

Representación del Conocimiento

Planeación

Dr. Alejandro Guerra-Hernández

Instituto de Investigaciones en Inteligencia Artificial
Universidad Veracruzana

*Campus Sur, Calle Paseo Lote II, Sección Segunda No 112,
Nuevo Xalapa, Xalapa, Ver., México 91097*

mailto:aguerra@uv.mx
<https://www.uv.mx/personal/aguerra/rc>

Maestría en Inteligencia Artificial 2023



Universidad Veracruzana

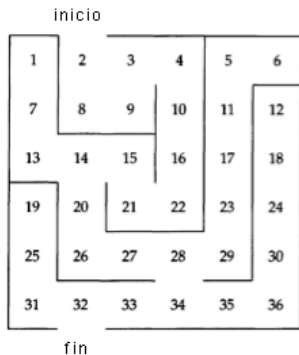
¿Qué estudia la Inteligencia Artificial?

- ▶ Según Patrick Winston, en su curso del MIT (6.034):
 - ▶ **Algoritmos** construidos a partir de
 - ▶ **restricciones** expuestas por
 - ▶ **representaciones** que dan soporte a
 - ▶ **modelos** del
 - ▶ **razonamiento**, la **percepción** y la **acción**.



Búsqueda como modelo de acción

- ▶ ¿Cómo construimos un **algoritmo** para salir del laberinto?



Representación lógica

- ▶ El laberinto se puede representar como un **grafo** usando la relación *conectado/2*:

```
27 %%% conectado/2
28 %%% computa si la Pos1 esta conectada con la Pos2
29
30 conectado(Pos1, Pos2) :- conecta(Pos1, Pos2).
31 conectado(Pos1, Pos2) :- conecta(Pos2, Pos1).
32
33 %%% Datos del laberinto (adyacencia)
34
35 conecta(inicio,2).
36 conecta(1,7).
37 conecta(2,8).
38 conecta(2,3). %%% agrega varias soluciones.
39 conecta(3,4).
40 conecta(3,9).
```



Algoritmo

- ▶ Una **búsqueda primero en profundidad**, evitando regresar por el mismo camino:

```
5   %%% sol/0
6   %%% computa los caminos de solución para el laberinto.
7
8   sol :- camino([inicio], Sol), write(Sol).
9
10  %%% camino/2
11  %%% si el camino llego al fin, regresa el camino de fin a inicio.
12  %%% en caso contrario busca ir a una posición que no se haya visitado
13  %%% anteriormente ( miembro )
14
15  camino([fin|RestoDelCamino], [fin|RestoDelCamino]).
16  camino([PosActual|RestoDelCamino], Sol) :-
17      conectado(PosActual,PosSiguiente),
18      \+ miembro(PosSiguiente, RestoDelCamino),
```



Consulta

```
1 ?- sol.  
2 [fin,32,33,34,28,27,26,20,14,15,21,22,16,10,4,3,9,8,2,inicio]  
3 true ;  
4 [fin,32,33,34,28,27,26,20,14,15,21,22,16,10,4,3,2,inicio]  
5 true ;  
6 false.
```



Ejercicio sugerido

- ▶ Consideren el programa para resolver el problema del granjero, el zorro, la gallina y el saco de maíz que deben cruzar el río (Tarea del curso de PIA).
- ▶ ¿Porqué es tan **corto** el algoritmo?
- ▶ ¿Es esto **planear**?



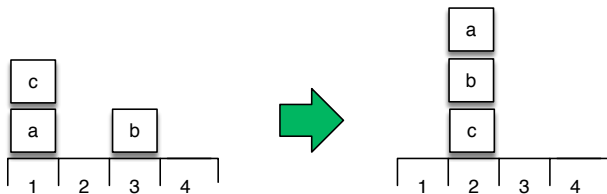
Planear

- ▶ Razonar **explícitamente** acerca de los **efectos** de las acciones y la secuencia en que estas se aplican para lograr un efecto acumulativo dado –Bratko [1]
- ▶ Se trata de un proceso **deliberativo** abstracto y explícito, que elige y organiza acciones anticipando sus **efectos**; la meta es alcanzar una serie de objetivos predefinidos de la **mejor manera** posible –Ghallab, Nau y Traverso [3]



Caso de estudio: el mundo de los bloques

- Usaremos las relaciones *en/2* y *libre/1* con la semántica intuitiva.



$$Edo_0 = [libre(2), libre(4), libre(c), libre(b), \\ en(a, 1), en(b, 3), en(c, a)]$$

$$Metas = [en(a, b), en(b, c)]$$



Acciones

- ▶ Usaremos una descripción estilo STRIPS (Fikes y Nilsson [2]):
 - ▶ **Condiciones.** Las condiciones que **debe observarse** en el estado del mundo, para que la acción pueda ejecutarse.
 - ▶ **Agregar.** Es una lista de observaciones que, se espera, **se satisfagan** después de ejecutarse la acción.
 - ▶ **Borrar.** Es una lista de observaciones que, se espera, **no se satisfagan** más, después de ejecutarse la acción.



Ejemplo: mover/3

- ▶ Usamos las relaciones *precond/2*, *agregar/2*, *borrar/2*:

```
1  %%% Acción mover en mundo de bloques
2
3  precond(mover(Bloque,De,A),
4          [libre(Bloque),libre(A),en(Bloque,De)]) :-
5          bloque(Bloque),
6          objeto(A),
7          A\==Bloque,
8          objeto(De),
9          De\==A,
10         Bloque\==De.
11
12  agregar(mover(Bloque,De,A),[en(Bloque,A),libre(De)]).
13
14  borrar(mover(Bloque,De,A),[en(Bloque,De),libre(A)]).
```



Ontología, estados y metas I

- ▶ La ontología suele definir **relaciones** usadas en la especificación de las acciones, los estados y las metas.

```
16 %%% Ontología
17
18 objeto(X) :- lugar(X); bloque(X).
19
20 bloque(a).
21 bloque(b).
22 bloque(c).
23
24 lugar(1).
25 lugar(2).
26 lugar(3).
27 lugar(4).
28
29 % Edo del mundo de los bloques
30 % c
31 % a b
32 % =====
```

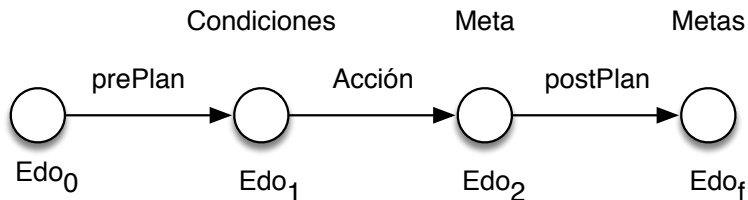


Ontología, estados y metas II

```
33  % 1234
34
35  estado([libre(2),libre(4),libre(b),libre(c),en(a,1),en(b,3),en(c,a)]).
36
37  metas([en(a,b)]).
```



Idea Intuitiva



Algoritmo

- ▶ Si todas las *Metas* son verdaderas en Edo_0 , entonces $Edo_f = Edo_0$.
En cualquier otro caso:
 1. Seleccionar una *Meta* no solucionada en *Metas*.
 2. Encontrar una *Accion* que agregue *Meta* al estado actual.
 3. Hacer posible *Accion* resolviendo sus *Condiciones*, para obtener el estado intermedio Edo_1 .
 4. Aplicar la *Accion* en el estado Edo_1 para obtener el estado intermedio Edo_2 donde *Meta* se cumple.
 5. Resolver *Metas* en el estado Edo_2 para llegar a Edo_f .



Implementación I

```
41 plan(Edo, Metas, [], Edo) :-
42     metas_satisfechas(Edo, Metas).
43
44 plan(Edo, Metas, Plan, EdoFinal) :-
45     append(PrePlan, [Accion|PostPlan], Plan),
46     seleccionar(Edo, Metas, Meta),
47     lograr(Accion, Meta),
48     precond(Accion, Condicion),
49     plan(Edo, Condicion, PrePlan, EdoInter1),
50     aplicar(EdoInter1, Accion, EdoInter2),
51     plan(EdoInter2, Metas, PostPlan, EdoFinal).
52
53 metas_satisfechas(_, []).
54
55 metas_satisfechas(Edo, [Meta|Metas]) :-
56     member(Meta, Edo),
57     metas_satisfechas(Edo, Metas).
58
59 seleccionar(Edo, Metas, Meta) :-
60     member(Meta, Metas),
61     not(member(Meta, Edo)).
```



Implementación II

```
62
63 lograr(Accion,Meta) :-
64     agregar(Accion,Metas),
65     member(Meta,Metas).
66
67 aplicar(Edo,Accion,NewEdo) :-
68     borrar(Accion,ListaBorrar),
69     borrar_todos(Edo,ListaBorrar,Edo1),!,
70     agregar(Accion,ListaAgregar),
71     append(ListaAgregar,Edo1,NewEdo).
72
73 % Borra de [X|L1] todos los elementos que aparecen en L2
74 % dando como resultado Diff.
75
76 borrar_todos([],_,[]).
77
78 borrar_todos([X|L1],L2,Diff) :-
79     member(X,L2),!,
80     borrar_todos(L1,L2,Diff).
81
```

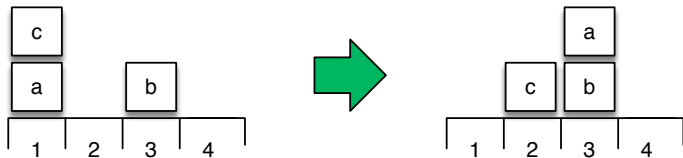


Implementación III

```
82 borrar_todos([X|L1],L2,[X|Diff]) :-  
83     borrar_todos(L1,L2,Diff).
```



Corrida



- 1 ?- estado(E), metas(M), plan(E,M,P, _).
- 2 E = [libre(2), libre(4), libre(b), libre(c), en(a,1), en(b,3), en(c,a)],
- 3 M = [en(a,b)],
- 4 P = [mover(c,a,2), mover(a,1,b)],

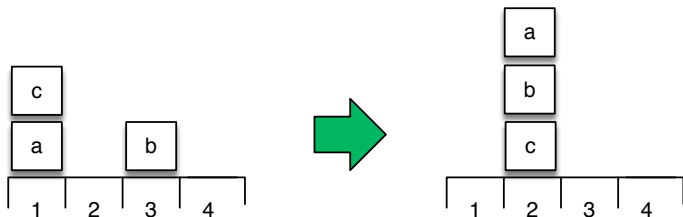


Explosión combinatoria

- ▶ El mundo de los bloques es mucho más **complejo** de lo que parece –Gupta y Nau [4]: NP-duro.
- ▶ El planificador tiene muchas más acciones **posibles** de las que son razonables bajo el principio de análisis medios-fines.
- ▶ Esto puede dar lugar a **defectos** en el planificador.



Un plan más complicado



- 1 ?- estado(E),plan(E, [en(a,b),en(b,c)], P, _).
- 2 E = [libre(2),libre(4),libre(b),libre(c),en(a,1),en(b,3),en(c,a)],
- 3 P = [mover(b,3,c),mover(b,c,3), mover(c,a,2),mover(a,1,b),
- 4 mover(a,b,1),mover(b,3,c), mover(a,1,b)]

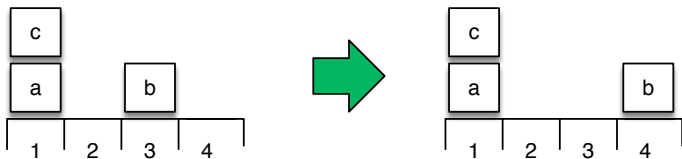


¿Qué está pasando?

Acción	Objetivo del planeador
$mover(b, 3, c)$	satisfacer $en(b, c)$
$mover(b, c, 3)$	satisfacer $clear(c)$ y ejecutar siguiente acción
$mover(c, a, 2)$	satisfacer $clear(a)$ y $mover(a, 1, b)$
$mover(a, 1, b)$	satisfacer $en(a, b)$
$mover(a, b, 1)$	satisfacer $clear(b)$ y $mover(b, 3, c)$
$mover(b, 3, c)$	satisfacer $en(b, c)$ otra vez
$mover(a, 1, b)$	satisfacer $en(a, b)$ otra vez



Catástrofe



- 1 `?- estado(E), plan(E, [libre(2), libre(3)], P, _).`
- 2 `ERROR: Out of global stack`



Planeador con metas protegidas

```
87 plan_metas_protegidas(EdoInicial, Metas, Plan, EdoFinal) :-
88     plan_mp(EdoInicial, Metas, [], Plan, EdoFinal).
89
90 plan_mp(Edo, Metas, _, [], Edo) :-
91     metas_satisfechas(Edo, Metas).
92
93 plan_mp(Edo, Metas, Protegido, Plan, EdoFinal) :-
94     append(PrePlan, [Accion|PostPlan], Plan),
95     seleccionar(Edo, Metas, Meta),
96     lograr(Accion, Meta),
97     precond(Accion, Condicion),
98     preservar(Accion, Protegido),
99     plan_mp(Edo, Condicion, Protegido, PrePlan, EdoInter1),
100    aplicar(EdoInter1, Accion, EdoInter2),
101    plan_mp(EdoInter2, Metas, [Meta|Protegido], PostPlan, EdoFinal).
102
103 preservar(Accion, Metas) :-
104    borrar(Accion, ListaBorrar),
105    not( (member(Meta, ListaBorrar),
106        member(Meta, Metas))).
```



Nueva corrida

- ▶ Ahora se puede ejecutar la consulta:

```
1 ?- estado1(E), plan_metas_protegidas(E,[libre(2), libre(3)],P,_).  
2 E = [libre(2),libre(4),libre(b),libre(c),en(a,1),en(b,3),en(c,a)],  
3 P = [mover(b,3,2), mover(b,2,4)]
```

- ▶ Aunque seguimos sin obtener el mejor plan!
- ▶ ¿Cómo **buscamos** la solución?



Aspectos procedurales 1

- ▶ Observen la meta:

```
1 append(PrePlan, [Accion|PostPlan], Plan)
```

- ▶ *Plan* no está instanciada cuando esta meta es alcanzada.
- ▶ Por lo que *append/3* genera al reconsiderar, candidatos alternativos para *PrePlan* en el siguiente orden:

```
1 PrePlan = [];  
2 PrePlan = [_];  
3 PrePlan = [_,_];  
4 PrePlan = [_,_,_];  
5 ...
```

- ▶ *PrePlan* establece las condiciones para ejecutar *Accion*.
- ▶ Esto permite encontrar una acción cuya condición puede satisfacerse por un plan **tan corto** como sea posible.



Aspectos procedurales 2

- ▶ *PostPlan* no está instanciada, y por tanto su longitud es **ilimitada**.
- ▶ La búsqueda es globalmente **primero en profundidad**, y localmente **primero en amplitud**.
- ▶ El encadenamiento hacía adelante de las acciones que se agregan al plan emergente, es una búsqueda primero en profundidad.
- ▶ Cada acción es validada por un *PrePlan*, este plan es por otra parte, buscado primero en amplitud.

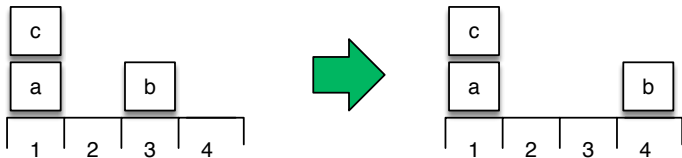


Solución: forzar planes cortos primero

```
1 plan_primero_amplitud(Edo, Metas, Plan, EdoFinal) :-  
2     candidato(Plan),  
3     plan(Edo, Metas, Plan, EdoFinal).  
4  
5 candidato([]).  
6  
7 candidato([Primero|Resto]) :-  
8     candidato(Resto).
```



Corrida



- 1 ?- estado(E), plan_primero_amplitud(E,[libre(2), libre(3)],P,_).
- 2 E = [libre(2), libre(4), libre(b), libre(c), en(a, 1), en(b, 3), en(c,
↪ a)],
- 3 P = [mover(b, 3, 4)]



O de una forma más elegante

```
121 plan_metas_protegidas_amplitud(EdoInicial, Metas, Plan, EdoFinal) :-
122     plan_mp_amplitud(EdoInicial, Metas, [], Plan, EdoFinal).
123
124 plan_mp_amplitud(Edo, Metas, _, [], Edo) :-
125     metas_satisfechas(Edo, Metas).
126
127 plan_mp_amplitud(Edo, Metas, Protegido, Plan, EdoFinal) :-
128     append(Plan, _, _),
129     append(PrePlan, [Accion|PostPlan], Plan),
130     seleccionar(Edo, Metas, Meta),
131     lograr(Accion, Meta),
132     precondition(Accion, Condicion),
133     preservar(Accion, Protegido),
134     plan_mp_amplitud(Edo, Condicion, Protegido, PrePlan, EdoInter1),
135     aplicar(EdoInter1, Accion, EdoInter2),
136     plan_mp_amplitud(EdoInter2, Metas, [Meta|Protegido], PostPlan, EdoFinal).
```



La elegancia cuesta

```

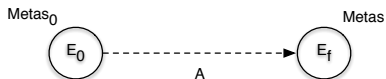
1  ?- estado(E),time(plan(E,[en(a,b),en(b,c)],P,_)).
2  % 6,949 inferences, 0.001 CPU in 0.001 seconds (99% CPU, 6656130 Lips)
3  E = [ libre(2), libre(4), libre(b), libre(c), en(a, 1), en(b, 3),en(c, a)
4  ↪ ],
5  P = [ mover(b, 3, c), mover(b, c, 3), mover(c, a, 2), mover(a, 1, b),
6  mover(a, b, 1), mover(b, 3, c), mover(a, 1, b)]
7  ?- estado(E),time(plan_metas_protegidas_amplitud(E,[en(a,b),en(b,c)],P,_)).
8  % 192,665 inferences, 0.030 CPU in 0.030 seconds (100% CPU, 6337039 Lips)
9  E = [ libre(2), libre(4), libre(b), libre(c), en(a, 1), en(b, 3),
10 en(c, a) ],
11 P = [ mover(c, a, 2), mover(b, 3, a), mover(b, a, c),
12 mover(a, 1, b) ]

```

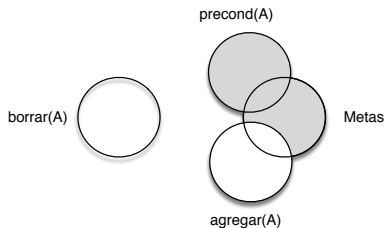


Graficamente

- ▶ ¿Qué $Metas_0$ deben **considerarse** en el estado E_0 , para que las $Metas$ sean verdaderas en el estado E_f , dada la acción A ?



- ▶ Propiedades de $Metas_0$:



Propiedades de E_0

- ▶ La acción A debe ser posible en E_0 , por lo que $Metas_0$ debe incluir las **condiciones** necesarias para ejecutar A .
- ▶ Para cada **meta** $M \in Metas$, se cumple que:
 - ▶ La acción A agrega M ; ó
 - ▶ $M \in Metas_0$ y A no borra M .



Algoritmo

- ▶ Si *Metas* se cumple en E_0 , entonces el plan vacío es suficiente;
- ▶ En cualquier otro caso, seleccionar una meta $M \in \text{Metas}$ y una acción A que agregue M ; entonces computar la regresión de *Metas* vía A obteniendo así *NuevasMetas* y buscar un plan para satisfacer *NuevasMetas* desde E_0 .
- ▶ Algunas combinaciones de metas son **imposibles**.



Regresión de metas en Prolog I

```
44 %%% metas incompatibles
45
46 imposible(en(X,X),_).
47
48 imposible(en(X,Y), Metas) :-
49     member(despejado(Y),Metas)
50     ;
51     member(en(X,Y1),Metas), Y1 \== Y
52     ;
53     member(en(X1,Y),Metas), X1 \== X.
54
55 imposible(despejado(X),Metas) :-
56     member(en(_,X),Metas).
57
58 %%% Planeador medios fines con regresión de metas
59
60 plan(Estado, Metas, []) :-
61     satisfecho(Estado, Metas).
62
63 plan(Estado, Metas, Plan) :-
64     append( PrePlan, [Accion], Plan),
```



Regresión de metas en Prolog II

```
65  seleccionar( Estado, Metas, Meta),
66  lograr(Accion, Meta),
67  precond(Accion, Condicion),
68  preservar(Accion, Metas),
69  regresion(Metas, Accion, MetasReg),
70  plan(Estado, MetasReg, PrePlan).
71
72  satisfecho(Estado, Metas) :-
73    borrar_todos(Metas, Estado, []).
74
75  seleccionar(_, Metas, Meta) :-
76    member( Meta, Metas).
77
78  lograr( Accion, Meta) :-
79    agregar( Accion, Metas),
80    member( Meta, Metas).
81
82  borrar_todos( [], _, []).
83
84  borrar_todos( [X | L1], L2, Diff) :-
85    member( X, L2), !,
```



Regresión de metas en Prolog III

```
86   borrar_todos( L1, L2, Diff).
87
88   borrar_todos( [X | L1], L2, [X | Diff]) :-
89     borrar_todos( L1, L2, Diff).
90
91   preservar(Accion, Metas) :-
92     borrar(Accion, ListaBorrar),
93     not( (member(Meta, ListaBorrar),
94          member(Meta, Metas))).
95
96   regresion(Metas, Accion, MetasReg) :-
97     agregar(Accion, NuevasRels),
98     borrar_todos(Metas, NuevasRels, RestoMetas),
99     precondition(Accion, Condicion),
100    agregarNuevo(Condicion, RestoMetas, MetasReg).
101
102    agregarNuevo([], L, L).
103
104    agregarNuevo([Meta|_], Metas, _) :-
105      imposible(Meta, Metas),
106      !,
```

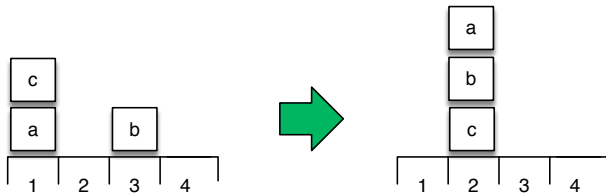


Regresión de metas en Prolog IV

```
107 fail.  
108  
109 agregarNuevo([X|L1],L2,L3) :-  
110     member(X,L2), !,  
111     agregarNuevo(L1,L2,L3).  
112  
113 agregarNuevo([X|L1],L2,[X|L3]) :-  
114     agregarNuevo(L1,L2,L3).
```



Consulta con regresión de metas



```

1  ?- estado(E), time(plan(E,[en(a,b),en(b,c)],P)).
2  % 35,280 inferences, 0.005 CPU in 0.005 seconds (100% CPU, 6739255 Lips)
3  E = [ libre(2), libre(4), libre(b), libre(c), en(a, 1), en(b, 3),en(c, a) ],
4  P = [ mover(c, a, 2), mover(b, 3, c), mover(a, 1, b) ]

```



¿Donde considerar una heurística?

En *seleccionar*(*Edo, Metas, Meta*). Cimentación, la relación $en/2$ más arriba de la torre, debería resolverse al último; Postergar las metas que ya se cumplen en el medio ambiente.

En *lograr*(*Accion, Meta*). O al procesar $precond/2$. Algunas acciones son “mejores” porque satisfacen más de una meta simultáneamente; Ciertas condiciones son más fáciles de satisfacer que otras.

En el conjunto de *regresión*. Seguir trabajando en el que parezca más fácil de resolver, por ejemplo, el más pequeño.



Solución adoptada

- ▶ Computar un **estimado heurístico** de la dificultad de los diversos conjuntos de regresión de metas alternativos, para expandir el más promisorio.
- ▶ **Ejemplo:** El tamaño del conjunto.
- ▶ Utilizar la heurística con el buscador **primero el mejor!**



Requisitos técnicos

- ▶ Una relación de **costo** $s/3$ entre nodos del espacio de búsqueda: $s(Nodo_1, Nodo_2, Costo)$.
- ▶ Los nodos **meta** en el espacio: $meta(Nodo)$.
- ▶ Una **función heurística** de la forma $h(Nodo, H_{estimado})$.
- ▶ El **nodo inicial** de la búsqueda.

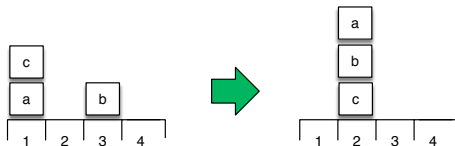


Implementación (más elaborada)

```
1 :- op(300,xfy, ->).
2
3 s(Metas -> AccSiguiente, MetasNuevas -> Accion, 1) :-
4     member(Meta, Metas),
5     lograr(Accion, Meta),
6     precondition(Accion, Cond),
7     preservar(Accion, Metas),
8     regression(Metas, Accion, MetasNuevas).
9
10 meta(Metas -> Accion) :-
11     inicio(Edo),
12     metas_logradas(Edo, Metas).
13
14 h(Metas -> Accion, H) :-
15     inicio(Edo),
16     borrar_todos(Metas, Edo, Insatisfecho),
17     length(Insatisfecho, H).
18
19 inicio([ en(a,1), en(b,3), en(c,a), libre(b), libre(c), libre(2),
20         libre(4) ]).
```



Corrida



```

1  ?- time(primerMejor([en(a,b), en(b,c)] -> stop, P)), imprime_plan(P).
2  % 10,699 inferences, 0.002 CPU in 0.002 seconds (97% CPU, 5767655 Lips)
3  mover(c,a,2), mover(b,3,c), mover(a,1,b), stop.
4  P = [[despejado(2), en(c, a), despejado(c), en(b, 3), despejado(b),
5  en(a, 1)]->mover(c, a, 2), [despejado(c), en(b, 3), despejado(a),
6  despejado(b), en(a, 1)]->mover(b, 3, c), [despejado(a), despejado(b),
7  en(a, 1), en(b, c)]->mover(a, 1, b), [en(a, b), en(b, c)]->stop]

```



Variables siempre instanciadas

- ▶ Ej. La relación *precond/2*, cuyo cuerpo incluye la meta *bloque(Bloque)*.
- ▶ Este tipo de metas hace que *Bloque* siempre esté instanciada.
- ▶ Esto puede llevar a la generación de numerosos movimientos alternativos **irrelevantes**.



Ejemplo

- ▶ La meta *despejar(a)* utiliza *lograr/2* para generar movimientos que *satisfagan libre(a)*:

```
1 mover(De,a,A)
```

- ▶ Se computan las *condiciones* necesarias para ejecutar esta acción:

```
1 precond(mover(De,a,A),Cond) :- bloque(De), ...
```

- ▶ Lo cual fuerza, al *reconsiderar*, varias instancias alternativas para *De* y *A*:

```
1 mover(b,a,1) mover(b,a,2) mover(b,a,3) mover(b,a,4) mover(b,a,c)
```

```
2 mover(b,a,1) mover(b,a,2)
```



Idea

- ▶ Para hacer más eficiente este paso del planeador, es posible permitir **variables no instanciadas** en las metas.
- ▶ **Ejemplo:** Las condiciones de *mover* serían definidas como:
 - 1 `precond(mover(Bloque,De,A), [libre(Bloque),libre(A),en(Bloque,De)])`.
- ▶ Si **reconsideramos** con esta nueva definición, la lista de condiciones computadas sería:
 - 1 `[libre(Bloque),libre(A),en(Bloque,A)]`
- ▶ Con **solución** directa: *Bloque/c* y *A/2*.



Costo

- ▶ El precio a pagar es una mayor **complejidad**.
- ▶ Para empezar, nuestro intento por definir *precond* para *mover/3* es **erróneo**, pues permite movimientos como *mover(c, a, c)*.
- ▶ Esto podría arreglarse si especificáramos que *De* y *A* deben ser **diferentes**:

```

1  precond(mover(Bloque,De,A),
2         [despejado(Bloque),despejado(A),en(Bloque,De), diferente(Bloque,A),
3         diferente(De,A), diferente(Bloque,De)]).
```

- ▶ donde *diferente/2* significa que los dos argumentos **no denotan al mismo objeto** Prolog.



Implementación

- ▶ Una manera de manejar estas pseudo-metas es agregar al predicado *metas_logradas/2* la siguiente cláusula:

```
1 metas_logradas(Estado, [Meta|Metas]) :-  
2   satisface(Meta),  
3   metas_logradas(Estado, Metas).
```

- ▶ De forma que debemos definir también:

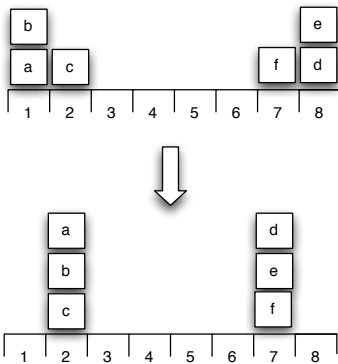
```
1 satisface(diferente(X,Y))
```

- ▶ Si X y Y no unifican al mismo objeto, la relación tiene **éxito**.
- ▶ Si X y Y son iguales, la relación **falla** y debería tratarse como una meta imposible.
- ▶ En otro caso, estamos ante falta de información y *satisface* se debería **postergar** hasta que ambas variables estén instanciadas.



El problema

- ▶ Considerar **todos los posibles ordenes** de las acciones, aún cuando las acciones son completamente **independientes**.



Independencia

- ▶ Las dos pilas puede construirse **independientemente** con los siguientes planes:

1 $\text{Plan1} = [\text{mover}(b,a,c), \text{mover}(a,1,b)]$

2 $\text{Plan2} = [\text{mover}(e,d,f), \text{mover}(d,8,e)]$

- ▶ Estos planes **no interaccionan** entre ellos, de forma que el orden de las acciones sólo es relevante dentro de cada plan.
- ▶ Tampoco importa si se ejecuta primero *Plan1* o *Plan2* y es incluso posible ejecutarlos de manera **alternada**, por ejemplo:

1 $[\text{mover}(b,a,c), \text{mover}(e,d,f), \text{mover}(d,8,e), \text{mover}(a,1,b)]$



Planeadores no lineales

- ▶ Sin embargo, nuestro planeador considerará las 24 **permutaciones** posibles de las cuatro acciones, aunque existan solo 4 alternativas: 2 permutaciones para cada uno de los planes.
- ▶ El problema se debe a que el planeador insiste en el orden **total** de las acciones en el plan.
- ▶ Una mejora se lograría si, en los casos donde el orden no es importante, la precedencia entre las acciones se mantiene indefinida, i.e., **planes parcialmente ordenados**.
- ▶ Los planeadores que aceptan este tipo de representación se conocen como planeadores **no lineales**.



Hacia una solución

- ▶ Analizando las metas $en(a, b)$ y $en(b, c)$ el planeador no lineal concluye que las siguientes dos acciones son necesarias en el plan:

1 M1 = mover(a, X, b)

2 M2 = mover(b, Y, c)

- ▶ No hay otra forma de resolver ambas metas, pero el orden de estas acciones es aún **indeterminado**.



Un poco de orden

- ▶ Ahora consideren las **condiciones** de ambas acciones.
- ▶ La condición de $mover(a, X, b)$ incluye $libre(a)$, la cual **no se satisface** en la situación inicial, por lo que necesitamos una acción de la forma:

1 $M3 = mover(\text{Bloque}, a, A)$.

que precede a $M1$.

- ▶ Ahora tenemos una **restricción** en el orden de las acciones:

1 $antes(M3, M1)$



Solución

- ▶ Ahora revisamos si $M3$ y $M1$ pueden ser el mismo movimiento.
- ▶ Como este no es el caso, el plan tendrá tres movimientos.
- ▶ ¿Hay una permutación de $[M1, M2, M3]$ tal que $M3$ preceda a $M1$ y que tal permutación sea ejecutable en el estado inicial del problema y las metas se cumplan en el estado resultante?
- ▶ Dadas las restricciones de orden anteriores, tres permutaciones de seis, cumplen con los requisitos temporales:

1 $[M3, M1, M2]$

2 $[M3, M2, M1]$

3 $[M2, M3, M1]$

- ▶ Solo la segunda cumple con el requisito de ser ejecutable bajo la sustitución $Bloque/c, A/2, X/1, Y/3$.



Comentario final

- ▶ Como se puede intuir, la complejidad computacional no puede ser evitada del todo por un planeador no lineal, pero puede ser **aliviada** considerablemente.
- ▶ El capítulo 18 de libro de Bratko [1] implementa un planificador no lineal.



Referencias I

- [1] I Bratko. *Prolog programming for Artificial Intelligence*. Addison-Wesley, 2001.
- [2] RE Fikes y NJ Nilsson. "STRIPS: a new approach to the application of theorem proving to problem solving". En: *Artificial Intelligence 2* (1971), págs. 189-208.
- [3] M Ghallab, D Nau y P Traverso. *Automated planning: theory and practice*. San Francisco, CA, USA: Morgan Kaufmann, 2004.
- [4] N Gupta y DS Nau. "On the complexity of blocks world planning". En: *Artificial Intelligence 56.2-3* (1992), págs. 223 -254.

