# A Textual Rewriting system for NLP

José João Dias de Almeida, Alberto Simões

Universidade Do Minho Largo do Paço 4704-553 Braga-Portugal
{jj, ambs}@di.uminho.pt

## Abstract

In this document we describe the use of Text::RewriteRules in several tasks related to Speech.

## 1.Introduction

In this document we will present a textual rewriting system tool - Text::RewriteRules[3] - and discuss its use for several tasks related to Natural Language Processing specially for tasks related to TTS.
Some of the examples presented are included in the tool Lingua::PT::Speaker[4,5] - a Perl module that implements a TTS for Mbrola[1]. During the presentation, a demonstration of Portuguese synthesis with Lingua::PT::Speaker will be done. Both Text::RewriteRules and Lingua::PT::Speaker are available from CPAN[9].
In order to completely understand the examples, some knowledge of Perl programming language is needed. In order to understand the rules, knowledge of regular expression is enough (see also ). Also, some knowledge of SAMPA[2] phonetic alphabet can help understanding rules.
We will start with a description of the rewriting system (section). After that, we will take a *rewriting system by example* approach (sections and ).

## 2.Short presentation of Text::RewriteRules

Text::RewriteRules is a Perl embedded Domain Specific Language that helps writing rules to process text. Basically, Text::RewriteRules is used to include rule-sets in a Perl program:

- each set of rules generates an independent Perl function;
- each rewriting step is a substitution:
    - either a Perl regular expression substitution;
    - or a substitution by the result of an expression.

### 2.1 "Hello world"-like example

The following example performs the expansion of some informal English constructions:

```
use Text::RewriteRules;

RULES formalizer
don't==>do not
doesn't==>does not
ENDRULES

while(<>) {
  print formalizer($_)
}
```

### 2.2 Rule types

Text::RewriteRules support several types of rules. The most relevant are:

- simple string substitution,

  regexp ==> substitution string

- conditional substitutions,

  regexp ==> substitution string !! condition

- rules with code evaluation (with optional conditions),

  regexp =e=> Perl expression
  regexp =e=> Perl expression !! condition

- initialization rules (begin rule),

  =b=> Perl initialization statement

### 2.3 Basic rewriting algorithms

There are two kind of rule-set:

- Point fix rewriting system (RULES f ... ENDRULES), that performs all possible substitutions until no rule can be applied;
- Sliding rewriting system (RULES/m f ... ENDRULES) that performs substitutions at a cursor position, that slides from the beginning to the end of the text.

## 3.Text::RewriteRules by example

### 3.1 PT Syllable

In the following example we define three rewriting systems:

- **divide** - separate words in syllable (regato - re | ga | to)
- **syllableaccent** - marks the tonic syllable (re | ga | to - re | "ga | to)
- **vowelaccent** - marks the tonic vowel (re | "ga | to - re | ga: | to)

```
$v  = qr{[aeiouáéíóúàãõâêôäëïöüyw]}i;          ## vowels
$c  = qr{[bcçdfghjklmñnpqrstvwyxz]}i;          ## consonant
$f  = qr{(?:[çdfgjkqtv]|ss|rr|[nlcs]h|bs|cc)}i;  ## strong consonant
$ac = qr{[áéíóúãõâêô]}i;                       ## accents
@br = qw{sl sm sn sc rn bc lr bc bd bj bp pt pc dj pç zm tp};

while(<>){
  s/(\w+)/vowelaccent(syllableaccent(divide($1)))/ge; }

RULES/i divide
($v)($c)($c)($v)==>$1$2|$3$4!! $br{"$2$3"}
($v)($c)($c)([lr])==>$1$2|$3$4!! $br{"$2$3"}
($v)($f)(?![l])==>$1|$2
(.[bclnprsx])($f)(?![l])==>$1|$2
($v)([bclmnprsxzh]$v)==>$1|$2
(\w)([lmnrsx])([bclmnprszx])($v)==>$1$2|$3$4!!"$2$3" ne "ss" && "$2$3" ne rr
($v|[lmnrsx])([bcp][lr]$v)==>$1|$2
($v)($c)($c)($v)==>$1|$2$3$4

([a])(i[ru])==>$1|$2              ##  quebra de ditongos / tritongos
([ioeê])([aoe])==>$1|$2
u(ai|ou)==>u|$1
([^qg]u)(ei|iu|ir|$ac)==>$1|$2
([aeio])($ac)==>$1|$2
([íúô])($v)==>$1|$2
ENDRULES
```

```
RULES/i syllableaccent            ##   mark the tonic syllable
"=last=>                       ## leave if we have a  tonic syllable
(\w*$ac)==>"$1                 ## mark a syllable when it has accent character
(\w*([zlr]|[iu]s?))$==>"$1     ## mark last syllable if ...
(\w+\|\w+)$==>"$1
(^\w+$)==>"$1
ENDRULES

RULES/i vowelaccent            ## mark the tonic vowel in the tonic syllable
:($ac)==>$1:
"(\w*?($v|[yw]))==>$1:
([gq]u):($v|[yw])==>$1$2:
"==>
:=last=>
ENDRULES
```

## 3.2 Numbers to words

The following rules convert numbers to words.

```
%fixnum=( 0=> "zero", 1=> "um", 2=> "dois", 3=> "três",...
        10=> "dez", 11=> "onze", 12=> "doze", ...
        20=> "vinte", 30=> "trinta", 40=> "quarenta", ...
        100=> "cem", 200=> "duzentos", 300=> "trezentos",...
        1000=> "mil", 1000000=> "um milhão");

RULES num2words
(\d+)[Ee](-?\d+)==>$1 vezes 10 levantado a $2    ## 12E-24    scientific
-(\d+)==>menos $1                          ## -34       negatives
(\d+)\s*\%==>$1 por cento                     ## 12%       percentual

(\d+)\.(\d{1,3})\b==>$1 ponto $2              ## 12.21     decimals
(\d+)\.(\d+)==>$1 ponto __digits$2.           ## 12.00324
__digits(\d+)\.=e=>join(" ",split(//,$1))     ## 12.00324

\b(\d+)\b==>$fixnum{$1}!!defined $fixnum{$1}     ## base (0 11 100 1000)

(\d+)(000000)\b==>$1 milhões                  ## 7000000
(\d+)(000)(\d{3})==>$1 milhão e $3!!     $1 == 1 ## 1000123
(\d+)(\d{3})(000)==>$1 milhão e $2 mil!! $1 == 1 ## 1123000
(\d+)(\d{6})==>$1 milhão, $2!!          $1 == 1 ## 1123123
(\d+)(000)(\d{3})==>$1 milhões e $3           ## 7000123
(\d+)(\d{3})(000)==>$1 milhões e $2 mil       ## 7123000
(\d+)(\d{6})==>$1 milhões, $2                 ## 7123123

(\d+)(000)\b==>$1 mil
(\d+)0(\d{2})==>mil e $2!!             $1 == 1
(\d+)(\d00)==>mil e $2!!               $1 == 1
(\d+)(\d{3})==>mil $2!!                $1 == 1
(\d+)0(\d{2})==>$1 mil e $2
(\d+)(\d00)==>$1 mil e $2
(\d+)(\d{3})==>$1 mil, $2

1(\d\d)==>cento e $1
0(\d\d)==>$1
(\d)(\d\d)==>${1}00 e $2
0(\d)==>$1
(\d)(\d)==>${1}0 e $2
0$==>zero
```

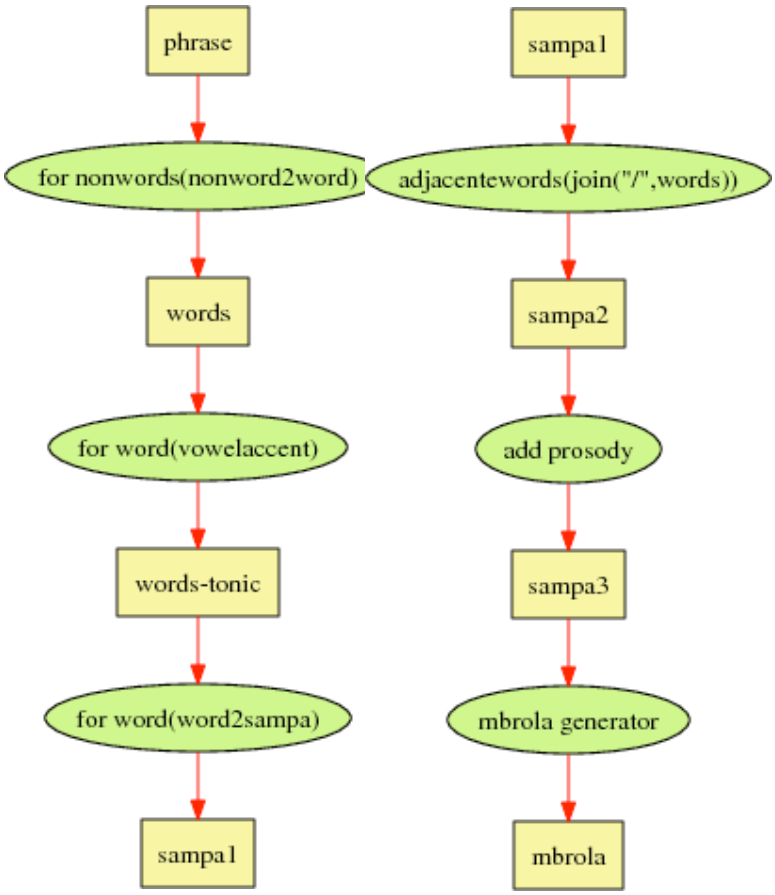
```
0==>
ENDRULES

### num2words("123.12"); # returns "cento e vinte e três ponto doze"
```

Each set of rules generate a Perl function. As standard functions, these can be composed. For example, in order to read a telephone number we can split the number and convert it to words:

```
RULES telefone
(\d{3})(\d{3})(\d{3})=e=>num2words("$1 $2 $3")
ENDRULES
```

## 4.Lingua::PT::Speaker: a TTS with Text::RewriteRules

A simple TTS was built by composition of several textual rewriting systems[5]. The following




diagram[1] describes the rewriting systems pipeline used to transform a text into a Mbrola text:

In [8] we can find several detailed algoritms regarding similar problems.

1 This is in fact a simplified version - some blocks like the nonAcentuatedWords dictionary and the exception dictionary are not presented in order to make the text less complex.

## 4.1 NonWords to words

In order to turn a real text into speech there is a large variety of situations to take care. For instance, we need to know how to read: numerals (see sec 3.2), ordinals, acronyms, emails, urls , math notation, etc.

### 4.1.1 Acronyms

```
%letra=( a => "á", b => "bê", c =>"cê", h => "agá", k => "kapa"...);

RULES/m sigla
([a-z])=e=> $letra{$1}
ENDRULES
```

### 4.1.2 Reading emails and URL

The following set of rules is used to read URL and Emails:

```
RULES/m email
\.==> ponto
\@==> arroba
:\/\/==> doispontos barra barra
:==> doispontos
(net)\b==> néte
(www)\b==> dablidablidabliw
(http)\b==> gátêtêpê
(com)\b==> cóme
(org)\b==> órg
([a-zA-Z]{1,3}?)\b=e=> sigla($1)
(.+?)\b==>$1
ENDRULES

### jj@di.uminho.pt  jota jota arroba dê i ponto uminho ponto pê tê
```

### 4.1.3 Reading Math expressions

In the following example we present some rules used in the task of translating math to text. This example is not complete.

```
RULES/m math
\(\s*(\d+)\s*\)==> $1
\(\s*(\w)\s*\)==> $letra{$1}
\(==> abre ,
\)==> , fecha

([a-z])(?=\(\s*(\w+)\s*\))==> $letra{$1} de
([a-z])(?=\(\s*\w+\s*(,\s*\w+\s*)*\))==> $letra{$1} de,
(\d+)(?=\s*\()==> $1 vezes

(\d+)\/(\d+)==> $1 sobre $2
(\w+)\/(\d+)==> $letra{$1} sobre $2
(\d+)\/(\w+)==> $1 sobre $letra{$2}
(\w+)\/(\w+)==> $letra{$1} sobre $letra{$2}

([a-z])\s+2\b==> $letra{$1} ao quadrado
([a-z])\s+3\b==> $letra{$1} ao cubo
([a-z])\s+(\d)\b==> $letra{$1} à $2ª
\^\s*2\b==> $letra{$1} ao quadrado
```

```
\^\s*3\b==> $letra{$1} ao cubo
\^\s*(\d)\b==> $letra{$1} à $2ª
\^\s*(\d+)\b==> $letra{$1} elevado a $2

(\d+)\s+2\b==> $1 ao quadrado
(\d+)\s+3\b==> $1 ao cubo
(\d+)\s+(\d)\b==> $1 à $2ª

sqrt\b==> reiís de

([^\w\s]+)==> $mat{$1} !! defined $mat{$1}
([a-z])\b==> $letra{$1}

log\b==>logaritmo de
exp\b==>exponencial de

cos\b==>cosseno de
s[ie]n\b==>seno de
mod\b==>módulo de
rand\d==>randome de
([a-zA-Z]+)==>$1

ENDRULES

### math("f(x)=4x 2 + exp(x)") returns
###  éfe de xis igual a quatro vezes xis ao quadrado, mais exponencial de xis
```

### 4.2 Words to SAMPA

Some trivial rules to convert from our alphabet to SAMPA:

```
  rr==>R              ## r
^r==>R
([nls])r==>$1R

ass==>6ss             ## s
s$==>S
($vogal)s($vogal)==>$1z$2

e$==>@                ## e

^h==>                 ## h
```

### 4.3 Adjacent words

The following set of rules is used to deal with adjacent words interference. The frontiers between words is represented by '/' character.

```
(e|a)/\1==>/$1              ## à/água         -> /água
6/6(?!~)==>/a               ## port6/6bert6   -> port/abert6
6/a==>/a                    ## 6/agu6         -> /agu6
S/([a\@eA6iouOE])==>z/$1    ## 6S/agu6S       -> 6z/agu6S
\@/([\@eaui6])==>/$1        ## @St@/ursu      -> @St/ursu
```

4.4 Prosody

We will not present the prosody rewriting system. In the tool Lingua::PT::Speaker the strategy used is to add simple prosodic markers that indicates pauses, duration and frequency variations. The markers are replaced by specific Mbrola syntax in the MBrolaGenerator rewriting system.

4.5 Local Accent

The following example makes a transformation of phonemes in order to naively simulate the voice traditionally associated with Viseu[6].
These rules are presented in order to show that is possible and easy to model and discuss certain phonetic phenomena.

```
RULES/m viseu

v==>b
s==>S
z==>Z
S==>Z

ENDRULES
### viv6 6 sidad@ d@ vizeu -> bib6 6 Sidad@ d@ BiZeu
```

## 5.Conclusion

We believe that Text::RewriteRules transfers algorithmic complexity to a set of rules, that are simpler to read and constitute a way for discussion and contribution of improvements.

Several other writing systems have been tested and are currently in use.

## References


1. *The MBROLA Project: Towards a Freely Available Multilingual Speech Synthesizer.* http://tcts.fpms.ac.be/synthesis/mbrola.html.
2. *SAMPA: computer readable phonetic alphabet.* http://www.phon.ucl.ac.uk/home/sampa/home.htm.
3. J.J. Almeida and Alberto Simoes. Text::RewriteRules - a system to rewrite text using regexp-based rules. (Perl module) Text::RewriteRules, CPAN - Comprehensive Perl Archive Network, 2007.
4. J.J. Almeida and Alberto Simoes. Lingua::PT::Speaker - perl extension for text to speech of portuguese. (Perl module) Lingua::PT::Speaker, CPAN - Comprehensive Perl Archive Network, 2008.
5. J.J. Almeida and A. M. Simões. Text to speech - a rewriting system approach. *Procesamiento del Lenguaje Natural*, 27:247-255, Sep. 2001.
6. J.J. Almeida and Alberto Manuel Simões. Geração de voz com sotaque. In *Actas do XVIII Encontro da Associação Portuguesa de Linguística*, Porto 2002, 2003.
7. Alan W Black and Kevin A. Lenzo. Building synthetic voices. Technical report, Language Technologies Institute, Carnegie Mellon University, 2007. http://www.festvox.org/festvox/festvox_toc.html.
8. Daniela Braga. *Algoritmos de PLN para conversão texto-fala em Português*. Tese de doutoramento, Universidade da Corunha, 2008.
9. Andreas Koenig. *CPAN - Comprehensive Perl Archive Network*, 1995. http://www.perl.org.

## A Regular expressions

```
\s          space (white space, tab)  [ \t\n]
\w          letters, digits _         [a-zA-Z0-9_áéíóúç...]
\d          digit                     [0-9]
\b          word boundary
^_          _ at the beggining of
_$          _ at the end of

_*          0 or more repetitions of _
_+          1 or more repetitions of _
_{8}        8 repetitions of _
_{2,8}      between 2 and 8 repetitions of _
_?          0 or 1 repetitions of _

(?=_)       _ positive lookahead
(?!_)       _ negative lookahead
```