

# Lógica Computacional

---

- Aplicações da Lógica
  - Verificação de Programas,
  - Bases de Dados,
  - Sistemas inteligentes
- Programação em Lógica
- SAT e Programação com Restrições
- Exemplos

# Programação em Lógica

---

- A lógica como linguagem de Programação
- Extensão de linguagens funcionais

Funções: Relações (argumentos de entrada/saída)

Chamadas de funções : Perguntas (queries)

Exemplo:

Listas definidas indutivamente: Uma lista é uma estrutura que

- **Claúsula Base:** é vazia (não tem elementos)
- **Cláusula de Indução:** é constituída por uma cabeça (H) e uma cauda (T) que é uma lista

Sintaticamente

- Lista vazia:  $\square$  em Prolog:  $[\ ]$
- Lista não vazia:  $\bullet (H, T)$  em Prolog:  $[H | T]$

# Programação em Lógica

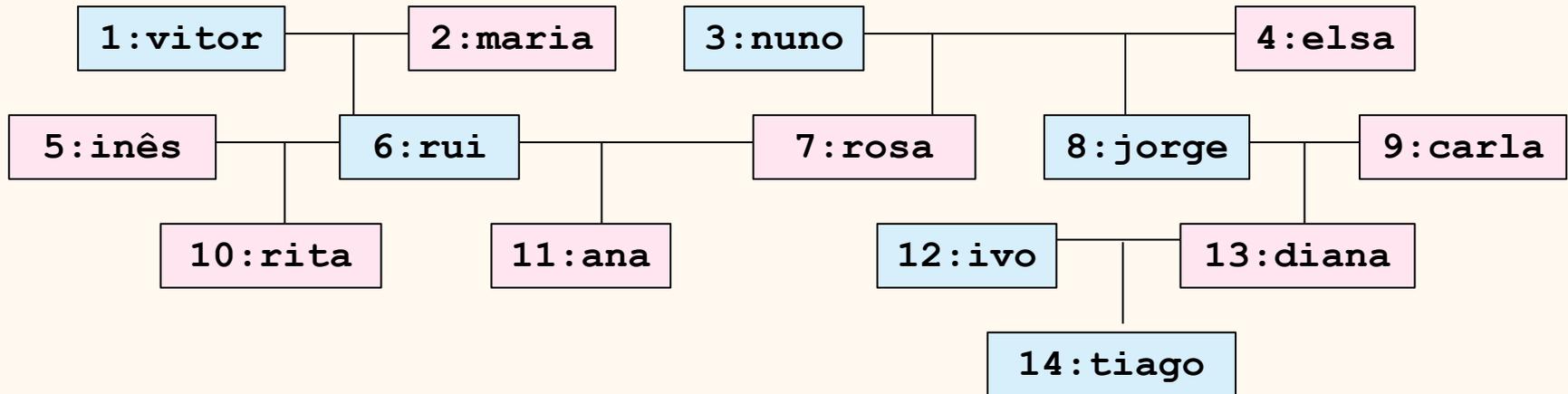
---

## - Exemplos de Tratamento de Listas

- `mb(H, [H|_]) .`                    `% (L = [H|T] ^ X = H) -> mb(X, L)`
- `mb(X, [_|T]) :- mb(X, T) .`       `% (L = [H|T] ^ mb(X, T)) -> mb(X, L) .`
  
- `cat([], L, L) .`
- `cat([H|T], L, [H|R]) :- cat(T, L, R) .`
  
- `nrev([], []) .`
- `nrev([H|T], L) :- nrev(T, R), cat(R, [H], L) .`
  
- `prf(P, L) :- cat(P, _, L) .`
- `sff(P, L) :- cat(_, P, L) .`
  
- `sbb([], []) .`
- `sbb([H|T], [H|L]) :- sb2(T, L) .`
- `sbb(T, [_|L]) :- sb2(T, L) .`

# Base de Conhecimentos: Família

---



**Base de Dados** - Tuplos em tabelas:

`<Nome, Sexo, Pai, Mãe>`

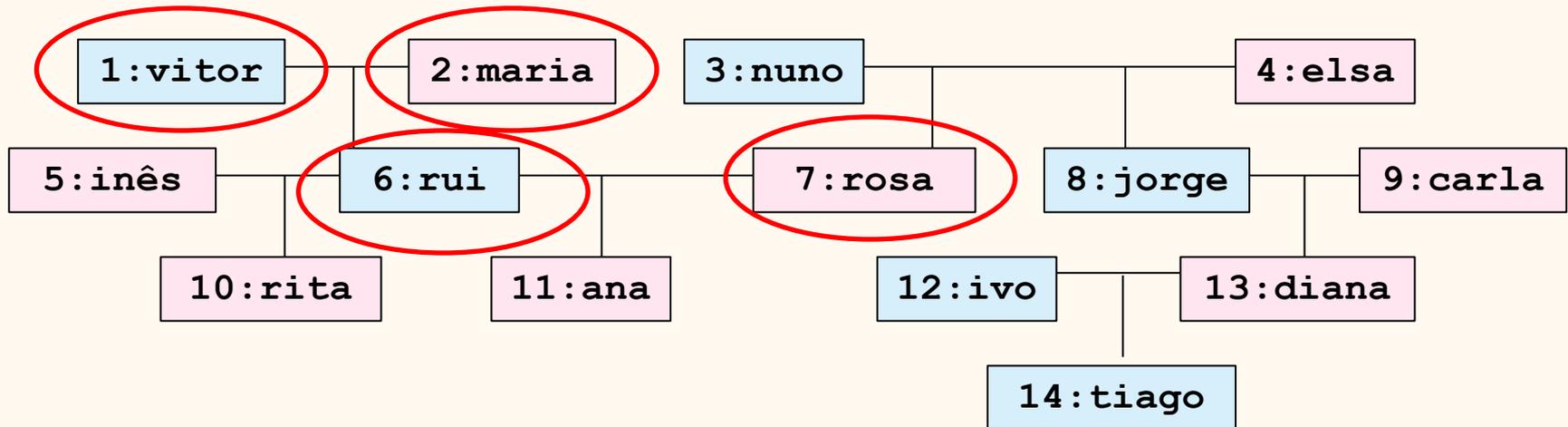
**Base de Conhecimentos** - Tuplos como Predicados

`id(Nome, Sexo, Pai, Mãe)`

# Base de Conhecimentos: Família

## Base de Conhecimentos - Tuplos como Predicados

Predicados representam propriedades/relações de/entre indivíduos.



- `id(vitor,m,x,y)` .
- `id(maria,f,x,y)` .
- ...
- `id(rui,m,vitor,maria)` .
- `id(rosa,f,nuno,elsa)` .
- ...

# Base de Conhecimentos: Família

---

Outras relações podem ser inferidas logicamente:

Directas:

$$\forall x \forall s \forall p \forall m (\text{id}(x,s,p,m) \rightarrow \text{PaiDe}(p,x)) .$$

- $\text{paiDe}(P,X) :- \text{id}(\_,X,\_,P,\_) , P \neq x .$
- $\text{maeDe}(M,X) :- \text{id}(\_,X,\_,\_,M) , M \neq y .$
- $\text{prgDe}(P,F) :- \text{paiDe}(P,F) .$
- $\text{prgDe}(M,F) :- \text{maeDe}(M,F) .$
- $\text{filDe}(F,P) :- \text{prgDe}(P,F) .$
- $\text{filhoDe}(F,P) :- \text{filDe}(F,P) , \text{sexDe}(F,m) .$
- $\text{filhaDe}(F,P) :- \text{filDe}(F,P) , \text{sexDe}(F,f) .$
- $\text{sexDe}(P,S) :- \text{id}(\_,P,S,\_,\_) .$
- $\text{avoDe}(A,N) :- \text{prgDe}(A,P) , \text{prgDe}(P,N) .$
- $\text{netDe}(N,A) :- \text{avoDe}(A,N) .$
- $\text{ascDe}(X,Y) :- \text{prgDe}(X,Y) .$
- $\text{ascDe}(X,Y) :- \text{prgDe}(X,Z) , \text{ascDe}(Z,Y) .$
- $\text{dscDe}(X,Y) :- \text{ascDe}(Y,X) .$

# Base de Conhecimentos: Família

---

Outras relações que podem ser logicamente inferidas:

“Laterais”

- `irmDe (I, J) :- paiDe (P, I) , maeDe (M, I) , paiDe (P, J) , maeDe (M, J) , I \= J.`
- `mirDe (I, J) :- prgDe (P, I) , prgDe (P, J) , I \= J.`
  
- `tioDe (T, S) :- irmDe (T, P) , prgDe (P, S) .`
  
- `casal (X, Y) :- paiDe (X, Z) , maeDe (Y, Z) .`
- `sogDe (X, Y) :- prgDe (X, Z) , casal (Z, Y) .`
- `sogDe (X, Y) :- prgDe (X, Z) , casal (Y, Z) .`
- `genDe (X, Y) :- sogDe (Y, X) .`
  
- `prmDe (X, Y) :- prgDe (Z, X) , prgDe (W, Y) , irmDe (Z, W) .`

# Satisfazibilidade Booleana

## Exemplo: 4-rainhas

Colocar 4 rainhas num tabuleiro de 4 X 4 de forma a que não se ataquem entre elas.

Solução possível:

	●		
			●
●			
		●	

$$Q_{12} = Q_{24} = Q_{31} = Q_{43} = \text{T}$$

$$Q_{11} = Q_{13} = Q_{14} = \text{F}$$

$$Q_{21} = Q_{22} = Q_{23} = \text{F}$$

$$Q_{32} = Q_{33} = Q_{34} = \text{F}$$

$$Q_{41} = Q_{42} = Q_{44} = \text{F}$$

Modelação: Através de 16 variáveis Booleanas

Q11	Q12	Q13	Q14
Q21	Q22	Q23	Q24
Q31	Q32	Q33	Q34
Q41	Q42	Q43	Q44

# Satisfazibilidade Booleana

---

## Exemplo: 4-rainhas

Cláusulas:

// Pelo menos uma rainha em cada linha

$Q_{11} \vee Q_{12} \vee Q_{13} \vee Q_{14}$

$Q_{21} \vee Q_{22} \vee Q_{23} \vee Q_{24}$

$Q_{31} \vee Q_{32} \vee Q_{33} \vee Q_{34}$

$Q_{41} \vee Q_{42} \vee Q_{43} \vee Q_{44}$

// No máximo uma rainha na linha 1

$\neg Q_{11} \vee \neg Q_{12}$

$\neg Q_{11} \vee \neg Q_{13}$

$\neg Q_{11} \vee \neg Q_{14}$

$\neg Q_{12} \vee \neg Q_{13}$

$\neg Q_{12} \vee \neg Q_{14}$

$\neg Q_{13} \vee \neg Q_{14}$

Q11	Q12	Q13	Q14
Q21	Q22	Q23	Q24
Q31	Q32	Q33	Q34
Q41	Q42	Q43	Q44

# Satisfazibilidade Booleana

## Exemplo: 4-rainhas

// No máximo uma rainha nas diagonais /

$\neg Q_{21} \vee \neg Q_{12} \quad i+j = 3$

$\neg Q_{31} \vee \neg Q_{22}$

$\neg Q_{31} \vee \neg Q_{13} \quad i+j = 4$

$\neg Q_{22} \vee \neg Q_{13}$

$\neg Q_{41} \vee \neg Q_{32}$

$\neg Q_{41} \vee \neg Q_{23}$

$\neg Q_{41} \vee \neg Q_{14} \quad i+j = 5$

$\neg Q_{32} \vee \neg Q_{23}$

$\neg Q_{32} \vee \neg Q_{14}$

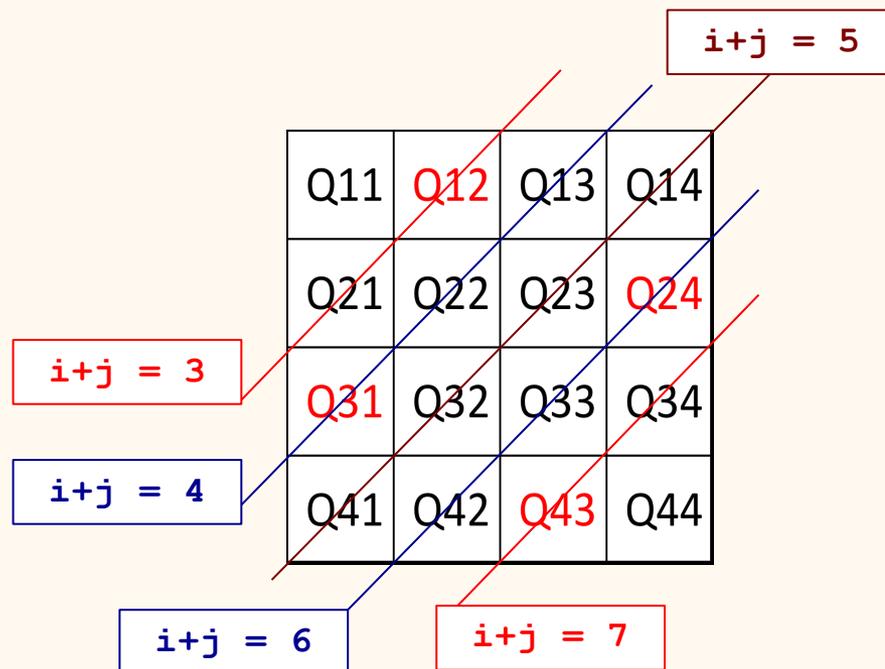
$\neg Q_{23} \vee \neg Q_{14}$

$\neg Q_{42} \vee \neg Q_{33}$

$\neg Q_{42} \vee \neg Q_{24} \quad i+j = 6$

$\neg Q_{33} \vee \neg Q_{24}$

$\neg Q_{43} \vee \neg Q_{34} \quad i+j = 7$



# Satisfazibilidade Booleana

---

## Exemplo: 4-rainhas (generalizável para n-rainhas)

Número de variáveis: 16

Número de cláusulas:

1 cláusula n-ária por linha e coluna

6 cláusulas binárias por linha e coluna

14 cláusulas binárias por cada diagonal

## Para uma dimensão arbitrária do tabuleiro (n-rainhas)

Número de variáveis:  $n^2$

Número de cláusulas:

1 cláusula n-ária por linha e coluna

$n(n-2)/2$  cláusulas binárias por linha e coluna

$n^3/3 - n^2/2 + n/6$  cláusulas binárias por cada diagonal

n	número variáveis	n-árias linha/coluna	2-árias	
			linha/coluna	diagonal
4	16	8	48	28
10	100	20	900	570
100	10 000	200	990 000	656 700
1000	1 000 000	2 000	999 000 000	665 667 000

# Satisfazibilidade Booleana

---

n	número variáveis	n-árias linha/coluna	2-árias	
			linha/coluna	diagonal
4	16	8	48	28
10	100	20	900	570
100	10 000	200	990 000	656 700
1000	1 000 000	2 000	999 000 000	665 667 000

Resolver este problema, e outros problemas igualmente exponenciais, envolve tomar decisões. Neste caso existem  $n$  decisões a tomar (uma para cada linha), cada uma com  $n$  hipóteses alternativas.

No pior caso, este problema teria assim uma complexidade exponencial de ordem  $O(n^n)$  o que torna impossível a sua resolução “ao calhas”.

Este problema é pois um exemplo (*imperfeito*) de um problema de satisfação Booleana (SAT) conhecido por ser representativo de uma classe de problemas combinatórios (NP-completos).

E no entanto, este problema pode ser resolvido em poucos segundos, mesmo com valores elevados de  $n$  : por exemplo  $n = 1000!$

# Programação com Restrições

---

SAT é um caso particular de Programação com Restrições em que:

- as variáveis tomam valores Booleanos; e
- as restrições correspondem a cláusulas.

Em muitos casos, é referível usar linguagens mais expressivas (e eficientes) em que as variáveis podem tomar valores finitos e as restrições são mais variadas (aritméticas, diferença, contagens, etc.)

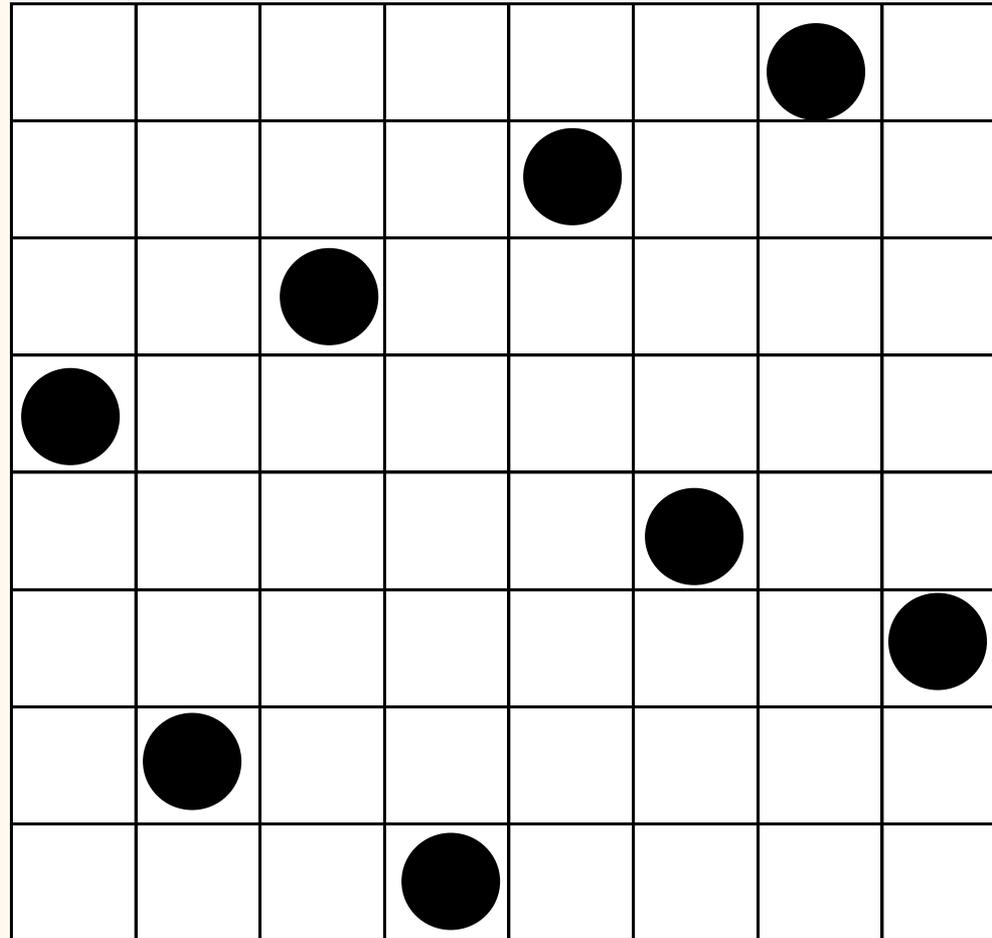
Por exemplo, o problemas das n-rainhas pode ser especificado como

```
int n = 8;
range Q = 1..n;
Solver<CP> cp();
    var q[Q] (cp, Q);
solve<cp> {
    forall (i in Q, j in Q: j > i){
        cp.post(q[i] != q[j]);
        cp.post(q[i]+ i != q[j] + j);
        cp.post(q[i]+ j != q[j] + i);
    }
}
```

# Programação com Restrições

---

7 5 3 1 6 8 2 4



Tests 0

Backtracks 0

# Continuação

---

## - Lógica e Bases de Dados

- Bases de Dados
- Sistemas de Bases de Dados
- Modelação de Dados

## - Lógica e Verificação de Programas

- Construção e Verificação de Software

## - Lógica Modelação e Resolução de Problemas

- Inteligência Artificial
- Representação de Conhecimento e Sistemas de Raciocínio
- Programação por Restrições
- Desenho de Algoritmos para Problemas de Otimização