Universidade Federal de Santa Catarina
Campus Araranguá
Engenharia de Computação
ARA7502 – Lógica Aplicada a Computação
Prof. Gustavo Mello Machado

Trabalho Prolog 03 - Aula 03/11/2016

Orientações preliminares.

- É permitida a realização deste trabalho individualmente ou em duplas.
- As entregas serão aceitas exclusivamente via Moodle
- Este trabalho comporá a nota da avaliação E1 como previsto no plano de ensino.

CONTROLE DE FLUXO

Cláusula de Horn

Em lógica, uma cláusula consiste de uma disjunção (ou mesmo conjunção) de literais, onde literais representam fórmulas atômicas. As Cláusulas de Horn são as fórmulas bem formadas cujos literais estão conectados pela operação de disjunção e no máximo um destes literais não se apresenta na negativa. Por exemplo, a seguinte fórmula

$$\neg A(x) \vee \neg B(x) \vee C(x)$$

representa uma Cláusula de Horn. Note que pela regra de equivalência de De Morgan esta fórmula equivale a

$$\neg[A(x) \wedge B(x)] \vee C(x)$$

que pela regra de equivalência da CONDICIONAL equivale a

$$A(x) \wedge B(x) \rightarrow C(x)$$

Esta é representada em Prolog pelas *regras*, como no seguinte exemplo

```
C(x) :- A(x),B(x).
```

Observe que os *fatos* em Prolog também representam Cláusulas de Horn, onde apenas o literal não negado se apresenta.

Motor de Inferência

Prolog inclui o Modus Ponens para aplicar o *método de resolução*, usado pelo seu núcleo chamado *motor de inferência*, e ser capaz de responder às *consultas*. Por exemplo, a partir de

$$A(a) \text{ e } \neg A(a) \vee B(b)$$

Prolog é capaz de inferir que $B(b)$. Variáveis em Prolog são tratadas como universalmente quantificadas, e, portanto, são aplicadas sucessivas particularizações universais durante o processo de resolução. Por exemplo, considere o seguinte banco

```
come(urso,peixe).
come(urso,raposa).
come(veado,grama).


animal(urso).
animal(peixe).
animal(raposa).
animal(veado).


planta(grama).


presa(X) :- come(Y,X),animal(X).
```

onde temos um conjunto de fatos baseados nos predicados `come` estabelecendo que o primeiro elemento se alimenta do segundo, `animal` que estabelece um elemento como sendo um animal e `planta` que estabelece um elemento com sendo uma planta. Por fim temos a regra `presa`, que estabelece que se um determinado elemento é comido e é um animal, então ele é uma presa. Ao realizarmos a consulta por "quais X são presas?", o *motor de inferência* do Prolog buscará em seu banco de dados por cláusulas que satisfaçam `presa(X)`, ou seja, que tenham como consequente `presa(X)`, e encontrará a regra

```
presa(X) :- come(Y,X),animal(X).
```

cuja representação em Cláusula de Horn é dada por

$$\neg come(y,x) \vee \neg animal(x) \vee presa(x)$$

Note que sua interpretação em Prolog seria na verdade

$$(\forall x)(\forall y)[\neg come(y,x) \vee \neg animal(x) \vee presa(x)]$$

Prolog então busca por outras cláusulas que podem ser resolvidas em conjunto a esta. No caso, são realizadas particularizações universais ao tentar-se encontrar cláusulas (*fatos* ou *regras*) que satisfaçam os predicados componentes desta regra da esquerda para a direita. O primeiro a ser encontrado seria

$$come(urso, peixe)$$

Que reduz a busca para

$$\neg animal(peixe) \vee presa(peixe)$$

Novamente ao se encontrar na base de dados o fato

$$animal(peixe)$$

pode se concluir que $presa(peixe)$, ou seja, o resultado da consulta por `presa(X)` será verdadeiro para `X = peixe`. Como este não seria o único valor de `X` a satisfazer a consulta, Prolog continua sua busca aplicando *backtracking* após o usuário pressionar o a tecla ponto e vírgula ';'.

Backtracking

Considere o seguinte programa em Prolog

Exemplo 5.1. Números binários com três dígitos

```
d(0).                              % cláusula 1
d(1).                              % cláusula 2
b([A,B,C]) :- d(A),d(B),d(C).      % cláusula 3
```

Ao realizar a seguinte consulta

```
?- b(N)
```

O motor de inferência selecionará a cláusula 3, fará `N = [A,B,C]` e reduzirá a consulta a

```
?- d(A),d(B),d(C)
```

Então, Prolog selecionará o predicado mais à esquerda e tentará resolvê-lo. Para tal, ele teria duas opções, i.e., a cláusula 1 e a cláusula 2. O motor de inferência então seleciona a primeira opção, que estabelece que `A = 0`, ou seja, determina que `N = [0, B, C]` e realiza a seguinte consulta

```
?- d(B),d(C)
```

E de maneira análoga resolve que `B = 0` e `C = 0`, obtendo `N = [0, 0, 0]`. Este resultado é impresso na saída e ao pressionarmos ponto e vírgula o sistema tentará buscar por uma solução alternativa. Para tal, o motor de inferência realiza um processo de *backtracking* onde ele retrocede da última escolha realizada e selecionará a próxima alternativa, ou seja, retrocederá do `C = 0` e escolherá `C = 1`, obtendo `N = [0, 0, 1]`. Este processo é realizado sucessivamente até que todas as alternativas sejam retornadas.

Cortes

Porém, nem sempre é desejável que todas as possibilidades sejam encontradas. Portanto, podemos instruir Prolog a não realizar retrocesso a partir de certo ponto por meio de uma cláusula de corte (ou poda) representada pelo símbolo `!`. Por exemplo, digamos que desejamos retornar os números binários de três dígitos, excluindo os que se iniciam com o número 1. Poderíamos incluir a seguinte regra

```
bin([A,B,C]) :- d(A),!,d(B),d(C).
```

Neste caso, ao realizar a consulta

```
?- bin(N).
```

o comando de corte evitaria o retrocesso após explorar todas as combinações possível de valores para N = `[A,B,C]` onde `A` terá apenas o primeiro valor encontrado, i.e., `A = 0`.

Estrutura Condicional

Em Prolog podemos criar um predicado para representar a estrutura condicional *if-then-else*.

Exemplo 5.2. Comando if-then-else

```
if(Condition,Then,Else) :- Condition, !, Then.    % cláusula 1
if(_,_,Else) :- Else.                            % cláusula 2
```

Para entendermos o funcionamento do exemplo 5.2 considere a seguinte consulta:

```
?- if(8 mod 2 =:= 0,write(par),write(ímpar)).
```

Primeiramente o motor de inferência procura avaliar esta consulta frente a cláusula 1, como `Condition` é tida como verdadeira, Prolog passa a avaliar na sequência `!` e `Then`. Ao avaliar `Then` como `write(par)` Prolog imprime na saída a palavra `par` e encerra a consulta ao cortar o processo de *backtracking* quando encontra `!`. No entanto, se realizarmos a consulta:

```
?- if(5 mod 2 =:= 0,write(par),write(ímpar)).
```

Primeiramente o motor de inferência tenta avaliar a cláusula 1. Como neste caso `Condition` é tido como falso e o motor de inferência fica sem opções para resolver a cláusula 1, ele passa a avaliar a cláusula 2. Neste caso, não importando os valores atribuídos a `Condition` e `Then`, o sistema irá chamar `write(ímpar)` para avaliar `Else`, o que irá imprimir `ímpar` na saída.

Usando Falhas

Prolog possui uma proposição de falha (`fail`), que indica falha na consulta para todos os casos. Esta pode ser usada, por exemplo, para impedir que o sistema pause e aguarde pela entrada ponto e vírgula do usuário para continuar a busca. Por exemplo, dado o exemplo 5.1 no início deste documento, se quisermos listar todos os números binários com três dígitos sem interação com o usuário, podemos escrever os resultados na saída padrão e usar o comando `fail` como segue:

```
bin :- d(A),d(B),d(C),write([A,B,C]),nl,fail.
```

Neste caso ao realizarmos a consulta

```
?- bin.
```

Sempre que Prolog avaliar `d(A)`, `d(B)` e `d(C)` como verdadeiros ele irá chamar `write([A,B,C])` para os valores de `A`, `B` e `C` que foram encontrados, o que imprimirá na saída com os respectivos valores seguida de uma quebra de linha `nl`. Como ao tentar avaliar `fail`, o motor de inferência encontrará uma falha para aquela combinação, ele continuará a sua busca, o que irá gerar a saída completa. Note que o sistema só pausa ao encontrar uma combinação válida de valores durante uma consulta e como o comando `fail` impede que qualquer combinação seja tida como válida, o sistema irá executar até esgotar todas as tentativas.

EXERCÍCIOS

1) O programa a seguir associa a cada pessoa seu esporte favorito

```
joga(ana,volei).
joga(bia,tenis).
joga(ivo,basquete).
joga(eva,volei).
joga(leo,tenis).
```

Podemos realizar uma consulta por parceiros `P` para jogar com Leo de duas maneiras:

a) `?- joga(P,X), joga(leo,X), P\=leo`
b) `?- joga(leo,X), joga(P,X), P\=leo`

Qual consulta é mais eficiente? Por quê?

2) O predicado `num` classifica números em três categorias: positivos, nulos e negativos. Como está definido, este predicado faz com que o motor de inferência realize retrocessos desnecessários, ou seja, segue realizando testes que sempre serão inválidos para determinadas consultas. Explique porque isto acontece e utilize cortes para eliminar estes retrocessos desnecessários.

```
num(N,positivo) :- N>0.
num(0,nulo).
num(N,negativo) :- N<0.
```

3) Dado o banco de dados a seguir, realize as consultas solicitadas para listar todas as respostas sem interrupção.

```
lingua(frances).
lingua(alemao).
lingua(italiano).
lingua(romanche).
lingua(ingles).

fala(alemanha,alemao).
fala(franca,frances).
fala(italia,italiano).
fala(canada,ingles).
fala(canada,frances).
fala(eua,ingles).
fala(congo,frances).
fala(suica,frances).
fala(suica,alemao).
fala(suica,romanche).
fala(suica,italiano).
```

a) Quais países falam francês?
b) Quais línguas são faladas no Canadá?
c) Quais línguas são faladas na Suíça?