O analisador sintáctico PEN

Hugo Gonçalo Oliveira

hroliv@dei.uc.pt

University of Coimbra
Faculty of Sciences and Technology
Department of Informatics Engineering

Knowledge and Intelligent Systems Laboratory
Cognitive and Media Systems Group
Centre of Informatics and Systems of the University of Coimbra

Introdução

- Analisador sintáctico, vulgo *parser*
- Implementa, em Java, o algoritmo de Earley
- Permite analisar qualquer linguagem livre de contexto
- Normalmente utilizado no processamento de linguagem natural

- Disponível sob uma licença BSD, a partir de http://code.google.com/p/pen/

Classes

- EarleyParser: classe principal, onde está implementado o algoritmo
- Grammar: gramática
  - PhraseRule: regra e seu corpo
  - TerminalRule: regra relativa a símbolo terminal
- Chart
  - ChartRow
- SemanticNode: nó da árvore
- Sentence: frase atomizada

Classes (cont.)

- Outputter: para imprimir árvores
- GrammarValidator: para validar gramáticas
  - Indica quando é encontrado o nome de uma regra não definida ou falta algum outro símbolo.

Utilização

- Utilização
  - `java EarleyParser <gramatica> <ficheiro>`
- Onde
  - `<gramatica>` é um ficheiro com regras utilizadas pelo PEN
  - `<ficheiro>` é um ficheiro com uma frase por linha

Gramáticas

- Constituídas por regras para derivar frases
- Uma regra representa-se da seguinte forma:

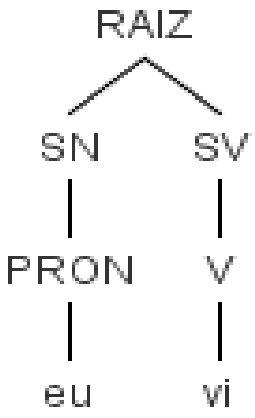
  `REGRA_UM ::= REGRA_DOIS <&> terminal`
- Os nomes de regras têm todos os caracteres maiúsculos ou "_"
- Os terminais têm de ter pelo menos um caracter minúsculo
- Todas as derivações começam pela regra `RAIZ`

Gramáticas (cont.)

- Primitivas
  - `: :=` corpo de uma regra
  - `<&>` conjunção
  - `<?>` qualquer átomo
  - `<>` átomo vazio
  - `[` comentário
  - `>` inclusão de outra gramática
  - `#` ou `<#>` (dependendo da versão) separador para o peso
  - `<@>` anotação

Derivações

- Gramática
  - RAIZ → SN SV
  - SN → PRON
  - SV → V
  - PRON → eu
  - V → vi
- Permite derivar *"eu vi"*



- No formato do PEN
  - RAIZ ::= SN <&> SV
  - SN ::= PRON
  - SV ::= V
  - PRON ::= eu
  - V ::= vi
- Com o Outputter

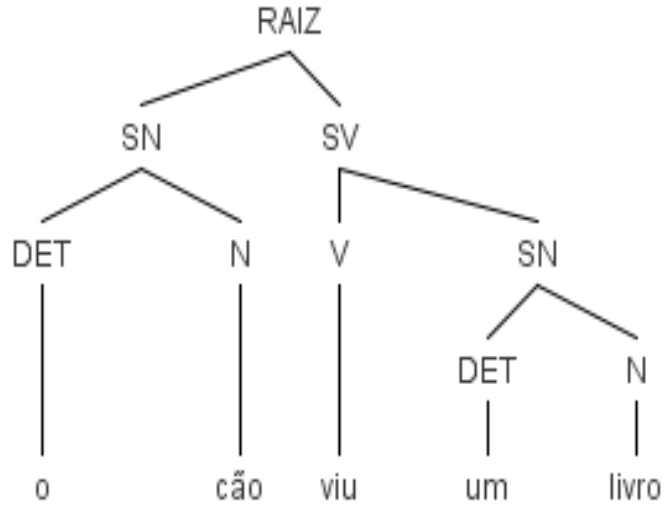
```
[RAIZ]
   [S]
      [SN]
         [PRON]
            > [eu]
      [SV]
         [V]
            > [vi]
```

Derivações

- Gramática
  - RAIZ → SN SV | SV
  - SN → DET N | PRON
  - SV → V | V SN
  - DET → o | um
  - N → cão | livro
  - PRON → eu
  - V → li | vi | leu | viu
- No formato do PEN
  - RAIZ ::= SN <&> SV
  - RAIZ ::= SV
  - SN ::= DET <&> N
  - SN ::= PRON
  - DET ::= o
  - DET ::= um
  - N ::= cão
  - N ::= livro
  - PRON ::= eu
  - V ::= li
  - V ::= vi
  - V ::= viu
  - V ::= leu

Derivações

- Permite derivar
  "o cão viu um livro"



- Com o Outputter

```
[RAIZ]
   [QUALQUERCOISA]
      > [o]
      [QUALQUERCOISA]
         > [cão]
   [S]
      [SV]
         [V]
            > [viu]
   [QUALQUERCOISA]
      > [um]
      [QUALQUERCOISA]
         > [livro]
```

Derivações

- Mas já não permite derivar:
  "eu vi um cão e um gato"
  "eu vi um cão, um gato e um rato"
  "eu li um livro completo"
  "alguém me disse que o cão leu um livro"
  "o cão viu o livro mas não o leu"
  ...
- Apenas é possível derivar frases em que todos os átomos são conhecidos!

Recursividade

- Por vezes é necessário ter regras que se referem a elas mesmas no seu corpo
- Para satisfazer essas situações, o PEN suporta recursividade

Recursividade

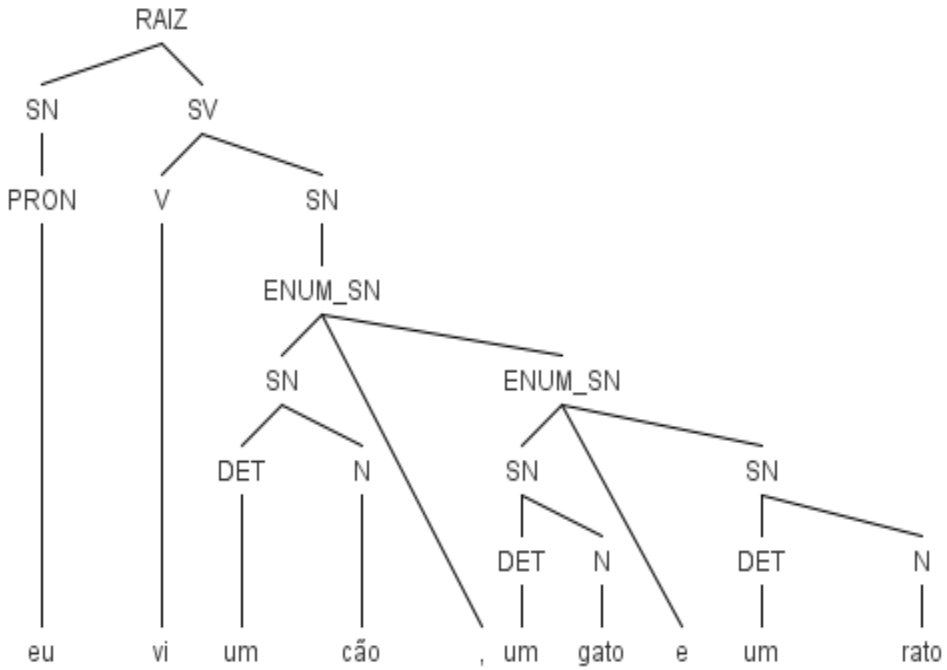
- Acrescentando regras recursivas
  - SN → ENUM_SN
  - ENUM_SN → SN CONJ SN
  - ENUM_SN → SN , ENUM_SN
  - CONJ → e | ou
- E mais alguns terminais
  - N → cão | gato | rato

- No PEN...
  - SN ::= ENUM_SN
  - ENUM_SN ::= SN <&> CONJ <&> SN
  - ENUM_SN ::= SN <&> , <&> ENUM_SN
  - CONJ ::= e
  - CONJ ::= ou
  - N ::= gato
  - N ::= rato

Recursividade

- Agora deriva

  "eu vi um cão, um gato e um rato"



- Com o Outputter

```
[RAIZ]
   [S]
      [SN]
         [PRON]
            > [eu]
      [SV]
         [V]
            > [vi]
         [SN]
            [ENUM_SN]
               [SN]
                  [DET]
                     > [um]
                  [N]
                     > [cão]
               > [,]
               [ENUM_SN]
                  [SN]
                     [DET]
                        > [um]
                     [N]
                        > [gato]
                  [CONJ]
                     > [e]
                  [SN]
                     [DET]
                        > [um]
                     [N]
                        > [rato]
```

Derivações “completas”

- Para derivar todas as frases, mesmo que não existam regras específicas para elas...
- Utilizar regras recursivas para “apanhar” qualquer coisa!
  - `QUALQUERCOISA → ? QUALQUERCOISA`
  - `QUALQUERCOISA → ?`
- No formato do PEN...
  - `QUALQUERCOISA ::= <?> <&> QUALQUERCOISA`
  - `QUALQUERCOISA ::= <?>`

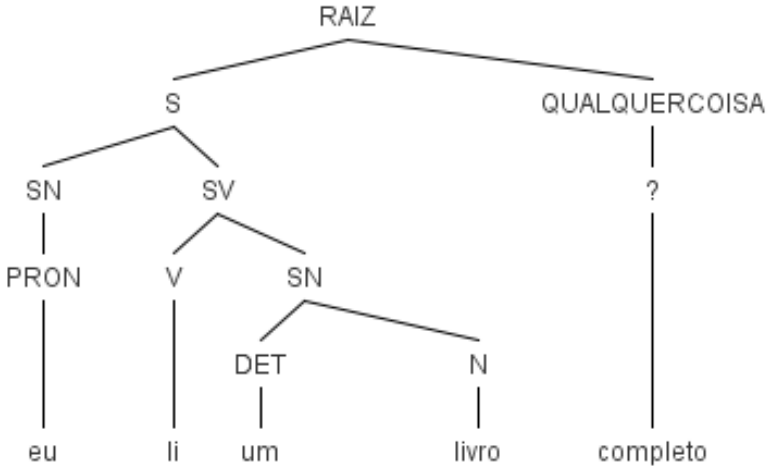
Derivações “completas”

- Acrescentando na gramática
  - RAIZ → QUALQUERCOISA S
  - RAIZ → S QUALQUERCOISA
  - RAIZ → QUALQUERCOISA S QUALQUERCOISA
  - RAIZ → S
  - S ::= SV
  - S ::= SN SV
- No formato do PEN
  - RAIZ ::= QUALQUERCOISA <&> S
  - RAIZ ::= S <&> QUALQUERCOISA
  - RAIZ ::= QUALQUERCOISA <&> S <&> QUALQUERCOISA
  - RAIZ ::= S
  - S ::= SV
  - S ::= SN <&> SV
- Apagar
  - RAIZ ::= SV
  - RAIZ ::= SN <&> SV

Derivações “completas”

- Desta forma é possível derivar

  “eu li um livro completo”



- E também

  “alguém me disse que o cão leu um livro”

  ...

- Com o Outputter...

```
[RAIZ]
   [S]
      [SN]
         [PRON]
            > [eu]
      [SV]
         [V]
            > [li]
         [SN]
            [DET]
               > [um]
            [N]
               > [livro]
   [QUALQUERCOISA]
      > [completo]
```

Inclusão de gramáticas

- Em regras utilizadas por várias gramáticas
- Para não ter de repetir a mesma regra em vários ficheiros
- Incluir gramáticas umas nas outras
  - Um ficheiro (`qqcoisa.txt`) com as regras QUALQUERCOISA pode ser incluído noutras gramáticas com:
    - `> qqcoisa.txt`
- A gramática funcionará como se tivesse todas as suas regras e ainda as incluídas

Pesos

- É ainda possível definir um peso, inteiro, para cada regra
  - `10 # RAIZ ::= S`
  - `N ::= cão`
  - `-1 # QUALQUERCOISA ::= <?> <&> QUALQUERCOISA`
- Ou na outra versão
  - `RAIZ ::= S <#> 10`
  - `N ::= cão`
  - `QUALQUERCOISA ::= <?> <&> QUALQUERCOISA <#> -1`
- O peso por omissão é 0.

Derivações “múltiplas”

- Dependendo da gramática, é possível obter mais de uma derivação para a mesma frase.
- Como no exemplo acima, para frases como “*eu vi um cão*”

```
[RAIZ]
   [QUALQUERCOISA]
      > [eu]
   [S]
      [SV]
         [V]
            > [vi]
   [QUALQUERCOISA]
      > [um]
      [QUALQUERCOISA]
         > [cão]
```

```
[RAIZ]
   [S]
      [SN]
         [PRON]
            > [eu]
      [SV]
         [V]
            > [vi]
      [QUALQUERCOISA]
         > [um]
         [QUALQUERCOISA]
            > [cão]
```

```
[RAIZ]
   [QUALQUERCOISA]
      > [eu]
   [S]
      [SV]
         [V]
            > [vi]
         [SN]
            [DET]
               > [um]
            [N]
               > [cão]
```

```
[RAIZ]
   [S]
      [SN]
         [PRON]
            > [eu]
      [SV]
         [V]
            > [vi]
         [SN]
            [DET]
               > [um]
            [N]
               > [cão]
```

Derivações “múltiplas”

- Quando se quer apenas uma, é necessário programar uma forma de escolher a “melhor” derivação, por exemplo:
  - menos nós desconhecidos,
  - árvore com melhor pontuação,
  - ...

Programação

- Instanciar o PEN
  - `EarleyParser parser = new EarleyParser(String pathGramatica);`
- Obter as derivações para uma frase
  - `ArrayList<SemanticNode> parses = parser.parse(new Sentence(frase));`
- Cada SemanticNode é a raíz de uma derivação

Programação

- Obter o peso de um nó
  - `int weight = node.getWeight();`
- Obter o peso do nós e todos os nós abaixo
  - `int score = node.getScore();`
- Obter todos os nós de determinado tipo (e.g. `DET`), abaixo de um determinado nó
  - `LinkedList<SemanticNode> nodes = node.getNodes("DET");`
- Obter todos os nós filhos
  - `LinkedList<SemanticNode> children = node.getChildren();`
- Calcular o número de filhos de um nó
  - `int childCount = node.getChildCount();`

Desafio

- Gramática para a extracção de relações de hiperonímia, utilizando *padrões de Hearst*
  - `HIPERONIMO, tais/tal como (DET* HIPONIMO ,/e/ou)+`
  - `HIPERONIMO, incluindo (DET* HIPONIMO ,/e/ou)+`
  - `HIPONIMO e outros/outras HIPERONIMO`
- Implementar um programa para extrair estas relações a partir das derivações do PEN
- Definir e implementar uma forma de seleccionar apenas uma derivação