Programação em Lógica

Prof. A. G. Silva

15 de setembro de 2016

Recomendações de estilo

- Cláusula de mesmo predicado em linhas consecutivas, separando diferentes predicados com uma ou mais linhas em branco
- Caso uma cláusula não caiba em um linha ($\sim$70 caracteres), deixa-se apenas a cabeça e o ":-" na primeira linha; nas seguintes são submetas do corpo endentadas (terminadas por vírgula, exceto a última, por ponto)
- Predicados com muitas regras podem ser quebrados em vários
- O ";" pode eventualmente ser substituído por mais de uma cláusula
- Variáveis anônimas usadas para aquelas que ocorrem apenas uma vez em uma cláusula

Cuidados ao definir um predicado

- Verificação da grafia do nome em todas as ocorrências
- Verificação do número de argumentos, certificando-se de que combina com o projeto do predicado
- Identificação de todos os operadores usados e suas precedências, associatividades e argumentos. Uso de parênteses em caso de dúvida
- Observação do escopo de cada variável, do compartilhamento de valor ao instanciar uma delas, e se todas as variáveis da cabeça de uma regra aparecem no seu corpo
- Identificação se todas as condições de parada (ou caso base de recursão) estão contempladas

Resolução comum de problemas

- Ponto final ao término de cada cláusula
- No final do arquivo, *newline* após o último ponto final
- Casamento dos parêntes e colchetes
- Grafia dos nomes de predicados pré-definidos, baseada no manual da implementação de Prolog em uso
- Ao carregar um arquivo, *warnings* do tipo "singleton variable" referem-se a variáveis que aparecem uma vez só numa regra ou fato
- Números muito grandes, como `2 ^ fat(7)`, podem ser interpretados como infinito; pode haver igualdade entre dois infinitos, mesmo se expressões não forem iguais

Alguns predicados pré-definidos

- Predicados pré-definidos importantes que não foram tratados até agora, organizados em:
  - Tipos
  - Listas
  - Conjuntos
  - Coleção de soluções
  - Outros

Tipos

- `var(X)` é satisfeito quando `X` é uma variável não instanciada
- `nonvar(X)` é satisfeito quando `X` é um termo ou uma variável instanciada. O contrário de `var(X)`
- `atom(X)` é satisfeito quando `X` é um átomo (constante não numérica)
- `number(X)` é satisfeito quando `X` é um número
- `atomic(X)` é satisfeito quando `X` é um átomo ou um número

Listas

- `last(X, L)` é satisfeito quando `X` é o último elemento da lista `L`
- `reverse(L, M)` é satisfeito quando a lista `L` é a reversa da lista `M`
- `delete(X, L, M)` é satisfeito quando a lista `M` é obtida da lista `L` pela remoção de todas as ocorrências de `X` em `L`

Conjuntos (listas sem repetições)

- `subset(X, Y)` é satisfeito quando `X` é um subconjunto de `Y`, isto é, todos os elementos de `X` estão em `Y`
- `intersection(X, Y, Z)` é satisfeito quando a lista `Z` contém todos os elementos comuns a `X` e a `Y`, e apenas estes
- `union(X, Y, Z)` é satisfeito quando a lista `Z` contém todos os elementos que estão em `X` ou em `Y`, e apenas estes

Coleção de soluções

- Considerando a seguinte base de dados:

```
filha(marta,charlotte).
filha(charlotte,caroline).
filha(caroline,laura).
filha(laura,rose).

descendente(X,Y) :- filha(X,Y).

descendente(X,Y) :- filha(X,Z),
                    descendente(Z,Y).
```

E a questão:

```
descendente(marta,X).
```

Há quatro soluções (`X=charlotte`, `X=caroline`, `X=laura`, e `X=rose`).

Coleção de soluções – `findall`

- `findall(X, M, L)` instancia `L` a uma lista contendo todos os objetos `X` para os quais a meta `M` é satisfeita. O argumento `M` é um termo que será usado como meta. A variável `X` deve aparecer em `M`.

- Exemplo:

```
findall(X, descendente(marta,X), Z).
```

  Resposta:

```
X = _7489
Z = [charlotte,caroline,laura,rose]
```

Coleção de soluções – `findall` (cont...)

- O `findall` reune todas as soluções. Exemplo:

```
findall(Filha, descendente(Mae,Filha), Lista).
```

Resposta:

```
Filha = _6947
Mae = _6951
Lista = [charlotte,caroline,laura,rose,caroline,
         laura,rose,laura,rose,rose]
```

Coleção de soluções – bagof

- O `bagof` agrupa soluções para cada instância de uma variável.
  Exemplo:

```
bagof(Filha, descendente(Mae,Filha), Lista).
```

  Resposta:

```
Filha = _7736
Mae = caroline
Lista = [laura,rose] ;

Filha = _7736
Mae = charlotte
Lista = [caroline,laura,rose] ;

Filha = _7736
Mae = laura
Lista = [rose] ;

Filha = _7736
Mae = marta
Lista = [charlotte,caroline,laura,rose] ;

no
```

Coleção de soluções – bagof (cont...)

- Outro uso (mais flexível) de `bagof`:

```
bagof(Filha, Mae ^ descendente(Mae,Filha), Lista).
```

Dê uma lista de todos os valores de `Filha` para `descendente(Mae,Filha)` e coloque os resultados em uma lista, mas não se preocupando sobre a geração de listas separadas para cada valor de `Mae`

```
Filha = _7870
Mae = _7874
Lista = [charlotte,caroline,laura,rose,caroline,
         laura,rose,laura,rose,rose]
```

- Observação: enquanto `findall` retorna lista vazia se não houver nenhuma resposta, o `bagof` falha retornando `no`

Coleção de soluções – setof

- O mesmo que bagof, mas com a ordenação das respostas e sem repetições. Exemplo:

```
age(harry,13).
age(draco,14).
age(ron,13).
age(hermione,13).
age(dumbledore,60).
age(hagrid,30).


findall(X, age(X,Y), Out).
X = _8443
Y = _8448
Out = [harry,draco,ron,hermione,dumbledore,hagrid]


setof(X, Y ^ age(X,Y), Out).
X = _8711
Y = _8715
Out = [draco,dumbledore,hagrid,harry,hermione,ron]
```

Coleção de soluções – setof (cont...)

```
age(harry,13).
age(draco,14).
age(ron,13).
age(hermione,13).
age(dumbledore,60).
age(hagrid,30).
```

```
findall(Y, age(X,Y), Out).
Y = _8847
X = _8851
Out = [13,14,13,13,60,30]
```

```
setof(Y, X ^ age(X,Y), Out).
Y = _8981
X = _8985
Out = [13,14,30,60]
```

Outros

- `X =.. L` é satisfeito se `X` é um termo e `L` é uma lista onde aparecem o funtor e os argumentos de `X` na ordem. Exemplos:

```
?- gosta(maria, pedro) =.. L.
L = [gosta, maria, pedro]

?- X =.. [a, b, c, d].}
X = a(b, c, d)
```

- `random(N)` em SWI Prolog é um operador que pode ser usado em uma expressão aritmética à direita de `is`, e produz um inteiro aleatório no intervalo `0` a `N-1`. Exemplo: `X is random(30000).`

Outros (cont...)

- `;` é um operador binário que significa "ou". É satisfeito quando uma das duas metas é satisfeita. Em geral, pode ser substituído por duas cláusulas. Por exemplo,

  ```
  atomic(X) :- (atom(X) ; number(X)).
  ```

  é equivalente a

  ```
  atomic(X) :- atom(X).
  atomic(X) :- number(X).
  ```

Depuração (I)

- Mesmo com os cuidados, podem ocorrer erros de execução ou respostas inesperadas

- Há vários predicados pré-definidos de depuração – auxílio à localização e correção de erros – em Prolog (existenstes em SWI Prolog; em outras implementações, podem variar)

- O predicado `trace`, sem argumentos, liga o mecanismo de acompanhamento de metas. Eventos possíveis:
  - **Call** quando ocorre uma tentativa de satisfação da meta
  - **Exit** quando a meta é satisfeita
  - **Redo** quando a meta é ressatisfeita
  - **Fail** quando a meta falha

- Para cancelar este efeito, há o predicado `notrace`

Depuração (II)

- O predicado `spy(P)` acompanha eventos relacionados às metas do predicado `P`
- Para cancelar este efeito, `nospy(P)`
- `debug` habilita o modo "debug", onde Prolog pára em pontos previamentes estabelecido
- `nodebug` desbilita o modo "debug"
- `debugging` para indicar o status da depuração e listagem de todos os predicados sob espionagem

Depuração (III)

- O acompanhamento de metas, quando ligado, pára a execução em cada evento relevante

- Opções de controle, escolhidas por teclas (primeira letra de um verbo em inglês que lembra a ação), de como continuar o acompanhamento:

Opção	Verbo	Descrição
w	write	imprime a meta
c	creep	segue para o próximo evento
s	skip	salta até o próximo evento desta meta
l	leap	salta até o próximo evento acompanhado
r	retry	volta à primeira satisfação da meta
f	fail	causa a falha da meta
b	break	inicia uma sessão recursiva do interpretador
a	abort	interrompe a depuração

Depuração (IV)

- Exemplo de base:

```
progenitor(maria,joao).
progenitor(jose,joao).
progenitor(maria,ana).
progenitor(jose,ana).
```

- Exemplo de depuração:

```
?- trace, progenitor(maria,X).
   Call: (7) progenitor(maria, _G222) ? creep
   Exit: (7) progenitor(maria, joao) ? creep
X = joao ;
   Redo: (7) progenitor(maria, _G222) ? creep
   Exit: (7) progenitor(maria, ana) ? creep
X = ana.
```

Exercícios (I)

Aplique a depuração, usando uma lista com cinco números, nos seguintes programas recursivos:

- Comprimento da lista:

```
listlen([ ], 0).
listlen([H|T], N) :- listlen(T, N1), N is N1 + 1.
```

- Cálculo de máximo da lista:

```
maximo_lista([X],X) :- !.
maximo_lista([X|Xs], M):- maximo_lista(Xs, M), M >= X.
maximo_lista([X|Xs], X):- maximo_lista(Xs, M), X > M.
```

- Estude e execute os 26 primeiros exercícios de P-99

Exercícios (II)

- Fibonacci - versão ineficiente (tempo exponencial):

```
fib(0,0).
fib(1,1).
fib(N,F) :- N>1,
    N1 is N-1, fib(N1,F1), N2 is N-2, fib(N2, F2),
    F is F1+F2.
```

- Fibonacci - versão eficiente com acumulador (tempo linear):

```
fibacc(N,N,F1,F2,F) :-          %caso base ao atingir N
    F is F1+F2.
fibacc(N,I,F1,F2,F) :- I<N,     %contador < N
    Ipls1 is I+1, F1New is F1+F2, F2New is F1,
    fibacc(N,Ipls1,F1New,F2New,F).
```

- Defina `fibo(N,F)`, para `N>1`, usando `fibacc(N,2,1,0,F)`