Proposta de Solução para o Problema de Planejamento do Mundo dos Blocos com Tamanhos Variáveis Utilizando NuSMV

Disciplina: Fundamentos de Inteligência Artificial **Professor:** Edjard Mota **Data:** 04 de Outubro de 2025

1.1. O Problema Canônico do Mundo dos Blocos

O problema do "Mundo dos Blocos" é um dos exemplos mais canônicos e estudados no campo do planejamento automatizado em Inteligência Artificial. Em sua forma clássica, ele consiste em um conjunto de blocos de tamanho uniforme, uma mesa e um braço de robô capaz de realizar uma única ação: mover um bloco de um lugar para outro. A simplicidade do domínio permite uma ilustração clara de princípios fundamentais de planejamento, como busca em espaço de estados, análise de meios-fins e regressão de objetivos. A representação padrão para este problema é o formalismo STRIPS (Stanford Research Institute Problem Solver), que modela o estado do mundo como um conjunto de fórmulas lógicas e as ações como operadores que modificam esse conjunto de fórmulas.

1.2. O Desafio da Heterogeneidade: Blocos com Tamanhos Variáveis

O modelo clássico, embora pedagogicamente poderoso, falha ao tentar representar cenários mais realistas. A introdução de blocos com tamanhos (larguras) heterogêneos transforma fundamentalmente o problema. Essa mudança introduz duas novas classes de restrições físicas que precisam ser gerenciadas pelo planejador:

1. **Ocupação Espacial:** Um bloco largo ocupa múltiplos espaços discretos em uma mesa, e o planejador deve garantir que haja espaço contíguo suficiente antes de mover um bloco para a mesa. O predicado `on(a, 1)` do modelo clássico é insuficiente, pois trata a localização '1' como um identificador abstrato sem dimensão.
2. **Estabilidade Física:** Em um mundo físico, um bloco largo não pode ser empilhado de forma estável sobre um bloco mais estreito. O planejador deve incorporar essa "lei da física" para evitar a geração de planos que resultem em configurações instáveis e inválidas.

Essas restrições tornam o esquema de representação de conhecimento clássico fundamentalmente inadequado.

1.3. Abordagem Proposta: Planejamento como Verificação de Modelo

Para superar essas limitações, propomos uma abordagem que utiliza a técnica de "Planejamento como Verificação de Modelo" (

Planning as Model Checking). A ideia central é codificar todo o problema de planejamento — os estados, as ações e as restrições — em um sistema de transição de estados formal e, em seguida, usar um verificador de modelos, como o NuSMV, para encontrar a solução.

A estratégia é a seguinte:

- O estado do mundo (a posição de cada bloco) é representado por variáveis de estado no NuSMV.
- As ações (mover um bloco) são modeladas como as transições possíveis entre os estados. As pré-condições da ação (bloco livre, espaço disponível, estabilidade) atuam como guardas que determinam se uma transição é válida.
- O estado inicial e o estado objetivo são definidos formalmente.
- Uma propriedade, expressa em Lógica Temporal, é especificada para afirmar que o estado objetivo
  nunca pode ser alcançado.
- O NuSMV tenta provar essa propriedade. Se a propriedade for falsa, significa que o objetivo é, de fato, alcançável. Como prova, o NuSMV gera um **contraexemplo**: uma sequência de estados e ações que leva do estado inicial até o estado objetivo. Essa sequência é, precisamente, o plano que procuramos.

Este relatório detalha cada passo desse processo, desde a análise dos modelos alternativos até a construção completa do modelo em NuSMV e a interpretação dos resultados.

---

2. Análise Comparativa de Paradigmas de Representação

Antes de construir o modelo NuSMV, é crucial entender as forças e fraquezas das abordagens alternativas, que informam nosso projeto de representação.

2.1. O Modelo Clássico STRIPS e Suas Limitações

O formalismo STRIPS representa o mundo através de três componentes principais:

- **Estado:** Um conjunto finito de relações lógicas (predicados atômicos) que são consideradas verdadeiras, como `on(a, b)` (bloco 'a' está sobre o bloco 'b') e `clear(a)` (a superfície do bloco 'a' está livre).
- **Operador de Ação:** Um esquema que define uma ação, como `move(Block, From, To)`. Cada operador possui:
  - **Pré-condições:** Condições que devem ser verdadeiras no estado atual para que a ação seja executável (ex: `clear(Block)`, `clear(To)`).
  - **Lista de Adição (Add-List):** Novos predicados que se tornam verdadeiros após a ação (ex: `on(Block, To)`).
  - **Lista de Exclusão (Delete-List):** Predicados que deixam de ser verdadeiros (ex: `on(Block, From)`).

As

limitações críticas desta abordagem para o nosso problema são:

- **Ausência de Propriedades Físicas:** O modelo trata todos os blocos como entidades uniformes, sem atributos como tamanho ou largura. Isso impossibilita o raciocínio sobre estabilidade.

- **Inadequação para Raciocínio Espacial:** A representação `on(a, 1)` é puramente relacional. Os "lugares" (1, 2, 3...) são identificadores abstratos sem propriedades espaciais como adjacência, distância ou tamanho. O modelo não consegue expressar que um bloco ocupa um intervalo de coordenadas (ex: slots 2 e 3 simultaneamente).

2.2. A Solução Estendida em Programação Lógica (Prolog)

Os documentos fornecidos propõem uma representação em Prolog que supera diretamente essas limitações. Esta proposta serve como um "projeto" conceitual para o nosso modelo NuSMV.

- **Representação de Conhecimento Estático:** A principal extensão é a introdução de um banco de fatos imutáveis que descrevem as propriedades físicas dos objetos. O predicado `tamanho(Bloco, Largura)` associa a cada bloco uma dimensão.
- **Representação de Estado Dinâmico:** O modelo unifica a representação da posição com o predicado `pos(Bloco, Suporte)`. O `Suporte` pode ser `mesa(X)`, que ancora o bloco a uma coordenada horizontal absoluta, ou `sobre(OutroBloco)`, que define uma relação de empilhamento. Esta representação é superior porque é inequívoca e captura tanto o suporte vertical quanto a localização horizontal em uma única estrutura.
- **Conhecimento Derivado:** Em vez de sobrecarregar o estado com predicados como `ocupado(3)`, `ocupado(4)`, a disponibilidade de espaço na mesa é tratada como conhecimento derivado. Predicados auxiliares, como `is_free(Slot, State)`, calculam essa informação sob demanda a partir das posições (`pos/2`) e tamanhos (`tamanho/2`) dos blocos. Isso mantém a representação de estado mínima e consistente.
- **Operadores com Física Embutida:** A ação de movimento é redefinida com pré-condições rigorosas que impõem as leis físicas do domínio, incluindo a verificação de estabilidade (`tamanho(B1) =< tamanho(B2)`) e a verificação de espaço contíguo disponível na mesa.

2.3. Tabela Comparativa de Conceitos

A tabela a seguir, adaptada da fonte, resume a evolução da representação.

Conceito	Modelo Clássico STRIPS	Modelo Estendido (Prolog / NuSMV)	Justificativa para a Mudança
Propriedades do Bloco	`bloco(X).`	`DEFINE size_x := W;`	Para modelar as dimensões físicas exigidas pelos cenários de estabilidade e ocupação espacial.

Posição do Bloco	`on(Bloco, Lugar)` ou `on(Bloco, OutroBloco)`	`VAR pos_bloco: {on_outro, table_x, ...};`	Para unificar a representação e capturar a posição horizontal absoluta, passando de identificadores abstratos para uma grade espacial concreta.
Espaço Livre	`clear(Objeto).`	`DEFINE is_free(s) := ...;` (Derivado)	Para distinguir entre vacância vertical (em um bloco) e vacância horizontal (na mesa), e para manter o estado mínimo calculando o espaço livre sob demanda.
Estabilidade	Inexistente.	`size_bloco1 <= size_bloco2`	Uma nova restrição, ausente no modelo clássico, para impedir a criação de pilhas fisicamente instáveis.

Exportar para as Planilhas

---

3. Metodologia: Planejamento como Verificação de Modelo

3.1. Conceitos Fundamentais da Verificação de Modelos

A verificação de modelos é uma técnica de verificação formal automatizada. O processo envolve três componentes principais:

1. **Modelo do Sistema:** Uma descrição formal de um sistema como um sistema de transição de estados (um grafo onde os nós são estados e as arestas são transições). Em nosso caso, este é o modelo do Mundo dos Blocos, com todas as suas configurações e ações possíveis.
2. **Especificação da Propriedade:** Uma propriedade formal, geralmente escrita em uma lógica temporal como LTL (Linear Temporal Logic) ou CTL (Computation Tree Logic), que descreve o comportamento desejado (ou indesejado) do sistema.
3. **Verificador:** Um algoritmo que explora exaustivamente o espaço de estados do modelo para determinar se a especificação da propriedade é verdadeira ou falsa.

A ideia chave para o planejamento é codificar o problema de encontrar uma sequência de ações em um problema de verificação de modelo.

3.2. A Estratégia do Contraexemplo para Geração de Planos

A nossa abordagem, conforme descrito na literatura, utiliza a capacidade do verificador de modelos de gerar contraexemplos. Um contraexemplo é uma prova que demonstra por que uma propriedade é falsa.

O processo funciona da seguinte forma:

1. **Codificação:** O estado inicial (`I`), os operadores de ação (`OP`) e o estado objetivo (`G`) do problema de planejamento são codificados no NuSMV.
2. **Especificação Negativa:** Em vez de perguntar "É possível alcançar o objetivo G?", formulamos a pergunta de forma negativa: "É verdade que o objetivo G **nunca** é alcançável?". Em CTL, isso é expresso como `!EF(goal)`.
    - `E` significa "Existe um caminho (path)".
    - `F` significa "Em algum ponto no Futuro".
    - Portanto, `EF(goal)` significa "Existe um caminho onde, em algum ponto no futuro, o objetivo é alcançado".
    - `!EF(goal)` é a negação: "Não existe um caminho onde o objetivo é alcançável".
3. **Verificação e Geração do Plano:** O NuSMV assume que a especificação `!EF(goal)` é verdadeira e tenta encontrar uma prova.
    - **Se a propriedade for verdadeira**, significa que o objetivo é de fato inalcançável, e o problema de planejamento não tem solução.
    - **Se a propriedade for falsa**, significa que a suposição de que o objetivo era inalcançável estava errada. Para provar que é falsa, o NuSMV deve fornecer um **contraexemplo**. Este contraexemplo é uma trilha de execução — uma sequência de estados e transições (ações) — que começa no estado inicial e termina no estado `goal`.
    - **Este contraexemplo é o plano que procuramos.** Ele representa a sequência exata de ações que leva do estado inicial ao estado objetivo, respeitando todas as regras e restrições codificadas no modelo.

Essa metodologia transforma um problema de busca (planejamento) em um problema de verificação, alavancando os algoritmos de exploração de espaço de estados altamente otimizados dos verificadores de modelo modernos.

---

4. Modelagem Detalhada em NuSMV - Parte 1: Estrutura e Estado

Nesta seção, iniciamos a construção do modelo NuSMV, traduzindo os conceitos da representação estendida para a sintaxe formal da ferramenta.

4.1. Estrutura Geral do Módulo NuSMV

Todo modelo NuSMV é encapsulado dentro de um módulo, tipicamente chamado `main`. A estrutura do nosso arquivo `.smv` seguirá uma organização lógica clara, separando conhecimento estático, estado dinâmico, conhecimento derivado e a lógica de transição.

Snippet de código
```
MODULE main
  -- Seção 1: Definição do Domínio Estático (DEFINE)
  -- Seção 2: Representação do Estado Dinâmico (VAR)
  -- Seção 3: Conhecimento Derivado (DEFINE)
  -- Seção 4: Lógica de Transição (ASSIGN, TRANS)
  -- Seção 5: Especificação (DEFINE goal, CTLSPEC)
```

4.2. Definição do Domínio Estático: Propriedades dos Blocos (`DEFINE`)

As propriedades intrínsecas e imutáveis dos blocos, como seus tamanhos, são definidas como constantes simbólicas usando

`DEFINE`. Isso é computacionalmente mais eficiente do que usar variáveis de estado, pois esses valores são fixos e não contribuem para a explosão do espaço de estados.

Snippet de código
```
-- Definição das propriedades estáticas dos blocos (largura em slots)
DEFINE
  size_a := 1;  -- Bloco 'a' tem largura 1
  size_b := 1;  -- Bloco 'b' tem largura 1
  size_c := 2;  -- Bloco 'c' tem largura 2
  size_d := 3;  -- Exemplo: Bloco 'd' tem largura 3
```

4.3. Representação do Estado Dinâmico: Posição e Ação (`VAR`)

As variáveis de estado (`VAR`) definem os aspectos do mundo que mudam ao longo do tempo. Em nosso modelo, estas são as posições de cada bloco e a ação que está sendo executada em um determinado passo.

Variáveis de Posição: Para cada bloco, criamos uma variável de estado enumerada. O tipo enumerado inclui todos os possíveis "suportes" para o bloco: qualquer outro bloco (`on_x`) ou qualquer coordenada inicial válida na mesa (`table_y`). Esta abordagem implementa diretamente o predicado unificado

`pos(Bloco, Suporte)` da proposta em Prolog.

Snippet de código
```
VAR
  -- Para cada bloco, uma variável representa sua posição.
  -- O suporte pode ser a mesa (table_X) ou outro bloco (on_Y).
  pos_a: { on_b, on_c, on_d, table_0, table_1, table_2, table_3, table_4, table_5, table_6 };
```

pos_b: { on_a, on_c, on_d, table_0, table_1, table_2, table_3, table_4, table_5, table_6 };
pos_c: { on_a, on_b, on_d, table_0, table_1, table_2, table_3, table_4, table_5, table_6 };
pos_d: { on_a, on_b, on_c, table_0, table_1, table_2, table_3, table_4, table_5, table_6 };

Variável de Ação: Para modelar a execução de ações, introduzimos uma variável `action`. Em cada passo de tempo, o modelo pode escolher não deterministicamente uma ação desta lista. A lógica de transição usará o valor desta variável para determinar como o estado do sistema deve evoluir.

Snippet de código

```
-- Variável de controle para representar a ação a ser executada em cada passo.
action: {
    move_a_to_b, move_a_to_c, move_a_to_d, move_a_to_table_0, move_a_to_table_1,
    move_b_to_a, move_b_to_c, move_b_to_d, move_b_to_table_0, move_b_to_table_1,
    move_c_to_a, move_c_to_b, move_c_to_d, move_c_to_table_0, move_c_to_table_1,
    move_d_to_a, move_d_to_b, move_d_to_c, move_d_to_table_0, move_d_to_table_1,
    none -- Ação nula, representa a ausência de movimento.
};
```

Nota: A lista de ações deve ser exaustiva para todas as combinações válidas de bloco-destino.

---

5. Modelagem Detalhada em NuSMV - Parte 2: Conhecimento Derivado

Um princípio fundamental de uma boa representação de conhecimento é manter um estado mínimo e não redundante. Propriedades que podem ser logicamente inferidas a partir do estado principal devem ser tratadas como conhecimento derivado. Em NuSMV, isso é elegantemente alcançado usando

`DEFINE` para criar macros reutilizáveis.

5.1. Predicado de Vacância Vertical (`clear`)

O predicado

`clear(Objeto)` do modelo clássico (ou

`livre(Bloco)` na proposta Prolog) afirma que a superfície superior de um objeto está desobstruída. Em nosso modelo, um bloco

`X` está livre se nenhum outro bloco `Y` estiver posicionado sobre ele (ou seja, `pos_y != on_x`).

Snippet de código
DEFINE

```
-- Um bloco X está livre (clear) se nenhum outro bloco Y está sobre ele (pos_y != on_x).
clear_a := (pos_b != on_a) & (pos_c != on_a) & (pos_d != on_a);
clear_b := (pos_a != on_b) & (pos_c != on_b) & (pos_d != on_b);
clear_c := (pos_a != on_c) & (pos_b != on_c) & (pos_d != on_c);
clear_d := (pos_a != on_d) & (pos_b != on_d) & (pos_c != on_d);
```

Esta definição é uma tradução direta da lógica relacional para o formalismo do NuSMV.

5.2. Raciocínio sobre Ocupação Espacial na Mesa

Este é um dos aspectos mais críticos do modelo estendido. Precisamos de uma maneira de determinar quais "slots" da mesa estão ocupados por cada bloco, considerando seu tamanho. Usamos uma estrutura `case...esac` para definir a "pegada" de cada bloco.

Snippet de código

```
-- Macro parametrizada para determinar se um bloco X ocupa um slot S.
-- Exemplo para o bloco 'a' (tamanho 1)
occupied_by_a(s) := case
    pos_a = table_0 : s = 0;
    pos_a = table_1 : s = 1;
    pos_a = table_2 : s = 2;
    pos_a = table_3 : s = 3;
    pos_a = table_4 : s = 4;
    pos_a = table_5 : s = 5;
    pos_a = table_6 : s = 6;
    TRUE : FALSE; -- Se 'a' não está na mesa, não ocupa nenhum slot.
esac;

-- Exemplo para o bloco 'c' (tamanho 2)
occupied_by_c(s) := case
    pos_c = table_0 : (s = 0) | (s = 1);
    pos_c = table_1 : (s = 1) | (s = 2);
    pos_c = table_2 : (s = 2) | (s = 3);
    pos_c = table_3 : (s = 3) | (s = 4);
    pos_c = table_4 : (s = 4) | (s = 5);
    pos_c = table_5 : (s = 5) | (s = 6);
    TRUE : FALSE;
esac;

-- (Definições análogas para occupied_by_b e occupied_by_d)
```

Esta representação captura elegantemente o conceito de que um bloco largo pode ocupar vários espaços de coordenadas simultaneamente.

5.3. Predicado de Vacância Horizontal (`is_free`)

Com a capacidade de determinar o que cada bloco ocupa, podemos agora definir o que significa para um slot da mesa estar livre. Um slot

`s` está livre se não estiver ocupado por *nenhum* dos blocos no mundo.

Snippet de código
```
  -- Um slot 's' na mesa está livre se não estiver ocupado por nenhum bloco.
   is_free(s) := !(occupied_by_a(s) | occupied_by_b(s) | occupied_by_c(s) |
occupied_by_d(s));
```

Esta única linha de código implementa um raciocínio espacial complexo, mantendo o estado do sistema enxuto e consistente, conforme recomendado pelo princípio do conhecimento derivado.

---

6. Definição da Dinâmica do Sistema: A Física do Mundo

A dinâmica do sistema descreve como o mundo evolui de um estado para o outro. Em NuSMV, isso é definido através dos blocos `INIT`, `ASSIGN` e `TRANS`.

6.1. Definição do Estado Inicial (`INIT`)

A palavra-chave `INIT` é usada para restringir o estado inicial do sistema. Devemos traduzir a configuração inicial do problema de planejamento para uma atribuição de valores iniciais às nossas variáveis de estado.

Exemplo: Traduzindo a "Situação 3" do cenário

O estado inicial da Situação 3 mostra uma única pilha `d-b-a-c` na coordenada 0.

`EstadoInicial = [pos(d, table(0)), pos(b, on(d)), pos(a, on(b)), pos(c, on(a)), clear(c)]`.

A tradução para NuSMV é direta:

Snippet de código
```
ASSIGN
   init(pos_d) := table_0;
   init(pos_b) := on_d;
   init(pos_a) := on_b;
   init(pos_c) := on_a;
   init(action) := none; -- O sistema começa sem nenhuma ação em andamento.
```

6.2. Lógica de Transição (`ASSIGN` e `TRANS`)

A lógica de transição é o coração do modelo, definindo as regras que governam a evolução do estado. Ela é dividida em duas partes:

1. **`ASSIGN` para `next(...)`:** Define o *efeito* de uma ação, ou seja, como o valor de uma variável de estado muda no próximo passo de tempo com base na ação atual. Isto corresponde à Add-List e Delete-List do STRIPS.
2. **`TRANS`:** Define as *pré-condições* de uma ação. É uma restrição global sobre as transições, atuando como um filtro ou guarda que permite apenas movimentos fisicamente válidos e logicamente possíveis.

Definindo os Efeitos com `ASSIGN`: Para cada variável de posição, definimos seu próximo valor com base na `action` atual.

Snippet de código
```
  -- Atribuição do próximo estado para cada variável de posição
   next(pos_a) := case
      action = move_a_to_b : on_b;
      action = move_a_to_c : on_c;
      action = move_a_to_d : on_d;
      action = move_a_to_table_0 : table_0;
      -- (continua para todas as outras ações envolvendo 'a')
      TRUE : pos_a; -- Se nenhuma ação mover 'a', sua posição permanece a mesma.
   esac;

   -- (Blocos 'ASSIGN' análogos para next(pos_b), next(pos_c), e next(pos_d))
```

6.3. Codificando as Pré-condições como Guardas de Transição (`TRANS`)

O bloco `TRANS` é uma única fórmula booleana que deve ser verdadeira para qualquer transição de estado. Usamos uma estrutura `case...esac` para listar as pré-condições para cada ação possível. Se a ação escolhida não deterministicamente pelo modelo não satisfizer suas pré-condições, a transição é bloqueada, e o modelo não explorará aquele caminho.

Snippet de código
```
TRANS
   case
      -- Pré-condições para mover um bloco X para um bloco Y
      (action = move_a_to_b) :
         clear_a &         -- Mobilidade do Bloco: O bloco a ser movido deve estar livre[cite:
180].
         clear_b &         -- Acessibilidade do Alvo: O alvo deve estar livre[cite: 187].
         (size_a <= size_b) & -- Estabilidade: Bloco movido não pode ser maior que o
suporte[cite: 192, 439].
         (pos_a != on_b);     -- Evitar ações redundantes (não mover para onde já está).

      (action = move_d_to_a) :
```

```
            clear_d &
            clear_a &
            (size_d <= size_a) & -- Esta condição falhará (3 <= 1 é falso), podando este ramo da busca.
            (pos_d != on_a);

        -- Pré-condições para mover um bloco X para a mesa na coordenada Y
        (action = move_c_to_table_2) :
            clear_c &                    -- Mobilidade do Bloco[cite: 180].
            is_free(2) & is_free(3) &    -- Disponibilidade de Espaço: Todos os slots necessários devem estar livres[cite: 201].
            (pos_c != table_2);          -- Evitar ações redundantes.

        -- (Continua para todas as outras ações)

        (action = none) : TRUE; -- A ação nula é sempre permitida.

        -- Nenhuma outra transição é permitida. Isso previne o modelo de chegar a um estado sem saída.
        TRUE : FALSE;
    esac;
```

Este bloco implementa rigorosamente as "leis da física" do nosso domínio, garantindo que o planejador só explore sequências de ações válidas.

---

7. Geração e Validação do Plano

Com o modelo e sua dinâmica totalmente definidos, o passo final é especificar o objetivo e solicitar ao NuSMV que encontre a solução.

7.1. Definição do Estado Objetivo (`goal`)

Assim como o estado inicial, o estado objetivo é uma configuração específica das posições dos blocos. Usamos `DEFINE` para criar uma macro booleana que será verdadeira apenas quando o sistema estiver no estado objetivo.

Exemplo: Objetivo da "Situação 3"

O objetivo é uma pilha `b-a-c-d` na coordenada 2.

`GoalState = [pos(b, table(2)), pos(a, on(b)), pos(c, on(a)), pos(d, on(c))]` .

A tradução para NuSMV:

Snippet de código
```
DEFINE
    goal := (pos_b = table_2) & (pos_a = on_b) & (pos_c = on_a) & (pos_d = on_c);
```

7.2. A Especificação em Lógica Temporal (CTL)

Agora, aplicamos a estratégia do contraexemplo. Afirmamos que o objetivo definido é inalcançável a partir de qualquer estado inicial válido.

Snippet de código
```
-- Especificação CTL para encontrar o plano
-- A propriedade afirma que "Não existe um caminho futuro onde o objetivo é alcançado".
CTLSPEC
    !EF(goal)
```

Esta é a única especificação necessária. Ela instrui o NuSMV a buscar por qualquer caminho que viole essa afirmação, ou seja, qualquer caminho que *atinja* o `goal`.

7.3. Execução e Interpretação da Saída

Para executar o modelo, salvamos o código em um arquivo (ex: `mundo_blocos.smv`) e o executamos com o NuSMV. Se um plano existir, a saída será algo como:

```
-- specification !EF(...) is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  pos_d = table_0
  pos_b = on_d
  pos_a = on_b
  pos_c = on_a
  action = none
-> State: 1.2 <-
  action = move_c_to_table_4
-> State: 1.3 <-
  pos_c = table_4
  action = move_a_to_table_1
-> State: 1.4 <-
  pos_a = table_1
  action = move_b_to_table_2
-> State: 1.5 <-
  pos_b = table_2
  action = move_a_to_b
-> State: 1.6 <-
  pos_a = on_b
  action = move_c_to_a
```

```
-> State: 1.7 <-
  pos_c = on_a
  action = move_d_to_c
-> State: 1.8 <-
  pos_d = on_c
  -- O estado objetivo foi alcançado
```

Interpretação: O plano é a sequência de valores da variável `action` que leva à mudança de estado. Lendo a trilha, o plano é:

1. `move_c_to_table_4`
2. `move_a_to_table_1`
3. `move_b_to_table_2`
4. `move_a_to_b`
5. `move_c_to_a`
6. `move_d_to_c`

Esta sequência é a solução garantida para o problema de planejamento.

---

8. Exemplo de Código Completo para um Cenário

8.1. Modelando a "Situação 3"

A "Situação 3" é um excelente caso de teste, pois requer a desmontagem completa de uma torre e a reconstrução em uma nova ordem e local.

- **Estado Inicial:** `[pos(d, table(0)), pos(b, on(d)), pos(a, on(b)), pos(c, on(a)), clear(c)]`.
- **Estado Objetivo:** `[pos(b, table(2)), pos(a, on(b)), pos(c, on(a)), pos(d, on(c))]`.

8.2. Código-Fonte `.smv` Consolidado

A seguir, um esqueleto do código completo para este cenário. *(Nota: A lista completa de ações e `case` statements é extensa e foi abreviada para clareza).*

Snippet de código
```
MODULE main

-- Seção 1: Domínio Estático
DEFINE
   size_a := 1; size_b := 1; size_c := 2; size_d := 2; -- Tamanhos do cenário

-- Seção 2: Estado Dinâmico
```

```
VAR
   pos_a: { on_b, on_c, on_d, table_0, table_1, table_2, table_3, table_4, table_5, table_6 };
   pos_b: { on_a, on_c, on_d, table_0, table_1, table_2, table_3, table_4, table_5, table_6 };
   pos_c: { on_a, on_b, on_d, table_0, table_1, table_2, table_3, table_4, table_5, table_6 };
   pos_d: { on_a, on_b, on_c, table_0, table_1, table_2, table_3, table_4, table_5, table_6 };
   action: { move_a_to_b, move_c_to_table_4, ..., none }; -- Lista abreviada

-- Seção 3: Conhecimento Derivado
DEFINE
   clear_a := (pos_b != on_a) & (pos_c != on_a) & (pos_d != on_a);
   -- (clear_b, clear_c, clear_d)
   -- (occupied_by_a(s), occupied_by_b(s), etc.)
   -- is_free(s) := ...;

-- Seção 4: Lógica de Transição
ASSIGN
   -- Estado Inicial (Situação 3)
   init(pos_d) := table_0; init(pos_b) := on_d; init(pos_a) := on_b; init(pos_c) := on_a;
init(action) := none;

   -- Próximo Estado
   next(pos_a) := case (action = move_a_to_b) : on_b; ... TRUE : pos_a; esac;
   -- (next para pos_b, pos_c, pos_d)

TRANS
   case
      (action = move_c_to_a) : clear_c & clear_a & (size_c <= size_a);
      -- (... todas as outras pré-condições de ação ...)
      (action = none) : TRUE;
      TRUE : FALSE;
   esac;

-- Seção 5: Especificação
DEFINE
   goal := (pos_b = table_2) & (pos_a = on_b) & (pos_c = on_a) & (pos_d = on_c);

CTLSPEC !EF(goal)
```

---

9. Análise de Desempenho e Complexidade

9.1. Impacto das Restrições na Complexidade da Busca

À primeira vista, o modelo estendido parece computacionalmente mais caro do que o clássico, pois as pré-condições envolvem aritmética e um raciocínio mais complexo do que simples consultas de pertinência em uma lista. No entanto, essa análise é superficial.

A verdadeira medida da eficiência de um planejador é o tamanho do espaço de busca que ele precisa explorar. Nesse aspecto, o nosso modelo oferece uma vantagem massiva. As fortes restrições físicas (estabilidade, ocupação espacial) atuam como uma heurística de poda extremamente eficaz e específica do domínio.

O planejador clássico, cego a essas restrições, desperdiçaria tempo explorando inúmeros ramos da árvore de busca que são sintaticamente válidos, mas fisicamente impossíveis. Por exemplo, ele poderia gerar um plano longo que começa com a ação

`move(d, on(a))`. O modelo estendido elimina esse ramo em sua raiz. No momento em que essa ação é considerada, a pré-condição de estabilidade (

`size_d <= size_a`) falha imediatamente, e todo o subespaço de busca que se seguiria a essa ação inválida é podado.

9.2. Comparação com a Abordagem Clássica

O custo computacional de realizar a verificação de tamanho é trivial em comparação com o custo de explorar a vasta sub-árvore de planos que se originaria de um movimento inválido. Portanto, há uma compensação: um custo marginalmente maior por expansão de nó em troca de um espaço de busca efetivo drasticamente menor. Para domínios de planejamento regidos por leis físicas, incorporar essas leis como restrições fortes nas definições dos operadores é frequentemente a estratégia mais eficaz para alcançar a eficiência geral.

---

10. Conclusão e Trabalhos Futuros

10.1. Sumário da Solução e Suas Vantagens

Este relatório detalhou o projeto e a implementação de um modelo em NuSMV para resolver um problema complexo do Mundo dos Blocos com tamanhos heterogêneos e restrições espaciais. Ao traduzir sistematicamente uma representação de conhecimento rica, inspirada em Prolog, para um sistema de transição de estados formal, criamos uma solução robusta. O núcleo da inteligência do modelo reside na codificação das leis físicas de estabilidade e disponibilidade de espaço como pré-condições invioláveis na lógica de transição.

A abordagem de "Planejamento como Verificação de Modelo" oferece vantagens significativas:

- **Corretude Garantida:** O plano gerado (o contraexemplo) é, por construção, uma solução correta que satisfaz todas as restrições do modelo.
- **Automação Completa:** O processo de busca pelo plano é totalmente automatizado pelo verificador de modelos.
- **Expressividade:** A lógica temporal e a estrutura do modelo permitem representar domínios complexos e com restrições ricas.

O sistema resultante não é apenas uma solução para um quebra-cabeça, mas uma demonstração de uma abordagem baseada em princípios para modelar domínios de planejamento fisicamente fundamentados.

10.2. Potenciais Extensões do Modelo

O modelo apresentado é uma base sólida que pode ser estendida em várias direções para aumentar seu realismo e poder de resolução. Com base nas sugestões da literatura fornecida, os trabalhos futuros poderiam incluir:

- **Espaço Tridimensional:** A extensão mais natural seria introduzir uma terceira dimensão, atribuindo altura aos blocos e rastreando uma coordenada Z. Isso exigiria um raciocínio mais complexo sobre a folga vertical e o acesso do braço do robô.
- **Rotação de Blocos:** Poderíamos definir blocos com largura e profundidade (ex: `dimensions(c, 2, 1)`) e introduzir uma nova ação `rotate(Block)` que troca essas dimensões. Isso adicionaria uma camada estratégica ao planejamento, onde o sistema teria que decidir não apenas onde colocar um bloco, mas também em qual orientação.
- **Planejamento de Ordem Parcial:** O planejador atual produz um plano totalmente ordenado. Para problemas com sub-objetivos independentes (ex: construir duas torres em lados opostos da mesa), um planejador de ordem parcial seria mais eficiente, produzindo um plano mais flexível onde a ordem entre ações independentes não é especificada.

Essas extensões destacam a flexibilidade e a escalabilidade da abordagem de verificação de modelos para resolver classes cada vez mais complexas de problemas de planejamento.
