Manual do Utilizador do JSpell

José João Almeida
Ulisses Pinto

October 31, 1995

Contents

1 Introdução

1.1 Motivação

Grande parte das aplicações em linguagem natural necessitam de um mecanismo de classificação léxico, que dada uma palavra obtenha várias informações a seu respeito, a mais evidente das quais é se existe ou não no dicionário, mas não menos importante é conhecer a sua origem, bem como a 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, dada a grandeza dos números em questão - mais de 600000 palavras - 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).

Já existe disponível uma ferramenta, o ISpell, que implementa uma grande parte dos requisitos pretendidos, e como esta ferramenta se encontra disponível em código fonte e com permissão para alterações, optou-se por lhe acrescentar funcionalidades. 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.

1.2 Descrição sucinta

Este programa baseia-se num dicionário e num conjunto de regras (ficheiro dos afixos).

Tem 4 modos básicos de utilização:

- como programa interactivo com menus e opções (usando `termcap`)
- 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 (é bufferizado - primeiro recebe todo o input e só então escreve o output), é o caso da opção -l que escreve uma lista de palavras com erros ortográficos.
- na forma de uma biblioteca C.

No modo interactivo com menus funciona de forma semelhante a um corrector ortográfico, detectando erros, dando sugestões de correcção e permitindo inserir num dicionário pessoal.

A existência de uma biblioteca C é praticamente indispensável para a construção de camadas por cima do JSpell, tanto em C como noutras linguagens.

Um programa auxiliar mas indispensável é o JBuild (derivado do código original BuildHash) que cria uma estrutura em disco a partir de um dicionário e de um conjunto de regras, normalmente expressas num ficheiro com a extensão .AFF.

2 Utilização do JSpell como programa

```
jspell [flags-comuns] [-M|-N] [-L contexto] [-V] ficheiros
jspell [flags-comuns] [-f ficheiro] [-s] {-a|-A}
jspell [-d ficheiro] [-w cars] [-o formato] -c
jspell [-d ficheiro] [-w cars] [-o formato] -e [e]
jspell [-d ficheiro]  -D
jspell -v [v]

flags-comuns:
 [-t] [-n] [-b] [-x] [-B] [-C] [-P] [-m] [-S] [-d ficheiro]
 [-p ficheiro] [-w cars] [-W n] [-T tipo] [-o formato] [-g] [-y]
 [-u] [-z]


jbuild [-s] fich-dicionário fich-afixos fich-hash
jbuild -s contador fich-afixos
```

2.1 Modo de correcção interactivo com menus

O jspell é invocado com o nome do texto a verificar, portando-se do seguinte modo:

Cada palavra que não existe no dicionário é mostrada em *reverse vídeo* no cimo do écran, dentro do contexto em que estava, sendo dado ao utilizador várias sugestões de correcção numeradas (caso existam), estas sugestões são formadas de duas formas:

- através da detecção de "palavras aproximadas" (ex. palavras a que têm uma letra diferente, a que falta uma letra, que têm uma letra a mais ou que têm 2 letras seguidas trocadas).
- através do uso de regras de formação a partir de uma raiz conhecida, apesar desta não prever o uso dessa *flag*.

Descrição do mini menu

Na penúltima linha do ecran são escritas as várias opções que o utilizador pode escolher:

<number> - digitar o no da sugestão escolhida para substituir no texto original.

Space - Aceitar a palavra só desta vez.

R - Substituir a palavra

E - Substituir todas as ocorrências da palavra no texto

A - Aceitar a palavra durante o resto desta sessão Jspell.

I, U - Aceitar a palavra (no caso I com as maiúsculas/minúsculas idênticas às do texto, na opção U em minúsculas), e actualizar o dicionário pessoal. De notar no entanto que o nosso dicionário para além da palavra propriamente dita mantém informação sobre a classificação das palavras bem como das *flags* que se lhe aplicam e um eventual comentário, assim é também pedida uma Classificação, Flags e Comentário, ou em alternativa pode-se escolher algumas sugestões que o Jspell formou usando as regras do .AFF, no caso de a palavra raiz das sugestões já existir, é escrito um '*' à frente da palavra indicando uma maior probabilidade de sucesso.

L - Procurar palavras no dicionário do sistema (controlado pela opção de compilação WORDS)

X - Escrever o resto do ficheiro corrente tal como está, ignorando palavras com erros e começar a correcção do próximo texto (caso exista).

Q - Sair imediatamente e deixar o ficheiro sem alterações.

! - Saída para a Shell

^**L** - Redesenhar o ecran

^**Z** - Suspender Jspell

? - Mostrar ecran de ajuda

2.2 Opções da linha de comando

-M Indica que deve ser mostrado um mini-menu no fundo do ecran.

-N Suprime o mini-menu. (Nota: o mini-menu é mostrado por defeito).

-L Indica o nº de linhas de contexto a serem mostradas.

-V Mostra os caracteres que usem mais que 7 bits no estilo de "cat -v", mesmo que sejam caracteres ISO Latin-1 válidos. Esta opção é útil quando se trabalha com terminais antigos que só suportam correctamente caracteres de 7 bits.

-t O ficheiro de input está no formato TeX ou LaTeX. De notar que este modo é automaticamente seleccionado caso o ficheiro de input tenha a extensão ".tex".

-n O ficheiro de input está no formato nroff/troff.

-b Criar um ficheiro de *backup* acrescentando ".bak" ao nome do ficheiro de input.

-x Não criar ficheiro de *backup*.

-B Considerar erros palavras juntas sem espaços a separá-las.

ex. "patobravo" seria considerado erro.

-C Considerar correctas palavras juntas sem espaços a separá-las. Esta opção é útil em línguas como o alemão, em que muitas palavras compostas são formadas por concatenação.

ex. "patobravo" seria considerado correcto.

-P Não gerar sugestões de combinações raiz/afixo para serem possivelmente adicionadas ao dicionário pessoal.

-m Tornar possíveis combinações raiz/afixo que não estão no dicionário.

-S Ordenar a lista de sugestões por probabilidade de correcção, em vez da habitual ordem alfabética.

-d *fich* Especificar um dicionário alternativo.

-p *fich* Especificar um dicionário pessoal alternativo. Se o ficheiro não começar com "/" presume-se o prefixo $HOME. Se se especificar um dos `fich-hash` *default* do dicionário da biblioteca e exista o ficheiro ".jspell_hashfile", este será usado como dicionário pessoal. Se nenhuma destas condições se verificar usa-se o ficheiro ".jspell_words".

Se não se especificar a opção -p, o Jspell irá procurar os dicionários pessoais tanto na directoria corrente como na directoria *home*. Se existirem dicionários em ambos os locais, serão ambos carregados.

-w *cars* Especificar caracteres adicionais que podem ser parte de uma palavra.

ex. -w "&" faz com que seja considerada legal a palavra "AT&T".

-W *n* Especificar o tamanho máximo de palavras a serem consideradas sempre legais. Se se pretende fazer a verificação de todas as palavras independentemente do seu tamanho, é aconselhável usar "-W 0".

-T *tipo* Assumir um determinado tipo de formatação para todos os ficheiros. O argumento tipo tanto pode ser um dos nomes únicos definidos no ficheiro de afixos (ex. nroff) como um sufixo de ficheiro contendo um ponto (ex. .tex).

-l É usado para produzir uma lista de palavras mal escritas a partir do *standard input*.

-a Foi pensado para ser usado a partir de outros programas através de *pipe*. Não é escrito nenhum menu, nem são pedidas respostas ao utilizador.

Se a palavra foi encontrada directamente no dicionário, ou através do uso de *flags*, aparece a informação sobre a raiz da palavra, as *features* da raiz e as *features* do afixo/prefixo usado. Esta informação aparece segundo um formato que pode ser alterado pelo utilizador.

Se a palavra não está no dicionário, a linha de output contém um '&', um espaço, a palavra original, um espaço, o nº de caracteres entre o início da linha e o início da palavra, ':' uma lista das "palavras aproximadas" em que aparece o nome da palavra '=' seguido da sua classificação usando o formato de output especificado. Se a palavra podia ser formada através da adição ilegal de afixos a uma raiz conhecida, também é apresentada uma lista de sugestões assim formadas.

Se não houver "palavras aproximadas", mas apenas formação a partir de afixos ilegais, a linha tem um formato idêntico ao anterior, excepto que em vez de "&" aparece "?".

Em resumo:

Se a palavra Existe no dicionário, o output tem o formato:

```
* <original> <cont> <offset>: <solução>, <solução>, ...
```

Se Não existe no dicionário:

```
& <original> <cont> <offset>: <falha>, <falha>, ...,
      <sugestão afixo>, ...
```

`<falha>` e `<sugestão afixo>` têm o formato:

```
<palavra> = formato(<raiz>, <features_raiz>, <features_prefixo>,
                    <features_sufixo>, <features_sufixo2>)
```

o formato é definível pelo utilizador sendo por *default*:

```
lex(<raiz>, [<features_raiz>], [<features_prefixo>],
    [<features_sufixo>], [<features_sufixo2>])
```

De notar que os separadores ',', '=' e ':' estão em "#define" podendo assim ser alterados.

No modo -a, também são aceites linhas prefixadas por '*', '@', '&', '+', '-', '~', '#', '!', '%', '$' ou '^'. Estes caracteres funcionam como comandos vários que podem conter argumentos:

* Acrescentar ao dicionário pessoal, também se pode acrescentar a classe, as *flags* e os comentários usando o separador usado no dicionário.

@ Aceitar a palavra, mas deixá-la fora do dicionário

& Acrescentar ao dicionário pessoal a palavra convertida para minúsculas

Gravar o dicionário pessoal corrente

~ Indicar os parâmetros baseados no ficheiro, ex. tipo de *strings* por *default*

+ Entrar no modo TeX

- Sair do modo TeX

! Entrar em modo "terse".

% Sair do modo "terse".

$ *flag* Altera o modo de funcionamento do mesmo modo que a função `init_modes` (ver Cap. "Utilização da Biblioteca").

^ Verificar a ortografia do resto da linha.

De notar que no modo "terse" não é escrita a informação relativa às palavras correctas, isto aumenta a performance de programas que não necessitem desta informação.

-A Funciona como -a, excepto que se a linha começar com a *string* "&Include_File&", o resto da linha é considerado como sendo o nome de um ficheiro a ser lido para tratar mais palavras.

-s Se for especificado, o Jspell irá parar com o *signal* SIGSTP depois de lida uma linha de input, e só continua a ler a próxima linha quando receber o *signal* SIGCONT.

Só é válido em conjunto com as opções -a ou -A e em sistemas derivados do BSD.

-f Se for especificado, o Jspell irá escrever os resultados para o ficheiro especificado, em vez do *standard output*. Só é válido em conjunto com as opções -a ou -A.

-v Faz com que o Jspell escreva a sua versão corrente.

Se a opção for duplicada (-vv), também escreve as opções com que foi compilado.

-c Faz com que seja lida uma lista de palavras do *standard input*, e para cada uma delas seja escrita uma lista de raízes possíveis, sua classificação, e classificação da palavra original derivada desta maneira, bem como as *flags* usadas. De notar que as raízes geradas podem não se encontrar no dicionário.

ex. o input "batatas" gera:

```
batatas lex(batata, [CAT=adj_nc], [N=p], []),
        lex(batatar, [CAT=v], [CAT=v,P=2,N=s,T=p], [])
```

-z Faz com que a *flag* usada também seja escrita ex.:
```
lex(batata, [CAT=adj_nc], [N=p], [])/p
```

-e É o inverso de -c; a partir de uma palavra e de uma *flag*, gera as várias hipóteses de palavras derivadas possíveis segundo as regras da *flag* dada.

ex. batata/p gera:
```
batata batatas= lex(batata, [], [N=p], [])
```

-D Faz com que as tabelas de afixos do dicionário sejam escritos no *standard output*.

-o *formato* Define o formato de output. O formato deve ser uma *string* que contenha cinco "%s" para serem preenchidos com a raiz da palavra, a classificação da raiz e a classificação associada à *flag*. O *default* é:
```
lex(%s, [%s], [%s], [%s], [%s])
```

-g Indica que devem ser mostradas apenas as soluções, não se pretendem sugestões. O seu uso aumenta a performance.

-y Indica que se pretende obter apenas as sugestões criadas usando *flags* não previstas para a palavra em questão, não havendo portanto o cálculo de "near misses" (sugestões por alteração de letras na palavra original). O seu uso aumenta a performance (mas não tanto como o -g).

-u Ignorar a pontuação, que deixa de ser considerada uma palavra. De notar que existe um #define DEFAULT_SIGNS que indica o que é que é considerado um sinal de pontuação.

O output na opção -a, -e, -c usa determinados separadores que estão em *#define* e podem ser redefinidos:
SEP1 ", "
SEP2 "; "
SEP3 "= "
SEP4 "\n"

O SEP1 é usado para separar as várias hipóteses de solução.

O SEP2 é usado quando se mostram as "near misses" para indicar que se chegou ao fim deste tipo de sugestões

O SEP3 é usado para separar a palavra de que se pediu informação e que é repetida no output, da informação propriamente dita acerca dela.

SEP4 indica que se chegou ao final de todo o tipo sugestões ou soluções relativas à palavra dada.

2.3 jbuild

Este programa constrói um ficheiro no disco contendo a informação das palavras do dicionário especificado organizadas numa tabela de *hashing* e contendo também a informação do ficheiro de afixos.

fich-dicionário contém a lista de palavras

fich-afixos contém as regras associadas a *flags* fich-hash é onde é escrito o output deste programa que irá ser usado pelo Jspell.

-s Suprime as mensagens que são habitualmente escritas no *standard error* para indicar o estado actual do processamento.

3 Utilização da Biblioteca

Os programas que queiram usar esta biblioteca devem fazer `#include "jslib.h"` e *linkar* com "jspell.a".

Os programas devem sempre fazer a inicialização da biblioteca com `init_jspell("...")`, e só nesta altura poderão invocar as outras funções disponibilizadas.

3.1 Funções disponibilizadas

init_jspell(char *opt)

Inicializa o Jspell, usando as *flags* da *string* `opt`, que têm um funcionamento idêntico ao já documentado acima, quando o Jspell é chamado em comando de linha para funcionar sem os mini-menus. Como os mini-menus não fazem sentido na biblioteca a opção -a costuma ser sempre posta.

ex.

```
init_jspell("-d dic-pe -W 0 -a -cf")
```

word_info(char *word, sols_type solutions, sols_type near_misses)

Dá informação acerca da palavra `word`, que é procurada no dicionário. Se ela for encontrada, as formas de a obter são postas no *array* `solutions`, em que cada elemento deste *array* é uma *string* que indica qual a raiz qual a classificação da raiz, e a classificação associada à *flag* usada para formar a palavra dada.

Caso a palavra dada não tenha sido encontrada é posto em `near_misses` várias sugestões de palavras alternativas.

As *strings* em `solutions` e `near_misses` usam o formato de output especificado para este tipo de informação.

Se `solutions[i]` ou `near_misses[i]` tiverem a *string* vazia, significa que não há mais soluções / sugestões respectivamente.

void init_modes(char *modes)

Esta função permite mudar o modo de mostrar sugestões ao utilizador. Existem dois tipos de sugestões possíveis: as que são dadas fazendo pequenas alterações na palavra original - designadas por "near misses" - e as que são dadas utilizando afixos não previstos para essa palavra.

As *flags* disponíveis são pois:

g - não dar sugestões de outras palavras (elimina "near misses").

G - desligar opção g

P - não dar sugestões por combinação raiz/afixo de *flags* lidas

m - desligar opção P

y - não dar sugestões por alteração de letras na palavra original

Y - desligar opção y

z - indicar quais as *flags* usadas

Z - desligar opção z

char *get_next_word(char *buf, char *next_word)

Dado um *buffer*, põe em `next_word` a primeira palavra válida que encontrar.

Retorna um apontador para a posição em `buf`, imediatamente a seguir ao fim da palavra encontrada. Caso não seja encontrada qualquer palavra retorna NULL.

get_roots(char *word, sols_type solutions, char in_dic[MAXPOSSIBLE])

Dada uma palavra procura origens possíveis para ela, mesmo que não estejam no dicionário. As várias possibilidades são postas no *array* `solutions`, tendo cada hipótese a indicação da raiz, classificação da raiz, e a classificação associada à *flag* usada, esta informação é posta numa *string* usando o formato de *output* habitual. Cada entrada no *array* `in_dic` indica se a raiz existe ou não no dicionário.

Se `solutions[i]` tiver a *string* vazia, significa que não há mais soluções.

insert_word(char *word, char *class, char *flags, char*comm)

Insere uma palavra (com respectiva classificação, *flags* e comentários) no dicionário pessoal.

accept_word(char *word, char *class, char *flags, char *comm)

Aceita uma palavra (com respectiva classificação, *flags* e comentários) durante o resto desta utilização com a biblioteca.

char *replace_word(char *start, char *word, char *curchar)

Substitui a palavra existente no texto na posição `start` do *buffer*, pela palavra `word` sendo indicado em `curchar` onde terminava a palavra anterior.

Retorna a posição no *buffer* onde termina a nova palavra.

save_pers_dic()

Grava em disco o dicionário pessoal no estado em que estiver.

ID_TYPE word_id(char *word, char *feats, int *status)

Para uma dada palavra retorna um identificador único.

char *word_f_id(ID_TYPE id)

Dado um identificador, retorna um apontador para a posição da palavra respectiva.

char *class_f_id(ID_TYPE id)

Dado um identificador, retorna um apontador para a posição da classificação respectiva.

char *flags_f_id(ID_TYPE id)

Dado um identificador de palavra, retorna uma *string* com a identificação das *flags* respectivas.

3.2 Ex. de utilização da Biblioteca

```
#include "jslib.h"

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

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

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

         word_info(w, solutions, near_misses);
         puts("solutions");
         i = 0;
         while (solutions[i][0]) {
            puts(solutions[i]);
            i++;
         }
         puts("near_misses");
         i = 0;
         while (near_misses[i][0]) {
            puts(near_misses[i]);
            i++;
         }
      }
   }
}
```

Compilação do programa

Partindo do princípio que o programa acima listado tem o nome de exp-lib.c a sua compilação seria feita da seguinte forma:

```
cc exp-lib.c jspell.a
```

4 Formato do Ficheiro de Afixos (.AFF)

4.1 Descrição da Gramática

```
 table    : [headers] [prefixes] [sufixes]
headers   : [options] char-sets
options   : {fmtr-stmt | bool-stmt | flag-stmt | num-stmt}
fmtr-stmt : {nroff-stmt | tex-stmt}
nroff-stmt : {nroffchars | troffchars} string
tex-stmt  : {TeXchars | texchars} string
bool-stmt : {cmpnd-stmt | aff-stmt}
cmpnd-stmt: compoundwords on-or-off
aff-stmt  : allafixes on-or-off
on-or-off : {on | off}
flag-stmt : flagmarker character
num-stmt  : compoundmin digit
char-sets : norm-sets [alt-sets]
norm-sets : [deftype] charset-group
deftype   : defstringtype name deformatter suffix*
charset-group: {char-stmt | string-stmt | dup-stmt}*
char-stmt : wordchars character-range
          | wordchars lowercase-range uppercase-range
          | boundarychars character-range
          | boundarychars lowercase-range uppercase-range
string-stmt : stringchar string
          | stringchar lowercase-string uppercase-string
alt-sets  : alttype [alt-stmt*]
alttype   : altstringtype name suffix*
alt-stmt  : altstringchar alt-string std-string

prefixes  : prefixes flagdef*
suffixes  : suffixes flagdef*
flagdef   : flag [*|+] char : [; classification] repl*
repl      : condition* > [- strip-string,] append-string [; classification]
```

Podemos encontrar uma descrição mais detalhada desta gramática no manual do Ispell. No entanto existem acrescentos no programa Jspell que evidentemente não podem estar na documentação original.

4.2 Classificação de palavras

O expressividade foi aumentada através de um campo *classification* (que é opcional), que não é mais do que uma *string* (deve estar entre aspas) onde se pode pôr o que se quiser, mas que em princípio deverá ter a classificação da palavra.

ex.

```
flag p: ; "CAT=adj_nc"

I L > -L, S ; "N=p"                # il -> is   ex. funil -> funis
```

Neste caso `"N=p"` significaria que através da aplicação daquela regra o Número (N) passava a plural (p).

A *string* `"CAT=adj_nc"` só tem utilidade quando usada com -c, ou através da chamada da biblioteca `get_roots(...)`, dado que permite encontrar a classificação de uma palavra raiz, mesmo que esta não se encontre no dicionário.

4.3 Sufixos de sufixos

Outro acrescento foi o modificador de *flags* "+" que permite que a *flag* assim assinalada seja usada a seguir a outra *flag*, ou seja pode ser usado como um segundo sufixo.

ex.

```
flag X:
A R       >  -AR,O       ; "CAT=v,P=1,N=s,T=p"
flag +P:      ; "CAT=v"
          >      \-A     ; "G=f,N=s"
```

Neste caso, o palavra "amo-a" podia ser encontrada a partir da entrada do dicionário "amar" usando primeiro a *flag* X (obtendo-se a palavra "amo") seguida da *flag* P, que lhe acrescentaria a "-a".

5 Formato do dicionário

5.1 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
```

O comentário é opcional.

5.2 Mascarar Radicais

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 deveria ter apenas uma entrada no infinitivo. No entanto a existência de irregularidades tornam impossível o uso deste princípio em todos os casos, já que há palavras que derivam de outras de uma forma irregular que não pode ser regulada por uma *flag*. Para estes casos foi criado um mecanismo que permite que palavras sejam acrescentadas ao dicionário, tendo informação sobre o radical de que derivam de forma irregular:
palavra/$palavra raiz$classificação palavra raiz$ classificação 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])
```

5.3 Pontuação

Como no JSpell os sinais de pontuação são considerados palavras, e não são passados à frente, eles podem existir no dicionário num formato igual ao de qualquer outra entrada.

extrato do port.dic:

```
,/CAT=punct//
./CAT=punct//
:/CAT=punct//
;/CAT=punct//
?/CAT=punct//
```

5.4 Macro Substituição

Verficou-se que neste tipo de dicionários é muito frequente a repetição de *strings* na classificação; para uma maior economia de espaço, disponibiliza-se a possibilidade de macro substituição; que permite a definição de *shortcuts* para *strings*. As macros devem começar sempre por '#' (eventualmente redefenível no ficheiro *config.X*), e serem seguidos por letras e números que identificam a macro.
ex.

```
#vt/CAT=v,TR=t

abaixar/#vt/XYL/
```

é equivalente a:

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

5.5 Repetição de entradas

São também permitidas entradas ≪repetidas≫ no dicionário, i.e. entradas em que a palavra é igual, mas podendo ter classificação e *flags* diferentes, sendo estas correctamente tratadas. Assim a mesma palavra pode aparecer em 2 soluções independentes.