

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Disciplina: Paradigmas de Linguagens de Programação

Professor: Franklin Ramalho

Alunos: Ana Cláudia Olinto Rocha	matr.: 20411015
Clerton Ribeiro de Araújo Filho	matr.: 20411018
Hugo de Sousa Marques	matr.: 20411030
Mariana Romão do Nascimento	matr.: 20421006

Prolog

Campina Grande, 14 de setembro de 2006

1. Tipos Primitivos.....	4
2. Tipos Compostos.....	4
2.1. Tipos recursivos.....	5
3. Sistema de Tipos.....	5
4. Checagem de Tipos.....	5
5. Tipos Persistentes.....	5
6. Variáveis.....	5
7. Valores storable.....	7
8. Arrays.....	7
9. Expressões.....	7
10. Comandos.....	7
10.1. Atribuições.....	7
10.2. Expressões com efeitos colaterais.....	7
11. Associações.....	8
12. Escopo das variáveis.....	9
13. Estrutura de blocos.....	10
14. Declarações.....	10
14.1. Declaração de tipos.....	10
14.2. Declaração de tipos.....	10
14.3. Declaração de Variáveis.....	10
14.4. Declaração Sequencial.....	10
14.5. Declaração Colateral.....	11
14.6. Bloco.....	11
14.7. Modularizações.....	11
14.8. Recursão.....	11
15. Abstração.....	11
16. Ordem de Avaliação.....	12
16.1. Propriedade de Church-Rosser.....	12
17. Encapsulamento.....	12
18. Sistema de Tipos.....	12
18.1. Coerção, Subtipos e Herança.....	12
19. Seqüenciadores:.....	12
19.1. Escape: predicado de corte (cut):.....	13
19.2. Desvios incondicionais e Tratamento de exceções.....	13
20. Concorrência.....	13

Introdução

Prolog é uma linguagem de programação declarativa e interpretada que foi inventada por volta de 1970 por Alain Colmerauer e seus associados.

Enquadrada no paradigma de programação em Lógica Matemática, Prolog usa uma base de dados de fatos e de relações lógicas que exprimem o domínio relacional do problema a resolver.

Nesta linguagem de programação, ao invés de explicitamente estabelecermos o que a máquina fará para resolver o problema (programação *procedural* – Pascal, C) ou de especificar uma transformação funcional de dados em repostas (programação *funcional* – LISP), o que se faz é criar um conjunto de regras que guiam o problema que pretendemos resolver e, então, faz-se perguntas para encontrar a solução (programação *lógica*).

Pode-se verificar na tabela abaixo um rol de compiladores Prolog, que variam segundo o sistema operacional e a linguagem que seus compiladores são feitos.

Comparison of Prolog Implementations													
Platform			Features						ToolKit			Prolog Mechanics	
Name	OS	License	Native Graphics	Unicode	Object Oriented	Native OS Control	Stand Alone Executable	C Interface	Java Interface	Interactive Interpreter	Debugger	Code Prolifer	Syntax
DLI-FEOLLO	MS-DOS	shareware	TRUE	TRUE		TRUE	TRUE				TRUE		Edinburgh Prolog
Open Prolog	Mac OS	Freeware									TRUE		
GNU Prolog	Unix, Windows	GPL					TRUE	TRUE	TRUE	TRUE	TRUE		ISO-Prolog
GNU Prolog	Linux, Windows	GPL				TRUE	TRUE	TRUE		TRUE	TRUE		ISO-Prolog
Visual Prolog	Windows	Freeware, Commercial	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE			TRUE		
SWI-Prolog	Unix, Windows, Mac OS X	LGPL	TRUE	TRUE			TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	ISO-Prolog, Edinburgh Prolog
Ju Prolog	IBM	GPL	TRUE	TRUE				TRUE	TRUE	TRUE	TRUE		ISO-Prolog

Fonte: <http://en.wikipedia.org/wiki/Prolog>

Quanto à sintaxe, Prolog é baseada num conjunto da lógica de 1ª ordem (cláusulas de Horn). Portanto, um programa Prolog é uma série de cláusulas (axiomas).

Quanto à semântica, Prolog é baseada nos modelos de Herbrand.

Prolog tem sido aceita pra o Desenvolvimento de Sistemas Especialistas, Interfaces de Linguagem Natural, Banco de Dados, Prototipagem, Prova Automática de Teoremas. Além disso, tem se tornado cada vez mais popular, pois foi selecionada para ser a linguagem básica para o projeto japonês de Quinta Geração.

1. Tipos Primitivos

- Boolean
- Integer

Exemplo:

```
1      | ?- X is 6*7.  
2      X = 42
```

- Float

Exemplo:

```
1      | ?- X is 6.0*7.0.  
2      X = 42.0
```

- Átomos: correspondem a nomes próprios escritos em linguagem natural e que representam relações, funções ou objetos. Podem ser escritos de 3 (três) formas:
 - Seqüências de letras, dígitos e “_”, iniciadas por letra minúscula.

Exemplo:

```
hE_l_l_o99.  
escola.
```

- Seqüências de caracteres especiais (+, -, >, <, *, etc). Neste caso é necessário ter cuidado, uma vez que algumas seqüências têm significado pré-definido (como por exemplo, “:-“).

Exemplo:

```
$$
```

- Seqüências de caracteres entre aspas simples.

Exemplo:

```
´Bom dia´  
´744´
```

- Variáveis: representam objetos definidos, mas não identificados; variáveis no sentido matemático, mas não no sentido usual atribuído nas linguagens imperativas. São denotadas por um string consistindo de letras, números e *underscores* (“_”) e começam com letra maiúscula ou *underscore*.

Exemplo:

```
_al  
Xyz
```

- String: são seqüências de caracteres escritos entre aspas. Internamente são representados como seqüências de códigos de caracteres.

Exemplo:

```
“casa”  
“pessoa”
```

2. Tipos Compostos

Em Prolog não possuímos tipos compostos definidos, ao invés disso utilizamos predicados para simularmos a existência dos diversos tipos compostos.

Por exemplos, se quisermos simular a existência de um registro do tipo “book” que é um tipo composto do tipo produto cartesiano poderíamos usar um predicado onde esse predicado seria o nosso “tipo”. Logo, no exemplo, abaixo temos o “tipo” book que é definido a partir de um author, um title e uma data.

Exemplo:

```
book(author(aaby, anthony), title(lab_manual), data(1991))
```

2.1. Tipos recursivos

Uma lista pode ser definida recursivamente, usando o termo ' . ':

1. O átomo [] é uma lista vazia

2. Se T é uma lista e H é um elemento, então o termo ' . ' (H, T) é uma lista onde H é a cabeça da lista e é seguido pelos componentes do resto da lista (T). A lista [1, 2, 3] poderia ser internamente representada por ' . ' (1, ' . ' (2, ' . ' (3, []))). O *shortcut* [$H \mid T$], é mais utilizado na construção das regras.

Para conveniência do programador, uma lista pode ser construída e destruída de várias formas:

- Enumeração de elementos: [abc, 1, f(x), Y, g(A, rst)]
- Expansão de termos: ' . ' (abc, ' . ' (1, ' . ' (f(x), ' . ' (Y, ' . ' (g(A, rst), []))))))
- Além disso listas contam com uma série de funções prontas para operá-las, resgatar seu conteúdo, entre outras. Por exemplo, o predicado append que é usado para concatenar listas portanto também é usado para construí-las como no exemplo a seguir:
- Através do predicado append:

```
1      append([], Ys, Ys) .
2      append([X|Xs], Ys, [X|Zs]) :-
3          append(Xs, Ys, Zs) .
```

3. Sistema de Tipos

Prolog é uma linguagem fracamente tipada .

4. Checagem de Tipos

Prolog é uma linguagem dinamicamente tipada.

5. Tipos Persistentes

Para armazenamento persistente de dados, Prolog possui arquivos. A gravação em arquivo é feita usando predicados built in, ou seja, predicados já definidos na própria linguagem. Se desejamos

Exemplo:

```
tell(A).
write(Informação).
```

Assim, o predicado built-in tell irá direcionar a saída de Prolog para o arquivo “A”, portanto o write irá escrever a “Informação” em “A”, bem como qualquer predicado built-in que escreva algo.

Note que em Prolog a tela de saída padrão(o terminal Prolog) também é considerada um arquivo com nome user, ou glass_tty, logo se o usuário deseja volta a escrever no terminal basta usar o tell(user) por exemplo.

6. Variáveis

Uma variável em Prolog sempre começa com uma letra maiúscula.

Em Prolog, variáveis são cadeias de letras, números e underscores (“_”). Elas podem ser:

- Variáveis primitivas: Sabendo que as variáveis primitivas são variáveis de tipo primitivo, em Prolog temos as seguinte:

- Boolean: true, fail

- Integer: números inteiros. Exemplo:

```
X = 2
```

Onde X é uma variável primitiva do tipo *integer*

- Real: números em ponto flutuante. Em muitas implementações em Prolog, não há distinção entre números inteiros e números reais.

```
y = 4.321
```

Onde y é uma variável do tipo *real*.

- Atom: letras (A..Z, a..z), dígitos (0..9), underscore (_) e símbolos (+, -, *, /, \, ., ^, >, <, =, ~, :, ?, @, #, \$, &). Os átomos sempre começam com letra minúscula e podem ser construídos de 3 formas:

- Cadeias de letras e/ou dígitos, podendo conter o caracter underscore (_).

Exemplos:

```
peessoa
    _idade
menu_1
```

- Cadeias de símbolos.

Exemplos:

```
<----->
++++
::=
```

- Cadeias de caracteres quaisquer, podendo incluir até mesmo espaços em branco, desde que delimitados por apóstrofo (').

Exemplos:

```
'Rua Treze de Maio'
'Rio Grande do Norte'
'10 de agosto de 2006'
```

- Variáveis compostas: em Prolog, são as variáveis do tipo:

- List: uma lista pode ser vista intuitivamente como um array, sendo que em Prolog, embora as listas sejam ordenadas, não existem os índices como ocorre em Pascal, Java, etc. Mas existem predicados para obter o índice de uma lista.

-

Exemplos:

```
[arara, tigre, macaco]
```

Onde [] é a representação de lista vazia

```
•(arara, •(tigre, •(macaco, [])))).
```

Onde • é um *functor*

- Variável heap: em Prolog, podemos criar e destruir variáveis heap usando, respectivamente, o alocador `new` e o desalocador `dispose`.

Exemplo:

```
1 new(+Type, -Datum),
2 new(+Type, +Size, -Datum) and
3 dispose(+Datum).
```

7. Valores storable

Em Prolog, os valores storable são:

- Boolean
- Atom
- Integer
- Real

8. Arrays

Prolog não tem arrays.

9. Expressões

Expressões aritméticas são avaliadas com a construção de predicado `is` que é usado como um operador infixo da seguinte forma.

Note na linha 1 do código abaixo a expressão aritmética $3*4$ é associada a variável `X`.

Exemplo:

```
1      ?- X is 3*4.
2      X = 12
3      yes
```

10. Comandos

Como Prolog é uma linguagem declarativa, não há nela o conceito de comandos como atribuição.

10.1. Atribuições

Em Prolog o conceito de atribuição está intimamente ligado ao conceito de unificação.

Quando se faz em Prolog qualquer coisa como:

```
X = b,
```

o que acontece é que a variável `X` é unificada com `b`, passando assim a ser a mesma coisa que `b` (o termo `b = X` tem exatamente o mesmo efeito). O mecanismo de unificação de variáveis combina os efeitos de atribuição e de condição e quando uma variável assume um valor (por unificação), já não poderá assumir outro (na mesma solução). Assim, se torna evidente a diferença entre unificação e atribuição de valor a uma variável.

A seguir damos um exemplo de uma atribuição de valor (definitivo):

```
1      | ?- X is 5-3.
2      X = 2
```

10.2. Expressões com efeitos colaterais

Um efeito colateral é o efeito causado por uma expressão, comando ou procedimento que permanece após a execução do mesmo e não é o objetivo principal dessa expressão, comando ou procedimento. Prolog possui efeitos colaterais, por exemplo, tem um número de predicados predefinidos para a leitura e para a escrita.

Um exemplo seria o predicado built-in “write”, que como efeito colateral escreve as respostas no arquivo de saída corrente.

Abaixo, temos um código que faz uso do predicado `write`:

Torre de Hanói:

```
1      hanoi(N) :- move(N, left, centre, right). % :- indica um comando
2      condicional
3      move(0, _, _, _) :- !. % Observe aqui o predicado de corte (!)
4      move(N, A, B, C) :-
5      M is N-1, % este é um exemplo de expressão
6      move(M, A, C, B), inform(A, B), move(M, C, B, A).
7      inform(X, Y) :-
8      write('move a disc from the '), write(X), write(' pole to the '),
9      write(Y), write(' pole'),
10     nl.
```

Nas linhas 7 e 8 usamos o predicado write para escrevermos o resultado do movimento no terminal.

11.Associações

Em Prolog, as associações são feitas pelo símbolo '=' e 'is', com quaisquer tipos de dados, e utiliza-se o conceito de unificação. Esta representa o mecanismo de associação de identificadores de variáveis e que podem ser vistos como um tipo de atribuição. É a substituição das [variáveis](#) presentes em uma série expressões, de forma que estas expressões fiquem idênticas.

Abaixo seguem vários exemplos de como as associações são feitas, bem como exemplos onde essas associações não são possíveis:

- $A = A$: Válida. Tautologia
- $A = B, B = abc$: Ambos A e B são unificados com o átomo abc .
- $xyz = C, C = D$: A unificação é simétrica
- $abc = abc$: Unificação válida
- $Abc = xyz$: Falso para unificação porque os átomos são distintos
- $f(A) = f(B)$: A é unificado com B
- $f(A) = g(B)$: Falso porque o nome das funções é distinto.
- $f(A) = f(B, C)$: Falsa a unificação porque as funções têm numero de termos em parênteses distintos.
- $f(g(A)) = f(B)$: Unifica B com o termo $g(A)$
- $f(g(A), A) = f(B, xyz)$: Unifica A com o átomo xyz e B com o termo $g(xyz)$
- $A = f(A)$: Unificação infinita, A é unificado com $f(f(f(...)))$. Em muitos compiladores modernos de Prolog isso é proibido.
- $A = abc, xyz = X, A = X$: Falso para unificação, pois efetivamente $abc = xyz$.

Variáveis diferentes podem assumir o mesmo valor, como exemplos vamos definir a seguinte relação de irmã:

```
1      pai_ou_mae(tom,bob) .
2      pai_ou_mae(tom,liz) .
3      pai_ou_mae(bob,ana) .
```

```

4 pai_ou_mae(bob,pat).
5 pai_ou_mae(pat,jim).
6 mulher(ana).
7 mulher(pat).
8 irma(X,Y) :- pai_ou_mae(Z,X),
9 pai_ou_mae(Z,Y),
10 mulher(Y).

```

• Consulta:

```

1 | ?- irma(pat,X).
2 X = ana ;
3 X = pat ;

```

Como pat acima possui 2 irmãs que satisfazem a relação presente nas linhas 8, 9 e 10 então a variável X que será retornada com a resposta retorna com os 2 valores possíveis, no caso, Ana e Pat.

12. Escopo das variáveis:

O escopo de todas as variáveis é limitado a uma única cláusula, logo o escopo é estático. Não existem variáveis globais.

Segue-se um exemplo de variáveis que possuem escopo apenas na cláusula que está definida.

```

1 pai_ou_mae(tom,bob).
2 pai_ou_mae(tom,liz).
3 pai_ou_mae(bob,ana).
4 pai_ou_mae(bob,pat).
5 pai_ou_mae(bob,joe).
6 pai_ou_mae(pat,jim).
7 mulher(ana).
8 mulher(pat).
9 homem(joe).
10 homem(bob).
11 irma(X,Y) :- pai_ou_mae(Z,X),
12 pai_ou_mae(Z,Y),
13 mulher(Y).
14 irmao(X,Y) :- pai_ou_mae(Z,X),
15 pai_ou_mae(Z,Y),
16 homem(Y).
17 sobrinho(X,Y) :- irma(X,Z),
18 pai_ou_mae(Z,Y).
19 sobrinho(X,Y) :- irmao(X,Z),
20 pai_ou_mae(Z,Y).

```

Como Prolog não é uma linguagem fortemente tipada, permite amarração dinâmica tanto de valores quanto de tipos. Não há o conceito de amarração estática (i.e., em tempo de compilação), as restrições de tipagem acumulam-se dinamicamente no transcórre da execução do código de um programa, fazendo com que o conjunto potencial de valores de uma variável possa ser drasticamente reduzido e levando à convergência de uma solução que satisfaça às restrições. Essa propagação de restrições é muito importante em metodologias cujos problemas requerem que os valores de certas entidades permaneçam desconhecidos até que informações suficientes sejam obtidas para determinar uma solução satisfatória. Essa característica de retardamento de amarração é chamada amarração atrasada de variáveis e pode ser entendida ainda como a habilidade de manter uma variável explicitamente ilimitada enquanto ela é manipulada no código de um programa.

13.Estrutura de blocos

Em Prolog não há estrutura de blocos.

14.Declarações

14.1.Declaração de tipos

É possível, conforme mostra o exemplo:

```
tipo_lista_de_inteiros = interger*
```

Onde o * é que indica que lista_de_inteiros é uma lista, composta por elementos do tipo inteiro.

14.2.Declaração de tipos

É possível, conforme mostra o exemplo:

```
tipo_lista_de_inteiros = interger*
```

Onde o * é que indica que lista_de_inteiros é uma lista, composta por elementos do tipo inteiro.

14.3.Declaração de Variáveis

Por ser uma linguagem auto-declarativa, não se declaram variáveis em Prolog. Caso seja necessário usar uma variável, apenas se faz uso dela sem a necessidade de declará-la anteriormente.

Exemplo:

```
1    ehPar(X) :- X mod 2 == 0.
```

A variável X pode ser usada mesmo que não seja declarada e nem associada a nenhum tipo em específico.

14.4.Declaração Sequencial

Prolog admite declarações seqüenciais da seguinte maneira:

```
1    Pai(x,z) <- X eh Pai de Z
2    Pai(k,y) <- K eh Pai de Y
```

Ambas as declarações são independentes e podem ser executadas independentemente da ordem sempre da mesma maneira.

14.5.Declaração Colateral

Exemplo:

```
1     avo(X, Y):- pai(X, Z), pai(Z, Y). <- X é avo de Y se X for pai
de um Z e Z for pai de Y
```

A inversão da ordem altera completamente o sentido, ou seja, avo depende da forma como os pais estão dispostos.

14.6.Bloco

Nas consultas o símbolo “:-” marca o início dos blocos e “.” o final.

14.7.Modularizações

Prolog não conta com modularizações.

14.8.Recursão

As regras em Prolog são implicações lógicas, podem depender de fatos, de outras regras e da própria regra com definição recursiva.

Exemplo:

Na relação a seguir temos que “Y está na cadeia alimentar de X”

```
na-cadeia-alimentar(X,Y)
```

Isso pode significar duas coisas:

- 1) X come Y diretamente
- 2) X come algum animal que come algum animal que come algum animal...que come Y

15.Abstração

A idéia de abstração em Prolog está relacionada à de relações seletoras, isto é, relações que selecionam componentes particulares da estrutura sobre a qual se aplicam.

Considere um programa que implemente a caracterização de uma família (pai, mãe, filho, etc). Tais relações seletoras selecionam componentes particulares da estrutura sobre a qual se aplicam. O nome de cada relação seletora é o próprio nome do objeto que ela seleciona e seus argumentos são dois: o objeto que representa a estrutura da qual desejamos selecionar um determinado componente. e o próprio componente a ser selecionado. No caso a seguir, as relações seletoras são pai, mãe e primogênito.

```
1     pai(família(Pai,_,_), Pai).
2     mãe(família(_,Mãe,_) ,Mãe).
3     primogênito(família(_,_, [Prim | _]), Prim_).
```

Uma vez que as relações seletoras estejam definidas, o usuário pode esquecer a forma particular usada na representação de sua estrutura original. Para criar e manipular tal informação é necessário somente conhecer os nomes das relações seletoras e empregar tais nomes ao longo do programa. No caso de representações complicadas, isso é muito mais simples do que usar a representação original de modo implícito.

O uso de relações seletoras também torna os programas mais fáceis de modificar. Suponha que fosse desejado aumentar a eficiência de um programa, mudando a forma de representar sua informação. Tudo que é necessário fazer é mudar as definições das relações seletoras e o restante do programa funcionará sem qualquer alteração com a nova representação.

Visto que Prolog não é uma programação procedural nem funcional, não há exemplos de abstração de procedimento ou de função.

16. Ordem de Avaliação

Prolog possui função estrita, ou seja, a execução da função depende da total avaliação de seus argumentos e utiliza Eager Evaluation, assim avalia o parâmetro real uma vez e substitui seu valor em cada ocorrência do parâmetro associado.

16.1. Propriedade de Church-Rosser

Se uma expressão pode ser avaliada completamente, pode ser avaliada usando Normal-Order; Se uma expressão pode ser avaliada em diferentes ordens, então em todas essas ordens deve ser retornado o mesmo valor.

Como toda linguagem com efeitos colaterais não possui a propriedade de Church-Rosser, podemos concluir que Prolog não possui a referida propriedade visto que Prolog possui efeitos colaterais.

17. Encapsulamento

Prolog não proporciona esta capacidade.

18. Sistema de Tipos

Não existe noção independente de tipos em Prolog, existem predicados que definem implicitamente tipos.

Exemplo: Tipo macho definido como o conjunto dos termos X tais que macho(X) é verdade.

Polimorfismo

“Prolog, na verdade, não é uma linguagem tipada. Logo, não é polimórfica. Mas, por essa mesma razão, você pode dizer que ele é polimórfico. Na verdade, como o Prolog é uma linguagem de programação em lógica, tudo é termo. Por isso não é tipada.

Por outro lado, por não ser tipada, você pode definir qualquer coisa como termo. Assim, você tem listas de inteiros e outros termos que são tratadas por predicados genéricos para listas quaisquer, sem precisar definir um predicado de concatenação de duas listas, por exemplo, para listas só de inteiros, ou listas só de determinados termos.”.

18.1. Coerção, Subtipos e Herança

Como Prolog não é uma linguagem de programação tipada, não há coerção, subtipos ou herança.

19. Seqüenciadores:

Prolog é uma linguagem de programação seqüencial na qual pode-se observar a presença dos seguintes tipos de seqüenciadores:

19.1.Escape: predicado de corte (cut):

O predicado de corte (!) funciona como um tipo de seqüenciador, uma vez que previne a execução do backtracking quando este não for desejado. Observe no exemplo da Torre de Hanói apresentado anteriormente:

```
1     hanoi(N) :- move(N, left, centre, right).
2     move(0, _, _, _) :- !. % Observe aqui o predicado de corte (!)
3     move(N, A, B, C) :- M is N-1, move(M, A, C, B), inform(A, B), move(M,
4     C, B, A).
5     inform(X, Y) :- write('move a disc from the '), write(X), write(' pole to
6     the '), write(Y), write(' pole'), nl.
```

19.2.Desvios incondicionais e Tratamento de exceções

Nenhum desses tipos de seqüenciadores são encontrados em Prolog.

20.Concorrência

As linguagens lógicas concorrentes são linguagens lógicas de programação que podem especificar sistemas abertos reativos e assim podem ser usadas para executar sistemas concorrentes e algoritmos paralelos. Um programa lógico concorrente é um programa lógico com não-determinístico aumentado com sincronização. Um programa lógico aumentado assim pode realizar as noções básicas da concorrência: processos, comunicação, sincronização, e indeterminismo. A leitura de processos de programas lógicos, empregada por programas lógicos concorrentes é diferente da leitura processual empregada por Prolog, por isso Prolog não é uma linguagem de programação concorrente.

Bibliografia

<http://en.wikipedia.org/wiki/Prolog>
<http://www.criarweb.com/artigos/239.php>
<http://www.computacao.uvanet.br/pub/aulas/ia/>
<http://www.inf.uri.com.br/~carla/IA/Aula10.ppt>
http://cs.wvc.edu/~cs_dept/KU/PR/Prolog.html
<http://www.visual-prolog.com/vip6/tutorial/tut04/default.htm>
<http://www.visual-prolog.com/vip6/tutorial/tut04b/default.htm>