Group	: Computational Linguistics
Topic	: Rosetta3.software

Title	: **Syntax of the Rosetta Lexicon Language**
Author	: Joep Rous

Doc.Nr.	: 305
Date	: April 12, 1989
Status	: concept
Supersedes	: 270
Distribution	: Project
Clearance	: Project
Keywords	: Dictionary, testing

Institute for Perception Research

Contents

1 Introduction

In this document we will describe the syntax of the language which is to be used for specification of the Rosetta lexicons. First a short overview of the Rosetta dictionaries is given. In Chapter 3 some remarks are made on the lexicon compiler. In Chapter 4 the syntax of the dictionary file will be given, elucidated with some examples.

It is possible to specify constraints on the attribute values occurring in the lexicon. The syntax of the constraint specification notation is given in Chapter 5. The usage of the compiler is described in Chapter 6. Finally, in Chapter 7 it is explained how the keys that are used in the dictionary files can be defined[1].

2 Overview

2.1 FIXIDDICT

In Rosetta3 the FIXIDDICT lexicon is used for recognizing *fixed idioms* in the input sentence. It contains a list of all the fixed idioms that Rosetta3 must be able to deal with. Idioms can be treated as *fixed idioms* if they obey the following criteria:

1. The words of the idiom always occur in a fixed order (e.g. "by and large", "op en top").
2. No other words can occur in between the idiom words.
3. All idiom words, except the last one, are unchangeable, that is, not sensible for inflection and/or derivation. The last word of the idiom may take inflectional or derivational *suffixes*. (e.g. "kant en klaar" can be treated as a fixed idiom, but "heer des huizes" cannot be a fixed idiom)

If an idiom from the FIXIDDICT lexicon appears in the input sentence it is treated by the Rosetta3 morphological module as if it were one single word. Of course, the non-idiom reading is also tried by the system.

2.2 MDICT

The MDICT lexicon of Rosetta3 is used to make the transition from strings to fkeys and vice versa. It contains 5-tuples of the form:

```
<stem, FON, CC, AG, fkey>
```

[1] In the examples of chapter 4 I will use integer keys instead of the string keys which are introduced in chapter 6

```
Key: {(stem, fkey, FON)}
```

Here, FON specifies the phonetical information associated with the word, CC specifies the context condition and AG specifies whether the entry is only relevant for analysis, generation or for both.

(Notice: a stem may also be a fixed idiom, e.g. "by and large")

2.3 SDICT

After the analytical segmentation fkeys are translated into skeys. The fkey-skey mapping is specified in the SDICT lexicon. E.g. the lexicon contains information for translating the fkey of "lopen" into the skeys belonging to "lopen", "oplopen", "aflopen", etc.. This is accomplished by specifying a set of pairs <skey, fkey>. Moreover, with each pair <skey, fkey> a kind of condition on the context can be specified. This condition must be satified before the translation will be made. The context condition can be specified by means of a number of fkeys, of which the appearance in the input sentence is obligatory. In generation, the condition is not relevant because the M-grammar guarantees that it is satisfied by the M-grammar.

For efficiency reasons we give the fkey and skey of most of the <skey, fkey> pairs the same value. Only in cases where we have an 1 to n relation between fkey and skey their value will be different. In the implementation SDICT contains only these deviating <skey, fkey> pairs. If the morphological analyzer cannot find an fkey in SDICT it assumes that the corresponding skey has the same value (in generation the same holds for skeys and fkeys resp.).

2.4 BLEX

The BLEX dictionary contains, as it did in Rosetta2, pairs

```
< skey, basic S-tree record >
```

2.5 ILDICT

The ILDICT lexicon of Rosetta3 is used to translate skeys in the corresponding mkeys (= meaning keys). The structure of the lexicon entries is as follows:

```
<AG, skey, mkey, md, spref, mpref>

Key: {(skey,mkey)}
```

Here, *md* is a meaning description of the entry, *spref* specifies an order between entries with the same skey and *mpref* specifies an order between entries with the same mkey. The attribute *AG* indicates whether the current entry is to be used in analysis, in generation or in both.

2.6 SEMI-IDDICT

The SEMI-IDDICT lexicon defines a relation between basic expressions with a literal interpretation and their corresponding semi-idiomatic counterparts. The dictionary contains tuples of the form (cf. document r0269):

```
< skey-litteral, skey-argument, nr-argument, skey-semiidiom >

Key: {(skey-semiidiom), (skey-litteral, skey-argument, nr-argument)}
```

2.7 IDDICT

The IDDICT lexicon defines a relation between a sequence of basic expressions which are used in the idiom and a basic expression for the idiom as a whole. The dictionary contains tuples of the form:

```
< skey-sequence, idiom-pattern, skey-idiom >

Key: {(skey-sequence, idiom-pattern), (idiom-pattern, skey-idiom)}
```

3 The dictionary compiler

The 7 lexicons that were mentioned in the previous sections should all be filled in a consistent way, e.g. if an idiom is present in FIXIDDICT it also should appear in MDICT, each BLEX entry should have at least one corresponding MDICT entry, etc.. It seemed to us that the linguist filling the lexicon would get a better overview if (s)he could specify all information for all lexicons in one single dictionary in a clear way.

The idea is that each dictionary entry is specified according to a fixed syntax, which allows the user to specify in each dictionary-entry information for each of the 7 Rosetta3 lexicons. The NEWDICTGEN compiler checks this and extracts from each entry specification the information needed to add one or more entries to the Rosetta3 lexicons.

One can look upon this process as a kind of normalization action that is performed by the dictionary compiler.

The dictionary will contain attribute values of which the type has been defined in Domain T. The compiler checks whether these attribute values are compatible with the current Domain. For that purpose the domain compiler generates a number of PASCAL modules which are able to convert attribute values of type STRING to the corresponding values which are defined in the PASCAL version of Domain T. If this conversion fails the compiler knows that it is an erroneous attribute value.

This first version of the compiler is not very subtle in the sense that it does not perform any consistency checks as mentioned in the first alinea of this section. That is, however, no fundamental problem but merely a lack of time. The consistency checks can easily be incorporated in the compiler in the future.

4 The dictionary syntax

At the most global level the syntax of the dictionary is:

Dictionary :: { Entry-part } "@"
Entry-part :: MDICT-part ":" SDICT-part [":" SEMI-IDDICT-part]
["":" IDDICT-part] ":" ILDICT-part ":" BLEX-part ";"

4.1 The MDict part

The syntax of the MDICT-part is as follows:

MDICT-part :: [[AG switch] string-part [fon-spec] [CC-spec]]
fon-spec :: "<" identifier { "," identifier } ">"
CC-spec :: identifier
AG-switch :: "/" ("A" | "G" | "AG")

The *MDICT-part* is optional because it must be possible to specify an abstract BLEX entry. The *string-part* in an *MDICT-part* specification may not contain blanks and the string must at least contain one non-blank character. If the string is a fixed idiom, the words of the idiom should be separated by an underscore symbol "_". At one place in the idiom the character "#" may appear. This character is a directive for the compiler not to put the complete idiom in the FIXIDDICT lexicon but just the idiom string up to the "#" character. A reserved character, e.g. ":" , "/" or "#", in a string should be preceded by a backslash "\" character.

In the *fon-spec* part the attribute values according to the phonetical information should be specified.

The *AGswitch* part specifies whether the string is relevant for Analysis (=/**A**), Generation (=/**G**) or both (=/**AG**).

Example :

```
Duitsland               : .....................
actrice < true, false > : .....................
op_en_top               : .....................
kant_en_kla#ar          : .....................

an      CCvowel         : .....................
a       CCcons          : .....................
```

4.2 The SDict part

The syntax of the SDICT-part is:

SDICT-part :: skey ["," fkey [context-set]]
context-set :: "<" cckey { "," cckey } ">"
cckey :: dictionary-key
skey :: dictionary-key
fkey :: dictionary-key

In the Rosetta3 system the value of an fkey is by default the same as the value of the corresponding skey. This fact is expressed in the above syntax specification, if the fkey is omitted the compiler assumes that it has the same value as the skey. Only in the case that an fkey corresponds with a number of skeys it is allowed to specify the fkey. If the compiler discovers that an fkey is specified it takes the MDICT-part specification of the entry as comment. Therefore, the fkey in these entries should refer to an entry that does have a correct MDICT-part. E.g. the verbs "lopen", "aflopen", "oplopen" could have the following entries:

```
loop    : 525      : ..........
loop_af : 526, 525 : ..........
loop_op : 527, 525 : ..........
```

In this example, the stem "loop" corresponds with fkey and skey 525. However, the stem of the verb "aflopen" is also "loop" and corresponds with fkey 525, whereas the skey is 526. The verb "oplopen" corresponds with fkey 525 and skey 527.

In the above example we could also specify conditions on the context, because the particle "af" should be present if fkey 525 is to be translated in skey 526 . Suppose the fkey value of "af" is 201 and that of "op" is 604 then the entries could also look like:

```
loop    : 525               : ..........
loop_af : 526, 525 < 201 > : ..........
loop_op : 527, 525 < 604 > : ..........
```

Notice that the strings "loop_af" and "loop_op" will not be found in MDICT because the MDICT-parts of these two entries will be viewed as comment.

Verbs with a particle, like "afkondigen" and "aankondigen", of which the form without the particle ("kondigen") does not exist, should be specified as follows:

```
kondigen{_af}  : 932, 932 <201> : ...........
kondigen_aan   : 933, 932 <620> : ...........
```

The entry for the verb "afkondigen" is now used to define the fkey and the stem of the verb (therefore "_af" must placed between comment brackets).

4.3 The Semi-IdDict part

The syntax of the Semi-IdDict-part is:

Semi-IdDict-part :: sidkey-specs
sidkey-specs :: sidkey-spec { "," sidkey-spec }
sidkey-spec :: "#"arg-nr "=" arg-skey sid-skey sid-mkey
arg-nr :: dictionary-key
arg-skey :: dictionary-key
sid-skey :: dictionary-key
sid-mkey :: dictionary-key

It is obvious that the semi-idiom part has to be an optional part of a lexicon entry. This fact has already been expressed by the definition of the *Entry-part* (the first definition of section 4). The *arg-nr* part specifies the argument position of the semi-idiom argument, whereas the *arg-skey* part specifies the skey of the argument. For each semi-idiom a specific skey is introduced. This skey is specified in the *sid-skey* part. The corresponding mkey is specified in the *sid-mkey* part.

Examples:

```
les        : 234 : ..............;
voordracht : 745 : ..............;
geef       : 23  : #2=234 567 6789,
                   #2=745 890 2332 : ................;
```

4.4 The IdDict part

The syntax of the IdDict-part is:

IdDict-part :: id-specs
id-specs :: id-spec { "," id-spec }
id-spec :: skey-seq idpattern-set id-skey id-mkey { id-mkey }
skey-seq :: "<" arg-skey { arg-skey } ">"
idpattern-set :: "[" idpattern { idpattern } "]"
arg-skey :: dictionary-key
id-skey :: dictionary-key
id-mkey :: dictionary-key
idpattern :: identifier

The idiom part is an optional part of a lexicon entry as can be deduced from the definition of the *Entry-part* (the first definition of section 4). An idiom specification consists of a sequence of skeys which occur in the idiom (the *skey-seq* part), the set of idiom patterns which define the ways in which the idiom can be used in a sentence (the *idpattern-set* part), one specific skey for the idiom (*the id-skey*) and its corresponding mkey(s) (*id-mkey*).

`Examples:`

```
pijp       : 234 : ..............;
Maarten    : 745 : ..............;
aan        : 1256: ..............;
geef       : 23  : <23 234 1256 745> [ synidpat1 synidpat2 ] 23567 789
                 : ..............;
```

4.5 The ILDict part

The syntax of the ILDICT-part is:

ILDICT-part :: [mkey-specs]
mkey-specs :: mkey-spec { "," mkey-spec }
mkey-spec :: mkey ["A"|"G"] ["s" pref] ["m" pref] [""" md """] [info]
mkey :: dictionary-key
pref :: number
md :: string
info :: "<" string ">"

The ILDICT-part is optional because certain basic expressions are introduced syncategorematically in the M-grammar. By means of the of the attribute "A" or "G" it can be specified whether the translation is relevant for analysis or for generation only. If the attribute is omitted the translation will be possible in both analysis and generation. The compiler uses the default value 0 if the value of the preference bonusses (*pref*) is omitted. The "m" preference bonus is used to distinguish between the alternatives during the mkey → skey translation, whereas the "s" preference bonus is used to order alternatives during the skey → mkey translation.

The *info* part can be used to specify whether this mkey translation is meant for a specific target language. For instance, if the mkey is meant for a translation from Dutch to English, the info part will probably be: < DE >. The lexicon coordinator will define the strings that can be used inside the *info* part. If no *info* part has been specified it is assumed that the translation is meant for all possible target languages.

Examples:

```
   loop : 525 : 10987 s1 "gaan"       ,
                10988 s2 "hard-lopen"
                : BVERB(.........);
   loop : 788 : 12345 s1 "het lopen",
                54321 s2 "van geweer"
                : BNOUN(.........);
```

4.6 The BLex part

The syntax of the BLEX-part is:

BLEX-part :: [category "(" [attrvalue { "," attrvalue }] ")"]

The order and values of the attribute in the BLEX-part specification should be according to DOMAIN-T.

Example:

```
welk :176: 32400 :
         BDET (
         {req:}             omegapol,
         {possnietnp:}      false,
         {definite:}        indef,
         {posspred:}        false,
         {possnumbers:}     [singular, plural],
         {posscomas:}       [count, mass],
         {detnpmood:}       wh,
         {eFormation:}      RegEformation,
         {enFormation:}     false);
vulkaan :207: 457890 : BNOUN (
         {req:}              omegapol,
         {pluralforms:}      [enPlural],
         {genders:}          [mascgender],  {*morph* ++}
         {class:}            omegaTimeAdvClass,
         {deixis:}           omegadeixis,
         {aspect:}           omegaAspect,
```

```
{retro:}          false,
{sexes:}          [],                    {9/3}
{subcs:}          [othernoun],
{temporal:}       false,
{possgeni:}       false,
{animate:}        inanimate,
{human:}          nohuman,               {9/3}
{posscomas:}      [count],
{thetanp:}        omegathetanp,
{nounpatterns:}   [],
{prepkey:}        0,
{personal:}       true);
```

The BLEX-part is optional because it is possible that several MDICT entries correspond with one and the same BLEX entry. In these cases the BLEX-part should be specified in one of the dictionary entries, e.g. the determiners "a" and "an" in ENGLISH should be specified as:

```
a   CCcons   : 512 : :DET();
an  CCvowel  : 512 : :;
```

5 Lexicon constraints

The lexicon constraints must be specified in a separate file with the name **constraints.constr**. In the file only constraints on the level of records may be specified, that is, dependencies between attributes or sets of attributes within one record. The syntax of the constraint specification is as follows:

constraint-spec :: { cat-section } "@"
cat-section :: "<" category { clause } ">"
clause :: ":" bool-expression errormessage
errormessage :: [peculiar] "”" string "”"
peculiar :: "$P"

The file **constraints.constr** may contain for each lexical category one *cat-section* in which the constraints for the records corresponding that syntactic *category* can be specified. Such a specification may consist of several *clauses*. Each *clause* contains a boolean expression *bool-expression* on the set of attributes of the current category and an *errormessage*. The boolean expression is evaluated for each record of syntactic category *category* in the lexicon. If the evaluation yields the value "FALSE" then the constraint is violated and the specified *errormessage* is printed by the lexicon compiler. The errormessage can be preceded by the *peculiar* symbol. The expressions followed by the peculiar symbol are not interpreted as real constraints but are used to inform

the user of lexical entries with a peculiar record value. Normal constraint violation messages are preceded by the string "WARNING", peculiar messages, however, are preceded by "PECULIAR". Constraint conflicts are not interpreted as errors by the compiler, they will only cause warning messages.

The syntax of the boolean expression is equivalent to the PASCAL syntax of boolean expression, that is, operators like 'AND', 'OR' and 'NOT' can be used. Furthermore, there are two predefined boolean functions, 'IMPLIES' and 'IFF' each taking two parameters of type boolean. The semantics of these functions is as follows::

IMLPLIES$(a, b) \equiv \neg a \vee b$
IFF$(a, b) \equiv (a \wedge b) \vee (\neg a \wedge \neg b)$

```
Example

< BVERB
:  IMPLIES((particle <> 0), (verbraiser = noVR))
      "Particle and verbraiser mutually exclusive"
:  IMPLIES(((prepkey1 <> 0) AND
              (verbraiser IN [optionalVR, obligatoryVR])),
              false)
      $P "verbraisers take no prepositional objects"
>

< BNOUN
:  IMPLIES((human = yeshuman),(animate = yesanimate))
      "yeshuman implies yesanimate"
>
```

6 Compiler usage

At the moment the lexicon is divided into a number of sub-lexicons. Each sub-lexicon contains entries with a specific syntactical category, i.e.:

```
bnoun.dict
bverb.dict
badv.dict
badj.dict
prep.dict
particle.dict
pronoun.dict
conj.dict
misc.dict
```

```
  auxverb.dict
```

Each of these sub-lexicons must obey the syntax which is defined in the previous sections of this document.

The correctness of the sub-lexicons, that is, the correctness according to the lexicon syntax and the correctness according to DOMAIN-T, can be checked by means of the following command:

```
   lbuild dutch:blexbnoun.isf
or
   sbuild english:blexprep.isf
etc.
```

The final Rosetta3 lexicons (that is, blex, fixid, sdict, siddict, iddict, mdict and ildict) for a specific language can be created by entering one single command, as follows:

```
   lbuild dutch:blex.isf
```

7 Key definitions

The syntax definitions of the previous sections specified that the keys of the lexicons should be *dictionary-keys*. In this section we will give a definition of the the term dictionary-key.

In order to improve the readability of the dictionary we allow dictionary keys to be of type string preceded by a dollar character '$':

dictionary-key :: ("$" string) | integer

(To be as flexible as possible, we also allow the user to specify the internal integer dictionary key value. This, however, may cause errors because the integer value may already be used by the dictionary compiler itself.)

Examples:

```
   loop : $s_loop_verb : $m_loop_gaan s1 "gaan"       ,
                         $m_loop_hard s2 "hard-lopen"
                : BVERB(.........);
   loop : $s_loop_noun : $m_het_lopen s1 "het lopen",
                         $m_loop_v_geweer s2 "van geweer"
                : BNOUN(.........);
```

There will be 3 types of dictionary keys:

1. dictionary-*fkeys*
2. dictionary-*skeys*
3. dictionary-*mkeys*

All fkeys and skeys should be specified in file **language:skeydef.kdf** and mkeys in **interlingua:mkeydef.kdf**. The syntax of these key definition files is as follows:

```
KeyDefinition :: { string } "@"
string        :: (letter | cypher | "_") { letter | cypher | "_" }
```

8 Conclusions

The dictionary approach presented in this document has certainly some disadvantages. It has not the userfriendly userinterface of the Editor-Generator we previously had in mind, instead, the standard text editor must be used. The approach is not intended for a dictionary with a large number of entries. If the number of entries in the dictionary grows, the entry look-up speed for the user will decrease .

An advantage is that we have no serious B-LEX conversion problems (cf. document 196) . If DOMAIN-T is altered in such a way that it is incompatible with the dictionary it will not be possible to integrate it. Because the dictionaries depend on the DOMAIN-T file the RBS-system will try to generate new Rosetta3 dictionaries if DOMAIN-T changes. This will fail, however, if they are incompatible. The user can only put his stuff into the archive if he also updates the lexicons.