expr,
        pp:sem:var = postp:sem:var,
        postp:sem:expr2 = s:sem:expr,
        pp:tree = 'pp(s:tree, postp:tree).
pp_gen ---> np, p_gen |
        phon_struc(pp_gen,np,p_gen),
        np:syn:harm:nani = nan,
        pp_gen:sem:expr = p_gen:sem:expr,
        p_gen:sem:var = np:sem:expr,
        pp_gen:tree = 'pp_gen(np:tree, p_gen:tree).
/*
quant(<SynFeatures>,<QuantifierHead>,<Variable>,
   restr(<HeadRestriction>,<Modifiers>,<Genitives>),
   <Scope>)
*/
np ---> vk |
        np:phon = vk:phon,
        vk:syn:st = v_sahen,
        vk:syn:flex = renyo_noun,
        vk:syn:pre = sino_jp,
        np:syn:adv_use = no,
        np:syn:topic = vk:syn:topic,
        np:syn:s_use = yes,
        np:sem:expr = vk:sem:expr,
        np:tree = 'np(vk:tree).
np ---> np1 |
        np:phon = np1:phon,
        np:syn:polite = np1:syn:polite,
        np:syn:adv_use = np1:syn:adv_use,
        np:syn:topic = np1:syn:topic,
        np:syn:s_use = np1:syn:s_use,
        np:syn:harm = np1:syn:harm,
        np:sem = np1:sem,
        np:tree = 'np(np1:tree).
np ---> np_coord |
        np:phon = np_coord:phon,
        np:syn:topic = np_coord:syn:topic,
        np:syn:s_use = no,
        np:sem:expr = np_coord:sem:expr,
        np:tree = 'np(np_coord:tree).
np ---> np_coord1 |
        np:phon = np_coord1:phon,
        np:syn:topic = np_coord1:syn:topic,
        np:syn:copula = np_coord1:syn:copula,
        np:syn:s_use = no,
        np:sem:expr = np_coord1:sem:expr,
        np:tree = 'np(np_coord1:tree).
np1 ---> np1:a, suf_noun |
        phon_liste(np1,a,suf_noun),
        a:syn:polite = suf_noun:syn:polite,
```

```
        np1:syn:polite = a:syn:polite,
        np1:syn:adv_use = suf_noun:syn:adv_use,
        np1:syn:topic = a:syn:topic,
        np1:syn:s_use = no,
        a:syn:suf = suf_noun:syn:inh,
        np1:sem:expr = suf_noun:sem:expr,
        a:sem:expr = suf_noun:sem:expr2,
        np1:tree = 'np1(a:tree, suf_noun:tree).
np1 ---> np1:a, suf_wh |
        phon_liste(np1,a,suf_wh),
        a:syn:polite = suf_wh:syn:polite,
        np1:syn:polite = a:syn:polite,
        np1:syn:adv_use = suf_wh:syn:adv_use,
        np1:syn:s_use = yes,
        np1:syn:topic = a:syn:topic,
        a:syn:suf = suf_wh:syn:inh,
        np1:sem:expr = suf_wh:sem:expr,
        a:sem:expr = suf_wh:sem:expr2,
        np1:tree = 'np1(a:tree, suf_wh:tree).
np1 ---> n1 |
        np1:phon = n1:phon,
        np1:syn = n1:syn,
        np1:sem:expr = 'quant(syn(bare,n1:syn:polite,n1:syn:topic),_,
                              n1:sem:var,n1:sem:expr,_),
        n1:sem:gen = [],
        n1:sem:mod = [],
        np1:tree = 'np1(n1:tree).
n1 ---> rel_s, n1:h |
        phon_struc(n1,rel_s,h),
        n1:syn = h:syn,
        rel_s:topic:sem:expr = h:sem:var,
        h:sem:expr = n1:sem:expr,
        h:sem:mod = '[rel_s:sem:expr|n1:sem:mod],
        n1:sem:gen = h:sem:gen,
        n1:tree = 'n1(rel_s:tree, h:tree).
n1 ---> pp_gen, n1:h |
        phon_liste(n1,pp_gen,h),
        n1:syn = h:syn,
        n1:syn:topic = h:syn:topic,
        h:sem:expr = n1:sem:expr,
        h:sem:gen = '[pp_gen:sem:expr|n1:sem:gen],
        n1:sem:mod = h:sem:mod,
        n1:tree = 'n1(pp_gen:tree, h:tree).
n1 ---> pre_n |
        n1:phon = pre_n:phon,
        n1:syn = pre_n:syn,
        n1:sem:expr = 'restr(pre_n:sem:expr,n1:sem:gen,n1:sem:mod),
        n1:tree = 'n1(pre_n:tree).
n1 ---> name |
        n1:phon = name:phon,
        n1:syn:polite = nonpolite,
        n1:syn:adv_use = no,
        n1:syn:topic = nil,
        n1:syn:n_mod = no,
        n1:sem:expr = 'restr(name:sem:expr,n1:sem:gen,n1:sem:mod),
        n1:tree = 'n1(name:tree).
n1 ---> adresse |
        n1:phon = adresse:phon,
        n1:syn:polite = nonpolite,
        n1:syn:adv_use = no,
        n1:syn:topic = nil,
        n1:syn:n_mod = no,
        n1:sem:expr = 'restr(adresse:sem,n1:sem:gen,n1:sem:mod),
        n1:tree = 'n1(adresse:tree).
n1 ---> adv, n1:h |
        phon_struc(n1,adv,h),
        n1:syn = h:syn,
        n1:syn:topic = h:syn:topic,
        adv:syn:n_mod = yes,
        h:syn:n_mod = yes,
```

```
        h:sem:expr = n1:sem:expr,
        h:sem:mod = '[adv:sem:expr|n1:sem:mod],
        n1:sem:gen = h:sem:gen,
        n1:tree = 'n1(adv:tree, h:tree).
n1 ---> att, n1:h |
        phon_struc(n1,att,h),
        n1:syn = h:syn,
        n1:syn:topic = h:syn:topic,
        h:sem:expr = n1:sem:expr,
        h:sem:mod = '[att:sem:expr|n1:sem:mod],
        n1:sem:gen = h:sem:gen,
        n1:tree = 'n1(att:tree, h:tree).
n1 ---> att_p, n1:h |
        phon_struc(n1,att_p,h),
        att_p:syn:flex = rentai,
        att_p:syn:ad:att = yes,
        n1:syn = h:syn,
        n1:syn:topic = h:syn:topic,
        h:sem:expr = n1:sem:expr,
        h:sem:mod = '[att_p:sem:expr|n1:sem:mod],
        n1:sem:gen = h:sem:gen,
        n1:tree = 'n1(att_p:tree, h:tree).
pre_n ---> wh_pron |
        pre_n:phon = wh_pron:phon,
        pre_n:syn = wh_pron:syn,
        pre_n:syn:suf:wh_ka = yes,
        pre_n:syn:n_mod = no,
        pre_n:sem:expr = wh_pron:sem:expr,
        pre_n:tree = 'pre_n(wh_pron:tree).
pre_n ---> pron |
        pre_n:phon = pron:phon,
        pre_n:syn = pron:syn,
        pre_n:syn:topic = pron:syn:topic,
        pre_n:syn:adv_use = yes,
        pre_n:syn:suf:wh_ka = no,
        pre_n:syn:n_mod = no,
        pre_n:sem:expr = pron:sem:expr,
        pre_n:tree = 'pre_n(pron:tree).
pre_n ---> month_day |
        pre_n:phon = month_day:phon,
        pre_n:syn:adv_use = yes,
        pre_n:syn:suf:wh_ka = no,
        pre_n:syn:n_mod = no,
        pre_n:sem:expr = month_day:sem:expr,
        pre_n:tree = 'pre_n(month_day:tree).
pre_n ---> day |
        pre_n:phon = day:phon,
        pre_n:syn:suf:wh_ka = no,
        pre_n:syn:n_mod = no,
        pre_n:sem:expr = day:sem:expr,
        pre_n:tree = 'pre_n(day:tree).
pre_n ---> price |
        pre_n:phon = price:phon,
        pre_n:syn:suf:wh_ka = no,
        pre_n:syn:n_mod = yes,
        pre_n:sem:expr = price:sem:expr,
        pre_n:tree = 'pre_n(price:tree).
pre_n ---> letter |
        pre_n:phon = letter:phon,
        pre_n:syn:suf:wh_ka = no,
        pre_n:syn:n_mod = no,
        pre_n:sem:expr = letter:sem:expr,
        pre_n:tree = 'pre_n(letter:tree).
pre_n ---> pre_num, edition |
        phon_struc(pre_n,pre_num,edition),
        pre_n:syn:suf:wh_ka = no,
        pre_n:syn:n_mod = no,
        pre_n:sem:expr = edition:sem:expr,
        pre_n:tree = 'pre_n(pre_num:tree, edition:tree).
```

```
pre_n ---> pre, n |
        phon_liste(pre_n,pre,n),
        pre:syn:form = n:syn:pre_form,
        pre_n:syn:adv_use = n:syn:adv_use,
        pre_n:syn:s_use = no,
        pre_n:syn:polite = polite,
        pre_n:syn:suf = n:syn:suf,
        pre_n:syn:suf:wh_ka = no,
        pre_n:syn:n_mod = no,
        pre_n:sem = n:sem,
        pre_n:tree = 'pre_n(pre:tree, n:tree).
pre_n ---> n |
        pre_n:phon = n:phon,
        pre_n:syn:adv_use = n:syn:adv_use,
        pre_n:syn:s_use = no,
        pre_n:syn:polite = nonpolite,
        pre_n:syn:suf = n:syn:suf,
        pre_n:syn:suf:wh_ka = no,
        pre_n:syn:n_mod = no,
        pre_n:sem = n:sem,
        pre_n:tree = 'pre_n(n:tree).
np_coord ---> np:a, koord, np:b |
        phon_struc(np_coord,a,koord,b),
        np_coord:syn:topic = b:syn:topic,
        np_coord:sem:expr = 'coord(koord:sem:expr,a:sem:expr,b:sem:expr),
        np_coord:tree = 'np_coord(a:tree, koord:tree, b:tree).
np_coord1 ---> np_coord2, postp |
        phon_struc(np_coord1,np_coord2,postp),
        np_coord1:syn:topic = np_coord2:syn:topic,
        np_coord1:syn:copula = postp:syn:copula,
        postp:syn:coord_use = yes,
        np_coord1:sem:expr = 'coord(postp:sem:expr,np_coord2:sem:expr),
        np_coord1:tree = 'np_coord1(np_coord2:tree,postp:tree).
np_coord2 ---> np, np_coord2:a |
        phon_liste(np_coord2,np,a),
        np_coord2:syn:topic = a:syn:topic,
        np_coord2:sem:expr = 'coord(np:sem:expr,a:sem:expr),
        np_coord2:tree = 'np_coord2(np:tree,a:tree).
np_coord2 ---> np |
        np_coord2:phon = np:phon,
        np_coord2:syn = np:syn,
        np_coord2:sem:expr = np:sem:expr,
        np_coord2:tree = 'np_coord2(np:tree).
pre_adj_n ---> pre, adj_n |
        phon_liste(pre_adj_n,pre,adj_n),
        pre:syn:form = adj_n:syn:pre_form,
        pre_adj_n:syn:polite = polite,
        pre_adj_n:sem:expr = 'quant(syn(pre_adj_n:syn:polite,
                                        pre_adj_n:syn:topic),_,
                                adj_n:sem:var,
                                adj_n:sem:expr,
                                _),
        pre_adj_n:tree = 'pre_adj_n(pre:tree, adj_n:tree).
pre_adj_n ---> adj_n |
        pre_adj_n:phon = adj_n:phon,
        pre_adj_n:syn:polite = nonpolite,
        pre_adj_n:sem:expr = 'quant(syn(pre_adj_n:syn:polite,
                                        pre_adj_n:syn:topic),_,
                                    adj_n:sem:var,adj_n:sem:expr,_),
        pre_adj_n:tree = 'pre_adj_n(adj_n:tree).
att_p ---> att, aux |
        phon_struc(att_p,att,aux),
        att_p:syn:flex = aux:syn:flex,
        att_p:syn:ad:att = aux:syn:ad:att,
        att_p:sem:expr = 'att_p(att:sem:expr, aux:sem:expr),
        att_p:tree = 'att_p(att:tree, aux:tree).
v ---> v_stem, plus, v_flex |
        phon_liste(v,v_stem,v_flex),
        v_stem:syn:st = v_flex:syn:st,
```

```
        v:syn:st = v_stem:syn:st,
        v:syn:flex = v_flex:syn:flex,
        v:syn:sc = v_stem:syn:sc,
        v:syn:pre = v_stem:syn:pre,
        v:syn:te = v_stem:syn:te,
        v:syn:topic = v_stem:syn:topic,
        v:sem = v_stem:sem,
        v:tree = 'v(v_stem:tree,plus,v_flex:tree).
v ---> v_stem |
        v:phon = v_stem:phon,
        v_stem:syn:st = itidan,
        v:syn:st = v_stem:syn:st,
        v:syn:flex in {mizen,mizen_o,mizen_caus,renyo,renyo_i},
        v:syn:sc = v_stem:syn:sc,
        v:syn:pre = v_stem:syn:pre,
        v:syn:te = v_stem:syn:te,
        v:syn:topic = v_stem:syn:topic,
        v:sem = v_stem:sem,
        v:tree = 'v(v_stem:tree).
v ---> v_sahen_stem, plus, v_flex |
        phon_liste(v,v_sahen_stem,v_flex),
        v_sahen_stem:syn:st = v_flex:syn:st,
        v_sahen_stem:syn:pre = v_flex:syn:pre,
        v_sahen_stem:syn:polite =  v_flex:syn:polite,
        v:syn:st = v_sahen_stem:syn:st,
        v:syn:flex = v_flex:syn:flex,
        v:syn:sc = v_sahen_stem:syn:sc,
        v:syn:pre = v_sahen_stem:syn:pre,
        v:syn:te = v_sahen_stem:syn:te,
        v:syn:topic = v_sahen_stem:syn:topic,
        v:sem = v_sahen_stem:sem,
        v:tree = 'v(v_sahen_stem:tree,plus,v_flex:tree).
v ---> v_sahen_stem |
        v:phon = v_sahen_stem:phon,
        v_sahen_stem:syn:st in {pre_v_sahen,v_sahen},
        v:syn:st = v_sahen_stem:syn:st,
        v:syn:flex = renyo_noun,
        v:syn:sc = v_sahen_stem:syn:sc,
        v:syn:pre = v_sahen_stem:syn:pre,
        v:syn:te = v_sahen_stem:syn:te,
        v:syn:topic = v_sahen_stem:syn:topic,
        v:sem = v_sahen_stem:sem,
        v:tree = 'v(v_sahen_stem:tree).
v ---> v_ka_sahen |
        v:phon = v_ka_sahen:phon,
        v:syn:st = v_ka_sahen:syn:st,
        v:syn:flex = v_ka_sahen:syn:flex,
        v:syn:sc = v_ka_sahen:syn:sc,
        v:syn:pre = v_ka_sahen:syn:pre,
        v:syn:te = v_ka_sahen:syn:te,
        v:syn:topic = v_ka_sahen:syn:topic,
        v:sem = v_ka_sahen:sem,
        v:tree = 'v(v_ka_sahen:tree).
v ---> v_gozaru |
        v:phon = v_gozaru:phon,
        v:syn:st = v_gozaru:syn:st,
        v:syn:flex = v_gozaru:syn:flex,
        v:syn:sc = v_gozaru:syn:sc,
        v:syn:pre = v_gozaru:syn:pre,
        v:syn:te = v_gozaru:syn:te,
        v:syn:topic = v_gozaru:syn:topic,
        v:sem = v_gozaru:sem,
        v:tree = 'v(v_gozaru:tree).
v ---> v_aru |
        v:phon = v_aru:phon,
        v:syn:st = v_aru:syn:st,
        v:syn:flex = v_aru:syn:flex,
        v:syn:sc = v_aru:syn:sc,
        v:syn:pre = v_aru:syn:pre,
        v:syn:te = v_aru:syn:te,
        v:syn:topic = v_aru:syn:topic,
        v:sem = v_aru:sem,
        v:tree = 'v(v_aru:tree).
v ---> v_neg |
```

```
        v:phon = v_neg:phon,
        v:syn:st = v_neg:syn:st,
        v:syn:flex = v_neg:syn:flex,
        v:syn:sc = v_neg:syn:sc,
        v:syn:pre = v_neg:syn:pre,
        v:syn:te = v_neg:syn:te,
        v:syn:topic = v_neg:syn:topic,
        v:sem = v_neg:sem,
        v:tree = 'v(v_neg:tree).

v ---> v_adj_stem, plus, v_adj_flex |
        phon_liste(v,v_adj_stem,v_adj_flex),
        v_adj_stem:syn:st = v_adj_flex:syn:st,
        v:syn:st = v_adj_stem:syn:st,
        v:syn:flex = v_adj_flex:syn:flex,
        v:syn:sc = v_adj_stem:syn:sc,
        v:syn:pre = v_adj_stem:syn:pre,
        v:syn:te = v_adj_stem:syn:te,
        v:syn:topic = v_adj_stem:syn:topic,
        v:sem = v_adj_stem:sem,
        v:tree = 'v(v_adj_stem:tree,plus,v_adj_flex:tree).

v ---> v_adj_ii |
        v:phon = v_adj_ii:phon,
        v:syn:st = v_adj_ii:syn:st,
        v:syn:flex = v_adj_ii:syn:flex,
        v:syn:sc = v_adj_ii:syn:sc,
        v:syn:te = v_adj_ii:syn:te,.
        v:syn:topic = v_adj_ii:syn:topic,
        v:sem = v_adj_ii:sem,
        v:tree = 'v(v_adj_ii:tree).

h_adj ---> h_adj_stem, plus, v_adj_flex |
        phon_liste(h_adj,h_adj_stem,v_adj_flex),
        h_adj_stem:syn:st = v_adj_flex:syn:st,
        h_adj:syn:st = h_adj_stem:syn:st,
        h_adj:syn:flex = v_adj_flex:syn:flex,
        h_adj:syn:te = h_adj_stem:syn:te,
        h_adj:syn:topic = h_adj_stem:syn:topic,
        h_adj:sem = h_adj_stem:sem,
        h_adj:tree = 'h_adj(h_adj_stem:tree,plus,v_adj_flex:tree).

adresse ---> adresse_germ |
        adresse:phon = adresse_germ:phon,
        adresse:sem = adresse_germ:sem,
        adresse:tree = 'adre_germ(adresse_germ:tree).

adresse_germ ---> adr_strasse, adr_ort |
        phon_liste(adresse_germ, adr_strasse, adr_ort),
        adresse_germ:sem:expr = 'adresse_germ(adr_strasse:sem:expr,
                                adr_ort:sem:expr),
        adresse_germ:tree = 'adre_germ(adr_strasse:tree,adr_ort:tree).

adr_strasse ---> adr1_germ, zahl_germ |
        phon_struc(adr_strasse, adr1_germ, zahl_germ),
        adr_strasse:sem:expr = 'adr_strasse(adr1_germ:sem:expr,
                               zahl_germ:sem:expr),
        adr_strasse:tree = 'adr_strasse(adr1_germ:tree,zahl_germ:tree).

adr_ort ---> zahl_plz, adr2_germ, zahl_pzb |
        phon_struc(adr_ort,zahl_plz, adr2_germ, zahl_pzb),
        adr_ort:sem:expr = 'adr_ort(zahl_plz:sem:expr,
                           adr2_germ:sem:expr,zahl_pzb:sem:expr),
        adr_ort:tree = 'adr_ort(zahl_plz:tree,adr2_germ:tree,zahl_pzb:tree).

zahl_germ ---> num_unit |
        zahl_germ:phon = num_unit:phon,
        zahl_germ:sem:expr = 'zahl_germ(num_unit:sem:expr),
        zahl_germ:tree = 'zahl_germ(num_unit:tree).

zahl_plz ---> num_unit |
        zahl_plz:phon = num_unit:phon,
        zahl_plz:sem:expr = 'zahl_plz(num_unit:sem:expr),
        zahl_plz:tree = 'zahl_plz(num_unit:tree).

zahl_pzb ---> num_unit |
        zahl_pzb:phon = num_unit:phon,
        zahl_pzb:sem:expr = 'zahl_pzb(num_unit:sem:expr),
        zahl_pzb:tree = 'zahl_pzb(num_unit:tree).
```

```
tel_num ---> tel_num_jp |
        tel_num:phon = tel_num_jp:phon,
        tel_num:sem = tel_num_jp:sem,
        tel_num:tree = 'tel_num_jp(tel_num_jp:tree).
tel_num_jp ---> tel_num3_gen, tel_num4 |
        phon_liste(tel_num_jp,tel_num3_gen,tel_num4),
        tel_num_jp:sem:expr = 'tel_num_jp(tel_num3_gen:sem:expr,
                                tel_num4:sem:expr),
        tel_num_jp:tree = 'tel_num_jp(tel_num3_gen:tree, tel_num4:tree).
tel_num3_gen ---> tel_num3, p_gen |
        phon_struc(tel_num3_gen,tel_num3,p_gen),
        tel_num3_gen:sem:expr = 'tel_num3_gen(tel_num3:sem:expr,p_gen:sem:expr),
        tel_num3_gen:tree = 'tel_num3_gen(tel_num3:tree, p_gen:tree).
tel_num3 ---> num0_9, num0_9:a, num0_9:b |
        phon_liste(tel_num3,num0_9,a,b),
        num0_9:syn:harm:neutral = yes,
        num0_9:syn:harm:hpb in {h,p,b,nil},
        a:syn:harm:neutral = yes,
        a:syn:harm:hpb in {h,p,b,nil},
        b:syn:harm:neutral = yes,
        b:syn:harm:hpb in {h,p,b,nil},
        tel_num3:sem:expr = 'tel_num3(num0_9:sem:expr,a:sem:expr,b:sem:expr),
        tel_num3:tree = 'tel_num3(num0_9:tree,a:tree,b:tree).
tel_num4 ---> num0_9, num0_9:a, num0_9:b, num0_9:c |
        tel_num4:phon:p1 = num0_9:phon:p1,
        num0_9:phon:p2 = a:phon:p1,
        a:phon:p2 = b:phon:p1,
        b:phon:p2 = c:phon:p1,
        c:phon:p2 = tel_num4:phon:p2,
        num0_9:syn:harm:neutral = yes,
        num0_9:syn:harm:hpb in {h,p,b,nil},
        a:syn:harm:neutral = yes,
        a:syn:harm:hpb in {h,p,b,nil},
        b:syn:harm:neutral = yes,
        b:syn:harm:hpb in {h,p,b,nil},
        c:syn:harm:neutral = yes,
        c:syn:harm:hpb in {h,p,b,nil},
        tel_num4:sem:expr = 'tel_num4(num0_9:sem:expr,
                              a:sem:expr,b:sem:expr,c:sem:expr),
        tel_num4:tree = 'tel_num4(num0_9:tree,a:tree,b:tree,c:tree).
num0_9 ---> num0 |
        num0_9:phon = num0:phon,
        num0_9:syn = num0:syn,
        num0_9:sem:expr = num0:sem:expr,
        num0_9:tree = num0:tree.
num0_9 ---> num1_9 |
        num0_9:phon = num1_9:phon,
        num0_9:syn = num1_9:syn,
        num0_9:sem:expr = num1_9:sem:expr,
        num0_9:tree = num1_9:tree.
price ---> num_unit, currency_jp |
        phon_liste(price,num_unit,currency_jp),
        price:sem:expr = currency_jp:sem:expr,
        num_unit:sem:expr = currency_jp:sem:expr2,
        price:tree = 'price(num_unit:tree,currency_jp:tree).
price ---> num_unit, currency_dt |
        phon_liste(price,num_unit,currency_dt),
        price:sem:expr = currency_dt:sem:expr,
        num_unit:sem:expr = currency_dt:sem:expr2,
        price:tree = 'price(num_unit:tree,currency_dt:tree).
letter ---> num_unit, letter_zi |
        phon_liste(letter,num_unit,letter_zi),
        letter:sem:expr = letter_zi:sem:expr,
        num_unit:sem:expr = letter_zi:sem:expr2,
        letter:tree = 'letter(num_unit:tree,letter_zi:tree).
num_unit ---> num1_9 |
        num_unit:phon = num1_9:phon,
        num1_9:syn:harm:neutral = yes,
```

```
        num1_9:syn:harm:hpb in {h,p,b,nil},
        num_unit:sem:expr = 'num_unit(num1_9:sem:expr),
        num_unit:tree = 'num_unit(num1_9:tree).
num_unit ---> zyu_unit |
        num_unit:phon = zyu_unit:phon,
        num_unit:sem:expr = 'num_unit(zyu_unit:sem:expr, '0'),
        num_unit:tree = 'num_unit(zyu_unit:tree).
num_unit ---> hyaku_unit |
        num_unit:phon = hyaku_unit:phon,
        num_unit:sem:expr = 'num_unit(hyaku_unit:sem:expr, '00'),
        num_unit:tree = 'num_unit(hyaku_unit:tree).
num_unit ---> sen_unit |
        num_unit:phon = sen_unit:phon,
        num_unit:sem:expr = 'num_unit(sen_unit:sem:expr, '000'),
        num_unit:tree = 'num_unit(sen_unit:tree).
num_unit ---> man_unit |
        num_unit:phon = man_unit:phon,
        num_unit:sem:expr = 'num_unit(man_unit:sem:expr, '0000'),
        num_unit:tree = 'num_unit(man_unit:tree).
num_unit ---> zyu_unit, num1_9 |
        phon_liste(num_unit,zyu_unit, num1_9),
        num1_9:syn:harm:neutral = yes,
        num1_9:syn:harm:hpb in {h,p,b,nil},
        num_unit:sem:expr = 'num_unit(zyu_unit:sem:expr,num1_9:sem:expr),
        num_unit:tree = 'num_unit(zyu_unit:tree, num1_9:tree).
num_unit ---> hyaku_unit, zyu_unit |
        phon_liste(num_unit,hyaku_unit, zyu_unit),
        num_unit:sem:expr = 'num_unit(hyaku_unit:sem:expr,
                            zyu_unit:sem:expr, '0'),
        num_unit:tree = 'num_unit(hyaku_unit:tree, zyu_unit:tree).
num_unit ---> man_unit, sen_unit |
        phon_liste(num_unit,man_unit, sen_unit),
        num_unit:sem:expr = 'num_unit(man_unit:sem:expr,
                            sen_unit:sem:expr, '000'),
        num_unit:tree = 'num_unit(man_unit:tree, sen_unit:tree).
num_unit ---> man_unit, hyaku_unit |
        phon_liste(num_unit,man_unit, hyaku_unit),
        num_unit:sem:expr = 'num_unit(man_unit:sem:expr, '0',
                            hyaku_unit:sem:expr, '00'),
        num_unit:tree = 'num_unit(man_unit:tree, hyaku_unit:tree).
num_unit ---> sen_unit, hyaku_unit |
        phon_liste(num_unit,sen_unit, hyaku_unit),
        num_unit:sem:expr = 'num_unit(sen_unit:sem:expr,
                            hyaku_unit:sem:expr, '00'),
        num_unit:tree = 'num_unit(sen_unit:tree, hyaku_unit:tree).
num_unit ---> man_unit, sen_unit, hyaku_unit |
        phon_liste(num_unit,man_unit, sen_unit, hyaku_unit),
        num_unit:sem:expr = 'num_unit(man_unit:sem:expr,
                            sen_unit:sem:expr, hyaku_unit:sem:expr, '00'),
        num_unit:tree = 'num_unit(man_unit:tree, sen_unit:tree,hyaku_unit:tree).
man_unit ---> num1_9, man_unit1 |
        phon_liste(man_unit,num1_9, man_unit1),
        num1_9:syn:harm = man_unit1:syn:harm,
        man_unit:sem:expr = num1_9:sem:expr,
        man_unit:tree = 'man_unit(num1_9:tree, man_unit1:tree).
sen_unit ---> sen_unit1 |
        sen_unit:phon = sen_unit1:phon,
        sen_unit1:syn:harm:voiceless = s,
        sen_unit1:syn:harm:hpb = nil,
        sen_unit:sem:expr = 1,
        sen_unit:tree = 'sen_unit(sen_unit1:tree).
sen_unit ---> num1_9, sen_unit1 |
        phon_liste(sen_unit,num1_9, sen_unit1),
        num1_9:syn:harm = sen_unit1:syn:harm,
        sen_unit:sem:expr = num1_9:sem:expr,
        sen_unit:tree = 'sen_unit(num1_9:tree, sen_unit1:tree).
```

```
hyaku_unit ---> hyaku_unit1 |
        hyaku_unit:phon = hyaku_unit1:phon,
        hyaku_unit1:syn:harm:hpb = h,
        hyaku_unit:sem:expr = 1,
        hyaku_unit:tree = 'hyaku_unit(hyaku_unit1:tree).
hyaku_unit ---> num1_9, hyaku_unit1 |
        phon_liste(hyaku_unit,num1_9, hyaku_unit1),
        num1_9:syn:harm = hyaku_unit1:syn:harm,
        hyaku_unit:sem:expr = num1_9:sem:expr,
        hyaku_unit:tree = 'hyaku_unit(num1_9:tree, hyaku_unit1:tree).
zyu_unit ---> zyu_unit1 |
        zyu_unit:phon = zyu_unit1:phon,
        zyu_unit:sem:expr = 1,
        zyu_unit:tree = 'zyu_unit(zyu_unit1:tree).
zyu_unit ---> num1_9, zyu_unit1 |
        phon_liste(zyu_unit,num1_9, zyu_unit1),
        num1_9:syn:harm:zyu = yes,
        num1_9:syn:harm:hpb in {h,p,b,nil},
        zyu_unit:sem:expr = num1_9:sem:expr,
        zyu_unit:tree = 'zyu_unit(num1_9:tree, zyu_unit1:tree).
name ---> name_germ |
        name:phon = name_germ:phon,
        name:sem:expr = name_germ:sem:expr,
        name:tree = 'name(name_germ:tree).
name ---> name_germ, suf_name |
        phon_struc(name,name_germ,suf_name),
        name:sem:expr = suf_name:sem:expr,
        name_germ:sem:expr = suf_name:sem:expr2,
        name:tree = 'name(name_germ:tree, suf_name:tree).
name_germ ---> name1_germ1, name2_germ2 |
        phon_struc(name_germ,name1_germ1,name2_germ2),
        name_germ:sem:expr = 'name_germ(name1_germ1:sem, name2_germ2:sem),
        name_germ:tree = 'name_germ(name1_germ1:tree, name2_germ2:tree).
name1_germ1 ---> name1_germ |
        name1_germ1:phon:p1 = '[name1_germ:phon:p1|name1_germ1:phon:p2],
        name1_germ:phon:p2 = [],
        name1_germ1:sem = name1_germ:sem:expr,
        name1_germ1:tree = 'name1_germ1(name1_germ:tree).
name2_germ2 ---> name2_germ |
        name2_germ2:phon:p1 = '[name2_germ:phon:p1|name2_germ2:phon:p2],
        name2_germ:phon:p2 = [],
        name2_germ2:sem = name2_germ:sem:expr,
        name2_germ2:tree = 'name2_germ2(name2_germ:tree).
month_day ---> month, day |
        phon_struc(month_day,month,day),
        month_day:sem:expr = 'date(month:sem:expr,day:sem:expr),
        month_day:tree = 'month_day(month:tree,day:tree).
month ---> num1_9, suf_mon |
        phon_struc(month,num1_9,suf_mon),
        num1_9:syn:harm:gatu = yes,
        num1_9:syn:harm:hpb in {h,p,b,nil},
        month:sem:expr = suf_mon:sem:expr,
        num1_9:sem:expr = suf_mon:sem:expr2,
        month:tree = 'month(num1_9:tree,suf_mon:tree).
month ---> num10_12, suf_mon |
        phon_struc(month,num10_12,suf_mon),
        num10_12:syn:harm:gatu = yes,
        month:sem:expr = suf_mon:sem:expr,
        num10_12:sem:expr = suf_mon:sem:expr2,
        month:tree = 'month(num10_12:tree,suf_mon:tree).
day ---> day1_10 |
        day:phon = day1_10:phon,
        day:sem:expr = day1_10:sem:expr,
        day:tree = day1_10:tree.
day ---> day11_31, suf_day |
        phon_struc(day,day11_31,suf_day),
        day11_31:syn:harm:niti = suf_day:syn:harm:niti,
```

```
        day:sem:expr = suf_day:sem:expr,
        day11_31:sem:expr = suf_day:sem:expr2,
        day:tree = 'day(day11_31:tree,suf_day:tree).
day11_31 ---> num10, num1_9 |
        phon_liste(day11_31,num10,num1_9),
        day11_31:syn:harm:niti = num1_9:syn:harm:niti,
        day11_31:sem:expr = 'day(num10:sem:expr, num1_9:sem:expr),
        day11_31:tree = 'day(num10:tree,num1_9:tree).
day11_31 ---> num20, num1_9 |
        phon_liste(day11_31,num20,num1_9),
        day11_31:syn:harm:niti = num1_9:syn:harm:niti,
        day11_31:sem:expr = 'day(num20:sem:expr, num1_9:sem:expr),
        day11_31:tree = 'day(num20:tree,num1_9:tree).
day11_31 ---> num30_31 |
        day11_31:phon = num30_31:phon,
        day11_31:syn:harm:niti = num30_31:syn:harm:niti,
        day11_31:sem:expr = num30_31:sem:expr,
        day11_31:tree = 'day(num30_31:tree).
edition ---> num1_9, suf_edit |
        phon_struc(edition,num1_9,suf_edit),
        num1_9:syn:harm:hpb = suf_edit:syn:harm:hpb,
        edition:sem:expr = suf_edit:sem:expr,
        num1_9:sem:expr = suf_edit:sem:expr2,
        edition:tree = 'edition(num1_9:tree,suf_edit:tree).
trace(ana,p_casep) |
        p_casep:phon:p2 = '[p_casep:phon:p1],
        p_casep:empty = yes,
        p_casep:syn:case in {ga,ni,o},
        p_casep:tree = 'p_casep(p_casep:syn:case,'TRACE').
```

C.4 Lexicon

This is a partial listing of the lexicon.

```
lexicon(moshimoshi,interj) |
        phon_lex(interj,interj(moshimoshi,4,1,2)),
        interj:sem:expr = 'tel_phatic(interj:sem:var),
        interj:tree = interj(moshimoshi).
lexicon(douitashimashite, interj) |
        phon_lex(interj,interj('douitashimashi#te',8,1,2)),
        interj:sem:expr = 'greeting_welcome(interj:sem:var),
        interj:tree = interj(douitashimashite).
lexicon(moushiwakearimasenga, idiom) |
        phon_lex(idiom,idiom(moushiwakearimaseNga,11,9,1)),
        idiom:sem:expr = 'infattd(idiom:sem:var),
        idiom:tree = idiom(moushiwakearimasenga).
lexicon(mada, adv) |
        phon_lex(adv,adv(mada,2,1,2)),
        adv:syn:bare = bare,
        adv:syn:n_mod = no,
        adv:sem:expr = 'tloc(quant(syn(adv:syn:polite)),mada,adv:sem:var),
        adv:tree = adv(mada).
lexicon(sudeni, adv) |
        phon_lex(adv,adv(sudeni,3,1,2)),
        adv:syn:bare = bare,
        adv:syn:n_mod = no,
        adv:sem:expr = 'tloc(quant(syn(adv:syn:polite)),sudeni,adv:sem:var),
        adv:tree = adv(sudeni).
lexicon(sono, att) |
        phon_lex(att,att(sono,2,0,2)),
        att:sem:expr = 'att(far,att:sem:var),
        att:tree = att(sono).
lexicon(itsu, wh_pron) |
        phon_lex(wh_pron,wh_pron(itsu,2,1,2)),
        wh_pron:syn:suf:full = no,
        wh_pron:sem:expr = 'quant(syn(wh_pron(when,wh_pron:syn:topic),
```

```
                                whpron,X,restr(wh(X),[],[]),_)),
        wh_pron:tree = wh_pron(itsu).
lexicon(sochira, pron) |
        phon_lex(pron,pron(sochira,3,0,2)),
        pron:sem:expr = 'quant(syn(prs2(deix_far),hearer,_,pron:syn:topic),
                          perspron,X,restr(personal(X),[],[]),_),
        pron:tree = pron(sochira).
lexicon(watashi, pron) |
        phon_lex(pron,pron(watashi,3,0,2)),
        pron:sem:expr = 'quant(syn(prs1(watashi),speaker,_,pron:syn:topic),
                          perspron,X,restr(personal(X),[],[]),_),
        pron:tree = pron(watashi).
lexicon(wa, p_case) |
        phon_lex(p_case,p_case(wa,1,0,1)),
        p_case:syn:topic = wa,
        p_case:syn:case in {ga,o},
        p_case:tree = p_case(wa).
lexicon(ni, p_case) |
        phon_lex(p_case,p_case(ni,1,0,1)),
        p_case:syn:topic = nil,
        p_case:syn:case = ni,
        p_case:tree = p_case(ni).
lexicon(o, p_case) |
        phon_lex(p_case,p_case(o,1,0,1)),
        p_case:syn:topic = nil,
        p_case:syn:case = o,
        p_case:tree = p_case(o).
lexicon(ga, p_case) |
        phon_lex(p_case,p_case(ga,1,0,1)),
        p_case:syn:topic = nil,
        p_case:syn:case = ga,
        p_case:tree = p_case(ga).
lexicon(no, p_gen) |
        phon_lex(p_gen,p_gen(no,1,0,1)),
        p_gen:sem:expr = 'genitive(no,p_gen:sem:var),
        p_gen:tree = p_gen(no).
lexicon(ni, postp) |
        phon_lex(postp,postp(ni,1,0,1)),
        postp:syn:topic = nil,
        postp:syn:copula = no,
        postp:syn:ad:verb = yes,
        postp:syn:flex = rentai,
        postp:syn:adv_use = yes,
        postp:syn:coord_use = no,
        postp:sem:expr = 'postp_optn(ni,postp:sem:var,postp:sem:expr2),
        postp:tree = postp(ni).
lexicon(to, koord) |
        phon_lex(koord,koord(to,1,0,1)),
        koord:sem:expr = 'and(to),
        koord:tree = koord(to).
lexicon(no, no) |
        phon_lex(no,no(no,1,0,1)),
        no:tree = no(no).
lexicon(te, te) |
        phon_lex(te,te(te,1,0,1)),
        te:syn:te = te,
        te:syn:topic = nil,
        te:tree = te(te).
lexicon(ka, s_prt) |
        phon_lex(s_prt,s_prt(ka,1,0,2)),
        s_prt:syn:quote = direct,
        s_prt:syn:inh:flex = syusi,
        s_prt:sem:expr = 'request_inf(s_prt:sem:expr2),
        s_prt:tree = s_prt(ka).
lexicon(tara, conj) |
        phon_lex(conj,conj(tara,2,0,2)),
        conj:syn:inh:flex = renyo_i,
        conj:syn:inh:te = te,
```

```
        conj:syn:locate = mid,
        conj:sem:expr = 'cond_tara(conj:sem:var,conj:sem:expr2),
        conj:tree = conj(tara).
lexicon(to, quote_p) |
        phon_lex(quote_p,quote_p(to,1,0,1)),
        quote_p:syn:topic = nil,
        quote_p:sem:expr = 'quote(to,quote_p:sem:var,quote_p:sem:expr2),
        quote_p:tree = quote_p(to).
lexicon(go, pre) |
        phon_lex(pre,pre(go,1,0,11)),
        pre:syn:form = go,
        pre:tree = pre(go).
lexicon(dai, pre_num) |
        phon_lex(pre_num,pre_num(dai,2,1,13)),
        pre_num:tree = pre_num(dai).
lexicon(kaigizimukyoku, n) |
        phon_lex(n,n(kaigizimukyoku,7,5,2)),
        n:syn:pre_form = nil,
        n:sem:expr = 'kaigizimukyoku(n:sem:var),
        n:tree = n(kaigizimukyoku).
lexicon(kaigi, n) |
        phon_lex(n,n(kaigi,3,1,2)),
        n:syn:pre_form = nil,
        n:syn:adv_use = no,
        n:sem:expr = 'kaigi(n:sem:var),
        n:tree = n(kaigi).
lexicon(tourokuyoushi, n) |
        phon_lex(n,n(tourokuyoushi,7,5,2)),
        n:syn:pre_form = go,
        n:syn:adv_use = no,
        n:sem:expr = 'tourokuyoushi(n:sem:var),
        n:tree = n(tourokuyoushi).
lexicon(ten, n) |
        phon_lex(n,n(teN,2,0,2)),
        n:syn:pre_form = nil,
        n:sem:expr = 'ten(n:sem:var),
        n:tree = n(ten).
lexicon(annaisho, n) |
        phon_lex(n,n(aNnaisho,5,0,2)),
        n:syn:pre_form = nil,
        n:syn:adv_use = no,
        n:sem:expr = 'annaisho(n:sem:var),
        n:tree = n(annaisho).
lexicon(ronbun, n) |
        phon_lex(n,n(roNbuN,4,0,2)),
        n:syn:pre_form = nil,
        n:syn:adv_use = no,
        n:sem:expr = 'ronbun(n:sem:var),
        n:tree = n(ronbun).
lexicon(dairinin, n) |
        phon_lex(n,n(dairiniN,5,0,2)),
        n:syn:pre_form = nil,
        n:syn:adv_use = no,
        n:sem:expr = 'dairinin(n:sem:var),
        n:tree = n(dairinin).
lexicon(daimoku, n) |
        phon_lex(n,n(daimoku,4,0,2)),
        n:syn:pre_form = nil,
        n:syn:adv_use = no,
        n:sem:expr = 'daimoku(n:sem:var),
        n:tree = n(daimoku).
lexicon(baai, n) |
        phon_lex(n,n(baai,3,0,2)),
        n:syn:pre_form = nil,
        n:syn:adv_use = yes,
        n:sem:expr = 'baai(n:sem:var),
        n:tree = n(baai).
lexicon(kyoutokokusaikaigizyou, n) |
```

```
        phon_lex(n,n('kyoutokoku#saikaigijou',12,1,2)),
        n:syn:pre_form = nil,
        n:syn:adv_use = no,
        n:sem:expr = 'kyoutokokusaikaigizyou(n:sem:var),
        n:tree = n(kyoutokokusaikaigizyou).
lexicon(siemens, n) |
        phon_lex(n,n(siimeNsu,5,0,1)),
        n:syn:pre_form = nil,
        n:syn:adv_use = no,
        n:sem:expr = 'siemens(n:sem:var),
        n:tree = n(siemens).
lexicon(kanou, adj_n) |
       phon_lex(adj_n,adj_n(kanou,3,0,2)),
       adj_n:syn:pre_form = nil,
       adj_n:syn:adv_use = no,
       adj_n:sem:expr = 'possible(adj_n:sem:var),
       adj_n:tree = adj_n(kanou).
lexicon(desu, cop) |
        phon_lex(cop,cop('desu#',2,0,1)),
        cop:syn:sc:arg1 = ga,
        cop:syn:st = excep,
        cop:syn:flex = syusi,
        cop:syn:polite = polite,
        (cop:syn:sc:arg2 = np,
          cop:sem:expr = 'identical(quant(syn(cop:syn:polite),ex,cop:sem:var,
                                    restr(nil,[],[]),_),
                                    cop:sem:sc:arg1,cop:sem:sc:arg2)
         ;cop:syn:sc:arg2 = adv,cop:sem:expr = cop:sem:sc:arg2),
        cop:tree = cop(desu).
lexicon(reru, aux) |
        phon_lex(aux,aux(reru,2,1,1)),
        (godan(aux:syn:inh:st),aux:syn:inh:flex = mizen
         ;aux:syn:inh:st = v_sahen,aux:syn:inh:flex = mizen_caus),
        aux:syn:st = itidan,
        aux:syn:flex in {syusi,rentai},
        aux:syn:ad:att = no,
        aux:syn:ad:se = yes,
        aux:syn:ad:no_desu = no,
        aux:sem:expr = 'passive(aux:syn:topic,aux:sem:expr2),
        aux:tree = aux(reru).
lexicon(kudasai, h_aux) |
        phon_lex(h_aux,h_aux(kudasai,4,3,1)),
        h_aux:syn:inh:flex =renyo_noun,
        h_aux:syn:st = godan_r,
        h_aux:syn:flex = meirei_pol,
        h_aux:syn:ad:te = yes,
        h_aux:syn:ad:h_verb = yes,
        h_aux:sem:expr = 'give_favor(h_aux:syn:topic,h_aux:sem:expr2),
        h_aux:tree = h_aux(kudasai).
lexicon(waka, v_stem:v) |
        phon_lex(v,v(waka,2,0,2)),
        sc(v,ga,ga,nil),
        v:syn:st = godan_r,
        v:syn:pre = jp,
        v:syn:te = te,
        v:sem:expr = 'wakaru(quant(v:syn:topic,_,ex,v:sem:var,
                                restr(nil,[],[]),_),
                                v:sem:sc:arg1,v:sem:sc:arg2),
        v:tree = v(waka).
lexicon(kaka, v_stem:v) |
        phon_lex(v,v(kaka,2,0,2)),
        sc(v,ga,nil,nil),
        v:syn:st = godan_r,
        v:syn:pre = jp,
        v:syn:te = te,
        v:sem:expr = 'kakaru(quant(v:syn:topic,_,ex,v:sem:var,
                                restr(nil,[],[]),_),
                                v:sem:sc:arg1),
        v:tree = v(kaka).
lexicon(oku, v_stem:v) |
```

```
        phon_lex(v,v(oku,2,0,2)),
        sc(v,ga,ni,o),
        v:syn:st = godan_r,
        v:syn:pre = jp,
        v:syn:te = te,
        v:syn:polite = nonpolite,
        v:sem:expr = 'okuru(quant(v:syn:topic,v:syn:polite,ex,v:sem:var,
                              restr(nil,[],[]),_),
                              v:sem:sc:arg1,v:sem:sc:arg2,v:sem:sc:arg3),
        v:tree = v(oku).
lexicon(sanka, v_sahen_stem:v) |
        phon_lex(v,v(saNka,3,0,2)),
        sc(v,ga,ni,nil),
        v:syn:st = v_sahen,
        v:syn:pre = sino_jp,
        v:syn:te = te,
        v:sem:expr = 'sanka_suru(quant(v:syn:topic,v:syn:polite,ex,v:sem:var,
                                  restr(nil,[],[]),_),
                                  v:sem:sc:arg1,v:sem:sc:arg2),
        v:tree = v(sanka).
lexicon(kaisai, v_sahen_stem:v) |
        phon_lex(v,v(kaisai,4,0,2)),
        sc(v,ga,o,nil),
        v:syn:st = v_sahen,
        v:syn:pre = sino_jp,
        v:syn:te = te,
        v:sem:expr = 'kaisai_suru(quant(v:syn:topic,v:syn:polite,ex,v:sem:var,
                                   restr(nil,[],[]),_),
                                   v:sem:sc:arg1,v:sem:sc:arg2),
        v:tree = v(kaisai).
lexicon(omochi, v_sahen_stem:v) |
        phon_lex(v,v(omochi,3,0,2)),
        sc(v,ga,o,nil),
        v:syn:st = pre_v_sahen,
        v:syn:pre = pre_jp,
        v:syn:te = te,
        v:syn:polite = polite,
        v:sem:expr = 'motsu(quant(v:syn:topic,v:syn:polite,ex,v:sem:var,
                              restr(nil,[],[]),_),
                              v:sem:sc:arg1,v:sem:sc:arg2),
        v:tree = v(omochi).
lexicon(okiki, v_sahen_stem:v) |
        phon_lex(v,v('oki#ki',3,0,2)),
        sc(v,ga,o,nil),
        v:syn:st = pre_v_sahen,
        v:syn:pre = pre_jp,
        v:syn:te = te,
        v:syn:polite = polite,
        v:sem:expr = 'kiku(quant(v:syn:topic,v:syn:polite,ex,v:sem:var,
                             restr(nil,[],[]),_),
                             v:sem:sc:arg1,v:sem:sc:arg2),
        v:tree = v(okiki).
lexicon(oshirase, v_sahen_stem:v) |
        phon_lex(v,v(oshirase,4,0,2)),
        sc(v,ga,ni,o),
        v:syn:st = pre_v_sahen,
        v:syn:pre = pre_jp,
        v:syn:te = te,
        v:syn:polite = polite,
        v:sem:expr = 'siraseru(quant(v:syn:topic,v:syn:polite,ex,v:sem:var,
                                restr(nil,[],[]),_),
                                v:sem:sc:arg1,v:sem:sc:arg2,v:sem:sc:arg3),
        v:tree = v(oshirase).
lexicon(gozai, v_gozaru:v) |
        phon_lex(v,v(gozai,3,0,2)),
        sc(v,ga,nil,nil),
        v:syn:st = godan_r,
        v:syn:flex = renyo,
        v:syn:pre = jp,
        v:syn:te = te,
        v:syn:polite = polite,
```

```
        v:sem:expr = 'aru_polite(quant(v:syn:topic,v:syn:polite,ex,v:sem:var,
                                       restr(nil,□,□),_),
                                       v:sem:sc:arg1),
        v:tree = v(gozai).
lexicon(nai, v_neg) |
        phon_lex(v_neg,v_neg(nai,2,1,2)),
        sc(v_neg,ga,nil,nil),
        v_neg:syn:st = keiyosi,
        v_neg:syn:flex in {syusi,rentai},
        v_neg:syn:pre = jp,
        v_neg:sem:expr = 'negate(aru(quant(v_neg:syn:topic,_,ex,v_neg:sem:var,
                                        restr(nil,□,□),_), v_neg:sem:sc:arg1)),
        v_neg:tree = v_neg(nai).
lexicon(kuru,v_ka_sahen) |
        phon_lex(v_ka_sahen,v_ka_sahen(kuru,2,1,2)),
        sc(v_ka_sahen,ga,nil,nil),
        v_ka_sahen:syn:st = kahen,
        v_ka_sahen:syn:flex in {syusi,rentai},
        v_ka_sahen:syn:pre = jp,
        v_ka_sahen:sem:expr = 'kuru(quant(v_ka_sahen:syn:topic,_,ex,
                                    v_ka_sahen:sem:var,restr(nil,□,□),_),
                                    v_ka_sahen:sem:sc:arg1),
        v_ka_sahen:tree = v_ka_sahen(kuru).
lexicon(yo, v_adj_stem) |
        phon_lex(v_adj_stem,v_adj_stem(yo,1,0,2)),
        sc(v_adj_stem,ga,ga,nil),
        v_adj_stem:syn:st = keiyosi,
        v_adj_stem:syn:te = te,
        v_adj_stem:sem:expr = 'yoi(quant(v_adj_stem:syn:topic,_,ex,
                                    v_adj_stem:sem:var,
                                    restr(nil,□,□),_),
                                    v_adj_stem:sem:sc:arg1,
                                    v_adj_stem:sem:sc:arg2),
        v_adj_stem:tree = v_adj_stem(yo).
lexicon(hoshi, h_adj_stem) |
        phon_lex(h_adj_stem,h_adj_stem(hoshi,2,0,2)),
        h_adj_stem:syn:st = keiyosi,
        h_adj_stem:syn:te = te,
        h_adj_stem:sem:expr = 'will(h_adj_stem:syn:topic,h_adj_stem:sem:expr2),
        h_adj_stem:tree = h_adj_stem(hoshi).
lexicon(susanne, name1_germ) |
        phon_lex(name1_germ,name1_germ(suzaNne,4,2,4)),
        name1_germ:sem:expr = susanne,
        name1_germ:tree = 'name1_germ(susanne).
lexicon(kaiser, name2_germ) |
        phon_lex(name2_germ,name2_germ(kaizaa,4,1,4)),
        name2_germ:sem:expr = kaiser,
        name2_germ:tree = 'name2_germ(kaiser).
lexicon(ka, v_flex) |
        phon_lex(v_flex,v_flex(ka,1,0,1)),
        v_flex:syn:st in {godan_k,godan_k_iku},
        v_flex:syn:flex in {mizen,mizen_caus},
        v_flex:tree = 'v_flex(ka).
lexicon(i, v_adj_flex) |
        phon_lex(v_adj_flex,v_adj_flex(i,1,0,2)),
        v_adj_flex:syn:st = keiyosi ,
        v_adj_flex:syn:flex in {syusi,rentai} ,
        v_adj_flex:tree = 'v_adj_flex(i).
lexicon(zyuu, num10_12) |
        phon_lex(num10_12,num10_12(juu,2,1,1)),
        num10_12:syn:harm:neutral = yes,
        num10_12:syn:harm:voiceless = nil,
        num10_12:syn:harm:hpb = nil,
        num10_12:syn:harm:gatu = yes,
        num10_12:syn:harm:niti = nil,
        num10_12:sem:expr = 10,
        num10_12:tree = 'num10_12(zyuu).
lexicon(igo, suf_noun) |
```

```
        phon_lex(suf_noun,suf_noun(igo,2,1,2)),
        suf_noun:syn:inh:full = no,
        suf_noun:syn:inh:after = yes,
        suf_noun:syn:inh:wh_ka = no,
        suf_noun:syn:adv_use = yes,
        suf_noun:sem:expr = 'after(suf_noun:sem:expr2),
        suf_noun:tree = suf_noun(igo).
lexicon(ka, suf_wh) |
        phon_lex(suf_wh,suf_wh(ka,1,0,1)),
        suf_wh:syn:inh:full = no,
        suf_wh:syn:inh:after = no,
        suf_wh:syn:inh:wh_ka = yes,
        suf_wh:syn:adv_use = no,
        suf_wh:sem:expr = 'wh_ka(suf_wh:sem:expr2),
        suf_wh:tree = suf_wh(ka).
lexicon(sama, suf_name) |
        phon_lex(suf_name,suf_name(sama,2,0,1)),
        suf_name:sem:expr = 'sama(suf_name:sem:expr2),
        suf_name:tree = suf_name(sama).
lexicon(gatsu, suf_mon) |
        phon_lex(suf_mon,suf_mon(gatsu,2,0,1)),
        suf_mon:sem:expr = 'month(suf_mon:sem:expr2),
        suf_mon:tree = suf_mon(gatsu).
lexicon(nichi, suf_day) |
        phon_lex(suf_day,suf_day(nichi,2,0,1)),
        suf_day:syn:harm:niti = n,
        suf_day:sem:expr = 'day(suf_day:sem:expr2),
        suf_day:tree = suf_day(nichi).
lexicon(en, currency_jp) |
        phon_lex(currency_jp,currency_jp(eN,2,1,3)),
        currency_jp:sem:expr = 'jp_en(currency_jp:sem:expr2),
        currency_jp:tree = currency_jp(en).
lexicon(zi, letter_zi) |
        phon_lex(letter_zi,letter_zi(zi,1,0,1)),
        letter_zi:sem:expr = 'letter_zi(letter_zi:sem:expr2),
        letter_zi:tree = letter_zi(zi).
lexicon(han, suf_edit) |
        phon_lex(suf_edit,suf_edit(haN,2,0,2)),
        suf_edit:syn:harm:hpb = h,
        suf_edit:sem:expr = 'suf_edit(suf_edit:sem:expr2),
        suf_edit:tree = suf_edit(han).
lexicon(man, man_unit1) |
        phon_lex(man_unit1,man_unit1(maN,2,0,4)),
        man_unit1:syn:harm:neutral = yes,
        man_unit1:syn:harm:hpb in {h,p,b,nil},
        man_unit1:sem:expr = [],
        man_unit1:tree = 'man_unit(man).
lexicon(sen, sen_unit1) |
        phon_lex(sen_unit1,sen_unit1(seN,2,0,4)),
        sen_unit1:syn:harm:voiceless = s,
        sen_unit1:syn:harm:hpb in {h,p,b,nil},
        sen_unit1:sem:expr = [],
        sen_unit1:tree = 'sen_unit(sen).
lexicon(hyaku,hyaku_unit1 ) |
        phon_lex(hyaku_unit1,hyaku_unit1(hyaku,2,0,4)),
        hyaku_unit1:syn:harm:hpb = h,
        hyaku_unit1:sem:expr = [],
        hyaku_unit1:tree = 'hyaku_unit(hyaku).
lexicon(zyuu, zyu_unit1) |
        phon_lex(zyu_unit1,zyu_unit1(juu,2,0,1)),
        zyu_unit1:syn:harm:zyu = yes,
        zyu_unit1:syn:harm:hpb = nil,
        zyu_unit1:sem:expr = [],
        zyu_unit1:tree = 'zyu_unit(zyuu).
lexicon(tsuitachi,day1_10) |
        phon_lex(day1_10,day1_10(tsuitachi,4,0,1)),
        day1_10:sem:expr = 'day(1),
        day1_10:tree = 'day(tsuitachi).
```

Appendix D

Generation Results

This is a list of phon informations extracted from generation results of Dialogues A and B. The phon information is sent to the Japanese speech synthesis system ν-talk.

```
phon = [interj(moshimoshi,4,1,2)].

phon = [[[pron(sochira,3,0,2),p_case(wa,1,0,1)],[n(kaigizimukyoku,7,5,2),cop(desu#,
2,0,1)]],s_prt(ka,1,0,2)].

phon = [interj(hai,2,1,2)].

phon = [[adv(sou,2,1,2),cop(desu#,2,0,1)]].

phon = [[[[n(kaigi,3,1,2),p_case(ni,1,0,1)],[v(moushi#ko,4,0,2),v_flex(mi,1,0,1),au
x(tai,2,1,4),no(no,1,0,1),aux(desu#,2,1,2)]],conj(ga,1,0,1)]].

phon = [[[[n(kaigi,3,1,2),p_case(ni,1,0,1)],[v(moushi#ko,4,0,2),v_flex(mi,1,0,1),au
x(tai,2,1,4),no(N,1,0,1),aux(desu#,2,1,2)]],conj(ga,1,0,1)]].

phon = [[[adv(sudeni,3,1,2)],[[n(tourokuyoushi,7,5,2),p_case(wa,1,0,1)],[v(omochi,3
,0,2),aux(desho,2,0,2),aux(u,1,0,1)]]],s_prt(ka,1,0,2)].

phon = [[[n(tourokuyoushi,7,5,2),p_case(wa,1,0,1)],[[adv(sudeni,3,1,2)],[v(omochi,3
,0,2),aux(desho,2,0,2),aux(u,1,0,1)]]],s_prt(ka,1,0,2)].

phon = [interj(iie,3,0,2)].

phon = [[adv(mada,2,1,2),cop(desu#,2,0,1)]].

phon = [[v(waka,2,0,2),v_flex(ri,1,0,1),aux(mashi#,2,1,2),aux(ta,1,0,1)]].

phon = [[[adv(soredewa,4,3,2)],[[n(tourokuyoushi,7,5,2),p_case(o,1,0,1)],[v(ookuri,
4,0,2),h_aux(itashi#,3,0,1),aux(masu#,2,1,2)]]]].

phon = [[[n(tourokuyoushi,7,5,2),p_case(o,1,0,1)],[[adv(soredewa,4,3,2)],[v(ookuri,
4,0,2),h_aux(itashi#,3,0,1),aux(masu#,2,1,2)]]]].

phon = [[[[pre(go,1,0,11),n(juusho,3,1,2),koord(to,1,0,1),pre(o,1,0,11),n(namae,3,0
,2)],p_case(o,1,0,1)],[v(onegai,4,0,2),v_flex(shi,1,0,1),aux(masu#,2,1,2)]]].

phon = [[[n(juusho,3,1,2),p_case(wa,1,0,1)],[[adr1_germ(sheriNgudouri,9,0,2),zyu_un
it1(juu,2,0,1),num1_9(ni,1,0,1)],[num1_9(has,1,0,1),sen_unit1(seN,2,0,4),adr2_germ(
myuNheN,4,1,2),num1_9(yoN,2,1,1),zyu_unit1(juu,2,0,1)],cop(desu#,2,0,1)]]].

phon = [[[n(namae,3,0,2),p_case(wa,1,0,1)],[[[name1_germ(beruta,3,1,4)],[name2_germ
(shu#taNguru,5,2,4)]],cop(desu#,2,0,1)]]].

phon = [[v(waka,2,0,2),v_flex(ri,1,0,1),aux(mashi#,2,1,2),aux(ta,1,0,1)]].

phon = [[[adv(shi#kyuu,3,0,2)],[[n(tourokuyoushi,7,5,2),p_case(wa,1,0,1)],[v(oku,2,
0,2),v_flex(ra,1,0,1),aux(se,1,0,1),te(te,1,0,1),h_aux(itadaki,4,0,1),aux(masu#,2,1
,2)]]]].
```

```
phon = [[[n(tourokuyoushi,7,5,2),p_case(wa,1,0,1)],[[adv(shi#kyuu,3,0,2)],[v(oku,2,
0,2),v_flex(ra,1,0,1),aux(se,1,0,1),te(te,1,0,1),h_aux(itadaki,4,0,1),aux(masu#,2,1
,2)]]]].

phon = [[[[[[v(waka,2,0,2),v_flex(ra,1,0,1),aux(nai,2,1,1)],n(teN,2,0,2)],p_case(ga
,1,0,1)],[v(gozai,3,0,2),aux(mashi#,2,1,2)]],conj(tara,2,0,2)],[[adv(itsudemo,4,1,2
)],[v(oki#ki,3,0,2),h_aux(kudasai,4,3,1)]]].

phon = [[v(arigatougozai,8,2,2),aux(masu#,2,1,2)]].

phon = [[[adv(soredewa,4,3,2)],[v(shi#tsurei,4,2,2),v_flex(shi,1,0,1),aux(masu#,2,1
,2)]]].

phon = [[[adv(doumo,3,1,2)],[v(shi#tsurei,4,2,2),h_aux(itashi#,3,0,1),aux(masu#,2,1
,2)]]].

phon = [interj(moshimoshi,4,1,2)].

phon = [[[pron(kochira,3,0,2),p_case(wa,1,0,1)],[n(kaigizimukyoku,7,5,2),cop(desu#,
2,0,1)]]].

phon = [[[[n(kaigi,3,1,2),p_case(ni,1,0,1)],[v(saNka,3,0,2),v_flex(shi,1,0,1),aux(t
ai,2,1,4),no(no,1,0,1),aux(desu#,2,1,2)]],conj(ga,1,0,1)]].

phon = [[[[n(kaigi,3,1,2),p_case(ni,1,0,1)],[v(saNka,3,0,2),v_flex(shi,1,0,1),aux(t
ai,2,1,4),no(N,1,0,1),aux(desu#,2,1,2)]],conj(ga,1,0,1)]].

phon = [[[adv(dou,2,1,2),[v_ka_sahen(sure,2,1,2)]],conj(ba,1,0,1)],[v_adj_stem(yoro
shi,3,0,2),v_adj_flex(i,1,0,2),aux(desu#,2,1,2)],s_prt(ka,1,0,2)].

phon = [[[adv(dou,2,1,2),[v_ka_sahen(sure,2,1,2)]],conj(ba,1,0,1)],[v_adj_stem(yoro
shi,3,0,2),v_adj_flex(i,1,0,2),no(no,1,0,1),aux(desu#,2,1,2)],s_prt(ka,1,0,2)].

phon = [[[adv(dou,2,1,2),[v_ka_sahen(sure,2,1,2)]],conj(ba,1,0,1)],[v_adj_stem(yoro
shi,3,0,2),v_adj_flex(i,1,0,2),no(N,1,0,1),aux(desu#,2,1,2)],s_prt(ka,1,0,2)].

phon = [[[[adv(mazu,2,1,2)],[[n(tourokuyoushi,7,5,2),postp(de,1,0,1)],[[n(tetsuzuki
,4,2,2),p_case(o,1,0,1)],[v_ka_sahen(shi,1,0,2),te(te,1,0,1),h_aux(itadaka,4,0,1),a
ux(naku,2,1,1),te(tewa,2,1,2),h_aux(nari,2,1,1),aux(maseN,3,2,2)]]]],conj(ga,1,0,1)
]].

phon = [[[[adv(mazu,2,1,2)],[[n(tetsuzuki,4,2,2),p_case(o,1,0,1)],[[n(tourokuyoushi
,7,5,2),postp(de,1,0,1)],[v_ka_sahen(shi,1,0,2),te(te,1,0,1),h_aux(itadaka,4,0,1),a
ux(naku,2,1,1),te(tewa,2,1,2),h_aux(nari,2,1,1),aux(maseN,3,2,2)]]]],conj(ga,1,0,1)
]].

phon = [[[[n(tetsuzuki,4,2,2),p_case(o,1,0,1)],[[adv(mazu,2,1,2)],[[n(tourokuyoushi
,7,5,2),postp(de,1,0,1)],[v_ka_sahen(shi,1,0,2),te(te,1,0,1),h_aux(itadaka,4,0,1),a
ux(naku,2,1,1),te(tewa,2,1,2),h_aux(nari,2,1,1),aux(maseN,3,2,2)]]]],conj(ga,1,0,1)
]].

phon = [[[adv(mou,2,0,2)],[[n(tourokuyoushi,7,5,2),p_case(wa,1,0,1)],[v(omochi,3,0,
2),aux(desho,2,0,2),aux(u,1,0,1)]]],s_prt(ka,1,0,2)].

phon = [[[n(tourokuyoushi,7,5,2),p_case(wa,1,0,1)],[[adv(mou,2,0,2)],[v(omochi,3,0,
2),aux(desho,2,0,2),aux(u,1,0,1)]]],s_prt(ka,1,0,2)].

phon = [[adv(mada,2,1,2),cop(desu#,2,0,1)]].

phon = [[[n(youshi,3,0,2),p_case(o,1,0,1)],[v(oku,2,0,2),v_flex(t,1,0,1),te(te,1,0,
1),h_aux(kudasai,4,3,1)]]].

phon = [[[adv(dewa,2,1,2)],[[[pre(go,1,0,11),n(juusho,3,1,2),koord(to,1,0,1),pre(o,
1,0,11),n(namae,3,0,2)],p_case(o,1,0,1)],[v(onegai,4,0,2),v_flex(shi,1,0,1),aux(mas
u#,2,1,2)]]]].

phon = [[[[pre(go,1,0,11),n(juusho,3,1,2),koord(to,1,0,1),pre(o,1,0,11),n(namae,3,0
,2)],p_case(o,1,0,1)],[[adv(dewa,2,1,2)],[v(onegai,4,0,2),v_flex(shi,1,0,1),aux(mas
u#,2,1,2)]]]].

phon = [[[n(juusho,3,1,2),p_case(wa,1,0,1)],[[adr1_germ(ruudobihhidouri,9,0,1),num1
_9(saN,2,0,1),zyu_unit1(juu,2,0,1),num1_9(ni,1,0,1)],[num1_9(has,1,0,1),sen_unit1(s
```

```
eN,2,0,4),adr2_germ(myuNheN,4,1,2),num1_9(yoN,2,1,1),zyu_unit1(juu,2,0,1)],cop(desu
#,2,0,1)]]].

phon = [[[n(namae,3,0,2),p_case(wa,1,0,1)],[[[name1_germ(harutomuuto,6,4,4)],[name2
_germ(rafuraa,4,1,4)]],cop(desu#,2,0,1)]]].

phon = [[v(waka,2,0,2),v_flex(ri,1,0,1),aux(mashi#,2,1,2),aux(ta,1,0,1)]].

phon = [[[n(saNkaryou,5,3,2),p_case(wa,1,0,1)],[v(i,1,0,2),v_flex(ru,1,0,1),aux(des
ho,2,0,2),aux(u,1,0,1)]],s_prt(ka,1,0,2)].

phon = [[[n(saNkaryou,5,3,2),p_case(wa,1,0,1)],[v(i,1,0,2),v_flex(ru,1,0,1),no(no,1
,0,1),aux(desho,2,0,2),aux(u,1,0,1)]],s_prt(ka,1,0,2)].

phon = [[[n(saNkaryou,5,3,2),p_case(wa,1,0,1)],[v(i,1,0,2),v_flex(ru,1,0,1),no(N,1,
0,1),aux(desho,2,0,2),aux(u,1,0,1)]],s_prt(ka,1,0,2)].

phon = [interj(hai,2,1,2)].

phon = [[[n(touroku#hi,5,4,2),postp(toshi#te,3,1,2)],[[pre(o,1,0,11),n(hi#tori,3,2,
2)],[[num1_9(saN,2,0,1),man_unit1(maN,2,0,4),num1_9(go,1,0,1),sen_unit1(seN,2,0,4),
currency_jp(eN,2,1,3),p_case(ga,1,0,1)],[adj_n(hi#tsuyou,4,0,2),cop(desu#,2,0,1)]]]
]].

phon = [[[n(touroku#hi,5,4,2),postp(toshi#te,3,1,2)],[[pre(o,1,0,11),n(hi#tori,3,2,
2)],[[num1_9(saN,2,0,1),man_unit1(maN,2,0,4),num1_9(go,1,0,1),sen_unit1(seN,2,0,4),
currency_jp(eN,2,1,3)],[adj_n(hi#tsuyou,4,0,2),cop(desu#,2,0,1)]]]]].

phon = [[[n(touroku#hi,5,4,2),postp(toshi#te,3,1,2)],[[num1_9(saN,2,0,1),man_unit1(
maN,2,0,4),num1_9(go,1,0,1),sen_unit1(seN,2,0,4),currency_jp(eN,2,1,3),p_case(ga,1,
0,1)],[[pre(o,1,0,11),n(hi#tori,3,2,2)],[adj_n(hi#tsuyou,4,0,2),cop(desu#,2,0,1)]]]
]].

phon = [[[n(touroku#hi,5,4,2),postp(toshi#te,3,1,2)],[[num1_9(saN,2,0,1),man_unit1(
maN,2,0,4),num1_9(go,1,0,1),sen_unit1(seN,2,0,4),currency_jp(eN,2,1,3)],[[pre(o,1,0
,11),n(hi#tori,3,2,2)],[adj_n(hi#tsuyou,4,0,2),cop(desu#,2,0,1)]]]]].

phon = [[[num1_9(saN,2,0,1),man_unit1(maN,2,0,4),num1_9(go,1,0,1),sen_unit1(seN,2,0
,4),currency_jp(eN,2,1,3),p_case(ga,1,0,1)],[[n(touroku#hi,5,4,2),postp(toshi#te,3,
1,2)],[[pre(o,1,0,11),n(hi#tori,3,2,2)],[adj_n(hi#tsuyou,4,0,2),cop(desu#,2,0,1)]]]
]].

phon = [[[num1_9(saN,2,0,1),man_unit1(maN,2,0,4),num1_9(go,1,0,1),sen_unit1(seN,2,0
,4),currency_jp(eN,2,1,3)],[[n(touroku#hi,5,4,2),postp(toshi#te,3,1,2)],[[pre(o,1,0
,11),n(hi#tori,3,2,2)],[adj_n(hi#tsuyou,4,0,2),cop(desu#,2,0,1)]]]]].

phon = [[adv(sou,2,1,2),cop(desu#,2,0,1)],s_prt(ka,1,0,2)].

phon = [[[adv(doumo,3,1,2)],[v(arigatougozai,8,2,2),aux(mashi#,2,1,2),aux(ta,1,0,1)
]]].

phon = [[v(shi#tsurei,4,2,2),h_aux(itashi#,3,0,1),aux(masu#,2,1,2)]].
```