Trabalho 1- Mundo dos Blocos

Gilmar Couto Júnior [22152247] | Luís Eduardo [22251141] | Gabrielly Rodrigues [22152262]

Representação do conhecimento

1. Visão Geral do Projeto

1.1 Link do Projeto Github

https://github.com/GilmarCoutoJr/Trab_Mundo_dos_Blocos

1.2 Descrição do problema

O problema consiste em um mundo formado por blocos que podem ser empilhados uns sobre os outros ou colocados diretamente sobre a mesa. Os blocos são de diferentes **comprimentos** (larguras), mas **iguais em altura**. A principal diferença visual entre eles é, portanto, o comprimento. O objetivo do sistema é mover os blocos de modo a transformar um estado inicial do mundo em um estado final desejado, respeitando restrições como não poder colocar blocos sobre outros blocos que já estão "ocupados" (ou seja, que já têm algo em cima).

Por exemplo, podemos ter blocos nomeados por letras (a, b, c, d, etc.), e relações como:

- on(a, b) (o bloco **a** está sobre o bloco **b**);
- on(b, table) (o bloco **b** está diretamente sobre a mesa).

O sistema precisa ser capaz de gerar planos (sequências de movimentos) para reorganizar os blocos a partir de um estado inicial até atingir um estado final.

2. Conceito de livre (clear) e outras definições necessárias. Estado do mundo como uma lista com as relações de sobreposição (como on(X,Y))

I. Blocos e Estado do Mundo:

- Cada bloco é identificado por um único nome (como a, b, c...).
- A mesa é representada por um termo especial, geralmente table.
- O estado do mundo é descrito por uma lista de fatos da forma on(X, Y), que significa "o bloco X está sobre Y", onde Y pode ser outro bloco ou a mesa (table).

Exemplo de estado:

```
[on(a, b), on(b, table), on(c, table)]
```

II. Definição de livre (clear):

- Um bloco é considerado livre (clear) se não há nenhum outro bloco sobre ele.
- Em Prolog, podemos definir isso verificando se não existe nenhum fato on(AlgumBloco, X) no estado atual.

III. Outros conceitos necessários:

- Move (mover): Ação de mover um bloco B da posição Pi para uma nova posição Pj.
- Posição válida: Um bloco só pode ser movido se estiver livre, e só pode ser colocado sobre outro bloco se o destino também estiver livre (ou diretamente sobre a mesa).

IV. Estado válido:

- Cada bloco está em exatamente uma posição (ou seja, há exatamente um fato on(B, AlgumaCoisa) para cada bloco B).
- Não existem ciclos (por exemplo, não pode haver on(a, b) e on(b, a)).

3. Representação adequada de termos em lógica para representar os elementos descritos no documento anexo, sem o uso de assign e retract via chatGPT

I. Estado do Mundo:

- Representamos o estado como uma lista de fatos do tipo on(X, Y), por exemplo:

```
[on(c,table), on(a,table), on(b,table), supports(d,[a, b])]
```

Obs.: "`supports(d,[a, b])`**" representa que "**`d`**" está sobre "**`a`**" e "**`b`**" ao mesmo tempo.**

II. Predicados livre/2 e livre_destino/2:

- Um bloco X é livre (clear) se nenhum outro bloco está sobre ele no estado atual. Adicionalmente, também precisamos saber se o local para onde o bloco vai está liberado. Podemos definir isso como:

```
livre(Bloco, Estado) :-

    \+ (member(bloco(_, _, Bloco), Estado)).

livre_destino(Posicao, Estado) :-

    \+ member(bloco(_, Posicao), Estado).
```

III. Predicados retira_bloco/4 e coloca_bloco/4

- Estes predicados servem, respectivamente, para atualizar o estado ao remover um bloco de sua posição e colocar um bloco em uma nova posição, também atualizando o estado:

```
retira_bloco(Bloco, Posicao, [bloco(Bloco, Posicao) | Resto],
Resto) :-

     write('Retirando bloco '), write(Bloco), write(' da posição
'), write(Posicao), nl.

retira_bloco(Bloco, Posicao, [OutroBloco | Resto], [OutroBloco |
NovoResto]) :-

    retira_bloco(Bloco, Posicao, Resto, NovoResto).
```

```
coloca_bloco(Bloco, Posicao, Estado, [bloco(Bloco, Posicao) | Estado]) :-

    write('Colocando bloco '), write(Bloco), write(' na posição '), write(Posicao), nl.
```

III. Predicado move/5:

- Para mover um bloco B de sua posição Pi para uma nova posição Pj, gerando um novo estado:

```
move(Bloco, De, Para, Estado, NovoEstado) :-
    livre(Bloco, Estado),
    livre_destino(Para, Estado),
    \+ member(bloco(Bloco, Para), Estado),
    retira_bloco(Bloco, De, Estado, TempEstado),
    coloca_bloco(Bloco, Para, TempEstado, NovoEstado).
```

IV. Predicado para checar objetivo (satisfied_goal/1):

- Predicado para verificar se uma meta já está satisfeita:

```
satisfied_goal(livre(Bloco)) :-
    nonvar(Bloco),
    estado_inicial(Estado),
```

```
    livre(Bloco, Estado).


satisfied_goal(livre_destino(Posicao)) :-
    nonvar(Posicao),
    estado_inicial(Estado),
    livre_destino(Posicao, Estado).
```

V. Planejamento (plan/3):

- Gera um plano de ações para mover os blocos do estado inicial para o final:

```
plan(_, _, Plano) :-
    estado_inicial(EstadoInicial), % Expande o estado inicial
    estado_final(EstadoFinal),     % Expande o estado final
    regress(EstadoFinal, _, Plano),
    executar_plano(Plano, EstadoInicial, EstadoFinal).
```

Obs.: Em nenhum momento foi utilizado *assert/retract*.

Raciocínio e planejamento automático

Cada integrante utilizou um chatbot diferente, realizando diversas perguntas e passando algumas restrições. Infelizmente, não foram estabelecidos padrões exatos para a formulação das instruções, mas, as restrições de maior interesse seriam os parâmetros que definem um bloco livre (que pode ser movido), além da própria definição do mundo dos blocos que estamos tratando: Um mundo com blocos de diferentes comprimentos, porém de mesma altura sobre uma mesa. Vale ressaltar que a mesa do nosso mundo dos blocos é segmentada de uma forma diferente do que algumas .

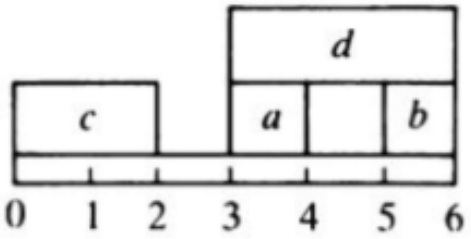


Imagem retirada do livro do Bratko

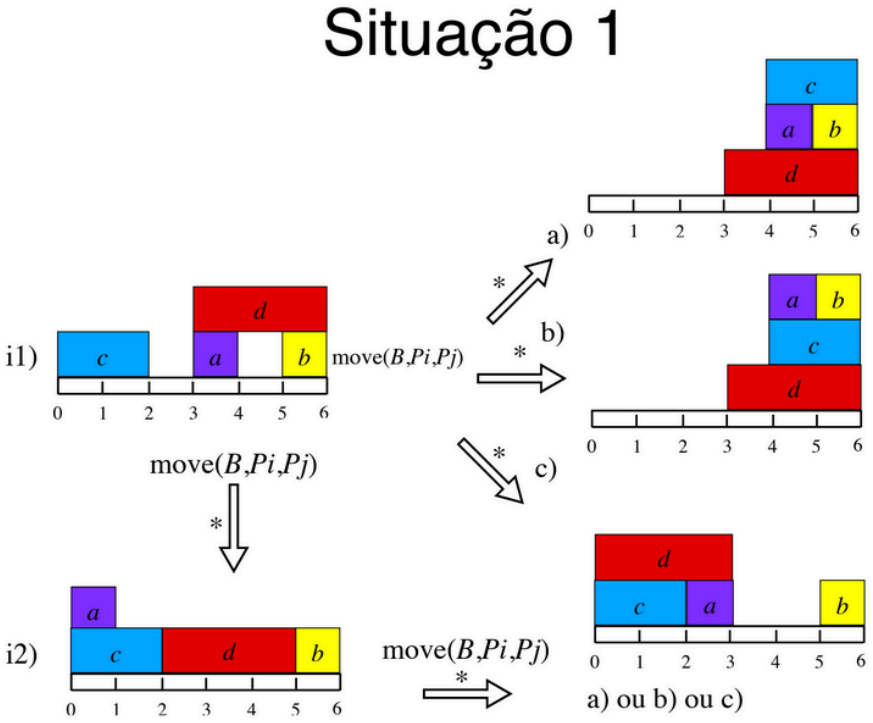
Observa-se que a mesa é segmentada de uma forma bem peculiar, ao invés de considerarmos uma partição da mesa como uma posição no mundo dos blocos, devemos contar o final da partição e o final dela, ou seja, neste mundo um bloco “f” que antes estava ocupava a posição 2 em outro mundo, agora ocupa a posição [1, 2] (1 seria o início do segmento e 2 o final dele).

Devido à complexidade dessa representação, foi difícil traduzir todas as restrições apenas por meio da linguagem natural. Isso levou a equipe a optar pelo uso de imagens como forma de complementar a comunicação com os chatbots, o que facilitou o avanço no desenvolvimento do projeto.

Após o estabelecimento das restrições, cada integrante copiou o código disponibilizado no livro do Bratko e utilizou com seus respectivos chatbots. Notou-se que o chatbot mais promissor na criação do código foi o ChatGPT. Vale ressaltar que vários fatores podem ter influenciado essa conclusão, como a falta do estabelecimento de um padrão apropriado para formular as perguntas (como foi dito anteriormente) ou a qualidade das próprias perguntas feitas por cada integrante.

A seguir, são apresentados os detalhes de cada transição entre estados para resolver a **situação 1**, incluindo os movimentos necessários e as respectivas saídas obtidas a partir da execução do código:

1. s_inicial=i1 até o estado s_final=i2
2. s_inicial=i2 até o estado s_final=i2 (a).
3. s_inicial=i2 até o estado s_final=i2 (b).
4. s_inicial=i2 até o estado s_final=i2 (c).
5. (i1) para o estado (i2)



move(B, Pi, Pj) == move(BlocoASerMovido, PosInicial, PosFinal)

1. Para sair do estado i1 inicial e chegar ao estado i2 final (i1 a i1(c)), primeiro movemos o bloco **d** para cima do bloco **c**, e assim ele já fica na sua posição final, move(D, (a,b), c(0,3)). Depois, basta mover o bloco **a** para sua posição final, move(A, table(3,4), table(2,3)).

Saída do código:

```
Block c is free
Block a is free
Position [2, 3] is free
```

```
Action move(a, _, [2,3]) can be done with preconditions
[livre(a),livre_destino([2,3])]
Regressed goals: [livre(a), livre_destino([2, 3]), bloco(c,[0,
2]), bloco(b,[5, 6]), bloco(d,[0, 3])]
Block b is free
Block d is free
Position [0, 3] is free
Action move(d, _, [0,3]) can be done with preconditions
[livre(d),livre_destino([0,3])]
Regressed goals: [livre(d), livre_destino([0, 3]), bloco(c,[0,
2]), bloco(a,[2, 3]), bloco(b,[5, 6])]
```

2. Para sair do estado inicial i2, e chegar no estado final i2(a), o primeiro movimento será posicionar o bloco **b** sobre o bloco **c**, considerando que ele vai ficar no canto direito de **c**, move(B, (5,6), c(1,2)). O segundo bloco movimento será mover o bloco **d**, cele vai ficar posicionado no canto, sua posição final, move(D, (2,5), (3,6)). Depois os blocos **a** e **b** vão sair de cima do **c** para que ele possa se movimentar, e vão ficar nas suas posições finais, move(A c(0,1), d(4,5)) e move(B, c(1,2), d(5,6)). Por fim, o bloco **c** vai para cima dos blocos **a** e **b**, move(C, (0,2) (a,b)).

   Saída do código:

```
Block c is free
Position [4, 6] is free
Action move(c, _, [4,6]) can be done with preconditions
[livre(c),livre_destino([4,6])]
Regressed goals: [livre(c), livre_destino([4, 6]), bloco(a,[4,
5]), bloco(b,[5, 6]), bloco(d,[3, 6])]
Block a is free
Position [4, 5] is free
Action move(a, _, [4,5]) can be done with preconditions
[livre(a),livre_destino([4,5])]
Regressed goals: [livre(a), livre_destino([4, 5]), bloco(c,[4,
6]), bloco(b,[5, 6]), bloco(d,[3, 6])]
Block b is free
Block d is free
Position [3, 6] is free
Action move(d, _, [3,6]) can be done with preconditions
[livre(d),livre_destino([3,6])]
Regressed goals: [livre(d), livre_destino([3, 6]), bloco(c,[4,
6]), bloco(a,[4, 5]), bloco(b,[5, 6])]
```

3. Para sair do estado inicial i2, e chegar no estado final i2(b), o primeiro movimento será tirar o bloco **b** da posição atual para que o bloco **d** possa se movimentar,

move(B, (5,6), c(1,2)). Depois, o bloco **d** se movimenta para sua posição final, move(D, (2,5), (3,6)). Nos próximos 2 movimentos, os blocos **a** e **b** vão sair de cima do bloco **c** para que ele possa se movimentar para sua posição final, move(A, c(0,1), d(3,4)) e move(B, c(1,2), a(3,4)). Depois, o bloco **c** pode se mover para sua posição final sobre o bloco **d**, move(C, (0,2), d(4,6)). Por fim, os dois últimos movimentos são para posicionar os blocos **a** e **b** nas suas posições finais, sobre o bloco **c**, move(B, a(3,4), c(5,6)) e move(A, d(3,4), c(4,5)).

Saída do código:

```
Block a is free
Position [4, 5] is free
Action move(a, _, [4,5]) can be done with preconditions
[livre(a),livre_destino([4,5])]
Regressed goals: [livre(a), livre_destino([4, 5]), bloco(b,[5,
6]), bloco(c,[4, 6]), bloco(d,[3, 6])]
Block b is free
Block c is free
Position [4, 6] is free
Action move(c, _, [4,6]) can be done with preconditions
[livre(c),livre_destino([4,6])]
Regressed goals: [livre(c), livre_destino([4, 6]), bloco(a,[4,
5]), bloco(b,[5, 6]), bloco(d,[3, 6])]
Block d is free
Position [3, 6] is free
Action move(d, _, [3,6]) can be done with preconditions
[livre(d),livre_destino([3,6])]
Regressed goals: [livre(d), livre_destino([3, 6]), bloco(a,[4,
5]), bloco(b,[5, 6]), bloco(c,[4, 6])]
```

4. Para sair do estado inicial i2, e chegar no estado final i2(c), o primeiro movimento será tirar o bloco **a** de cima do bloco **c**, move(A, c(0,1), b(5,6)). Depois disso, o bloco **d** pode ir para sua posição final, sobre o bloco **c**, move(D, (2,5), c(0,3)). Por fim, o bloco **a** pode ir para sua posição final, move(A, b(5,6), table(2,3)).

Saída do código:

```
Block c is free
Block a is free
Position [2, 3] is free
Action move(a, _, [2,3]) can be done with preconditions
[livre(a),livre_destino([2,3])]
Regressed goals: [livre(a), livre_destino([2, 3]), bloco(c,[0,
2]), bloco(b,[5, 6]), bloco(d,[0, 3])]
Block b is free
Block d is free
```

```
Position [0, 3] is free
Action move(d, _, [0,3]) can be done with preconditions
[livre(d),livre_destino([0,3])]
Regressed goals: [livre(d), livre_destino([0, 3]), bloco(c,[0,
2]), bloco(a,[2, 3]), bloco(b,[5, 6])]
```

5. Para sair do estado i1 e chegar ao estado i2, o primeiro movimento é mover o bloco **d** para cima do bloco **c,** move(D, (a,b), c(0,3)). Depois, o bloco **A** deve ser movido para cima do bloco **b** para abrir espaço onde o bloco **d** será posicionado, (move(A, (3,4), b(5,6)). Depois, o bloco **d** pode ser movido para a posição final que ele vai ficar, move(D, c(0,3), (2,5)). Por fim, o bloco **a** vai para sua posição final, acima do **c,** move(a, b(5,6), c(0,1)).

   Saída do código:

```
Block a is free
Position [0, 1] is free
Action move(a, _, [0,1]) can be done with preconditions
[livre(a),livre_destino([0,1])]
Regressed goals: [livre(a), livre_destino([0, 1]), bloco(c,[0,
2]), bloco(d,[2, 5]), bloco(b,[5, 6])]
Block c is free
Block d is free
Position [2, 5] is free
Action move(d, _, [2,5]) can be done with preconditions
[livre(d),livre_destino([2,5])]
Regressed goals: [livre(d), livre_destino([2, 5]), bloco(a,[0,
1]), bloco(c,[0, 2]), bloco(b,[5, 6])]
Block b is free
```