Aula 8: Mais DCGs

- Teoria
  - Examinar duas características importantes oferecidas pela notação da DCG:
    - Argumentos extras
    - Testes extras
  - Discutir o status e as limitações das DCGs

Argumentos extras

- Na aula anterior introduzimos a notação DCG básica.
- Mas as DCGs oferecem mais do que foi visto até agora
  - DCGs permitem a especificação de **argumentos extras**
  - Estes argumentos extras podem ser usados para muitos propósitos

Estendendo a gramática

- Ao lado está a gramática simples da aula anterior
- Suponha que também queiramos lidar com sentenças contendo Pronomes, tais como

  ela bate nele

  e

  ele bate nela.
- O quê precisamos fazer?

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

Estendendo a gramática

- Adicionar regras para os pronomes;
- Adicionar uma regra que diga que os sintagmas nominais podem ser pronomes.

- Esta nova DCG é boa?
- Qual é o problema?

```
s --> sn, sv.
sn --> det, n.
sn --> pro.
sv --> v, sn.
sv --> v.
det --> [o].
det --> [a].
n --> [mulher].
n --> [homem].
v --> [bate].
pro --> [ele].
pro --> [ela].
pro --> [nele].
pro --> [nela].
```

Alguns exemplos de cadeias gramaticais aceitas por esta DCG

```
?- s([ela,bate,nele],[ ]).
true
?- s([a,mulher,bate,nele],[ ]).
true
```

```
s --> sn, sv.
sn --> det, n.
sn --> pro.
sv --> v, sn.
sv --> v.
det --> [o].
det --> [a].
n --> [mulher].
n --> [homem].
v --> [bate].
pro --> [ele].
pro --> [ela].
pro --> [nele].
pro --> [nela].
```

Alguns exemplos de cadeias agramaticais aceitas por esta DCG

```
?- s([a,mulher,bate,ele],[ ]).
true
?- s([nela,bate,a,homem],[ ]).
true
?- s([nela,bate,nela],[ ]).
true
```

```
s --> sn, sv.
sn --> det, n.
sn --> pro.
sv --> v, sn.
sv --> v.
det --> [o].
det --> [a].
n --> [mulher].
n --> [homem].
v --> [bate].
pro --> [ele].
pro --> [ela].
pro --> [nele].
pro --> [nela].
```

O que está errado?

- A DCG ignora alguns fatos básicos sobre o português
  - *ele* e *ela* são pronomes pessoais do caso reto e não podem ser utilizados na posição de objetos diretos
  - *nele* e *nela* são contrações de preposição e pronome pessoal reto e não podem ser usados na posição de sujeito.
- É óbvio o que temos que fazer: estender a DCG com informações sobre sujeito e objeto.
- Como fazer isto?

Um jeito ingênuo…

```
s --> sn_sujeito, sv.
sn_sujeito --> det, n.
sn_sujeito --> pro_sujeito.
sv --> v, sn_objeto.
sv --> v.
det --> [o].
det --> [a].
n --> [mulher].
n --> [homem].
v --> [bate].
pro_sujeito --> [ele].
pro_sujeito --> [ela].
pro_objeto --> [nele].
pro_objeto --> [nela].
```

```
sn_objeto --> det, n.
sn_objeto --> pro_objeto.
```

Um jeito melhor usando argumentos extras

```
s --> sn(sujeito), sv.
sn(_) --> det, n.
sn(X) --> pro(X).
sv --> v, sn(objeto).
sv --> v.
det --> [o].
det --> [a].
n --> [mulher].
n --> [homem].
v --> [bate].
pro(sujeito) --> [ele].
pro(sujeito) --> [ela].
pro(objeto) --> [nele].
pro(objeto) --> [nela].
```

Isto funciona…

```
s --> sn(sujeito), sv.
sn(_) --> det, n.
sn(X) --> pro(X).
sv --> v, sn(objeto).
sv --> v.
det --> [o].
det --> [a].
n --> [mulher].
n --> [homem].
v --> [bate].
pro(sujeito) --> [ele].
pro(sujeito) --> [ela].
pro(objeto) --> [nele].
pro(objeto) --> [nela].
```

```
?- s([ela,bate,nele],[ ]).
true
?- s([ela,bate,ele],[ ]).
false
?-
```

O que realmente está acontecendo?

- Lembre que a regra:

  s --> sn, sv.

  É, de fato, somente um açúcar sintático para:

  s(A,B):- sn(A,C), sv(C,B).

O que realmente está acontecendo?

- Lembre-se que a regra:

  s --> sn,sv.

  É, de fato, somente um açúcar sintático para:

  s(A,B):- sn(A,C), sv(C,B).

- A regra

  s --> sn(sujeito), sv.

  é traduzida em:

  s(A,B):- sn(sujeito,A,C), sv(C,B).

Listando sintagmas nominais

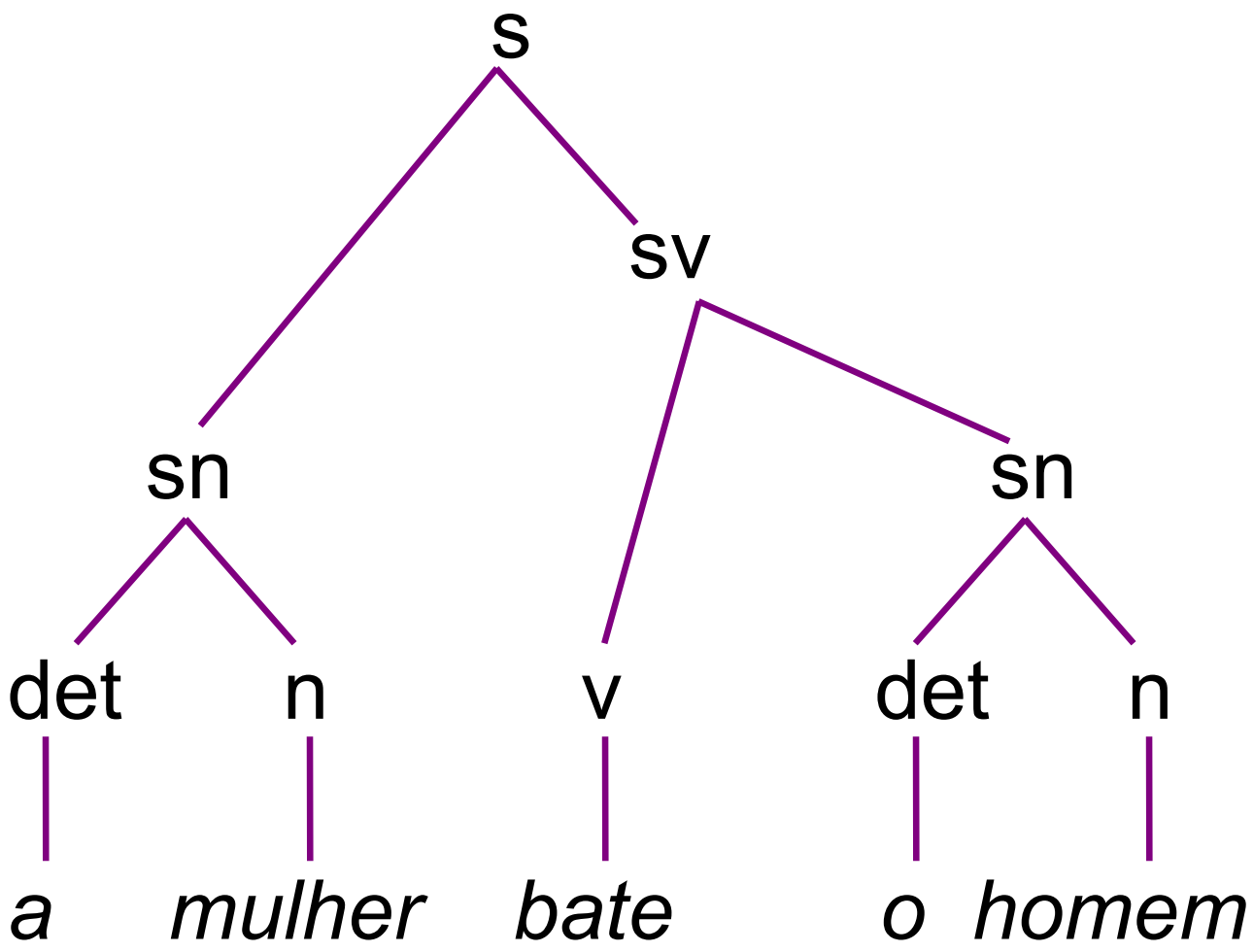
```
s --> sn(sujeito), sv.
sn(_) --> det, n.
sn(X) --> pro(X).
sv --> v, sn(objeto).
sv --> v.
det --> [o].
det --> [a].
n --> [mulher].
n --> [homem].
v --> [bate].
pro(sujeito) --> [ele].
pro(sujeito) --> [ela].
pro(objeto) --> [nele].
pro(objeto) --> [nela].
```

```
?- sn(Tipo, SN, [ ]).
SN = [o,mulher];

SN = [o,homem];

SN = [a,mulher];

SN = [a,homem];

Tipo =sujeito
SN = [ele]
```

Construindo árvores sintáticas

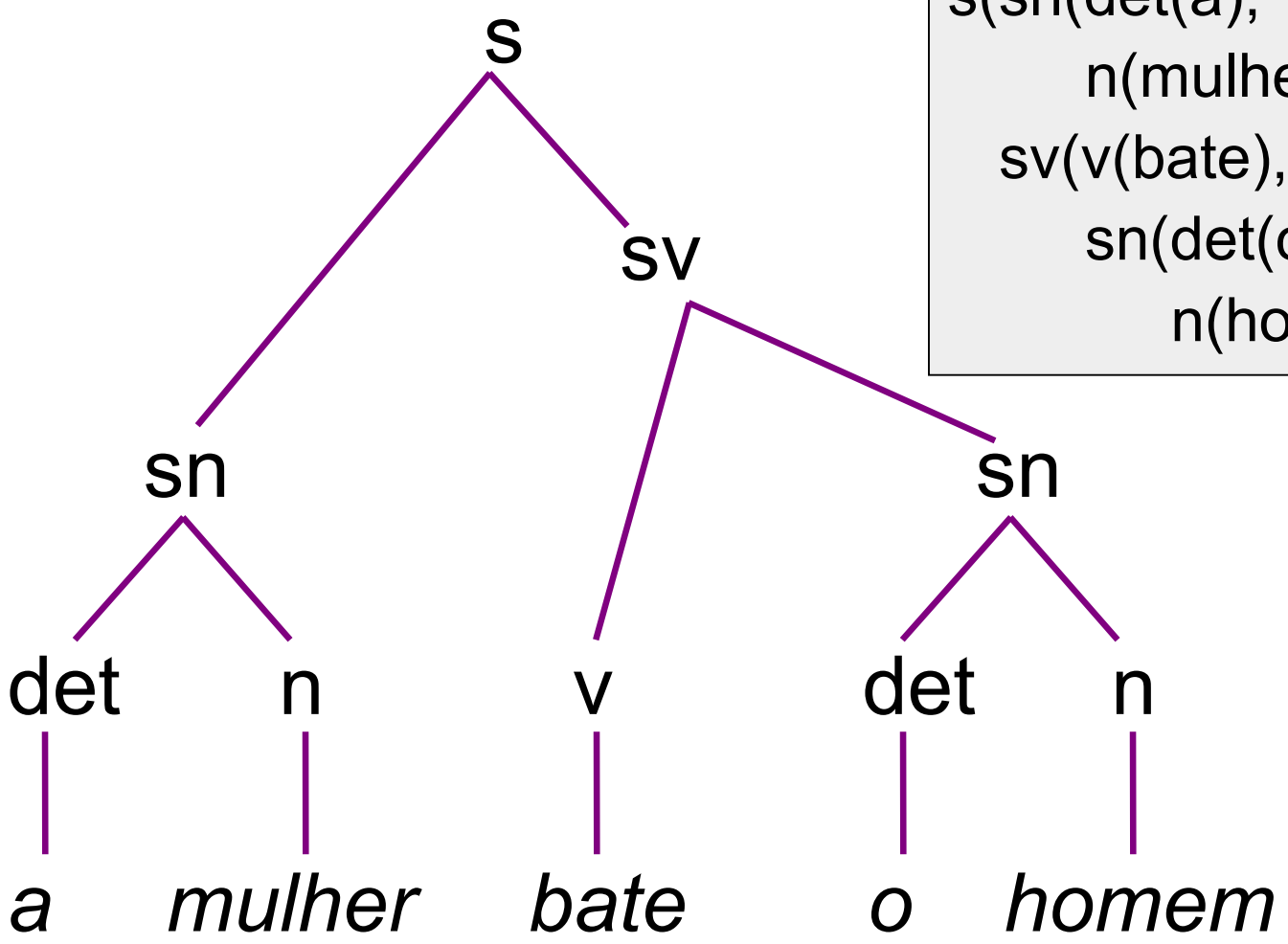
- Os programas discutidos até agora conseguem reconhecer as estruturas gramaticais das sentenças
- Mas gostaríamos também de ter um programa que nos forneça uma análise destas estruturas.
- Em particular, gostaríamos de ver as árvores sintáticas que a gramática atribui às sentenças.

Exemplo de árvore sintática



Árvore sintática em Prolog

```
s(sn(det(a),
       n(mulher)),
  sv(v(bate),
       sn(det(o),
            n(homem)))))
```



DCG que constrói a árvore sintática

```
s --> sn(sujeito), sv.
sn(_) --> det, n.
sn(X) --> pro(X).
sv --> v, sn(objeto).
sv --> v.
det --> [o].
det --> [a].
n --> [mulher].
n --> [homem].
v --> [bate].
pro(sujeito) --> [ele].
pro(sujeito) --> [ela].
pro(objeto) --> [nele].
pro(objeto) --> [nela].
```

DCG que constrói a árvore sintática

```
s --> sn(sujeito), sv.
sn(_) --> det, n.
sn(X) --> pro(X).
sv --> v, sn(objeto).
sv --> v.
det --> [o].
det --> [a].
n --> [mulher].
n --> [homem].
v --> [bate].
pro(sujeito) --> [ele].
pro(sujeito) --> [ela].
pro(objeto) --> [nele].
pro(objeto) --> [nela].
```

```
s(s(SN,SV)) --> sn(sujeito,SN), sv(SV).
sn(_, sn(Det,N)) --> det(Det), n(N).
sn(X, sn(Pro)) --> pro(X,Pro).
sv(sv(V,SN)) --> v(V), sn(objeto,SN).
sv(sv(V)) --> v(V).
det(det(o)) --> [o].
det(det(a)) --> [a].
n(n(mulher)) --> [mulher].
n(n(homem)) --> [homem].
v(v(bate)) --> [bate].
pro(sujeito,pro(ele)) --> [ele].
pro(sujeito,pro(ela)) --> [ela].
pro(objeto,pro(nele)) --> [nele].
pro(objeto,pro(nela)) --> [nela].
```

Gerando árvores sintáticas

```
?- s(A,[ele,bate],[]).
A = s(sn(pro(ele)),
      sv(v(bate)))
true
```

```
s(s(SN,SV)) --> sn(sujeito,SN), sv(SV).
sn(_, sn(Det,N)) --> det(Det), n(N).
sn(X, sn(Pro)) --> pro(X,Pro).
sv(sv(V,SN)) --> v(V), sn(objeto,SN).
sv(sv(V)) --> v(V).
det(det(o)) --> [o].
det(det(a)) --> [a].
n(n(mulher)) --> [mulher].
n(n(homem)) --> [homem].
v(v(bate)) --> [bate].
pro(sujeito,pro(ele)) --> [ele].
pro(sujeito,pro(ela)) --> [ela].
pro(objeto,pro(nele)) --> [nele].
pro(objeto,pro(nela)) --> [nela].
```

Gerando árvores sintáticas

```
?- s(Arvore,S,[]).
```

```
s(s(SN,SV)) --> sn(sujeito,SN), sv(SV).
sn(_, sn(Det,N)) --> det(Det), n(N).
sn(X, sn(Pro)) --> pro(X,Pro).
sv(sv(V,SN)) --> v(V), sn(objeto,SN).
sv(sv(V)) --> v(V).
det(det(o)) --> [o].
det(det(a)) --> [a].
n(n(mulher)) --> [mulher].
n(n(homem)) --> [homem].
v(v(bate)) --> [bate].
pro(sujeito,pro(ele)) --> [ele].
pro(sujeito,pro(ela)) --> [ela].
pro(objeto,pro(nele)) --> [nele].
pro(objeto,pro(nela)) --> [nela].
```

Além de gramáticas livres de contexto

- Na aula anterior apresentamos as DCGs como uma ferramenta útil para se trabalhar com gramáticas livres de contexto
- Entretanto, as DCGs podem lidar com gramáticas ainda mais descritivas que as gramáticas livres de contexto
- Os argumentos extras nos dão a habilidade para tratar qualquer linguagem computável.
- Ilustraremos isto por meio da linguagem formal $a^n b^n c^n \backslash \{\varepsilon\}$

Um exemplo

- A linguagem $a^n b^n c^n \backslash \{\varepsilon\}$ consiste das cadeias tais como abc, aabbcc, aaabbbccc e assim em diante.

- Esta linguagem não é livre de contexto – é impossível escrever uma gramática livre de contexto que produza exatamente estas cadeias.
- Mas é fácil escrever uma DCG que a reconheça.

DCG para $a^n b^n c^n \backslash \{\varepsilon\}$

```
s(Conta) --> as(Conta), bs(Conta), cs(Conta).

as(0) --> [].
as(suc(Conta)) --> [a], as(Conta).

bs(0) --> [].
bs(suc(Conta)) --> [b], bs(Conta).

cs(0) --> [].
cs(suc(Conta)) --> [c], cs(Conta).
```

DCG para $a^n b^n c^n \backslash \{\varepsilon\}$

```
?- s(C,S,[]).
C = 0,
S = [] ;

C = suc(0),
S = [a, b, c] ;

C = suc(suc(0)),
S = [a, a, b, b, c, c] ;

C = suc(suc(suc(0))),
S = [a, a, a, b, b, b, c, c, c] ;
...
```

```
s(Conta) --> as(Conta),
   bs(Conta), cs(Conta).

as(0) --> [].
as(suc(Conta)) --> [a],
   as(Conta).

bs(0) --> [].
bs(suc(Conta)) --> [b],
   bs(Conta).

cs(0) --> [].
cs(suc(Conta)) --> [c],
   cs(Conta).
```

Metas extras

- Qualquer regra da DCG é, de fato, uma estrutura sintática para uma regra Prolog comum.
- Desta forma, não é tão surpreendente que possamos chamar qualquer predicado Prolog no lado direito de uma regra da DCG.
- Isto é feito colocando o predicado entre chaves.

Exemplo: DCG para $a^n b^n c^n \setminus \{\varepsilon\}$

```
s(Conta) --> as(Conta), bs(Conta), cs(Conta).

as(0) --> [].
as(NovoCnt) --> [a], as(Cnt), {NovoCnt is Cnt + 1}.

bs(0) --> [].
bs(NovoCnt) --> [b], bs(Cnt), {NovoCnt is Cnt + 1}.

cs(0) --> [].
cs(NovoCnt) --> [c], cs(Cnt), {NovoCnt is Cnt + 1}.
```

Separando as regras do léxico

- Uma aplicação clássica das metas extras nas DCGs em linguística computacional é a separação das regras gramaticais do restante do léxico.
- O que isto significa?
  - Eliminação de todas as menções a palavras individuais na DCG.
  - Registro de todas as informações sobre palavras individuais em um léxico separado.

A gramática básica

```
s --> sn, sv.

sn --> det, n.

sv --> v, sn.
sv --> v.

det --> [o].
det --> [a].

n --> [mulher].
n --> [homem].
v --> [bate].
```

A gramática modular

```
s --> sn, sv.

sn --> det, n.

sv --> v, sn.
sv --> v.

det --> [o].
det --> [a].

n --> [mulher].
n --> [homem].
v --> [bate].
```

≡

```
s --> sn, sv.

sn --> det, n.

sv --> v, sn.
sv --> v.

det --> [Palavra], {lex(Palavra,det)}.

n --> [Palavra], {lex(Palavra,n)}.

v --> [Palavra], {lex(Palavra,v)}.
```

+

```
lex(o, det).
lex(a,    det).
lex(mulher,  n).
lex(homem, n).
lex(bate, v).
```

Observações finais

- As DCGs são ferramentas simples para a codificação de gramáticas livres de contexto.
- Mas, na verdade, as DCGs formam uma linguagem de programação completa e podem ser utilizadas para muitas finalidades diferentes.
- Para propósitos linguísticos, a DCG possui algumas desvantangens:
  - Regras recursivas à esquerda.
  - As DCGs são interpretadas de forma *top-down*
- As DCGs não são mais o estado-da-arte, mas ainda permanecem como uma ferramenta útil.

Próxima aula

- Um olhar mais atento aos termos
  - Introduzir o predicado identidade
  - Olhar mais de perto a estrutura de um termo
  - Introduzir os predicados Prolog pré-definidos que testam se um dado termo é de um certo tipo.
  - Mostrar como definir novos operadores em Prolog