Inteligencia Artificial
Planeamiento: Strips

Diego Marcovecchio (LU: 83815) Tomás Touceda (LU: 84024)

24 de Noviembre de 2010

Índice general

Capítulo 1

Introducción

1.1. Descripción

Este proyecto consiste en la implementación de un planificador STRIPS para un Mundo de Bloques similar al utilizado para los trabajos prácticos durante el cuatrimestre. El mundo posee un estado inicial fijo en el que todos los bloques se encuentran apoyados sobre la mesa y sin otros bloques encima de ellos; es decir, el estado inicial del mundo está definido con:

```
estadoInicial([
        libre(A),
        libre(B),
        libre(C),
        enMesa(A),
        enMesa(B),
        enMesa(C)
])
```

El objetivo del planificador es: dado un conjunto de metas, encontrar una secuencia de acciones que permita alcanzar todas ellas en un estado final.

1.2. Modularización

Dada la sencillez del planificador, a diferencia de los proyectos anteriores, toda la implementación del proyecto se encuentra en el archivo *strips.pl*. Más adelante en el informe se detallará cada uno de los predicados utilizados.

Capítulo 2

Estado del mundo

2.1. Relaciones entre bloques

El estado del mundo se describe mediante un conjunto de predicados que denotan relaciones entre bloques. Dichas relaciones son:

- *sobre(A, B)*: indica que el bloque A está inmediatamente arriba del bloque B.
- *libre(A)*: indica que el bloque A no tiene ningún bloque encima.
- *enMesa(A)*: es verdadera cuando el bloque A está apoyada encima de la mesa (es decir, no hay ningún bloque *debajo* de A).

2.2. Acciones

Las únicas acciones que pueden realizarse en este mundo son las acciones de apilar, y desapilar bloques. Presentamos, a continuación, el formato de dichas acciones en la notación de STRIPS:

- *apilar(A, B)*: coloca al bloque A inmediatamente arriba del bloque B

  Precondiciones

  ```
  [libre(A), libre(B), enMesa(A)]
  ```

  Add_List

  ```
  [sobre(A, B)]
  ```

  Del_List

  ```
  [libre(B), enMesa(A)]
  ```

- *desapilar(A, B)*: coloca el bloque A que está encima del bloque B sobre la mesa. Precondiciones

  ```
  [sobre(A, B), libre(A)]
  ```

  Add_list

  ```
  [enMesa(A), libre(B)]
  ```

  Del_list

  ```
  [sobre(A, B)]
  ```

Capítulo 3

Estrategia del planeador

3.1. Decisiones de diseño

Es importante destacar que el algoritmo trabaja realcanzando las metas que figuran en la delete_list de cada acción, en lugar de ordenar las acciones para proteger las metas. Ésta decisión fue tomada debido a que no siempre es posible reordenar las metas para evitar los conflictos. A partir de esta decisión, también podemos concluir que el plan encontrado no necesariamente será minimal (pues para ello puede ser necesario realizar dicho reordenamiento).

3.2. Algoritmo general

El planeador está basado en la estrategia presentada en la sección «The STRIPS Planner» del capítulo 8 del libro *Computational Intelligence: a logical approach* (Poole, Mackworth & Goebel). Se corrigieron algunos errores del pseudocódigo allí presentado, y se pasó a utilizar un parámetro más en el predicado *achieve_all/4* para representar las metas que ya se cumplieron. De esta manera, el algoritmo general del planeador es:

- Obtener la primer meta de la lista de metas pendientes.
- Analizar el conjunto de acciones necesarias para alcanzar dicha meta.
- Observar cuáles de las metas alcanzadas previamente fueron deshechas por el segundo paso, y volver a agregarlas a la lista de metas pendientes.
- Repetir el algoritmo, quitando de la lista de metas pendientes a todas aquellas que hayan sido alcanzadas por la acción realizada.

Capítulo 4

Implementación

4.1. Descripción de los predicados

En esta sección mostraremos todos los predicados utilizados en la implementación, su signatura, y una descripción para cada uno de ellos.

- *plan(+Metas, -Plan)*: es el predicado principal del planeador, respetando la convención impuesta por la cátedra. Dado un conjunto de metas, devuelve una planificación según el estado inicial establecido.
- *undone(+Done, +W, -Undone)*: Dado un conjunto de metas que se creen ya realizadas *Done* en el mundo *W*, *Undone* se unifica con las metas de *Done* que no se mantienen en *W*.
- *achieve_all(+Goals, +DoneGoals, +W0, -W2)*: *W2* es el mundo que resulta luego de cumplir cada meta especificada en *Goals* en el mundo *W1*. *DoneGoals* es utilizado para saber qué metas ya se cumplieron, y saber cuáles rehacer en caso que ya no se cumplan en mundos futuros. Si no hay metas para cumplir, el mundo se mantiene igual. Si hay metas a cumplir, se selecciona una del conjunto, y se obtiene un conjunto de acciones para alcanzarla. Si cumplir la meta actual resulta en que metas anteriores se deshagan, dichas metas se eliminan de DoneGoals, y se agregan al final de la lista de metas a realizar.
- *achieve(+Goal, +World1, -World2)*: *World2* es el mundo que resulta de cumplir una única meta *Goal* en el mundo *World1*. Si la meta ya se cumple en el mundo actual, no se registra ningún cambio.
- *holds(+C, +W)*: es verdadero si la relación *C* se cumple en el mundo *W*.
- *holdsall(+Metas, +W)*: es verdadero si cada una de las metas en *Metas* se cumple en el mundo *W*.
- *notin(+C, +DL)*: predicado auxiliar. Es verdadero si *C* no es miembro de la lista *DL*.
- *remove(-G, +Goals, -Rem_Gs)*: Selecciona la primer meta de *Goals* en *G*, y el resto de las metas en *Rem_Gs*.
- *achieves(+Action, +Goal)*: verdadero si la meta *Goal* es parte de la add_list de *Action*.
- *translate(+World, -Plan)*: traduce un mundo *World* a una secuencia de acciones en *Plan*; es decir, un mundo *do(Action, do(Action2, ...))* resulta en una secuencia *[Action, Action2, ...]*.

4.2. Código

Por último, detallaremos a continuación el código del planeador:

```
% Se declara dinamico el predicado estadoInicial para poder realizar
    pruebas de
% forma mas comoda
:- dynamic estadoInicial/1.

% Estado inicial definido en el enunciado
estadoInicial([sobre(a, c), libre(a), enMesa(c), enMesa(b), libre(b)]).

% Accion: apilar
preconditions(apilar(A, B), [libre(A), libre(B), enMesa(A)]).
add_list(apilar(A, B), [sobre(A, B)]).
del_list(apilar(A, B), [libre(B), enMesa(A)]).

% Accion: desapilar
preconditions(desapilar(A, B), [sobre(A, B), libre(A)]).
add_list(desapilar(A, B), [enMesa(A), libre(B)]).
del_list(desapilar(A, B), [sobre(A, B)]).

% plan(+Metas, -Plan)
% Dadas las Metas devuelve una planificacion segun el estadoInicial
    establecido
plan(Metas, Plan):-
  estadoInicial(L),
  achieve_all(Metas, [], L, DoPlan),
  translate(DoPlan, Plan).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                PLANNER
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% undone(+Done, +W, -Undone)
% Dadas las metas que se cree que ya estan hechas (Done) en el mundo W, se
% devuelven en Undone las metas de Done que no se mantienen en W.
undone([], _, []).
undone([Done|Dones], W, Undone):-
  holds(Done, W),
  undone(Dones, W, Undone), !.
undone([Done|Dones], W, [Done|Undone]):-
  undone(Dones, W, Undone).

% achieve_all(+Goals, +DoneGoals, +W0, -W2)
% W2 es el mundo que resulta despues de cumplir cada meta en Goals en el
    mundo
% W1.
% DoneGoals es utilizado para saber que metas ya se cumplieron, y saber
    cuales
% rehacer en caso que ya no se cumplan en mundos futuros.
%
% Si no hay metas a cumplir, el mundo se mantiene igual
achieve_all([], _, W0, W0).
% Si hay metas a cumplir, seleccionar una del conjunto y cumplirla.
% Si cumplir la meta actual resulta en que metas cumplidas con anterioridad
```

```
    se
% deshagan, sacar las metas deshechas de DoneGoals, y agregarlas al final
    del
% resto de las metas a realizar.
achieve_all(Goals, DoneGoals, W0, W2):-
  remove(G, Goals, Rem_Gs),
  achieve(G, W0, W1),
  undone(DoneGoals, W1, Undone),
  append(Rem_Gs, Undone, Total_Gs),
  subtract(DoneGoals, Undone, ReallyDone),
  achieve_all(Total_Gs, [G|ReallyDone], W1, W2).

% achieve(+Goal, +World1, -World2)
% World2 es el mundo resultante luego de cumplir la meta Goal en el mundo
% World1.
%
% Si la meta ya se cumple en el mundo actual, no se hace nada y el mundo se
% mantiene.
achieve(G, W, W):-
  holds(G, W).

% Si la meta es una es de la forma G:-B. donde B es una disyuncion de
    clausulas,
% cumplir B, para que por consecuencia se cumpla G.
achieve(G, W0, W1):-
  clause(G, B),
  achieve_all(B, [], W0, W1).

% Si G se puede cumplir realizando una accion especifica, cumplir las
% preconficiones de esa accion, y luego realizar la accion en cuestion
achieve(G, W0, do(Action, W1)):-
  achieves(Action, G),
  preconditions(Action, Pre),
  achieve_all(Pre, [], W0, W1).

% holds(+C, +W)
% Este predicado es verdadero si C se cumple en el mundo W.
%
% Sean P las precondiciones de la accion A, si se cumplen todas las
% precondiciones en el mundo W,  y C es parte de la add_list de A, entonces
    C se
% cumple.
holds(C, do(A, W)):-
  preconditions(A, P),
  holdsall(P, W),
  add_list(A, AL),
  member(C, AL).

% Sean P las precondiciones de la accion A, si se cumplen las
    precondiciones en
% el mundo W, y C no esta en la del_list de la accion A, es verdadero si C
    se
% cumple en W
holds(C, do(A, W)):-
  preconditions(A, P),
  holdsall(P, W),
```

```
  del_list(A, DL),
  notin(C, DL),
  holds(C, W).

%C se cumple si es parte del mundo actual W
holds(C, W):-
  member(C, W).

% holdsall(+Metas, +W)
% Verdadero si cada una de las metas en Metas se cumple en el mundo W.
holdsall([C|L], W):-
  holds(C, W),
  holdsall(L, W).

holdsall([], _).

% notin(+C, +DL)
% Verdadero si C no es miembro de la lista DL
notin(C, DL):-
  not(member(C, DL)).

% remove(-G, +Goals, -Rem_Gs)
% Selecciona la primer meta de Goals en G, y el resto de las metas en
    Rem_Gs.
remove(G, [G|Rem_Gs], Rem_Gs).

% achieves(+Action, +Goal)
% Verdadero si Goal es parte de la add_list de Action
achieves(Action, G):-
  add_list(Action, AL),
  in(G, AL).

% in(+Goal, +AL).
% Verdadero si G es parte de AL.
in(G, [G|_AL]).
in(G, [_|AL]):-
  in(G, AL).

% translate(+World, -Plan)
% Dado un mundo World, traduce el mismo a una lista de acciones en Plan.
translate([_|_], []).
translate(do(Action, W), NPlan):-
  translate(W, Plan),
  append(Plan, [Action], NPlan).
```