Universidade Federal do Amazonas
Hannah Lisboa Barreto - 22053199
Matheus Rocha Canto - 22250353
Paulo Vitor de Castro Freitas - 21855092

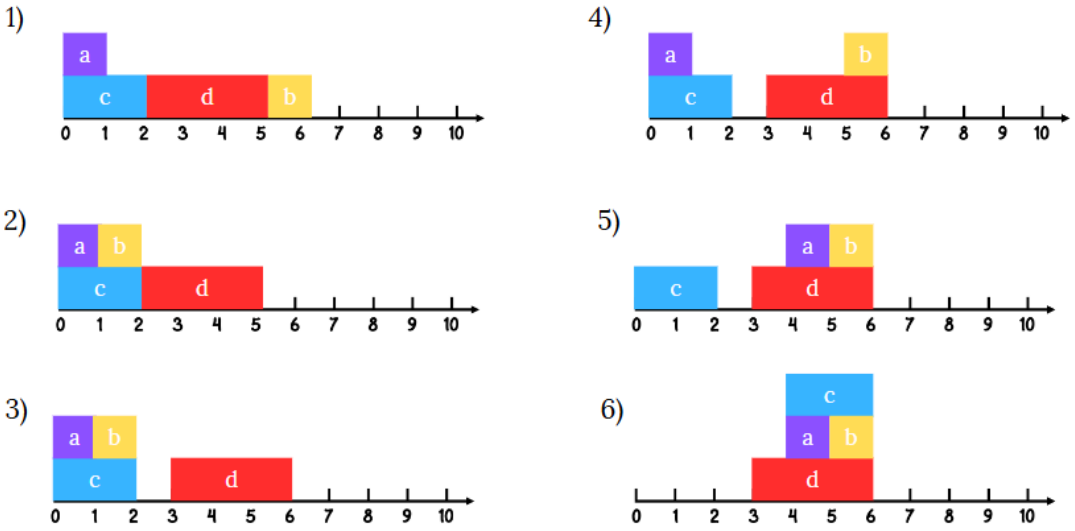
Inteligência Artificial
Planejador para Empilhar

Manaus-AM
2024

1. Resposta explicativa para cada item do PDF

1.1. Situação 1

1.1.1. s_inicial=i2 até o estado s_final=i2 (a).



A partir da análise da Situação 1, que vai do ponto inicial i2 até o destino i2(a), percebemos que, com base nas regras estipuladas e aplicadas pelo grupo, há um caminho possível de ações que levam à meta final. Assim, com base na representação dos estados inicial (1) e final (6), temos o seguinte plano, mostrado na Figura 1:

Manualmente (linguagem natural)

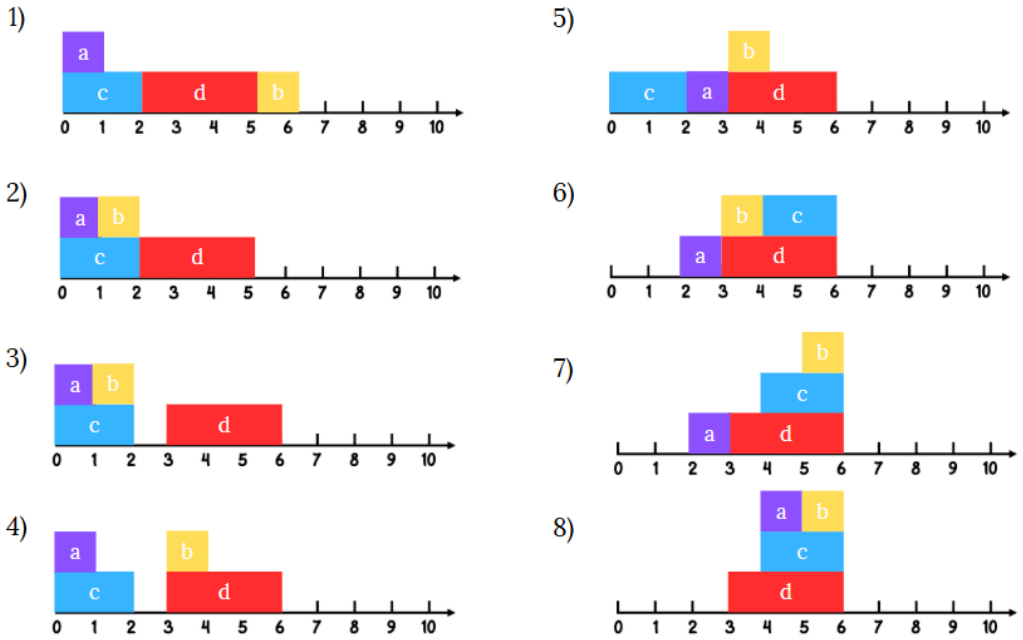
1. Começo na configuração inicial;
2. O bloco "b" é transferido da posição 5 (mesa) para cima do bloco "c";
3. O bloco "d" é realocado das posições 2, 3 e 4 para as posições 3, 4 e 5;
4. "b" é reposicionado de cima de "c" para sobre "d";
5. "a" é movido de cima de "c" para cima de "d";
6. "c" é colocado sobre "a" e "b", os quais estão sobre "d".

Prolog

```
move(b, pos(5), on(c)).
move(d, [pos(2), pos(3), pos(4)], [pos(3), pos(4), pos(5)]).
move(b, on(c), on(d)).
```

move(a, on(c), on(d)).
move(c, [pos(0), pos(1)], [on(a), on(b)]).

1.1.2. s_inicial=i2 até o estado s_final=i2 (b).



Com as mesmas restrições, também é possível chegar do ponto i2 ao objetivo i2(b). A sequência de ações é:

Manualmente (linguagem natural)

1. Estado inicial;
2. "b" vai da posição 5 para cima de "c";
3. "d" é reposicionado nas posições 3, 4 e 5;
4. "b" é colocado sobre "d";
5. "a" é movido para a posição 2 (mesa);
6. "c" vai para cima de "d";
7. "b" vai para cima de "c";
8. "a" é movido da mesa para cima de "c".

Prolog

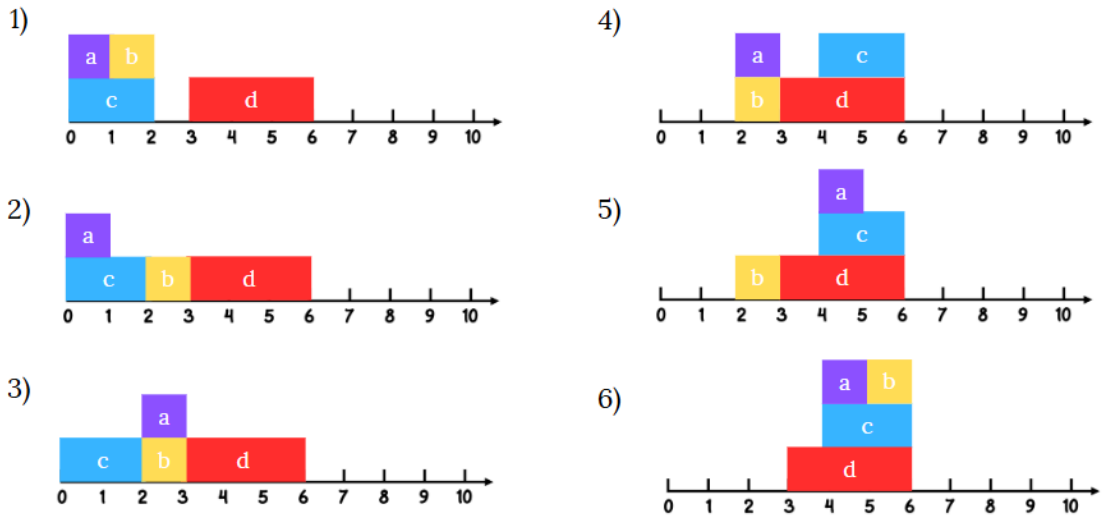
move(b, pos(5), on(c)).
move(d, [pos(2), pos(3), pos(4)], [pos(3), pos(4), pos(5)]).

move(b, on(c), on(d)).
move(a, on(c), pos(2)).
move(c, [pos(0), pos(1)], [on(d), on(d)]).
move(b, on(d), on(c)).
move(a, pos(2), on(c)).

1.1.3. s_inicial=i2 até o estado s_final=i2 (c).

1.1.4. (i1) para o estado (i2)

1.2. Situação 2



Neste caso, a sequência de ações já está determinada. Verificamos que as etapas respeitam as limitações aplicadas pelo planejador, permitindo sua execução com sucesso:

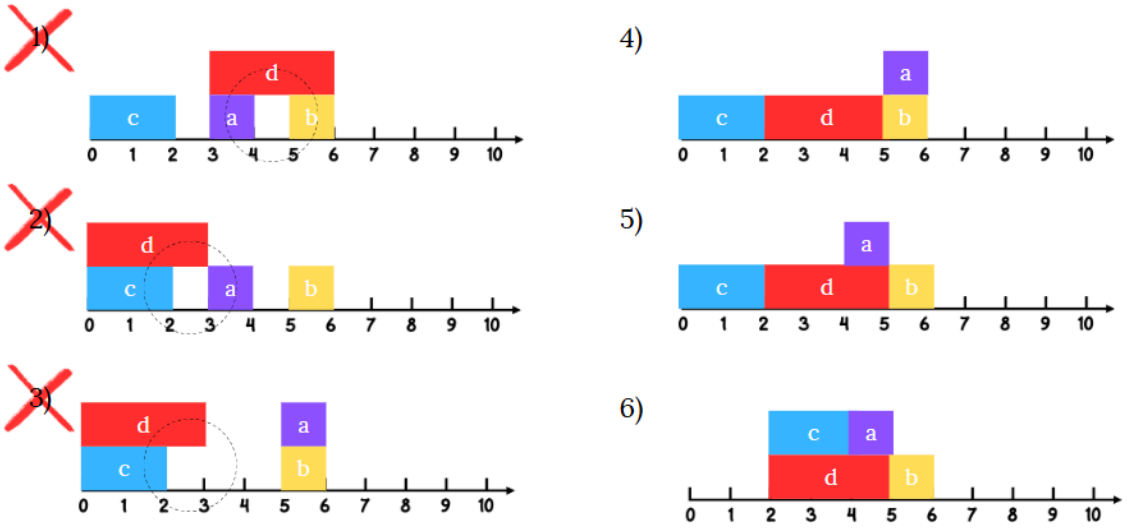
Manualmente (linguagem natural)

1. Posição inicial;
2. "b" é retirado de cima de "c" e colocado na mesa;
3. "a" é colocado sobre "b";
4. "c" vai para cima de "d";
5. "a" é reposicionado sobre "c";
6. "b" também vai para cima de "c".

Prolog

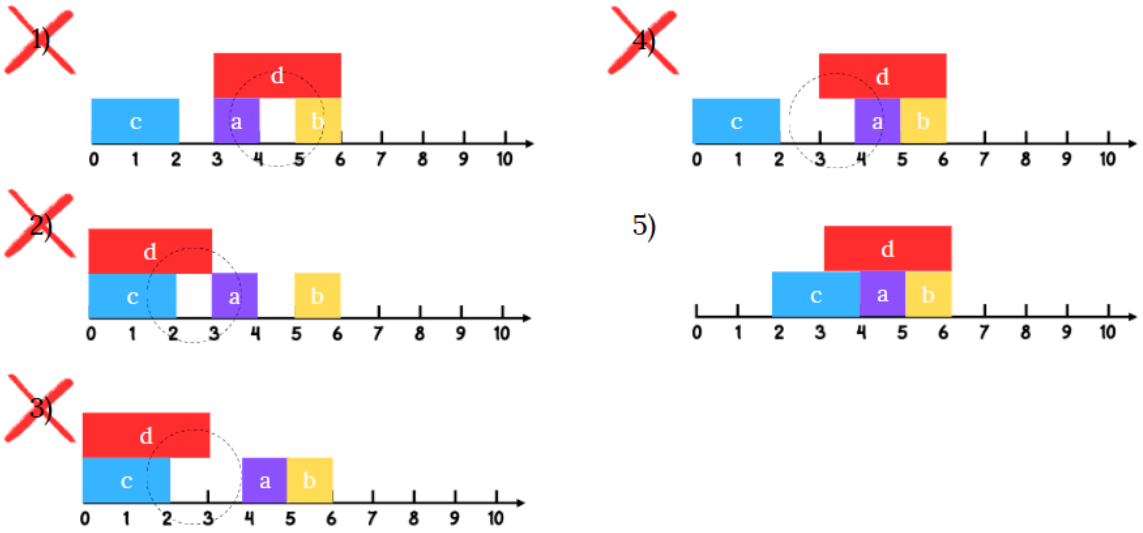
```
move(b, on(c), pos(2)).
move(a, on(c), on(b)).
move(c, [pos(0), pos(1)], [on(d), on(d)]).
move(a, on(b), on(c)).
move(b, pos(2), on(c)).
```

1.3. Situação 3



Nesta situação, a execução do plano falha, pois os três primeiros passos desrespeitam as regras de estabilidade e dimensões definidas. Assim, o planejador não pode dar continuidade correta ao plano.

1.4. Situação 4



Assim como na situação anterior, a sequência não é válida, pois os primeiros quatro passos infringem as condições de empilhamento estabelecidas. A execução não pode prosseguir de forma adequada.

2. Interações com o Chatbot

2.1. Representação do Conhecimento

2.1.1. Descrição do problema em linguagem natural

O problema envolve blocos de diferentes comprimentos, mas todos têm a mesma altura. Eles podem ser empilhados uns sobre os outros ou colocados diretamente em prateleiras. O objetivo é movimentar os blocos de um estado inicial para um estado final, respeitando restrições de empilhamento.

2.1.2. Definição de Conceitos

- **on(X, Y)** → Bloco X está em cima do bloco Y;
- **ontable(X)** → Bloco X está diretamente sobre a mesa;
- **clear(X)** → Nada está sobre o bloco X;
- **holding(X**) → Está segurando o bloco X (caso deseje simular braço robótico);
- **handempty** → A mão está vazia (se usar esse modelo).

2.1.3. Representação de Termos

Pedi ao ChatGPT que sugerisse uma representação de conhecimento para o mundo dos blocos, sem usar assert ou retract. A resposta foi a seguinte:

```
% Representação de um estado do mundo:
% Estado = [on(a,b), on(b,c), ontable(c), clear(a)]

% Significados:
% - on(a, b): a está sobre b
% - ontable(c): c está diretamente na mesa
% - clear(a): nada está sobre a

% Isso representa a pilha: a em cima de b, b em cima de c, e c sobre a mesa
```

2.2. Parte Raciocínio e Planejamento Automático

2.1.1. Definição de Restrições

Restrições para o Mundo dos Blocos com Dimensões Diferentes

Baseado no capítulo 17 do livro "Prolog Programming for Artificial Intelligence", vou definir as restrições necessárias para o problema dos blocos com dimensões diferentes, utilizando a abordagem STRIPS-like apresentada no documento.

Restrições em Linguagem Natural

Restrição de Movimento Básico:

- Um bloco só pode ser movido se estiver livre (nada em cima dele).
- A posição de destino deve estar livre (se for outro bloco) ou ser uma posição válida na mesa.
- O bloco deve estar atualmente em uma posição conhecida.

Restrição de Tamanho:

- Um bloco só pode ser colocado sobre outro bloco se o bloco de baixo tiver largura suficiente para suportá-lo.
- Um bloco não pode ser colocado parcialmente fora de outro bloco ou da mesa.

Restrição de Estabilidade:

- Todo bloco deve estar completamente apoiado (não pode ter partes suspensas).
- Blocos não podem flutuar no ar - devem estar sobre a mesa ou sobre outros blocos.

Restrições de Estado Inválido:

- Um bloco não pode estar sobre si mesmo.
- Dois blocos não podem ocupar a mesma posição simultaneamente.
- Um bloco não pode estar em dois lugares ao mesmo tempo.

Restrições de Ação:

Não pode mover um bloco para a mesma posição onde já está.
Não pode mover um bloco para uma posição ocupada por outro bloco, a menos que esteja livre.
A trajetória do movimento deve ser livre (para cima, depois horizontal, depois para baixo).

2.1.2. Implementação em Prolog

3. Código comentado em Prolog

```
can( move( Block, From, To), [clear( Block), clear( To), on( Block, From)]) :-
        block( Block),
        object( To),
        To \== Block,
        object( From),
        From \== To,
        Block \== From.

adds( move(X, From, To), [on(X, To), clear(From)]).

deletes( move(X, From, To), [on(X, From), clear(To)]).

object( X) :-
        place(X)
        ;
        block(X).

block(a).
block(b).
block(c).
block(d).

place(1).
place(2).
place(3).
place(4).
place(5).
place(6).

impossible( on(X, X), _).

impossible( on(X, Y), Goals) :-
        member( clear(Y), Goals)
        ;
        member( on( X, Y1), Goals), Y1 \== Y
        ;
        member( on( X1, Y), Goals), X1 \== X.

impossible( clear(X), Goals) :-
        member(on(_, X), Goals).

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).

satisfied( State, Goals) :-
        delete_all( Goals, State, []).

select( State, Goals, Goal) :-
        member( Goal, Goals).

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

preserves( Action, Goals) :-
        deletes(Action, Relations),
        \+ (member( Goal, Relations),
        member(Goal, Goals)).

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

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([], _, []).
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).

initial_state([
        on(a, 4),
        on(b, 6),
        on(d, a),
        on(c, 1),
        clear(3),
        clear(5),
        clear(2),
        clear(b),
        clear(c),
  clear(d)
]).

goal_state([
        on(a, c)
]).
```

4. Instruções de como executar

Este guia explica como executar o código do mundo dos blocos com planejamento baseado em goal regression e means-ends analysis usando o interpretador **SWI-Prolog**.

Requisitos:

- Ter o **SWI-Prolog** instalado em sua máquina. Se ainda não tiver:
    - Acesse: https://www.swi-prolog.org/Download.html
    - Siga as instruções de instalação para o seu sistema operacional.

Arquivo Necessário

Salve o seguinte arquivo em sua máquina:

Nome do arquivo: blocks_world_with_planner.pl

Este arquivo contém:

- Definição do domínio do mundo dos blocos;
- Implementação do planejador com regressão de metas;
- Estado inicial e objetivo;
- Predicado run/0 para executar o plano.

Etapas para Executar

1. **Abra o SWI-Prolog.**

2. **Carregue o arquivo.**
   No prompt do Prolog, navegue até a pasta onde está o arquivo e digite:
   ?- [blocks_world_with_planner].
   ou, se preferir o caminho completo:
   ?- consult('Caminho/para/blocks_world_with_planner.pl').
3. **Execute o planejador.**
   Após o carregamento bem-sucedido, execute:
   ?- run.

Veja o resultado.
O Prolog irá retornar a lista de ações planejadas que transformam o estado inicial no estado final (objetivo), por exemplo:
Plano encontrado:

4. [move(a, p([4]), c), move(d, p([a,b]), p([4,6]))]
   (O plano é apenas ilustrativo; o resultado real depende das regras e do estado definidos.)

Dicas adicionais

- Se quiser testar com outro estado inicial ou outro objetivo, edite os predicados initial(...) e goal(...) no final do arquivo.
- Para debugar ou entender melhor os passos, você pode modificar o código para imprimir os estados intermediários.
- Use o comando trace. antes de run. para acompanhar a execução passo a passo.

Conclusão

Este sistema demonstra como utilizar técnicas clássicas de IA, como regressão de metas e análise de meios-fins, para resolver problemas de planejamento no domínio do mundo dos blocos, conforme apresentado no livro de Ivan Bratko.