# Jspell – um módulo para análise léxica genérica de linguagem natural

José João Dias de Almeida
Ulisses Pinto
Departamento de Informática
Universidade do Minho
Campus de Gualtar, 4700 Braga
{jj,ulisses}@di.uminho.pt

26 de Agosto de 1994

**Resumo**

Neste texto apresenta-se o JSPELL um módulo analisador léxico-morfológico genérico, parametrizado por dicionário (lista de palavras) e regras de flexão externos a ser usado como biblioteca C , como comando de linha, como "pipe" ou como corrector ortográfico.

O JSPELL foi construído com base no corrector ortográfico de domínio público ISPELL estendido no sentido de associar a cada entrada no dicionário um conjunto de pares atributo-valor (features) e um conjunto de regras morfológicas aplicáveis.

A cada regra morfológica está também associada um conjunto de pares atributo-valor correspondentes. Deste modo a análise de palavra retorna um conjunto de possíveis interpretações da palavra e cada interpretação contêm o radical e um conjunto de pares atributo-valor.

O módulo fornece também funções para procurar palavras aproximadas de uma palavra não existente no dicionário, funções para extracção de palavras de um texto, funções para manutenção dum dicionário particular, etc.

Palavras chave: linguística computacional, processamento de linguagem natural

# 1 Introdução

## 1.1 Motivação

Grande parte das aplicações em linguagem natural necessitam de um mecanismo de classificação léxica, que dada uma palavra obtenha várias informações a seu respeito tal como se existe ou não no dicionário, conhecer a sua origem, categoria gramatical a que pertence e muitas outras *features* como o Género, o Número, etc.

A enumeração exaustiva de todas as palavras e desinências é não apenas muito morosa mas mesmo virtualmente impraticável para casos reais. Uma forma mais racional de obter o mesmo resultado é armazenar apenas uma lista com as raízes das palavras e ter **regras de formação** de desinências, devendo cada uma das palavras da lista dada indicar através de "flags" quais as regras de formação que lhe podem ser aplicadas (caso existam).

Optou-se pelos seguintes requisitos gerais da ferramenta a construir:

- dicionários e regras de derivação externos para mais fácil possibilidade de alteração
- possibilidade de definição de *features* tanto a nível de dicionário como de regras de derivação
- Fácil interface com linguagens de programação como o C, PROLOG e PERL
- possibilidade de funcionamento como corrector ortográfico
- possibilidade de lidar com várias sintaxes de definição de caracteres [1]

Existe disponível uma ferramenta, o ISPELL (corrector ortográfico), que implementa uma parte dos requisitos pretendidos, e como esta ferramenta se encontra disponível em código fonte e com permissão para alterações, optou-se por lhe acrescentar funcionalidade. Esta opção teve vantagens a vários níveis:

- Diminuiu consideravelmente o tempo de desenvolvimento da nova ferramenta.
- Aumentou a fiabilidade (dado que a maior parte do código já está testado).
- Facilita a utilização da nova ferramenta por pessoas já familiarizadas ao ISPELL .

[1] Esta questão torna-se particularmente importante para caracteres acentuados e cedilhados, cuja representação interna varia bastante

## 1.2 Descrição sucinta

O JSPELL baseia-se em 3 dicionários (geral, pessoal e temporário) e num conjunto de regras (ficheiro dos afixos).

Cada dicionário é uma lista de palavras com a respectiva classificação e conjunto de regras de expansão aplicáveis.

Este módulo tem 4 modos básicos de utilização:

- como programa interactivo de correcção ortográfica com menus e opções
- como interpretador de linha – o utilizador introduz uma palavra e recebe imediatamente informação sobre essa palavra. Não é bufferizado e pode por isso ser usado em `pipes` .
- comando de linha (exemplo para escrever a lista de palavras dum texto que não existem num dicionário como o `spell` do UNIX ).
- na forma de uma biblioteca C .

Em modo interactivo funciona como corrector ortográfico, detectando erros, dando sugestões de correcção e permitindo inserir novas palavras num dicionário pessoal.

A existência de uma biblioteca C é praticamente indispensável para a construção de ferramentas usando o JSPELL , tanto em C como noutras linguagens.

O modo de funcionamento em "pipe" destina-se a cooperar com programas já existentes que prevejam esse tipo de funcionamento (Ex. EMACS, PERL etc).

Para a criação de dicionários é necessário usar um comando auxiliar JBUILD (derivado do código original BUILDHASH ) que cria uma estrutura de acesso (uma tabela de hashing) em disco a partir de um dicionário textual e de um conjunto de regras, vulgarmente arrumadas num ficheiro com a extensão `aff` .

## 1.3 Funcionalidade

O módulo de analise léxica permite:

- obter o(s) radical(is) duma palavra
- obter a(s) classificação(ões) a ela associadas
- obter palavras aproximadas (palavras à distância de Hamming de 1 da palavra original) para o caso de palavras inexistentes.
- obter possíveis combinações de radical/regras que produzam uma dada palavra
- funções para extracção de palavras duma linha
- acrescentar palavras a dicionários pessoais

As classificações associadas a palavras contêm para além do radical de base 4 conjuntos de pares atributo valor correspondentes a:

- classificação do radical
- possíveis prefixos
- possíveis sufixos1
- possíveis sufixos2 (exemplo: clíticos)

## 1.4 Exemplos de análise

A utilização do JSPELL pode ser controlada de vários modos dispondo-se de cerca de 30 opções (muitas das quais herdadas do ISPELL ) que podem ser usadas de modo a definir:

- modo de utilização (interactivo, comando, pipe, etc)
- tipos de codificação de caracter a usar,
- tipo de documento a analisar,
- formato de apresentação das análises,
- aspectos visuais (quando em modo interactivo)
- interesse ou não na determinação de palavras aproximadas
- nome dos dicionários a usar
- etc.

Como resultado de analisar uma palavra, é retornado uma string que agrupa o radical e os quatro pares atributo-valor atrás referidos. A maneira de compor esses constituintes é parametrizável por um formato [2].

```
gatinhando :
lex(gatinhar,[CAT=v,TR=i],[],[T=g],[])

reanalisar:
lex(analisar,[CAT=v,TR=t],[FSEM=outra],[],[])

lavei-me :
lex(lavar,[CAT=v,TR=t],[],[P=1,N=s,T=pp],[P=1,N=s])
```

[2] Por defeito o formato é `"lex(%s, [%s], [%s], [%s], [%s])"` e corresponde ao formato de escrita da função `printf` da linguagem C

# 2 Estrutura interna

Como referimos anteriormente o sistema funciona com base em três dicionários:

- dicionário geral
- dicionário pessoal (onde há possibilidade de acrescentar palavras)
- dicionário temporário onde são guardadas as palavras definidas como válidas somente para esta sessão

Os dicionários podem referir identificadores de regras. As regras serão especificadas em formato próprio que se descreverá seguidamente.

## 2.1 Dicionário

### Entrada típica

Uma entrada típica no dicionário é uma linha do tipo:

```
palavra/classificação/flags[/comentário]
```

ex.

```
gato/CAT=nc,G=m,N=s/p/é um lindo animal
```

A classificação pode ser qualquer string de caracteres (no nosso exemplo foi usado uma lista de pares atributo valor). A classificação pode usar "macros" (ver abaixo).

A lista de flags indica o nome das regras que lhe serão aplicáveis (neste caso aparece apenas a regra `p` que corresponde ao plural e a alterar a classificação em `N=p` ).

O comentário é opcional.

### Macro Substituição

Verificou-se que neste tipo de dicionários é muito frequente a repetição de *strings* de classificação; para uma maior economia de espaço e facilidade de definição, disponibiliza-se a possibilidade de macro substituição, permitindo a definição de *abreviaturas* para strings. As macros começam por `#` (eventualmente redefinível no ficheiro `config.X` ), e são seguidas dum identificador de macro e sua definição. Ex.

```
#vt/CAT=v,TR=t            (macro para verbo transitivo)

abaixar/#vt/XYL/          (utilização da macro)
```

é equivalente a:

```
abaixar/CAT=v,TR=t/XYL/
```

### Repetição de entradas

São permitidas entradas repetidas no dicionário, i.e. entradas em que a palavra é igual, mas podendo ter classificação e flags diferentes. Assim a mesma palavra pode aparecer em 2 ou mais soluções independentes.

### Irregularidades e radicais associados

Normalmente, uma entrada do dicionário contém uma palavra que é um radical, e que através de *flags* pode gerar outras palavras (por exemplo, um verbo poderá ter apenas uma entrada no infinitivo associada a uma regra que produza todas as formas verbais). No entanto a existência de irregularidades tornam impossível o uso deste método em todos os casos, já que há palavras que derivam de outras de uma forma irregular que não pode ser descrita por uma regra minimamente genérica.

Para estes casos foi criado um mecanismo que permite que palavras sejam acrescentadas ao dicionário, contendo informação sobre o radical de que derivam de forma irregular e atributos da forma:

```
palavra/\$palavra raiz\$classif. palavra raiz\$classif. palavra/flags
```

ex.

```
sou/$ser$CAT=v,T=inf$T=p,N=s,P=1/
```

Quando fosse pedida informação sobre a palavra *sou*, obter-se-ia:

```
lex(ser, [CAT=v,T=inf],[],[T=p,N=s,P=1],[])
```

## 2.2 Regras de flexão

O ficheiro que contem as regras de flexão contêm as seguintes partes:

- Definição de conjunto de caracteres possíveis
- Regras de prefixação
- Regras de sufixação

A descrição de cada regra será feita do seguinte modo:

```
"flag" ["*"|"+"] iden ":" [";" classif] linha_regra*
```

E cada ”linha de regra” será:

```
 terminação > ["-" str-retirar ","] str-acresc [";" classif]
```

Considere-se o seguinte exemplo (extraído do ficheiro de regras para o português):

```
# nte
flag *n:                ; "CAT=v,T=inf"

[AE] R >  -R,NTE     ; "CAT=adj,N=s,FSEM=nte"  # falar->falante
I R    > -IR,ENTE    ; "CAT=adj,N=s,FSEM=nte"  # aderir->aderente

[AE] R >  -R,NTES    ; "CAT=adj,N=p,FSEM=nte"  # falar->falantes
I R    > -IR,ENTES   ; "CAT=adj,N=p,FSEM=nte"  # aderir->aderentes
```

Cada regra:

- tem um identificador (flag) associado (no exemplo `n` )
- tem informação acerca do modo como se pode compor com outras regras (no exemplo `*` )
- tem informação (string de classificação) sobre os radicais a que é aplicável (no exemplo `"CAT=v,T=inf"` )
- tem linhas de regra que permitem a partir duma palavra radical obter zero ou mais formas derivadas e respectivas classificações.

As linhas de *regra* são constituídas por:

- uma condição que indica as letras com que a palavra radical pode terminar
- uma *string* a ser retirada ao radical
- outra *string* a ser acrescentada ao radical
- classificação da palavra derivada assim construída.

# 3 Exemplos

Nesta secção apresentam-se alguns exemplos de análise usando o sistema e um dicionário local de português.

### Características consideradas

As features existentes neste dicionário são: CAT - Categoria
G - Género: m-masculino; f-feminino; _-ambos
N - Número: s-singular; p-plural; _-ambos
P - Pessoa: 1; 2; 3;
T - Tempo
TR - Transitividade: GR - Grau: FSEM - Função semântica

**Categorias consideradas**

nc - nome comum
np - nome próprio
adj - adjectivo
a_nc - adjectivo e/ou nome comum
v - verbo
art - artigo
ncard - numeral cardinal
nord - numeral ordinal
nmult - numeral multiplicativo
nfrac - numeral fraccionário
prep - preposição
adv - advérbio
ppes - pron. pessoal
pdem - pron. demonstrativo
ppos - pron. possessivo
pind - pron. indefinido
prel - pron. relativo
pint - pron. interrogativo
conj - conjunção
in - interjeição

## 3.1 Utilização em programação

Nesta secção apresentaremos alguns exemplos pouco comentados e cuja total compreensão exige o conhecimento da respectiva linguagem.

O objectivo é apenas o de dar uma leve ideia da complexidade ligada à utilização em programação. Para mais detalhes ver o respectivo manual de utilização [2][3].

**Como biblioteca C**

Para se ter uma ideia (vaga) do que é a utilização do módulo como biblioteca C , apresenta-se um exemplo em que, para cada palavra lida é apresentado o conjunto dos respectivos radicais de base e informação de classificação associada. No caso de não existir a palavra são apresentadas as "palavras aproximadas" dum modo idêntico ao anterior.

```
#include "jslib.h"

main()
{ int i;
   char X[BUFSIZ], char w[100], *p;

   init_jspell("-d dict -W 0 -a");
```

```
    while (gets(X)) {
      for(p = X; p=get_next_word(p, w); ) {

        word_info(w, solutions, near_misses);

        puts("solutions");
        for(i = 0;solutions[i][0];i++) puts(solutions[i]);
        puts("near_misses");
        for(i = 0;near_misses[i][0];i++) puts(near_misses[i]);
      }
    }
}
```

### Como pipe em script PERL

Nesta secção apresenta-se um exemplo de utilização do JSPELL como pipe invocada de um programa PERL [6]. O perl está a ser usado pela sua grande versatilidade em usar padrões ( e expressões regulares para a definir) bem como pela facilidade de usar pipes.

A linguagem PERL é conhecida pelo seu poder expressivo, pela sua escrita compacta e pela sua pouca legibilidade...

O exemplo que se apresenta pretende analisar um ficheiro com terminologia e detectar erros ortográficos nas palavras que estão nos campos de língua portuguesa (no exemplo abaixo os campos `PT` e `DF` ).

Exemplo de registos a analisar:

```
PT gato
IN cat
FR ...
DF animal que tem 7 vidas e meiiiia
FO ...
...
```

Para tal escreveu-se um pequeno programa PERL que arranca com um processo JSPELL que será usado para fazer a análise morfolófica de cada palavra. JSPELL responde *dizendo* para cada palavra se ela existe (linhas com `*` ) ou não (linhas com `&` ) no dicionário (ver Fig.1). Com base nessa informação é construído um relatório com os erros e respectivos números de linha.

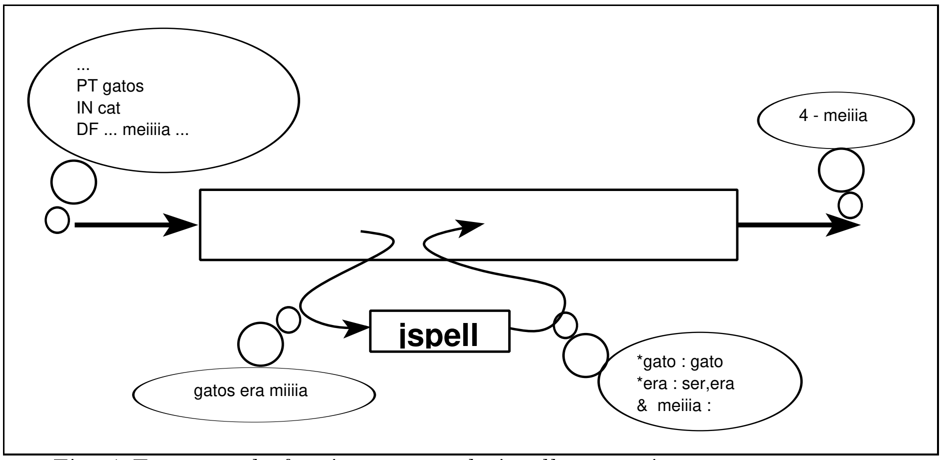


Fig. 1 Esquema do funcionamento do jspell como pipe

```
#!/usr/local/bin/perl

require('open2.pl');
$pid= &open2(I,O,"jspell -d port -o'%s ' -y -a");

while (<>) { $line++;
  if(/^(PT|DF) (.*)$/) {                     # as linhas com "PT" ou "DF"
    print O "$2\n";                          # são enviadas ao jspell.
    for ($F=<I>;($F ne "\n");$F=<I>){        # lê as respostas
      ($in,$pal)=($F=~ /^([*&]) ([^ ]*) /);  # extrai a informação
      if($in eq "&") {                       # se não existe no dicionário
         printf "$line - $pal\n"; }}}        # dá a linha e palavra
}
```

Quando aplicado ao exemplo anterior daria:

```
4 - meiiiia
```

# 4 Conclusões e trabalho futuro

Este trabalho corresponde a um compromisso entre um desejo de generalidade, facilidade de alteração/adaptação e a necessidade de rapidez de desenvolvimento e de simplicidade.

A nível de teste, tem sido usado um dicionário Português com cerca de 40 000 entradas (correspondendo a cerca de 400 000 formas) e um ficheiro de regras com cerca de 900 linhas (incluindo comentários), e tratando quatro tipos de formatos (LaTeX, latin1, pre-acentuadas e pos-acentuadas).

Estão ainda em fase de desenvolvimento o interface a PROLOG , o tratamento de contracções e a possibilidade de entradas com mais que uma palavra.

Estão em desenvolvimento outros módulos complementares no sentido dum mais poderoso uso a nível sintáctico.

O sistema pode ser obtido livremente contactando os seus autores.

# Referências


[1] Nicoletta Calzolari. Computational lexicons. reader of 5.LLI, Faculdade de Letras, Universidade de Lisboa, 1993.

[2] José J. Dias de Almeida and Ulisses Pinto. Manual de utilizador do jspell. Manual, Universidade do Minho, Julho 1994.

[3] Ulisses Pinto. Processamento léxico de linguagem natural. Relatorio de estágio, Universidade do Minho, Julho 1994.

[4] Graeme D. Ritchie, Graham J. Russel, Alan W. Black, and Stephen G. Pulman. *Computational Morphology: Practical Mechanisms for the English Lexicon.* Series in Natural Lnaguage Processing. The MIT Press, 1992.

[5] Mário Vilela. *Estudos de Lexicologia do Português.* Almedina – Coimbra, 1994.

[6] Johan Vromans. Perl reference guide. Manual 4.036, 1990. Perl was designed by Larry Wall.