Aula 7: Gramáticas de cláusulas definidas

- Teoria
  - Introduzir **gramáticas livres de contexto** e alguns conceitos relacionados
  - Introduzir **gramáticas de cláusulas definidas**, o jeito Prolog de trabalhar com gramáticas livres de contexto (e outras gramáticas também).

Gramáticas livres de contexto

- Prolog oferece uma notação especial para a definição de **gramáticas**, chamada DCGs ou gramáticas de cláusulas definidas.
- Assim o que é uma gramática?
- Responderemos a esta questão por meio da discussão de **gramáticas livres de contexto.**
- GLCs são um mecanismo muito poderoso e podem tratar muitos aspectos sintáticos das **línguas naturais** (tais como o português ou o inglês)

Exemplo de uma GLC

s → sn sv

sn → det n

sv → v sn

sv → v

det → *o*

det → *a*

n → *homem*

n → *mulher*

n → *bola*

v → *chuta*

Ingredientes de uma gramática

- O símbolo → é usado para definir as regras
- Os símbolos **s**, **sn**, **sv**, **det**, **n**, **v** são denominados símbolos não-terminais
- Os símbolos em itálico são os símbolos terminais:
  o, a, homem, mulher, bola, chuta

s → sn sv
sn → det n
sv → v sn
sv → v
det → *o*
det → *a*
n → *homem*
n → *mulher*
n → *bola*
v → *atira*

Um pouquinho de linguística

- Os símbolos não-terminais nesta gramática possuem um significado tradicional em linguística:
  - **sn**: sintagma nominal
  - **sv**: sintagma verbal
  - **det**: determinante
  - **n**: nome
  - **v**: verbo
  - **s**: sentença

Mais linguística

- Em uma gramática linguística, os símbolos não-terminais em geral correspondem a **categorias gramaticais**
- Em uma gramática linguística, os símbolos terminais são chamados de **itens lexicais**, ou simplesmente palavras (um cientista da computação os chamaria de **alfabeto**)

Regras livres de contexto

- A gramática contém dez regras livres de contexto
- Uma regra livre de contexto consiste de:
  - Um único símbolo não-terminal
  - seguido por →
  - seguido por uma sequência finita de símbolos terminais ou não-terminais

s → sn sv
sn → det n
sv → v sn
sv → v
det → *o*
det → *a*
n → *homem*
n → *mulher*
n → *bola*
v → *chuta*

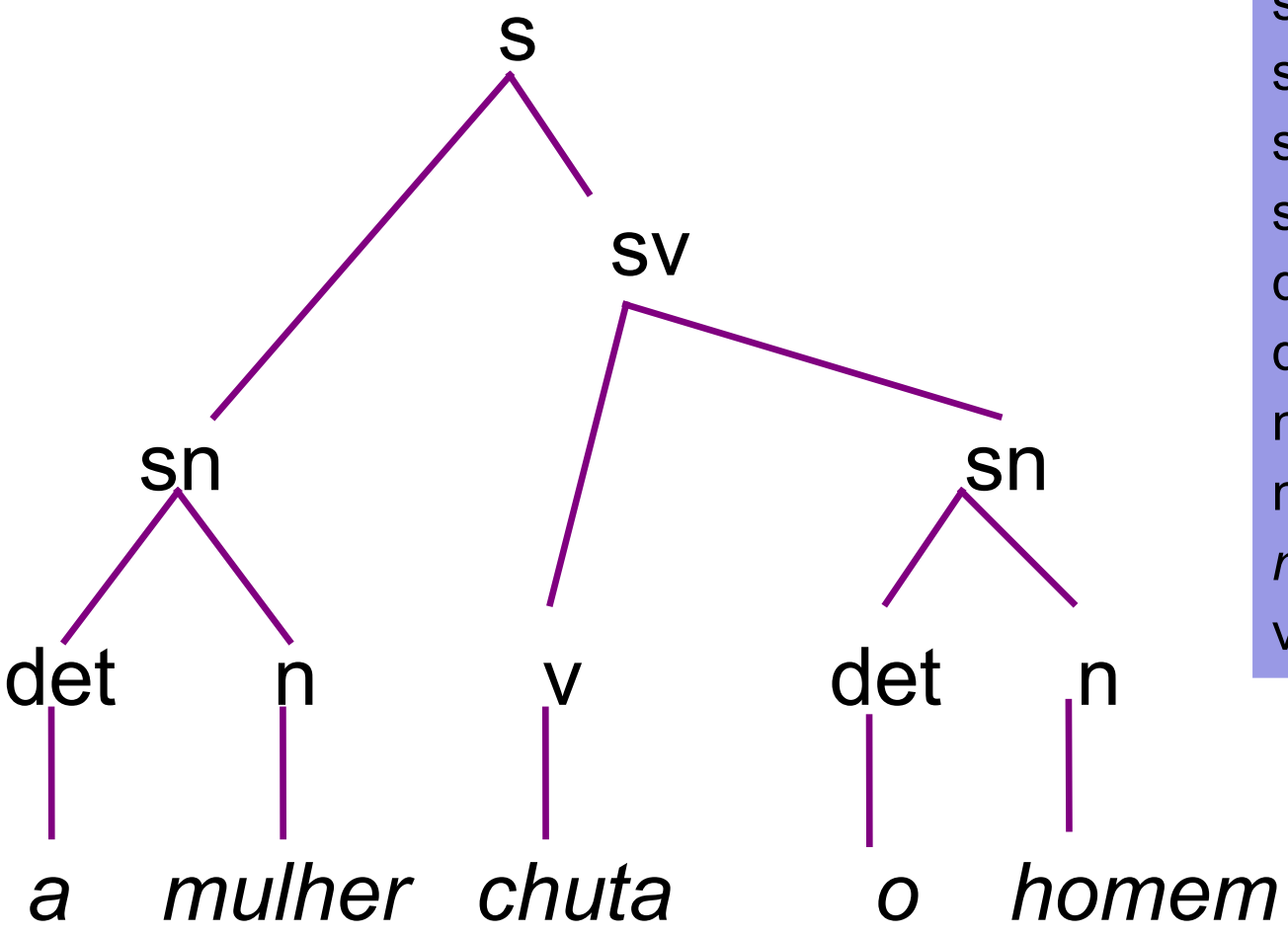
Cobertura gramatical

- Considere a frase seguinte:

  a mulher chuta o homem

- Esta frase é gramatical de acordo com nossa gramática?
- Se for, qual estrutura sintática ela exibe?

Estrutura sintática



s → sn sv
sn → det n
sv → v sn
sv → v
det → *o*
det → *a*
n → *homem*
n → *mulher*
n → *bola*
v → *chuta*

Árvores sintáticas

- Árvores que representam a estrutura sintática de uma frase, representadas aqui por strings de caracteres, são chamadas de **árvores sintáticas**

- Árvores sintáticas são importantes, pois nos dão:
  - informações sobre a string
  - informações sobre a estrutura

Strings gramaticais

- Se dadas uma string de palavras e uma gramática, podemos contruir uma árvore sintática, então dizemos que a string é **gramatical** (em relação à gramática dada)
  - Ex: *o homem chuta* é gramatical
- Se não pudermos construir uma árvore sintática, a string dada é **agramatical** (em relação à gramática dada)
  - Ex: *a chuta mulher* é agramatical

Linguagem gerada

- A **linguagem gerada por uma gramática** consiste de todas as strings que a gramática classifica como gramatical.

Por exemplo:

a mulher chuta o homem

o homem chuta

pertencem à linguagem gerada por nossa pequena gramática.

Reconhecedor

- Um **reconhecedor** livre de contexto é um programa que sempre responde corretamente se uma string pertence ou não à linguagem gerada por uma gramática livre de contexto.
- Em outras palavras, um **reconhecedor** é um programa que corretamente classifica as strings como gramaticais ou agramaticais.

Informação sobre a estrutura

- Mas, tanto na linguística quanto na ciência da computação, nós não estamos meramente interessados se uma string é gramatical ou não.
- Também queremos saber *porquê* ela é gramatical: queremos saber qual é sua estrutura.
- A árvore sintática nos dá esta estrutura.

Analisador sintático

- Um analisador sintático livre de contexto (**parser**) corretamente decide se uma string pertence à linguagem gerada por uma gramática livre de contexto.
- E ele também nos diz qual é a estrutura
- Resumindo:
  - Um reconhecedor somente diz *sim* ou *não*
  - Um analisador sintático nos dá uma árvore sintática.

Linguagem livre de contexto

- Já sabemos o que é uma gramática livre de contexto, mas o que é uma linguagem livre de contexto?
- Simples: uma linguagem livre de contexto é uma linguagem que pode ser gerada por uma gramática livre de contexto.
- Algumas línguas humanas são livres de contexto, outras não.
  - Inglês e italiano são provavelmente livres de contexto
  - Holandês e o Alemão suíço não são livres de contexto

Teoria vs. Prática

- Até agora a teoria, mas como vamos trabalhar com gramáticas livres de contexto em Prolog?
- Assuma que seja dada uma gramática livre de contexto.
  - Como poderemos escrever um reconhecedor para ela?
  - Como poderemos escrever um analisador para ela?
- Nesta aula, veremos como definir um reconhecedor.

Reconhecimento de GLC em Prolog

- Usaremos listas para representar strings
  [a,mulher,chuta,o,homem]
- Podemos pensar na regra **s → sn sv** como a concatenação de uma lista **sn** com uma lista **sv** resultando em uma lista **s**
- Sabemos como concatenar listas em Prolog: basta usar append/3
- Então, vamos transformar esta ideia em código Prolog

Reconhecimento de GLC usando append/3

```
s(C):- sn(A), sv(B), append(A,B,C).
sn(C):- det(A), n(B), append(A,B,C).
sv(C):- v(A), sn(B), append(A,B,C).
sv(C):- v(C).
det([o]).          det([a]).
n([homem]).   n([mulher]).    n([bola]).
v([chuta]).
```

Reconhecimento de GLC usando append/3

```
s(C):- sn(A), sv(B), append(A,B,C).
sn(C):- det(A), n(B), append(A,B,C).
sv(C):- v(A), sn(B), append(A,B,C).
sv(C):- v(C).
det([o]).          det([a]).
n([homem]).   n([mulher]).   n([bola]).
v([chuta]).
```

```
?- s([a,mulher,chuta,o,homem]).
true.
?-
```

Reconhecimento de GLC usando append/3

```
s(C):- sn(A), sv(B), append(A,B,C).
sn(C):- det(A), n(B), append(A,B,C).
sv(C):- v(A), sn(B), append(A,B,C).
sv(C):- v(C).
det([o]).          det([a]).
n([homem]).   n([mulher]).   n([bola]).
v([chuta]).
```

```
?- s(S).
S = [o,homem,chuta,o,homem];
S = [o,homem,chuta,o,mulher];
S = [o,mulher,chuta,o,homem]
…
```

Reconhecimento de GLC usando append/3

```
s(C):- sn(A), sv(B), append(A,B,C).
sn(C):- det(A), n(B), append(A,B,C).
sv(C):- v(A), sn(B), append(A,B,C).
sv(C):- v(C).
det([o]).          det([a]).
n([homem]).   n([mulher]).   n([bola]).
v([chuta]).
```

```
?- sn([a,mulher]).
True .
?- sn(X).
X = [o,homem];
X = [o,mulher]
```

Problemas com este reconhecedor

- Ele não utiliza a string de entrada para guiar a busca.
- Metas tais como sn(A) e sv(B) são chamadas com variáveis não instanciadas.
- Mover as metas append/3 para a frente não resolverá o problema, pois existirão um monte de chamadas para append/3 com variáveis não instanciadas.

Listas de diferenças

- Uma implementação mais eficiente pode ser obtida com o uso das **listas de diferenças**
- Esta é uma técnica sofisticada do Prolog para representar e trabalhar com listas
- Exemplos:

[a,b,c]-[ ]	é a lista [a,b,c]
[a,b,c,d]-[d]	é a lista [a,b,c]
[a,b,c|T]-T	é a lista [a,b,c]
X-X	é a lista vazia [ ]

Reconhecimento de GLC usando listas de diferenças

```
s(A-C):- sn(A-B), sv(B-C).
sn(A-C):- det(A-B), n(B-C).
sv(A-C):- v(A-B), sn(B-C).
sv(A-C):- v(A-C).
det([o|P]-P).          det([a|P]-P).
n([homem|P]-P).    n([mulher|P]-P).      n([bola|P]-P).
v([chuta|P]-P).
```

Reconhecimento de GLC usando listas de diferenças

```
s(A-C):- sn(A-B), sv(B-C).
sn(A-C):- det(A-B), n(B-C).
sv(A-C):- v(A-B), sn(B-C).
sv(A-C):- v(A-C).
det([o|P]-P).          det([a|P]-P).
n([homem|P]-P).    n([mulher|P]-P).      n([bola|P]-P).
v([chuta|P]-P).
```

```
?- s([o,homem,chuta,o,homem]-[ ]).
true .
?-
```

Reconhecimento de GLC usando listas de diferenças

```
s(A-C):- sn(A-B), sv(B-C).
sn(A-C):- det(A-B), n(B-C).
sv(A-C):- v(A-B), sn(B-C).
sv(A-C):- v(A-C).
det([o|P]-P).          det([a|P]-P).
n([homem|P]-P).    n([mulher|P]-P).      n([bola|P]-P).
v([chuta|P]-P).
```

```
?- s(X-[ ]).
S = [o,homem,chuta,o,homem];
S = [o,homem,chuta,o,mulher];
….
```

Resumo até agora

- O reconhecedor usando listas de diferenças é muito mais eficiente que aquele usando append/3
- No entanto, ele não é tão fácil de compreender e é um problema ter que gerenciar todas as variáveis das listas de diferenças.
- Seria bom ter um reconhecedor tão simples quanto o primeiro e tão eficiente quanto o segundo.
- Isto é possível: use DCGs

Gramáticas de cláusulas definidas

- O que é uma DCG?
- Ela é uma notação amigável para se escrever gramáticas, que esconde as variáveis subjacentes das listas de diferenças.
- Vamos ver três exemplos

DCGs: primeiro exemplo

```
s --> sn, sv.
sn --> det, n.
sv --> v, sn.
sv --> v.
det --> [o].          det --> [a].
n --> [homem].        n --> [mulher].      n --> [bola].
v --> [chuta].
```

DCGs: primeiro exemplo

```
s --> sn, sv.
sn --> det, n.
sv --> v, sn.
sv --> v.
det --> [o].          det --> [a].
n --> [homem].        n --> [mulher].      n --> [bola].
v --> [chuta].
```

```
?- s([o,homem,chuta,a,mulher],[ ]).
true .
?-
```

DCGs: primeiro exemplo

```
s --> sn, sv.
sn --> det, n.
sv --> v, sn.
sv --> v.
det --> [o].          det --> [a].
n --> [homem].        n --> [mulher].      n --> [bola].
v --> [chuta].
```

```
?- s(X,[ ]).
S = [o,homem,chuta,o,homem];
S = [o,homem,chuta,o,mulher];
….
```

DCGs: segundo exemplo

```
s --> s, conj, s.          s --> sn, sv.
sn --> det, n.             sv --> v, sn.          sv --> v.

det --> [o].               det --> [a].
n --> [homem].             n --> [mulher].        n --> [bola].
v --> [chuta].
conj --> [e].              conj --> [ou].         conj --> [mas].
```

- Nós adicionamos algumas regras recursivas à gramática…
- Quantas e quais são as sentenças que esta gramática gera?
- O que o Prolog faz com esta DCG?

DCG sem regras recursivas à esquerda

```
s --> s_simples.
s --> s_simples, conj, s.
s_simples --> sn, sv.
sn --> det, n.
sv --> v, sn.
sv --> v.

det --> [o].          det --> [a].
n --> [homem].        n --> [mulher].      n --> [bola].
v --> [chuta].
conj --> [e].         conj --> [ou].       conj --> [mas].
```

DCG não é mágica!

- Moral da história: DCG é uma notação amigável, mas não se pode escrever gramáticas livres de contexto arbitrárias como uma DCG e executá-las sem problemas.
- DCGs são formadas por regras Prolog comuns, mas disfarçadas
- Assim, **evite recursão à esquerda**!

DCGs: terceiro exemplo

- Nós definiremos uma DCG para uma linguagem formal
- Uma linguagem formal é simplesmente um conjunto de strings
  - Linguagens formais são objetos que cientistas da computação e matemáticos definem e estudam
  - Línguas naturais são linguagens que os seres humanos utilizam para se comunicar.
- Definiremos a linguagem $a^n b^n$

DCGs: terceiro exemplo

- Definiremos a linguagem formal $a^n b^n$

```
s --> [].
s --> e,s,d.
e --> [a].
d --> [b].
```

```
?- s([a,a,a,b,b,b],[ ]).
true
?- s([a,a,a,a,b,b,b],[ ]).
false
```

DCGs: terceiro exemplo

- Definiremos a linguagem formal $a^n b^n$

```
s --> [].
s --> e,s,d.
e --> [a].
d --> [b].
```

```
?- s(X,[ ]).
X = [ ];
X = [a,b];
X = [a,a,b,b];
X = [a,a,a,b,b,b]
….
```

Resumo desta aula

- Nós explicamos o que são gramáticas e gramáticas livres de contexto.
- Introduzimos a técnica Prolog de utilizar listas de diferenças.
- Mostramos que listas de diferenças podem ser usadas para descrever gramáticas.
- Gramática com Cláusulas Definidas (DCG) é somente uma notação Prolog mais amigável para a programação com listas de diferenças.

Próxima aula

- Mais Gramáticas com Cláusulas Definidas
  - Examinaremos duas capacidades importantes oferecidas pela notação DCG:
    - Argumentos extras
    - Testes extras
  - Discutiremos o status e as limitações das gramáticas com cláusulas definidas.