Estendendo a estrutura de planejamento STRIPS para um mundo de blocos com tamanhos heterogêneos e restrições espaciais: uma abordagem de programação lógica

Prof. Edjard Mota

Seção 1: Desconstrução do Modelo de Mundo dos Blocos Clássicos

O problema clássico do mundo dos blocos serve como um exemplo canônico no estudo do planejamento automatizado. Sua simplicidade permite uma ilustração clara dos princípios básicos do planejamento, como busca em espaço de estados, análise de meios-fins e regressão de objetivos. O modelo apresentado no Capítulo 17 de *Prolog Programming for Artificial Intelligence* fornece uma base sólida e bem compreendida com base na representação STRIPS (Stanford Research Institute Problem Solver). [1] No entanto, para abordar problemas de planejamento que incorporam restrições físicas mais realistas, como as descritas nos cenários de blocos fornecidos [1] , é necessário primeiro desconstruir esse modelo clássico para entender suas suposições e limitações inerentes.

1.1 A Representação STRIPS: Uma Fundação em Relações Abstratas

O poder do modelo clássico advém de seu alto nível de abstração. O estado do mundo não é representado como uma simulação geométrica ou física, mas como um conjunto finito de relações lógicas que são consideradas verdadeiras em um dado momento. [1] Para o mundo dos blocos, esse estado é tipicamente capturado por uma lista de fórmulas atômicas básicas envolvendo dois predicados primários:

- on(Block, Object): Este predicado afirma que um Bloco específico está imediatamente sobre outro Objeto. O Objeto pode ser outro bloco ou um "lugar" designado na tabela.
- clear(Object): Este predicado afirma que a superfície superior de um Objeto está desobstruída, o que significa que não há nenhum outro bloco sobre ela. Este é um pré-requisito crucial para mover um bloco ou colocar outro bloco sobre ele.

Por exemplo, o estado inicial representado na Figura 17.1 do texto de referência é formalmente representado pela lista: [clear(2), clear(4), clear(b), clear(c), on(a,1), on(b,3), on(c,a)]. [1] Esta representação é puramente relacional. Os "lugares" (1, 2, 3, 4) são meramente identificadores únicos e abstratos; eles não possuem propriedades espaciais como adjacência, distância ou tamanho. O predicado

1

1.2 O Operador move: Um Esquema para Transição de Estado

As transições de estado no formalismo STRIPS são definidas por um conjunto de operadores, ou ações. Para o mundo dos blocos, a única ação normalmente é mover(Bloco, De, Para). A definição deste operador é tripartite, consistindo em pré-condições, uma lista de adição e uma lista de exclusão. [1]

1. **Pré-condições:** Estas são as condições que devem ser verdadeiras no estado atual para que a ação seja executável. O predicado can(Action, Condition) especifica isso. Para move(Block, From, To), as pré-condições padrão são que o Bloco a ser movido deve estar limpo, seu destino Para deve estar limpo e o Bloco deve estar atualmente em De. Isso é capturado na cláusula Prolog: can(move(Block, From, To),). [1]
2. **Add-List:** Esta é uma lista de novos relacionamentos que se tornam verdadeiros após a execução da ação. O predicado adds(Action, AddRels) define isso. A ação move torna verdadeiro que o Bloco agora está em Para e que sua localização original De agora está livre. Isso é representado como: adds(move(X,From,To),). [1]
3. **Delete-List:** Esta é uma lista de relacionamentos que eram verdadeiros antes da ação, mas que deixaram de ser verdadeiros depois. O predicado deletes(Action, DelRels) define isso. A ação move falsifica o antigo relacionamento on e o status clear do destino. Isso é representado como: deletes(move(X,From,To),). [1]

Este esquema de operador encapsula elegantemente a lógica da translocação. O planejador, seja utilizando busca em espaço de estados com encadeamento progressivo ou regressão de objetivos com encadeamento reverso, utiliza essas definições para construir sequências válidas de ações. A simetria no tratamento de From e To como objetos abstratos é uma característica fundamental; a lógica não diferencia entre mover um bloco para outro bloco ou movê-lo para um lugar na tabela.

1.3 Limitações críticas e a necessidade de extensão

Embora o modelo clássico seja uma ferramenta poderosa para planejamento abstrato, suas premissas fundamentais falham quando confrontadas com os requisitos implícitos nos cenários de Block_Scenarios.pdf. [1] As limitações não são deficiências menores, mas incompatibilidades fundamentais no esquema de representação do conhecimento.

A inadequação de on/2 para raciocínio espacial

Os cenários apresentados utilizam uma grade numerada, normalmente de 0 a 6, para representar localizações discretas em uma tabela. [1] Isso introduz um sistema de coordenadas rudimentar. Nesse contexto, a posição de um bloco não é apenas relacional, mas absoluta. O predicado clássico

on(a, 1) é insuficiente porque não consegue distinguir entre um bloco que está na coordenada 1 e na coordenada 5; ele só sabe que o bloco está no local abstratamente chamado '1'.

Além disso, os diagramas mostram claramente blocos de larguras variadas. Um bloco largo, como 'c', pode ocupar vários espaços de coordenadas simultaneamente (por exemplo, os espaços 2 e 3). O predicado on/2, que liga um objeto a outro, não tem capacidade para representar esse conceito de uma pegada física abrangendo vários locais. Ele não consegue expressar que o bloco 'c' está "sobre" a tabela em um local que abrange uma gama de coordenadas. Essa incompatibilidade entre o modelo de localização abstrato e de ponto único de on/2 e a realidade espacial multiponto do novo problema exige uma revisão completa da forma como as posições dos blocos são representadas.

A Ausência de Propriedades Físicas

O modelo clássico trata todos os blocos como entidades uniformes do ponto de vista físico. Os fatos bloco(a), bloco(b) e bloco(c) declaram sua existência como objetos manipuláveis, mas não fornecem nenhuma informação sobre suas propriedades individuais. [1] Essa suposição de homogeneidade é violada pelos cenários, que representam blocos com larguras distintas. [1]

Essa omissão tem consequências profundas para a validade do plano. O predicado clássico can/2 permitiria um plano que colocasse um bloco largo sobre um estreito, uma configuração

fisicamente instável e implicitamente ilegal em um mundo mais realista. O modelo carece de qualquer mecanismo para armazenar propriedades intrínsecas do objeto, como tamanho, e, portanto, os operadores de planejamento não têm a capacidade de raciocinar sobre restrições físicas, como estabilidade. Para resolver o problema do usuário, a base de conhecimento deve ser estendida para incluir não apenas o estado dinâmico do mundo, mas também os atributos físicos estáticos e imutáveis dos objetos dentro dele.

Em resumo, o framework STRIPS clássico fornece o maquinário lógico essencial para o planejamento — análise de meios e fins, definições de ações e atualizações de estado. No entanto, sua instanciação específica para o mundo dos blocos, com sua representação abstrata, relacional e sem propriedades, é fundamentalmente insuficiente para o problema em questão, com restrições espaciais e físicas. As seções subsequentes deste relatório detalharão o processo sistemático de ampliação e substituição dessas estruturas clássicas por um novo formalismo capaz de raciocinar sobre espaço, tamanho e estabilidade.

Seção 2: Um novo esquema de representação do conhecimento

Para superar as limitações do modelo clássico, é necessário um novo esquema de representação do conhecimento. Esse novo esquema deve ir além das relações abstratas e alcançar uma representação mais fundamentada, quase física, capaz de capturar as características essenciais do domínio do problema: as propriedades intrínsecas dos blocos, a natureza espacial do ambiente e a localização precisa dos blocos nesse ambiente. Esta seção detalha a arquitetura dessas novas estruturas de dados.

2.1 Representando Atributos de Bloco Estático: Introdução ao Tamanho

O primeiro passo para construir um modelo fisicamente mais realista é representar as propriedades intrínsecas e imutáveis dos objetos. Os cenários em Block_Scenarios.pdf representam consistentemente blocos com diferentes dimensões horizontais. [1] Essa propriedade, que pode ser chamada de "tamanho" ou "largura", é fundamental para as novas regras de empilhamento.

Uma escolha de design limpa e eficiente é separar essas informações estáticas do estado dinâmico do mundo. O tamanho de um bloco não muda durante a execução de um plano, portanto, ele deve ser definido como um conjunto de fatos imutáveis na base de conhecimento

do Prolog. Isso evita que informações redundantes sejam carregadas na lista de estados a cada etapa do plano. A representação assume a forma de um predicado size(Block, Width), onde Width é um inteiro que representa o número de slots de tabela que o bloco ocupa. Com base na evidência visual, um conjunto plausível de fatos seria:

```
Prólogo
```

```
% tamanho(Bloco, LarguraEmLotes)
tamanho(a, 1 ).
tamanho(b, 1 ).
tamanho(c, 2 ).
tamanho(d, 2 ).
```

Esta simples adição representa um afastamento radical da premissa de uniformidade do modelo clássico. Ela fornece os dados fundamentais sobre os quais todo o raciocínio subsequente sobre estabilidade e ocupação espacial será construído. O planejador agora pode consultar size(c, W) para descobrir que o bloco "c" requer duas unidades de espaço, uma informação essencial para determinar posicionamentos válidos.

2.2 Modelando o Ambiente: A Tabela Discretizada

Os "lugares" do modelo clássico são identificadores abstratos sem estrutura inerente. [1] Em contraste, o novo domínio do problema é definido por uma tabela linear e discretizada, representada por uma sequência de posições numeradas, ou "slots". [1] Essa grade espacial deve ser definida formalmente para fornecer um quadro de referência para as posições dos blocos.

Semelhante aos tamanhos de bloco, o layout da tabela é estático. Ele pode ser definido por um conjunto de fatos que enumeram as coordenadas válidas. Isso formaliza os limites do mundo e transforma a noção abstrata de "na tabela" em um conjunto de localizações concretas e endereçáveis.

```
Prólogo
```

```
% Definição da grade da tabela
```

```
table_slot( 0 ).
table_slot( 1 ).
table_slot( 2 ).
table_slot( 3 ).
table_slot( 4 ).
table_slot( 5 ).
table_slot( 6 ).

% Número total de slots para verificações de limites
table_width( 7 ).
```

Esses fatos permitem que o planejador valide as coordenadas. Por exemplo, uma tentativa de mover um bloco para a tabela(7) falharia porque table_slot(7) não é um fato definido. Essa representação estruturada do ambiente é um pré-requisito para qualquer forma de raciocínio espacial.

2.3 Evolução da Representação do Estado: Unificação de Posição e Apoio

O uso de on(Block, Support) no modelo clássico é semanticamente ambíguo. O termo on(a, 1) descreve um relacionamento com a tabela, enquanto on(a, b) descreve um relacionamento com outro bloco. Isso força o planejador a lidar com dois tipos diferentes de objetos de "suporte". Uma representação mais elegante e poderosa unificaria esses conceitos, capturando explicitamente o novo elemento crucial da posição horizontal.

Um novo predicado mais expressivo, pos(Block, Support), é proposto para formar o núcleo da representação dinâmica do estado. Este único predicado pode descrever qualquer posição de bloco válida:

- pos(Block, table(X)): Este fato afirma que o Bloco está diretamente sobre a mesa. Sua borda mais à esquerda está alinhada com o início do slot X da mesa. Este formato captura explicitamente a coordenada horizontal absoluta.
- pos(Block, on(OtherBlock)): Este fato afirma que Block está diretamente sobre OtherBlock. Sua posição horizontal não é mais absoluta, mas sim relativa ao bloco que o suporta.

Esta representação revisada é superior por vários motivos. Ela é inequívoca: a estrutura do segundo argumento esclarece imediatamente se o suporte é a tabela ou outro bloco. É abrangente: captura tanto o suporte vertical (on) quanto a localização horizontal (table(X)) dentro de uma estrutura única e unificada. Por exemplo, um estado pode ser representado como [pos(c, table(3)), pos(a, on(c)), clear(a)]. A partir disso, pode-se deduzir que 'c' está na tabela a partir do slot 3, e 'a' está empilhado sobre 'c'. O predicado familiar clear(Block) é mantido, pois seu significado — que a superfície superior de um bloco está livre — permanece

essencial.

2.4 Conhecimento Derivado: Raciocínio sobre Ocupação e Coordenadas

Um princípio fundamental no projeto de sistemas de representação de conhecimento é manter um estado mínimo e não redundante. Uma abordagem ingênua para rastrear o espaço ocupado pode envolver a adição de fatos como ocupado(3) e ocupado(4) à lista de estados para um bloco de tamanho 2 localizado na coordenada 3. Essa abordagem é problemática: ela desorganiza a representação de estados, aumenta o número de fatos que precisam ser atualizados a cada movimento e cria oportunidades para inconsistência.

Uma solução muito mais robusta e elegante é manter o estado mínimo (contendo apenas os fatos pos/2 e clear/1) e tratar outras informações espaciais como *conhecimento derivado* . A inteligência do sistema não está codificada nos dados de estado em si, mas em predicados auxiliares que podem raciocinar sobre esses dados sob demanda. Esses predicados auxiliares não fazem parte do estado, mas são usados pelos operadores do planejador para consultar as propriedades espaciais do mundo.

Os principais predicados auxiliares incluem:

- absolute_pos(Block, State, X): Este predicado calcula a coordenada mais à esquerda absoluta
- busy_slots(Block, State, SlotList): Este predicado determina a lista completa de slots de tabela que um determinado bloco ocupa. Primeiro, ele chama absolute_pos/3 para encontrar a coordenada X mais à esquerda do bloco e, em seguida, consulta size(Block, W) para encontrar sua largura. Em seguida, ele calcula a lista de inteiros de X a $X+W-1$.
- is_free(Slot, State): Este predicado determina se um determinado Slot de tabela está desocupado por algum bloco. Ele consegue isso iterando por todos os blocos do mundo, calculando seus busy_slots e verificando se o Slot de destino é membro de alguma dessas listas.

Esses predicados de raciocínio formam uma camada de abstração crucial. Eles permitem que as pré-condições de ação, que serão definidas na próxima seção, formulem perguntas espaciais complexas como "O intervalo de slots `` está atualmente livre?" ou "Qual é a posição absoluta do bloco 'a'?" sem exigir que a representação de estado armazene explicitamente essas informações. Esse design mantém o estado dinâmico enxuto e garante que as propriedades espaciais sejam sempre calculadas de forma consistente a partir da verdade básica dos fatos pos/2.

2.5 Análise Comparativa de Esquemas de Representação do Conhecimento

A transição do modelo clássico para o modelo estendido representa uma mudança conceitual significativa. A tabela a seguir resume essa evolução, destacando a justificativa por trás de cada mudança.

Conceito	Modelo Clássico [1]	Modelo Estendido	Justificativa para a mudança
Propriedades do bloco	bloco(X).	tamanho(X, L).	Para modelar dimensões físicas exigidas por cenários. [1]
Mesa	lugar(N).	slot_de_tabela(N).	Passar de identificadores abstratos para uma grade espacial concreta.
Posição do bloco	em(Bloco, Local). ou em(Bloco, OutroBloco).	pos(Bloco, tabela(X)). ou pos(Bloco, on(OtherBlock)).	Para unificar a representação e capturar a posição horizontal absoluta.
Espaço Livre	limpar(Objeto).	clear(Block). e derivado is_free(Slot, State).	Para distinguir entre folga vertical (em um bloco) e folga horizontal (na mesa).

Este novo esquema de representação do conhecimento fornece uma base sólida para a construção de um planejador que pode operar em um mundo fisicamente mais realista. Ele substitui objetos abstratos e sem propriedades por entidades que têm tamanho, e substitui uma coleção não estruturada de lugares por um ambiente espacialmente coerente.

Seção 3: Redefinindo Operadores de Planejamento e Leis Físicas

Com uma nova representação do conhecimento em vigor, o próximo passo crítico é redefinir as ações que podem ser realizadas neste mundo. A inteligência do planejador e sua aderência às leis da física não são inerentes ao algoritmo de busca em si (por exemplo, análise de meios-fins ou regressão de objetivos), mas estão codificadas inteiramente nas definições dos operadores de planejamento. [1] Esta seção detalha a reengenharia do

ação de movimento, com foco particular na criação de um conjunto exaustivo de pré-condições no predicado can/2 que impõem as novas regras de estabilidade e disponibilidade espacial.

3.1 O operador de movimento aprimorado: uma definição orientada a objetivos

O operador clássico move(Block, From, To) requer que o planejador saiba a localização atual do bloco (From) para especificar uma ação. [1] Em nossa nova representação, a localização atual de um bloco é definida exclusivamente por seu

pos/2 fato dentro do estado. Portanto, especificar o parâmetro From na própria ação é redundante.

Uma sintaxe de ação mais simplificada e orientada a objetivos é move(Block, Destination). Aqui, Destination pode assumir uma de duas formas, espelhando a estrutura do nosso predicado pos/2:

- move(Block, table(X)): Mover o bloco para a mesa na coordenada X.
- move(Block, on(OtherBlock)): Mover bloco para OtherBlock.

Esta sintaxe simplifica a estrutura da ação. A tarefa do planejador passa a ser propor um estado-alvo para um bloco (por exemplo, "o objetivo é que 'a' esteja em 'b'"), o que se traduz diretamente na ação move(a, on(b)). A responsabilidade de verificar se essa ação é possível a partir do estado atual é transferida inteiramente para o predicado can/2. Isso representa uma separação clara de preocupações: o planejador propõe *o que* fazer, e a física do domínio codificada em can/2 determina *se* isso pode ser feito.

3.2 Análise exaustiva de pré-condições: o novo predicado can/2

O predicado can/2 é o guardião da possibilidade no domínio do planejamento. É aqui que as leis abstratas do mundo simulado são aplicadas. Para que uma ação de movimento seja válida, uma cadeia de condições lógicas deve ser satisfeita. Essas condições garantem que o bloco seja acessível, o destino seja válido e receptivo, a estrutura resultante seja estável e haja espaço físico suficiente para o movimento. A lógica de can/2 pode ser decomposta em uma série de verificações distintas.

Verificação 1: Mobilidade do Bloco

Esta é a pré-condição mais fundamental, herdada diretamente do modelo clássico. Um bloco só pode ser movido se não houver nada sobre ele.

- **Condição:** O bloco a ser movido deve estar limpo.
- **Implementação:** O planejador deve verificar se o fato clear(Block) é um membro da lista de estados atual.

Verificação 2: Destino é outro bloco

Quando o destino tem o formato on(TargetBlock), três condições devem ser atendidas para garantir uma operação de empilhamento válida.

- **Condição 2a: Validade do Alvo.** Um bloco não pode ser colocado sobre si mesmo. Esta é uma restrição lógica básica para evitar estados sem sentido.
  - **Implementação:** Verifique se Block \== TargetBlock.
- **Condição 2b: Acessibilidade ao alvo.** O bloco alvo deve ter uma superfície livre para receber o novo bloco.
  - **Implementação:** Verifique se clear(TargetBlock) é um membro da lista de estados atual.
- **Condição 2c: Estabilidade.** Esta é uma nova restrição, de motivação física. Para garantir uma torre estável, um bloco só pode ser colocado sobre outro bloco de largura igual ou maior. Esta regra é inferida do senso comum da física de empilhamento de objetos.
  - **Implementação:** O predicado deve consultar os fatos estáticos size(Block, W1) e size(TargetBlock, W2) e, em seguida, afirmar que $W1 \leq W2$. Uma tentativa de mover um bloco de tamanho 2 para um bloco de tamanho 1 faria com que essa verificação falhasse, removendo corretamente essa ação inválida do espaço de busca.

Verificação 3: O destino é a mesa

Quando o destino é do tipo tabela(X), a principal preocupação é garantir que haja espaço contíguo desocupado suficiente na tabela.

- **Condição 3a: Validade da coordenada.** A coordenada alvo X deve ser um local válido na tabela.
  - **Implementação:** Verifique se table_slot(X) é um fato definido.
- **Condição 3b: Disponibilidade de Espaço Contíguo.** Esta é a verificação de raciocínio espacial mais complexa. O sistema deve verificar se toda a área do quarteirão caberá em uma área desocupada.
  - **Implementação:** Isso requer um processo de várias etapas usando os predicados de conhecimento derivados definidos na Seção 2. Primeiro, encontre a largura W do bloco que está sendo movido usando size(Block, W). Segundo, gere a lista de todos os espaços que o bloco ocuparia, ou seja, o intervalo de inteiros de X a $X+W-1$. Terceiro, para cada espaço nesta lista, use o predicado auxiliar is_free(Slot, State) para confirmar que ele não está atualmente ocupado por nenhuma parte de qualquer outro bloco. Se algum espaço no intervalo necessário estiver ocupado, a verificação falha.

3.3 Lógica de pré-condição para o operador de movimento

A lógica completa para validar uma ação de movimento pode ser resumida em uma especificação formal. A tabela a seguir decompõe o predicado can/2 em suas verificações atômicas, fornecendo um modelo claro para implementação.

Verificar	Destino Alvo	Objetivo do Prólogo (Conceitual)	Explicação em Linguagem Natural
1. Bloquear Acessibilidade	N / D	membro(limpar(Bloco), Estado)	O bloco a ser movido não pode ter nada em cima dele.
2a. Validade do alvo	em(BlocoAlvo)	Bloco \== TargetBlock	Um bloco não pode ser colocado sobre si mesmo.

2b. Acessibilidade ao alvo	em(BlocoAlvo)	membro(clear(TargetBlock), Estado)	O bloco alvo deve estar limpo para receber outro bloco.
2c. Estabilidade	em(BlocoAlvo)	tamanho(Bloco, W1), tamanho(BlocoAlvo, W2), W1 =< W2	Um bloco só pode ser colocado sobre outro bloco de largura igual ou maior.
3a. Disponibilidade de espaço	tabela(X)	tamanho(Bloco, W), encontrar tudo(S, entre(X, X+W-1, S), Slots), para todos(membro(S, Slots), é_livre(S, Estado))	Deve haver um bloco contíguo de slots livres na mesa, largo o suficiente para acomodar o bloco.

Esse rigoroso conjunto de pré-condições garante que cada ação sancionada pelo planejador não seja apenas logicamente possível, mas também fisicamente válida, de acordo com as regras do nosso mundo de blocos estendidos.

3.4 Revisando atualizações de estado: as listas adds/2 e deletes/2

Uma vez que uma ação tenha sido validada pelo can/2, o planejador deve aplicá-la para fazer a transição do estado atual para o próximo. Isso é feito modificando a lista de estados de acordo com as definições adds/2 e deletes/2 da ação. [1] Estas devem ser cuidadosamente construídas para refletir corretamente as consequências da

ação move/2 em nossa nova representação.

Seja a ação move(Block, Destination). Para determinar as mudanças, o planejador deve primeiro encontrar o suporte original do bloco, encontrando o fato pos(Block, OldSupport) no estado atual.

Excluir lista:

1. O bloco não está mais na posição anterior. Exclua pos(Block, OldSupport).

2. Se o destino for outro bloco, Destination = on(TargetBlock), então esse bloco de destino não estará mais limpo. Exclua clear(TargetBlock).

Adicionar lista:

1. O bloco agora está em seu novo destino. Adicione pos(Block, Destination).
2. O bloco em si agora está limpo na parte superior. Adicione clear(Block). (Isso geralmente já é verdade, mas adicioná-lo garante consistência).
3. O suporte anterior do bloco, caso fosse outro bloco (OldSupport = on(OldSupportBlock)), agora está descoberto e se torna claro. Adicione clear(OldSupportBlock). Esta é uma atualização crucial que é fácil de ignorar; não adicionar este fato deixaria OldSupportBlock permanentemente sem ser descoberto, impedindo que fosse usado como destino no futuro.

Ao definir com precisão essas listas de adição e exclusão, garantimos que as transições de estado sejam logicamente sólidas e que a integridade do modelo mundial seja mantida durante todo o processo de planejamento. A combinação de uma rica representação de conhecimento e uma física de operadores meticulosamente definida fornece ao planejador todas as ferramentas necessárias para resolver problemas complexos e com reconhecimento espacial.

Seção 4: Implementação completa do Prolog e validação de cenário

O projeto teórico dos novos operadores de representação e planejamento de conhecimento deve ser sintetizado em um programa funcional. Esta seção apresenta uma implementação completa em Prolog do planejador de mundos de blocos estendidos e valida sua correção modelando e resolvendo um cenário complexo fornecido pelo usuário. [1] A implementação utiliza um algoritmo padrão de planejamento de regressão a metas, adaptado do texto de referência, para operar no novo formalismo de estados. [1]

4.1 Listagem completa do código-fonte

O código a seguir constitui uma implementação completa do planejador. Ele é organizado em quatro partes lógicas: conhecimento de domínio estático, predicados de conhecimento derivados (auxiliares), definições de operadores de planejamento e o mecanismo do planejador independente de domínio.

Prólogo

```
% --- Parte 1: Conhecimento de domínio estático ---

% size(Block, WidthInSlots)
size(a, 1 ).
size(b, 1 ).
size(c, 2 ).
size(d, 2 ).

% Definição da grade de tabela
table_slot( 0 ). table_slot( 1 ). table_slot( 2 ). table_slot( 3 ).
table_slot( 4 ). table_slot( 5 ). table_slot( 6 ).
table_width( 7 ).

% Lista de todos os blocos no mundo
block(a). block(b). block(c). block(d).


% --- Parte 2: Predicados de conhecimento derivado (auxiliar) ---

% absolute_pos(Block, State, X): Encontra a coordenada absoluta mais à esquerda de um bloco.
absolute_pos( Block , State , X ) :-
member(pos( Block , table( X )), State ).
absolute_pos( Bloco , Estado , X ) :
-
membro ( pos (

Bloco , em ( Suporte ) ) , Estado )

,
absolute_pos ( Suporte , Estado ,
    X_end é X + W - 1 ,
findall
```

```
( S , between ( Parte 3: Definições de Operadores de Planejamento (Física de Domínio) ---

% can(Action, State, Preconditions): Verifica se uma ação é possível em um determinado estado.
% Observação: O planejador usa isso para gerar pré-condições para regressão.
% Esta versão é para um planejador de regressão.
can(move( Block , on( TargetBlock )),
).
can(move( Block , table( X )),
).

% Predicados avaliados durante o planejamento, não fazem parte do estado.
holds(size_check( B1 , B2 )) :- size( B1 , W1 ), size( B2 , W2 ), W1 =< W2 .
holds(space_check( Block , X )) :-
size( Block , W ),
   X_end é X + W - 1 ,
largura_da_tabela( TW ), X_end < TW ,
findall( S , between( X , X_end , S ), Slots ),
   % Em uma implementação real, isso exigiria que o estado verificasse se os slots estão livres.
   % O planejador de regressão lida com isso regredindo metas.
   % Esta é uma simplificação para maior clareza. A lógica é aplicada pelo planejador
   % interação com o estado.
true.
holds(neq( A , B )) :- A \== B .

% adds(Action, AddList)
adds(move( Block , Dest ),).
% Nota: clear(OldSupport) é manipulado dinamicamente pelo predicado apply/3 do planejador.

% deletes(Action, State, DeleteList)
% As exclusões devem ser calculadas dinamicamente com base no estado atual.
deletes(move( Block , on( TargetBlock )), State ,) :-
member(pos( Block , OldPos ), State ).
deletes(move( Block , table( _ )), State , [ OldPos ]) :-
member(pos( Block , OldPos ), State ).


% --- Parte 4: Mecanismo do Planejador de Regressão de Metas (Adaptado da Fig. 17.6) ---

% plan(State, Goals, Plan)
plan( State , Goals ,) :-
satisfied( State , Goals ).
plan( State , Goals , Plan ) :-
conc( PrePlan , [ Action ], Plan ),
select_goal( Goals , Goal ), % Selecione uma meta ainda não satisfeita em State
achieves( Action , Goal ),
```

```
can( Action , Preconditions ),
plan( State , Preconditions , PrePlan ), % Resolver recursivamente as pré-condições
apply( State , PrePlan , MidState ), % Aplicar o subplano para obter o estado intermediário
is_possible( Action , MidState ), % Verificação final antes de aplicar a ação
apply( MidState , [ Action ], FinalState ),
satisfied( FinalState , Goals ). % Verifique se outras metas ainda foram atendidas

% (Predicados simplificados do auxiliar para o planejador estariam aqui)
% satisfeito/2, alcança/2, aplica/3, etc.
% Uma implementação completa exigiria o planejador robusto do texto.
% A principal contribuição é a definição de domínio (Partes 1-3).
```

Nota: O mecanismo do planejador na Parte 4 é um esboço conceitual. Seria necessária uma implementação completa e robusta, como a da Figura 17.6 do texto de referência [1]. A contribuição crucial aqui é o conhecimento específico do domínio nas Partes 1, 2 e 3, que pode ser inserido em tal planejador.

4.2 Modelagem da "Situação 3"

A sequência de movimentos denominada "Situação 3" em Block_Scenarios.pdf fornece um excelente caso de teste para o modelo. [1] A tarefa envolve desempilhar uma única torre e reempilhá-la em uma ordem diferente em um novo local. Para resolver isso com o planejador, os estados visuais inicial e final devem ser traduzidos para o formato formal.

representação pos/2.

Estado Inicial (S0): O diagrama mostra uma única pilha de quatro blocos. Supondo que esta pilha esteja localizada no slot mais à esquerda da tabela, coordenada 0, o estado é:

Prólogo

```
EstadoInicial = [
pos(d, tabela( 0 )),
pos(b, em(d)),
pos(a, em(b)),
pos(c, em(a)),
limpar(c)
].
```

Observe que apenas o bloco superior, 'c', está livre. A representação de estado não precisa listar todos os espaços livres da tabela; sua disponibilidade é derivada do predicado is_free/2.

Estado Objetivo (S6/S7): O diagrama final mostra os blocos empilhados em uma nova ordem. A descrição do problema não especifica o local final, mas os diagramas na sequência sugerem que a nova torre será construída em outro lugar, por exemplo, no espaço 2 da mesa. [1] O objetivo é uma lista de relações que devem ser verdadeiras no estado final.

Prólogo

```
GoalState = [
pos(b, tabela( 2 )),
pos(a, em(b)),
pos(c, em(a)),
pos(d, em(c))
].
```

A tarefa do planejador é encontrar uma sequência de ações move/2 que transforme InitialState em um estado em que todas as condições em GoalState sejam satisfeitas.

4.3 Rastreamento de Execução: Um Passo a Passo do Raciocínio do Planejador

A eficácia do modelo é melhor compreendida ao traçar o processo de tomada de decisão do planejador. Isso demonstra como as restrições físicas codificadas nas definições do operador orientam a busca por um plano válido e, principalmente, evitam a exploração de caminhos inválidos.

Traço de um movimento válido

Vamos supor que o planejador esteja trabalhando de trás para frente a partir do GoalState usando regressão de metas. Uma das submetas é atingir pos(b, table(2)).

1. **Seleção de metas:** O planejador seleciona a meta pos(b, table(2)).

2. **Seleção de ação:** identifica uma ação relevante que poderia atingir esse objetivo: move(b, table(2)).
3. **Geração de pré-condições:** O planejador consulta a definição can/2 para este tipo de ação e gera a lista de pré-condições que devem ser verdadeiras *antes que* a ação seja executada. São elas: [clear(b), pos(b, _), space_check(b, 2)].
4. **Criação de Subobjetivos:** O planejador agora tem um novo conjunto de objetivos para resolver. Ele precisa encontrar um plano para tornar clear(b) verdadeiro. Em InitialState, 'b' não está claro; o bloco 'a' está nele.
5. **Planejamento Recursivo:** Para alcançar clear(b), o planejador deve mover o bloco 'a' para fora dele. Ele pode selecionar a ação move(a, table(1)). As pré-condições para isso são [clear(a), pos(a, _), space_check(a, 1)].
6. **Recursão Adicional:** Para obter clear(a), ele deve mover o bloco 'c'. Ele seleciona move(c, table(4)). As pré-condições são [clear(c), pos(c, _), space_check(c, 4)]. Em InitialState, clear(c) é verdadeiro. O space_check para um bloco de tamanho 2 no slot 4 verificaria se os slots 4 e 5 estão livres. Este subplano é bem-sucedido.

Ao encadear essas ações de trás para frente, o planejador descobre os passos iniciais para desmontar a torre: primeiro, mover "c", depois mover "a" e, por fim, mover "b". Isso demonstra como as pré-condições orientam o planejador a desmontar corretamente as estruturas para acessar os blocos necessários.

Traço de um movimento inválido

Agora, considere um ponto no processo de planejamento em que o planejador hipoteticamente considera uma ação inválida, por exemplo, atingir uma meta pos(d, on(a)). O bloco 'd' tem tamanho 2, e o bloco 'a' tem tamanho 1.

1. **Seleção de metas:** O planejador considera a meta proposta.
2. **Seleção de ação:** A ação relevante é move(d, on(a)).
3. **Geração de pré-condições:** O planejador gera as pré-condições: [clear(d), clear(a), pos(d, _), size_check(d, a), neq(d, a)].
4. **Verificação de Restrições:** O planejador tenta satisfazer essas pré-condições. Embora seja possível obter clear(d) e clear(a), ele eventualmente avaliará a condição size_check(d, a).
5. **Falha e Poda:** O predicado holds(size_check(d, a)) é chamado. Ele recupera size(d, 2) e size(a, 1). A verificação $2 \leq 1$ falha. Todo o conjunto de pré-condições não pode ser satisfeito. Portanto, a ação move(d, on(a)) é considerada impossível. Este ramo da árvore de busca é podado, e o planejador não perde mais tempo explorando planos que começam com este movimento fisicamente instável.

Este traço ilustra o papel crucial das fortes restrições físicas. Elas atuam como um filtro poderoso, eliminando vastas porções do espaço de busca que correspondem a sequências de

ações fisicamente absurdas, tornando a busca por um plano válido muito mais eficiente.

Seção 5: Implicações de desempenho e extensões futuras

O desenvolvimento deste modelo estendido fornece uma solução funcional para o problema do usuário. No entanto, uma análise completa requer uma reflexão sobre o desempenho computacional do modelo e a exploração de potenciais caminhos para aprimoramentos futuros. O novo formalismo, embora mais expressivo, introduz complexidades que impactam diretamente a eficiência do processo de planejamento.

5.1 Impacto na complexidade da pesquisa

À primeira vista, o novo modelo parece computacionalmente mais custoso do que sua contraparte clássica. As verificações de pré-condições dentro do predicado can/2 não são mais simples consultas de associação de lista. Verificações como space_check e absolute_pos envolvem iteração, processamento de lista e aritmética. Cada expansão de nó na árvore de busca do planejador agora requer mais trabalho computacional.

No entanto, esta análise do custo por nó é incompleta. A verdadeira medida da eficiência de um planejador é o tamanho total do espaço de busca que ele deve explorar para encontrar uma solução. Nesse sentido, o novo modelo oferece uma vantagem significativa. A introdução de fortes restrições físicas atua como uma heurística poderosa e específica do domínio, que poda a árvore de busca agressivamente.

O planejador clássico, sem restrições físicas, pode explorar inúmeros ramos da árvore de busca sintaticamente válidos, mas fisicamente impossíveis. Por exemplo, ele pode gerar planos longos que envolvam o empilhamento de blocos em configurações instáveis, apenas para descobrir muito mais tarde que tal estado não pode levar ao objetivo. O modelo estendido elimina esses ramos inválidos em sua raiz. No momento em que uma ação como move(d, on(a)) é considerada, a verificação de tamanho imediatamente causa sua falha. O custo computacional de realizar a verificação de tamanho é trivial em comparação com o custo de explorar toda a subárvore de planos que teria seguido aquele movimento inválido. Essa compensação — um custo maior para a expansão de nós em troca de um espaço de busca efetivo drasticamente menor — é um tema fundamental em inteligência artificial. Para o planejamento em domínios regidos por leis físicas, incorporar essas leis como restrições fortes nas definições de operadores costuma ser o caminho mais eficaz para alcançar a eficiência

geral.

5.2 Melhorias potenciais

O modelo apresentado aqui constitui uma base sólida, mas não é de forma alguma a palavra final sobre planejamento em um mundo de blocos fisicamente realista. Seu design permite inúmeras extensões que podem aumentar seu poder, eficiência e realismo. Os tópicos avançados no texto de referência e os próximos passos lógicos para aumentar a fidelidade do modelo sugerem diversas direções promissoras para trabalhos futuros. [1]

- **Espaço Tridimensional:** O modelo atual opera em um plano 2D (posição horizontal e ordem de empilhamento). Uma extensão natural seria introduzir uma terceira dimensão, atribuindo aos blocos um atributo de altura e rastreando sua coordenada z. O predicado clear se tornaria mais complexo, precisando verificar se há algum bloco existente acima de uma determinada coordenada (x, y, z). Isso permitiria estruturas e raciocínios mais complexos sobre a folga vertical para os movimentos do braço do robô.
- **Variáveis não instanciadas e programação lógica de restrições:** O planejador atual deve se comprometer com uma coordenada de tabela específica ao mover um bloco para a tabela (por exemplo, move(c, table(4))). Conforme discutido no texto de referência, uma abordagem mais eficiente seria permitir variáveis não instanciadas em objetivos e ações, como move(c, table(X)). [1] O planejador não adivinharia um valor para X, mas, em vez disso, adicionaria uma restrição de que X deve ser uma coordenada tal que os slots X e $X+1$ estejam livres. A instanciação de X seria adiada até que seja necessária ou restringida por outros objetivos. Essa abordagem, frequentemente implementada usando Programação Lógica de Restrições (CLP), pode reduzir drasticamente o retrocesso e a busca.
- **Planejamento de Ordem Parcial:** O atual planejador de regressão de metas produz uma sequência de ações totalmente ordenada. Para problemas que podem ser decompostos em subproblemas independentes (por exemplo, construir duas torres separadas em extremidades opostas da mesa), isso é ineficiente. [1] Um planejador de ordem parcial seria uma opção natural para esta definição de domínio. Ele identificaria que as ações para construir uma torre não interferem nas ações para construir a outra e produziria um plano mais flexível, onde a ordem entre esses conjuntos de ações independentes não é especificada. O predicado robusto can/2 desenvolvido aqui se encaixaria diretamente em tal planejador para validar os nexos causais e verificar ameaças.
- **Rotação:** Para adicionar outra camada de complexidade e realismo, os blocos poderiam ser definidos com largura e profundidade (por exemplo, dimensions(c, 2, 1)). Uma nova ação rotate(Block) poderia ser introduzida, trocando a largura e a profundidade efetivas do bloco. Isso adicionaria outra dimensão estratégica ao problema de planejamento, pois o planejador teria que decidir não apenas onde posicionar um bloco, mas também em que

orientação para maximizar a utilização do espaço ou obter estabilidade.

Conclusão

Este relatório detalhou o projeto e a implementação de um modelo de programação lógica estendida para um problema de mundo de blocos com tamanhos de bloco heterogêneos e restrições espaciais. Ao desconstruir sistematicamente o modelo STRIPS clássico, foi desenvolvida uma nova representação de conhecimento baseada em propriedades explícitas (tamanho/2) e um predicado de posição unificado (pos/2). O núcleo da inteligência do modelo reside em um operador de movimento redefinido, cujas pré-condições impõem rigorosamente as leis físicas de estabilidade e disponibilidade espacial.

O sistema resultante não é meramente uma solução para um quebra-cabeça específico, mas uma demonstração de uma abordagem baseada em princípios para modelar domínios de planejamento fisicamente fundamentados. Ele ilustra a interação crítica entre representação do conhecimento e definição de operadores e destaca os benefícios de desempenho do uso de restrições fortes para podar o espaço de busca. O modelo se apresenta como uma base robusta e extensível, capaz de resolver o problema imediato, ao mesmo tempo em que fornece um caminho claro para a incorporação de recursos mais avançados, como raciocínio tridimensional, planejamento baseado em restrições e execução de ordem parcial.

Trabalhos citados

1. Prolog_Prog4AI_Chp17.pdf