Introdução

Paradigma de Programação Lógico

Marco A L Barbosa

Conteúdo

O estudo utilizando apenas este material **não é suficiente** para o entendimento do conteúdo. Recomendamos a leitura das referências no final deste material e a resolução (por parte do aluno) de todos os exercícios indicados.

Introdução

Introdução

No paradigma de **programação declarativo**, as estruturas e os elementos do programa são escritos de maneira a especificar a lógica da computação sem descrever o fluxo de controle.

Consequência da transparência referencial.

Imperativo vs declarativo

Imperativo vs declarativo

- Imperativo
  - Modelo de computação baseado em sequência passo a passo de comandos
  - Atribuições destrutivas
  - Ordem de execução é crucial, os comandos só podem ser entendidos no contexto das computações anteriores devido aos efeitos colaterais
  - Controle é responsabilidade do programados
  - Exemplos: Java, C, Pascal

Imperativo vs declarativo

- Declarativo
  - Modelo de computação baseado em um sistema onde as relações são especificadas diretamente em termos da entrada
  - Atribuição não destrutiva
  - A ordem de execução não importa (não tem efeitos colaterais)
  - O programador não é responsável pelo controle
  - Exemplos: SQL, Prolog, Haskell
- Alguns autores consideram “como” (imperativo) vs “o que” (declarativo)

Funcional vs lógico

Funcional vs lógico

- Os dois principais paradigmas declarativos são o funcional e o lógico
- Funcional
  - Baseado em declaração e aplicação de funções (cálculo lambda)
  - Todos os parâmetro de uma função precisam estar instanciados
  - Clara distinção entre entrada e saída
- Lógico
  - Baseado no cálculo de predicados
  - Objetos e relações
  - A computação é feita usando um mecanismo de inferência lógico
  - A computação pode ser realizada com variáveis não instanciadas

Prolog

Prolog

- Para estudar o paradigma lógico vamos utilizar a linguagem Prolog
- Existem muitas implementações
- Vamos utilizar o SWI-Prolog

Primeiros passos

Instalação e execução

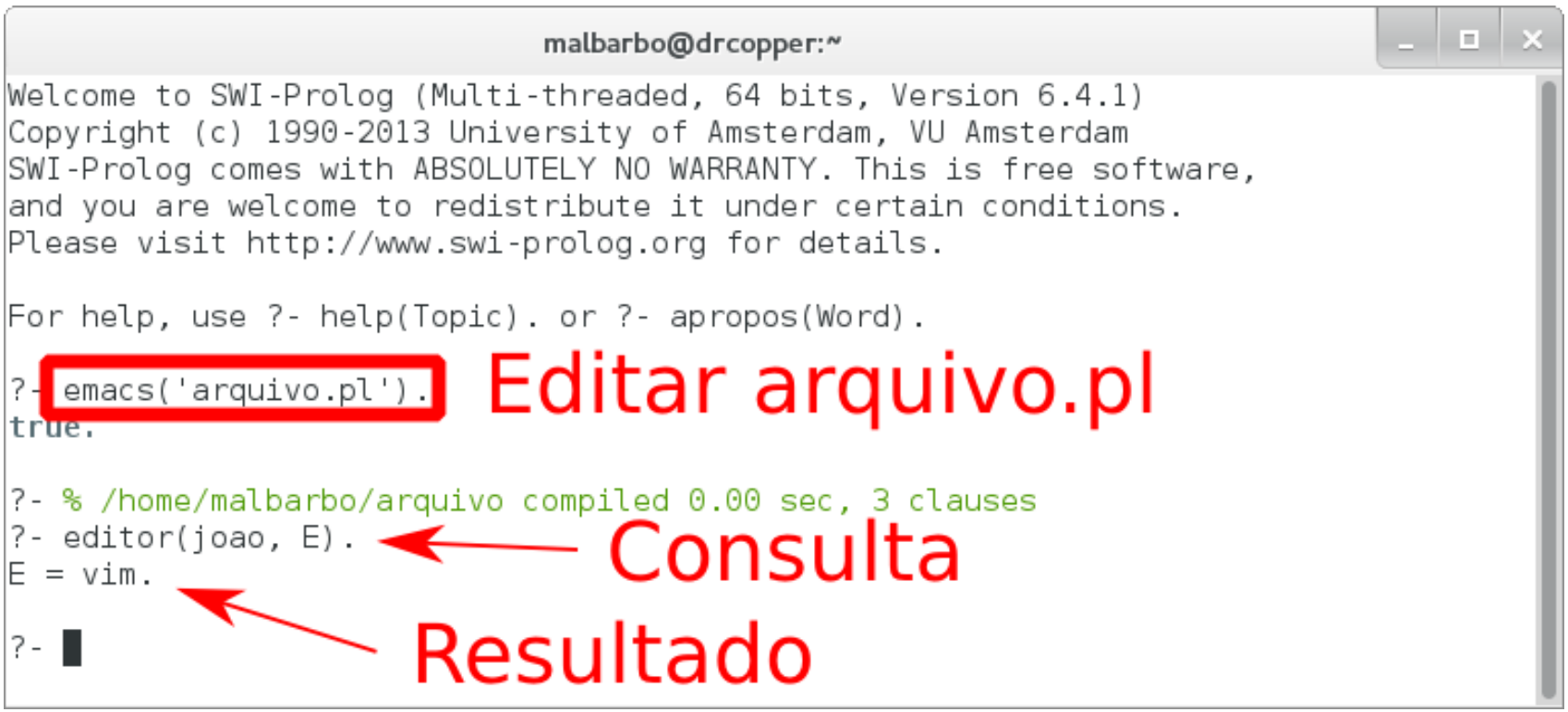
- Instalação

```
$ apt-get install swi-prolog
```

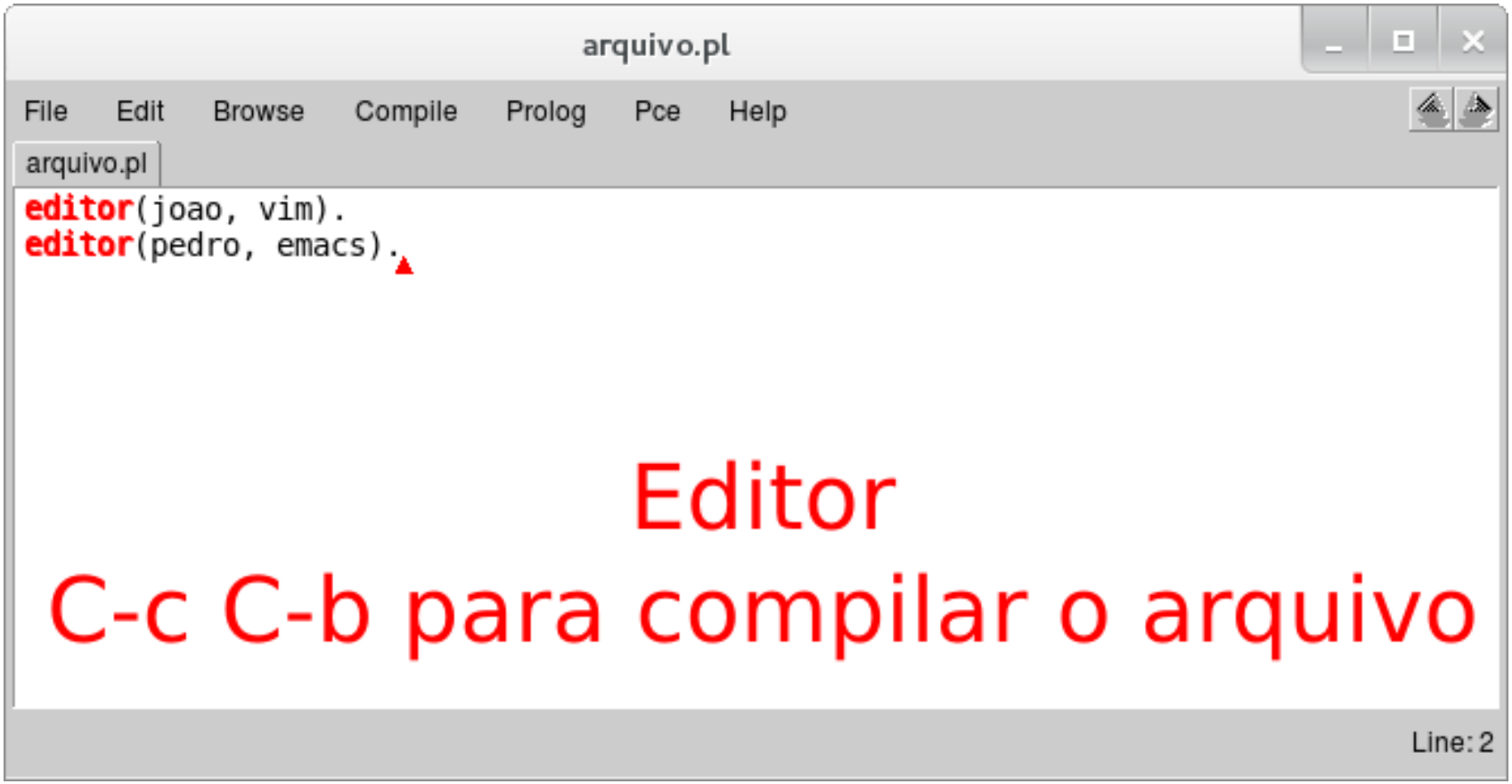
- Execução

```
$ swipl
```

Edição e consulta com editor integrado



Editor integrado



Edição e consulta com outro editor

- Editar o arquivo usando o editor de sua preferência
- Ler o arquivo no swipl

```
?- consult('arquivo.pl').
```

- Fazer consultas

```
?- editor(joao, E).
E = emacs.
```

- Depois de alterar o arquivo, ele deve ser lido novamente

Tutorial

Tutorial

- Neste tutorial não seremos muito formais
- Programar em Prolog consiste em
  - Especificar fatos sobre objetos e suas relações
  - Definir regras sobre objetos e suas relações
  - Fazer consultas (perguntas) sobre objetos e suas relações

Fatos

Fatos

- Um fato é algo que é verdadeiro sobre uma relação de objetos
- Fato: João utiliza o editor vim.

```
editor(joao, vim).
```

- `joao` e `vim` são objetos, `editor` é uma relação
- Os nomes das relações e dos objetos devem começar com letras minúsculas
- A ordem dos objetos é arbitrária, mas você deve ser consistente
- Os objetos de uma relação são chamados de argumentos
- O nome da relação é chamado de predicado
- O número de argumentos de um predicado é a aridade do predicado

Fatos

- Uma relação pode ter qualquer quantidade de argumentos
- Fato: Está chovendo.

Fatos

- Uma relação pode ter qualquer quantidade de argumentos
- Fato: Está chovendo.

```
chovendo.
```

Fatos

- Uma relação pode ter qualquer quantidade de argumentos
- Fato: Está chovendo.

```
chovendo.
```

- Fato: Maria comprou um livro do Jorge.

Fatos

- Uma relação pode ter qualquer quantidade de argumentos
- Fato: Está chovendo.

```
chovendo.
```

- Fato: Maria comprou um livro do Jorge.

```
comprou(maria, livro, jorge).
```

Consultas

Consultas

- Podemos fazer consultas sobre os fatos que foram definidos
- Dado os seguintes fatos

```
editor(joao, vim).
editor(pedro, emacs).
```

- Podemos fazer algumas consultas
  - É verdade que o João utiliza o editor vim?

Consultas

- Podemos fazer consultas sobre os fatos que foram definidos
- Dado os seguintes fatos

```
editor(joao, vim).
editor(pedro, emacs).
```

- Podemos fazer algumas consultas
  - É verdade que o João utiliza o editor vim?

```
?- editor(joao, vim).
true.
```

Consultas

- Podemos fazer consultas sobre os fatos que foram definidos
- Dado os seguintes fatos

```
editor(joao, vim).
editor(pedro, emacs).
```

- Podemos fazer algumas consultas
  - É verdade que o João utiliza o editor vim?

```
?- editor(joao, vim).
true.
```

  - É verdade que o João utiliza o editor emacs?

Consultas

- Podemos fazer consultas sobre os fatos que foram definidos
- Dado os seguintes fatos

```
editor(joao, vim).
editor(pedro, emacs).
```

- Podemos fazer algumas consultas
  - É verdade que o João utiliza o editor vim?

```
?- editor(joao, vim).
true.
```

  - É verdade que o João utiliza o editor emacs?

```
?- editor(joao, emacs).
false.
```

- Observe que a forma de uma consulta é a mesma de um fato

Consultas

- Quando uma consulta é realizada o Prolog faz uma busca sequencial por fatos que unificam com o fato que está sendo consultado
  - Dois fatos unificam se os predicados são os mesmos e cada argumento correspondente é o mesmo
- Se um fato que unifica com a consulta for encontrado, o Prolog irá responder `true`, caso contrário o Prolog responderá `false`
- A resposta `false` significa que não foi encontrado um fato que unifica com a questão

Consultas

- Fatos

```
humano(socrates).
humano(aristoteles).

ateniense(socrates).
```

- Consulta

Consultas

- Fatos

```
humano(socrates).
humano(aristoteles).

ateniense(socrates).
```

- Consulta

```
?- ateniense(aristoteles).
false.
```

- Apesar de poder ser verdade no mundo real que Aristóteles era ateniense (viveu em Atenas), nós não podemos provar isto a partir dos fatos dados

Variáveis

Variáveis

- Para fazer perguntas que as respostas não sejam apenas `true` e `false` usamos variáveis
- Fatos

```
editor(joao, vim).
editor(joao, emacs).
editor(pedro, emacs).
```

- Consulta
  - Existe algum E tal que Pedro utiliza o editor E?

Variáveis

- Para fazer perguntas que as respostas não sejam apenas `true` e `false` usamos variáveis
- Fatos

```
editor(joao, vim).
editor(joao, emacs).
editor(pedro, emacs).
```

- Consulta
  - Existe algum E tal que Pedro utiliza o editor E?

```
?- editor(pedro, E).
E = emacs.
```

- Observe que as variáveis começam com letra maiúscula

Variáveis

- O Prolog realiza uma busca da mesma forma que antes, mas considera que uma variável não instanciada unifica com qualquer objeto
- Quando o Prolog encontra um fato que unifica com a consulta, ele marca o fato e exibe os valores unificados com as variáveis
  - Se o utilizador pressionar a tecla enter, a busca é finalizada
  - Se o utilizador pressionar a tecla ; a busca é reiniciada a partir da marca

Variáveis

- Fatos

```
editor(joao, vim).
editor(joao, emacs).
editor(pedro, emacs).
```

- Consulta
  - Existe algum E tal que João utiliza o editor E?

Variáveis

- Fatos

```
editor(joao, vim).
editor(joao, emacs).
editor(pedro, emacs).
```

- Consulta
  - Existe algum E tal que João utiliza o editor E?

```
?- editor(joao, E).
E = vim ;
E = emacs.
```

Conjunções

Conjunções

- Também é possível fazer consultas mais elaboradas usando conjunções (e)
- Fatos

```
editor(joao, vim).
editor(joao, emacs).
editor(pedro, emacs).
editor(maria, vim).
```

- Consultas
  - João e Pedro utilizam o editor emacs?
  - João utiliza o editor emacs e Pedro utiliza o editor emacs?

Conjunções

- Também é possível fazer consultas mais elaboradas usando conjunções (e)
- Fatos

```
editor(joao, vim).
editor(joao, emacs).
editor(pedro, emacs).
editor(maria, vim).
```

- Consultas
  - João e Pedro utilizam o editor emacs?
  - João utiliza o editor emacs e Pedro utiliza o editor emacs?

```
?- editor(joao, emacs), editor(pedro, emacs).
true.
```

- O símbolo "," é pronunciado "e"

Conjunções

- Quando uma sequência de metas separadas por vírgula é dada para o Prolog, ele tenta satisfazer uma meta por vez
- Todas as metas devem ser satisfeitas para a consulta ser satisfeita

Conjunções

- Fatos

```
editor(joao, vim).
editor(joao, emacs).
editor(pedro, emacs).
editor(maria, vim).
```

- Consulta
  - Existe algum E tal que João e Maria utilizam o editor E?
  - Existe algum E tal que João utiliza o editor E e Maria utiliza o editor E?

Conjunções

- Fatos

```
editor(joao, vim).
editor(joao, emacs).
editor(pedro, emacs).
editor(maria, vim).
```

- Consulta
  - Existe algum E tal que João e Maria utilizam o editor E?
  - Existe algum E tal que João utiliza o editor E e Maria utiliza o editor E?

```
?- editor(joao, E), editor(maria, E).
E = vim ;
false.
```

Conjunções

- Fatos

```
editor(joao, vim).
editor(joao, emacs).
editor(pedro, emacs).
editor(maria, vim).
```

- Consulta
  - Existem X e Y tal que X e Y utilizam o editor emacs?
  - Existem X e Y tal que X utiliza o editor emacs e Y utiliza o editor emacs?

Conjunções

- Fatos

```
editor(joao, vim).
editor(joao, emacs).
editor(pedro, emacs).
editor(maria, vim).
```

- Consulta
  - Existem X e Y tal que X e Y utilizam o editor emacs?
  - Existem X e Y tal que X utiliza o editor emacs e Y utiliza o editor emacs?

```
?- editor(X, emacs), editor(Y, emacs).
X = Y, Y = joao ;
X = joao,
Y = pedro ;
X = pedro,
Y = joao ;
X = Y, Y = pedro.
```

Conjunções

- Fatos

```
editor(joao, vim).
editor(joao, emacs).
editor(pedro, emacs).
editor(maria, vim)
```

- Consulta
  - Existem X e Y tal que X e Y utilizam o editor emacs e o nome de X "vem antes" que o de Y?

Conjunções

- Fatos

```
editor(joao, vim).
editor(joao, emacs).
editor(pedro, emacs).
editor(maria, vim)
```

- Consulta
  - Existem X e Y tal que X e Y utilizam o editor emacs e o nome de X "vem antes" que o de Y?

```
?- editor(X, emacs), editor(Y, emacs), X @< Y.
X = joao,
Y = pedro ;
false.
```

Conjunções

- Fatos

```
editor(joao, vim).
editor(joao, emacs).
editor(pedro, emacs).
editor(maria, vim).
```

- Consulta
  - Existem X, Y e Z tal que X e Y utilizam o editor Z?

Conjunções

- Fatos

```
editor(joao, vim).
editor(joao, emacs).
editor(pedro, emacs).
editor(maria, vim).
```

- Consulta
  - Existem X, Y e Z tal que X e Y utilizam o editor Z?

```
?- editor(X, Z), editor(Y, Z).
```

  - Qual é a resposta?

Regras

Regras

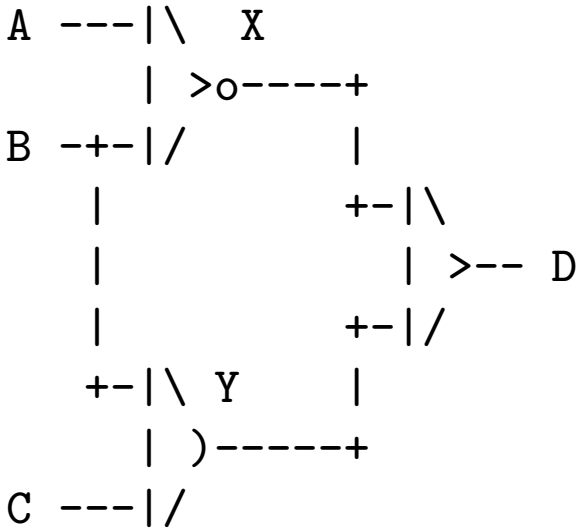
- Forma de abstração utilizada pelo Prolog
- Usamos regras para dizer que um fato depende de um outro grupo de fatos
- Uma **regra** é um sentença genérica sobre objetos e suas relações
- Exemplo: dois programadores podem fazer um par para programação se eles utilizam o mesmo editor

```
par(X, Y) :-
    editor(X, Z),
    editor(Y, Z),
    X @< Y.
```

Exemplos

Exemplo 1.1

Defina um predicado `circuito(A, B, C, D)` que é verdadeiro se as entradas A, B e C produzem a saída D no circuito abaixo.



Exemplo 1.1

```
%% circuito(A?, B?, C?, D?) is nondet
%
%  Verdadeiro se o circuito exemplo com as entradas A, B e C produz
%  a saída D.

circuito(A, B, C, D) :-
    nand(A, B, X),
    or(B, C, Y),
    and(X, Y, D).
```

Exemplo 1.1

```
%% and(A?, B?, C?) is nondet
%
%  Verdadeiro se C = A and B.

and(0, 0, 0).
and(0, 1, 0).
and(1, 0, 0).
and(1, 1, 1).

%% or(A?, B?, C?) is nondet
%
%  Verdadeiro se C = A or B.

or(0, 0, 0).
or(0, 1, 1).
or(1, 0, 1).
or(1, 1, 1).
```

Exemplo 1.1

```
%% not(A?, B?) is nondet
%
%  Verdadeiro se A = not B.

not(0, 1).
not(1, 0).

%% and(A?, B?, C?) is nondet
%
%  Verdadeiro se C = not (A and B).

nand(A, B, C) :-
    and(A, B, S),
    not(S, C).
```

Exemplo 1.1

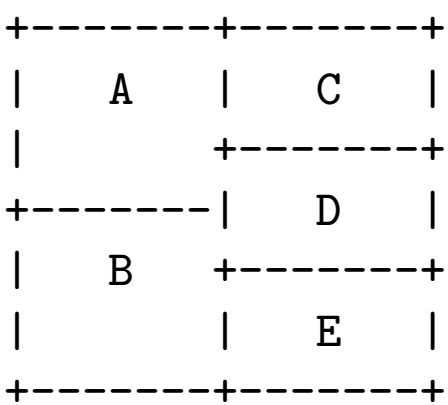
```
?- circuito(1, 0, 1, 1).
true ;
false.

?- circuito(A, B, C, 1).
A = B, B = 0,
C = 1 ;
A = C, C = 0,
B = 1 ;
A = 0,
B = C, C = 1 ;
A = C, C = 1,
B = 0 ;
false.
```

- Inicialmente fizemos o predicado pensando em especificar as entradas do circuito e obter a saída, mas é possível especificar a saída e obter as entradas!

Exemplo 1.2

Defina um predicado `coloracao(A, B, C, D, E)` que é verdadeiro se A, B, C, D, E são cores que podem colorir as respectivas regiões do mapa abaixo de maneira que duas regiões adjacentes não tenham a mesma cor.



Exemplo 1.2

```
%% colocarao(A?, B?, C?, D?, E?) is nondet
%
%  Verdadeiro se A, B, C, D, E são cores que podem colorir as
%  respectivas regiões do mapa exemplo de maneira que duas
%  regiões adjacentes não tenham a mesma cor.

coloracao(A, B, C, D, E) :-
    cor_dif(A, C),
    cor_dif(A, D),
    cor_dif(A, B),
    cor_dif(B, D),
    cor_dif(B, E),
    cor_dif(C, D),
    cor_dif(D, E).
```

Exemplo 1.2

```
%% cor_dif(A?, B?) is nondet
%
%  Verdeiro se A é uma cor diferente da cor B.

cor_dif(A, B) :-
    cor(A),
    cor(B),
    A \== B.

%% cor(A?) is nondet
%
%  Verdadeiro se A é uma cor.

cor(verde).
cor(azul).
cor(amarelo).
```

Exemplo 1.2

```
?- coloracao(A, B, C, D, E).
A = E, E = verde,
B = C, C = azul,
D = amarelo ;
A = E, E = verde,
B = C, C = amarelo,
D = azul
...
```

Leitura recomendada

Leitura recomendada

- The principal programming paradigms
- Declarative programming
- Logic programming
- Prolog