1. Proposta de Linguagem de Representação

Para modelar este mundo dos blocos com dimensões variadas, proponho uma representação em Prolog que estende o modelo clássico, capturando as propriedades físicas e espaciais necessárias. A principal dica é usar uma grade onde cada espaço tem o tamanho do menor bloco.

a) Representação de Estado: Um estado do mundo será representado por uma lista de predicados que descrevem a posição e o estado de cada bloco.

- **Propriedades Estáticas (definidas como fatos no banco de dados):**
  - `tamanho(Bloco, Largura)`: Associa a cada bloco uma largura em unidades da grade.
    - *Exemplo:* `tamanho(a, 1).`, `tamanho(c, 2).`, `tamanho(d, 3)`.
  - `bloco(Nome)`: Define quais objetos são blocos manipuláveis.
    - *Exemplo:* `bloco(a).`, `bloco(b).`, `bloco(c).`, `bloco(d).`
- **Propriedades Dinâmicas (parte da lista de estado):**
  - `pos(Bloco, Suporte)`: Descreve a posição de um `Bloco`. O `Suporte` pode ser a mesa ou outro bloco.
    - `pos(Bloco, mesa(X))`: O `Bloco` está na mesa, com sua borda esquerda na coordenada `X`.
    - `pos(Bloco, sobre(OutroBloco))`: O `Bloco` está diretamente sobre `OutroBloco`.
  - `livre(Bloco)`: Indica que a superfície superior de um `Bloco` está desocupada, permitindo que ele seja movido ou que outro bloco seja colocado sobre ele.

b) Justificativa e Comparação com o Modelo Clássico (Figura 17.1 do Cap. 17):

- **Tamanho do Bloco:** O modelo clássico assume que todos os blocos são uniformes. A introdução do predicado `tamanho/2` é a principal extensão, sendo crucial para verificar as condições de estabilidade.
- **Posição na Mesa:** No modelo clássico, `on(a, 1)` trata o "lugar 1" como um identificador abstrato. A minha proposta `pos(a, mesa(0))` é mais rica, pois `0` é uma coordenada específica em uma grade espacial. Isso permite raciocinar sobre o espaço lateral ocupado por um bloco. Por exemplo, se `c` tem `tamanho(c, 2)` e está em `pos(c, mesa(0))`, sabemos que ele ocupa os espaços 0 e 1.
- **Condição de Vacância:** O predicado `clear(Objeto)` do modelo clássico é análogo ao meu `livre(Bloco)`. No entanto, a minha proposta lida com a vacância horizontal de forma diferente. Em vez de um predicado de estado, a verificação de espaço livre na mesa é feita dinamicamente pelo planejador, que calcula os espaços

necessários (`Largura` do bloco) e verifica se eles não estão ocupados por nenhum outro bloco. Isso evita redundância na representação do estado.
- **Estabilidade:** Esta é uma condição nova, inexistente no modelo clássico. Ela é implementada como uma pré-condição na ação de movimento: ao tentar mover `B1` para cima de `B2`, o planejador deve verificar se `tamanho(B1, L1), tamanho(B2, L2), L1 =< L2`. Isso impede a criação de pilhas fisicamente instáveis.

2. Modificação do Código do Planejador

O planejador da Figura 17.6 do livro do Bratko precisa ser modificado para lidar com variáveis não instanciadas em objetivos e ações, conforme a discussão na seção 17.5 do livro.

A principal mudança é na forma como o planejador lida com a ação de mover um bloco para a mesa. Em vez de gerar ações concretas como `move(a, mesa(0))`, `move(a, mesa(1))`, etc., e testá-las uma a uma, o planejador deve usar uma ação com uma variável: `move(a, mesa(X))`.

Explicação da Mudança:

1. **Ação com Variável:** O planejador, ao tentar satisfazer um objetivo como "colocar o bloco `c` na mesa", selecionaria a ação `move(c, mesa(X))`, onde `X` é uma variável livre.
2. **Adiamento da Instanciação (Princípio do Menor Compromisso):** Em vez de "adivinhar" um valor para `X`, o planejador adiciona as pré-condições para a ação à sua lista de sub-objetivos. Uma dessas pré-condições seria a verificação de espaço: `espaco_livre_na_mesa(X, Largura_de_c)`.
3. **Resolução por Restrições:** O planejador só instanciará a variável `X` quando for estritamente necessário ou quando outras partes do plano restringirem seu valor. Ele buscará no estado atual um `X` que satisfaça a condição de que os `Largura_de_c` espaços a partir de `X` estão livres.

Essa modificação torna o planejador mais eficiente, pois ele não perde tempo explorando caminhos com posicionamentos inválidos na mesa. Ele raciocina sobre a *restrição* (precisa de um espaço livre) em vez de testar todas as *instâncias* possíveis.

3. Geração Manual de Plano para a Situação 1

Usando a linguagem proposta, vamos gerar os planos para a

Situação 1.

- **Blocos e Tamanhos:** `tamanho(a, 1)`, `tamanho(b, 1)`, `tamanho(c, 2)`, `tamanho(d, 3)`.
- **Estado Inicial (i1):** `[pos(d, mesa(3)), pos(c, sobre(d)), pos(a, sobre(c)), pos(b, pos(a, 5)), livre(a), livre(b)]`.

- **Estado Final (i2 - figura b):** `[pos(c, mesa(0)), pos(d, mesa(2)), pos(b, mesa(5)), pos(a, sobre(c)), livre(d), livre(b), livre(a)]`.

Plano de `s_inicial = i1` para `s_final = i2` (figura b): O objetivo é mover `c` e `d` para a mesa e colocar `a` sobre `c`. Para mover `c`, primeiro precisamos mover `a`.

1. **`move(a, mesa(0))`**: Movemos o bloco `a` para um local temporário para liberar `c`.
    - *Pré-condições:* `livre(a)` (verdadeiro), `espaco_livre_na_mesa(0, 1)` (verdadeiro).
2. **`move(c, mesa(0))`**: Agora que `c` está livre, movemos para sua posição final.
    - *Pré-condições:* `livre(c)` (agora é verdadeiro), `espaco_livre_na_mesa(0, 2)` (agora é verdadeiro, pois `a` foi movido).
3. **`move(d, mesa(2))`**: Movemos `d` para sua posição final.
    - *Pré-condições:* `livre(d)` (verdadeiro), `espaco_livre_na_mesa(2, 3)` (verdadeiro, pois os espaços 2, 3 e 4 estão livres).
4. **`move(a, sobre(c))`**: Movemos `a` de sua posição temporária para sua posição final sobre `c`.
    - *Pré-condições:* `livre(a)` (verdadeiro), `livre(c)` (agora é verdadeiro), `tamanho(a,1) =< tamanho(c,2)` (verdadeiro).

Plano Final (sequência de ações): `[move(a, mesa(0)), move(c, mesa(0)), move(d, mesa(2)), move(a, sobre(c))]`

(Obs: Os outros planos solicitados na questão 3 seguiriam um raciocínio análogo de decomposição de objetivos e verificação de pré-condições).

4. Análise para as Situações 2 e 3

Para que o planejador gere planos para as **Situações 2 e 3**, é necessário que a implementação dos operadores e da busca seja robusta o suficiente para lidar com múltiplos níveis de sub-objetivos e interdependências.

- **Linguagem de Representação:** A linguagem proposta já é suficiente. Ela captura todas as informações necessárias (tamanho, posição, suporte) para validar qualquer estado nessas situações.
- **Padrões e Desafios:**
    - **Desmontagem de Pilhas:** Em ambas as situações, o planejador precisa primeiro "desmontar" as pilhas existentes para acessar blocos que estão na base (como na Situação 3, para mover o bloco `c`, é preciso primeiro mover `d`). O planejador fará isso naturalmente através da recursão de objetivos: para satisfazer `move(c, ...)`, ele gerará o sub-objetivo `livre(c)`, o que por sua vez exigirá a ação `move(d, ...)`.
    - **Gerenciamento de Espaço Temporário:** O desafio mais complexo é o gerenciamento do espaço na mesa. Muitas vezes, um bloco precisa ser movido para um local temporário para liberar outro bloco, antes de ser movido para sua posição final.

- **Eficiência da Busca:** A modificação discutida no item 2 (uso de variáveis não instanciadas) é **crucial** aqui. Sem ela, o planejador gastaria um tempo enorme testando todos os possíveis locais temporários para cada bloco que precisa ser movido. Com a abordagem de restrições, ele pode simplesmente postular a necessidade de "um lugar livre com largura suficiente" e prosseguir com o planejamento, tornando a busca muito mais eficiente.

Em resumo, para resolver as Situações 2 e 3, o planejador precisa de uma implementação completa do mecanismo de regressão de objetivos que possa:

1. Gerar sub-objetivos recursivamente (ex: para mover `c`, precisa tornar `c` livre).
2. Manipular ações com variáveis (`move(B, mesa(X))`) para encontrar eficientemente espaço vago na mesa.
3. Verificar rigorosamente todas as pré-condições a cada passo, incluindo as de estabilidade (`tamanho(B1) =< tamanho(B2)`) e de espaço.