Programação em Lógica

Prof. A. G. Silva

22 de setembro de 2016

Entrada e saída de dados

- Leitura e escrita de termos
- Leitura e escrita de caracteres
- Leitura e escrita de arquivos
- Influência dos operadores no modo como a leitura e a escrita ocorrem
- Outros predicados pré-definidos
- Descrição baseada no SWI Prolog (outros sistemas podem diferir a implementação)

Leitura de termos

- Predicado pré-definido `read` para a entrada de termos
- A meta `read(X)` é satisfeita quando `X` unifica com o próximo termo lido no dispositivo de entrada
- É preciso colocar um ponto final para sinalizar o fim do termo, sendo que este ponto não é considerado parte do termo lido
- Unificando ou não, o termo lido é consumido, ou seja, a próxima leitura seguirá daí para frente
- O termo lido pode conter variáveis, que serão tratadas como tal, mas seu escopo se restringe ao termo lido

Leitura de termos (cont...)

- Se o termo lido não tiver a sintaxe de um termo, ocorre erro de leitura
- Se o fim do arquivo for encontrado, `X` será instanciada ao átomo especial `end_of_file`
- Ocorre erro tentar ler após encontrar o fim do arquivo
- Em caso de ressatisfação, `read` falha
- Exemplo (o prompt "`|:`" indica a espera por um termo):

```
pequeno :- read(N), N < 50.

?- pequeno.
|: 40.
true.
```

Apresentação de texto formatada

- Exemplo de uso do predicado pré-definido `format`:

```
?- X='Maria', Y='José',
   format('~w e ~w são irmãos', [X,Y]).

Maria e José são irmãos
X = 'Maria',
Y = 'José'.
```

Escrita de termos

- Predicado pré-definido `write` para a escrita de termos
- Aceita um argumento e imprime o termo instanciado a este argumento no dispositivo de saída
- Se o argumento contém variáveis não instanciadas, estas serão impressas com seus nomes internos, geralmente um "_", seguido de um código interno alfanumérico
- Há também o predicado pré-definido `nl`, sem argumento, para mudança de linha (*newline*). Sua meta também é satisfeita uma vez:

```
?- write(pedro), nl, write(ama), nl, write(maria).
pedro
ama
maria
true.
```

Leitura de caracteres

- Constantes são denotadas com aspas simples. Exemplo, 'e', '\n', etc
- Predicado pré-definido `get_char(X)`, satisfeito unificando `X` com o próximo caractere lido do dispositivo de entrada
- O caractere lido é consumido independentemente de `get_char(X)` ser satisfeito ou não
- O predicado `get_char` falha em tentativas de ressatisfação
- Ao chegar ao fim do arquivo, o átomo especial `end_of_file` é retornado

Leitura de caracteres (cont...)

- Exemplo de leitura em série

```
?- get_char(A), get_char(B), get_char(C), get_char(D).
|: UFSC
A = 'U',
B = 'F',
C = 'S',
D = 'C'.
```

- Exemplo de um predicado que lê e informa o número de caracteres de uma linha, exceto o *newline*:

```
conta_linha(N) :- conta_aux(0, N).

conta_aux(A, N) :- get_char('\n'), !, A = N.
conta_aux(A, N) :- A1 is A + 1, conta_aux(A1, N).
```

Escrita de caracteres

- Predicado pré-definido `put_char(X)`, onde `X` deve ser um caractere, ou um átomo cujo nome tem apenas um caractere

- Se `X` não estiver instanciada ou for outro tipo de termo, ocorre erro

```
?- put_char('A').
A
true.

?- put_char(a).
a
true.

?- put_char('AB').
ERROR
```

Ler e escrever arquivos

- Exemplo de escrita e leitura (cada entrada com ponto final)

```
escrita :-
    open('exemplo.txt', write, X),
    write(X, '\'Universidade Federal de SC\'.'), nl(X), write(X, '2015.'),
    close(X).

leitura :-
    open('exemplo.txt', read, X),
    read(X, U), read(X, A),
    close(X),
    write(U), nl, write(A).
```

- Exemplo de execução

```
?- escrita.
true.

?- leitura.
Universidade Federal de SC
2015
true.
```

- Mais exemplos neste link

Ler e escrever arquivos (cont...)

- Dispositivos correntes
  - `current_input(X)` instancia `X` ao dispositivo corrente de entrada (normalmente o teclado)
  - `current_output(X)` instancia `X` ao dispositivo corrente de saída (normalmente a tela)
- É possível trocar os dispositivos correntes de entrada e saída para arquivos
- Após abrir um arquivo, associando-o a um dispositivo (também chamado de *stream* em Prolog), pode-se usá-lo como entrada ou saída usando os predicados pré-definidos `set_input` e `set_output`

Carregando um banco de dados

- Para carregar arquivos com todas as cláusulas definidas, utilizamos o predicado `consult`
- Quando `X` está instanciado ao nome de um arquivo, a meta `consult(X)` causa a leitura e armazenamento no banco de dados de Prolog das cláusulas contidas neste arquivo
- Esta operação é tão comum que há uma abreviatura para consulta de vários arquivos em uma lista:

```
?- [arq1, arq2, arq3].
```

- O predicado `consult` remove as cláusulas dos predicados consultados no banco de dados antes de carregar as novas definições

Operadores

- Operadores conferem maior legibilidade permitindo formas prefixa, infixa ou posfixa
- É necessário informar a precedência e a associatividade destes operadores
- Apenas funtores de aridade um ou dois podem ser operadores
- Prolog oferece um predicado pré-definido `op(Prec, Espec, Nome)` para definir novos operadores
  - O argumento `Prec` indica a precedência – um inteiro entre 1 e 1200 – e, quanto mais alto este número, maior a precedência
  - O argumento `Espec` serve para definir a aridade, a posição e a associatividade do operador

Operadores (cont...)

- Os seguinte átomos podem ser usados no segundo argumento (`Espec`):

  `xfx xfy yfx yfy`
  `fx fy`
  `xf yf`

  - `f` indica a posição do operador (funtor), e `x` e `y` as posições dos argumentos
  - Na primeira linha, especificações para operadores binários infixos
  - Na segunda linha, especificações para operadores unários prefixos
  - Na última linha, especificações para operadores unários posfixos
  - As letras `x` e `y` dão informações de associatividade
    - `yfx` significa que o operador associa à esquerda
    - `xfy`, à direita
    - `xfx` não associa

Operadores (cont...)

- Exemplos das definições de alguns operadores vistos:

```
?- op(1200, xfx, ':-').
?- op(1200, fx, '?-').
?- op(1000, xfy, ',').
?- op(900, fy, '\+').
?- op(700, xfx, '=').
?- op(700, xfx, '<').
?- op(700, xfx, '>').
?- op(700, xfx, 'is').
?- op(500, yfx, '+').
?- op(500, yfx, '-').
?- op(400, yfx, '*').
?- op(400, yfx, '//').
?- op(400, yfx, '/').
?- op(400, yfx, 'mod').
?- op(200, fy, '-').
```

Outros predicados pré-definidos

- Predicados pré-definidos importantes que não foram tratados até agora, organizados em (alguns vistos na última aula):
  - Tipos X
  - Listas X
  - Conjuntos X
  - Coleção de soluções X
  - Verdadeiros
  - Banco de dados

Aplicação em aprendizagem

- Onde estou?

```
:- dynamic estou/1.   % declara modificação dinâmica

estou('R. Lauro Linhares').

ando(Y) :-
    retract(estou(X)),
    asserta(estou(Y)),
    format('Ando da ~w até a ~w', [X,Y]).
```

- Funcionamento do programa:

```
?- estou(X).
X = 'R. Lauro Linhares'
Yes

?- ando('R. Edu Vieira').
Ando da R. Lauro Linhares até a R. Edu Vieira
Yes

?- estou(X).
X = 'R. Edu Vieira'
Yes
```

Persistência da base de dados

- Os predicados pré-definidos `tell` e `told` podem ser utilizados para a gravação das atualizações em disco:

```
grava(Predicado,Arquivo) :-
  tell(Arquivo),
  listing(Predicado),
  told.
```

- Para recuperar uma base salva em disco, basta efetuar nova consulta (predicado `consult`)

- Veja outros predicados (tell, telling, told, see, seeing, seen, append) neste link

Exercícios

1. Escreva um predicado `estrelas(N)` que imprime `N` caracteres "`*`" no dispositivo de saida.

2. Escreva um predicado `guess(N)` que incita o usuário a adivinhar o número `N`. O predicado repetidamente lê um número, compara-o com `N`, e imprime "`Muito baixo!`", "`Acertou!`", "`Muito alto!`, conforme o caso, orientando o usuário na direção certa.

3. Escreva um predicado que lê uma linha e imprime a mesma linha trocando todos os caracteres 'a' por 'b'.

Exercícios (cont...)

4. Implemente os predicados `liga`, `desliga` e `lâmpada` para que eles funcionem conforme indicado pelos exemplos a seguir:

```
?- liga, lâmpada(X).
X = acessa
Yes

?- desliga, lâmpada(X).
X = apagada
Yes
```

5. O predicado `asserta` adiciona um fato à base de dados, incondicionalmente, mesmo que ele já esteja lá. Para impedir essa redundância, defina o predicado `memorize`, tal que ele seja semelhante a `asserta`, mas só adicione à base de dados fatos inéditos.

Exercícios (cont...)

7. Modifique o programa desenvolvido no exercício anterior de modo que, quando for solicitado ao robô pegar um objeto cuja posição é desconhecida, ele pergunte ao usuário onde está esse objeto e atualize a sua base de dados com a nova informação. Veja um exemplo:

```
?- pos(O,L).
O = robô
L = cozinha ;
O = tv
L = quarto ;
No

?- pegue(lixo), ande(rua), solte(lixo), ande(garagem).
Onde está lixo? quintal
anda de cozinha até quintal
pega lixo
anda de quintal até rua
solta lixo
anda de rua até garagem
Yes

?- pos(O,L).
O = robô
L = garagem ;
O = lixo
L = rua ;
O = tv
L = quarto ;
No
```

Exercícios (cont...)

8. Acrescente também ao programa do robô o predicado `leve(Obj,Loc)`, que leva um objeto até um determinado local. Por exemplo:

```
?- leve(tv,sala).
anda de garagem até quarto
pega tv
anda de quarto até sala
solta tv
Yes
```