MAQUINA DE APRENDIZAJE Y META-INTERPRETACION PROLOG

Lopes de Siqueira Neto José
Universidade Federal de Minas de Gerais UFMG - Brasil

Correo electrónico: jose@ufmg.br

PROLOG, BREVE HISTORIA DEL LENGUAJE

- Prolog es la abreviación de Programación Lógica
- Prolog es un lenguaje de uso general basada en lógica de primer orden, utilizado en sistemas de Inteligencia Artificial
- El primer intérprete de Prolog fue desarrollado en 1972 por Alain Colmerauer y Philippe Roussel en Marsella, Francia.
- Objetivo: emplear Prolog para investigación en lingüística computacional
- Primer contacto con Prolog: CTI, Campiñas, 1983
- Fue un amor a primera vista
- Sistema de interpretación simbólica en Prolog

```
d(KW) :- number(K), d(U,X,W).
d(U+V,X,B+U+A+V) :- d(U,X,A), d(V,X,B).
d(U/V,X,W) :- d(U*V^(-1),X,W).
d(U,X,V+W+U^(V+ (-1))) :- number(V), d(U,X,W).
d(U,X,Z+log(U)+U+V+W+U^V+ (-1))) :- d(U,X,W), d(V,X,Z).
d(log(T),X,R+T^(-1)) :- d(T,X,R).
d(exp(T),X,R+exp(T)) :- d(T,X,R).
d(sin(T),X,R+cos(T)) :- d(T,X,R).
d(cos(T),X,-R+sin(T)) :- d(T,X,R).
d(tan(T),X,W) :- d(sin(T)/cos(T),X,W).
```

- Con aplicación en generación de modelos dinámicos de robots de hasta 6 grados de libertad.
- Problema: ¿Como multiplicar matrices recursivamente?
- Algoritmo procedimental:

```
int main(){
  int i,j,k;
  int A[K][L],B[L][M],C[K][M];

  for(i=0;i<K;i++){
    for(j=0;j<L;j++){
      for(k=0;k<M;K++){
        C[i][k] += A[i][k]+b[k][j]
      }
    }
  }
  return 0;
}
```

- La salvación: Lógica de combinadores de John Backus
- Mi mentor: Prof. Antonio Eduardo Costa, de la UFU
- En lógica de combinadores, no se permite una multiplicación de matrices recursivamente, como la derivada parcial de matrices.

CARACTERÍSTICAS GENERALES

- Programación declarativa (el qué) x programación procedimental (el qué y cómo).
- Programación relacional.
- Reversibilidad de entrada y salida en programas
- Programación simbólica: no existen tipos en Prolog.
- Meta interpretación: programas Prolog pueden ser datos de otros programas Prolog.
- Variables lógicas: se lee un programa Prolog como un texto de matemática
- No existe atribuciones en Prolog.
- Fin del absurdo: x=x+1 -> x=x+1 -> 0=1?

- Prolog trae dos mecanismos poderosos embutidos:
  - ✓ Backtracking (vuelta al último punto escogido)
  - ✓ Unificación (igualdad sintáctica de términos)
- El backtracking permite emular programación no determinística
- La unificación permite la utilización de variables lógicas en sustitución de variables por términos del lenguaje.

DOS EJEMPLOS DE PROGRAMAS PROLOG:

```
permuta1([],[]).
permuta1(L,[X|Perm]):-
   delete(X,L,LsemX),
   permuta1(LsemX,Perm).

delete(X,[X|L],L).
delete(X,[Y|L],[Y|LsemX]):-
  delete(X,L,LsemX).
```

```
permuta2([],[]).
permuta2([X|L],Perm):-
   permuta2(L,LPerm),
   insere(X,LPerm,Perm).

insere(X,L,[X|L]).
insere(X,[Y|L],[Y|LcomX]):-
  insere(X,L,LcomX).
```

EJECUCIÓN DE 2 PROGRAMAS PROLOG:

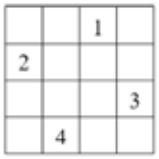
```
| ?- permuta1([1,2,3],P).

P = [1,2,3] ? a

P = [1,3,2]

P = [2,1,3]

P = [2,3,1]

P = [3,1,2]

P = [3,2,1]

no
```

```
| ?- permuta2([1,2,3],P).

P = [1,2,3] ? a

P = [2,1,3]

P = [2,3,1]

P = [1,3,2]

P = [3,1,2]

P = [3,2,1]

no
```

EJEMPLO DEL PARADIGMA DE GENERACIÓN Y PRUEBAS:

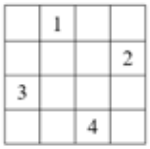
¿Cómo colocar N reinas en un tablero de ajedrez NxN sin que ninguna reina ataque las demás?

Ejemplo para N=4:



[2, 4, 1, 3]

2da solución para N=4:



[3, 1, 4, 2]

Programa Prolog para N reinas:

```
sol(L,Qs):- permute(L,Qs), safe(Qs).

safe([]).
safe([Q|Qs]):-
  nodiag(Q,Qs,1),
  safe(Qs).

nodiag(_,[],_).
nodiag(Q1,[Q2|Qs],N):-
  noatk(Q1,Q2,N), N1 is N + 1,
  nodiag(Q1,Qs,N1).

noatk(Q1,Q2,N):- Q1 > Q2, D is Q1 - Q2, D =\= N.

noatk(Q1,Q2,N):- Q1 < Q2, D is Q2 - Q1, D =\= N.
```

INEFICIENCIA DEL PROGRAMA PROLOG GENERACIÓN Y PRUEBAS

Problema: Un programa para Generación y Pruebas en Prolog, es extremadamente ineficiente.

- La razón: Una falla en una prueba, los términos involucrados en una prueba que falla, no son considerados en la próxima en la siguiente generación.
- El culpable: el mecanismo de backtracking cronológico de Prolog.
- Propuesta de solución general: investigaciones en backtracking inteligente, en la década del 90.
- Más backtracking inteligente no es eficiente: regra 90/10.

PROPUESTA DE GENERACIÓN Y PRUEBA OPTIMIZADOS – GTO

Desarrollar un nuevo programa Prolog de Generación y Prueba más eficiente

Un programa desarrollado por GTO, incluye las pruebas durante la generación justo después del punto en que los términos involucrados en éstas pruebas se generan.

- En la ejecución del programa desarrollado, los siguientes términos solo serán generados si los anteriores hubieran pasado en todas la pruebas en las que estuvieron involucradas.
- Esto hace que el programa de generación y prueba sea mucho mas eficiente que el original.

META INTERPRETADOR PROLOG SIMPLE:

```
prove(true).
prove((A,B)):-
        prove(A),
        prove(B).
prove((A;B)):-
        prove(A)
        ;
        prove(B).
prove(A):-
        clause(A,B),
        prove(B).
```

INCLUSIONES NECESARIAS PARA LA EJECUCIÓN (BUILT-INS):

```
prove(A > B):- !,
        A > B.
prove(A < B):-  !,
        A < B.
prove(A is B):- !,
        A is B.
prove(A =\= B):- !,
        A =\= B.
```

EJECUCIÓN DE META-INTERPRETACIÓN SIMPLE:

```
| ?- prove(sol([1,2,3,4],Qs)).

Qs = [2,4,1,3] ? ;

Qs = [3,1,4,2] ? ;

no
```

APRENDIZAJE SIMBÓLICO:

- Es una forma de inferencia
- Deductiva (EBG), es hecha a partir de pruebas.
- Inductiva (ILP), es hecha a partir de ejemplos.
- Necesidad de generalización y síntesis.
- Explanation-Based Generalization, Mitchel, 1986.
- EBG = Partial Evaluation, van Harmelen e Bundy, 1986.

GENERACIÓN Y PRUEBAS OPTIMIZADOS: EL ALGORITMO

- Entrada: un programa Prolog de generación y prueba inefiiente.
- Salida: un programa Prolog de generación y prueba equivalente, más eficiente.
- Algoritmo:
  - Meta-interpreta un programa original, sin ejecutarlo, con una entrada cualquiera.
  - Una meta-interpretación recoge partes del árbol de ejecución generalizadas:
    - Las llamadas para el predicado de generación;
    - Todas las pruebas, sin ejecutarlos.
    - Desarrollar un nuevo programa a partir de los indicados arriba.
- Resultado: un nuevo programa desarrollado, que es más eficiente que el original.

EJEMPLO DESARROLLADO PARA LAS 4 REINAS:

```
perm4([],[]).
perm4(A,[B,C,D,E|J]):-
     del(B,A,F),
     del(C,F,G),
     noatk(B,C,1),
     del(D,G,H),
     noatk(B,D,2),
     noatk(C,D,1),
     del(E,H,I),
     noatk(B,E,3),
     noatk(C,E,2),
     noatk(D,E,1),
     perm4(I,J).
```


bit@bit Vol. 2 Nº 3 ISSN 2519 - 741X, Pág. 63 – 69. Junio 2017

PROGRAMA DESARROLLADO PARA 8 REINAS (I):

```
perm8([],[]).
perm8(A,[B,C,D,E,F,G,H,I|R]):-
        del(B,A,J),
        del(C,J,K),
        noatk(B,C,1),
        del(D,K,L),
        noatk(B,D,2),
        noatk(C,D,1),
        del(E,L,M),
        noatk(B,E,3),
        noatk(C,E,2),
        noatk(D,E,1),
        del(F,M,N),
        noatk(B,F,4),
        noatk(C,F,3),
```

PROGRAMA DESARROLLADO PARA 8 REINAS (II):

```
noatk(D,F,2),
noatk(E,F,1),
del(G,N,O),
noatk(B,G,5),
noatk(C,G,4),
noatk(D,G,3),
noatk(E,G,2),
noatk(F,G,1),
del(H,O,P),
noatk(B,H,6),
noatk(C,H,5),
noatk(D,H,4),
noatk(E,H,3),
noatk(F,H,2),
noatk(G,H,1),
```

PROGRAMA DESARROLLADO PARA 8 REINAS (III):

```
del(I,P,Q),
noatk(B,I,7),
noatk(C,I,6),
noatk(D,I,5),
noatk(E,I,4),
noatk(F,I,3),
noatk(G,I,2),
noatk(H,I,1),
perm8(Q,R).
```

RESULTADOS PARA TODAS LAS SOLUCIONES PARA 4 REINAS:

Hardware de pruebas: CPU Intel Core I7 5500U 2.4 GHz (2 núcleos/4 hilos)

Tabela: Comparação para 4 rainhas

Rainhas	Tempo de chamadas	Original	Otimizado	Ganho	Ganho total
4	Tempo (ms)	0	0		
	delete	128	64	100%	
	noattack	93	87	7%	
	perm	88	5	1660%	97%
	soluções	2	2		

RESULTADOS PARA TODAS LAS SOLUCIONES PARA 8 REINAS:

Tabela: Comparação para 8 rainhas

Rainhas	Tempo de chamadas	Original	Otimizado	Ganho	Ganho total
8	Tempo (ms)	576	24	2.300%	
	delete	219.200	11.016	1.890%	
	noattack	352.635	32.916	971%	
	perm	149.920	185	80.938%	1.533%
	soluções	92	92	4.500%	

RESULTADOS PARA TODAS LAS SOLUCIONES PARA 9 REINAS:

Tabela: Comparação para 9 rainhas

Rainhas	Tempo de chamadas	Original	Otimizado	Ganho	Ganho total
9	Tempo (ms)	3.144	80	3.830%	
	delete	1.972.818	48.022	4.008%	
	noattack	3.626.406	165.969	2.085%	
	perm	149.920	705	191.289%	3.131%
	soluções	352	352	283%	

RESULTADOS PARA TODAS LAS SOLUCIONES PARA 8 REINAS:

Tabela: Comparação para 10 rainhas

Rainhas	Tempo de chamadas	Original	Otimizado	Ganho	Ganho total
10	Tempo (ms)	32.296	312	10.251%	
	delete	19.728.200	220.008	8.867%	
	noattack	40.755.519	809.658	4.934%	
	perm	13.492.900	1.449	931.087%	7.070%
	soluções	724	724	106%	

RESULTADOS PARA TODAS LAS SOLUCIONES PARA 11 REINAS:

Tabela: Comparação para 11 rainhas

Rainhas	Tempo de chamadas	Original	Otimizado	Ganho	Ganho total
11	Tempo (ms)	431.332	2.772	15.460%	
	delete	217.010.222	1.092.714	19.760%	
	noattack	498.146.763	4.462.782	11.062%	
	perm	148.421.911	5.361	80.938%	2.768.449%
	soluções	2.680	2.680	270%	

RESULTADOS PARA TODAS LAS SOLUCIONES PARA 12 REINAS (SOLO OPTIMIZADO)

Tabela: Comparação de 12 rainhas com 11 rainhas

Rainhas	Tempo de chamadas	Otimizado	Ganho	Ganho total
12	Tempo (ms)	14.188	411,8%	00:14,2
	delete	5.831.480	433,7%	
	noattack	26.573.097	495%	
	perm	28.401	430%	483%
	soluções	14.200	430%	

RESULTADOS PARA TODAS LAS SOLUCIONES PARA 13 REINAS (SOLO OPTIMIZADO)

Tabela: Comparação de 13 rainhas com 12 rainhas

Rainhas	Tempo de chamadas	Otimizado	Ganho	Ganho total
13	Tempo (ms)	87.444	5178%	00:14,2
	delete	5.831.480	465%	
	noattack	165.198.555	522%	
	perm	147.425	419%	511%
	soluções	73.712	419%	

RESULTADOS PARA TODAS LAS SOLUCIONES PARA 14 REINAS (SOLO OPTIMIZADO)

Tabela: Comparação de 14 rainhas com 13 rainhas

Rainhas	Tempo de chamadas	Otimizado	Ganho	Ganho total
14	Tempo (ms)	743.064	750%	12:23,1
	delete	198.561.008	503%	
	noattack	1.082.676.480	555%	
	perm	731.193	547%	483,2%
	soluções	365.596	396%	

RESULTADOS PARA LA PRIMERA SOLUCIÓN, PARA 15 REINAS (SOLO OPTIMIZADO)

Tabela: Tempo de execução e nº de chamadas para 15 rainhas

Rainhas	Tempo de chamadas	Otimizado	Ganho	Ganho total
15	Tempo (ms)	60	–	00:00,1
	delete	14.181	–	
	noattack	71.382	–	
	perm	2	–	
	soluções	1	–	

RESULTADOS PARA LA PRIMERA SOLUCIÓN, PARA 16 REINAS (SOLO OPTIMIZADO)

Tabela: Comparação de 16 rainhas com 15 rainhas

Rainhas	Tempo de chamadas	Otimizado	Ganho	Ganho total
16	Tempo (ms)	364	507%	00:00,4
	delete	105.818	646%	
	noattack	587.325	723%	710%
	perm	2	–	
	soluções	1	–	

RESULTADOS PARA LA PRIMERA SOLUCIÓN, PARA 32 REINAS

Tabela: Comparação de 32 rainhas com 16 rainhas

Rainhas	Tempo de chamadas	Otimizado	Ganho	Ganho total
32	Tempo (ms)	7.527.944	2.068.017%	05:27,9
	delete	1.638.476.004	1.548.291%	
	noattack	16.137.710.867	2.747.563%	2.564.477%
	perm	2	–	
	soluções	1	–	

CONCLUSIONES

- GTO es una herramienta de aprendizaje simbólico para el paradigma de resolución de problemas de generación y prueba.
- Posee todas las características deseables de un sistema de aprendizaje: aprendizaje incremental, generalización, aumento de eficiencia de la nueva regla desarrollada, inferencia del nuevo conocimiento a través de una nueva regla simbólica.
- Meta-interpretación Prolog y generalización son medios poderosos de aprendizaje simbólico.

TRABAJOS FUTUROS:

- Generación y prueba no es útil.
- ¡Cómo que no! El algoritmo A* de optimización combinatoria, muy utilizado en Inteligencia Artificial, es un algoritmo de generación y prueba. En él se generan a partir de un nodo inicial y prueban los nodos más prometedores con una función heurística, a fin de desarrollarlos en el próximo paso del algoritmo.
- GTO genera una nueva regla más eficiente para una instancia del problema. El desafío es generar una regla más general a partir de reglas particulares: en vez de tener N reglas eficientes, tener una única regla eficiente para cualquier instancia del problema.