# Aula prática II

Alexsandro Santos Soares
prof.asoares@gmail.com

Programação Lógica
Faculdade de Computação
Universidade Federal de Uberlândia

28 de março de 2011

## 1 Objetivos

Esta aula tem por objetivos:

- Consolidação do uso de regras de implicação e conjunção.
- Utilização de definições recursivas.
- Utilização do processamento de listas em Prolog.

## 2 Ambiente de Desenvolvimento Integrado (IDE)

O SWI-Prolog vem com uma coleção de ferramentas de desenvolvimento que podem ser acessadas via linha de comando. Algumas destas ferramentas serão descritas abaixo.

### 2.1 Comandos da IDE

```
?- help.
```

Abre uma cópia local do manual do SWI em uma nova janela.

- Clique sobre a índice à esquerda para navegar pelo manual. **Importante**: No capítulo `Built-in Predicates` você encontrará a documentação dos predicados padrões do Prolog.
- Digite um termo (ex. "member") na parte inferior direita e clique `Help` para saltar para a página do manual deste termo.
- Digite uma palavra-chave (ex. "sel") e clique `Search` para uma lista de emparelhamentos parciais.

```
?- guitracer.
```

Chaveia para o modo gráfico de depuração, utilizado com os comandos `trace` e `spy`.

### 2.2 Alguns outros comandos úteis

Os comandos seguintes podem ser digitados na linha de comando do SWI-Prolog.

```
?- listing.
```

Exibe uma listagem do conteúdo corrente da base de dados do Prolog (fatos e predicados).

```
?- make.
```

Procura alterações nas datas dos arquivos consultados pelo Prolog e os reconsulta se ocorrer alguma alteração.

```
?- help(Topico).
```

Abre o manual no tópico especificado, ex. `help(member).`

```
?- apropos(Palavra).
```

Abre o manual e faz uma busca usando a palavra digitada, ex. `apropos(append).`

```
?- halt.
```

Fecha o SWI-Prolog, sem qualquer pedido de confirmação.

## 3 Uso de regras

**Ex.** 1 Considere a seguinte situação:

```
Um objeto A está sobre uma mesa.
Um objeto B está sobre o objeto A.
```

(a) Instrua o interpretador de Prolog sobre esta situação, usando para isto o predicado `sobre/2`.

(b) Expresse por meio de fórmulas lógicas (e em seguida na notação do Prolog) a seguinte regra de conhecimento:

```
Se um objeto está sobre outro,
então este objeto está acima do outro.
```

O predicado `acima_de` é transitivo.

(c) Coloque a seguinte questão ao interpretador Prolog: "B está acima da mesa?".

**Ex. 2** Escreva um provador de teoremas para o cálculo proposicional. O provador deverá ser capaz de lidar com equivalências, implicações, disjunções, conjunções e negação. A seguir descrevem-se os predicados a serem usados nas definições.

- `equiv`: para a equivalência;
- `impl`: para a implicação;
- `ou`: para a disjunção;
- `e`: para a conjunção;
- `nao`: para a negação.

Defina o provador através do predicado `prova/1`. Teste o provador com os seguintes exemplos:

```
?- prova(impl(falso,verdade)).
true

?- prova(imp(verdade,falso)).
false

?- prova(equiv(equiv(falso,verdade),falso)).
true

?- prova(impl(verdade,X)).
X = verdade

?- prova(impl(falso,X)).
X = _G155
```

# 4 Recursão

**Ex. 3** Imagine que a base de conhecimentos seguinte descreva um labirinto. Os fatos determinam quais pontos estão conectados, ou seja, os pontos que se pode alcançar com exatamente um passo. Além disto, imagine que todos os caminhos são ruas de mão única, tal que você somente pode caminhar por elas em uma direção. Assim, você poderá ir do ponto 1 para o ponto 2, mas não poderá ir de 2 para 1.

```
conectado(1,2).
conectado(3,4).
conectado(5,6).
conectado(7,8).
conectado(9,10).
conectado(12,13).
conectado(13,14).
```

```
conectado(15,16).
conectado(17,18).
conectado(19,20).
conectado(4,1).
conectado(6,3).
conectado(4,7).
conectado(6,11).
conectado(14,9).
conectado(11,15).
conectado(16,12).
conectado(14,17).
conectado(16,19).
```

Escreva um predicado `caminho/2` que diga até onde se pode chegar partindo de um determinado ponto, seguindo as conexões na base de conhecimento. Pode-se ir do ponto 5 para o ponto 10? Em quais outros pontos se pode chegar partindo do ponto 1? E quais pontos podem ser alcançados saindo do ponto 13?

**Ex. 4** Imagine que você resolva passear mundo afora e possua a seguinte base de conhecimento sobre opções de transporte entre cidades:

```
deCarro(auckland,hamilton).
deCarro(hamilton,raglan).
deCarro(valmont,saarbruecken).
deCarro(valmont,metz).

deTrem(metz,frankfurt).
deTrem(saarbruecken,frankfurt).
deTrem(metz,paris).
deTrem(saarbruecken,paris).

deAviao(frankfurt,bangkok).
deAviao(frankfurt,singapore).
deAviao(paris,losAngeles).
deAviao(bangkok,auckland).
deAviao(losAngeles,auckland).
```

Escreva um predicado `viagem/2` que determine se é possível viajar de um lugar a outro usando qualquer meio de transporte disponível: carro, trem e avião. Por exemplo, seu programa deveria responder `true` para a consulta `viagem(valmont,raglan)`.

**Ex. 5** Usando o predicado `viagem/2` para consultar a base de dados anterior pode-se deduzir que é possível ir de Valmont para Raglan. No caso de estar planejando uma viagem, isto é uma boa informação, mas o que você realmente deseja saber é *como* exatamente ir de Valmont a Raglan. Escreva um predicado `viagem/3` que diga a você como viajar de um lugar para o outro. Exemplos de consultas:

```
?- viagem(valmont,paris,vai(valmont,metz,vai(metz,paris))).
true

?- viagem(valmont,losAngeles,X).
X = vai_de(valmont,metz,vai_de(metz,paris,vai_de(paris,losAngeles)))
```

**Ex. 6** Estenda o predicado `viagem/3` tal que ele não somente diga a você em quais cidades intermediárias você passará, mas também *como* ir de uma cidade a outra, ou seja, de carro, trem ou avião.

## 5 Listas

**Ex. 7** Como Prolog responde às seguintes consultas?

(a) `[a,b,c,d] = [a,[b,c,d]].`

(b) `[a,b,c,d] = [a|[b,c,d]].`

(c) `[a,b,c,d] = [a,b,[c,d]].`

(d) `[a,b,c,d] = [a,b|[c,d]].`

(e) `[a,b,c,d] = [a,b,c,[d]].`

(f) `[a,b,c,d] = [a,b,c|[d]].`

(g) `[a,b,c,d] = [a,b,c,d,[]].`

(h) `[a,b,c,d] = [a,b,c,d|[]].`

(i) `[] = _.`

(j) `[] = [_].`

(k) `[] = [_|[]].`

**Ex. 8** Suponha que seja dada uma base de conhecimento com os seguintes fatos:

```
trad(eins,um).
trad(zwei,dois).
trad(drei,tres).
trad(vier,quatro).
trad(fuenf,cinco).
trad(sechs,seis).
trad(sieben,sete).
trad(acht,oito).
trad(neun,nove).
```

Escreva um predicado `tradlista(A,P)` que traduz uma lista de números escritos em alemão, para uma lista correspondente em português. Alguns exemplos de consultas:

```
?- tradlista([eins,neun,zwei],X).
X = [um,nove,dois]

?- tradlista(X,[um, sete, seis, dois]).
X = [eins,sieben,sechs,zwei].

?-
```

Dica: para responder a esta questão, pergunte-se "Como eu traduzo uma lista *vazia* de números?". Este é o caso base. Para listas não vazias, primeiro traduza a cabeça da lista, depois use recursão para traduzir a cauda.

**Ex. 9** Escreva um predicado `duasVezes(Ent,Sai)` cujo argumento `Ent` é uma lista e cujo argumento `Sai` é uma lista consistindo de todos os elementos da primeira lista escritos duas vezes. Por exemplo,

```
?- duasVezes([a,4,bonde],X).
X = [a,a,4,4,bonde,bonde].

?- duasVezes([1,2,1,1],X).
X = [1,1,2,2,1,1,1,1].
```

Dica: para responder a esta questão, primeiro pergunte-se "O que deveria acontecer quando o primeiro argumento é a lista *vazia*?". Este é o caso base. Para listas não vazias, pense sobre o que você deveria fazer com a cabeça e use recursão para tratar a cauda.