Corte e costura no AC/DC: auxiliando a melhoria da anotação nos corpos

Cristina Mota
cmota@ist.utl.pt

16 de Abril de 2014

1 Apresentação

Este documento – que é uma versão actualizada de [7] – descreve o `corte-e-costura`, um programa criado no âmbito do projecto AC/DC [10, 11, 2] com o fim de auxiliar a anotação humana dos corpos com informação referente a campos semânticos.

Mais especificamente, o programa aplica um conjunto de regras a um corpo previamente anotado pelo PALAVRAS[1] e transformado no "formato AC/DC"[8, 9]. As regras, estabelecidas pelos anotadores num formato o mais próximo possível do usado para interrogar os corpos (o formato do IMS-CWB), modificam, eliminam ou acrescentam novos atributos à anotação existente no corpo.

Como já referido, as regras foram primordialmente definidas para melhorar a anotação semântica, ou seja, para modificarem os atributos `sema` e `grupo`. Contudo, muitos dos erros a corrigir da anotação semântica automática eram provenientes de uma deficiente anotação sintáctica inicial, donde o programa `corte-e-costura` foi pensado e utilizado para estabelecer tantas correcções quantas as que fosse possível especificar.

Por outro lado, outras funcionalidades foram sendo acrescentadas, aumentando-se a expressivade das regras. Assim, as regras podem também ser usadas para identificar casos de expressões com mais de uma palavra que devam ser consideradas como uma unidade em termos semânticos, o que corresponde, do ponto de vista do formato AC/DC, à marcação do atributo estrutural `<mwe>`, e modificar os seus atributos. Além disso, também é possível apagar tanto elementos estruturais como posicionais.

O primeiro caso a ser considerado, e aquele em que a maioria dos exemplos deste texto são apresentados, foi o campo da cor [14], mas o programa foi desenhado para permitir a melhoria da anotação a partir de regras independentemente dos campos a que elas se referem. Serão igualmente dados vários exemplos referentes a outros campos semânticos. Até ao momento, o `corte-e-costura` foi também usado para anotar vestuário [12], emoções [6] e partes do corpo [3].

A forma como o `corte-e-costura` é usado na anotação do campo semântico, em combinação com os programas `alinhavo`, que anota as palavras que à partida pertencem sempre ao campo semântico que se está a anotar, e `remate`, que acrescenta os grupos, é apresentado em [6].

No restante documento, apresentamos o manual de utilizador do `corte-e-costura`, com descrição detalhada de todas as opções que podem ser usadas na sua chamada, e alguma documentação técnica.

2 Manual do utilizador

O `corte-e-costura` é um programa escrito em Perl, que lê e aplica regras a um corpo no formato do AC/DC, e gera um novo corpo no mesmo formato resultante da aplicação das regras.

Para aplicar um conjunto de regras a um corpo, o programa é invocado com os seguintes argumentos:

```
acdc_corte-e-costura.pl
  -r REGRAS -p POSIÇÃO [-l LOG] [-i RECURSÃO] [-d DEPURAR] [-a AVISAR] [CORPO]
```

O novo corpo, que resulta da aplicação das regras, é escrito no STDOUT.

Caso se deseje ver uma breve descrição do programa e dos seus parâmetros, terminando a sua execução em seguida, o mesmo deverá ser invocado da seguinte forma:

```
acdc_corte-e-costura.pl -h
```

2.1 Descrição dos argumentos

Com excepção do `CORPO` que, quando é dado, tem sempre de ser o último argumento, todos os outros argumentos podem ser usados por qualquer ordem; apenas -r e -p são obrigatórios.

`REGRAS` é o nome do ficheiro que contém as regras que vão ser aplicadas ao corpo;

`CORPO` é o nome do ficheiro com o corpo a que vão ser aplicadas as regras; caso `CORPO` não seja dado, este é lido do STDIN;

`POSIÇÃO` é o índice da coluna onde se encontra o lema. Por exemplo, 5 para o CONDIV, e 1 para o ENPCPUB;

`LOG` é o nome do ficheiro para escrever um registo de aplicação das regras; caso este pârametro não seja dado, o registo é feito no STDERR;

`RECURSÃO` toma o valor 0 ou 1; com valor 1 repete a aplicação das regras à frase até não haver diferenças antes e depois da aplicação das regras; com 0 ou omitindo o parâmetro aplica as regras a cada frase apenas uma vez.

`DEPURAR` com o valor 1 escreve no ficheiro de registo o formato interno de todas as regras que foram lidas do ficheiro de regras; se for 0 ou estiver omitido não regista a representação interna.

`AVISAR` com o valor 1 escreve no ficheiro de registo mensagens de aviso de possíveis conflitos entre as regras. Também alerta para a existência de caracteres especiais não protegidos (ver secção 2.3.4).

2.2 Exemplos

```
acdc_corte-e-costura.pl -r regras.txt -p 3 corpo.txt
cat corpo.txt | acdc_corte-e-costura.pl -r regras.txt -p 3
```

Duas formas diferentes de invocar o mesmo comando produzindo o mesmo resultado. Gera um novo corpo por aplicação das regras do ficheiro `regras.txt` ao corpo `corpo.txt`. O lema encontra-se na posição 3 de `corpo.txt`. As regras que disparam são mostradas no STDERR indicando a linha do ficheiro original que possibilitou a activação da regra.

```
acdc_corte-e-costura.pl -r regras.txt -p 3 -l log.txt corpo.txt > novo.txt
```

Cria um ficheiro (`novo.txt`) que corresponde ao resultado de aplicar as regras representadas em `regras.txt` a `corpo.txt`. O registo das regras activadas é feito no ficheiro `log.txt`.

```
# --------------------------------
# Exemplo de um ficheiro de regras
# --------------------------------

## Regras implícitas para delimitar expressões multi-palavra

## sema=cor pos=ADJ
amarelo sol
azul alentejano

## sema=corpo pos=N
abóbada de+o pé :: abóbada do pé[]
braço[:outros] armado

## Regras de correcção
[word="Terroso" & lema="terroso"] >> [pos="PROP" & lema="Terroso"]
[word="Branco" & lema="branco"] >> [pos="PROP" & lema="Branco"]

[lema="cor"] a:[lema="aço"] >> a:[pos="ADJ"]

## Regras de marcação semântica
[lema="cor"] [lema="de"] z:[lema="laranja"] [lema="e"] b:[lema="amarelo"] >>
    z:[sema="cor"] b:[sema="cor"]

[lema="máscara"] a:[lema="cor"] [lema="de"] b:[lema="laranja"]  >>
    <mwe lema="cor=de=laranja" sema="cor"> a: b: </mwe> a:[sema="cor"]
```

Figura 1: Exemplo de ficheiro de regras

2.3 Formato do ficheiro de regras

O ficheiro de regras é constituído por uma regra em cada linha, podendo haver uma ou mais linhas de separação vazias entre as regras (ver figura 1).

Qualquer linha começada por `#` é considerada um comentário e, consequentemente, é ignorada pelo programa. A única excepção é quando a linha é iniciada por `#` e inclui logo a seguir os campos `sema` e `pos`, como se pode ver na figura 1. O valor dos atributos é usado para auxiliar a anotação de expressões multi-palavra quando estas são fornecidas em formato de lista de palavras, sendo obrigatório nesse caso que o ficheiro inclua pelo menos uma linha deste tipo, imediatamente antes de uma lista de palavras (a secção 2.3.2 descreve com maior detalhe o formato desta lista de palavras).

Cada regra é constituída por um antecedente e um consequente separados por `>>`. Deve existir pelo menos um espaço antes e depois do separador `>>` Tanto o antecedente como o consequente são constituídos por um ou mais termos.

É possível omitir o consequente quando se trata de uma regra para delimitar novas expressões multi-palavra sem usar contexto. Nesse caso, o antecedente limita-se a ser constituído pela expressão multi-palavra que se pretende anotar. Exemplo: `cor de rosa`

Formato dos termos válidos em ambos os lados da regra:

```
REFERÊNCIA:[ATRIBUTO="VALOR"( & ATRIBUTO="VALOR")*]
REFERÊNCIA:<mwe( ATRIBUTO="VALOR")*>
<mwe( ATRIBUTO="VALOR")*>
</mwe>
```

Formato dos termos válidos apenas do lado do antecedente:

```
[ATRIBUTO="VALOR"( & ATRIBUTO="VALOR")*]
*
PALAVRA( PALAVRA)* # Apenas se não tiver consequente, ou seja em regras implícitas.
```

Formato dos termos válidos apenas do lado do consequente:

```
REFERÊNCIA:
REFERÊNCIA:APAGAR
```

Formato dos termos válidos apenas em regras implícitas:

```
PALAVRA{[SEMA|:SEMA]}( PALAVRA{[SEMA|:SEMA]})*
PALAVRA{[SEMA|:SEMA]}( PALAVRA{[SEMA|:SEMA]})* :: PALAVRA( PALAVRA)*
```

Descrição das variáveis

`REFERÊNCIA` Qualquer letra (minúscula) de `a` a `z` usada para nomear um termo no antecedente de modo a que se possa referenciar esse termo no consequente com o fim de modificar a sua anotação. A referência tem de ser única no antecedente e pode ser usada mais do que uma vez no consequente. Tanto no antecedente como no consequente as referências podem ser usadas por qualquer ordem e não precisam ser sequenciais. Caso o consequente sirva para delimitar expressões multi-palavra, então a referência deve ser usada sem especificar o par atributo-valor (ver secção 2.3.2).

A referência pode ser omitida em cada uma das seguintes situações:

- existe apenas um termo em cada lado da regra. Exemplo:

  ```
  [word="Marinho" & lema="marinho"] >> [pos="PROP" & lema="Marinho"]
  ```

- no antecedente, quando o termo não é modificado pelo consequente. Exemplo:

  ```
  [lema="cor"] a:[lema="aço"] >> a:[pos="ADJ"]
  ```

- no consequente, quando os termos estruturais são usados para indicar os limites de novas expressões multi-palavra. Exemplo:

  ```
  a:[lema="azul"] b:[lema="marinho"] >> <mwe lema="azul=marinho"> \
    a: b: </mwe>
  ```

`ATRIBUTO` Qualquer dos atributos previstos para o formato usado no AC/DC.

`VALOR` Valor do atributo no corpo; caso, no consequente, tenha o valor `0` e o atributo seja `sema` então verifica se o atributo `grupo` também fica a `0`, pois caso não fique escreve uma mensagem de aviso no ficheiro de registo. Ver secção 2.3.4 para informações sobre a utilização de caracteres especiais nesta varíavel.

`PALAVRA` Palavra de um composto. Quando se usa uma sequência de palavras para expressar o antecedente, esta não pode ser usada em combinação com mais nenhum tipo de termo e não pode existir consequente. A secção 2.3.2 explica em maior detalhe alguns parâmetros que podem ser usados quando se especifica listas de multi-palavras.

* Termo usado no antecedente para especificar expressões multi-palavra com comprimento variável. É obrigatório que seja especificado entre marcadores estruturais, tal como se ilustra nos seguintos exemplos:

```
<mwe> * </mwe>
<mwe pos="N"> * </mwe>
```

2.3.1 Regras para modificar os atributos de unidades

As regras para modificar atributos de unidades existentes no corpo são sempre constituídas por antecedente e consequente.

Existem dois tipos de unidades nos corpos do AC/DC: posicionais e estruturais. Tanto o antecedente como o consequente da regra podem ser formados combinando termos de ambos os tipos, o que permite alterar atributos de ambos os tipos ao mesmo tempo.

Exemplos:

```
a:[lema="camisa"] b:[lema="salmão" & pos="N"] >> a:[sema="roupa"] \
  b:[pos="ADJ" & gen="F" & sema="cor"]
```

Neste exemplo, o programa adiciona (ou substitui, se já existir) o atributo `sema` do termo referido por `a`, "camisa", com o valor "roupa" e os atributos `pos`, `gen` e `sema` do termo referido por `b`, "salmão", com os respectivos valores indicados no consequente.

```
[pos="N"] a:<mwe pos="N"> b:[lema="cor"] [lema="de"] [lema="laranja"] </mwe> >> \
  a:<pos="ADJ"> b:[sema="cor"]
```

Este exemplo modifica o atributo `pos` do marcador estrutural que delimita a expressão multi-palavra "cor de laranja" e além disso adiciona (ou modifica) o atributo `sema` do primeiro elemento do composto, "cor".

Se o atributo a modificar for `sema` e o novo valor for `0` então além de actualizar o atributo no corpo com esse valor, o programa verifica se o valor do grupo também ficou a `0`. Caso não tenha ficado, é escrita uma mensagem de aviso no ficheiro de registo.

2.3.2 Regras de delimitação de compostos

Existem duas formas de delimitar compostos: implicitamente e explicitamente.

De forma implícita, a regra é geralmente constituída apenas pela sequência de lemas que se deseja delimitar. Exemplo:

```
cor de açúcar queimado
```

No início da lista de palavras correspondente a um campo semântico diferente, deve ser incluída uma linha iniciada pelo símbolo de comentário `#` seguido dos atributos `sema` e `pos`, associados aos valores com que se deve anotar no corpo os compostos dessa lista, bem como o primeiro constituinte do composto (apenas em relação ao `sema`).

Exemplo:

```
# sema=cor pos=ADJ
amarelo canário
amarelo limão
amarelo ouro

# sema=roupa pos=N
fato de macaco
chapéu de palha

# sema=corpo:faculdade pos=N
ouvido musical
ouvido absoluto
```

Quando não se quer atribuir o campo semântico definido no início da lista de palavras à primeira palavra do composto, quer por se querer especificar um outro valor para o campo semântico quer por se querer atribuir o campo semântico a outra ou outras palavras do composto, cada

palavra à qual deva ser associado o campo semântico pré-definido deve ser seguida por `[]`, ou por `[CAMPO_SEMANTICO]` para atribuir um campo diferente do pré-definido. Caso o campo semântico seja o mesmo, mas a classe seja mais específica, antecede-se o `CAMPO_SEMANTICO` por `:`. Ilustram-se os vários casos a seguir:

```
# sema=corpo pos=N

dor de cotovelo[]
ter ouvido[:faculdade]
cabelo[] branco[cor]
```

Finalmente, quando o lema que se deseja atribuir ao composto é diferente do lema que está anotado no corpo, explicita-se o novo lema a seguir ao lema que deverá estar no corpo, separando um do outro por `::`. Exemplo:

```
segundo mão :: segunda mão
```

Isto indica que os lemas a procurar no corpo são "segundo mão", mas o lema do composto deverá ser "segunda mão".

A regra seguinte exemplifica uma regra explícita sem contexto, que é equivalente à definida mais acima implicitamente para `cor de açúcar queimado`:

```
a:[lema="cor"] [lema="de"] [lema="açúcar"] b:[lema="queimado"] >> a:[sema="cor"] \
<mwe lema="cor=de=açúcar=queimado" pos="ADJ" sema="cor"> a: b: </mwe>
```

A regra seguinte exemplifica uma regra explícita com contexto:

```
[lema="máscara"] a:[lema="cor"] [lema="de"] b:[lema="laranja"]  >> \
a:[sema="cor"] <mwe> a: b: </mwe>
```

Como se mostra nos dois últimos exemplos, quando a delimitação dos compostos é feita de forma explícita, o primeiro e o último constituinte do composto devem ser precedidos no antecedente por uma referência. Desse modo será possível indicar no consequente que se deve colocar o marcadores estruturais `<mwe>` e `</mwe>`, respectivamente, antes e depois desses constituintes.

Se a expressão multi-palavra que se pretende anotar já existir no corpo, então o programa em vez de adicionar os marcardores estruturais faz a fusão das propriedades existentes na expressão no corpo com as especificadas no consequente da regra: substitui os valores dos atributos que já existiam e adiciona os novos que ainda não existirem.

Ao mesmo tempo que se adiciona os marcadores também é possível modificar qualquer outro termo presente no antecedente. No exemplo anterior, o primeiro elemento do composto recebe o campo semântico `cor`.

2.3.3 Regras para eliminar elementos estruturais ou posicionais

Qualquer elemento, seja estrutural ou posicional, pode ser eliminado usando o termo APAGAR a seguir à referência desse elemento no consequente. Exemplo:

```
a:<mwe> [lema="cor"] [lema="de"] [lema="laranja"] b:</mwe> >> a:APAGAR b:APAGAR

a:[lema="cor"] b:[lema="de"] c:[lema="laranja"] >> \
  a:[word="cor=de=laranja" & lema="cor=de=laranja"] b:APAGAR c:APAGAR
```

No primeiro caso, a delimitação da expressão multi-palavra será eliminada do corpo. No segundo caso, são eliminadas as unidades cujos lemas são "de" e "laranja" e adicionalmente a forma "cor" é transformada em "cor=de=laranja", bem como o seu lema.

2.3.4 Protecção de símbolos especiais no valor de atributos

Caso seja necessário incluir caracteres com significado especial para o `corte-e-costura` ou para o IMS-CWB, como sejam:

`., [, ], |, <, >, *, ?, {, }, \`

estes devem ser precedidos por \. Para verificar se o ficheiro de regras contém caracteres especiais não protegidos, deve-se invocar o `corte-e-costura` adicionalmente com o argumento opcional `-a 1`. Se forem encontradas regras com caracteres especiais não protegidos, o `corte-e-costura`, avisa e termina a execução sem aplicar as regras ao corpo. Depois de confirmar que os caracteres não protegidos estão a ser bem usados, o `corte-e-costura` deve ser invocado sem `-a 1` ou com com `-a 0`.

3 Documentação técnica

Nesta pequena secção indicamos brevemente a forma como o programa foi idealizado, quais os requisitos para a sua invocação, problemas possíveis, e melhorias que terão de ficar para o futuro.

3.1 Descrição do funcionamento

Dado um corpo, C, e um ficheiro de regras, R, o programa é executado de acordo com o algoritmo incluído na figura 2.

```
.Converter as regras de R num formato interno R'
. Para cada frase F do corpo C aplicar regras R' da seguinte forma:
.. F'<- F
.. Enquanto houver diferenças entre F e F' repetir:
... Para cada elemento E de F executar:
.... Para cada regra de R' verificar:
..... Se a regra é activada a partir do elemento E então executa consequente da
      regra modificando F
..... Senão passa à próxima regra
.. Apresentar F como resultado
```

Figura 2: Algoritmo de processamento do corpo

As regras são todas lidas uma única vez do ficheiro de regras, sendo guardadas em memória durante a execução do programa. Cada regra é convertida num formato interno, em que os antecedentes são guardados num vector e os consequentes noutro vector. Este formato interno é semelhante ao formato em que as regras são escritas, mas facilita a aplicação das regras.

Se existir alguma regra que após convertida para o formato interno tem o mesmo antecedente de uma regra lida anteriormente, o programa alerta que foi encontrada uma regra nessas condições, mas apenas no caso de o argumento `-a 1` ter sido especificado.

3.1.1 Formato interno das regras

Cada termo do antecedente e do consequente é convertido em um ou mais termos indexados por um algarismo que indica a sua posição na sequência de termos que constitui o antecedente. Assim, uma regra como:

```
[word="Marinho" & lema="marinho"] >> [pos="PROP" & lema="Marinho"]
```

é convertida no seguinte formato interno:

```
0[word="Marinho"] 0[lema="marinho"] >> 0[pos="PROP"] 0[lema="Marinho"]
```

Isto quer dizer que cada atributo de um mesmo termo no antecedente da regra original irá constituir um novo termo no antecedente da regra no formato interno, e será indexado pelo mesmo algarismo. Esse algarismo representa a posição do termo original no antecedente. Uma vez que só existe um termo no antecedente e no consequente, o termo do consequente é igualmente indexado pelo algarismo `0`.

A regra seguinte:

```
<mwe> [word="azul"] [word="prússia"] </mwe> [lema=","] c:[lema="sangue-de-boi"] \
  >> c:[sema="cor"]
```

é por sua vez transformada em:

```
0<mwe.*> 1[word="azul"] 2[word="prússia"] 3</mwe> 4[lema=","] \
  5[lema="sangue-de-boi"] >> 5[sema="cor"]
```

Neste caso, o termo do consequente no formato interno é indexado pelo índice numérico do termo que é referido no antecedente por `c`, uma vez que é essa a referência do único termo no consequente. Além disso, o termo `<mwe>` do antecedente foi convertido em `<mwe.*>` para expressar que este termo deverá emparelhar com qualquer marcador estrutural `<mwe>` no corpo quaisquer que sejam os seus atributos.

A seguinte regra, que insere uma nova unidade multi-palavra:

```
a:[word="peito"] [word="de"] b:[word="rola"] [word="e"] c:<mwe> \
  d:[word="aurora"] [word="boreal"] </mwe> >> a:[sema="cor"] \
  <mwe lema="peito=de=rola" pos="N" sema="cor"> a: b: </mwe> \
  c:<lema="aurora=boreal" sema="cor"> d:[sema="cor"]
```

é convertida internamente em:

```
0[word="peito"] 1[word="de"] 2[word="rola"] 3[word="e"]
  4<mwe.*> 5[word="aurora"] 6[word="boreal"] 7</mwe> >>
  0[sema="cor"] 0<mwe> 0<lema="peito=de=rola"> 0<pos="N">
  0<sema="cor"> 2</mwe> 4<lema="aurora=boreal"> 4<sema="cor"> 5[sema="cor"]
```

Dado que se trata da inserção de uma multi-palavra, os termos `<mwe...>` e `</mwe>` do consequente serão indexados pelos índices numéricos das referências que se encontram delimitadas por esses marcadores, respectivamente. Indicando que o termo `0` deverá ser antecedido de `<mwe>` e o termo `2` deverá ser precedido por `</mwe>`.

Finalmente, com a regra:

```
a:<mwe sema="cor"> * </mwe> [lema="ou"] [lema="de"] d:[lema="pêssego"] >> \
  d:[sema="cor"]
```

ilustramos a conversão de uma regra cujo antecedente inclui uma multi-palavra de comprimento variável, e que é convertida em:

```
0<mwe.*> 0<sema="cor"> 1* 2</mwe> 3[lema="ou"] 4[lema="de"] \
  5[lema="pêssego"] >> 5[sema="cor"]
```

Dado que, no momento em que a regra é lida, não se sabe ainda o comprimento da multi-palavra, o que depende das multi-palavras a que se aplicar, os termos a seguir a `*` são normalmente indexados sequencialmente a partir do índice de `*`. No entanto, quando se aplica a regra, o termo `1*` vai emparelhar enquanto não se encontrar o marcador final `</mwe>`, e os índices dos termos seguintes terão em conta o comprimento da unidade multi-palavra que se encontra no corpo (ver secção 3.1.2).

Verificação do formato das regras Mesmo que uma das regras não verifique o formato estabelecido, o programa continua a ler as regras seguintes até ter lido todas as regras. No entanto, sempre que o programa encontra uma regra não válida produz uma mensagem de erro, e no fim de ter lido todas as regras, caso tenha sido encontrado alguma regra inválida, a execução do programa é interrompida.

As regras nas seguintes condições também serão consideradas inválidas e uma mensagem de erro a indicá-lo será produzida:

- Repitação de referências no antecedente;
- Uso de referências no consequente que não existem no antecedente;

3.1.2 Aplicação das regras

As regras são aplicadas sequencialmente frase a frase pela ordem em que se encontram no ficheiro de regras. Em cada iteração, apenas uma frase se encontra em memória, sendo esta representada por um vector de unidades desde `<s.*>` até `</s>`.

Cada regra é aplicada desde o início da frase, a cada elemento da mesma. Uma regra só é activada e consequentemente executada, se existir pelo menos uma sequência que comece no elemento que está a ser nesse momento processado e em que cada um dos seus elementos satisfaça um a um cada um dos termos do antecedente da regra. Uma regra pode ser activada mais do que uma vez para a mesma frase numa mesma iteração, se existir mais do que uma sequência na frase que satisfaça as restrições expressas no antecedente da regra, assim como pode existir mais do que uma regra a ser activada numa mesma iteração para sequências diferentes.

Sempre que uma regra é activada é executada de imediato. Isso quer dizer que uma regra pode tanto fornecer a alteração necessária para que uma outra regra possa ser por sua vez activada como impedir que uma regra posterior possa ser activada.

Assim que existir um termo de uma regra que não é verificado, o processamento avança para a próxima regra.

A execução da regra corresponde a actualizar os atributos dos elementos da sequência que fez disparar a regra, de acordo com os termos e respectivos índices especificados no consequente. Ou seja, se o consequente incluir, por exemlo, um termo indexado por 3 cujo par atributo-valor é `sema="cor"`, quer dizer que o atributo `sema` do quarto elemento da sequência que fez disparar a regra será actualizado com o valor `cor`.

Por causa de ser possível especificar no antecedente termos que emparelham com mais do que um termo no corpo, sempre que uma regra é aplicada é criado um mapa de indexação entre os índices presentes no formato interno do antecedente da regra e os índices finais que serão usados pelo consequente da regra. Caso a regra não contenha unidades multi-palavra de comprimento variável, o mapa é equivalente à identidade, ou seja, não há alteração dos índices.

Regras com unidades multi-palavra de comprimento variável As regras cujo antecedente inclui unidades multi-palavra de comprimento variável, correspondente à sequência de termos `<mwe> * </mwe>`, são aplicadas de um modo ligeiramente diferente de todas as outras. Em particular, quando o termo `*` é encontrado depois de ter sido validado o termo `<mwe>`, o programa aceita como estando conforme ao antecedente da regra todos os elementos dessa unidade multi-palavra no corpo, até encontrar o termo `</mwe>`. Além disso, o mapeamento dos índices dos restantes termos do antecedente será feito tendo em conta o comprimento da unidade multi-palavra que foi encontrada recorrendo à sequência de termos `<mwe> * </mwe>`.

Desta forma, se tivermos a regra:

```
<mwe> * </mwe> a:[pos="N"]  >>  a:[pos="ADJ"]
```

cuja representação interna é:

```
0<mwe> 1* 2</mwe> 3[pos="N"]  >>  3[pos="ADJ"]
```

e esta regra for disparada por existir uma unidade multi-palavra de comprimento 3 no corpo, em vez de ser criado um mapa de indexação `M((0,0),(1,1),(2,2),(3,3))`, será criado o mapa `M'((0,0),(1,1),(2,4),(3,5))`.

Dessa forma, quando o consequente é executado, o elemento que será actualizado na sequência não será o quarto (dado pelo índice `3` original), mas sim o sexto (dado pelo índice correspondente a `3` no mapa, ou seja, `5`).

Regras de inserção de unidades multi-palavra A inserção de novas unidades multi-palavra é expressa no consequente escrevendo a sequência de termos:

```
<mwe( $ATRIBUTO="$VALOR")*> REF1: REF2: </mwe>
```

cuja representação interna é:

```
i:<mwe> ( i:<$ATRIBUTO="$VALOR">)* j:</mwe>
```

em que i e j são os índices correspondentes às referências `REF1` e `REF2` no antecedente. Quando o consequente é executado, se não existir ainda uma unidade multi-palavra na sequência que fez disparar a regra que coincida com a unidade multi-palavra que se está a tentar inserir, o marcador inicial var ser concatenado ao ínicio do elemento i dessa sequência e o marcador final vai ser concatenado ao fim do elemento j dessa sequência; caso a unidade multi-palavra já esteja marcada no corpo, então em vez de inserir novamente os marcadores, os atributos do marcador inicial são actualizados com os valores que estiverem especificados no marcador inicial presente no consequente.

Isto quer dizer que depois de ser inserida a unidade multi-palavra o comprimento da frase representada em memória por um vector de unidades (posicionais e estruturais) continua o mesmo.

Com a representação actual da frase em vector, a inserção, idealmente em termos de representação do problema, mas talvez não em termos de eficiência computacional num programa que já de si é lento, deveria aumentar o tamanho do vector e inserir dois novos elementos. Não tentámos no entanto esta abordagem.

3.2 Requisitos técnicos

Este programa, escrito em Perl, foi testado com Perl v5.8.5 e v5.8.8, em sistema operativo Linux e Mac OS X.

Tanto os corpos como as regras estavam codificados em ISO-8859-1.

As versões dos corpos do AC/DC eram as de Setembro de 2009.

3.3 Problemas

Embora vários testes tenham sido levados a cabo, é possível que configurações inesperadas dos atributos de novos corpos produzam problemas.

O programa aplica as regras a unidades delimitadas por `<s>`. Se houver casos em que essa delimitação não ocorra ou esteja deficiente é possível que o programa perca partes do corpo.

O maior problema conhecido é o tempo de execução: o programa é extremament lento.

Não foram feitos testes para perceber o que acontece quando não é possível escrever mais (nos ficheiros de diário, ou no STDOUT). Caso os ficheiros não possam ser criados ou não existam, a execução do programa termina.

3.4 Melhorias

Uma melhoria óbvia seria aumentar o desempenho, em termos de tempo de execução, deste programa.

Também se baseia na informação rígida da ordem das colunas: para tornar o programa mais flexível poder-se-ia imaginar que a ordem fosse variada e especificada por corpo, ou por invocação.

Finalmente, a sintaxe de especificação de regras idealmente seria em tudo semelhante à do IMS-CWB, permitindo assim regras que recorressem a expressões como `within s` ou `[pos!="N.*"]*`. Isso facilitaria a escrita das regras ao linguista, assim como permitiria uma expressividade muito superior.

4 Trabalho futuro

Este programa foi desenvolvido no âmbito da melhoria do processo de revisão da anotação dos corpos do grupo de projectos relacionados com o AC/DC (que herdam ou herdaram a maior parte do processo de anotação), que já desde a altura da revisão da anotação sintáctica do COMPARA [5, 4] sabíamos que era um processo moroso, complicado, e repetitivo.

O nosso objectivo era produzir mecanismos e processos de uma anotação semântica semi-automática que produzissem bons resultados em pouco tempo, e iniciámos essa tentativa pelo campo da cor, seguido pelos da roupa, este último inspirado pelo ConDiv [13], das emoções e das partes do corpo.

Finalmente, não é de excluir a hipótese de que uma reimplementação deste programa venha a ser levada a cabo com o formalismo da CG em que o PALAVRAS foi escrito, se se considerar que a anotação semântica e a sua correcção sejam mais naturalmente levadas a cabo nesse ambiente.

Referências

[1] Eckhard Bick. *The Parsing System "Palavras": Automatic Grammatical Analysis of Portuguese in a Constraint Grammar Framework*. Tese de doutoramento. Aarhus University. Aarhus University Press. Novembro de 2000.

[2] Luís Costa, Diana Santos e Paulo Alexandre Rocha. Estudando o português tal como é usado: o serviço AC/DC. Em *The 7th Brazilian Symposium in Information and Human Language Technology (STIL 2009)*. São Carlos, Brasil. 8-11 de Setembro de 2009.

[3] Cláudia Freitas. Esqueleto: anotação das palavras do corpo humano. 15 de novembro de 2013. `http://www.linguateca.pt/acesso/Esqueleto.pdf`.

[4] Susana Inácio e Diana Santos. Syntactical Annotation of COMPARA: Workflow and First Results. Em Renata Vieira, Paulo Quaresma, Maria da Graça Volpes Nunes, Nuno J. Mamede, Cláudia Oliveira e Maria Carmelita Dias, editores, *Computational Processing of the Portuguese Language: 7th International Workshop, PROPOR 2006. Itatiaia, Brazil, May 2006 (PROPOR'2006)*. 13-17 de Maio de 2006. p. 256–259.

[5] Susana Inácio e Diana Santos. Documentação da anotação morfossintáctica da parte portuguesa do COMPARA. Dezembro de 2008. `http://www.linguateca.pt/COMPARA/DocAnotacaoPortCOMPARA.pdf`.

[6] Cristina Mota. Anotação de emoções nos corpos do AC/DC. 2013. `http://www.linguateca.pt/documentos/Mota2013.pdf`.

[7] Cristina Mota e Diana Santos. Corte e costura no AC/DC: auxiliando a melhoria da anotação nos corpos. Setembro de 2009. `http://www.linguateca.pt/acesso/corte-e-costura2009.pdf`.

[8] Diana Santos. Curso avançado de estudos contrastivos usando o COMPARA como ferramenta. 3-5 Novembro de 2008. `http://www.linguateca.pt/documentos/cursoCOMPARASantosEBRALC2008.pdf`.

[9] Diana Santos. Corpos linguísticos da Linguateca: apresentação. 3 de Julho de 2008. `http://www.linguateca.pt/documentos/SantosWorkshopTaLC2008.pdf`.

[10] Diana Santos e Eckhard Bick. Providing Internet access to Portuguese corpora: the AC/DC project. Em Maria Gavrilidou, George Carayannis, Stella Markantonatou, Stelios Piperidis e Gregory Stainhauer, editores, *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC 2000)*. Athens. 31 May-2 June de 2000. p. 205–210.

[11] Diana Santos e Luís Sarmento. O projecto AC/DC: acesso a corpora/disponibilização de corpora. Em Amália Mendes e Tiago Freitas, editores, *Actas do XVIII Encontro Nacional da Associação Portuguesa de Linguística (APL 2002)*. Porto, Portugal. 2-4 de Outubro de 2002 de 2003. p. 705–717.

[12] Diana Santos, Augusto Soares da Silva e Cristina Mota. Guarda-fatos: notas sobre a anotação do campo semântico do vestuário em português. 2009. `http://www.linguateca.pt/acesso/GuardaFatos.pdf`.

[13] Augusto Soares da Silva. O corpus CONDIV e o estudo da convergência e divergência entre variedades do português. Em Luís Costa, Diana Santos e Nuno Cardoso, editores, *Perspectivas sobre a Linguateca / Actas do encontro Linguateca : 10 anos*. Linguateca. 11 de Setembro de 2008. p. 25–28. `http://www.linguateca.pt/LivroL10/Cap04-Costaetal2008-Silva.pdf`.

[14] Rosário Silva e Diana Santos. Arco-íris: notas sobre a anotação do campo semântico da cor em português. 25 de Junho de 2009. `http://www.linguateca.pt/acesso/ArcoIris.pdf`.