Trabalho 1 - Mundo dos Blocos

Pedro Miguel Pinto Botelho, Ricardo Augusto Silva Bonfim,
Rômulo José Pereira Da Costa Junior

[1]Instituto de Computação – Universidade Federal do Amazonas(UFAM)
Av. Gen. Rodrigo Octávio,6200, Coroado I,Setor Norte do Campus–69080–900

{pedro.botelho, ricardo.bonfim, romulo.junior}@icomp.ufam.edu.br

Introdução

Neste trabalho, criamos uma representação do mundo dos blocos e um planejador que realiza a tarefa de empilhar blocos de diferentes tamanhos. O objetivo é explorar as complicações que surgem quando precisamos organizar diferentes blocos. Usamos o livro do Ivan Bratko para encontrar uma solução, embora pareça fácil, foi um quebra-cabeça computacional. No fim, entendemos como a lógica de programação em Prolog pode ser uma ferramenta fundamental para resolver esse problema.

1. Proposição da linguagem

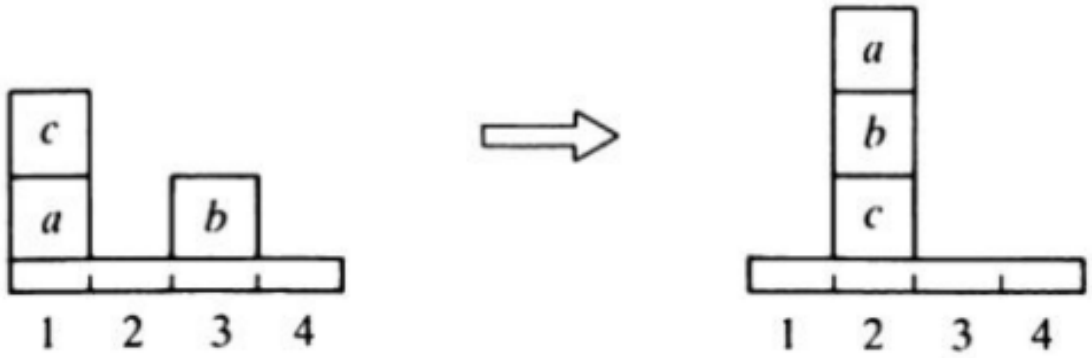

Figura 1. Figura do capítulo 17.1

1.1. Justificativa

Na nossa representação, temos os principais predicados como: **place(Number), at(X,Y), on(Block, at(X,Y)), can(move(Block,at(Xf, Yf), at(Xt, Yt)),L), adds(move(Block,at(Xf, Yf), at(Xt, Yt)),L), deletes(move(Block,at(Xf, Yf), at(Xt, Yt)),L)**:

- O predicado **"at"** representa uma coordenada no nosso plano, onde X é a posição no eixo X e Y é a posição no eixo Y.
- O predicado **"on"** é usado para representarmos a posição **(at)** de um bloco (Block) na grade.
- O predicado **"can"** é um predicado importante que usa muitos outros para verificar se uma ação **(move)** para uma posição **(at)** é de fato possível. Este faz considerações como tamanho do bloco (representado com **len(Block, BlockLength) e**

height(Block, BlockHeight)), posições livres no destino **(clearInterval)**, se um valor de posição está de fato na grade **(place)** e estabilidade do bloco.

- O predicado **"adds"** é um predicado similar ao **"can"** que usa muitos outros para verificar quais condições são **adicionadas** com uma ação.
- O predicado **"deletes"** é um predicado similar ao **"can"** que usa muitos outros para verificar quais condições são **removidas** com uma ação.

Observação: Consideramos o valor das coordenadas começando da esquerda de um bloco (para o eixo X) e de baixo pra cima (para o eixo Y).
Posições estão livres se for uma posição no chão **(at(X,0))** ou se há um bloco em baixo.
Estabilidade do bloco é checada se:

- Bloco possui **tamanho ímpar** e possui **pelo menos um** bloco sustentando o meio **ou dois** sustentando as pontas.
- Bloco possui **tamanho par** e possui **pelo menos dois** blocos sustentando as pontas.

Para simplificações, utilizamos apenas essas condições de estabilidade.

Dessa forma, para o estado à esquerda na imagem 17.1, podemos representá-lo com a seguinte lista: **[on(a, at(0,0)), on(b, at(2,0)), on(c, at(0,1)), clear(0,2), clear(1,0), clear(2,1), clear(3,0)]**. Para chegarmos no estado à direita na imagem, a lista deve conter: **on(a,at(1,2)), on(b, at(1,1)), on(c, at(1,0)), clear(0,0), clear(1,3), clear(2,0), clear(3,0)**. Note que como temos uma grade com tamanho 4 **(place(4))**, Logo, a posição mais alta e válida alcançável no estado à direita é a em cima de 'a'.

1.2. Código

```
%declaracao dos blocos
block(a).
block(b).
block(c).
block(d).


%declaracao dos lugares, funciona como uma matriz[6][6] (0..5)
place(0).
place(1).
place(2).
place(3).
place(4).
place(5).


%declaracao da altura dos blocos
height(a,1).
height(b,1).
height(c,1).
height(d,1).


%declaracao do comprimento dos blocos.
len(a,1).
```

```
len(b,1).
len(c,2).
len(d,3).


%declaracoes de estados de teste:
%final -> situação 1 final e state -> situação 1 inicial
%s30 -> situação 3 estado 0, s33 -> situação 3 estado 3, s37 -> situação 3 estado 7.

final([clear(0,0), clear(1,0), clear(2,0), on(d,at(3,0)), clear(3,1), on(a, at(4,1)), on(b, at(5,1)), on(c,at(4,2)), clear(4,3), clear(5,3)]).

state([on(c,at(0,0)), clear(0,1), clear(1,1), clear(2,0), on(a,at(3,0)), on(d, at(3,1)), clear(4,0), on(b,at(5,0)), clear(3,2), clear(4,2), clear(5,2)]).

s30([on(c, at(0,0)), on(d, at(3,1)), on(a,at(3,0)), on(b, at(5,0)), clear(0,1), clear(1,1), clear(2,0), clear(3,2), clear(4,2), clear(5,2), clear(4,0)]).

s37([on(a, at(0,1)), on(b,at(1,1)), on(c,at(0,0)), on(d, at(3,0)), clear(0,2), clear(1,2), clear(2,0), clear(3,1), clear(4,1), clear(5,1)]).

s36([on(a, at(0,1)), on(b,at(1,1)), on(c,at(0,0)), on(d, at(2,0)), clear(0,2), clear(1,2), clear(5,0), clear(3,1), clear(4,1), clear(2,1)]).

s34([on(a, at(0,1)), on(b,at(5,0)), on(c,at(0,0)), on(d, at(2,0)), clear(0,2), clear(1,1), clear(5,1), clear(3,1), clear(4,1), clear(2,1)]).

s33([on(a, at(5,1)), on(b,at(5,0)), on(c,at(0,0)), on(d, at(2,0)), clear(0,1), clear(1,1), clear(5,2), clear(3,1), clear(4,1), clear(2,1)]).


%funcao que me retorna uma lista de clear em um intervalo de x1 até um x2
clearInterval(X1, X2, _, []) :- X1 > X2, !. % Intervalo vazio


clearInterval(X1, X2, Y, [clear(X1, Y) | L]) :-
place(X1),
Xn is X1 + 1,
clearInterval(Xn, X2, Y, L).


%-----------------------------------------------
%predicado can -> temos uma lista L que tem as condições necessárias para realizar uma ação move


%definicao para bloco de comprimento 1
can(move(Block,at(Xf, Yf), at(Xt, Yt)),L):-
block(Block), % Verificando se é um bloco
len(Block, 1), % Verificando se tem comprimento 1
```

```
height(Block, BlockHeight),
place(Xf), %verificando os locais
place(Yf),
place(Xt),
place(Yt),
dif(at(Xf, Yf), at(Xt, Yt)), %verificando se a cordenada de origem é diferente da de destino
Y1 is Yf + BlockHeight, %altura em cima do
bloco original place(Y1),
addnew([on(Block, at(Xf, Yf)),clear(Xt, Yt)], [clear(Xf, Y1)], L). %adiciona as relacoes necessárias em L


%definicao para bloco de comprimento 2
can(move(Block,at(Xf, Yf), at(Xt, Yt)),[on(Block, at(Xf, Yf))|L]):-
block(Block),
len(Block, 2),
%verificando se comprimento é 2
height(Block, BlockHeight),
place(Xf),
place(Yf),
place(Xt),
place(Yt),
dif(at(Xf, Yf), at(Xt, Yt)),
X3 is Xf + 1,
X4 is Xt + 1,
place(X3), place(X4),
H is Yf + BlockHeight,
place(H), clearInterval(Xf, X3, H, L1), %verificando se nao tem nada no em cima bloco a ser movido
clearInterval(Xt, X4, Yt, L2), %verificando se o lugar onde o bloco irá ocupar está livre
append(L1, L2, L4),
XEnd is Xt + 1,
place(XEnd),
addnew([clear(Xt, Yt), clear(XEnd, Yt)], L4, L). %temos que garantir que o bloco está estável -> 2 blocos sustentando nas pontas


%caso bloco de comprimento nao par
%somente em blocos de comprimento impar podemos ter um bloco sustentando no meio
can(move(Block,at(Xf, Yf), at(Xt, Yt)),[on(Block, at(Xf, Yf))|L]):-
block(Block), % Verificando se um bloco
len(Block, BlockLength),
dif(BlockLength,1),
dif(BlockLength,2),
height(Block, BlockHeight),
place(Xf), %Verificando os locais
place(Yf),
```

```
place(Xt),
place(Yt),
dif(at(Xf, Yf), at(Xt, Yt)),
X3 is BlockLength + Xf - 1,
X4 is BlockLength + Xt - 1,
place(X3), place(X4),
Par is mod(BlockLength,2), %verificacao se eh impar
dif(Par,0),
H is Yf + BlockHeight,
place(H),
clearInterval(Xf, X3, Yf, L0),
clearInterval(Xf, X3, H, L1), %verificando se nao tem nada no em cima bloco a ser movido
clearInterval(Xt, X4, Yt, L2), %verificando se o lugar onde o bloco irá ocupar está livre
append(L1,L2,L3),
Mid is BlockLength // 2,
XMid is Xt + Mid,
place(XMid),
addnew([clear(XMid,Yt)], L3, L4), %adicionando cond de estabilidade -> se temos clear no meio -> bloco embaixo pois so temos clear em cima de um bloco.
delete_all(L4, L0, L).


can(move(Block,at(Xf, Yf), at(Xt, Yt)),[on(Block, at(Xf, Yf))|L]):-
block(Block), % Verificando se um bloco
len(Block, BlockLength),
dif(BlockLength,1),
dif(BlockLength,2),
height(Block, BlockHeight),
place(Xf), % Verificando os locais
place(Yf),
place(Xt),
place(Yt),
dif(at(Xf, Yf), at(Xt, Yt)),
X3 is BlockLength + Xf - 1,
X4 is BlockLength + Xt - 1,
place(X3), place(X4),
H is Yf + BlockHeight,
place(H),
clearInterval(Xf, X3, Yf, L0),
clearInterval(Xf, X3, H, L1), %verificando se nao tem nada no em cima bloco a ser movido
clearInterval(Xt, X4, Yt, L2), %verificando se o lugar onde o bloco irá ocupar está livre
append(L1,L2,L3),
XEnd is Xt + BlockLength - 1,
place(XEnd),
addnew([clear(Xt, Yt) ,clear(XEnd, Yt)], L3, L4), %cond de estabilidade -> temos blocos
```

```
sustentando nas pontas
delete_all(L4, L0, L).
%————————————————————————————-
% adds(Action, Relationships): Action establishes new Relationships
%caso de blocos de tamanho 1
adds(move(Block,at(Xf, Yf), at(Xt, Yt)),L):-
block(Block),
place(Xf),
place(Yf),
place(Xt),

place(Yt),
dif(at(Xf, Yf), at(Xt, Yt)),
len(Block, 1),
height(Block, BlockHeight),

Y1 is Yt + BlockHeight,
addnew([on(Block, at(Xt, Yt)), clear(Xt, Y1)], [clear(Xf, Yf)], L).


%caso de blocos de tamanho 2
adds(move(Block,at(Xf, Yf), at(Xt, Yt)),[on(Block,at(Xt, Yt))|L]):-
block(Block),
place(Xf),
place(Yf),
place(Xt),
place(Yt),
dif(at(Xf, Yf), at(Xt, Yt)),
len(Block, 2),
height(Block, BlockHeight),
X1 is Xt + 1,

Y1 is Yt + BlockHeight,
place(X1),
place(Y1),
clearInterval(Xt, X1, Y1, L1), %clear em cima do bloco no lugar novo
XMid is Xf + 1,
place(XMid),
addnew([clear(XMid,Yf), clear(Xf, Yf)], L1, L).


%caso geral
adds(move(Block,at(Xf, Yf), at(Xt, Yt)),[on(Block,at(Xt, Yt))|L]):-
block(Block),
place(Xf),
place(Yf),
place(Xt),
```

```
place(Yt),
dif(at(Xf, Yf), at(Xt, Yt)),
len(Block, BlockLength),
height(Block, BlockHeight),
X1 is Xt + BlockLength - 1,
X2 is Xf + BlockLength - 1,
Y1 is Yt + BlockHeight,
Y2 is Yf + BlockHeight,
place(X1), place(X2),

place(Y1), place(Y2),
clearInterval(Xf, X2, Y2, L5),
clearInterval(Xt, X1, Y1, L1), %clear em cima do bloco no lugar novo
XEnd is Xf + BlockLength - 1,
place(XEnd),
clearInterval(Xf, XEnd, Yf, L2),
clearInterval(Xt, X1, Yt, L3),
append(L1, L2, L4),
delete_all(L4, L3, L6), delete_all(L6, L5, L).
%-----------------------------------------------
% deletes(Action, Relationships): Action destroy Relationships
% caso bloco de tamanho 1
deletes(move(Block,at(Xf, Yf), at(Xt, Yt)),L):-
block(Block),
place(Xf),
place(Yf),
place(Xt),
place(Yt),
dif(at(Xf, Yf), at(Xt, Yt)),
len(Block, 1),
height(Block, BlockHeight),
Y1 is Yf + BlockHeight,
addnew([on(Block, at(Xf, Yf)), clear(Xf, Y1)], [clear(Xt, Yt)], L).

% caso bloco de tamanho 2
deletes(move(Block,at(Xf, Yf), at(Xt, Yt)),[on(Block,at(Xf, Yf))|L]):-
block(Block),
place(Xf),
place(Yf),
place(Xt),
place(Yt),
dif(at(Xf, Yf), at(Xt, Yt)),
len(Block, 2),
height(Block, BlockHeight),
X1 is Xf + 1,
Y1 is Yf + BlockHeight,
```

```
place(X1),
place(Y1),
clearInterval(Xf, X1, Y1, L1), %clear em cima do bloco no lugar antigo
XMid is Xt + 1,
place(XMid),
addnew([clear(XMid,Yt), clear(Xt, Yt)], L1, L).


%caso geral
deletes(move(Block,at(Xf, Yf), at(Xt, Yt)),[on(Block,at(Xf, Yf))|L]):-
block(Block),
place(Xf),
place(Yf),
place(Xt),
place(Yt),
dif(at(Xf, Yf), at(Xt, Yt)),
len(Block, BlockLength),
height(Block, BlockHeight),
X1 is BlockLength + Xf - 1,
Y1 is Yf + BlockHeight,
place(X1),
place(Y1),
X2 is BlockLength + Xt - 1,
Y2 is BlockHeight + Yt,
place(X2),
place(Y2),
clearInterval(Xt, X2, Y2, L5),
clearInterval(Xf, X1, Y1, L1), %clear em cima do bloco no lugar antigo
XEnd is Xt + BlockLength - 1,
Y3 is Yt,
place(XEnd),
place(Y3),
clearInterval(Xt, XEnd, Y3, L2), %clears dos blocos abaixo do bloco atual
clearInterval(Xf, X1, Yf, L3),
append(L1,L2,L4),
delete_all(L4, L3, L6),
delete_all(L6, L5, L).

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

2. O Planejador

2.1. Explicação

Considerando o código fornecido do planejador. Uma das principais mudanças da nossa abordagem é **sempre selecionar uma goal que ainda não está satisfeita** (presente no estado inicial e nas goals atuais). De resto, foram poucas mudanças em alguns predicados que não estavam com o funcionamento adequado, como o **"preserves"** e o **"select"**.

Tirando isso, o planejador segue o modelo de Means-Ends combinado com Goal Regression. O planejador seleciona uma goal ainda não resolvida, e verifica quais as possíveis ações que resultam nessa goal, uma vez achada uma ação compatível (não quebra as outras goals e implica na goal atual), regredimos a partir dessa ação e goals antigas, e obtemos novas goals (Tiramos as que foram satisfeitas e adicionamos as pré-condições).

2.2. Código

```
isEmpty([]).


%impossivel um bloco estar em 2 lugares ao mesmo tempo
impossible(on(Block,at(X1,Y1)),Goals):-
member(on(Block2,at(X1,Y1)),Goals),
dif(Block,Block2),
!.


%impossivel um bloco estar em diferentes cordenadas ao mesmo tempo
impossible(on(Block,at(X1,Y1)),Goals):-
member(on(Block,at(X2,Y2)),Goals),
dif(at(X1,Y1),at(X2,Y2)),
!.


%impossivel ter clear em goals se temos um bloco em goals naquela mesma posicao
impossible(clear(X1,Y1), Goals):-
member(on(_, at(X1, Y1)), Goals),
!.


plan(State, Goals, []):-
satisfied(State, Goals). % caso base


plan(State, Goals, Plan):-
append(PrePlan, [Action], Plan), % estrategia me apresentada no livro
select(State, Goals, Goal), % selecione um goal G não resolvido em Goals
achieves(Action, Goal), % procurar uma ação A que alcança G
preserves(Action,Goals), % garantir que A não quebre os Goals
regress(Goals, Action, RegressedGoals), % fazer a regressão
plan(State, RegressedGoals, PrePlan).


% verifica recursivamente tirando o cabeça da lista se todos os Goals
% estão presentes no estado atual 'State'
satisfied(_, []). % caso base (sem goals)


satisfied(State, [Goal|Goals]):-
member(Goal, State), % verifica de Goal está presente no estado atual State
satisfied(State, Goals). %verifica o resto das goals
```

```
select(State, Goals, Goal):- %verifica se Goal pertence a Goals
delete_all(Goals, State, GoalsNResolvidas), %gera um conjunto com apenas as goals ainda nao resolvidas
member(Goal, GoalsNResolvidas). %seleciona uma goal desse conjunto


% verifica se uma ação Action adiciona algo a lista Goals, e se Goal pertence a Goals.
% Ou seja, acredito que seja para verificar se ao realizar Action,
% algum efeito na lista Goals acontece.
achieves(Action, Goal):-
adds(Action, Goals),
member(Goal, Goals).


%verifica se uma ação não quebra algum Goal em Goals
preserves(Action , Goals):-
deletes(Action, Relations),
naoQuebra(Relations, Goals).


%verifica se alguma das goals está sendo quebrada
naoQuebra([H|_], Goals):-
member(H, Goals),
!,
fail.


naoQuebra([_|T], Goals):-
naoQuebra(T, Goals).


naoQuebra([], _).


regress(Goals, Action, RegressedGoals):-
adds(Action, NewRelations),
delete_all(Goals, NewRelations, RestGoals),
deletes(Action,Condition),
addnew(Condition, RestGoals, RegressedGoals).


%addnew(L1, L2, L3) -> (L2 - L1) + L2 = L3
addnew([], L, L).
addnew([Goal | _], Goals, _):-
impossible(Goal, Goals),
!,
fail.


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

```
member(X,L2),
!,
addnew(L1, L2, L3).

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

%delete_all(L1,L2,Diff): if Diff is set-difference of 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).

%funcao para testar a capacidade de gerar um plano L do Inicio até as Goals.
testarPlano(L):-
s33(Inicio),
s37(Goals),
plan(Inicio, Goals, L).
```

3. Geração Manual de Planos com a Linguagem

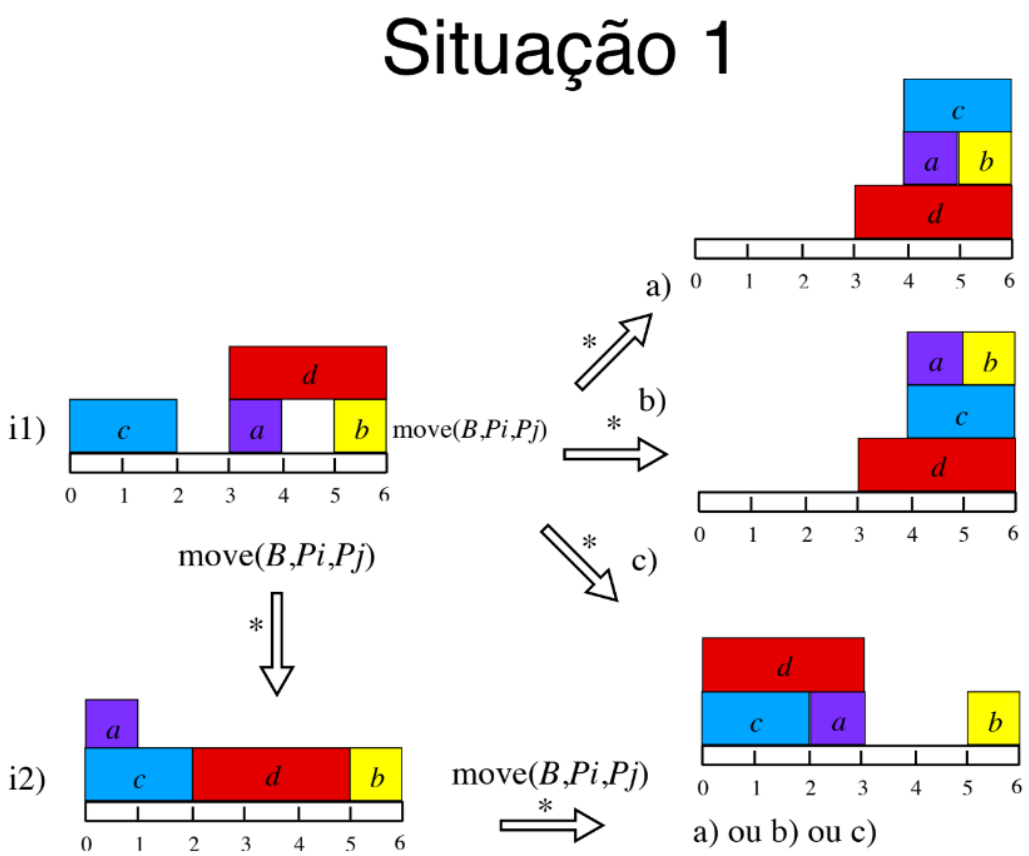

Figura 2. Figura da questão 3.

3.1. Questão 3.1: s_inicial=i1 ate o estado s_final=i2

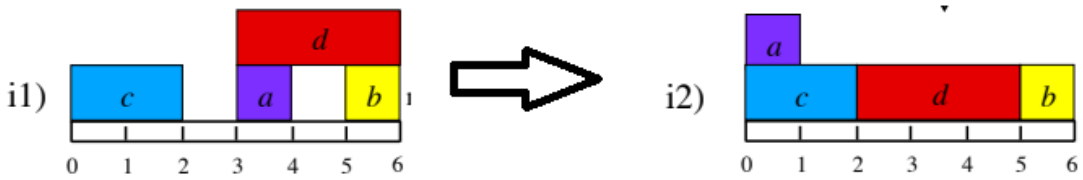

Figura 3. Figura da questão 3.1

Estado inicial: [on(a, at(1,0)), on(b, at(5,0)), on(c, at(0,0)), on(d,at(2,0)), clear(0,2), clear(1,1), clear(2,1), clear(3,1), clear(4,1), clear(5,1)];

Estado final: [on(a, at(3,0)), on(b, at(5,0)), on(c, at(0,0)), on(d,at(3,1)), clear(2,0), clear(4,0), clear(0,1), clear(1,1), clear(3,2), clear(4,2), clear(5,2)];

Ações estilo backtracking: move(a, at(0,1), at(5,0)), move(d, at(2,0), at(0,1)), move(a, at(5,0), at(3,0)), move(d, at(0,1), at(3,1)).

Ações reais: move(d, at(3,1), at(0,1), move(a, at(3,0), at(5,1)), move(d, at(0,1), at(2,0)), move(a, at(5,1), at(0,1))

3.2. Questão 3.2: s_inicial=i2 ate o estado s_final=i2 (a).

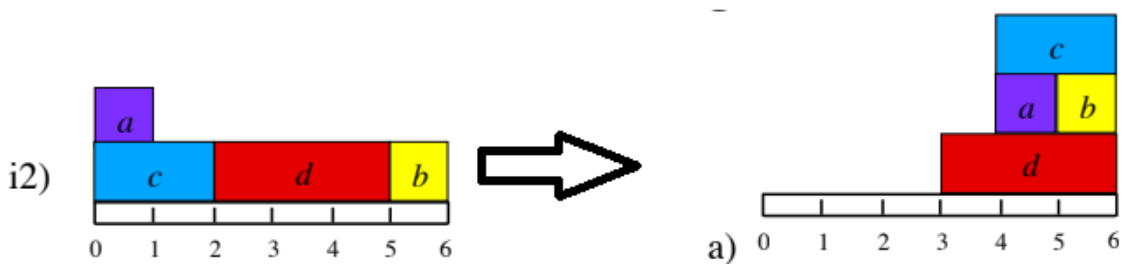

Figura 4. Figura da questão 3.2

Estado inicial: [on(a, at(4,1)), on(b, at(5,1)), on(c, at(4,2)), on(d, at(3,0)), clear(0,0), clear(1,0), clear(2,0), clear(3,1), clear(4,3), clear(5,3)];

Estado final: [on(a, at(0,1)), on(b, at(5,0)), on(c, at(0,0)), on(d, at(2,0)), clear(0,2), clear(1,1), clear(2,1), clear(3,1), clear(4,1), clear(5,1)];

Ações estilo backtracking: [move(c, at(0,0), at(4,2)), move(a, at(0,1), at(4,1)), move(b, at(1,1), at(5,1)), move(d, at(2,0), at(3,0)), move(b, at(1,1), at(5,0))];

Ações reais: [move(b, at(1,1), at(5,0)), move(d, at(2,0), at(3,0)), move(b, at(1,1), at(5,1)), move(a, at(0,1), at(4,1)), move(c, at(0,0), at(4,2));

3.3. Questão 3.3: s_inicial=i2 ate o estado s_final=i2 (b).

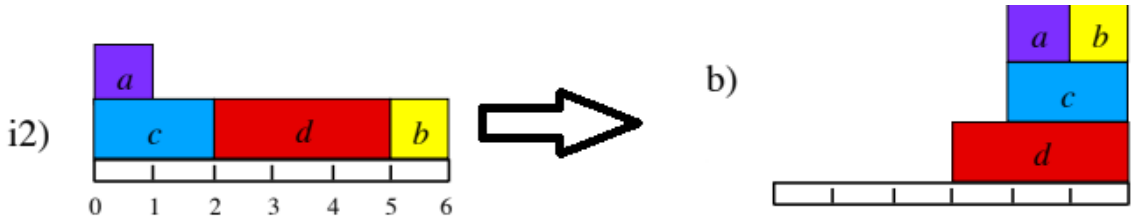

Figura 5. Figura da questão 3.3

Estado inicial: [on(a, at(4,2)), on(b,at(5,2)), on(c, at(4,1)), on(d, at(3,0)), clear(0,0), clear(1,0), clear(2,0), clear(3,1), clear(4,3), clear(5,3)];

Estado final: [on(a, at(0,1)), on(b, at(5,0)), on(c, at(0,0)), on(d, at(2,0)), clear(0,2), clear(1,1), clear(2,1), clear(3,1), clear(4,1), clear(5,1)];

Ações estilo backtracking: move(a, at(4,2), at(3,1)), move(b, at(5,2), at(2,0)), move(c, at(4,1), at(0,0), move(a, at(3,1), at(0,1)), move(b, at(2,0), at(1,1)), move(d, at(3,0), at(2,0), move(b, at(1,1), at(5,0));

Ações reais: move(b, at(5,0), at(1,1)), move(d, at(2,0), at(3,0)), move(b, at(1,1), at(2,0)), move(a, at(0,1), at(3,1)), move(c, at(0,0), at(4,1), move(b, at(2,0), at(5,2)), move(a, at(3,1), at(4,2));

3.4. Questão 3.4: s_inicial=i2 ate o estado s_final=i2 (c).

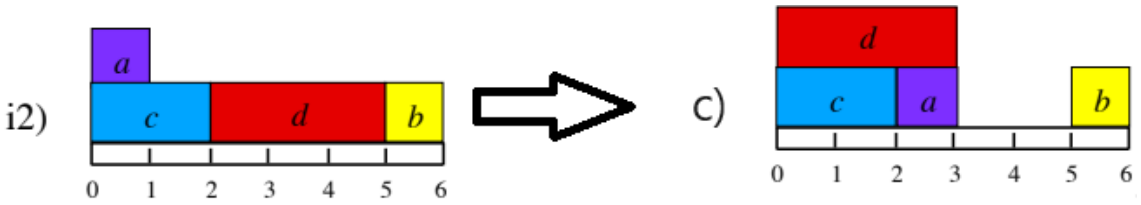

Figura 6. Figura da questão 3.4

Estado inicial: [on(a, at(4,2)), on(b,at(5,2)), on(c, at(4,1)), on(d, at(3,0)), clear(0,0), clear(1,0), clear(2,0), clear(3,1), clear(4,3), clear(5,3)];

Estado final: [on(a, at(0,1)), on(b, at(5,0)), on(c, at(0,0)), on(d, at(2,0)), clear(0,2), clear(1,1), clear(2,1), clear(3,1), clear(4,1), clear(5,1)];

Ações estilo backtracking: move(b, at(5,0), at(4,0)), move(d, at(0,1), at(3,1)), move(a, at(2,0), at(1,1)), move(d, at(3,0), at(0,2)), move(b, at(4,0), at(5,0)), move(d, at(0,2), at(2,0)), move(a, at(1,1), at(0,1));

Ações reais: move(a, at(0,1), at(1,1)), move(d, at(2,0), at(0,2)), move(b, at(5,0), at(4,0)), move(d, at(0,2), at(3,1)), move(a, at(1,1), at(2,0)), move(d, at(3,1), at(0,1)), move(b, at(4,0), at(5,0));

3.5. Questão 3.5

Para as situações 2 e 3, nosso planner precisa realizar uma regressão de goals, que consiste em a partir de um estado (S), descobrir quais goals em S são necessários para, a partir de uma ação A, garantir que esses goals sejam verdade em outro estado (S').

A ação A deve seguir o seguinte comportamento:

1. deve ser possível em S, ou seja, possui as pré-condições para A em S;
2. para cada goal (G) em goals, ou a ação A adiciona G, ou G está em goals em S e A não deleta G;

Nesse sentido, queremos em algum momento alcançar uma lista de objetivos que descrevem o estado atual do mundo. Para isso, usamos o seguinte algoritmo: se a lista de goals (Goals) já é satisfeita no estado atual, nada precisa ser feito. Caso contrário, selecionamos um goal (G) em Goals (Apenas os goals não pertencentes ao estado inicial) e a ação (A) que alcança G. Daí, regredimos Goals através de A obtendo novos goals e buscamos encontrar um novo plano a partir do estado inicial para alcançar os novos goals.

Para descobrir essas ações (A) que formam o plano, precisamos utilizar uma heurística de amplitude em primeiro lugar, onde os planos menores são testados primeiro.

Para eficiência, utilizamos predicados como impossible para eliminar predicados que não são possíveis.

Situação 2:

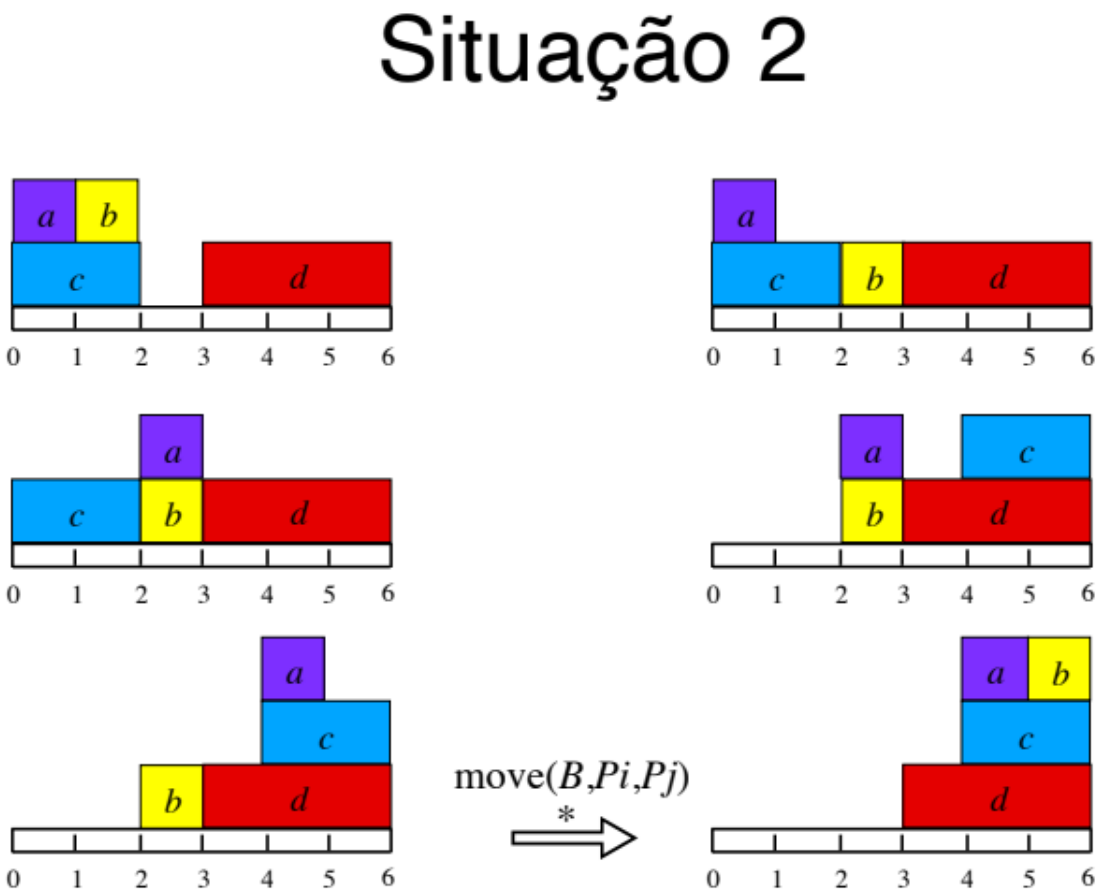

Figura 7. Figura da situação 2

Para esta situação, iniciamos do estado final representado pela lista de predicados a seguir: [on(a, at(4,2)), on(b, at(5,2)), on(c, at(4,1)), on(d, at(3,0)), clear(0,0), clear(1,0), clear(2,0), clear(3,1), clear(4,3), clear(5,3)]. Porém, queremos chegar no estado-Goal: [on(a, at(0,1)), on(b, at(1,1)), on(c,at(0,0)), on(d, at(3,0)), clear(0,2), clear(1,2), clear(2,0), clear(3,1), clear(4,1), clear(5,1)]. Logo, precisamos encontrar algum plano para chegar nesse estado por backtracking. Vou listar a sequência de passos:

Ação1: move(b, at(2,0), at(5,2))

- Removidos: on(b,at(2,0)), clear(2,1), clear(5,2)
- Adicionados: on(b, at(5,2)), clear(2,0), clear(5,3)
- Estado atual: [on(a, at(4,2)), on(b, at(2,0)), on(c, at(4,1)), on(d, at(3,0)), clear(0,0), clear(1,0), clear(2,1), clear(3,1), clear(4,3), clear(5,2)]

Ação2: move(a, at(2,1), at(4,2))

- Removidos: on(a,at(2,1)), clear(2,2), clear(4,2)
- Adicionados: on(a, at(4,2)), clear(2,1), clear(4,3)
- Estado atual: [on(a, at(2,1)), on(b, at(2,0)), on(c, at(4,1)), on(d, at(3,0)), clear(0,0), clear(1,0), clear(2,2), clear(3,1),clear(4,2), clear(5,2)]

Ação3: move(c at(0,0), at(4,1))

- Removidos: on(c,at(0,0)), clear(0,1), clear(1,1), clear(4,1), clear(5,1)

- Adicionados: on(c, at(4,1)), clear(0,0), clear(1,0), clear(4,2), clear(5,2)
- Estado atual: [on(a, at(2,1)), on(b, at(2,0)), on(c, at(0,0)), on(d, at(3,0)), clear(0,1), clear(1,1), clear(2,2), clear(3,1), clear(4,1), clear(5,1)]

Ação4: move(a at(0,1), at(2,1))

- Removidos: on(a,at(0,1)), clear(0,2), clear(2,1)
- Adicionados: on(a at(2,1)), clear(0,1), clear(2,2)
- Estado atual: [on(a, at(0,1)), on(b, at(2,0)), on(c, at(0,0)), on(d, at(3,0)), clear(0,2), clear(1,1), clear(2,1), clear(3,1), clear(4,1), clear(5,1)]

Ação5: move(b at(1,1), at(2,0))

- Removidos: on(b,at(1,1)), clear(1,2), clear(2,0)
- Adicionados: on(b, at(2,0)), clear(1,1), clear(2,1)
- Estado atual: [on(a, at(0,1)), on(b, at(1,1)), on(c, at(0,0)), on(d, at(3,0)), clear(0,2), clear(1,2), clear(2,0), clear(3,1), clear(4,1), clear(5,1)]

Note que estado atual agora é igual ao estadoGoal, seguindo as ações de 5 até 1, nessa ordem.

Situação 3:

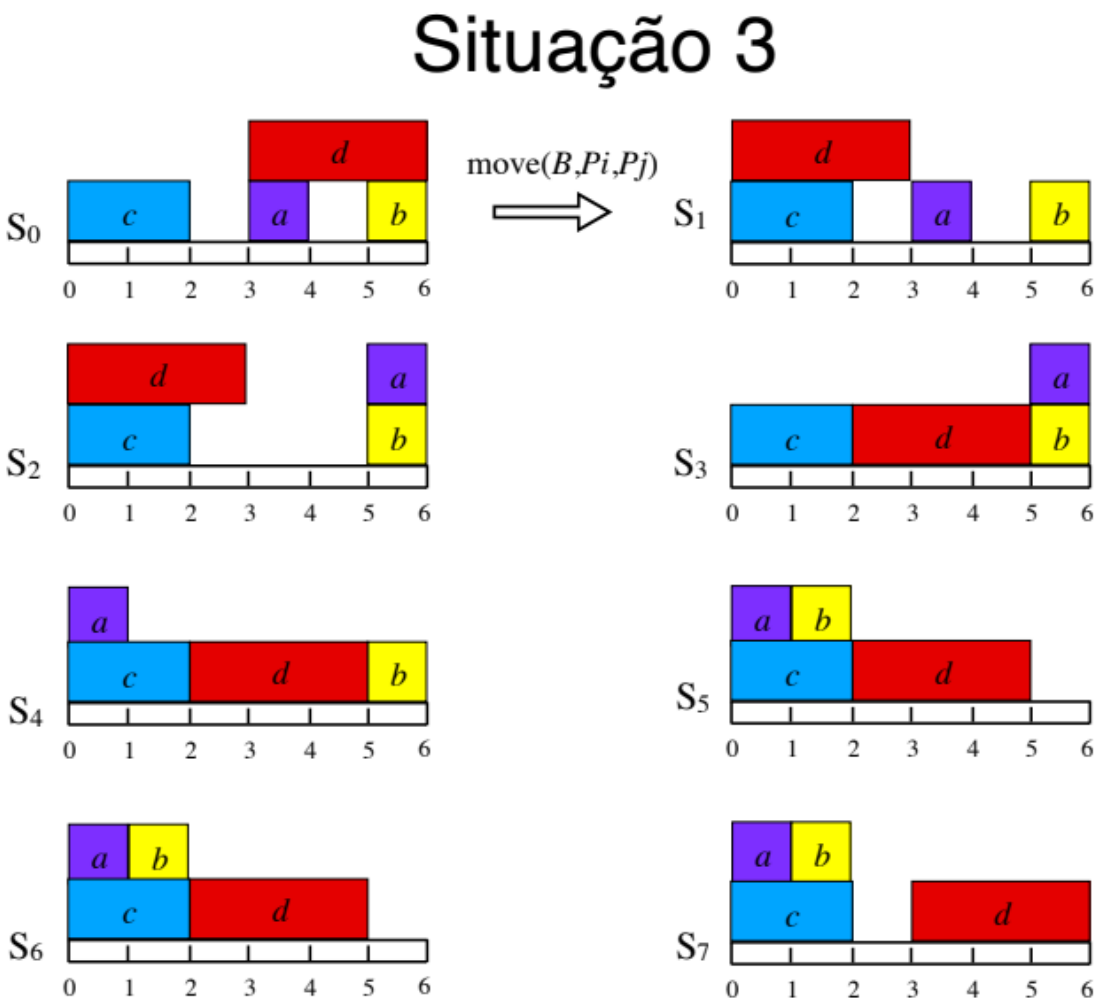

Figura 8. Figura da situação 3

Para esta situação, iniciamos do estado final representado pela lista de predicados a seguir: [on(a, at(0,1)), on(b, at(1,1)), on(c, at(0,0)), on(d, at(3,0)), clear(0,2), clear(1,2), clear(2,0), clear(3,1), clear(4,1), clear(5,1)]. Porém, queremos chegar no estado-Goal: [on(a, at(3,0)), on(b, at(5,0)), on(c,at(0,0)), on(d, at(3,1)), clear(0,1), clear(1,1), clear(2,0), clear(3,2), clear(4,2), clear(5,2)]. Logo, precisamos encontrar algum plano para chegar nesse estado por backtracking. Vou listar a sequência de passos:

Ação1: move(d, at(2,0), at(3,0))

- Removidos: on(d,at(2,0)), clear(2,1), clear(3,1), clear(4,1), clear(5,0)
- Adicionados: on(d, at(3,0)), clear(2,0), clear(5,1)
- Estado atual: [on(a, at(0,1)), on(b, at(1,1)), on(c, at(0,0)), on(d, at(2,0)), clear(0,2), clear(1,2), clear(2,1), clear(3,1), clear(4,1), clear(5,0)]

Ação2: move(b, at(5,0), at(1,1))

- Removidos: on(b,at(5,0)), clear(5,1), clear(1,1)
- Adicionados: on(b, at(1,1)), clear(5,0), clear(1,2),
- Estado atual: [on(a, at(0,1)), on(b, at(5,0)), on(c, at(0,0)), on(d, at(2,0)), clear(0,2), clear(1,1), clear(2,1), clear(3,1), clear(4,1), clear(5,1)]

Ação3: move(a, at(5,1), at(0,1))

- Removidos: on(a,at(5,1)), clear(5,2), clear(0,1)
- Adicionados: on(a, at(0,1)), clear(5,1), clear(0,2),
- Estado atual: [on(a, at(5,1)), on(b, at(5,0)), on(c, at(0,0)), on(d, at(2,0)), clear(0,1), clear(1,1), clear(2,1), clear(3,1), clear(4,1), clear(5,2)]

Ação4: move(d, at(0,1), at(2,0))

- Removidos: on(d,at(0,1)), clear(2,0), clear(3,0), clear(4,0), clear(0,2), clear(1,2), clear(2,2) Adicionados: on(d, at(2,0)), clear(0,1), clear(1,1), clear(2,1), clear(3,1), clear(4,1)
- Estado atual: [on(a, at(5,1)), on(b, at(5,0)), on(c, at(0,0)), on(d, at(0,1)), clear(0,2), clear(1,2), clear(2,2), clear(2,0), clear(3,0), clear(4,0), clear(5,2)]

Ação5: move(a, at(3,0), at(5,1))

- Removidos: on(a,at(3,0)), clear(3,1), clear(5,1)
- Adicionados: on(a, at(5,1)), clear(3,0), clear(5,2)
- Estado atual: [on(a, at(3,0)), on(b, at(5,0)), on(c, at(0,0)), on(d, at(0,1)), clear(0,2), clear(1,2), clear(2,2), clear(3,1), clear(4,0), clear(5,1)]

Ação6: move(d, at(3,1), at(0,1))

- Removidos: on(d,at(3,1)), clear(0,1), clear(1,1), clear(3,2) clear(4,2), clear(5,2)
- Adicionados: on(d, at(0,1)), clear(3,1), clear(5,1), clear(0,2), clear(1,2), clear(2,2), clear(4,0)
- Estado atual: [on(a, at(3,0)), on(b, at(5,0)), on(c, at(0,0)), on(d, at(3,1)), clear(0,1), clear(1,1), clear(2,0), clear(3,2), clear(4,2), clear(5,2)]