Desenvolvimento de Aplicações em Perl com FreeLing 3

Alberto Simões
Centro de Estudos Humanísticos
Universidade do Minho
ambs@ilch.uminho.pt

Nuno Carvalho
Departamento de Informática
Universidade do Minho
narcarvalho@di.uminho.pt

Resumo

O FreeLing é uma ferramenta para processamento de linguagem natural, em especial para análise morfossintáctica e cálculo de árvores de dependências. Embora a escolha de implementação em C++ seja relevante pela eficiência, torna complicado o desenvolvimento de pequenas ferramentas. Além disso, a interface Perl disponibilizada com o próprio FreeLing não é mais que um mapeamento directo da API C++ para Perl, o que não é o mais adequado.

Este artigo apresenta as decisões de implementação do módulo Perl `FL3`, e discute como esta interface torna simples a escrita de pequenos processadores de linguagem natural em Perl.

Palavras chave

FreeLing3, Perl

Abstract

FreeLing is a tool for processing natural languages, especially for morphological analysis and computation of dependency trees. Although C++ is a suitable language to implement this kind of tool given its efficiency, it makes it difficult to develop small tools. Also, the Perl interface available with the FreeLing package is not much more than a simple map from the C++ API to Perl, which isn't the most appropriate.

This article we presents some decisions made to implement the Perl module `FL3`, and discusses how this interfaces makes it easy to write small natural language processors in Perl.

Keywords

FreeLing3, Perl, Processamento de Linguagem Natural

1 Introdução

O FreeLing[1] (Padró, 2011; Padró et al., 2010) é uma biblioteca para a construção de analisadores (morfológicos, sintácticos, e outros) multilingues. A versão 3 inclui suporte para UTF-8 e um leque interessante de línguas, das quais realçamos as principais línguas ibéricas, o inglês e o russo.

A biblioteca é composta por classes que abstraem componentes linguísticos, como sejam parágrafos, frases, palavras ou mesmo análises morfológicas, que são usadas por outras classes que implementam algoritmos de análise morfológica com suporte para detecção de termos multipalavra (datas, números, locuções e entidades mencionadas), algoritmos de *tagging*, um baseado em *Relax* e um outro baseado em cadeias de Markov (HMM), um algoritmo de *parsing*, baseado em Charts; um algoritmo de cálculo de dependências (Atserias, Comelles e Mayor, 2005), entre outros.

O FreeLing está escrito em C++, o que lhe confere eficiência, mas que dificulta o desenvolvimento rápido de pequenas ferramentas, que se implementam mais rapidamente utilizando linguagens de programação ditas de *scripting*. A integração desta biblioteca em ferramentas já desenvolvidas noutras linguagens também se pode tornar complicada. Por estas razões, o FreeLing disponibiliza um conjunto de modelos para a geração de interfaces noutras linguagens, como Perl, Python ou Java. Estes modelos são usados pela ferramenta SWIG[2] (Simplified Wrapper and Interface Generator) para a geração de uma API (Application Programmer Interface) para cada uma dessas linguagens.

Esta abordagem, baseada em SWIG, permite ao programador da biblioteca a definição de interfaces para diferentes linguagens de uma forma rápida e uniforme, mas a API obtida é apenas um mapeamento directo entre as classes e métodos do FreeLing para a linguagem de destino. Esta interface nem sempre é a mais versátil, já que habitualmente não tira partido das funcionalidades nem filosofia da linguagem de destino.

Estas razões levaram à implementação de um módulo Perl (`Lingua::FreeLing3`) que abstrai a

[1] O FreeLing está disponível gratuitamente a partir de `http://nlp.lsi.upc.edu/freeling/`

[2] http://www.swig.org/

interface gerada pelo SWIG. Este módulo torna o desenvolvimento de ferramentas em Perl usando o FreeLing mais simples e rápido.

Este artigo está organizado como se segue: em primeiro lugar é explicada a arquitectura do módulo, e como este interage com a biblioteca FreeLing. Posteriormente apresentam-se algumas ferramentas desenvolvidas utilizando este módulo, juntamente com versões simplificadas do código Perl que as implementa[3]. É importante a apresentação deste código, já que só deste modo se poderá perceber como foi feita a abstracção da biblioteca FreeLing. Finalmente, apresentam-se algumas conclusões e planos de trabalho futuro.

2 Arquitectura do `Lingua::FreeLing3`

O desenvolvimento do `Lingua::FreeLing3` foi feito com base na interface obtida aplicando o modelo SWIG disponibilizado, que gera um módulo Perl que invoca funções de cola em C (Jenness e Cozens, 2002) para a ligação com a biblioteca. O `Lingua::FreeLing3` inclui um conjunto de módulos (ou classes) um para cada tipo de objecto ou ferramenta disponibilizada pelo FreeLing. Cada uma destas classes independentes em Perl define uma API mais sucinta e de alto nível (Dominus, 2005).

A figura 1 apresenta as várias camadas e de que forma elas interagem. As duas primeiras camadas correspondem à biblioteca FreeLing, e à API gerada pelo SWIG. Na camada seguinte estão os vários módulos de *objectificação* da API gerada pelo SWIG, bem como alguma gestão de memória que não é feita convenientemente pelo código gerado. Finalmente,uma classe de alto nível (`FL3`) para permitir o uso elegante dos vários algoritmos disponíveis.

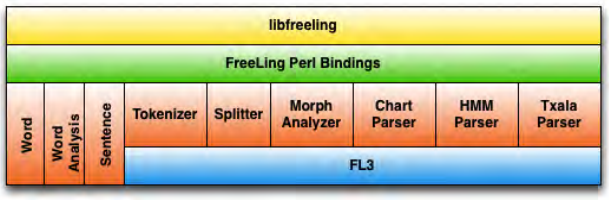

Figura 1: Níveis de Indirecção do FL3.

O módulo `FL3` simplifica a criação de processadores que tiram partido dos vários algoritmos de análise, usando um conjunto de opções por omissão.

[3]É importante salientar que se espera que o leitor tenho algum conhecimento do uso da linguagem de programação Perl, já que de outro modo se tornará difícil compreender completamente os exemplos apresentados. Além disso, os exemplos foram propositadamente simplificados em alguns aspectos, que não têm que ver com a interacção com a biblioteca FreeLing3.

3 Desenvolvimento com `Lingua::FreeLing3`

De modo a ilustrar o desenvolvimento de aplicações de processamento de linguagem natural usando o `Lingua::FreeLing3`, e o seu módulo de alto nível `FL3`, apresentam-se um conjunto de pequenas ferramentas úteis, e a sua implementação em Perl. Estas ferramentas não fazem parte do módulo `Lingua::FreeLing3`, mas estão disponíveis num módulo de ferramentas e utilidades: `Lingua::FreeLing3::Utils`.

3.1 N-Gramas

Em PLN um n-grama consiste num conjunto de sequências de n símbolos (tipicamente designados por *tokens*) consecutivos, calculados a partir de um texto. Apesar de estes *tokens* poderem ser de vários tipos, o mais comum é usarem-se n-gramas de caracteres ou de palavras, sendo estes últimos os mais usados para a criação de modelos de língua, associando o cálculo da probabilidade de ocorrência do respectivo n-grama no texto. A título de exemplo considere-se a seguinte frase: *"E o tempo responde ao tempo que o tempo tem tanto tempo quanto tempo o tempo tem."*. Podemos dividir a frase em 18 palavras (*tokens*), habitualmente designadas por formas. O cálculo de unigramas para este exemplo, bem como as respectivas probabilidades, são apresentados na tabela 1.

Unigramas	Ocor.	Probabilidade	
tempo	6	$P(tempo) = 6/18$	33%
o	3	$P(o) = 3/18$	17%
tem	2	$P(tem) = 2/18$	11%
E	1	$P(E) = 1/18$	6%
responde	1	$P(responde) = 1/18$	6%

Tabela 1: Unigramas e probabilidades associadas.

Para o cálculo das unigramas, o primeiro passo é dividir a frase em palavras (atomização ou *tokenization*), e o segundo, contar a ocorrência de cada uma das palavras. As respectivas probabilidades são calculadas dividindo o número de ocorrências pelo número total de palavras.

Efectuando o mesmo exercício mas para calcular bigramas obtemos alguns exemplos da tabela 2. O processo é similar, mas os *tokens* são agrupados dois a dois, e as respectivas probabilidades são calculados usando o teorema de Bayes.

Este processo é aplicado de forma semelhante para outros valores de n (Jurafsky et al., 2000).

Bigramas	Ocor.	Probabilidade	
o tempo	3	$P(tempo\|o) = 3/3$	100%
tempo tem	2	$P(tem\|tempo) = 1/3$	33%
tem tanto	1	$P(tanto\|tem) = 1/2$	50%

Tabela 2: Bigramas e probabilidades condicionadas associadas.

O cálculo de n-gramas é simples desde que seja possível calcular a sequência de palavras que compõem o texto. A tarefa complexa passa a ser calcular esta sequência de palavras, dado ser necessário lidar com uma série de problemas de solução não trivial: hifenização, sinais de pontuação, abreviaturas, etc.

O `FreeLing3` permite obter de forma relativamente fiável um conjunto de palavras a partir de um texto. Por exemplo, para obter a lista de *tokens* que formam o texto podemos simplesmente executar:[4]

```
my $tokens = tokenizer->tokenize($text);
```

A lista de símbolos torna-se disponível, e basta então percorrer essa sequência com uma janela deslizante de tamanho n para calcular uma lista de n-gramas.

Listagem 1: Cálculo de n-gramas.

```
# para todos os tokens da sequencia
for $c (0 .. @tokens - $n + 1)
{
  # calcular um n-gram com n elementos
  my @ngram = @$tokens[$c .. $c+$n-1];

  # calcular uma string...
  my $ngram = join(" ", @ngram);

  # incrementar o número de ocorrências
  $ngrams->{$ngram}{count}++;
}
```

Uma vez a lista de n-gramas e as respectivas ocorrências calculadas, o cálculo das probabilidades de cada um é apenas uma questão aritmética.

É bastante comum utilizar no cálculo de n-gramas os símbolos especiais `<s>` e `</s>` para marcar o início e fim de frase respectivamente, bastante úteis para rever quais as palavras com maior probabilidade nessas circunstâncias. Utilizando o `FreeLing3` é bastante fácil de produzir este efeito, uma vez que é possível a partir de uma lista de *tokens* obter uma lista de frases. A este processo chama-se normalmente segmentação (*segmentation*).

[4] É possível especificar a língua a usar passando-o como parâmetro: `tokenizer('en')->tokenize($text)`. Esta interface é semelhante para todas as outras funcionalidades do módulo.

Esta tarefa pode mais uma vez ser delegada para o `FreeLing3`. A partir de uma lista de *tokens* o FreeLing é capaz de os agrupar em frases:

```
my $sentences = splitter->split($tokens)
```

A listatem 2 ilustra o código necessário para implementar este processo.

Listagem 2: Adicionar símbolos de início e fim de frase.

```
# calcular tokens
my $tokens = tokenizer->tokenize($txt);

# calcular frases
my $sentences = splitter->split($tokens);

# para cada frase
foreach my $s (@$sentences) {
  # construir nova frase
  $s = sentence(word('<s>'),
                @$s,
                word('</s>'));
}
```

O resto do cálculo dos n-gramas é feito de modo análogo ao descrito anteriormente. A maior diferença nos resultados é que se passa a obter os novos *tokens*. Para o exemplo anterior passa a existir o bigrama "`<s>` O," que representa a palavra "O" no início da frase.

As operações descritas nesta seção foram implementadas numa ferramenta chamada `fl3-ngrams`. Segue-se um exemplo de execução para cálculo de bigramas:

```
$ fl3-ngrams -n 2 input.txt
# n-gram    count   prob
tempo o     1       0.16666667
tempo tem   2       0.33333333
E o         1       1.00000000
(...)
```

3.2 Análises Morfológicas

Não é uma aplicação muito habitual e, possivelmente, não muito útil por si só. Serve, no pior do casos, para consultar o dicionário de análise morfológica. Pretende-se uma aplicação que, dada uma palavra (ou uma sequência delas), nos permita obter todas as análises morfológicas possíveis para cada palavra. Ou seja, esta análise será não desambiguada.

No caso concreto desta aplicação não são precisas as fronteiras das frases, apenas de converter o texto num conjunto de palavras que possam ser analisadas. O código que se segue faz o processamento linha a linha, invocando a função `word_analysis` para obter as análises de cada pa-

lavra[5].

Listagem 3: Processamento de ficheiro para Análise Morfológica de Palavras

```
# processar cada linha
while (my $l = <>) {
  # atomizar a linha
  my $wrds = tokenizer->tokenize($l);

  # analisar cada palavra
  my @ws = word_analysis(@$wrds);

  # apresentar resultados
  while (@ws) {
    my $w = shift @$wrds;
    my $a = shift @ws;

    # apresentar forma da palavra
    print $w->form;

    # apresentar lema e POS possíveis
    for my $x (@$a) {
      print " [$x->{lemma}, $x->{tag}]"
} } }
```

Além disso, não se pretende que o Analisador Morfológico use informação de contexto e tente, por exemplo, fazer reconhecimento de entidades mencionadas ou locuções. Logo, é necessário desligar todas essas opções na função de cálculo de análises.

Listagem 4: Cálculo das Análise Morfológica

```
sub word_analysis {
  my @words = @_;

  # desactivar detecção de multipalavras
  my %conf = (
    ProbabilityAssignment  => 'no',
    QuantitiesDetection    => 'no',
    MultiwordsDetection    => 'no',
    NumbersDetection       => 'no',
    DatesDetection         => 'no',
    OrthographicCorrection => 'no',
    NERecognition          => 'no'
  );

  # criar frase artificial para analisar
  my $a = morph(%conf)->analyze(
     [ sentence(@words) ]
  );

  # converter cada conjunto de análises
  # numa lista de facetas
  return map {
    $_->analysis(FeatureStructure => 1)
  } $a->[0]->words;
}
```

3.3 NLGrep

Uma das aplicações típicas de corpos anotados é a pesquisa de concordância, procurando por determinado fenómeno linguístico, ou apenas para analisar o contexto de determinada palavra ou forma morfológica. Este processo é simples de se realizar depois de o corpo estar devidamente processado e disponível num motor adequado, como por exemplo o IMS Open Corpus Workbench (Christ, 1994).

Mas pode ser útil a pesquisa rápida sobre um texto não anotado, realizando anotação dinamicamente. Cada frase é anotada e o padrão indicado é procurado. Se for satisfeito, é apresentado o fragmento da frase que está de acordo com o padrão.

É certo que esta abordagem não é a melhor se o objectivo for realizar várias pesquisas sobre o mesmo texto, já que o texto será anotado para cada procura. Para resolver este problema é possível a criação de uma *cache* com o documento anotado, para ser usada em futuras execuções de modo a poupar tempo.

A linguagem de padrões implementada é relativamente simples, suportando apenas três tipos de condição por palavra: pesquisa de palavra exacta (`=palavra`); pesquisa por lema (`~lema`); ou pesquisa por (prefixo de) faceta morfológica (usando a etiqueta EAGLES `NCMS`, nome comum, masculino singular). Além disso, é possível usar o carácter sublinhado (`_`) como posição a associar a qualquer palavra (*wildcard*).

De seguida apresenta-se um exemplo do uso desta ferramenta, para a língua portuguesa, procurando num livro do Projecto Gutenberg. A expressão de pesquisa usada corresponde a uma forma do verbo "*ser*", seguida de um qualquer verbo, seguido da palavra "*o*", seguida de qualquer outra palavra:[6]

```
$ fl3-nlgrep pg33056.txt ~ser V =o _
era levantar o edificio
```

Embora extremamente simples, esta aplicação permite obter alguns resultados interessantes, como por exemplo, verificar que adjectivos são habitualmente usados com conjunções:

```
$ fl3-nlgrep -l pt pg33056.txt ~ser A C A
era grosso e baixo
era excellente e detestavel
é pura e severa
Sou exclusivo e pessoal
era orgulhoso e fraco
é independente e superior
era grande e vistosa
era justo nem bonito
é trivial e chocho
era restricta e mansa
```

[5]Neste momento a análise morfológica obtida é a etiqueta EAGLES usada no dicionário.

[6]Esta pesquisa concreta para o texto relativamente pequeno teve apenas um acerto.

A implementação desta ferramenta é igualmente simples. Para além do processamento típico já usado nas ferramentas anteriores (atomização, segmentação e anotação morfológica), o `fl3-nlgrep` adiciona uma nova camada de processamento usando um etiquetador (*tagger*). O FreeLing tem suporte para dois algoritmos de etiquetação, um baseado em cadeias de Markov (HMMTagger) e um outro baseado em Relax (RelaxTagger).

Listagem 5: Implementação do nlGrep

```
# inicializar analisador morfológico
morph(ProbabilityAssignment  => 'yes',
      QuantitiesDetection    => 'no',
      MultiwordsDetection    => 'no',
      NumbersDetection       => 'no',
      DatesDetection         => 'no',
      OrthographicCorrection => 'no',
      NERecognition          => 'no');

# abrir ficheiro a processar
open my $fh, "<:utf8", $filename;

# processar cada linha/frase
while (my $l = <$fh>) {
  my ($tokens, $frases);

  # de texto obter palavras
  $tokens=tokenizer->tokenize($l);
  # de palavras agrupar em frases
  $frases=splitter->split($tokens);
  # anotar morfologicamente
  $frases=morph->analyze($frases);

  # etiquetar usando cadeiras Markov
  $frases = hmm->tag($frases);

  # para cada frase
  for my $frase (@$frases) {
    my @words = $frase->words;

    # janela deslizante
    while (@words > @query) {
      if (match(\@words, \@query)) {
        show_match(@words[0..$#query])
      }
      shift @words;
} } }

# imprime as palavras.
sub show_match {
  print join(" ",map{$_->form} @_),"\n"
}

# verifica palavras contra expressão
# de pesquisa
sub match {
  for my $i (0 .. $#query) {
    # ignorar palavra se wildcard
    next if $query->[$i] eq "_";

    # se procuramos palavra exacta
    if ($query->[$i] =~ /^=(.*)$/) {
      if ($1 ne $words->[$i]->lc_form) {
        return 0
    } }
    # se procuramos por lema
    elsif ($query->[$i] =~ /^~(.*)$/) {
      if ($1 ne $words->[$i]->lemma) {
        return 0
    } }
    # caso contrário, etiqueta POS
    else {
      my $tag = $words->[$i]->tag;
      if ($tag !~ /^$query->[$i]/i) {
        return 0
  } } }
  return 1;
}
```

3.4 Extractor de EM

O último exemplo que aqui se apresenta tira partido de uma das funcionalidades do analisador morfológico do FreeLing, que é a detecção de multi-palavras, sejam quantidades, números, datas, locuções, ou genericamente, nomes próprios. Para além disso, o FreeLing incorpora um classificador de entidades mencionadas que é capaz de distinguir entre nomes próprios de pessoas, de organizações e de locais geográficos (bem como uma classe genérica, para todos outros tipos de classificação).

É importante realçar que nos exemplos apresentados não estamos a tentar demonstrar a qualidade, ou falta dela, do FreeLing. Note-se que ao aplicar várias ferramentas em cascata a percentagem de erro aumenta, como bola de neve. Além disso, os textos exemplos são de português antigo, e os ficheiros para detecção de entidades para a língua Portuguesa podem ainda levar bastantes melhorias.

As camadas de processamento utilizadas nesta ferramenta incluem, tal como nas anteriores, o atomizador e o segmentador. O analisador morfológico, com o módulo de detecção de nomes ligado e, finalmente, o módulo de classificação de entidades.

Listagem 6: Extracção de EM

```
# inicializar analisador morfológico
morph(ProbabilityAssignment  => 'yes',
      QuantitiesDetection    => 'no',
      MultiwordsDetection    => 'yes',
      NumbersDetection       => 'no',
      DatesDetection         => 'no',
      OrthographicCorrection => 'no',
      NERecognition          => 'yes');

# nomes das classes
my %classes = (NP00SP0 => 'Person',
               NP00G00 => 'Geographical',
               NP00O00 => 'Organization',
               NP00V00 => 'Others');

open my $fh, "< :utf8", $filename;
```

```
# processar cada linha...
while (my $l = <$fh>) {
  $tokens = tokenizer->tokenize($l);
  $frases = splitter->split($tokens);
  $frases = morph->analyze($frases);

  # classificar as entidades
  $frases = nec->analyze($frases);

  # fazer estatísticas das classificações
  for my $frase (@$frases) {
    for my $word ($frase->words) {

      # processar multi-palavras das
      # classes relevantes
      if ($word->is_multiword &&
          exists($classes{$word->tag})) {

        my $class = $classes{$word->tag};
        my $fw = $word->get_mw_words();
        $counts{$fw}{_}++;
        $counts{$fw}{$class}++;
      }
    }
  }
}

# imprimir resultados...
for my $mw (keys %counts) {
  print $mw;
  for my $clss (keys %{$counts{$mw}}) {
    next if $clss eq "_";
    printf "\t$clss (%.4f)",
      $counts{$mw}{$clss}/$counts{$mw}{_}
  }
  print "\n";
}
```

De seguida apresentam-se alguns exemplos. Note-se que por falta de espaço removeram-se alguns falsos positivos[7].

```
$ fl3-ner -l pt pg33056-pt.txt
João Braz         Per (0.80) Org (0.20)
Sacco do Alferes  Geo (1.00)
Sophia            Per (0.91) Geo (0.09)
Santa Thereza     Per (1.00)
Petropolis        Org (0.50) Geo (0.50)
Oriente           Geo (1.00)
Julia Costinha    Per (1.00)
Macbeth           Oth (1.00)
Joaquim           Per (1.00)
Luthero           Per (1.00)
Israel            Geo (1.00)
```

4 Conclusões e trabalho futuro

Este artigo apresenta o módulo Perl `Lingua::FreeLing3`, como método de interagir com a biblioteca FreeLing, e como, de forma simples, é possível implementar ferramentas úteis e relevantes.

Neste momento, estas quatro ferramentas estão a ser melhoradas e agrupadas num outro módulo Perl, de nome `Lingua::FreeLing3::Utils`, com o objectivo de disponibilizar algumas ferramentas básicas de PLN que actualmente não estão disponíveis para a linguagem Perl.

Agradecimentos

O trabalho aqui apresentado foi parcialmente financiado pelo projecto da Fundação para a Ciência e Tecnologia PTDC/CLE-LLI/108948/2008.

Os autores também agradecem a paciência e disponibilidade do principal autor do FreeLing, Lluís Padró. Só com a sua ajuda foi possível resolver vários problemas encontrados durante a implementação deste módulo.

Referências

Atserias, Jordi, Elisabet Comelles, e Aingeru Mayor. 2005. TXALA un analizador libre de dependencias para el castellano. *Procesamiento del Lenguaje Natural*, 35:455–456, Setembro, 2005.

Christ, O. 1994. A Modular and Flexible Architecture for an Integrated Corpus Query System.

Dominus, Mark Jason. 2005. *Higher-Order Perl: Transforming Programs with Programs*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Jenness, Tim e Simon Cozens. 2002. *Extending and Embedding Perl*. Manning Publications, August, 2002.

Jurafsky, D., J.H. Martin, A. Kehler, K. Vander Linden, e N. Ward. 2000. *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*, volume 2. Prentice Hall New Jersey.

Padró, Lluís. 2011. Analizadores multilingües en freeling. *Linguamática*, 3(2):13–20, Dezembro, 2011.

Padró, Lluís, Miquel Collado, Samuel Reese, Marina Lloberes, e Irene Castellón. 2010. FreeLing 2.1: five years of open-source language processing tools. La Valletta, Malta, Maio, 2010. ELRA, Proceedings of 7th Language Resources and Evaluation Conference (LREC 2010).

[7] Como já explicamos, neste artigo não pretendemos demonstrar a qualidade do FreeLing, apenas apresentar a interface Perl desenvolvida.