

Programação Lógica

Jomi F. Hübner

Universidade Federal de Santa Catarina
Departamento de Automação e Sistemas
<https://jomifred.github.io>



Conteúdo

- princípios & motivação
- cláusulas de Horn
- linguagem Prolog
- unificação & inferência & backtracking
- exemplos

Motivações

Motivações

- adequação da linguagem ao **problema**
- programação **declarativa**
- o programador estabelece a **teoria**
 - o interpretador é um **provador** automático de teoremas

Declarativo vs Imperativo

Declarativo vs Imperativo

Java

```
boolean member(int e, List<Integer> l) {  
    for (int i: l)  
        if (i == e)  
            return true;  
    return false;  
}
```

Declarativo vs Imperativo

Java

```
boolean member(int e, List<Integer> l) {  
    for (int i: l)  
        if (i == e)  
            return true;  
    return false;  
}
```

Prolog

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

Declarativo vs Imperativo

Declarativo vs Imperativo

Java

```
int max(List<Integer> l) {  
    m = l[0];  
    for (int i: l)  
        if (i > m)  
            m = i;  
    return m;  
}
```

Declarativo vs Imperativo

Java

```
int max(List<Integer> l) {  
    m = l[0];  
    for (int i: l)  
        if (i >  
            m = i;  
    return m;  
}
```

Prolog

```
max(V,L) :-  
    member(V,L),  
    not (member(V2,L), V \== V2, V2 > V).
```

o max é aquele que não tem um valor maior

Alto nível de abstração

Alto nível de abstração

```
% fatos (relação entre objetos)
```

```
irmao(bob,alice).
```

```
irmao(alice,tom).
```

```
irmao(tom,ze).
```

```
% propriedades da relação
```

```
 $\forall x,y,z \text{ irmao}(x,y) \wedge \text{irmao}(y,z) \Rightarrow \text{irmao}(x,z)$ 
```

```
irmao(X,Z) :- irmao(X,Y), irmao(Y,Z).
```

Alto nível de abstração

```
% fatos (relação entre objetos)
```

```
irmao(bob,alice).
```

```
irmao(alice,tom).
```

```
irmao(tom,ze).
```

```
% propriedades da relação
```

```
 $\forall x,y,z \text{ irmao}(x,y) \wedge \text{irmao}(y,z) \Rightarrow \text{irmao}(x,z)$ 
```

```
irmao(X,Z) :- irmao(X,Y), irmao(Y,Z).
```

Consulta

```
?- irmao(bob,ze).
```

```
yes
```

Alto nível de abstração

```
% fatos (relação entre objetos)
```

```
irmao(bob,alice).
```

```
irmao(alice,tom).
```

```
irmao(tom,ze).
```

```
% propriedades da relação
```

```
 $\forall x,y,z \text{ irmao}(x,y) \wedge \text{irmao}(y,z) \Rightarrow \text{irmao}(x,z)$ 
```

```
irmao(X,Z) :- irmao(X,Y), irmao(Y,Z).
```

Consulta

```
?- irmao(bob,ze).
```

yes

Consulta

```
?- irmao(bob,ana).
```

ERROR: Out of local stack

Ideal x Real

- lógica proposicional não é muito expressiva
- lógica de predicados precisa de métodos **eficientes** de prova
- solução
 - limitar a capacidade de **expressão** da linguagem
 - usar **resolução** SLD

Cláusulas de Horn

- cláusula
 - $P \vee Q \vee \neg R \vee \neg S$
- cláusulas de Horn tem só um literal positivo
 - $Q \vee \neg R \vee \neg S \equiv R \wedge S \Rightarrow Q$

SLD-Resolution

(Selective Linear Definite clause resolution)

- fórmulas são cláusulas de **Horn**
- inicia **refutando**
- **resolução** entre
 - o último resolvente (**linear**) e
 - o literal positivo de uma das cláusulas (**selected**)
- correto e completo

1	P	
2	$R \vee \neg P$	
3	$\neg R \vee S$	
<hr/>		
4	$\neg S$	
<hr/>		
5	$\neg R$	R, 4, 3
6	$\neg P$	R, 5, 2
7	\perp	R, 6, 1
8	S	

Prolog

Sintaxe

- Fatos (cláusulas sem literal negativo)
 - `feliz(marcos).`
 - `amigo(marcos,aline).`
- Regras (cláusulas)
 - `irmao(marcos,joao) :-`
 `pai(pedro,marcos),`
 `pai(pedro,joao).`

Sintaxe

predicados

constantes

- Fatos (cláusulas sem literal negativo)
 - `feliz(marcos).`
 - `amigo(marcos,aline).`
- Regras (cláusulas)
 - `irmao(marcos,joao) :-`
 `pai(pedro,marcos),`
 `pai(pedro,joao).`

operadores

Sintaxe

- Fatos (cláusulas sem literal negativo)

- `feliz(marcos).`

- `amigo(marcos,aline).`

cabeça

- Regras (cláusulas)

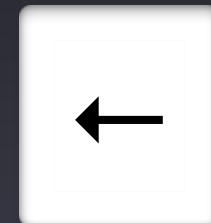
- `irmao(marcos,joao) :-`

`pai(pedro,marcos),
pai(pedro,joao).`

corpo

Sintaxe

- Fatos (cláusulas sem literal negativo)
 - `feliz(marcos).`
 - `amigo(marcos,aline).`
- Regras (cláusulas)
 - `irmao(marcos,joao) :-`
 `pai(pedro,marcos),`
 `pai(pedro,joao).`



Variáveis

- Regras (cláusulas)
 - `irmao(X,Y) :-`
 `pai(Z,X),`
 `pai(Z,Y).`

Unificação

- verificação de “**igualdade**” entre termos
 - sendo s e t termos e θ uma substituição
 - se $s\theta = t\theta$, então θ é uma unificação para s e t

Unificação

- verificação de “**igualdade**” entre termos
 - sendo s e t termos e θ uma substituição
 - se $s\theta = t\theta$, então θ é uma unificação para s e t
- identifica qual o valor das variáveis para que dois literais sejam iguais
 - $p(a) = p(X)$
 - $p(1, b(u, o), a) = p(1, X, Y)$
 - $p(A, 2) = p(3, B)$
 - $p(X, g(Y)) = p(a, g(X))$

Unificação

- verificação de “**igualdade**” entre termos
 - sendo s e t termos e θ uma substituição
 - se $s\theta = t\theta$, então θ é uma unificação para s e t
- identifica qual o valor das variáveis para que dois literais sejam iguais
 - $p(a) = p(X)$
 - $p(1, b(u, o), a) = p(1, X, Y)$
 - $p(A, 2) = p(3, B)$
 - $p(X, g(Y)) = p(a, g(X))$

$$X=a, g(Y)=g(X)$$

$$X=a, Y=X$$

$$X=a, Y=a$$

Unificação

- verificação de “**igualdade**” entre termos
 - sendo s e t termos e θ uma substituição
 - se $s\theta = t\theta$, então θ é uma unificação para s e t
- identifica qual o valor das variáveis para que dois literais sejam iguais
 - $p(a) = p(X)$
 - $p(1, b(u, o), a) = p(1, X, Y)$
 - $p(A, 2) = p(3, B)$
 - $p(X, g(Y)) = p(a, g(X))$
- mecanismo de **atribuição de valores**

$X=a, g(Y)=g(X)$

$X=a, Y=X$

$X=a, Y=a$

Unificador mais geral

- $p(X, Y) = p(g(Z), Z)$
 - $\theta = \{ X/g(Z), Y/Z \}$
 - $\theta' = \{ X/g(a), Y/a, Z/a \}$
 - θ é mais geral que θ'
- uma substituição θ é mais geral que θ' se houver uma substituição w tal que
 - $\theta' = \theta w$ $w = \{ Z/a \}$ no exemplo
- o unificador mais geral (mgu) é a substituição mais geral para dois termos

Inferência & Backtracking

```
chefe(a).  
chefe(b).  
chefe(c).  
legal(X) :- p1(X).  
legal(X) :- p2(X).  
p1(b).  
p2(c).  
chefe_legal(X) :-  
    chefe(X),  
    legal(X).
```

Inferência & Backtracking

```
chefe(a).  
chefe(b).  
chefe(c).  
legal(X) :- p1(X).  
legal(X) :- p2(X).  
p1(b).  
p2(c).  
chefe_legal(X) :-  
    chefe(X),  
    legal(X).
```

Consulta

```
?- chefe_legal(b).  
yes
```

Inferência & Backtracking

```
chefe(a).
chefe(b).
chefe(c).
legal(X) :- p1(X).
legal(X) :- p2(X).
p1(b).
p2(c).
chefe_legal(X) :-
    chefe(X),
    legal(X).
```

Consulta

?- chefe_legal(b).

y

Consulta

?- chefe_legal(a).

no

Inferência & Backtracking

```
chefe(a).  
chefe(b).  
chefe(c).  
legal(X) :- p1(X).  
legal(X) :- p2(X).  
p1(b).  
p2(c).  
chefe_legal(X) :-  
    chefe(X),  
    legal(X).
```

Consulta

?- chefe_legal(b).

y

Consulta

?- chefe_legal(a).

no

Consulta

?- chefe_legal(X).

X=b

Recursão

```
% fatos (relação entre objetos)
```

```
irmao(bob,alice).
```

```
irmao(alice,tom).
```

```
irmao(tom,ze).
```

```
% propriedades da relação
```

```
 $\forall x,y,z \text{ irmao}(x,y) \wedge \text{irmao}(y,z) \Rightarrow \text{irmao}(x,z)$ 
```

```
irmao(X,Z) :- irmao(X,Y), irmao(Y,Z).
```

Consulta

```
?- irmao(bob,ze).
```

yes

Recursão

Consulta

```
% fatos (relação entre objetos)
irmao(bob,alice).
irmao(alice,tom).
irmao(tom,ze).

% propriedades da relação
 $\forall x,y,z \text{ irmao}(x,y) \wedge \text{irmao}(y,z) \Rightarrow \text{irmao}(x,z)$ 
irmao(X,Z) :- irmao(X,Y), irmao(Y,Z).
```

```
?- irmao(bob,chico).
ERROR: Out of
local stack
```

Remoção da recursão a Esquerda

```
% fatos (relação entre objetos)
```

```
eh_irmao(bob,alice).
```

```
eh_irmao(alice,tom).
```

```
eh_irmao(tom,ze).
```

```
% propriedades da relação
```

```
     $\forall x,y,z \text{ irmao}(x,y) \wedge \text{irmao}(y,z) \Rightarrow \text{irmao}(x,z)$ 
```

```
irmao(X,Y) :- eh_irmao(X,Y).
```

```
irmao(X,Z) :- eh_irmao(X,Y), irmao(Y,Z).
```

SAT solver

- Verificar se uma fórmula com cinco proposições é satisfatível

SAT solver

SAT solver

Java

```
boolean sat(Formula f) {  
    for (boolean A: { true, false})  
        for (boolean B: { true, false})  
            for (boolean C: { true, false})  
                for (boolean D: { true, false})  
                    for (boolean E: { true, false})  
                        if (satisfaz(A,B,C,D,E,f))  
                            return true;  
    return false;  
}
```

SAT solver

SAT solver

Consulta

```
?- bool(A), bool(B),  
    bool(C), bool(D),  
    bool(E),  
    satisfaz(A,B,C,D,E).
```


SAT solver

Prolog

```
bool(true). bool(false).  
satisfaz(A,B,C,D,E) :-  
    ...
```

Consulta

```
?- bool(A), bool(B),  
    bool(C), bool(D),  
    bool(E),  
    satisfaz(A,B,C,D,E).
```

SAT solver

Prolog

```
bool(true). bool(false).  
satisfaz(A,B,C,D,E) :-
```

```
% (A v B v ¬C) ∧ (A v ¬C) ∧ ...
```

```
satisfaz(A,B,C,D,E) :-  
    c1(A,B,C), c2(A,C), ...
```

```
c1(true,_,_). c1(_,true,_). c1(_,_,false).  
c2(true,_).   c2(_,false).  
...
```

Consulta

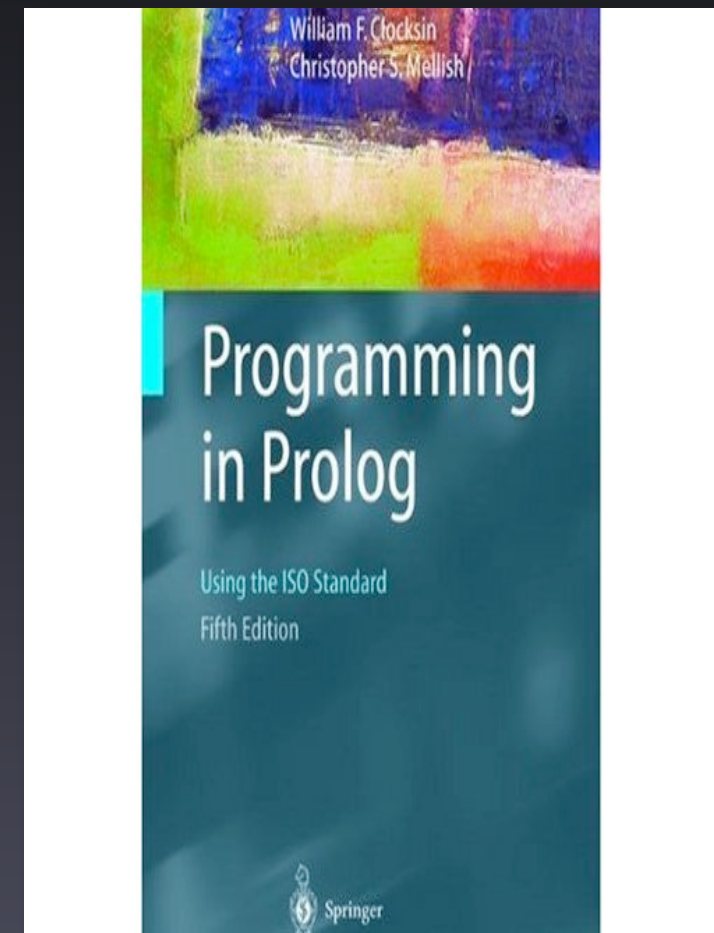
```
?- bool(A), bool(B),  
    bool(C), bool(D),  
    bool(E),  
    ,D,E).
```

Histórico

- 1970: interpretador (Edinburgh)
- Fifth Generation Computer (Japan)
- Hoje temos interpretadores
 - comerciais (Quintus Prolog)
 - open source (SWI-Prolog, g-Prolog)
- Extensões, pesquisas, ...
 - YAP-Prolog (high-performance, ...)
 - Qu-Prolog (quantifiers, concurrent programming, ...)
 - ECLiPSe (constraint programming, ...)

Bibliografia

- **Logic, Programming and Prolog**
(2ed) by Ulf Nilsson and Jan
Maluszynski
- download em
<http://www.ida.liu.se/~ulfni/lpp>



Bibliografia

Bibliografia

