Linguagens e Ambientes de Programação (Aula Teórica 17)

LEI - Licenciatura em Engenharia Informática

João Costa Seco (joao.seco@fct.unl.pt)

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Agenda

- Terceiro Trabalho - Esclarecimentos
- Interpretadores
- Sintaxe concreta, sintaxe abstrata, linguagens estruturadas

Terceiro trabalho - Datalog

- Datalog é uma linguagem **declarativa** que representa programas em lógica.
- Um programa Datalog é composto por um conjunto de regras na forma conjuntiva da forma:

```
H :- B1, B2, … Bn
```

$$B_1 \wedge B_2 \wedge \ldots \wedge B_n \implies H$$

- O predicado `H` é válido (provado) se os predicados `B1` a `Bn` também forem.
- Um facto é uma regra cujo corpo é vazio.

```
Person(John) :- .
```

```
A(0) :- .
```

$$\texttt{true} \implies H$$

- Os predicados têm parâmetros e um domínio (untyped) composto por valores.

```
Parent(John,Mary), Age(John,30), Person(X)
```

Terceiro trabalho - Datalog

- O programa

```
Parent(John, Mary) :- .
Parent(Mary, Ann) :- .
Ancestor(X,Y) :- Parent(X, Y).
Ancestor(X,Y) :- Parent(X, Z), Ancestor(Z, Y).
```

- Permite deduzir os seguintes factos

```
Parent(John, Mary)
Parent(Mary, Ann)
Ancestor(John, Mary)
Ancestor(Mary, Ann)
Ancestor(John, Ann)
```

Análise de programas (sketch)

```
i := 10
while i ≠ 0 {
  i := i - 1;
};
return i;
```

```
Block(0, “i := 10”) :-.
Block(1, “i ≠ 0”) :-.
Block(2, “i := i - 1”) :-.
Block(3, “return i”) :-.
Pred(0,1) :-
PredT(1,2) :-
PredF(1,3) :-
Pred(2,1) :-
Pred(X,Y) :- PredT(X,Y)
Pred(X,Y) :- PredF(X,Y)
NZero_i(X) :- Block(X, “i:=10”)
NZero_i(X) :- PredT(Y,X), Block(Y, “i≠0”), Not_i(X)
Zero_i(X) :- PredF(Y,X), Block(Y, “i≠0”), Not_i(X)
NZero_i(X) :- Pred(X,Z), NZero_i(Z), Not_i(X)
NZero_i(X) :- Pred(X,Z), NZero_i(Z), Not_i(X)

?- Zero_i(3)
```

Terceiro trabalho - Datalog

- Um algoritmo Top-Down, pesquisa uma prova para uma dada query.
- Para fazer essa pesquisa, tem que usar backtracking (ao falhar procura um caminho alternativo), ou para ter os resultados todos possíveis, tem que implementar backtracking mesmo quando não falha.
- Um algoritmo bottom up, deriva todos os resultados possíveis com um algoritmo de ponto fixo.
- Termina sempre porque
  - todos os parâmetros são valores simples ou variáveis (não há functores nos parâmetros)
  - não tem negação

Terceiro trabalho - Datalog

```
A(0,1) :- .
A(1,2) :- .
A(X,X) :- .
A(X,Y) :- A(X, Z), A(Z, Y).
```

- Permite deduzir os seguintes factos

```
A(0,1)
A(1,2)
A(0,2)
A(X,X)
```

- E permite responder positivamente a queries como

```
A(0,0)
A(1,1)
A(99,99)
```

Algoritmo de Ponto Fixo

- O algoritmo de (menor) ponto fixo permite implementar qualquer sistema indutivo (em tempo exponencial).

$$
\begin{array}{l}
F^0 := \emptyset \\
i := 0 \\
\text{repetir} \\
\quad F^{i+1} := \emptyset \\
\quad \text{para cada regra } H \text{ :- } B_1, B_2, \ldots, B_n \in P : \\
\quad\quad \text{se } B_1, B_2, \ldots, B_n \subseteq F^i \text{ então } F^{i+1} := F^{i+1} \cup \{H\} \\
\quad\quad i := i + 1 \\
\text{enquanto } F^{i+1} = F^i
\end{array}
$$

Algoritmo de Ponto Fixo

- O algoritmo de (menor) ponto fixo permite implementar qualquer sistema indutivo (em tempo exponencial).

$$F^0 := \emptyset$$
$$F^{i+1} = \{H \mid H \text{ :- } B_1, B_2, \ldots, B_n \in P \text{ e, tal que, } B_1, B_2, \ldots, B_n \subseteq F^i\}$$

Representação abstrata de programas (simples)

```
type term = string
type rule = term * term list
type program = rule list
type query = term

let program = [
  ("A", []);
  ("B", ["A"]);
  ("C", ["B"; "A"]);
  ("D", ["C"; "E"])
]

solve : program -> query -> bool
```

```
A :-.
B :- A.
C :- B, A.
D :- C, E.
```

Representação abstrata de programas (simples)

```
type parameter = Var of string | Value of int
type term = string * parameter list
type rule = term * term list
type program = rule list
type query = term

let program =[
  (("A", [Value 0]), [])
; (("B", [Var "X"]), [])
; (("C", [Var "Y"]), [( "A", [Var "Y"]); ( "B", [Var "Y"])])
; (("D", [Var "Y"]), [( "B", [Var "Y"])])
]

solve program ( "A", [Value 0] )
```

```
A(0) :-.
B(X) :-.
C(Y) :- A(Y), B(Y).
D(Y) :- B(Y).
```

```
A(0).
B(X).
C(0).
D(X).
```

Algoritmo de Ponto Fixo

- O algoritmo de (menor) ponto fixo permite implementar qualquer sistema indutivo (em tempo exponencial).

$$
\begin{array}{l}
F^0 := \emptyset \\
i := 0 \\
\text{repetir} \\
\quad F^{i+1} := \emptyset \\
\quad \text{para cada regra } H \text{ :- } B_1, B_2, \ldots, B_n \in P : \\
\quad\quad \text{para cada substituição } \theta \text{ unificando } B_1, B_2, \ldots, B_n \text{ em } F^i : \\
\quad\quad\quad F^{i+1} := F^{i+1} \cup \{H\theta\} \\
\quad\quad i := i + 1 \\
\text{enquanto } F^{i+1} = F^i
\end{array}
$$

Algoritmo de Ponto Fixo

- O algoritmo de (menor) ponto fixo permite implementar qualquer sistema indutivo (em tempo exponencial).

$$F^0 := \emptyset$$
$$i := 0$$

$$F^0 := \emptyset$$
$$F^{i+1} = \{H\theta \mid H \text{ :- } B_1, B_2, \ldots, B_n \in P, \\ \theta \text{ unifica } B_1, B_2, \ldots, B_n, \text{ tal que, } B_1\theta, B_2\theta, \ldots, B_n\theta \subseteq F^i\}$$

$$i := i + 1$$
enquanto $F^{i+1} = F^i$

Unification and Normalization of rules

```
A(0) :- .
B(X) :- .
C(Y) :- A(Y), B(Y).
D(Y) :- B(Y).

?- C(X) (No)

A(0).
B(X).
C(0).
D(X).
```

Unification and Normalization of rules

```
A(0) :- .
B(X) :- .
C(Y) :- A(Y), B(Y).
D(Y) :- B(Y).

?- A(0) (Yes)

A(0).
B(X).
C(0).
D(X).
```

Unification and Normalization of rules

```
A(0) :- .
B(X) :- .
C(Y) :- A(Y), B(Y).
D(Y) :- B(Y).

?- B(Y) (Yes ??)

A(0).
B(X).
C(0).
D(X).
```

Unification and Normalization of rules

```
A(0) :- .
B(_0) :- .
C(_0) :- A(_0), B(_0).
D(_0) :- B(_0).

?- B(_0) (Yes)

A(0).
B(_0).
C(0).
D(_0).
```

Unification and Normalization of rules

```
A(X,X) :- .
A(0,1) :- .
A(1,2) :- .
A(X,Y) :- A(X,W), A(W,Y)

?- A(0,0) (Yes)
?- A(1,1) (Yes)
?- A(8,8) (Yes)
?- A(0,2) (Yes)
?- A(Y,Y) (Yes ??)
```

Unification and Normalization of rules

```
A(_0,_0) :- .
A(0,1) :- .
A(1,2) :- .
A(_0,_1) :- A(_0,W), A(W,_1)

?- A(0,0) (Yes)
?- A(1,1) (Yes)
?- A(8,8) (Yes)
?- A(0,2) (Yes)
?- A(Y,Y) (Yes)
```