|  |  |
|---|---|
| Group | : Computational Linguistics |
| Topic | : Rosetta3.formalism |

|  |  |
|---|---|
| Title | : **Syntax and Semantics of the Rosetta3 Segmentation Rules** |
| Author | : J. Rous |

|  |  |
|---|---|
| Doc.Nr. | : 265 |
| Date | : 900124 |
| Status | : concept |
| Supersedes | : - |
| Distribution | : Project |
| Clearance | : Project |
| Keywords | : Morphology, segmentation rules, syntax, semantics |

## Contents

## 1 Introduction

This document contains a definition of the syntax and semantics of the segmentation rules. It is strongly recommended to read document nr. 210 before an attempt is made to read this document.

On a global level the syntax of the segmentation rules is the same for all the different kind of rules. In section 2, 3 and 4 a description can be found of the global syntax definitions. Section 4.1 to 4.5 give a detailed specification of the syntax and semantics of each rule type.

The definitions in this document contain quantifications over variables. In each of these definitions a specification of the domain of the quantified variable has been omitted. Instead the a naming convention has been used. Variables $\alpha_1$, $b$ are of type string. Variable $\kappa$ is either a prefix key or a suffix key and variable $m$ is a natural number.

## 2 General

The formalism prescribes 5 different segmentation rule types, namely, left, right and middle glue rules and left (prefix) and right (suffix) segmentation rules.

A complete specification of all rules of one type consists of three parts, one part in which 'variables' are declared, another part in which 'alias' names can be defined for the variables and a third part in which all rules are specified.

**rule specification** :: [ 'VAR' **variable definition part** ]
[ 'ALIAS' **alias definition part** ]
'TABLE' **rule definition part** 'END'

## 3 Variable and Alias definition part

The syntax of the variable definition part and the alias definition part is the same for all rule types. A variable definition introduces a variable name and the set of ( string ) values that the variable can take ( the domain of the variable ).

**variable definition part** :: { **variable name** '=' '[' **variable value**
{ ',' **variable value** } [ ',' ] ']' ';' }
**variable value** :: **single string** | ( **left string** <> **right string** )
**left string** :: **string**
**right string** :: **string**
**single string** :: **string**

Example:

    Vowels = [ a, o, u, e, i, a<>A, o<>O, u<>U, e<>E, i<>I ];

The meaning of left string, right string and single string will be explained in the next section. A single string is in fact an abbreviation for a left and right string with the same value. For example, 'ee' is interpreted as 'ee<>ee'. The empty string is also a possible variable value. In case the empty string is one of the possible variable values the closing bracket ']' of the variable definition must be preceded by a comma ','.

An alias definition introduces a set of alternative names for a specified variable. These names can be used instead of the variable name. The alias definition part has been introduced in order to increase the readability of the segmentation rules.

**alias definition part** :: { **variable name** '=' **string** ',' { **string** } ';' }

Example:

    Vowels = LeftVowel, RightVowel;

## 4 Rule definition part

On a global level all rules in the rule definition part of each type are of the same structure, that is:

**rule definition part** :: **rule definition** { **rule definition** }
**rule definition** :: **lefthand side** '::' **righthand side** ';'

The specification of the **left hand side** ( to be called **lhs** ) and **right hand side** ( to be called **rhs** ) will be different for each rule type. In the following sections a more detailed specification will be given of the **rule definition** for each type of rule.

All rule types have in common that the **lhs**-part contains *at least* one and the **rhs**-part contains *exactly* one so-called M-string ( the M- is an abbreviation for match- ). An M-string is a (possibly empty) string consisting of non-blank characters and 'variables'. A 'variable' in an M-string is specified by the variable name enclosed by an open '(' and closing ')' bracket. The string *a(K1)bc* is an example of an M-string in which the variable *'K1'* is used. Each variable which appears in the **lhs** of a rule must also appear in the **rhs** of a rule and vice versa.

During a rule application the input-string is matched with the M-strings of a rule. If the match is successful for all M-strings in the rule the rule application will be successful. During the process of matching, all variables in each M-string of the rule ( both the **lhs** and **rhs** M-strings ) are instantiated with each value of their domain. More formally, the semantics of a rule application can be defined by means of the function $\Phi$:

$$\Phi(\alpha) =_{def} \{r | \exists v_1, \ldots, v_n, d_1, \ldots, d_n : \\ \forall i : 1 \leq i \leq n : v_i \in VARS(\Phi) \wedge d_i \in D^{v_i} \wedge r \in R(\alpha, d_1, \ldots, d_n)\}$$

Here, parameter $\alpha$ contains the input string that has to be matched. The type of the result $r$ differs for the different type of rules. Also the analytical and generative interpretation of the rules will require a different type for $r$ as we will see in the next sections. The values $d_i$ of domain $D^{v_i}$ of variable $v_i$ are in fact left and right string pairs $< l_i, r_i >$. The actual rule is defined as the function $R$ which substitutes value $r_i$ for each occurence of the variable $v_i$ in the **rhs** M-string and $l_i$ for each occurence of the variable $v_i$ in the **lhs** M-strings of the rule before it starts the matching process. Each rule type has its own definition of the function $R$. These definitions will be given in the next sections.

It is obvious from the preceding definition that a segmentation or glue rule in fact defines a set of rules. The cardinality of this rule set depends on the size of the domains of the variables, as follows:

$$\prod_{i=1}^{n} \#D^{v_i}$$

This formula expresses the threat of a combinatorial explosion, e.g. a rule with 3 different variables, each having a domain of 10 elements in fact defines $10^3$ different rules. In the section on implementation of these rules I will describe how this danger is averted.

## 4.1 Left segmentation rules

A left segmentation rule has one of the next two forms:

**rule definition** :: **perfect match** | **left match**
**perfect match** :: **PF-K** '+' **q** '::' **p** [',' **F**] ';'
**left match** :: **PF-K** '+' **q**'*' '::' **p**'*' [',' **F**] ';'

Here, PF-K is a prefix key, $p$ and $q$ are M-strings and $F$ is the identification of a phonetical rule.

Example:

```
    VAR
        A = [a, b, c, d, f, g, h, j, k, l, m, n, o, p, q, r, s, t, v, w, x, y, z];
    TABLE
        PFKge + (A)* :: ge(A)*;
        PFKge + e* :: geë*;
```

The difference between the left match and the perfect match ruletype is that the first ruletype requires an exact match between the M-string $p$ and the input string [1],

[1]in generation the match will be with $q$

whereas the second type requires that $p$ matches with the left part of the input string.

In the previous section we have seen that the semantics of the rule can be defined by means of the function $\Phi$. I will define now the function $R$ which was used for the definition of $\Phi$ for each of the two forms of a left segmentation rule and also for the analytical and generative version of each form. The semantics associated with the analytical version of the **left match** rule is:

$$
\begin{aligned}
&R_l^A(\alpha, < l_1, r_1 >, \ldots, < l_n, r_n >) =_{def} \\
&\qquad \{r | \exists \alpha_1, b : \alpha = \alpha_1 \bullet b \wedge \alpha_1 \equiv p\{r_1/v_1, \ldots, r_n/v_n\} \wedge \neg(b = \epsilon \wedge p = \epsilon) \wedge r =< PFK, F, q \bullet b\{l_1/v_1, \ldots, l_n/v_n\} >\}
\end{aligned}
$$

In this definition $p$, $q$, $PFK$ and $F$ are rule-defined constants. They correspond with $p$, $q$, $PFK$ and $F$ in the syntax definition. The notation $p\{r_1/v_1, \ldots, r_n/v_n\}$ means "the string $p$ in which for each occurence of the variable $v_i$ the value $r_i$ has been substituted. The '$\bullet$' symbol is used to express a concatenation operation between two strings.

The semantics associated with the analytical version of the perfect match rule ( indicated by subscript $p$ ) is:

$$
\begin{aligned}
&R_p^A(\alpha, < l_1, r_1 >, \ldots, < l_n, r_n >) =_{def} \\
&\qquad \{r | \alpha \equiv p\{r_1/v_1, \ldots, r_n/v_n\} \wedge \\
&\qquad r =< PFK, F, \{l_1/v_1, \ldots, l_n/v_n\} >\}
\end{aligned}
$$

In generation the semantics of the two rule forms is as follows:

$$
\begin{aligned}
&R_l^G(\alpha, < l_1, r_1 >, \ldots, < l_n, r_n >) =_{def} \\
&\qquad \{r | \exists \alpha_1, b, \kappa : \alpha =< \kappa, \alpha_1 \bullet b > \wedge \alpha_1 \equiv q\{l_1/v_1, \ldots, l_n/v_n\} \wedge \\
&\qquad \neg(b = \epsilon \wedge q = \epsilon) \wedge \kappa = PFK \wedge r =< F, p \bullet b\{r_1/v_1, \ldots, r_n/v_n\} >\}
\end{aligned}
$$

$$
\begin{aligned}
&R_p^G(\alpha, < l_1, r_1 >, \ldots, < l_n, r_n >) =_{def} \\
&\qquad \{r | \exists \alpha_1, \kappa : \alpha =< \kappa, \alpha_1 \wedge \alpha_1 \equiv q\{l_1/v_1, \ldots, l_n/v_n\} \wedge \\
&\qquad \kappa = PFK \wedge r =< F, p\{r_1/v_1, \ldots, r_n/v_n\} >\}
\end{aligned}
$$

## 4.2 Right segmentation rules

A right segmentation rule has one of the next two forms:

**rule definition** :: **perfect match** | **right match**
**perfect match** :: **q** '+' **SF-K** '::' **p** [',' **F**] [, **CC**] ';'
**right match** :: '*'**q** '+' **SF-K** '::' '*'**p** [',' **F**] [, **CC**] ';'

Here, SF-K is a suffix key, $p$ and $q$ are M-strings, F is the identification of a phonetical rule and CC is the identification of a context condition.

The difference between these two kind of rules is of the same type as the difference between the two kinds of left segmentation rules.

Example:

```
    VAR
        C = [b, c, d, f, g, h, j, k, l, m, n, p, q, r, s, t, v, w, x, y, z];
        V = [a, e, i, o, u];
    TABLE
        *(C)(V)b + SFKje :: *(C)(V)bbetje, FONleegsjwa;
```

The definition of the analytical right match and perfect match semantics are:

$$\begin{aligned}&R_r^A(\alpha, < l_1, r_1 >, \ldots, < l_n, r_n >) =_{def}\\ &\quad \{r | \exists \alpha_1, b : \alpha = b \bullet \alpha_1 \wedge \alpha_1 \equiv p\{r_1/v_1, \ldots, r_n/v_n\} \wedge\\ &\quad \neg(b = \epsilon \wedge p = \epsilon) \wedge r =< SFK, F, CC, b \bullet q\{l_1/v_1, \ldots, l_n/v_n\} >\}\end{aligned}$$

$$\begin{aligned}&R_p^A(\alpha, < l_1, r_1 >, \ldots, < l_n, r_n >) =_{def}\\ &\quad \{r | \alpha \equiv p\{r_1/v_1, \ldots, r_n/v_n\} \wedge\\ &\quad r =< SFK, F, CC, q\{l_1/v_1, \ldots, l_n/v_n\} >\}\end{aligned}$$

The definition of the generative right match and perfect match semantics are:

$$\begin{aligned}&R_r^G(\alpha, < l_1, r_1 >, \ldots, < l_n, r_n >) =_{def}\\ &\quad \{r | \exists \alpha_1, b, \kappa : \alpha =< \kappa, b \bullet \alpha_1 > \wedge \alpha_1 \equiv q\{l_1/v_1, \ldots, l_n/v_n\} \wedge\\ &\quad \neg(b = \epsilon \wedge p = \epsilon) \wedge \kappa = SFK \wedge r =< F, CC, b \bullet p\{r_1/v_1, \ldots, r_n/v_n\} >\}\end{aligned}$$

$$\begin{aligned}&R_p^G(\alpha, < l_1, r_1 >, \ldots, < l_n, r_n >) =_{def}\\ &\quad \{r | \exists \alpha_1, \kappa : \alpha =< \kappa, alpha_1 > \wedge \alpha_1 \equiv q\{l_1/v_1, \ldots, l_n/v_n\} \wedge\\ &\quad \kappa = SFK \wedge r =< F, CC, p\{r_1/v_1, \ldots, r_n/v_n\} >\}\end{aligned}$$

## 4.3 Right glue rules

A right glue rule has the following form:

**rule definition** :: '*'$\mathbf{q}_1$ { '+' $\mathbf{q}_i$ $\}_{i=2}^{M}$ '::' '*'**p** ';'

Here, $p$ and $q_1, \ldots, q_M$ are M-strings. Notice that one cannot express the requirement for a perfect match with right glue rules.

Example:

```
    TABLE
        * + will :: *'ll;
        * + had :: *'d;
        * + would :: *'d;
```

The semantics of the analytical and generative rule-application are:

$R^A(\alpha, < l_1, r_1 >, \ldots, < l_n, r_n >) =_{def}$
$\quad \{r | \exists \alpha_1, b : \alpha = b \bullet \alpha_1 \wedge \alpha_1 \equiv p\{r_1/v_1, \ldots, r_n/v_n\} \wedge$
$\quad r =< M, < b \bullet q_1, \ldots, q_M > \{l_1/v_1, \ldots, l_n/v_n\} >\}$

$R^G(\alpha, < l_1, r_1 >, \ldots, < l_n, r_n >) =_{def}$
$\quad \{r | \exists \alpha_1, b, m : \alpha =< m, b \bullet \alpha_1 > \wedge$
$\quad \alpha_1 \equiv q_1 \bullet "|" \bullet \ldots \bullet "|" \bullet q_M\{l_1/v_1, \ldots, l_n/v_n\} \wedge$
$\quad m = M \wedge r = b \bullet p\{r_1/v_1, \ldots, r_n/v_n\}\}$

For implementation reasons the input parameter $\alpha$ of the generative version of the function $R$ is not a tuple of strings. Instead, the strings have been concatenated to one big string in which the individual strings are separated by the '|' character. In order to perform the matching process the M-strings in the lefthand side of the rule must be concatenated in the same way.

## 4.4 Left glue rules

A left glue rule has the following form:

**rule definition** :: $\{$ $\mathbf{q}_i$ '+' $\}_{i=1}^{M-1}$ $\mathbf{q}_M$'*' '::' $\mathbf{p}$'*' ';'

Here, $p$ and $q_1, \ldots, q_M$ are M-strings. Notice that one cannot express the requirement for a perfect match with left glue rules.

Example:
TABLE
tegen + * :: tegen*;
tegenover + * :: tegenover*;
tekeer + * :: tekeer*;

The semantics of the analytical and generative rule-application are:

$R^A(\alpha, < l_1, r_1 >, \ldots, < l_n, r_n >) =_{def}$
$\quad \{r | \exists \alpha_1, b : \alpha = \alpha_1 \bullet b \wedge \alpha_1 \equiv p\{r_1/v_1, \ldots, r_n/v_n\} \wedge$
$\quad r =< M, < q_1, \ldots, q_M \bullet b > \{l_1/v_1, \ldots, l_n/v_n\} >\}$

$R^G(\alpha, d_1, \ldots, d_n) =_{def}$
$\quad \{r | \exists \alpha_1, b, m : \alpha =< m, \alpha_1 \bullet b > \wedge$
$\quad \alpha_1 \equiv q_1 \bullet "|" \bullet \ldots \bullet "|" \bullet q_M\{l_1/v_1, \ldots, l_n/v_n\} \wedge$
$\quad m = M \wedge r = p \bullet b\{r_1/v_1, \ldots, r_n/v_n\}\}$

## 4.5 Middle glue rules

A middle glue rule has the following form:

**rule definition** :: $\{$ $\mathbf{q}_i$ '+' $\}_{i=1}^{M-1}$ $\mathbf{q}_M$ '::' $\mathbf{p}$ ';'

Here, $p$ and $q_1, \ldots, q_M$ are M-strings. Notice that one can only express the requirement for a perfect match with these rules.

Example:

```
    VAR        A = [se,te,me,le,lo,la,os,nos,les,los,las];
           B = [se,te,me,os,nos];
           C = [te,me,le,lo,la,os,nos,les,los,las];
    TABLE
           ten + (A) :: ten(A);
           se + (B) + (C) :: se(B)(C);
```

The semantics of the analytical and generative rule-application are:

$R^A(\alpha, < l_1, r_1 >, \ldots, < l_n, r_n >) =_{def}$
$\quad \{r | \alpha \equiv p\{r_1/v_1, \ldots, r_n/v_n\} \wedge$
$\quad r =< M, < q_1, \ldots, q_M > \{l_1/v_1, \ldots, l_n/v_n\} >\}$

$R^G(\alpha, < l_1, r_1 >, \ldots, < l_n, r_n >) =_{def}$
$\quad \{r | \exists m : \alpha =< m, \alpha_1 > \wedge$
$\quad \alpha_1 \equiv q_1 \bullet "|" \bullet \ldots \bullet "|" \bullet q_M\{l_1/v_1, \ldots, l_n/v_n\} \wedge$
$\quad m = M \wedge r = p\{r_1/v_1, \ldots, r_n/v_n\}\}$