

Metodología de Programación I Planeación

Dr. Alejandro Guerra-Hernández

Departamento de Inteligencia Artificial
Facultad de Física e Inteligencia Artificial
aguerra@uv.mx
<http://www.uv.mx/aguerra>

Maestría en Inteligencia Artificial 2011



Universidad Veracruzana

Planear

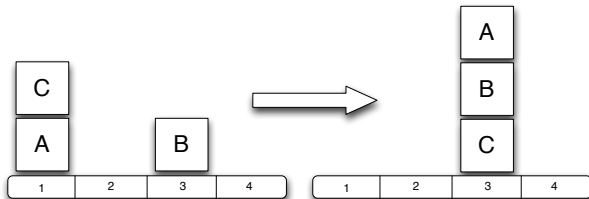
- ▶ Razonar **explícitamente** acerca de los efectos de las acciones y la secuencia en que estas se aplican para lograr un efecto acumulativo dado.



Universidad Veracruzana

Representación : Un escenario conocido

- ▶ Para el caso del mundo de los bloques, tales relaciones son *en/2* y *libre/1* con la semántica intuitiva.



Estado1 = [*libre*(2), *libre*(4), *libre*(c), *libre*(b),
en(a, 1), *en*(b, 3), *en*(c, a)]



Acciones

- ▶ **Precondición.** Las condiciones que debe satisfacerse, para que la acción pueda ejecutarse.
- ▶ **Agregar.** Es una lista de observaciones que, se espera, ocurran después de ejecutarse la acción.
- ▶ **Borrar.** Es una lista de observaciones que, se espera, dejen de ser verdaderas después de ejecutarse la acción.



Universidad Veracruzana

Ejemplo: mover/3

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

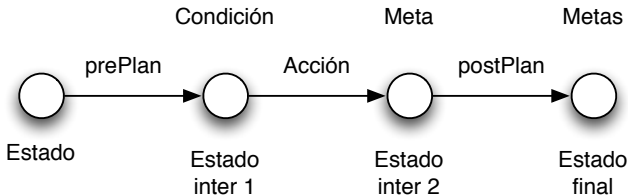


Ejemplo: estado y metas

```
1 objeto(X) :-
2     lugar(X) ; bloque(X).
3
4 bloque(a).
5 bloque(b).
6 bloque(c).
7
8 lugar(1).
9 lugar(2).
10 lugar(3).
11 lugar(4).
12
13 estado1( [ despejado(2), despejado(4),
14           despejado(b), despejado(c),
15           en(a,1), en(b,3), en(c,a) ] ).
16
17 metas1([en(a, b)]).
```



Análisis Medios-Fines: Idea Intuitiva



Análisis Medios-Fines: Algoritmo

- ▶ Si todas las *Metas* son verdaderas en $Estado_0$, entonces $Estado_f = Estado_0$. 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 *can* para obtener el estado intermedio $Estado_1$.
 4. Aplicar la *Accion* en el estado $Estado_1$ para obtener el estado intermedio $Estado_2$ donde *Meta* se cumple.
 5. Resolver *Metas* en el estado $Estado_2$ para llegar a $Estado_f$.



Plan/4

```
1  plan( Estado, Metas, [], Estado) :-
2      satisfecho( Estado, Metas).
3
4  plan( Estado, Metas, Plan, EstadoFinal) :-
5      append( PrePlan, [Accion | PostPlan], Plan
6          ),
7      seleccionar( Estado, Metas, Meta),
8      lograr( Accion, Meta),
9      precond( Accion, Condicion),
10     plan( Estado, Condicion, PrePlan,
11         EstadoInter1),
12     aplicar( EstadoInter1, Accion,
13         EstadoInter2),
14     plan( EstadoInter2, Metas, PostPlan,
15         EstadoFinal).
```



satisfecho/2

```
1  satisfecho( _, []).  
2  
3  satisfecho( Estado, [Meta | Metas]) :-  
4      member( Meta, Estado),  
5      satisfecho( Estado, Metas).
```



seleccionar/3 y lograr/2

```
1 seleccionar( Estado, Metas, Meta) :-  
2     member( Meta, Metas),  
3     not(member( Meta, Estado)).  
4  
5 lograr( Accion, Meta) :-  
6     agregar( Accion, Metas),  
7     member( Meta, Metas).
```



borrar_todos/3

```
1 borrar_todos( [], _, []).
2
3 borrar_todos( [X | L1], L2, Diff) :-
4     member( X, L2), !,
5     borrar_todos( L1, L2, Diff).
6
7 borrar_todos( [X | L1], L2, [X | Diff]) :-
8     borrar_todos( L1, L2, Diff).
```



Corrida

```
1  ?- estado1(E), metas1(M), plan(E,M,Plan,EstadoF).
2  E = [despejado(2), despejado(4), despejado(b),
        despejado(c), en(a, 1), en(b, 3), en(c, a)],
3  M = [en(a, b)],
4  Plan = [mover(c, a, 2), mover(a, 1, b)],
5  EstadoF = [en(a, b), despejado(1), en(c, 2),
              despejado(a), despejado(4), despejado(c), en(b
              , 3)]
```



Explosión combinatoria

- ▶ El mundo de los bloques es mucho más complejo de lo que parece.
- ▶ 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.



Universidad Veracruzana

Ejemplo: plan más complicado

```
1  ?- estado1(E), plan(E,[en(a,b), en(b,c)],Plan
   ,EstadoF).
2  E = [despejado(2), despejado(4), despejado(b)
   , despejado(c), en(a, 1), en(b, 3), en(c,
   a)],
3  Plan = [mover(b, 3, c), mover(b, c, 3),
4          mover(c, a, 2), mover(a, 1, b),
5          mover(a, b, 1), mover(b, 3, c),
6          mover(a, 1, b)],
7  EstadoF = [en(a, b), despejado(1), en(b, c),
   despejado(3), en(c, 2), despejado(a),
   despejado(4)]
```



¿QUÉ está pasando?

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 *on(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



Ejemplo: catastrofe

```
1  ?- estado1(E), plan(E,[despejado(2),
   despejado(3)],Plan,EstadoF).
2  ERROR: Out of global stack
```



Planeador con metas protegidas

```
1  plan_metas_protegidas(EstadoInicial, Metas,
   Plan, EstadoFinal):-
2  plan_mp(EstadoInicial, Metas, [], Plan,
   EstadoFinal).
3
4  plan_mp(Estado, Metas, _, [], Estado) :-
5  satisfecho(Estado, Metas).
6
7  plan_mp(Estado, Metas, Protegido, Plan,
   EstadoFinal) :-
8  append( PrePlan, [Accion | PostPlan], Plan),
9  seleccionar( Estado, Metas, Meta),
10 lograr( Accion, Meta),
11 precond( Accion, Condicion),
12 preservar(Accion, Protegido),
13 plan_mp( Estado, Condicion, Protegido,
   PrePlan,
14           EstadoInter1),
15 aplicar( EstadoInter1, Accion, EstadoInter2),
16 plan_mp( EstadoInter2, Metas, [Meta|Protegido
   ],
17           PostPlan, EstadoFinal).
```



preservar/2

```
1 preservar(Accion, Metas) :-  
2 borrar(Accion, ListaBorrar),  
3 not( (member(Meta, ListaBorrar),  
4      member(Meta, Metas))).
```



Nueva corrida

- ▶ Ahora se puede ejecutar la consulta:

```
1  ?- estado1(E), plan_metas_protegidas(E,[despejado(2),
    despejado(3)],Plan,EstadoF).
2  E = [despejado(2), despejado(4), despejado(b),
    despejado(c), en(a, 1), en(b, 3), en(c, a)],
3  Plan = [mover(b, 3, 2), mover(b, 2, 4)],
4  EstadoF = [en(b, 4), despejado(2), despejado(3),
    despejado(b), despejado(c), en(a, 1), en(c, a)]
```

- ▶ 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. *append/3* genera al reconsiderar, candidatos alternativos para *PrePlan* en el siguiente orden:

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



Aspectos procedurales 2

- ▶ *PrePlan* establece una condición para *Accion*. Esto permite encontrar una acción cuya condición puede satisfacerse por un plan tan corto como sea posible.
- ▶ *PostPlan* está totalmente no 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_primeros_amplitud(Estado, Metas, Plan,
2     EstadoFinal) :-
3     candidato(Plan),
4     plan(Estado, Metas, Plan, EstadoFinal).
5
6  candidato([]).
7
8  candidato([Primero|Resto]) :-
9     candidato(Resto).
```



Corrida

```
1  ?- estado1(E), plan_primero_amplitud(E,[despejado(2),
    despejado(3)],Plan,EstadoF).
2  E = [despejado(2), despejado(4), despejado(b),
    despejado(c), en(a, 1), en(b, 3), en(c, a)],
3  Plan = [mover(b, 3, 4)],
4  EstadoF = [en(b, 4), despejado(3), despejado(2),
    despejado(b), despejado(c), en(a, 1), en(c, a)]
```



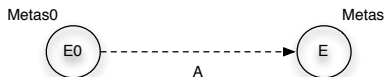
O de una forma más elegante

```
1  plan_metas_protegidas_amplitud(EstadoInicial ,
    Metas,Plan,EstadoFinal):-
2  plan_mp_amplitud(EstadoInicial,Metas,[],Plan,
    EstadoFinal).
3
4  plan_mp_amplitud(Estado,Metas,_,[],Estado) :-
5  satisfecho(Estado,Metas).
6
7  plan_mp_amplitud(Estado,Metas,Protegido,Plan,
    EstadoFinal) :-
8  append(Plan,_,_),
9  append( PrePlan, [Accion | PostPlan], Plan),
10 seleccionar( Estado, Metas, Meta),
11 lograr( Accion, Meta),
12 precond( Accion, Condicion),
13 preservar(Accion,Protegido),
14 plan_mp_amplitud( Estado, Condicion,
    Protegido, PrePlan, EstadoInter1),
15 aplicar( EstadoInter1, Accion, EstadoInter2),
16 plan_mp_amplitud( EstadoInter2, Metas, [Meta|
    Protegido], PostPlan, EstadoFinal).
```

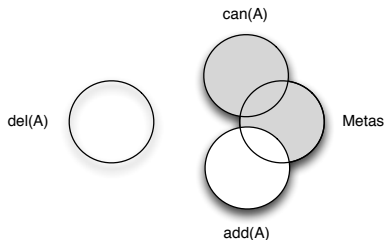


Graficamente

- ▶ Qué metas $Metas_0$ deben ser verdaderas en S_0 para que $Metas$ sea verdadero en S , dado A ?



- ▶ Propiedades de $Metas_0$:



Propiedades de E_0

- ▶ La acción A debe ser posible en E_0 , por lo que $Metas_0$ debe implicar la condición para A .
- ▶ Para cada meta M en $Metas$, se cumple que:
 - ▶ la acción A agrega M ; ó
 - ▶ $M \in Metas_0$ y A no borra M .



Algoritmo

- ▶ Si *Metas* se cumple en $Estado_0$, entonces el plan vacío es suficiente;
- ▶ En cualquier otro caso, seleccionar una meta $M \in Metas$ 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 $Estado_0$.
- ▶ Algunas combinaciones de metas son imposibles.



imposible/2

```
1  imposible(en(X,X),_).
2
3  imposible(en(X,Y), Metas) :-
4      member(despejado(Y),Metas)
5      ;
6      member(en(X,Y1),Metas), Y1 \== Y
7      ;
8      member(en(X1,Y),Metas) X1 \== X.
9
10 imposible(despejado(X),Metas) :-
11     member(en(_,X),Metas).
```



plan/3

```
1  plan(Estado, Metas, []) :-
2      satisfecho(Estado, Metas).
3
4  plan(Estado, Metas, Plan) :-
5      append( PrePlan, [Accion], Plan),
6      seleccionar( Estado, Metas, Meta),
7      lograr(Accion, Meta),
8      precond(Accion, Condicion),
9      preservar(Accion, Metas),
10     regresion(Metas, Accion, MetasReg),
11     plan(Estado, MetasReg, PrePlan).
```



satisfecho/2, seleccionar/3 y lograr/2

```
1  satisfecho(Estado, Metas) :-  
2      borrar_todos(Metas, Estado, []).  
3  
4  seleccionar(_, Metas, Meta) :-  
5      member( Meta, Metas).  
6  
7  lograr( Accion, Meta) :-  
8      agregar( Accion, Metas),  
9      member( Meta, Metas).
```



preservar/2 y regresion/3

```
1  preservar(Accion, Metas) :-
2      borrar(Accion, ListaBorrar),
3      not( (member(Meta, ListaBorrar),
4            member(Meta, Metas))).
5
6  regresion(Metas, Accion, MetasReg) :-
7      agregar(Accion, NuevasRels),
8      borrar_todos(Metas, NuevasRels, RestoMetas),
9      precond(Accion, Condicion),
10     agregarNuevo(Condicion, RestoMetas, MetasReg).
```



agregarNuevo/3

```
1  agregarNuevo([],L,L).
2
3  agregarNuevo([Meta|_],Metas,_) :-
4      imposible(Meta,Metas),
5      !,
6      fail.
7
8  agregarNuevo([X|L1],L2,L3) :-
9      member(X,L2), !,
10     agregarNuevo(L1,L2,L3).
11
12  agregarNuevo([X|L1],L2,[X|L3]) :-
13     agregarNuevo(L1,L2,L3).
```



Corrida

```
1  ?- estado1(E), plan(E,[en(a,b),en(b,c)],P).  
2  E = [despejado(2), despejado(4), despejado(b),  
3      despejado(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(Estado, 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 de metas debe considerarse a continuación. Seguir trabajando en el que parezca más fácil de resolver, buscando así el plan más corto.



Requisitos técnicos

- ▶ Una relación $s/3$ 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      :-
5      member(Meta, Metas),
6      lograr(Accion, Meta),
7      precond(Accion, Cond),
8      preservar(Accion, Metas),
9      regresion(Metas, Accion, MetasNuevas).
10
11 meta(Metas -> Accion) :-
12     inicio(Estado),
13     satisfecho(Estado, Metas).
14
15 h(Metas -> Accion, H) :-
16     inicio(Estado),
17     borrar_todos(Metas, Estado, Insatisfecho),
18     length(Instatisfecho, H).
19
20 inicio([en(a,1), en(b,3), en(c,a), despejado(b),
21         despejado(c), despejado(2), despejado(4)]).
```



Corrida

```
1  ?- primeroMejor([en(a,b), en(b,c)] -> stop, Plan).
2  Plan = [[despejado(2), en(c, a), despejado(c), en(b, 3),
3          despejado(b), en(a, 1)]->mover(c, a, 2),
4          [despejado(c), en(b, 3), despejado(a), despejado(b),
5          en(a, 1)]->mover(b, 3, c),
6          [despejado(a), despejado(b), en(a, 1), en(b, c)]
7          ->mover(a, 1, b),
8          [en(a, b), en(b, c)]->stop]
```

