Trabalho 1 - Mundo dos Blocos

Josival S.Monteiro júnior[1] **e Vinicius Luiz N. da Fonseca**[1]

[1]Instituto de Computação – Universidade Federal do Amazonas (UFAM)
Caixa Postal 69080-900 – Manaus – AM – Brazil

josival.salvador@icomp.ufam.edu.br, vinicius.fonseca@icomp.ufam.edu.br

1. Visão Geral do Projeto

1.1. Link do Projeto Github

https://github.com/JosivalSalvador/T1_MundoDosBlocos_01

1.2. Descrição Geral

Este projeto implementa, em Prolog, um cenário denominado "Mundo dos Blocos", onde um agente manipula blocos de diferentes tamanhos para organizá-los conforme objetivos predefinidos. A solução utiliza as técnicas descritas no capítulo 17 do livro de Bratko, com um planejador responsável por garantir a execução eficiente das tarefas. O objetivo principal é alcançar uma configuração estável e otimizada dos blocos.

Recentemente, foram feitas modificações no código para melhorar a eficiência e o controle do agente sobre o ambiente. Essas modificações incluem o uso de variáveis não instanciadas para lidar com metas parciais, otimizações para evitar backtracking desnecessário e novas verificações para garantir a preservação das metas durante a execução das ações.

1.3. Tabela de Componentes do Agente Robótico

A tabela que descreve os principais componentes do agente robótico permanece inalterada, mas agora o planejador lida com metas parcialmente definidas, aumentando a flexibilidade na organização dos blocos.

Tipo de Agente	Métrica de Desempenho	Ambiente	Atuadores	Sensores
Robô Organizador Autônomo	Número de movimentos e estabilidade do posicionamento	Mundo dos Blocos	Braços Articulados e Garra	Câmera, Sensor de Proximidade

Tabela 1. Componentes do agente robótico autônomo e seus elementos de interação com o ambiente.

2. Estados, Ações e Alterações no Código

2.1. Estado Inicial e Estado Final

O estado inicial e o estado final do mundo dos blocos continuam sendo descritos pelos predicados `em(Bloco, Posicao)` e `estado_final([em(a, 3), em(b, 1), em(c, 2)])`. No entanto, o código agora foi modificado para introduzir o uso de variáveis não instanciadas. Isso permite que o agente seja mais flexível, podendo lidar com metas que não especificam exatamente todas as posições finais dos blocos.

2.2. Ações de Movimento

As ações de movimento continuam a ser implementadas por meio do predicado `move(Bloco, De, Para)`. O código foi atualizado para verificar condições adicionais, como a possibilidade de mover um bloco com base em suas pré-condições e efeitos, utilizando os predicados `adds/2` e `deletes/2` para gerenciar os efeitos das ações. Essas mudanças visam garantir que o planejamento considere o impacto total das ações no estado do ambiente, mantendo a integridade do sistema.

Exemplo atualizado de código:

```
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),
    write('Moved block '), write(Bloco), write(' from '),
       write(De), write(' to '), write(Para), nl.
```

O novo predicado `move/5` lida com o estado atual do ambiente e atualiza as posições dos blocos de maneira eficiente, levando em consideração a liberdade das posições de destino e o estado atual dos blocos.

2.3. Planejamento Heurístico

O planejador foi modificado para incluir uma heurística mais refinada que prioriza blocos maiores durante o movimento, conforme o seguinte predicado:

```
heuristica(Estado, Score) :-
    findall(Tamanho, (member(bloco(_, Tamanho), Estado),
       Tamanho > 1), Tamanhos),
    sumlist(Tamanhos, Score),
    write('Heuristica calculada: '), write(Score), nl.
```

Essa heurística calcula uma pontuação com base no tamanho dos blocos e prioriza aqueles que exigem mais movimentação ou que são mais críticos para alcançar o estado final.

2.4. Ações de Percepção

Uma nova funcionalidade de percepção foi introduzida no código, permitindo ao agente ”olhar”para uma posição específica no ambiente e identificar o bloco presente. O predicado `look/3` simula a percepção do agente, oferecendo uma visão mais detalhada do ambiente:

```
look(Posicao, Objeto, Estado) :-
    member(bloco(Objeto, Posicao), Estado),
    write('Looking at position '), write(Posicao), write('
       and finding object '), write(Objeto), nl.
```

Essa ação de percepção é crucial para casos em que o agente não tem conhecimento completo do ambiente, permitindo que ele atualize suas informações ao interagir com os blocos.

2.5. Questão 5: Modificações no Planejador

Com base na Figura 17.6 do livro de Bratko, o planejador foi modificado para manipular variáveis não instanciadas, o que permite lidar com metas parcialmente definidas e aumentar a flexibilidade na execução do plano.

O predicado `regress/3` foi adaptado para regredir as metas com base nos efeitos das ações. A ideia central é que, após realizar uma ação, o agente possa reavaliar quais metas ainda precisam ser atingidas. Além disso, o predicado `achieves/2` foi adicionado para verificar se uma ação específica alcança uma meta. Isso ajuda a determinar quais ações são mais adequadas para o momento, conforme mostrado no exemplo abaixo:

```
achieves(Action, Goal) :-
    adds(Action, Effects),
    member(Goal, Effects),
    !,
    write('Action '), write(Action), write(' achieves goal
       '), write(Goal), nl.
```

Além disso, o predicado `preserves/2` foi adicionado para garantir que as metas atuais não sejam violadas por uma nova ação, preservando as metas já alcançadas:

```
preserves(Action, Goals) :-
    deletes(Action, DeletedEffects),
    \+ (member(Goal, DeletedEffects),
        member(Goal, Goals)),
    !,
    write('Action '), write(Action), write(' preserves all
       goals'), nl.
```

Essas modificações aumentam a eficiência do planejador, evitando o retrocesso desnecessário (backtracking) e otimizando a busca por soluções.

3. Planejamentos

Na figura 1 está representada as situções nas quais foram geradas os planejamentos.

3.1. s_inicial =i1 ate o estado s_final=i2

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

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

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

3.5. (i1) para o estado (i2)

4. Conclusão

As modificações realizadas no código original da Figura 17.6 de Bratko tiveram como principal objetivo aumentar a eficiência do planejamento e garantir que o agente manipulador de blocos pudesse lidar com metas parcialmente definidas. Além disso, a inclusão de ações de percepção, juntamente com o controle das pré-condições e dos efeitos das

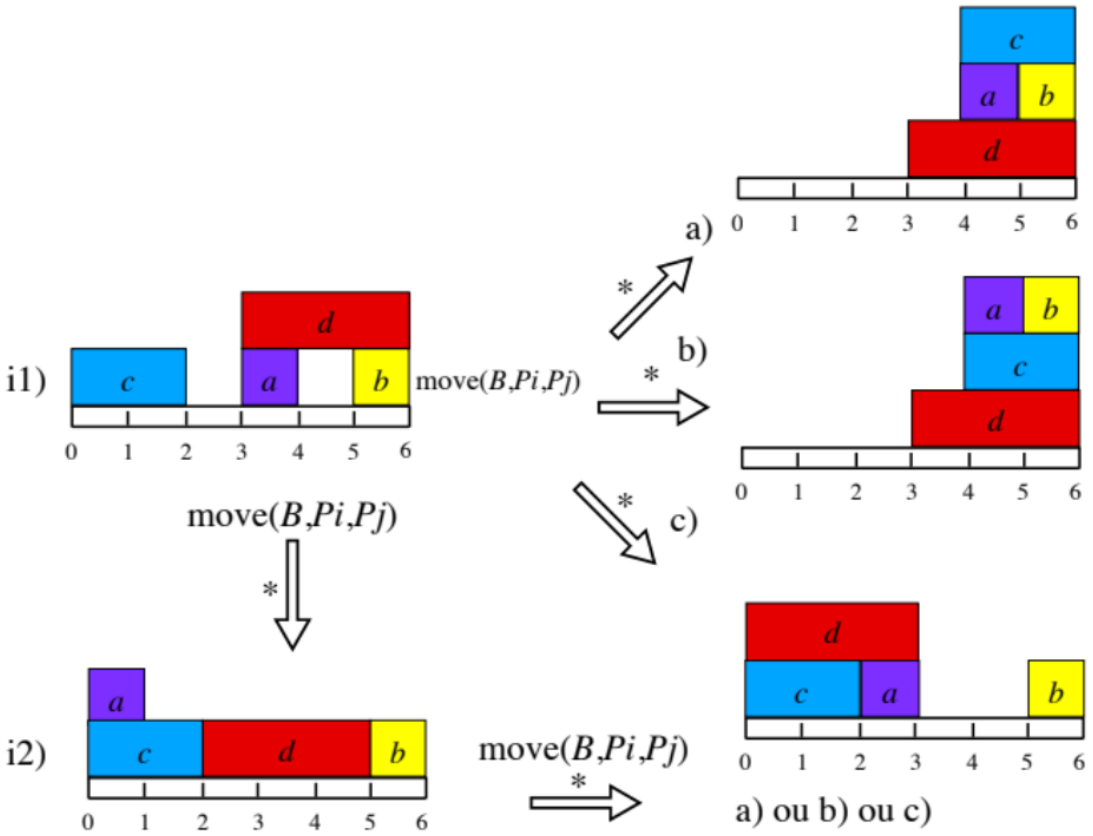

Figura 1. Situações para planejar

ações, permitiu que o planejador fosse capaz de trabalhar com mais flexibilidade e precisão, evitando problemas de backtracking desnecessário e garantindo a preservação de metas já alcançadas.

O uso de variáveis não instanciadas possibilitou que o sistema pudesse operar de maneira mais geral, sem depender de uma definição rígida das metas. Isso permitiu maior adaptabilidade ao ambiente, oferecendo soluções mais eficientes para a organização dos blocos. A heurística introduzida, que prioriza blocos maiores, também foi uma adição importante para otimizar o processo de planejamento.

Finalmente, o código foi testado em várias situações de planejamento e conseguiu alcançar o estado final desejado de maneira eficiente, como demonstrado na Figura 1. O projeto continuará a ser aprimorado com a inclusão de mais ações de percepção e um refinamento das heurísticas.