Programação Declarativa

Paradigmas de Programação Avançados

Salvador Abreu, Universidade de Évora, 2019/20

Gramáticas Lógicas

Definite Clause Grammars (DCGs)

Formulação

Integração com Prolog

Definite Clause Grammars

As gramáticas de cláusulas definidas (a.k.a DCGs) são uma notação para exprimir regras (produções) capazes de descrever linguagens mais gerais que as livres de contexto.

Exemplo, para expressões aritméticas:

```
expr(V) --> termo(V).
expr(V) --> termo(A), “+”, termo(B), { V is A+B }.
expr(V) --> termo(A), “-”, termo(B), { V is A-B }.

termo(V) --> fator(V).
termo(V) --> fator(A), “*”, fator(B), { V is A*B }.
termo(V) --> fator(A), “/”, fator(B), { V is A/B }.

fator(X) --> numero(X).
fator(X) --> “(“, expr(X), “)”.
```

Definite Clause Grammars

Elementos duma DCG

- Não-terminais (~ predicados)
- Terminais (listas de tokens, strings)
- Regras

Uma regra é composta assim:

- `CABEÇA --> CORPO.`
- Em que:
  - CABEÇA denota um símbolo não-terminal, possivelmente com argumentos
  - CORPO é uma sequência de símbolos (terminais ou não)

Exemplo:

```
numero --> digito, resto.

resto --> digito, resto.
resto --> [].

digito --> “0”.
digito --> “1”.
...
digito --> “9”.
```

DCGs: como funciona

Um símbolo numa DCG é expandido para um termo Prolog, acrescentando 2 argumentos **no final**, entendidos como listas, que denotam respetivamente:

- A lista de símbolos (terminais) a ser analisada (o primeiro)
- A lista de símbolos (terminais) que ficaram “fora” do presente símbolo não-terminal

Chamaremos a esses argumentos: **SYM** e **NEXT**.

Também podemos ter **ações semânticas**, entre { } que são entendidas como “Prolog normal”.

Exemplo

```
| ?- numero("123 vezes 33", X), format("resta \"~s\"\n", [X]).
resta " vezes 33"

X = [32,118,101,122,101,115,32,51,51] ?

yes
| ?-
```

DCGs: Prolog equivalente

Uma gramática é traduzida de forma a que os símbolos sejam “ligados” por via dos SYM e NEXT: o NEXT dum símbolo é **UNIFICADO** com o SYM do símbolo seguinte (i.e. usa-se a mesma variável).

Exemplo, para o não-terminal **numero** e seguintes:

```
numero(A, B) :-
    digito(A, C),
    resto(C, B).
```

```
resto(A, B) :-
    digito(A, C),
    resto(C, B).
resto(A, A).
```

```
digito([48|A], A).
digito([49|A], A).
digito([50|A], A).
digito([51|A], A).
digito([52|A], A).
digito([53|A], A).
digito([54|A], A).
digito([55|A], A).
digito([56|A], A).
digito([57|A], A).
```

DCGs com argumentos

Se tivermos uma nova versão do símbolo **numero**, desta vez com um argumento que se pretende denote o valor:

```
numero(N) --> digito(D), resto(D, N).

resto(D, N) --> digito(E), { F is 10*D + E }, resto(F, N).
resto(X, X) --> [].

digito(0) --> "0".
digito(1) --> "1".
digito(2) --> "2".
digito(3) --> "3".
digito(4) --> "4".
digito(5) --> "5".
digito(6) --> "6".
digito(7) --> "7".
digito(8) --> "8".
digito(9) --> "9".
```

DCGs com argumentos

Podemos usar isto assim, por exemplo:

```
| ?- numero(X, "123 mais", Y).

X = 123
Y = [32,109,97,105,115] ?

yes
| ?-
```

DCGs com argumentos: programa Prolog

```
numero(A, B) :-
    digito(A, C),
    resto(C, B).

resto(A, B) :-
    digito(A, C),
    resto(C, B).
resto(A, A).
```

```
digito([48|A], A).
digito([49|A], A).
digito([50|A], A).
digito([51|A], A).
digito([52|A], A).
digito([53|A], A).
digito([54|A], A).
digito([55|A], A).
digito([56|A], A).
digito([57|A], A).
```

DCGs para construir APTs

Em vez de *calcular* valores, podemos *construir* termos. Por exemplo:

```
expr(V) --> termo(V).
expr(mais(A,B)) --> termo(A), "+", termo(B).
expr(menos(A,B)) --> termo(A), "-", termo(B).

termo(V) --> fator(V).
termo(vezes(A,B)) --> fator(A), "*", fator(B).
termo(dividir(A,B)) --> fator(A), "/", fator(B).

fator(X) --> numero(X).
fator(X) --> "(", expr(X), ")".
```

```
| ?- expr(X, "(12+5)*3", []).

X = vezes(mais(12,5),3) ?

yes
| ?-
```

DCGs para construir APTs

Um exemplo mais complicado:

```
| ?- expr(X, "(12+5)/3+4*(5+2*6)", []).

X = mais(dividir(mais(12,5),3),vezes(4,mais(5,vezes(2,6)))) ?

(1 ms) yes
| ?-
```

DCGs com listas de **tokens**

Nada obriga que as listas SYM e NEXT sejam de inteiros (i.e. strings); podem ser listas dos termos que quisermos. Por exemplo:

```
% -- Pascal statements --

stmt --> [begin], slist, [end].
stmt --> [if], expr, [then], stmt, [else], stmt.
stmt --> [if], expr, [then], stmt.
stmt --> [while], expr, [do], stmt.
stmt --> expr, [assign], expr.    % :=

slist --> stmt, srest.
slist --> [].

srest --> [ssep], stmt, srest.    % ;
srest --> [].
```

DCGs usadas de forma **generativa**

Uma DCG pode ser usada para produzir uma lista, a partir dos seus argumentos.
Exemplo:

```
stack(mais(A,B)) --> stack(A), stack(B), [soma].
stack(menos(A,B)) --> stack(A), stack(B), [subtrai].
stack(vezes(A,B)) --> stack(A), stack(B), [multiplica].
stack(dividir(A,B)) --> stack(A), stack(B), [divide].
stack(X) --> [empilha(X)].
```

```
| ?- expr(X, "(12+5)/3+4*(5+2*6)", []), stack(X, S, []).

S = [empilha(12),empilha(5),soma,empilha(3),divide,empilha(4),
     empilha(5),empilha(2),empilha(6),multiplica,soma,
     multiplica,soma]
X = mais(dividir(mais(12,5),3),vezes(4,mais(5,vezes(2,6)))) ?

(1 ms) yes
| ?-
```

DCG generativa: código Prolog

```
| ?- listing(stack).

% file: /Volumes/gd-ue/My
Drive/aulas/2019-20/pd/code/dc
g_stack.pl

stack(mais(A, B), C, D) :-
    stack(A, C, E),
    stack(B, E, F),
    F = [soma|D].
stack(menos(A, B), C, D) :-
    stack(A, C, E),
    stack(B, E, F),
    F = [subtrai|D].
stack(vezes(A, B), C, D) :-
    stack(A, C, E),
    stack(B, E, F),
    F = [multiplica|D].
stack(dividir(A, B), C, D) :-
    stack(A, C, E),
    stack(B, E, F),
    F = [divide|D].
stack(A, [empilha(A)|B], B).

yes
| ?-
```