Proyecto 2
Inteligencia Artificial

Garay, Iñaki (L.U. 67387) Montenegro, Emiliano (L.U. 83864)

Segundo Cuatrimestre 2010

Índice general

Capítulo 1

Definición del mundo de bloques

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BLOCKS WORLD

%------------------------------------------------------------------------------%
% Accion: apilar(A, B)
% El bloque A se apila sobre el bloque B siempre y cuando ambos bloques esten
% libres y el bloque A se encuentre sobre la mesa.

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(A, B)
% Desapila sobre la mesa el bloque A que se encuentre sobre el B siempre y
% cuando el bloque A este sobre el bloque B y ademas este libre.

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

%------------------------------------------------------------------------------%

achieves( desapilar(A, B), Estado, libre(B))  :- member(sobre(A, B), Estado ).
achieves( desapilar(A, B), Estado, enMesa(A)) :- member(sobre(A, B), Estado).
achieves( apilar(A, B),   _Estado, sobre(A, B)).

%------------------------------------------------------------------------------%
% sobre(A, B) es verdadero si el bloque A esta sobre el bloque B.

%------------------------------------------------------------------------------%
% libre(A) is verdadera si el bloque esta libre.

%------------------------------------------------------------------------------%
% enMesa(A) es verdadero si el bloque A esta sobre la mesa.

%------------------------------------------------------------------------------%
% estadoInicial(L) representa el estado inicial. L es una lista de instancias
% de relaciones que se verifician en el estado inicial.

estadoInicial([ sobre(a, c), libre(a), libre(b), enMesa(c), enMesa(b) ]).
```

Capítulo 2

Tests

Ademas del predicado requerido `plan/2`, se proveen como facilidad los siguientes predicados: `requerimiento_funcional`, `test1`, `test2`, `test3`.

Al ejecutar la consulta:

```
?- requerimiento_funcional.
```

el sistema busca un plan en la situación especificada en el enunciado como requerimiento funcional del proyecto.

Al ejecutar la consulta:

```
?- test1.
```

o idem con `test2`, `test3`, el sistema ejecuta varios tests con situaciones y metas distintas.

Notese, por ejemplo, que el en algunos casos, como por ejemplo el de `test3`, el planificador encuentra un plan en el cual tiene que relograr una meta, deshaciendo una acción anterior.

```
%------------------------------------------------------------------------------%
% TESTS

requerimiento_funcional :-
    W = [ libre(a), libre(b), libre(c), enMesa(a), enMesa(b), enMesa(c) ],
    M = [ sobre(a,b), sobre(b,c) ],
    iplan(M, W, _P).

test1 :-
    W = [ enMesa(a), sobre(b,a), sobre(c,b), libre(c) ],
    M = [ sobre(a,b), sobre(b,c) ],
    iplan(M, W, _P).

test2 :-
    W = [ enMesa(a), sobre(b,a), sobre(c,b), libre(c) ],
    M = [ sobre(a,c), sobre(b,a) ],
    iplan(M, W, _P).

test3 :-
    W = [ enMesa(a), enMesa(b), sobre(c,a), sobre(d,b), libre(c), libre(d) ],
    M = [ sobre(a,b), sobre(b,c), sobre(c,d) ],
    iplan(M, W, _P).
```

Capítulo 3

Planificador

Se eligió la estrategia de realcanzar las metas dado que logra encontrar planes en situaciones donde la protección de metas no lo hace, aunque pueda ser menos eficiente y producir planes no minimales.

Además del predicado `plan/2` requerido por el enunciado, se proveen dos predicados más de acceso al planificador. El predicado `plan/3` permite pasarle al planificador un estado inicial por parámetro, y el predicado `iplan/3` muestra de manera gráfica e interactiva la ejecución del plan encontrado por el planificador.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% STRIPS PLANNER
%
%-------------------------------------------------------------------------------%
% plan/2
% plan(+Goals, -Plan)
%
% El predicado plan/2 implementa el planificador especificado por el enunciado.
% Asume que el estado inicial esta representado por el predicado estadoInicial/1

plan(Goals, Plan) :-
    estadoInicial(Current_World),
    achieve_all(Goals, Current_World, Plan, _Final_State).

%-------------------------------------------------------------------------------%
% plan/3
% plan(+Goals, +Initial_State, -Plan)
%
% El predicado plan/3 implementa el mismo planificador que plan/2, pero en lugar
% de asumir que el estado inicial esta representado por el predicado
% estadoInicial/1, espera que se le pase la representacion en el segundo
% argumento del predicado.
% Su proposito es facilitar el testeo del planificador.

plan(Goals, Initial_State, Plan) :-
    achieve_all(Goals, Initial_State, Plan, _Final_State).

%-------------------------------------------------------------------------------%
% iplan/3
% iplan(+Goals, +Initial_State, -Plan)
%
% El predicado iplan/3 implementa un planificador interactivo.
% Luego de calcular el plan muestra una representacion grafica del estado
% inicial del mundo y para cada accion en el mismo imprime el estado resultante
```

```
% de ejecutar la accion, hasta llegar al estado final.
% En cada paso se le pide un input al usuario.

iplan(Goals, Initial_State, Plan) :-
    achieve_all(Goals, Initial_State, Plan, __Final_State),
    write('Estado inicial: '), write(Initial_State), nl,
    write('Metas:          '), write(Goals), nl,
    write('Plan:           '), write(Plan), nl,
    write('Estado inicial:'), nl,
    imprimir_estado(Initial_State), nl, nl,
    print_plan(Plan, Initial_State), nl.

print_plan([], _).
print_plan([Action | Plan], Current_State) :-
    execute(Action, Current_State, Next_State),
    write('Despues de ejecutar la accion: '), write(Action), nl, nl,
    imprimir_estado(Next_State), nl, nl,
    write('Presione enter para continuar...'), get_char(_), nl,
    print_plan(Plan, Next_State).

%-----------------------------------------------------------------------------%
% achieve_all/4
% achieve_all(+Goals, +Current_World, -Plan, -World_After_Plan)
%
% El predicado achieve_all ...
%
% El predicado consta de tres clausulas:
% La primera cubre el caso en que el conjunto de metas es vacio.
%   En este caso, el plan vacio cumple con las metas en el mundo actual.
% La segunda cubre el caso en que el conjunto de metas ya vale en el mundo real.
% La tercera cubre el caso general.

achieve_all([],    Current_World, [], Current_World).

achieve_all(Goals, Current_World, [], Current_World) :-
    holds_all(Goals, Current_World).

achieve_all(Goals, Current_World, Plan, World_After_Plan) :-

    % Selecciona una meta a alcanzar del conjunto de metas.
    remove(Goals, Goal, Remaining_Goals),

    % Busca un plan que logre esa meta en particular.
    % Goal es la meta actual a alcanzar.
    % Current_World es el estado del mundo actual.
    % Goal_Plan es el plan para lograr la meta Goal.
    % World_After_Goal_Plan es el estado del mundo luego de ejecutar el plan.
    % para alzanzar la meta Goal.
    achieve(Goal, Current_World, Goal_Plan, World_After_Goal_Plan),

    % Busca un plan que logre el resto de las metas en el mundo despues de
    % ejecutar el plan para lograr la meta actual.
    % Remaining_Goals es el resto de las metas a alcanzar.
    % Remaining_Goals_Plan es el plan para lograr el resto de las metas.
    % World_After_Remaining_Goals_Plan es el mundo que resulta de ejecutar el
    % plan para lograr el resto de las metas.
    achieve_all(Remaining_Goals,
                World_After_Goal_Plan,
                Remaining_Goals_Plan,
```

```
                World_After_Remaining_Goals_Plan),
    append(Goal_Plan, Remaining_Goals_Plan, All_Goals_Plan),

    % Verifica si alguna accion deshizo meta necesaria.
    achieve_all(Goals,
                World_After_Remaining_Goals_Plan,
                Redo_Plan,
                World_After_Plan),
    append(All_Goals_Plan, Redo_Plan, Plan).

%----------------------------------------------------------------------------%
% holds/2
% holds(+Goal, +World)
%
% El predicado  holds(Goal, World) es verdadero si la meta Goal vale en el
% mundo World.
%
holds(Goal, World) :-
    member(Goal, World).

%----------------------------------------------------------------------------%
% holds_all/2
% holds_all(+Goals, World)
%
% El predicado holds_all(Goals, World) es verdadero si el conjunto de metas
% Goals vale en el mundo actual World.
% El predicado verifica esta propiedad asegurandose que holds/2 valga para
% cada meta en el conjunto de metas.

holds_all([], _Current_World).
holds_all([Goal | Remaining_Goals], Current_World) :-
    holds(Goal, Current_World),
    holds_all(Remaining_Goals, Current_World).

%----------------------------------------------------------------------------%
% remove/3
% remove(+Goals, +Goal, -Remaining_Goals)
%
% El predicado remove/3 selecciona una meta de un conjunto de metas y liga el
% tercer argumento con el conjunto de metas remanentes.

remove([Goal | Remaining_Goals], Goal, Remaining_Goals).

%----------------------------------------------------------------------------%
% achieve/4
% achieve(+Goal, +Current_World, -Goal_Plan, -World_After_Goal_Plan)
%
% Un plan vacio logra una meta Goal en el mundo actual Current_World si esa
% meta ya vale en el mundo actual.

achieve(Goal, Current_World, [], Current_World) :-
    member(Goal, Current_World).
achieve(Goal, Current_World, Goal_Plan, World_After_Goal_Plan):-
    % Elegir una accion que logre la meta en el mundo actual.
    achieves(Action, Current_World, Goal),
    preconditions(Action, Preconditions),
    achieve_all(Preconditions,
                Current_World,
                Preconditions_Plan,
```

```
                World_After_Preconditions_Plan),
    execute(Action, World_After_Preconditions_Plan, World_After_Goal_Plan),
    append(Preconditions_Plan, [Action], Goal_Plan).

%-----------------------------------------------------------------------------%
% execute/3
% execute(+Action, +Current_World, -Next_World)
%
% El predicado execute/3 calcula el efecto de ejecutar la accion Action en el
% mundo Current_World, y liga el mundo resultante con el argumento Next_World.

execute(Action, Current_World, Next_World) :-
    add_list(Action, Add_List),
    del_list(Action, Del_List),
    delete_from_world(Del_List, Current_World, New_World),
    add_to_world(Add_List, New_World, Next_World).

%-----------------------------------------------------------------------------%
% delete_from_world/3
% delete_from_world(+Relations, +Current_World, -New_World)
%
% delete_from_world/3 calcula el mundo resultante de eliminar un conjunto de
% relaciones de el.
% Su proposito es calcular el mundo resultante de eliminar las relaciones en el
% delete list de una accion que ha sido ejecutada.

delete_from_world([], World, World).
delete_from_world([Relation | Delete_List], World, New_World) :-
    delete(World, Relation, New_World1),
    delete_from_world(Delete_List, New_World1, New_World).

%-----------------------------------------------------------------------------%
% add_to_world/3
% add_to_world(+Add_List, +Current_World, +New_World)
%
% add_to_world/3 calcula el mundo resultante de agregar un conjunto de
% relaciones a el.
% Su proposito es calcular el mundo resultante de agregar las relaciones en el
% add list de una accion que ha sido ejecutada.

add_to_world(Add_List, Current_World, New_World) :-
    append(Add_List, Current_World, New_World).
```

Capítulo 4

Impresión de la representación grafica del mundo

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MODULO DE IMPRESION DEL MUNDO DE BLOQUES

imprimir_estado(Relations_List):-
    obtener_pilas(Relations_List, List_Of_Stacks),
    quicksort(List_Of_Stacks, Imprimible_List_Of_Stacks),
    imprimir(Imprimible_List_Of_Stacks).

%-------------------------------------------------------------------------------%
% Este predicado arma a partir de las relaciones las pilas del mundo de bloque
% obtener_pilas(+Lista de relaciones, -Lista de pilas del mundo)
obtener_pilas([], []).
obtener_pilas(Relations_List, List_Of_Stacks):-
    armar_bases_pilas(Relations_List,
                      Remaining_Relations_List,
                      [],
                      Out_Bases_Of_Stacks),
    colocar_elementos_en_pila(Remaining_Relations_List,
                              Out_Bases_Of_Stacks,
                              List_Of_Stacks).

%-------------------------------------------------------------------------------%
% Este predicado arma las bases de todas las pilas que existen en el mundo
% siguiendo el predicado enMesa(X) armar_bases_pilas(+Relaciones, -Resto de
% relaciones, +Bases de las pilas, -Bases de las pilas salida)

armar_bases_pilas([], [], Bases_Of_Stacks, Bases_Of_Stacks).

armar_bases_pilas([enMesa(A) | Relations],
                  Remaining_Relations_List,
                  Bases_Of_Stacks,
                  Out_Bases_Of_Stacks) :-
    armar_bases_pilas(Relations,
                      Remaining_Relations_List,
                      [[A] | Bases_Of_Stacks],
                      Out_Bases_Of_Stacks).

armar_bases_pilas([Relation_X | Relations],
                  [Relation_X | Remaining_Relations_List],
                  Bases_Of_Stacks,
                  Out_Bases_Of_Stacks) :-
    armar_bases_pilas(Relations,
```

```
                        Remaining_Relations_List,
                        Bases_Of_Stacks,
                        Out_Bases_Of_Stacks).

%-----------------------------------------------------------------------------%
% Arma a partir de la relacion sobre(A, B), que elementos estan sobre cuales en
% las pilas. Esto lo hace recorriendo varias veces la lista hasta no encontrar
% relaciones sobre. colocar_elementos_en_pila(+ Relaciones restantes luego de
% armar las bases, +Bases de las pilas, -Lista de pilas completas)

colocar_elementos_en_pila([], Bases_Of_Stacks, Bases_Of_Stacks).

colocar_elementos_en_pila([Relation | Relations],
                          Bases_Of_Stacks,
                          Preliminar_List_Of_Stacks) :-
    colocar_en_pila(Relation,
                    Bases_Of_Stacks,
                    Preliminar_List_Of_Stacks_Temp),
    colocar_elementos_en_pila(Relations,
                              Preliminar_List_Of_Stacks_Temp,
                              Preliminar_List_Of_Stacks).

colocar_elementos_en_pila([Relation | Relations],
                          Bases_Of_Stacks,
                          Preliminar_List_Of_Stacks) :-
    colocar_elementos_en_pila(Relations,
                              Bases_Of_Stacks,
                              Preliminar_List_Of_Stacks_Temp),
    colocar_en_pila(Relation,
                    Preliminar_List_Of_Stacks_Temp,
                    Preliminar_List_Of_Stacks).

%-----------------------------------------------------------------------------%
% Si la relacion es sobre(A,B), debe colocarse en una pila. En caso de no
% poderse falla para poder rehacerlo luego. colocar_en_pila(+relacion, +Lista
% de Pilas, -Salida de lista de pilas)
colocar_en_pila( sobre(A,B), Bases_Of_Stacks, Preliminar_List_Of_Stacks) :-
    buscar_y_poner_en_pila(A , B, Bases_Of_Stacks, Preliminar_List_Of_Stacks).

colocar_en_pila( Relation, Bases_Of_Stacks, Bases_Of_Stacks):-
    Relation \= sobre(_A,_B).

%-----------------------------------------------------------------------------%
% Busca la pila que tenga como tope al elemento q necesito, entonces lo coloca
% encima de ella.
%
% buscar_y_poner_en_pila(+Nuevo Bloque, +Bloque Destino, +Lista de Pilas,
% -Salida De Lista de Pilas

buscar_y_poner_en_pila(A, B, [Stack | Stacks], [ [A | Stack] | Stacks]) :-
    member(B, Stack).

buscar_y_poner_en_pila(A, B, [Stack | Stacks],[Stack | Preliminar_List_Of_Stacks]) :-
    buscar_y_poner_en_pila(A, B, Stacks, Preliminar_List_Of_Stacks).

%-----------------------------------------------------------------------------%
% quicksort/2

quicksort([], []).
```

```
quicksort([X|Xs], Ys) :-
    partition(Xs, X, Littles, Bigs),
    quicksort(Littles, Ls),
    quicksort(Bigs,    Bs),
    append(Ls, [X|Bs], Ys).

%--------------------------------------------------------------------------------%
% partition/4

partition([], _Y, [], []).
partition([X|Xs], Y, [X|Ls], Bs) :-
    length(X, FX),
    length(Y, FY),
    FX >= FY,
    partition(Xs, Y, Ls, Bs).
partition([X|Xs], Y, Ls, [X|Bs]) :-
    length(X, FX),
    length(Y, FY),
    FX < FY,
    partition(Xs, Y, Ls, Bs).

%--------------------------------------------------------------------------------%
% Recorre las pilas de bloques e imprime los caracteres necesarios para su
% representacion gr\'{a}fica
% imprimir(+ Lista de pilas)
imprimir(List_Of_Stacks):-
    List_Of_Stacks = [[] | _],
    length(List_Of_Stacks, Cubes_On_Table),
    % 5 de base para los cubos, la separacion entre las pilas y 1 mas por el
    % final.
    Printeable_Table is Cubes_On_Table * 5 + Cubes_On_Table + 1,
    print_line_of_char(1, Printeable_Table, '#').

imprimir(List_Of_Stacks) :-
    List_Of_Stacks = [H | _T],
    length(H, Max_Element),
    obtener_elementos_a_imprimir(List_Of_Stacks,
                                 Max_Element,
                                 Printeable_Line,
                                 Remaining_List_Of_Stacks),
    imprimir_la_linea_cubos(Printeable_Line),
    imprimir(Remaining_List_Of_Stacks).

%--------------------------------------------------------------------------------%
print_line_of_char(Max, Max, C) :-
    write(C).
print_line_of_char(I, Max, C) :-
    I<Max,
    write(C),
    Inc is I + 1,
    print_line_of_char(Inc, Max, C).

%--------------------------------------------------------------------------------%
obtener_elementos_a_imprimir([[] | _ ], _Max_Element, [], [[] | _]).
obtener_elementos_a_imprimir([Stack | Stacks], Max_Element,
                             [Elem | Printeable_Line],
                             [Remaining | Remaining_List_Of_Stacks]) :-
    length(Stack, Max_Element),
    Stack = [Elem | Remaining],
```

```
    obtener_elementos_a_imprimir(Stacks,
                                 Max_Element,
                                 Printeable_Line,
                                 Remaining_List_Of_Stacks).
obtener_elementos_a_imprimir( Stacks, _, [], Stacks).

%-------------------------------------------------------------------------------%
imprimir_la_linea_cubos(Printeable_Line) :-
    length(Printeable_Line, Size_Line),
    imprimir_techo(0, Size_Line), nl,
    imprimir_pared_uno(0, Size_Line), nl,
    imprimir_pared_medio(0, Size_Line, Printeable_Line), nl,
    imprimir_pared_uno(0, Size_Line), nl,
    imprimir_techo(0, Size_Line), nl.

%-------------------------------------------------------------------------------%
imprimir_techo(Max, Max).
imprimir_techo(I, Max) :-
    I < Max,
    print_line_of_char(1, 1, ' '),
    print_line_of_char(1, 5, '*'),
    Inc is I+1,
    imprimir_techo(Inc, Max).

imprimir_pared_uno(Max, Max).
imprimir_pared_uno(I, Max) :-
    I < Max,
    print_line_of_char(1, 1, ' '),
    print_line_of_char(1, 1, '*'),
    print_line_of_char(1, 3, ' '),
    print_line_of_char(1, 1, '*'),
    Inc is I+1,
    imprimir_pared_uno(Inc, Max).

imprimir_pared_medio(Max, Max, _Printeable_Line).
imprimir_pared_medio(I, Max, [H | Printeable_Line]) :-
    I < Max,
    print_line_of_char(1, 1, ' '),
    print_line_of_char(1, 1, '*'),
    print_line_of_char(1, 1, ' '),
    print_line_of_char(1, 1, H),
    print_line_of_char(1, 1, ' '),
    print_line_of_char(1, 1, '*'),
    Inc is I+1,
    imprimir_pared_medio(Inc, Max, Printeable_Line).
```

Capítulo 5

Ejemplo de corrida

A continuacion se muestra el resultado de la ejecución del segundo test.

```
?- test2.
Estado inicial: [enMesa(a),sobre(b,a),sobre(c,b),libre(c)]
Metas:          [sobre(a,c),sobre(b,a)]
Plan:           [desapilar(c,b),desapilar(b,a),apilar(a,c),apilar(b,a)]
Estado inicial:
 *****
 *   *
 * c *
 *   *
 *****
 *****
 *   *
 * b *
 *   *
 *****
 *****
 *   *
 * a *
 *   *
 *****
#######

Despues de ejecutar la accion: desapilar(c,b)

 *****
 *   *
 * b *
 *   *
 *****
 ***** *****
 *   * *   *
 * a * * c *
 *   * *   *
 ***** *****
#############
```

```
Presione enter para continuar...

Despues de ejecutar la accion: desapilar(b,a)

 ***** ***** *****
 *   * *   * *   *
 * b * * c * * a *
 *   * *   * *   *
 ***** ***** *****
###################

Presione enter para continuar...

Despues de ejecutar la accion: apilar(a,c)

 *****
 *   *
 * a *
 *   *
 *****
 ***** *****
 *   * *   *
 * c * * b *
 *   * *   *
 ***** *****
#############

Presione enter para continuar...

Despues de ejecutar la accion: apilar(b,a)

 *****
 *   *
 * b *
 *   *
 *****
 *****
 *   *
 * a *
 *   *
 *****
 *****
 *   *
 * c *
 *   *
 *****
#######
```

```
Presione enter para continuar...


true .
```