Proposta de Linguagem de Representação

Nomes:
Navvab Portela Salehi
Laura Couteiro Monteiro
Gabriel de Albuquerque Façanha
André Luiz de Souza Cruz
Filipe Carlos Olimpio
Abraão Nazareth Buzaglo
Oliver de Souza Nunes
Samuel Henrique Auzier Corrêa
Adriano Albert Lima de Oliveira

Representação de Blocos
Prolog
```
bloco(ID, Largura, Altura).
```

Posição na Mesa
Prolog
```
posicao(ID, X, Y).
```

Espaço na Grade
Prolog
```
ocupado(X, Y).
livre(X, Y).
```

Condições de Vacância
Prolog
```
espaco_livre(X, Y, Largura, Altura).
```

Estabilidade
Prolog
```
estavel_sobre(ID1, ID2) :-
        centro_de_massa(ID1, CM1),
        cobertura(ID2, X2, L2)
        CM1 >= X2,
        CM1 =< X2 + L2.
```

Centro de Massa
Prolog
```
centro_de_massa(ID, CM) :-
        posicao(ID, X, _),
        bloco(ID, L, _),
```

CM is X + L / 2.

Elemento	Figure 17.1	Proposta Atual
Bloco	`block(a)`	`bloco(a, 2, 1)`
Posição	`on(a,b)`	`posicao(a, 3, 2)`
Espaço livre	`clear(a)`	`livre(X,Y)` por célula
Mesa	`on_table(a)`	`posicao(a, X, 0)`
Estabilidade	Implícita	`estavel_sobre(ID1, ID2)` com centro de massa
Dimensão	Não representada	`Largura`, `Altura` explícitas

Modificação do Planner
Generalização de gols
goal(State) :- member(on(X,Y), State), objetivo(on(X,Y)).

Generalização de Ações
```
acao(move(B,De,Para), S1, S2) :-
    member(on(B,De), S1),
    clear(B, S1),
    clear(Para, S1),
    remove(on(B,De), S1, Temp),
    add(on(B,Para), Temp, S2).
```

Uso de Meta-Interpretação
```
plano(S, S, []).
plano(S1, Sgoal, [A|Resto]) :-
    acao(A, S1, S2),
    plano(S2, Sgoal, Resto).
```

Geração Manual de Planos
i1 -> i2
[move(d,c,b), move(c,b,a), move(b,a,mesa), move(a,mesa,c)]

i2 -> i2(a)
[move(a,c,mesa), move(c,b,a), move(b,a,c)]

i2 -> i2(b)
[move(a,c,mesa), move(c,b,a), move(b,a,d)]

i2 - > i2(b) (repetido)
[move(a,c,mesa), move(c,b,a), move(b,a,d)]

Requisitos para o Planejador

a) Reconhecimento de Padrões

- Identificar estruturas como torres, blocos isolados, blocos empilhados instáveis.

b) Avaliação de Estabilidade

- Verificar centro de massa e vacância no topo.

c) Generalização de Objetivos

- Permitir metas como `on(X,Y)` com variáveis.

d) Controle de Espaço

- Verificar se há espaço lateral e vertical para movimentar blocos grandes.

e) Heurísticas

- Priorizar ações que liberem blocos obstruídos ou instáveis.

Tabela consolidada dos conceitos abordados

Elemento	Representação Espacial & Física do Mundo	Proposta (predicado)	Observações e justificativa
Bloco	block(a)	bloco(ID, Largura, Altura)	Dimensões explícitas para lidar com blocos não-uniformes

Posição	on(a,b) / on_table(a)	posicao(ID, X, Y)	Posição em grade com eixo Y=0 na mesa; permite medir espaço lateral/vertical
Espaço ocupado	clear(a)	ocupado(X,Y) / livre(X,Y)	Granularidade por célula para checar sobreposição lateral e vertical
Vacância para inserção	implícita	espaco_livre(X,Y,L,A)	Verifica todos os slots necessários horizontal e verticalmente
Estabilidade	assumida implicitamente	estavel_sobre(ID1, ID2) com centro_de_massa(ID,CM)	Teste por centro de massa para bloco maior sobre menor
Centro de massa	não representado	centro_de_massa(ID, CM) := X + L/2	Necessário para decidir estabilidade em empilhamentos heterogêneos
Ação (STRIPS)	move(B,Pi,Pj) (fixo)	acao(move(B,Pi,Pj), S1, S2) paramétrica	Ação aceita variáveis e checa clear e espaco_livre
Planner	plano recursivo fixo	plano(S1,Sgoal,ListaAcoes) com metas variáveis	Meta-interpretação que aplica ações com variáveis sobre goals
Heurística / prioridades	não especificada	priorizar liberar blocos obstrutores; checar estabilidade	Necessário para eficiência e evitar movimentos instáveis
Model checking / NuSMV	não tratado	tamanhos como constantes; estados para ocupação	Dimensões modeladas como constantes de compilação para eficiência

---

✅ **Etapa 1 – Linguagem de Representação**

Foi criada uma linguagem baseada em uma grade bidimensional, onde cada célula representa uma unidade mínima de espaço. Os blocos são definidos com predicados que indicam suas dimensões (largura e altura), posição na mesa, ocupação de espaço, vacância horizontal e vertical, além de critérios físicos de estabilidade. A estabilidade é verificada por meio do centro de massa do bloco superior em relação à base do bloco inferior.

✅ Etapa 2 – Modificação do Planejador

O planejador original foi adaptado para lidar com variáveis em metas e ações. As ações foram generalizadas para aceitar qualquer bloco e qualquer posição, desde que respeitem as condições de espaço livre e estabilidade. A busca por planos foi implementada de forma recursiva, permitindo que o sistema encontre sequências de ações que levem de um estado inicial a um estado desejado.

✅ Etapa 3 – Geração Manual de Planos

Foram geradas manualmente sequências de ações para transformar estados iniciais em estados finais específicos. Cada plano foi construído com base na linguagem proposta, respeitando as regras de movimentação, vacância e estabilidade. As ações envolvem mover blocos entre posições, liberando espaço e reorganizando a estrutura conforme o objetivo.

✅ Etapa 4 – Análise das Situações 2 e 3

Foi feita uma análise dos requisitos necessários para que o planejador consiga gerar planos válidos para situações mais complexas. Isso inclui reconhecer padrões de empilhamento, avaliar estabilidade física, controlar espaço lateral e vertical, generalizar metas e aplicar heurísticas que priorizem ações estratégicas, como liberar blocos obstruídos ou instáveis.

✅ Etapa 5 – Consolidação dos Conceitos

Todos os elementos da linguagem e do planejador foram organizados em uma tabela comparativa, destacando como cada conceito foi tratado e ampliado. A proposta permite representar blocos com diferentes tamanhos, controlar o espaço ocupado, verificar estabilidade e gerar planos de ação eficientes e seguros.

Tipo de Restrição	Regra em Linguagem Natural	Implementação NuSMV (Ex.: move(C, A))	Implementação NuSMV (Ex.: move(C, table(2)))
Mobility	O bloco móvel **C** só pode ser movido se o seu topo estiver livre (ninguém em cima dele).	`next(pos[C]) = A -> clear[C]`	`next(pos[C]) = table(2) -> clear[C]`
Target Accessibility	O bloco alvo **A** só pode receber outro bloco se o seu topo estiver livre.	`next(pos[C]) = A -> clear[A]`	`next(pos[C]) = table(2) -> table_free(2)`
Stability	O bloco móvel **C** não pode ser mais largo que o bloco alvo **A** (senão não há suporte estável).	`next(pos[C]) = A -> size[C] <= size[A]`	`next(pos[C]) = table(2) -> TRUE` (a mesa sempre suporta)
Spatial Occupancy	Se um bloco ocupa mais de uma célula, todos os slots da mesa entre *i* e *i + size(C) - 1* devem estar livres.	*(não aplicável em cima de outro bloco)*	`next(pos[C]) = table(i) -> & !occupied[j] for j in {i..i+size(C)-1}`
Logical Validity	Um bloco não pode ser colocado sobre si mesmo.	`next(pos[C]) != C`	`next(pos[C]) = table(i)` (já válido por construção)

```
-- blocks_planning.smv
-- Modelo NuSMV para planejamento de blocos (3 blocos: a,b,c; mesa com 4 slots: 1..4)
-- Regras: mobility, target accessibility, stability, spatial occupancy, logical validity

MODULE main

VAR
  -- posição atual de cada bloco: pode estar sobre outro bloco (a,b,c) ou sobre a mesa em slots 1..4
  on_a : {b, c, 1, 2, 3, 4};
  on_b : {a, c, 1, 2, 3, 4};
  on_c : {a, b, 1, 2, 3, 4};

  -- ação não determinística a cada passo (nós permitimos muitas ações possíveis)
  move : { none,
           move_a_b, move_a_c, move_a_1, move_a_2, move_a_3, move_a_4,
           move_b_a, move_b_c, move_b_1, move_b_2, move_b_3, move_b_4,
           move_c_a, move_c_b, move_c_1, move_c_2, move_c_3, move_c_4 };

-- SIZES (constantes definidas via DEFINE)
DEFINE
  size_a := 2;   -- largura do bloco a
  size_b := 1;   -- largura do bloco b
  size_c := 3;   -- largura do bloco c

-- Define se cada célula da mesa está livre (não ocupada por nenhum bloco, considerando blocos que ocupam múltiplas células)
```

```
-- Observação: usamos igualdade simbólica com tokens 1..4; cada "on_X = k" indica que o bloco começa na célula k.
-- Para blocos que ocupam mais de 1 slot (size > 1) verificamos os possíveis *starts* que causam ocupação de cada slot.

  -- occupied_1 true se algum bloco cobre a célula 1
  occupied_1 :=
     (on_a = 1) |                    -- a inicia em 1 (ocupa 1 e 2)
     (on_b = 1) |                    -- b inicia em 1
     (on_c = 1);                     -- c inicia em 1 (ocupa 1,2,3)

  -- occupied_2
  occupied_2 :=
     (on_a = 1) | (on_a = 2) |         -- a inicia em 1 (ocupa 2) ou em 2 (ocupa 2,3)
     (on_b = 2) |
     (on_c = 1) | (on_c = 2);          -- c inicia em 1 or 2 (if in 2 occupies 2,3,4)

  -- occupied_3
  occupied_3 :=
     (on_a = 2) | (on_a = 3) |         -- a can occupy 3 if started at 2? (a size 2 -> starts 2 only gives 2 and 3)
     (on_b = 3) |
     (on_c = 1) | (on_c = 2);          -- c occupies 3 if start 1 or 2

  -- occupied_4
  occupied_4 :=
     (on_a = 3) |                     -- a start 3 would occupy 3 and 4 (but start 3 is allowed for a size 2)
     (on_b = 4) |
     (on_c = 2);                      -- c start 2 occupies 2,3,4

-- "clear" significa topo livre (para blocos) ou célula livre (para mesa)
  clear_a := (on_b != a) & (on_c != a);
  clear_b := (on_a != b) & (on_c != b);
  clear_c := (on_a != c) & (on_b != c);

  clear_1 := !occupied_1;
  clear_2 := !occupied_2;
  clear_3 := !occupied_3;
  clear_4 := !occupied_4;

-- utilitários: se um bloco X cabe começando na célula k da mesa (para evitar overflow)
-- (precomputado com base nos tamanhos dados)
  fits_a_1 := TRUE;
  fits_a_2 := TRUE;
  fits_a_3 := TRUE;
  fits_a_4 := FALSE;  -- size_a = 2 não cabe iniciando em 4 (precisa de 2 slots: 4 e 5 inexistente)
```

```
  fits_b_1 := TRUE;   -- size_b = 1 cabe em qualquer slot
  fits_b_2 := TRUE;
  fits_b_3 := TRUE;
  fits_b_4 := TRUE;

  fits_c_1 := TRUE;   -- size_c = 3 cabe iniciando em 1 (1,2,3)
  fits_c_2 := TRUE;   -- e em 2 (2,3,4)
  fits_c_3 := FALSE;  -- não cabe iniciando em 3 (3,4,5)
  fits_c_4 := FALSE;

-- GOAL exemplo (você pode alterar)
  goal := (on_a = c) & (on_b = 3) & (on_c = b);

-- Estado inicial
INIT
  on_a = 1 &
  on_b = 3 &
  on_c = a;

-- TRANSições para on_a (cada ação só altera o bloco envolvido; outras variáveis
permanecem)
TRANS
  next(on_a) =
    case
      -- Move A para outro bloco (mobility + target accessibility + stability + logical validity)
      move = move_a_b
        & clear_a              -- topo de A livre (mobility)
        & clear_b              -- topo de B livre (target accessibility)
        & size_a <= size_b       -- estabilidade: a não pode ser maior que b
        & on_a != b             -- validade lógica: não pode colocar sobre si mesmo (redundante
aqui)
        : b;

      move = move_a_c
        & clear_a
        & clear_c
        & size_a <= size_c
        & on_a != c
        : c;

      -- Move A para mesa: checar que cabe e que todos os slots necessários estejam livres
      move = move_a_1
        & fits_a_1
        & clear_a
        & clear_1
        & ( (size_a = 1) | (clear_2) )   -- se size_a>1, checar slot adicional(s). Para size_a=2:
precisa clear_2
```

```
        : 1;

      move = move_a_2
        & fits_a_2
        & clear_a
        & clear_2
        & ( (size_a = 1) | (clear_3) )
        : 2;

      move = move_a_3
        & fits_a_3
        & clear_a
        & clear_3
        & ( (size_a = 1) | (clear_4) )
        : 3;

      move = move_a_4
        & fits_a_4
        & clear_a
        & clear_4
        : 4;

      TRUE: on_a;
    esac;

TRANS
  next(on_b) =
    case
      move = move_b_a
        & clear_b
        & clear_a
        & size_b <= size_a
        & on_b != a
        : a;

      move = move_b_c
        & clear_b
        & clear_c
        & size_b <= size_c
        & on_b != c
        : c;

      move = move_b_1
        & fits_b_1
        & clear_b
        & clear_1
        : 1;
```

```
      move = move_b_2
        & fits_b_2
        & clear_b
        & clear_2
        : 2;

      move = move_b_3
        & fits_b_3
        & clear_b
        & clear_3
        : 3;

      move = move_b_4
        & fits_b_4
        & clear_b
        & clear_4
        : 4;

      TRUE: on_b;
    esac;

TRANS
  next(on_c) =
    case
      move = move_c_a
        & clear_c
        & clear_a
        & size_c <= size_a
        & on_c != a
        : a;

      move = move_c_b
        & clear_c
        & clear_b
        & size_c <= size_b
        & on_c != b
        : b;

      move = move_c_1
        & fits_c_1
        & clear_c
        & clear_1 & clear_2 & clear_3   -- c ocupa 3 slots: exige 1,2,3 livres
        : 1;

      move = move_c_2
        & fits_c_2
        & clear_c
        & clear_2 & clear_3 & clear_4
```

```
      : 2;

    move = move_c_3
      & fits_c_3
      & clear_c
      & clear_3 & clear_4          -- (fits_c_3 é FALSE, então este caso será desativado)
      : 3;

    move = move_c_4
      & fits_c_4
      & clear_c
      & clear_4
      : 4;

    TRUE: on_c;
  esac;

-- INVARIANTES / SPECs
-- 1) Mobilidade: um bloco só pode mover se seu topo estiver livre
--    (Expressamos como invariantes implicando que quando uma ação que move X é escolhida, clear_X é requerida;
--     as guards já garantem isso. Vamos duplicar como INVAR SPEC para reforçar)
INVARSPEC
  ( move = move_a_b | move = move_a_c | move = move_a_1 | move = move_a_2 | move = move_a_3 | move = move_a_4
    ) -> clear_a;

INVARSPEC
  ( move = move_b_a | move = move_b_c | move = move_b_1 | move = move_b_2 | move = move_b_3 | move = move_b_4
    ) -> clear_b;

INVARSPEC
  ( move = move_c_a | move = move_c_b | move = move_c_1 | move = move_c_2 | move = move_c_3 | move = move_c_4
    ) -> clear_c;


-- 2) Estabilidade: não colocar bloco maior sobre menor
INVARSPEC
  ( move = move_a_b ) -> (size_a <= size_b);

INVARSPEC
  ( move = move_a_c ) -> (size_a <= size_c);

INVARSPEC
  ( move = move_b_a ) -> (size_b <= size_a);
```

```
INVARSPEC
  ( move = move_b_c ) -> (size_b <= size_c);

INVARSPEC
  ( move = move_c_a ) -> (size_c <= size_a);

INVARSPEC
  ( move = move_c_b ) -> (size_c <= size_b);


-- 3) Validade lógica: não colocar um bloco sobre si mesmo
INVARSPEC (move = move_a_b) -> (b != a);
INVARSPEC (move = move_a_c) -> (c != a);
INVARSPEC (move = move_b_a) -> (a != b);
INVARSPEC (move = move_b_c) -> (c != b);
INVARSPEC (move = move_c_a) -> (a != c);
INVARSPEC (move = move_c_b) -> (b != c);

-- 4) Ocupação espacial: se um bloco começar na célula i e tiver tamanho >1, os slots
necessários têm de estar livres
--    (As guards nos TRANS já fazem isso; aqui só colocamos uma checagem global
adicional)
INVARSPEC ( on_a = 1 ) -> ( !occupied_1 & !occupied_2 );
INVARSPEC ( on_a = 2 ) -> ( !occupied_2 & !occupied_3 );
INVARSPEC ( on_a = 3 ) -> ( !occupied_3 & !occupied_4 );

INVARSPEC ( on_b = 1 ) -> ( !occupied_1 );
INVARSPEC ( on_b = 2 ) -> ( !occupied_2 );
INVARSPEC ( on_b = 3 ) -> ( !occupied_3 );
INVARSPEC ( on_b = 4 ) -> ( !occupied_4 );

INVARSPEC ( on_c = 1 ) -> ( !occupied_1 & !occupied_2 & !occupied_3 );
INVARSPEC ( on_c = 2 ) -> ( !occupied_2 & !occupied_3 & !occupied_4 );

-- PROPRIEDADES PARA VERIFICAR
-- Queremos checar se o goal é alcançável: EF goal
CTLSPEC EF goal

-- Queremos garantir que nunca se tenta mover um bloco para uma posição onde ele não
cabe (ou seja, moves inválidos são impossíveis)
-- Isso já está embutido nas guards; contra-exemplo: checar que nunca next pos coloca a
em 4 se a não cabe = false.
CTLSPEC AG !( next(on_a) = 4 & !fits_a_4 )

-- Exemplo de propriedade de segurança: nunca duas peças ocupam o mesmo topo (i.e., a
e b não podem estar um sobre o outro mutuamente)
CTLSPEC AG !( (on_a = b) & (on_b = a) )
```