

Inteligência Artificial

Programação em Lógica

PROLOG

- Introdução e Histórico
 - PROgramming in LOGic é fruto de pesquisas na área de Prova Automática de Teoremas.
 - Foi criada por Robert Kowalski (na parte teórica), Maarten van Emden (na demonstração experimental) e Alain Colmerauer (na implementação) por volta de 1970 na Universidade de Marselha, França.
 - O primeiro compilador eficiente foi desenvolvido na Universidade de Edimburgo, Escócia.
 - A linguagem PROLOG também é um provador automático de teoremas, onde a estratégia de cláusulas adotada é a “Selective Linear Resolution for Definite Clauses”.
 - É uma linguagem declarativa, onde se diz “o que fazer” para atingir um objetivo, o que leva a um nível mais elevado de abstração na solução dos problemas.

PROLOG

- Introdução e Histórico
 - Segundo Bratko, “pensar a respeito do problema e aprender a programar em PROLOG constitui-se em um desafio intelectual excitante”.
 - Cada linha de PROLOG corresponde a uma afirmação.
 - A variável compreendida na afirmação deve ser entendida como UNIVERSALMENTE quantificada. Assim, a declaração pai_de(X,Y) corresponde a $\forall X \forall Y \text{ pai_de}(X,Y)$.
 - PROLOG só admite em suas declarações CLÁUSULAS DE HORN.
 - CLÁUSULAS DE HORN só admitem um literal positivo.
 - Lembre-se que $A \rightarrow B$ pode ser escrita sob a forma $\neg A \vee B$.

PROLOG

- Introdução e Histórico
 - Sejam $A_i(x_1, x_2, \dots, x_k)$ e $B_i(x_1, x_2, \dots, x_k)$ fórmulas atômicas, então uma regra do tipo:
Se $A_1(x_1, x_2, \dots, x_k)$ e ... e $A_m(x_1, x_2, \dots, x_k)$
então $B_1(x_1, x_2, \dots, x_k)$ e ... e $B_n(x_1, x_2, \dots, x_k)$
pode ser escrita como
 - $\neg A_1(x_1, x_2, \dots, x_k)$ ou ... ou $\neg A_m(x_1, x_2, \dots, x_k)$ e
 $B_1(x_1, x_2, \dots, x_k)$ ou ... ou $B_n(x_1, x_2, \dots, x_k)$.
 - No entanto, PROLOG só admite declarações do tipo:
 - Se $A_1(x_1, x_2, \dots, x_k)$ e ... e $A_m(x_1, x_2, \dots, x_k)$ então $B_1(x_1, x_2, \dots, x_k)$;
 - $B_1(x_1, x_2, \dots, x_k)$;
 - Se A_1 e ... e A_m então B_1 ; e
 - B_1 .

PROLOG

- Introdução e Histórico
 - Normalmente, variáveis e constantes são diferenciadas pela primeira letra:
 - Símbolos iniciados por minúscula são constantes; e
 - Símbolos iniciados por letra maiúscula são variáveis.
 - O escopo léxico de nomes de variáveis é apenas uma cláusula.
 - Isto quer dizer que, por exemplo, se o nome de variável X25 ocorre em duas cláusulas diferentes, então ela está representando duas variáveis diferentes.
 - Por outro lado, toda ocorrência de X25 dentro da mesma cláusula quer significar a mesma variável.
 - Esta situação é diferente para as constantes: o mesmo nome sempre significa o mesmo objeto ao longo de todo o programa.

PROLOG

- Introdução e Histórico
 - Em PROLOG, as cláusulas são escritas na forma de regras, com a (única) conclusão no início.
 - $B1(X1, \dots, Xk) :- A1(X1, \dots, Xk), \dots, Am(X1, \dots, Xk).$
 - $B1(X1, \dots, Xk).$
 - $B1 :- A1, \dots, Am.$
 - $B1.$
 - O único literal positivo de uma cláusula (que aparece antes do símbolo :-) é chamado cabeça da cláusula.
 - Os literais negativos (que aparecem depois do símbolo :-) são chamados corpo da cláusula.

PROLOG

Exemplo Introdutório

- progenitor(maria,josé). % Maria é progenitor de José.
- progenitor(joão, josé).
- progenitor(joão, ana).
- progenitor(josé, júlia).
- progenitor(josé, íris).
- progenitor(íris, jorge).
- masculino(joão). % João é do sexo masculino.
- masculino(josé).
- masculino(jorge).
- feminino(maria). % Maria é do sexo feminino.
- feminino(ana).
- feminino(júlia).
- feminino(íris).

PROLOG

Exemplo Introdutório

- O efeito das entradas anteriores é o armazenamento destas fórmulas atômicas representando fatos em uma base de conhecimentos PROLOG.
- Se o programa for submetido a um sistema Prolog, este será capaz de responder algumas questões sobre a relação ali representada. Por exemplo: "José é o progenitor de Íris?".
 - ?-progenitor(josé, íris).
- Uma outra questão poderia ser: "Ana é um dos progenitores de Jorge?".
 - ?-progenitor(ana, jorge).

PROLOG

Exemplo Introdutório

- Perguntas mais interessantes podem também ser formuladas, por exemplo: "Quem é progenitor de Íris?"
 - ?-progenitor(X, íris).
- Da mesma forma a questão "Quem são os filhos de José?" pode ser formulada com a introdução de uma variável na posição do argumento correspondente ao filhos de José
 - ?-progenitor(josé, X).
- Uma questão mais geral para o programa seria: "Quem é progenitor de quem?"
 - ?-progenitor(X, Y).

PROLOG

Exemplo Introdutório

- Pode-se formular questões ainda mais complicadas ao programa, como "Quem são os avós de Jorge?". Como nosso programa não possui diretamente a relação avô, esta consulta precisa ser dividida em duas etapas. A saber:
 - (1) Quem é progenitor de Jorge? (Por exemplo, Y) e
 - (2) Quem é progenitor de Y? (Por exemplo, X)
- Esta consulta em Prolog é escrita como uma seqüência de duas consultas simples, cuja leitura pode ser: "Encontre X e Y tais que X é progenitor de Y e Y é progenitor de Jorge".
 - `?-progenitor(X, Y), progenitor(Y, jorge).`
 - `X=josé Y=íris`

PROLOG

Pontos Básicos

- Uma relação como progenitor pode ser facilmente definida em Prolog estabelecendo-se as tuplas de objetos que satisfazem a relação;
- O usuário pode facilmente consultar o sistema Prolog sobre as relações definidas em seu programa;
- Um programa Prolog é constituído de cláusulas, cada uma das quais é encerrada por um ponto (.);
- Os argumentos das relações podem ser objetos concretos (como júlia e íris) ou objetos genéricos (como X e Y). Objetos concretos em um programa são denominados átomos, enquanto que os objetos genéricos são denominados variáveis;

PROLOG

Pontos Básicos

- Consultas ao sistema são constituídas por um ou mais objetivos, cuja seqüência denota a sua conjunção;
- Uma resposta a uma consulta pode ser positiva ou negativa, dependendo se o objetivo correspondente foi alcançado ou não. No primeiro caso dizemos que a consulta foi bem-sucedida e, no segundo, que a consulta falhou;
- Se várias respostas satisfizerem a uma consulta, então o sistema Prolog irá fornecer tantas quantas forem desejadas pelo usuário.

PROLOG

- Exemplo Introdutório
 - A capacidade do PROLOG não se limita à busca em uma base de conhecimentos; é possível armazenar regras.
 - As regras definem as condições que devem ser satisfeitas para que uma certa declaração seja considerada verdadeira.
 - ? -mae(X,Y) :- progenitor(X,Y), feminino(X).
 - ? -pai(X,Y) :- progenitor(X,Y), masculino(X).
 - ? -avo(X,Z) :- progenitor(X,Y), progenitor(Y,Z).
 - Com estas definições podemos obter os seguintes resultados:
 - ? -mae(X,josé).
X = maria;

 - ? -pai(X,íris).
X = josé;

PROLOG

Pontos Básicos

- Programas Prolog podem ser ampliados pela simples adição de novas cláusulas;
- As cláusulas Prolog podem ser de três tipos distintos: fatos, regras e consultas;
- Os fatos declaram coisas que são incondicionalmente verdadeiras;
- As regras declaram coisas que podem ser ou não verdadeiras, dependendo da satisfação das condições dadas;
- Por meio de consultas podemos interrogar o programa acerca de que coisas são verdadeiras;

PROLOG

Pontos Básicos

- As cláusulas Prolog são constituídas por uma cabeça e um corpo. O corpo é uma lista de objetivos separados por vírgulas que devem ser interpretadas como conjunções;
- Fatos são cláusulas que só possuem cabeça, enquanto que as consultas só possuem corpo e as regras possuem cabeça e corpo;
- Ao longo de uma computação, uma variável pode ser substituída por outro objeto. Dizemos então que a variável está instanciada;
- As variáveis são assumidas como universalmente quantificadas nas regras e nos fatos e existencialmente quantificadas nas consultas

PROLOG

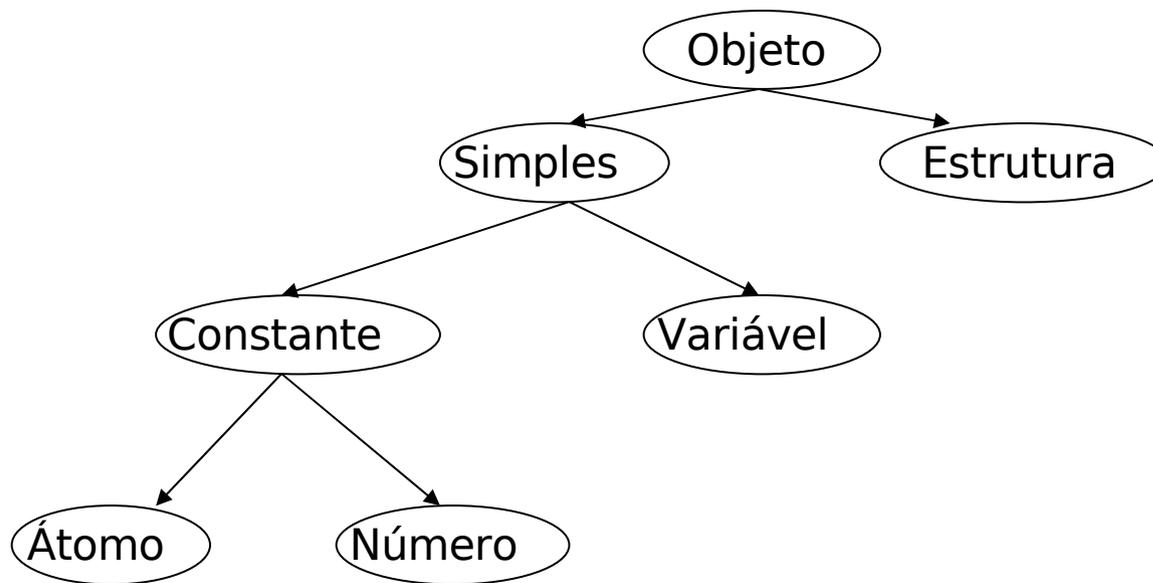
- Exemplo Introdutório
 - Além disso, as definições de regras podem ser recursivas, isto é, uma cláusula de definição de um predicado pode conter este predicado em seu corpo:
 - ? -antepassado(X,Z) :- progenitor(X,Z).
 - ? -antepassado(X,Z) :- progenitor(X,Y),
antepassado(Y,Z).
 - Com estas definições podemos obter os seguintes resultados:
 - ? -antepassado(X,jorge).
 - X = íris;
 - X = maria;
 - X = joão;
 - X = josé;

PROLOG

- Exemplo Introdutório
 - Usa-se o “_” (underscore) para indicar a irrelevância de um objeto
 - ? -aniversario(maria,data(25,janeiro,1979)).
 - ? -aniversario(joao,data(5,janeiro,1956)).
 - ? -signo(Pessoa,aquario) :- aniversario(Pessoa,data(Dia,janeiro,_)), Dia >=20.
 - Com estas definições podemos obter os seguintes resultados:
 - ? -signo(Pessoa,aquario).
- Pessoa = maria;
no
- Usa-se a “,” como operador de conjunção e usa-se o “;” como operador de disjunção (cláusulas começando com o mesmo predicado também indicam a disjunção)
 - ? -avo(X,Z) :- progenitor(X,Y), progenitor(Y,Z).
 - ? -amiga(X) :- (X = maria ; X = joana).

PROLOG

- **Sintaxe**
 - O sistema reconhece o tipo de um objeto no programa por meio de sua forma sintática.
 - Isto é possível porque o PROLOG especifica formas diferentes para cada tipo de objeto.



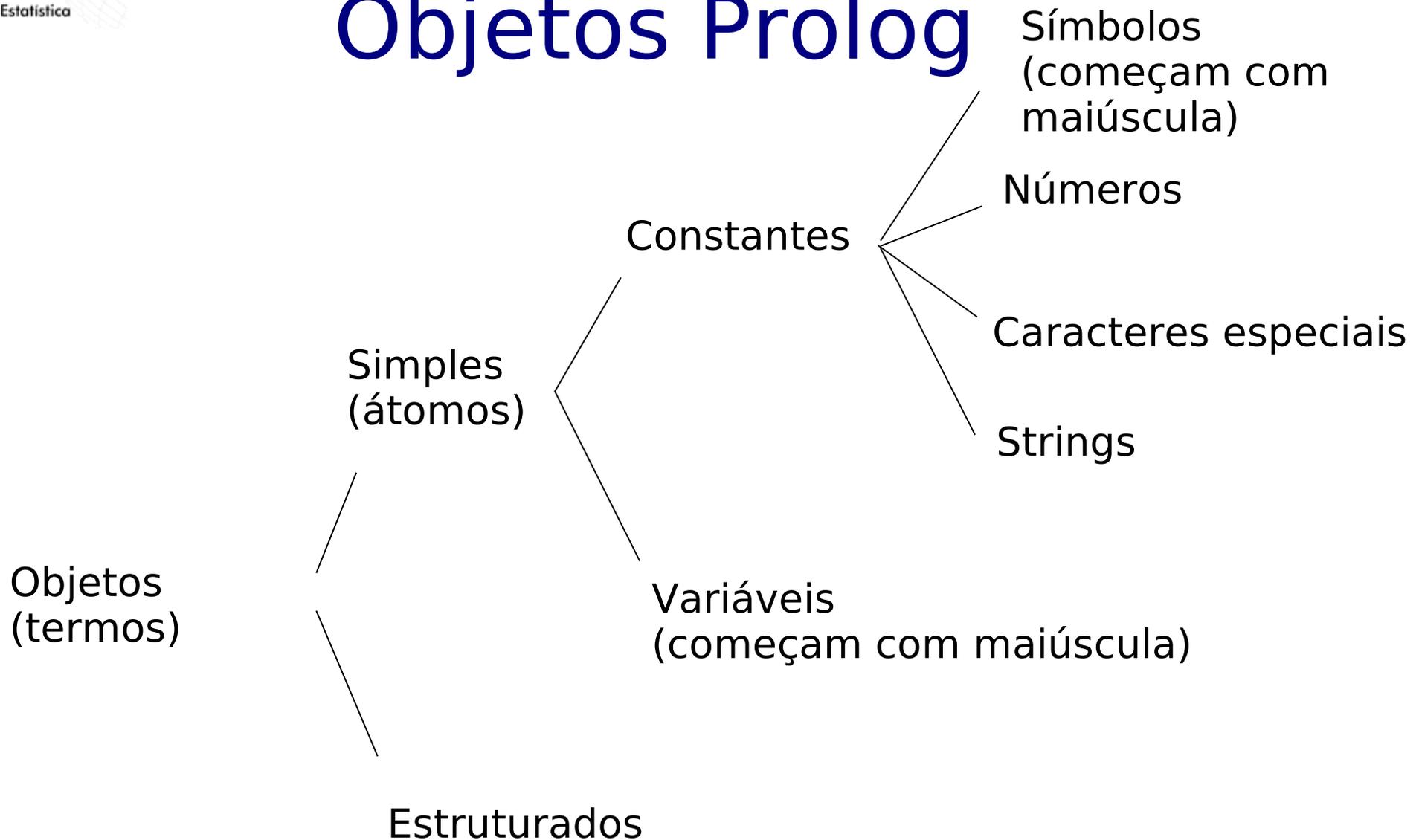
PROLOG

- Sintaxe

- Átomos e Números

- No exemplo introdutório viu-se informalmente alguns exemplos de átomos e variáveis. O alabeto básico adotado consiste dos seguintes símbolos:
 - Pontuação: () . ' "
 - Conectivos: , (conjunção)
 ; (disjunção)
 :- (implicação)
 - Letras: a, b, c, ..., z, A, B, C, ..., Z
 - Dígitos: 0, 1, 2, ..., 9
 - Especiais: + - * / < > = ...

Objetos Prolog



Objetos Prolog

- Símbolos e variáveis:
 - pais → constante
 - Pais → variável
- Termos estruturados:
 - <simbolo funcional>(<arg1>, <arg2>, ..., <argN>)
 - Onde arg1, arg2, etc, podem ser atomos ou termos estruturados
 - Exemplos:
 - pai(joao,jose)
 - pai(X,jose)
 - avo(jorge,filho(joao))

Fatos

- Conjunto de símbolos predicativos que estabelecem uma base de conhecimento relativo a um conjunto de relações entre objetos

pai(joao,jose).

pai(joao,maria).

mae(joana,jose).

automovel(carlos,corcel(vermelho,1980)).



Consulta

- É a maneira de recuperar informações da base de conhecimento

?- pai(joao,jose).

Yes.

Ou:

?- pai(X,jose).

Joao.

Regras

- É o mecanismo utilizado para inferir novos fatos a partir da base de conhecimento.

Pred(arg/Var, arg/Var):-
pred(arg/Var,arg/Var).

- Exemplo:
 - irmao(X,Y):-pai(Z,X),pai(Z,Y).
 - irmao(X,Y):-mae(Z,X),mae(Z,Y).
-
-

PROLOG

- **Sintaxe**
 - **Variáveis**
 - Variáveis PROLOG são cadeias de letras, dígitos e do caracter sublinhado (_), devendo iniciar com este ou com uma letra maiúscula.
 - **Estruturas**
 - Estruturas são objetos que possuem vários componentes.
 - Os próprios componentes, por sua vez, podem também ser estruturas.
 - Para combinar os elementos em uma estrutura é necessário um functor. Um functor é um símbolo funcional (nome de função) que permite agrupar diversos objetos em um único objeto estruturado.
 - `data(13, outubro, 1993)` - dois inteiros e um átomo.
 - `data(Dia, marco, 1996)` - um dia qualquer de marco.

PROLOG

- Sintaxe
 - Sintaticamente todos os objetos em PROLOG são denominados termos.
 - O conjunto de termos PROLOG é o menor conjunto que satisfaz às seguintes condições:
 - Toda constante é um termo;
 - Toda variável é um termo;
 - Se t_1, t_2, \dots, t_n são termos e f é um átomo, então $f(t_1, t_2, \dots, t_n)$ também é um termo, onde o átomo f desempenha o papel de um símbolo funcional n -ário. Diz ainda que a expressão $f(t_1, t_2, \dots, t_n)$ é um termo funcional PROLOG.

PROLOG

Consultas em Prolog

- Uma consulta em Prolog é sempre uma seqüência composta por um ou mais objetivos. Para obter a resposta, o sistema Prolog tenta satisfazer todos os objetivos que compõem a consulta, interpretando-os como uma conjunção. Satisfazer um objetivo significa demonstrar que esse objetivo é verdadeiro, assumindo que as relações que o implicam são verdadeiras no contexto do programa. Se a questão também contém variáveis, o sistema Prolog deverá encontrar ainda os objetos particulares que, atribuídos às variáveis, satisfazem a todos os sub-objetivos propostos na consulta. A particular instanciação das variáveis com os objetos que tornam o objetivo verdadeiro é então apresentada ao usuário. Se não for possível encontrar, no contexto do programa, nenhuma instanciação comum de suas variáveis que permita derivar algum dos sub-objetivos propostos então a resposta será "não".

PROLOG

- Unificação
 - É a operação mais importante entre dois termos PROLOG.
 - Dados dois termos, diz-se que eles se unificam se:
 - Eles são idênticos, ou
 - As variáveis de ambos os termos podem ser instanciadas com objetos de maneira que, após a substituição das variáveis por estes objetos, os termos se tornam idênticos.
 - Exemplo
 - os termos `data(D,M,1994)` e `data(X,marco,A)` unificam. Uma instanciação que torna os dois termos idênticos é:
 - D é instanciada com X;
 - M é instanciada com marco;
 - A é instanciada com 1994.
 - Por outro lado, os termos `data(D,M,1994)` e `data(X,Y,94)` não unificam, assim como não unificam `data(X,Y,Z)` e `ponto(X,Y,Z)`.

PROLOG

- Unificação
 - Se os termos não unificam dizemos, dizemos que o processo FALHA.
 - Se eles unificam, então o processo é bem-sucedido.
 - As regras gerais que determinam se dois termos S e T unificam são:
 - Se S e T são constantes, então S e T unificam somente se ambos representam o mesmo objeto;
 - Se S é uma VARIÁVEL E t É QUALQUER COISA, ENTÃO s E t UNIFICAM COM s INSTANCIADA EM t . Inversamente, se T é uma variável, então T é instanciada em S .
 - Se S e T são estruturas, unificam somente se:
 - S e T tem o mesmo functor principal, e
 - todos os seus componentes correspondentes também unificam. A instanciação resultante é determinada pela unificação dos componentes.

PROLOG

- Consultas em Prolog
 - antepassado(X, Z) :- % X é antepassado de Z se
progenitor(X, Z).% X é progenitor de Z. [pr1]
 - antepassado(X, Z) % X é antepassado de Z se
progenitor(X, Y),% X é progenitor de Y e
antepassado(Y, Z).% Y é antepassado de Z. [pr2]

PROLOG

Consultas em Prolog

- Um exemplo mais complexo:
?-antepassado(joão, íris).
- Sabe-se que progenitor(josé, íris) é um fato. Usando esse fato e a regra [pr1], podemos concluir antepassado(josé, íris). Este é um fato derivado. Não pode ser encontrado explícito no programa, mas pode ser derivado a partir dos fatos e regras ali presentes. Ou seja:
- "de progenitor(josé, íris) segue, pela regra [pr1] que antepassado(josé, íris)".
- Além disso sabemos que progenitor(joão, josé) é fato. Usando este fato e o fato derivado, antepassado(josé, íris), podemos concluir, pela regra [pr2], que o objetivo proposto, antepassado(joão, íris) é verdadeiro.

PROLOG

Consultas em Prolog

- Mostrou-se assim o que pode ser uma seqüência de passos de inferência usada para satisfazer um objetivo. Tal seqüência denomina-se seqüência de prova. A extração de uma seqüência de prova do contexto formado por um programa e uma consulta é obtida pelo sistema na ordem inversa da empregada anteriormente.

PROLOG

Consultas em Prolog

- Ao invés de iniciar a inferência a partir dos fatos, o Prolog começa com os objetivos e , usando as regras, substitui os objetivos correntes por novos objetivos até que estes se tornem fatos.
- Assim, para saber se João é antepassado de Íris, o sistema tenta encontrar uma cláusula no programa a partir da qual o objetivo seja consequência imediata. Obviamente, as únicas cláusulas relevantes para essa finalidade são [pr1] e [pr2], que são sobre a relação antepassado, porque são as únicas cujas cabeças podem ser unificadas com o objetivo formulado.
- Tais cláusulas representam dois caminhos alternativos que o sistema pode seguir. Inicialmente o Prolog irá tentar a que aparece em primeiro lugar no programa:

PROLOG

Consultas em Prolog

- `antepassado(X, Z) :- progenitor(X, Z).`
- Uma vez que o objetivo é `antepassado(joão, íris)`, as variáveis na regra devem ser instanciadas por `X=joão` e `Y=íris`. O objetivo inicial, `antepassado(joão, íris)` é então substituído por um novo objetivo:
`progenitor(joão, íris)`
- Não há, entretanto, nenhuma cláusula no programa cuja cabeça possa ser unificada com `progenitor(joão, íris)`, logo este objetivo falha. Então o Prolog retorna ao objetivo original (backtracking) para tentar um caminho alternativo que permita derivar o objetivo `antepassado(joão, íris)`. A regra [pr2] é então tentada:
- `antepassado(X, Z) :-
 progenitor(X, Y),
 antepassado(Y, Z).`

PROLOG

Consultas em Prolog

- Como anteriormente, as variáveis X e Z são instanciadas para João e Íris, respectivamente. A variável Y, entretanto, não está instanciada ainda. O objetivo original, `antepassado(joão, íris)` é então substituído por dois novos objetivos derivados por meio da regra [pr2]:
- `progenitor(joão, Y), antepassado(Y, íris)`.
- Encontrando-se agora face a dois objetivos, o sistema tenta satisfazê-los na ordem em que estão formulados. O primeiro deles é fácil: `progenitor(joão, Y)` pode ser unificado com dois fatos do programa: `progenitor(joão, José)` e `progenitor(joão, Ana)`. Mais uma vez, o caminho a ser tentado deve corresponder à ordem em que os fatos estão escritos no programa. A variável Y é então instanciada com José nos dois objetivos acima, ficando o primeiro deles imediatamente satisfeito.

PROLOG

Consultas em Prolog

- O objetivo remanescente é então:
- antepassado(josé, íris).
- Para satisfazer tal objetivo, a regra [pr1] é mais uma vez empregada.
- Essa segunda aplicação de [pr1], entretanto, nada tem a ver com a sua utilização anterior, isto é, o sistema Prolog usa um novo conjunto de variáveis na regra cada vez que esta é aplicada.

PROLOG

Consultas em Prolog

- A cabeça da regra deve então ser unificada como o nosso objetivo corrente, que é antepassado(josé, íris). A instanciação de X' e Y' fica: $X'=josé$ e $Y'=íris$ e o objetivo corrente é substituído por:
- progenitor(josé, íris)
- Esse objetivo é imediatamente satisfeito, porque aparece no programa como um fato. O sistema encontrou então um caminho que lhe permite provar, no contexto oferecido pelo programa dado, o objetivo originalmente formulado, e portanto responde "sim".

PROLOG

- Semântica
 - PROLOG tem três semânticas: a DECLARATIVA, a PROCEDURAL e a OPERACIONAL.
 - A semântica DECLARATIVA é aquela em que se escreve e lê o programa PROLOG e que é uma representação do que se conhece da definição do problema a resolver.
 - A semântica declarativa pode ser interpretada de três modos distintos:
 - Para resolver um problema dá-se uma nova cláusula, a pergunta, e o PROLOG tenta verificar se esta cláusula é compatível com o mundo definido;
 - Considera-se que os literais na cabeça e cauda de cada cláusula são objetivos a serem atingidos. Uma pergunta tem resposta afirmativa se o objetivo que ela define é satisfeito usando as regras do programa;
 - Olha as cláusulas como regras de uma gramática, onde cada regra de PROLOG corresponde a uma regra da gramática.

PROLOG

- Semântica
 - Exemplo
 - Seja $P :- Q, R$
onde P, Q e R possuem a sintaxe de termos PROLOG. Duas alternativas para a leitura declarativa destas cláusulas são:
 - P é verdadeira se Q e R são verdadeiras, e
 - De Q e R , segue P .
 - A semântica declarativa determina se um dado objetivo é verdadeiro e, se for, para que valores de variáveis isto se verifica.
 - Assim, dado um programa e um objetivo G , o significado declarativo nos diz que:
 - Um objetivo G é verdadeiro (isto é, é satisfável ou segue logicamente do programa) se e somente se há uma cláusula C no programa e uma instância I de C tal que:
 - A cabeça de I é idêntica a G , e
 - Todos os objetivos no corpo de I são verdadeiros.

PROLOG

- Semântica
 - A semântica PROCEDURAL define não apenas o relacionamento lógico existente entre a cabeça e o corpo da cláusula, como também exige a existência de uma ordem na qual os objetivos serão processados.
 - Exemplo
 - Seja $P :- Q, R$
onde P, Q e R possuem a sintaxe de termos PROLOG. Duas alternativas para a leitura procedural destas cláusulas são:
 - Para solucionar o problema P
primeiro solucione o subproblema Q
e depois solucione o subproblema R.
 - Para satisfazer P, primeiro satisfaça Q e depois R.

PROLOG

Semântica

- A semântica OPERACIONAL define como PROLOG responde a uma pergunta.
- Seria ótimo se não fosse necessário conhecer esta semântica, pois neste caso, PROLOG seria realmente uma implementação do paradigma de programação em lógica. Entretanto, este não é o caso.
- PROLOG pesquisa se a pergunta é verdadeira ou falsa construindo a árvore de possibilidades para trás e faz a busca em profundidade. Se a árvore a percorrer é muito grande, o tempo pode se tornar proibitivo e é conveniente restringir o espaço de busca o mais possível, o que só é possível sabendo como o PROLOG vai visitar os nós da árvore.
- O modo como PROLOG vai visitar os nós da árvore varia se as declarações são apresentadas em ordem diferente. Consequentemente, PROLOG não é realmente uma linguagem declarativa.

- Semântica

PROLOG

PROLOG

- Semântica
 - Suas entradas e saídas são:
 - entrada: um programa e uma lista de objetivos;
 - saída: um indicador de sucesso/falha e instanciações de variáveis.
 - O significado dos resultados de saída do *executor* é o seguinte:
 - O indicador de *sucesso/falha* tem o valor "sim" se os objetivos forem todos satisfeitos e "não" em caso contrário;
 - As *instanciações* são produzidas somente no caso de conclusão bem-sucedida e correspondem aos valores das variáveis que satisfazem os objetivos.

PROLOG

- Semântica (Resumo)
 - A interpretação declarativa de programas escritos em Prolog puro não depende da ordem das cláusulas nem da ordem dos objetivos dentro das cláusulas;
 - A interpretação procedimental depende da ordem dos objetivos e cláusulas. Assim a ordem pode afetar a eficiência de um programa. Uma ordenação inadequada pode mesmo conduzir a chamadas recursivas infinitas;
 - A semântica operacional representa um procedimento para satisfazer a lista de objetivos no contexto de um dado programa. A saída desse procedimento é o valor-verdade da lista de objetivos com a respectiva instanciação de suas variáveis. O procedimento permite o retorno automático (backtracking) para o exame de novas alternativas;

Semântica do prolog

- Semântica declarativa
 - $P:-Q,R$
 - P é verdadeiro se Q é verdadeiro e R é verdadeiro
 - Ou: de Q e R , segue P
- Semântica procedimental
 - Para solucionar P , primeiro selecione Q , depois R

Exemplo

nasceu_em(joao,porto_alegre).

nasceu_em(jean,paris).

fica_em(porto_alegre,brasil).

fica_em(paris,franca).

patria_de(X,Y):-nasceu_em(X,Z),fica_em(Z,Y).

?-patria_de(joao,brasil).

yes.

?-patria_de(A,B).

A=joao B=brasil;

A=jean B=franca;

no.

PROLOG

- Backtracking

- Na execução dos programas Prolog, a evolução da busca por soluções assume a forma de uma árvore - denominada "árvore de pesquisa" ou "search tree" - que é percorrida sistematicamente de cima para baixo (top-down) e da esquerda para direita, segundo o método denominado "depth-first search" ou "pesquisa primeiro em profundidade".

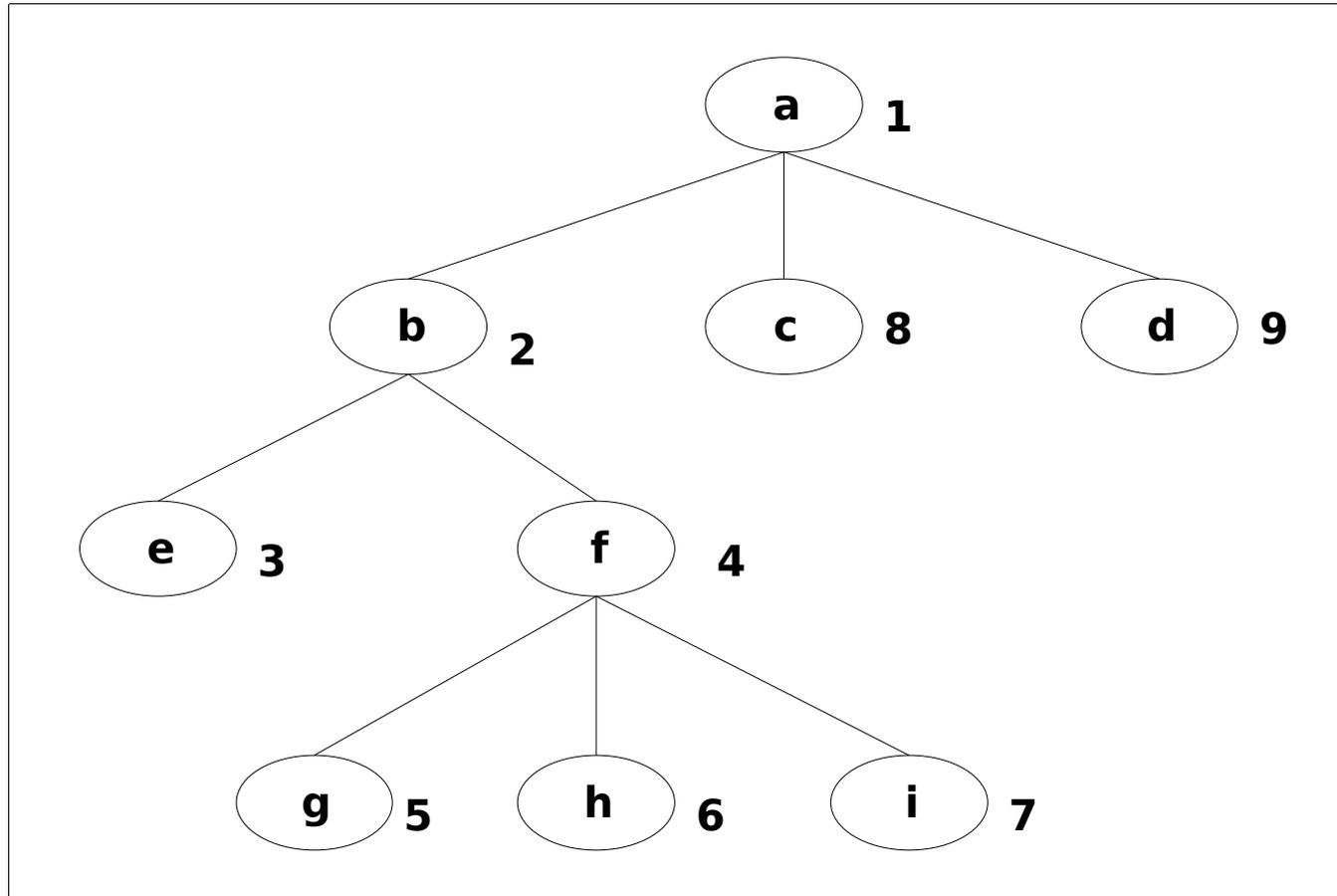
Exemplo

- Sejam a,b,c,etc... termos PROLOG

```
a :- b.  
a :- c.  
a :- d.  
b :- e.  
b :- f.  
f :- g.  
f :- h.  
f :- i.  
d.
```

PROLOG

- Backtracking
 - Ordem de visita aos nodos da árvore



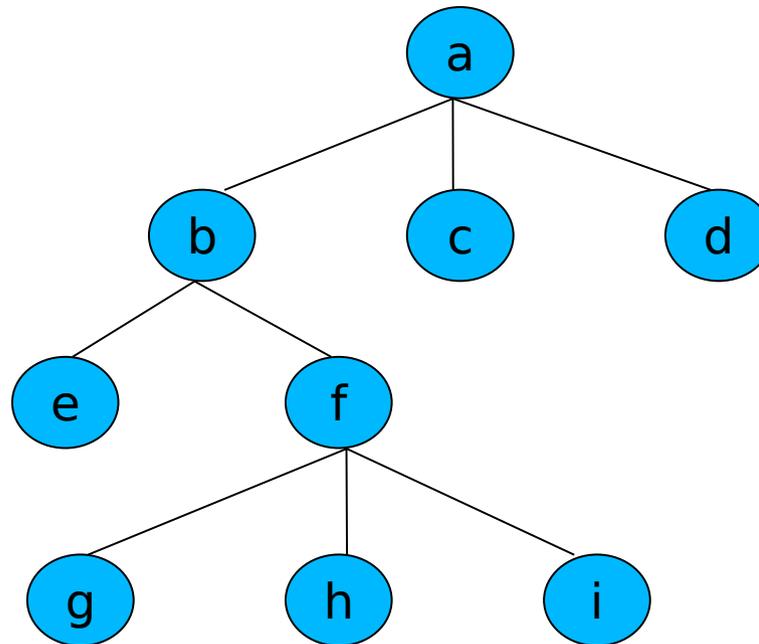
a, b, e, (b), f, g, (f), h, (f), i, (f), (b), (a), c, (a), d
 onde o caminho em backtracking é representado entre parênteses

Controle de backtracking

- Ordem de execução das cláusulas:
 - De cima para baixo, da esquerda para a direita.
- Backtracking:
 - Tentativa de resolver uma cláusula novamente por outra alternativa, em caso de falha.
 - Ordem de execução:
 - Da esquerda para a direita e de cima para baixo.

Backtracking

- a:-b.
- a:-c.
- a:-d.
- b:-e.
- b:-f.
- f:-g
- f:-h.
- f:-i.
- D.



Exemplo de aplicação

- Se $X < 3$, $Y = 0$
 - Se $3 \leq X < 6$, $Y = 2$
 - Se $X \geq 6$, $Y = 4$
 - Implementação trivial:
 - $f(X,0):- X < 3.$
 - $f(X,2):- 3 \leq X, X < 6.$
 - $f(X,4):- 6 \leq X.$
 - Testar com a consulta
?-f(1,Y),2 < Y.
-
-

PROLOG

- Backtracking
 - Como foi visto, os objetivos em um programa Prolog podem ser bem-sucedidos ou falhar.
 - Para um objetivo ser bem-sucedido ele deve ser unificado com a cabeça de uma cláusula do programa e todos os objetivos no corpo desta cláusula devem também ser bem-sucedidos. Se tais condições não ocorrerem, então o objetivo falha.
 - Quando um objetivo falha, em um nodo terminal da árvore de pesquisa, o sistema Prolog aciona o mecanismo de backtracking, retornando pelo mesmo caminho percorrido, na tentativa de encontrar soluções alternativas.
 - Ao voltar pelo caminho já percorrido, todo o trabalho executado é desfeito.

PROLOG

- Backtracking

Exemplo 2

- gosta(joão, jazz).
 - gosta(joão, renata).
 - gosta(joão, lasanha).
 - gosta(renata, joão).
 - gosta(renata, lasanha).
- queremos saber do que ambos, joão e renata, gostam. Isto pode ser formulado pelos objetivos:
- gosta(joão, X), gosta(renata, X).
1. Encontra que joão gosta de jazz
 2. Instancia X com "jazz"
 3. Tenta satisfazer o segundo objetivo, determinando se "renata gosta de jazz"
 4. Falha, porque não consegue determinar se renata gosta de jazz
 5. Realiza um backtracking na repetição da tentativa de satisfazer `gosta(joão, X)`, esquecendo o valor "jazz"

PROLOG

- Backtracking

Exemplo 2

6. Encontra que João gosta de Renata
7. Instancia X com "Renata"
8. Tenta satisfazer o segundo objetivo determinando se "Renata gosta de Renata"
9. Falha porque não consegue demonstrar que Renata gosta de Renata
10. Realiza um backtracking, mais uma vez tentando satisfazer $\text{gosta}(\text{João}, X)$, esquecendo o valor "Renata"
11. Encontra que João gosta de Lasanha
12. Instancia X com "Lasanha"
13. Encontra que "Renata gosta de Lasanha"
14. É bem-sucedido, com X instanciado com "Lasanha"

PROLOG

- Impurezas de PROLOG
 - O backtracking automático é uma ferramenta muito poderosa e a sua exploração é de grande utilidade para o programador. Às vezes, entretanto, ele pode se transformar em fonte de ineficiência. A seguir se introduzirá um mecanismo para "podar" a árvore de pesquisa, evitando o backtracking quando este for indesejável.
 - Para aumentar a eficiência no percurso da árvore de busca da solução do problema usam-se essencialmente dois operadores: CORTE ("CUT" representado por !) e FALHA ("FAIL").
 - Seu uso deve ser considerado pelas seguintes razões:
 - O programa irá executar mais rapidamente, porque não irá desperdiçar tempo tentando satisfazer objetivos que não irão contribuir para a solução desejada.
 - (ii) Também a memória será economizada, uma vez que determinados pontos de backtracking não necessitam ser armazenados para exame posterior.

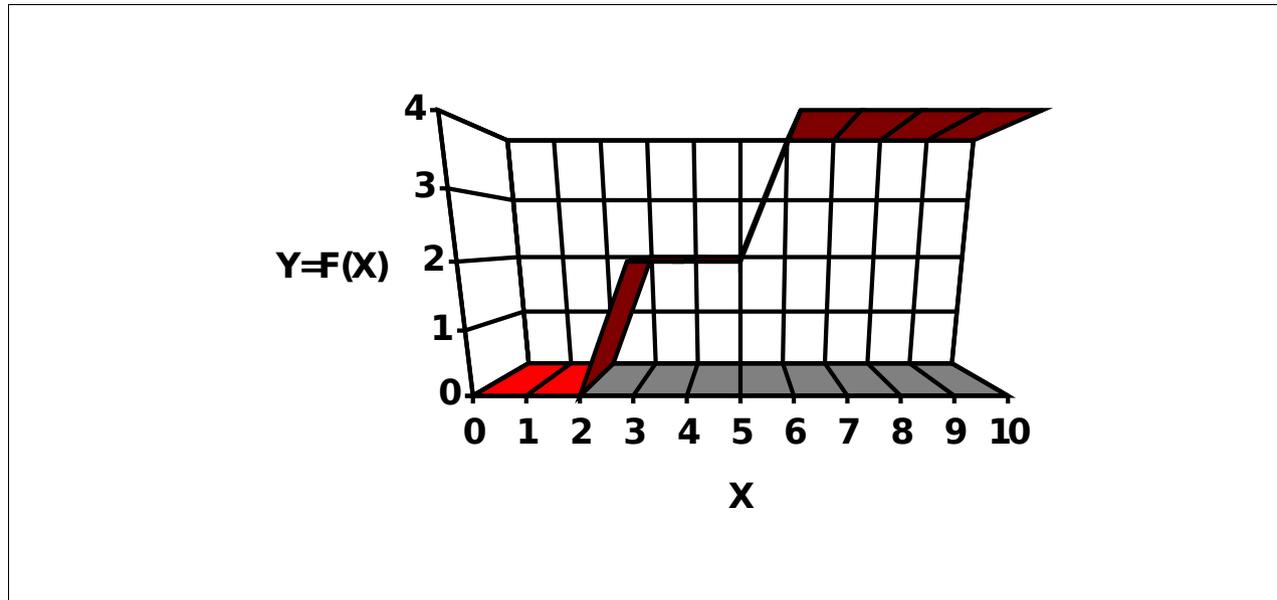
PROLOG

- CUT

- Algumas das principais aplicações do cut são as seguintes:
 - Unificação de padrões, de forma que quando um padrão é encontrado os outros padrões possíveis são descartados
 - Na implementação da negação como regra de falha
 - Para eliminar da árvore de pesquisa soluções alternativas quando uma só é suficiente
 - Para encerrar a pesquisa quando a continuação iria conduzir a uma pesquisa infinita, etc

PROLOG

- CUT
Exemplo



(1) Se $X < 3$, então $Y = 0$

(2) Se $3 \geq X$ e $X < 6$, então $Y = 2$

(3) Se $6 \geq X$, então $Y = 4$

que podem ser escritas em Prolog como uma relação binária $f(X, Y)$,
como se segue:

$f(X, 0) :- X < 3.$

$f(X, 2) :- 3 \leq X, X < 6.$

$f(X, 4) :- 6 \leq X.$

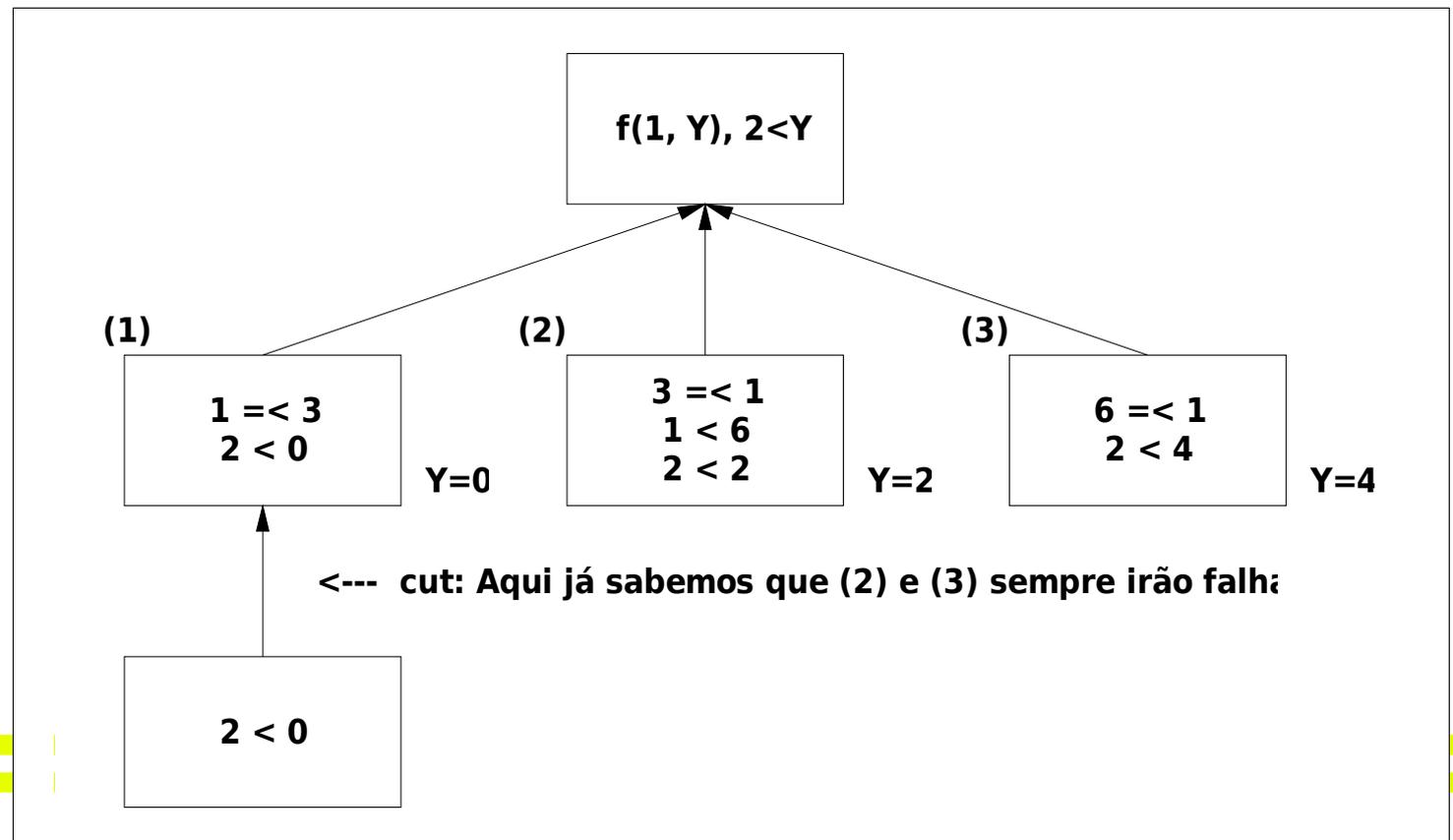
PROLOG

- **CUT**

Exemplo

– Vamos analisar o que ocorre quando a seguinte questão é formulada:

- $?-f(1, Y), 2 < Y$



PROLOG

- CUT
Exemplo
 - O programa do exemplo, reescrito com cuts assume o seguinte aspecto:
 - $f(X, 0) \text{ :- } X < 3, !.$
 - $f(X, 2) \text{ } 3 \leq X, X < 6, !.$
 - $f(X, 4) \text{ } 6 \leq X.$
- Aqui o símbolo "!" evita o backtracking nos pontos em que aparece no programa.

PROLOG

- FALHA (FAIL)

- Negação por Falha

- “Maria gosta de todos os animais, menos de cobras”. Como podemos dizer isto em Prolog? É fácil expressar uma parte dessa declaração: Maria gosta de X se X é um animal, isto é:

- $\text{gosta}(\text{maria}, X) \text{ :- animal}(X)$.

- mas é necessário ainda excluir as cobras. Isto pode ser conseguido empregando-se uma formulação diferente:

- Se X é uma cobra,

- então não é verdade que maria gosta de X

- senão se X é um animal, então maria gosta de X.

PROLOG

- FALHA (FAIL)

- Podemos dizer que alguma coisa não é verdadeira em Prolog por meio de um predicado pré-definido especial, "fail", que sempre falha, forçando o objetivo pai a falhar. A formulação acima pode ser dada em Prolog com o uso do fail da seguinte maneira:
 - `gosta(maria, X) :- cobra(X), !, fail.`
 - `gosta(maria, X) :- animal(X).`
- Aqui a primeira regra se encarrega das cobras. Se X é uma cobra, então o cut evita o backtracking (assim excluindo a segunda regra) e o fail irá ocasionar a falha da cláusula. As duas regras podem ser escritas de modo mais compacto como uma única cláusula, por meio do uso do conetivo " ; " :
 - `gosta(maria, X) :- cobra(X), !, fail; animal(X).`

Unificação

- É o mecanismo de ligação entre variáveis e objetos dentro de uma cláusula, de acordo com o seguinte:
 - Simbolos funcionais iguais
 - Mesmo numero de argumentos
 - Argumentos constantes iguais
 - Exemplos:
 - $\text{pai}(\text{joao}, \text{jose}) :- \text{pai}(X, \text{jose}) \rightarrow X/\text{joao}$
 - $\text{automovel}(\text{carlos}, \text{corsesel}(1992, \text{vermelho})) :- \text{automovel}(X, \text{corsesel}(1992, Y)) \rightarrow X/\text{carlos} \quad Y/\text{vermelho}$

Exercício

- Verifique se são unificáveis os seguintes pares de termos. Nos casos em que a unificação for possível, indique as substituições necessárias.
 - i) `funcionário(123, josé):- funcionário(123,X)`
 - ii) `livro(lusíadas,autor(camões)) :-
livro(camões,autor(lusíadas))`
 - iii) `tem(josé, livro(lusíadas, autor(camões))) :-
tem(X, livro(Y, autor(Z)))`
 - iv) `pertence(veículo(auto,vw,1978),antônio) :-
pertencec(X,Y)`
 - v) `f(f(X,Y), f(Y,X)) :- f(f(2,3),Z)`
 - **vi) `f(f(X,X), f(Y,X)) :- f(f(2,3),Z)`**

Exercicio

- Verifique se são unificáveis os seguintes pares de termos. Nos casos em que a unificação for possível, indique as substituições necessárias.
 - vii) $s(s(s(s(0)))) :- s(s(X))$
 - viii) $f(g(X), h(f(g(x)))) :- f(g(4), h(Y))$
 - ix) $f(g(X), h(f(X))) :- f(g(4), g(Y))$
 - x) $disc(info5001, prog_em_lógica) :- disc(info5001, prog_em_lógica, ano(2002))$
 - xi) $result(3) :- result(soma(1,2))$
 - xii) $tem(josé, avião) :- possui(josé, avião)$
-
-

Operadores

- Aritméticos

+ -
* /
mod is

- Lógicos

> <
>= <=
:= \=

- Exemplo:

```
fatorial(0,1).  
fatorial(X,Y):-  
    X1 is X-1,  
    fatorial(X1,Y1),  
    Y is X*Y1.
```



Listas

- Representação de listas

lista([a,b,c,d,e,f]).

lista([X|Y]).

X=a Y=[bcde].

Lista vazia: []

- Exemplos

- Verificar se um elemento é membro de uma lista:

- Member(X, [X|_]).

- member(X, [_|T]) :- member(X, T).



Operações com listas

```
member(H, [H|_]).  
member(H, [_|T]) :-  
member(H, T).
```

```
member_check( H, [H|_] ) :- !.  
member_check(H, [_|T]) :-  
member_check(H, T).
```

```
append([],L,L).  
append([H|T], L, [H|R]) :-  
append(T, L, R).
```

Operações com listas

```
intersection([], _, []) :-!.
```

```
intersection([H|L1], L2, [H|L3]) :-  
member(H, L2),
```

```
intersection(L1,L2,L3), !.
```

```
intersection([_|L1],L2,L3) :-  
intersection(L1,L2,L3).
```

```
subset([], _).
```

```
subset([A|X], Y) :-
```

```
member(A, Y),
```

```
subset(X, Y).
```



Estruturas de dados

- As estruturas de dados ocorrem na forma:

`pred(arg, estrut(atrib1,...,atrib n)).`

ou

`pred(estrut(atrib1,...,atrib n), arg)`

- As estruturas servem para aninhar dados de forma organizada, estruturada. Elas permitem a distinção de atributos para um argumento. Por exemplos:

`ficha(num,func(nome,end,cargo)).`

`ficha(num,func(dados_pes(nome,end),cargo(funcao,salario(basico,descontos,vantagens)))).`

Prolog procedural

- Cut e fail
- Built in predicates
- Metalogic predicates



Usando o operador CUT (!)

- $f(X,0):- X<3,!.$
 - $f(X,2):-3=<X,X<6,!.$
 - $f(X,4):-6=<X.$
 - Ou ainda:
 - $f(X,0):- X<3,!.$
 - $f(X,2):-X<6,!.$
 - $f(X,4).$
 - (if, them else)
-
-

Outros exemplos de cut

- Computando o maior entre dois números:

```
max(X,Y,X):- X>=Y.
```

```
max(X,Y,Y):- X<Y.
```

```
?- max(X,Y,Max).
```

- Usando cut:

```
max(X,Y,X):- X>=Y,!.
```

```
max(X,Y,Y).
```



Outros exemplos de cut

- Pertencimento a uma lista

```
member(H, [H|_]).  
member(H, [_|T]) :-  
member(H, T).
```

```
member_check( H, [H|_] ) :- !.  
member_check(H, [_|T]) :-  
member_check(H, T).
```

- Inserindo sem repetição:

```
add(X,L,L):- member(X,L),!.  
add(X,L,[X|L]).
```

Combinação cut - fail

- fail é um predicado interno do Prolog que força um retrocesso na avaliação das regras, como se indicasse uma falha.
- Exemplo: mary gosta de animais mas não de cobras.
- Em prolog:

```
likes(mary, X) :-snake(X),!,fail.
```

```
likes(mary, X) :-animal(X).
```

Combinação cut - fail

- Exemplo: definição de diferença entre dois termos
- Em prolog:

```
different(X,X) :- !,fail.  
different(X,Y).
```
- Ou:

```
different(X,Y):- X=Y,! ,fail;  
true.
```
- true é um predicado interno do prolog
que é o oposto do fail.

Predicado not

not(P):-

P,!,fail;

true.

Algumas implementações implementam o operador not como um predicado interno.

likes(mary, X):-

animal(X),

not snake(x).

different(X,Y):- not (X=Y).

predicados de entrada e saída em PROLOG

- see(Filename) muda o input stream corrente
- tell(Filename) muda o output stream corrente
- o dispositivo de entrada e saída padrão é o teclado e o monitor, referido como user os predicados seen e told fecham os arquivos correntes de entrada e saída respectivamente. predicados para leitura e escrita de caractere no arquivo corrente são
- put(C),
- get(C) e
- get0(C) onde C é um inteiro que representa o código ASC do caractere
- predicados para leitura e escrita de termos PROLOG em arquivos são
- % read(X) e write(X) onde X teve ter a sintaxe valida para termos os predicados
- tab(N) e nl são para controle de saída: tabulação de N espaços e inserção de nova linha, respectivamente.

```
cube:- write('Next item, please: '),
       read(X),
       process(X).
process(stop):-
    !.
process(N):-
    C is N*N*N,
    write('Cube of '),
    write(N),
    write(' is '),
    write(C),
    nl,
    cube.
```

Predicados metalógicos

functor(Term,T,Arg)

exemplos:

functor(filho(X,Y),filho,2)

functor(T, pai,2) -> T=pai(X,Y)

arg(N,Term,Arg)

exemplos:

arg(1,pai(joao,jose),joao)

arg(2,pai(joao,jose),X) -> X = jose

var(X)

atom(X)

integer(X)

real(X) ou float()

- Manipulação de termos

- A soma dos números X, Y é Z

`soma1(X,Y,Z):-integer(X),integer(Y), Z is X
+Y.`

`soma(X,Y,Z):- novar(X),novar(Y), Z is X+Y.`

`soma(X,Y,Z):- novar(X),novar(Z), Y is Z-X.`

`soma(X,Y,Z):- novar(Y),novar(Z), X is Z-Y.`



o novo termo é o resultado da troca de todas as ocorrências de Velho, no termo original, por Novo

```
substitui(Velho, Novo, Velho, Novo).
substitui(Velho, Novo, Termo, Termo1):-
atom(Termo),
Termo\=Termo1.
substitui(Velho, Novo, Termo, Termo1):-
compound(Termo),
functor(Termo, T, N),
functor(Termo1, T, N),
substitui(N, Velho, Novo, Termo, Termo1).
substitui(N, Velho, Novo, Termo, Termo1):-
N>0,
arg(N, Termo, Arg),
substitui(Velho, Novo, Arg, Arg1),
arg(N, Termo1, Arg1),
N1 is N-1,
substitui(N1, Velho, Arg1, Termo, Termo1),
N1 is N+1,
substitui(N1, Velho, Novo, Termo, Termo1).
substitui(0, Velho, Novo, Termo, Termo1).
```

Predicados metalógicos

compound(X)

exemplo

compound(pai(X,Y))

a(X,f(b))

length(List,N)

exemplo:

length([1,2,3,4],N) -> N=4

ground = unificado
