Inteligência Artificial - 1º Trabalho Prático: Planejador para Empilhar blocos de diferente dimensões

---

Equipe

1. Alberth Viana de Lima
2. Ana Júlia Pereira Corrêa
3. Guilherme Sahdo Maciel

QUESTÕES - MUNDO DOS BLOCOS

1) Descrever o problema em linguagem natural, falando dos blocos de comprimento diferentes, iguais apenas na altura

O problema consiste no empilhamento de blocos retangulares que possuem diferentes comprimentos, mas todos têm a mesma altura unitária (altura = 1). Os blocos podem ter comprimento 1, 2 ou 3 unidades, e são posicionados em uma grade bidimensional de tamanho 6x4 (6 posições no eixo X e 4 no eixo Y). Cada bloco ocupa uma sequência contínua de posições horizontais na mesma linha da grade.

O objetivo é encontrar uma sequência de movimentos que transforme uma configuração inicial dos blocos em uma configuração final desejada, respeitando regras de estabilidade física e ocupação do espaço. Essas regras garantem que os blocos não se sobreponham indevidamente e que blocos maiores sejam posicionados de forma estável, considerando o suporte abaixo deles.

O estado do mundo é representado por uma lista de fatos que indicam quais posições estão ocupadas, quais estão livres (clear), e a posição inicial de cada bloco (on(Block, Position)).

2) Definir os conceitos de livre (clear) e outros necessários (vide descrição em anexo), estado do mundo deve ser uma lista com as relações de sobreposição, isto é on(X,Y)

- **clear((X,Y)):** Indica que a posição (X,Y) está livre, ou seja, não há nenhum bloco ocupando essa posição nem nenhum bloco acima dela que impeça a movimentação.
- **occupied((X,Y)):** Indica que a posição (X,Y) está ocupada por algum bloco.

- **on(Block, (X,Y)):** Indica que o bloco está posicionado começando na coordenada **(**X,Y). Para blocos com comprimento maior que 1, ele ocupa posições consecutivas no eixo X.
- **stable((X,Y), Size, Conditions):** Verifica se as condições para que um bloco de tamanho Size posicionado em (X,Y) esteja estável são satisfeitas, ou seja, se há suporte adequado abaixo do bloco.
- **clear_above((X,Y), Size, List, ClearList):** Garante que as posições acima do bloco estejam livres para permitir o movimento.

O estado do mundo é uma lista que contém fatos do tipo occupied, clear e on, representando a configuração atual dos blocos na grade.

3) Pedir ao chat GPT uma representação adequada de termos em lógica para representar os elementos descritos no documento anexo, sem o uso de assign e retract.

A representação lógica usada no código é adequada e segue o padrão:

- **block(Block, Size):** define o bloco e seu tamanho.
- **place((X,Y)):** define as posições válidas na grade.
- **stateX([...]):** define o estado do mundo como uma lista de fatos.
- **on(Block, Position):** posição inicial do bloco.
- **occupied(Position):** posição ocupada.
- **clear(Position):** posição livre.
- **action(moveN(Block, From, To)):** ação de mover um bloco de tamanho N de uma posição para outra.
- **can(Action, Conditions):** condições para que a ação seja possível.
- **adds(Action, AddList):** fatos adicionados após a ação.
- **deletes(Action, DeleteList):** fatos removidos após a ação.

Essa representação evita o uso de assert e retract, mantendo o estado do mundo como uma lista imutável que é transformada a cada ação. A representação lógica adotada para o problema utiliza predicados que descrevem os blocos, posições e estado do mundo, mantendo a imutabilidade do estado (sem assert ou retract):

```
% Definição dos blocos e seus tamanhos

block(a, 2).

block(b, 1).

block(c, 3).
```

```
block(d, 2).


% Definição das posições válidas na grade
place((X,Y)) :- X >= 1, X =< 6, Y >= 1, Y =< 4.


% Estado do mundo representado como lista de fatos
state1([
    on(a, (1,1)),
    on(b, (3,1)),
    on(c, (1,2)),
    on(d, (4,2)),
    clear((5,1)),
    clear((6,1)),
    clear((2,2)),
    clear((3,2))
]).


% Relações básicas
on(Block, Position).      % Bloco Block está na posição Position
occupied(Position).       % Posição ocupada por algum bloco
clear(Position).          % Posição livre


% Ação de mover um bloco de uma posição para outra
action(moveN(Block, From, To)).
```

```
% Condições para que a ação seja possível

can(Action, Conditions).



% Fatos adicionados após a ação

adds(Action, AddList).



% Fatos removidos após a ação

deletes(Action, DeleteList).
```

4) Definir as restrições em linguagem natural e pedir para o chat utilizar as descrições por ele geradas para escrever as restrições em código Prolog.

Restrições:

- Um bloco só pode ser movido para uma posição válida dentro da grade.
- O bloco não pode se mover para uma posição já ocupada.
- Para blocos maiores que 1, todas as posições que ele ocupar devem estar livres.
- O bloco deve estar estável na nova posição, ou seja, deve haver suporte adequado abaixo.
- Não pode haver blocos "acima" do bloco que impeçam o movimento.
- O movimento deve respeitar a integridade do estado, atualizando as posições ocupadas e livres corretamente.

Restrições em Prolog (exemplos do código):

```
can(move1(Block, From, To), [on(Block, From) | Conditions]) :-

    block(Block, 1),

    place(To),

    stable(To, 1, OccpList),

    From \== To,

    \+ above_itself(From, 1, OccpList),

    clear_above(From, 1, [], ClearList),
```

```
    append([clear(To) | ClearList], OccpList, Conditions).
```

Este predicado verifica se o movimento do bloco de tamanho 1 é possível, garantindo que a posição de destino esteja estável, que não haja blocos acima impedindo o movimento, e que as posições necessárias estejam livres.

4.1 Fornecer exatamente o código do livro (disponível neste material do Classroom), principalmente as constraints (restrições)

O código base do planejador por regressão de metas (goal regression) do livro do Bratko (capítulo 17) é o seguinte:

me_goal_reg_planner.pl

```
%-----------------------------------------------------------
%   Figure  17.8 (3rd) or 17.6 (4th) Edition
%   A planner based on goal regression.
%   This planner searches in iterative-deepening style.



%   A means-ends planner with goal regression
%   plan( State, Goals, Plan)
plan( State, Goals, []):-
  satisfied( State, Goals).                  % Goals true in State

plan( State, Goals, Plan):-
   append( PrePlan, [Action], Plan),           % Divide plan achieving
breadth-first effect
  select( State, Goals, Goal),                % Select a goal
  achieves( Action, Goal),
  can( Action, Condition),                     % Ensure Action contains
no variables
  preserves(Action,Goals),                    % Protect Goals
   regress( Goals, Action, RegressedGoals),     % Regress Goals through
Action
  plan( State, RegressedGoals, PrePlan).

satisfied( State, Goals)  :-
  delete_all( Goals, State, []).              % All Goals in State
% -----------------------------------------
select( State, Goals, Goal)  :-               % Select Goal from Goals
```

```
   member( Goal, Goals).                          % A simple selection
principle

% ----------------------------------------
achieves( Action, Goal)  :-
  adds( Action, Goals),
  member( Goal, Goals).

% ----------------------------------------
preserves( Action, Goals)  :-                 % Action does not destroy
Goals
  deletes( Action, Relations),
  \+  (member( Goal, Relations),              % not member
       member( Goal, Goals) ).

% ----------------------------------------
regress( Goals, Action, RegressedGoals)   :-          % Regress Goals
through Action
  adds( Action, NewRelations),
  delete_all( Goals, NewRelations, RestGoals),
  can( Action, Condition),
  addnew( Condition, RestGoals, RegressedGoals).  % Add precond., check
imposs.

% ----------------------------------------
% addnew( NewGoals, OldGoals, AllGoals):
%   OldGoals is the union of NewGoals and OldGoals
%   NewGoals and OldGoals must be compatible
addnew( [], L, L).

addnew( [Goal | _], Goals, _)  :-
  impossible( Goal, Goals),         % Goal incompatible with Goals
  !,
  fail.                             % Cannot be added

addnew( [X | L1], L2, L3)  :-
  member( X, L2),  !,               % Ignore duplicate
  addnew( L1, L2, L3).

addnew( [X | L1], L2, [X | L3])  :-
  addnew( L1, L2, L3).

% ----------------------------------------
```

```
% delete_all( L1, L2, Diff): Diff is set-difference of lists L1 and L2
delete_all( [], _, []).

delete_all( [X | L1], L2, Diff)  :-
  member( X, L2), !,
  delete_all( L1, L2, Diff).

delete_all( [X | L1], L2, [X | Diff])  :-
  delete_all( L1, L2, Diff).

% ------------------------------------------
member(X,[X|_]).
member(X,[_|T]):-
     member(X,T).

delete(X,[X|Tail],Tail).
delete(X,[Y|Tail],[Y|Tail1]):-
   delete(X, Tail,Tail1).
e blocks_world_bratko_17_9_constraints.pl:
final([clear(1),clear(2),clear(3),on(d,p([4,6])),on(c,p([d,d])),on(a,c)
,on(b,c)])
%                                  on(Block,p())
% state1([clear(3),on(c,p([1,2])),on(b,6),on(a,4), on(d,p([a,b]))]).

%-----------------------------------------------------------
% Definition of action move(Block, From, To)
% can(Action, Condition): Action possible if Condition true

can(move(Block,p([Oi,Oj]),p([Bi,Bj])),[clear(Block),clear(Bi),clear(Bj)
,on(Block,p([Oi,Oj]))]):-
    block(Block),           % There is this Block to move
    block(Bi),              %
    block(Bj),
    Bi \== Block,           % Block cannot be moved to itself
    block(Oi), block(Oj),   % 'From' is a block or place
    Oi \== Bi,              % 'To' is a new position to move
    Oj \== Bj,              % 'To' is a new position to move
    Block \== Oi,
    size(Block,Sb),
    size(Bi,Si),
    size(Bj,Sj),
    SizeTo is Si + Sj,
    Sb =< SizeTo + 1.       % Bloco on the top cannot exceed 1 of To
```

```
% This case assume a Block 2 units bigger than the single block B
% will always be placed in the middle to maintain its centroid
% aligned with Bs centroid

can(move(Block,p([Oi,Oj]),p(B,B)),[clear(Block),clear(B),on(Block,p([Oi
,Oj]))]):-
    block(Block),           % There is this Block to move
    block(B),
    B \== Block,            % Block cannot be moved to itself
    block(Oi), block(Oj),   % 'From' is a block or place
    Oi \== B,               % 'To' is a new position to move
    Oj \== B,
    size(Block,SizeB),
    size(B,Sb),
    SizeB =< Sb + 2.

%can(move(Block,p(To,To),p(Bi,Bj)),[clear(Block),clear(Bi),clear(Bj),on
(Block,p(Oi,Oj))]):-

can(move(Block,From,To),[clear(Block),clear(To),on(Block,From)]):-
    block(Block),           % There is this Block to move
    object(To),             % 'T' is a block or place
    To \== Block,           % Block cannot be moved to itself
    object(From),           % 'From' is a block or place
    From \== To,            % 'To' is a new position to move
    Block \== From.         % Bloco not moved from itself

%-------------------------------------------------------------
% adds(Action, Relationships): Action establishes new Relationships
adds(move(X,From,To),[on(X,To),clear(From)]).

%-------------------------------------------------------------
% deletes(Action, Relationships): Action destroy Relationships
deletes(move(X,From,To),[on(X,From),clear(To)]).

object(X):-
    place(X)
    ;
    block(X).
%-------------------------------------------------------------
impossible(on(X,X),_).
impossible(on(X,Y),Goals):-
```

```
    member(clear(Y),Goals)
    ;
    member(on(X,Y1),Goals), Y1 \== Y   % Block cannot be in two places
    ;
     member(on(X1,Y), Goals), X1 \== X. % Two blocks cannot be at the
same place
impossible(clear(X),Goals):-
    member(on(_,X),Goals).
```

Baseado no livro, os códigos feitos para solucionar o Mundo Dos Blocos estipulado no PDF usaram:

```
% Planejador baseado em regressão de metas

plan(State, Goals, []) :-

    satisfied(State, Goals).


plan(State, Goals, Plan) :-

    append(PrePlan, [Action], Plan),

    select(State, Goals, Goal),

    achieves(Action, Goal),

    can(Action, Condition),

    preserves(Action, Goals),

    regress(Goals, Action, RegressedGoals),

    plan(State, RegressedGoals, PrePlan).



% Verifica se todos os objetivos estão satisfeitos no estado atual

satisfied(State, Goals) :-

    delete_all(Goals, State, []).
```

```
% Seleciona um objetivo da lista de objetivos
select(_, Goals, Goal) :-
    member(Goal, Goals).

% Verifica se uma ação alcança um objetivo
achieves(Action, Goal) :-
    adds(Action, Goals),
    member(Goal, Goals).

% Verifica se uma ação preserva os objetivos
preserves(Action, Goals) :-
    deletes(Action, Relations),
    \+ (member(Goal, Relations), member(Goal, Goals)).

% Regressão dos objetivos através da ação
regress(Goals, Action, RegressedGoals) :-
    adds(Action, NewRelations),
    delete_all(Goals, NewRelations, RestGoals),
    can(Action, Condition),
    addnew(Condition, RestGoals, RegressedGoals).

% Outros predicados auxiliares (addnew, delete_all, member, etc.) seguem conforme o código do livro
```

4.2 Pedir para o Chatbot implementar o descrito, considerando goal regression e means-ends do livro.

No contexto do livro do Bratko, o planejador utiliza regressão de metas para gerar planos recursivamente, partindo do estado final desejado e regressando até o estado inicial, garantindo que apenas ações válidas sejam consideradas.

O predicado principal é:

```
plan(State, Goal, Plan).
```

que gera uma sequência de ações Plan que transforma o State inicial no Goal desejado.

Para se adaptar ao domínio do Mundo dos Blocos, deve-se usar as regras lógicas definidas (como on/2, clear/1, block/2, can/2, etc.) e as restrições para garantir que o planejador só considere movimentos válidos.

5) Testar com os exemplos do documento em anexo e corrigir os erros do código usando trace.

Código desenvolvido com base no livro com as técnicas BFS e A*:

1. **BFS**

```
trace, testar_best_first(state1, goali2).
```

```
 Call:testar_best_first(state1,goali2)

 Call:state1(_3858)

     Exit:state1([occupied((1,1)),     clear((1,2)),     clear((1,3)),
clear((1,4)),     occupied((2,1)),     clear((2,2)),     clear((2,3)),
clear((2,4)),  clear((3,1)),  clear((3,2)),  clear((3,3)),  clear((3,4)),
occupied((4,1)),     occupied((4,2)),     clear((4,3)),     clear((4,4)),
clear((5,1)),     occupied((5,2)),     clear((5,3)),     clear((5,4)),
occupied((6,1)),     occupied((6,2)),     clear((6,3)),     clear((6,4)),
occupied((1,0)),  occupied((2,0)),  occupied((3,0)),  occupied((4,0)),
occupied((5,0)),     occupied((6,0)),     on(a,(4,1)),     on(b,(6,1)),
on(c,(1,1)), on(d,(4,2))])

Plano encontrado de state1 para goali2 em 154 ms:

[move3(d,(4,2),(1,2)),move1(a,(4,1),(6,2)),move3(d,(1,2),(3,1)),move1(a
,(6,2),(1,2))]
```

```
 Exit:testar_best_first(state1,goali2)

 Call:testar_best_first(state1,goali2)
```

trace, testar_best_first(state2, goali2).

```
Call:testar_best_first(state2,goali2)

 Call:state2(_3858)

    Exit:state2([occupied((1,1)),    occupied((1,2)),    clear((1,3)),
clear((1,4)),     occupied((2,1)),     clear((2,2)),     clear((2,3)),
clear((2,4)),     occupied((3,1)),     clear((3,2)),     clear((3,3)),
clear((3,4)),     occupied((4,1)),     clear((4,2)),     clear((4,3)),
clear((4,4)),     occupied((5,1)),     clear((5,2)),     clear((5,3)),
clear((5,4)),     occupied((6,1)),     clear((6,2)),     clear((6,3)),
clear((6,4)),   occupied((1,0)),   occupied((2,0)),   occupied((3,0)),
occupied((4,0)),    occupied((5,0)),    occupied((6,0)),    on(a,(1,2)),
on(b,(6,1)), on(c,(1,1)), on(d,(3,1))])

Plano encontrado de state2 para goali2 em 0 ms:

[]

 Exit:testar_best_first(state2,goali2)

 Call:testar_best_first(state2,goali2)
```

trace, testar_best_first(status2, goali2).

```
Call:testar_best_first(status2,goali2)

 Call:status2(_3858)

    Exit:status2([occupied((1,1)),    occupied((1,2)),    clear((1,3)),
clear((1,4)),    occupied((2,1)),    occupied((2,2)),    clear((2,3)),
clear((2,4)),  clear((3,1)),  clear((3,2)),  clear((3,3)),  clear((3,4)),
occupied((4,1)),     clear((4,2)),     clear((4,3)),     clear((4,4)),
occupied((5,1)),     clear((5,2)),     clear((5,3)),     clear((5,4)),
occupied((6,1)),     clear((6,2)),     clear((6,3)),     clear((6,4)),
occupied((1,0)),  occupied((2,0)),  occupied((3,0)),  occupied((4,0)),
occupied((5,0)),     occupied((6,0)),     on(a,(1,2)),     on(b,(2,2)),
on(c,(1,1)), on(d,(4,1))])
```

```
Plano encontrado de status2 para goali2 em 199 ms:

[move1(b,(2,2),(6,2)),move1(b,(6,2),(5,2)),move1(b,(5,2),(3,1)),move1(b
,(3,1),(1,3)),move3(d,(4,1),(2,2)),move3(d,(2,2),(3,1)),move1(b,(1,3),(
6,1))]

 Exit:testar_best_first(status2,goali2)

 Call:testar_best_first(status2,goali2)
```

trace, testar_best_first(status3e4, goali2).

```
Call:testar_best_first(status3e4,goali2)

 Call:status3e4(_3858)

    Exit:status3e4([occupied((1,1)),     clear((1,2)),     clear((1,3)),
clear((1,4)),      occupied((2,1)),      clear((2,2)),      clear((2,3)),
clear((2,4)),  clear((3,1)),  clear((3,2)),  clear((3,3)),  clear((3,4)),
occupied((4,1)),     occupied((4,2)),     clear((4,3)),     clear((4,4)),
clear((5,1)),      occupied((5,2)),      clear((5,3)),      clear((5,4)),
occupied((6,1)),     occupied((6,2)),     clear((6,3)),     clear((6,4)),
occupied((1,0)),   occupied((2,0)),   occupied((3,0)),   occupied((4,0)),
occupied((5,0)),      occupied((6,0)),      on(a,(4,1)),      on(b,(6,1)),
on(c,(1,1)), on(d,(4,2))])

Plano encontrado de status3e4 para goali2 em 116 ms:

[move3(d,(4,2),(1,2)),move1(a,(4,1),(6,2)),move3(d,(1,2),(3,1)),move1(a
,(6,2),(1,2))]

 Exit:testar_best_first(status3e4,goali2)

 Call:testar_best_first(status3e4,goali2)
```

Testando com trace ainda os códigos do BFS, aparecem os seguintes avisos de limite excedido.

trace, *testar_best_first*(state1, goalia).

Call:*testar_best_first(*state1,goalia)

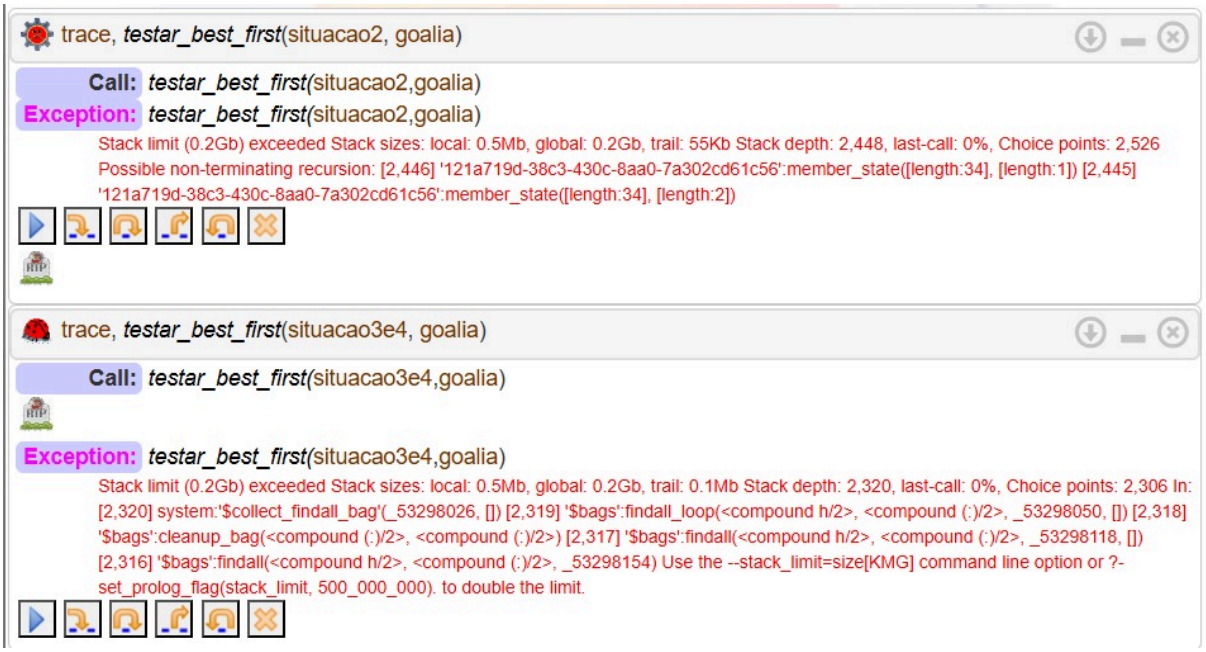
Call:*state1(*_3858)

Exit: *state1*([*occupied*((1,1)), *clear*((1,2)), *clear*((1,3)), *clear*((1,4)), *occupied*((2,1)), *clear*((2,2)), *clear*((2,3)), *clear*((2,4)), *clear*((3,1)), *clear*((3,2)), *clear*((3,3)), *clear*((3,4)), *occupied*((4,1)), *occupied*((4,2)), *clear*((4,3)), *clear*((4,4)), *clear*((5,1)), *occupied*((5,2)), *clear*((5,3)), *clear*((5,4)), *occupied*((6,1)), *occupied*((6,2)), *clear*((6,3)), *clear*((6,4)), *occupied*((1,0)), *occupied*((2,0)), *occupied*((3,0)), *occupied*((4,0)), *occupied*((5,0)), *occupied*((6,0)), *on*(a,(4,1)), *on*(b,(6,1)), *on*(c,(1,1)), *on*(d,(4,2))])

Exception: *testar_best_first*(state1,goalia)

> Stack limit (0.2Gb) exceeded Stack sizes: local: 0.5Mb, global: 0.2Gb, trail: 0.1Mb Stack depth: 2,320, last-call: 0%, Choice points: 2,306 In: [2,320] system:'$collect_findall_bag'(_53298308, []) [2,319] '$bags':findall_loop(<compound h/2>, <compound (:)/2>, _53298332, []) [2,318] '$bags':cleanup_bag(<compound (:)/2>, <compound (:)/2>) [2,317] '$bags':findall(<compound h/2>, <compound (:)/2>, _53298400, []) [2,316] '$bags':findall(<compound h/2>, <compound (:)/2>, _53298436) Use the --stack_limit=size[KMG] command line option or ?- set_prolog_flag(stack_limit, 500_000_000). to double the limit.

Para outros estados, o mesmo problema precede.



2. **A***

```
trace, testar_astar(state1, goali2).
```

```
Call:testar_astar(state1,goali2)

Call:state1(_3832)
```

```
     Exit:state1([occupied((1,1)),     clear((1,2)),     clear((1,3)),
clear((1,4)),     occupied((2,1)),     clear((2,2)),     clear((2,3)),
clear((2,4)),  clear((3,1)),  clear((3,2)),  clear((3,3)),  clear((3,4)),
occupied((4,1)),     occupied((4,2)),     clear((4,3)),     clear((4,4)),
clear((5,1)),     occupied((5,2)),     clear((5,3)),     clear((5,4)),
occupied((6,1)),     occupied((6,2)),     clear((6,3)),     clear((6,4)),
occupied((1,0)),  occupied((2,0)),  occupied((3,0)),  occupied((4,0)),
occupied((5,0)),     occupied((6,0)),     on(a,(4,1)),     on(b,(6,1)),
on(c,(1,1)), on(d,(4,2))])

Plano encontrado de state1 para goali2 em 115 ms:

[move3(d,(4,2),(1,2)),move1(a,(4,1),(6,2)),move3(d,(1,2),(3,1)),move1(a
,(6,2),(1,2))]

 Exit:testar_astar(state1,goali2)

 Call:testar_astar(state1,goali2)
```

trace, testar_astar(state2, goali2).

```
 Call:testar_astar(state2,goali2)

 Call:state2(_3832)

    Exit:state2([occupied((1,1)),    occupied((1,2)),    clear((1,3)),
clear((1,4)),     occupied((2,1)),     clear((2,2)),     clear((2,3)),
clear((2,4)),     occupied((3,1)),     clear((3,2)),     clear((3,3)),
clear((3,4)),     occupied((4,1)),     clear((4,2)),     clear((4,3)),
clear((4,4)),     occupied((5,1)),     clear((5,2)),     clear((5,3)),
clear((5,4)),     occupied((6,1)),     clear((6,2)),     clear((6,3)),
clear((6,4)),   occupied((1,0)),   occupied((2,0)),   occupied((3,0)),
occupied((4,0)),    occupied((5,0)),    occupied((6,0)),    on(a,(1,2)),
on(b,(6,1)), on(c,(1,1)), on(d,(3,1))])

Plano encontrado de state2 para goali2 em 0 ms:

[]

 Exit:testar_astar(state2,goali2)

 Call:testar_astar(state2,goali2)
```

trace, testar_astar(status2, goali2).

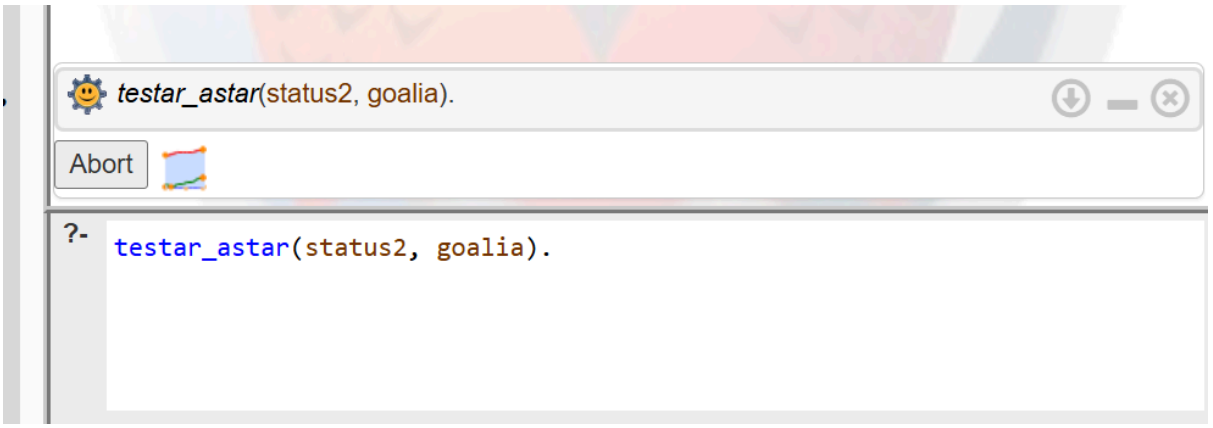
```
 Call:testar_astar(status2,goali2)

 Call:status2(_3832)

    Exit:status2([occupied((1,1)),    occupied((1,2)),    clear((1,3)),
clear((1,4)),    occupied((2,1)),    occupied((2,2)),    clear((2,3)),
clear((2,4)),  clear((3,1)),  clear((3,2)),  clear((3,3)),  clear((3,4)),
occupied((4,1)),     clear((4,2)),     clear((4,3)),     clear((4,4)),
occupied((5,1)),     clear((5,2)),     clear((5,3)),     clear((5,4)),
occupied((6,1)),     clear((6,2)),     clear((6,3)),     clear((6,4)),
occupied((1,0)),   occupied((2,0)),   occupied((3,0)),   occupied((4,0)),
occupied((5,0)),     occupied((6,0)),     on(a,(1,2)),     on(b,(2,2)),
on(c,(1,1)), on(d,(4,1))])

Plano encontrado de status2 para goali2 em 202 ms:

[move1(b,(2,2),(1,3)),move3(d,(4,1),(2,2)),move3(d,(2,2),(3,1)),move1(b
,(1,3),(6,1))]

 Exit:testar_astar(status2,goali2)

 Call:testar_astar(status2,goali2)
```

trace, testar_astar(status3e4, goali2).

```
 Call:testar_astar(status3e4,goali2)

 Call:status3e4(_3832)

    Exit:status3e4([occupied((1,1)),    clear((1,2)),    clear((1,3)),
clear((1,4)),     occupied((2,1)),     clear((2,2)),     clear((2,3)),
clear((2,4)),  clear((3,1)),  clear((3,2)),  clear((3,3)),  clear((3,4)),
occupied((4,1)),    occupied((4,2)),    clear((4,3)),    clear((4,4)),
clear((5,1)),     occupied((5,2)),     clear((5,3)),     clear((5,4)),
occupied((6,1)),    occupied((6,2)),    clear((6,3)),    clear((6,4)),
occupied((1,0)),   occupied((2,0)),   occupied((3,0)),   occupied((4,0)),
occupied((5,0)),     occupied((6,0)),     on(a,(4,1)),     on(b,(6,1)),
on(c,(1,1)), on(d,(4,2))])

Plano encontrado de status3e4 para goali2 em 120 ms:

[move3(d,(4,2),(1,2)),move1(a,(4,1),(6,2)),move3(d,(1,2),(3,1)),move1(a
,(6,2),(1,2))]
```

```
Exit:testar_astar(status3e4,goali2)

Call:testar_astar(status3e4,goali2)
```

Um caso em que há erro é com goalia, onde há um excesso do limite de memória, pelo objetivo "a" ser complexo demais para os códigos desenvolvidos. O que ocorre é quando compilado, ele demora em excesso para dar um resultado e no final, não resulta em nada porque o tempo dele expira.



trace, *testar_astar*(status2, goalia).

Call:*testar_astar(*status2,goalia)

Call:*status2(*_3832)

Exit:*status2(*[*occupied(*(1,1)), *occupied(*(1,2)), *clear(*(1,3)), *clear(*(1,4)), *occupied(*(2,1)), *occupied(*(2,2)), *clear(*(2,3)), *clear(*(2,4)), *clear(*(3,1)), *clear(*(3,2)), *clear(*(3,3)), *clear(*(3,4)), *occupied(*(4,1)), *clear(*(4,2)), *clear(*(4,3)), *clear(*(4,4)), *occupied(*(5,1)), *clear(*(5,2)), *clear(*(5,3)), *clear(*(5,4)), *occupied(*(6,1)), *clear(*(6,2)), *clear(*(6,3)), *clear(*(6,4)), *occupied(*(1,0)), *occupied(*(2,0)), *occupied(*(3,0)), *occupied(*(4,0)), *occupied(*(5,0)), *occupied(*(6,0)), *on(*a,(1,2)), *on(*b,(2,2)), *on(*c,(1,1)), *on(*d,(4,1))])

Time limit exceeded