# Proposta de Linguagem de Representação

**Nomes:**

| | |
|---|---|
| Navvab Portela Salehi | navvab.salehi@icomp.ufam.edu.br |
| Laura Couteiro Monteiro | laura.couteiro@icomp.ufam.edu.br |
| Gabriel de Albuquerque Façanha | gabriel.facanha@icomp.ufam.edu.br |
| André Luiz de Souza Cruz | andre.cruz@icomp.ufam.edu.br |
| Filipe Carlos Olimpio | filipe.olimpio@icomp.ufam.edu.br |
| Abraão Nazareth Buzaglo | abraao.buzaglo@icomp.ufam.edu.br |
| Oliver de Souza Nunes: | oliver.nunes@icomp.ufam.edu.br |
| Samuel Henrique Auzier Corrêa | samuel.correa@icomp.ufam.edu.br |
| Adriano Albert Lima de Oliveira | adriano.oliveira@icomp.ufam.edu.br |


**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.

---

**Tabela Regras de Transição e Validação para Movimentação de Blocos (NuSMV)**

| **Tipo de Restrição** | **Destino** | **Regra em Linguagem Natural** | **Implementação NuSMV (Ex.: `move(C, A)`)** | **Implementação NuSMV (Ex.: `move(C, table(2))`)** |
|---|---|---|---|---|
| **Mobility** | Bloco móvel C | O bloco C só pode ser movido se seu topo estiver livre. | next(pos[C]) = A -> clear[C] | next(pos[C]) = table(2) -> clear[C] |
| **Target Accessibility** | Bloco alvo A | O bloco A só pode receber outro bloco se seu topo estiver livre. | next(pos[C]) = A -> clear[A] | next(pos[C]) = table(2) -> table_free(2) |
| **Stability** | Bloco alvo A | O bloco C não pode ser mais largo que o bloco A. | next(pos[C]) = A -> size[C] <= size[A] | `next(pos[C]) = table(2) -> TRUE` (a mesa sempre suporta) |
| **Spatial Occupancy** | Mesa (table(i)) | Todos os slots de i até 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** | Bloco C | Um bloco não pode ser colocado sobre si mesmo. | next(pos[C]) != C | (já válido por construção) |

**Código NuSMV:**

```
MODULE main

--  Declaração das variáveis de estado
VAR
  -- Posição atual do bloco A (pode estar sobre outro bloco ou sobre a mesa)
  on_a : {b, c, 1, 2, 3, 4};
  -- Posição atual do bloco B
  on_b : {a, c, 1, 2, 3, 4};
  -- Posição atual do bloco C
  on_c : {a, b, 1, 2, 3, 4};

  -- Ação de movimento a ser executada no próximo passo
  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 };

--  Definições auxiliares para regras e condições
DEFINE
  -- Tamanhos dos blocos (largura)
  size_a := 2;
  size_b := 1;
  size_c := 3;

   -- Verifica se cada célula da mesa está ocupada, considerando blocos que ocupam múltiplas células
  occupied_1 := (on_a = 1) | (on_b = 1) | (on_c = 1);
  occupied_2 := (on_a = 1) | (on_a = 2) | (on_b = 2) | (on_c = 1) | (on_c = 2);
  occupied_3 := (on_a = 2) | (on_a = 3) | (on_b = 3) | (on_c = 1) | (on_c = 2);
  occupied_4 := (on_a = 3) | (on_b = 4) | (on_c = 2);

  -- Verifica se o topo de cada bloco está livre (ninguém em cima dele)
  clear_a := (on_b != a) & (on_c != a);
  clear_b := (on_a != b) & (on_c != b);
  clear_c := (on_a != c) & (on_b != c);

  -- Verifica se cada célula da mesa está livre
  clear_1 := !occupied_1;
  clear_2 := !occupied_2;
  clear_3 := !occupied_3;
  clear_4 := !occupied_4;
```

```
  -- Verifica se o bloco cabe na célula da mesa (suporte físico)
  fits_a_1 := TRUE;
  fits_a_2 := TRUE;
  fits_a_3 := TRUE;
  fits_a_4 := FALSE;

  fits_b_1 := TRUE;
  fits_b_2 := TRUE;
  fits_b_3 := TRUE;
  fits_b_4 := TRUE;

  fits_c_1 := TRUE;
  fits_c_2 := TRUE;
  fits_c_3 := FALSE;
  fits_c_4 := FALSE;

  -- Objetivo do sistema: colocar o bloco A na célula 1
  goal := (on_a = 1);

--  Estado inicial do sistema
INIT
  on_a = b &  -- A está sobre B
  on_b = c &  -- B está sobre C
  on_c = 4;     -- C está sobre a célula 4 da mesa

--  Regras de transição para o bloco A
TRANS
  next(on_a) =
          case
          -- Movimentos válidos de A sobre outro bloco, respeitando restrições
          move = move_a_b & clear_a & clear_b & size_a <= size_b & on_a != b :
b;
          move = move_a_c & clear_a & clear_c & size_a <= size_c & on_a != c :
c;

          -- Movimentos válidos de A para a mesa, verificando espaço e suporte
          move = move_a_1 & clear_a & fits_a_1 & clear_1 & clear_2 : 1;
          move = move_a_2 & clear_a & fits_a_2 & clear_2 & clear_3 : 2;
          move = move_a_3 & clear_a & fits_a_3 & clear_3 & clear_4 : 3;
          move = move_a_4 & clear_a & fits_a_4 & clear_4 : 4;
```

```
        -- Caso nenhuma condição seja satisfeita, mantém a posição atual
        TRUE : on_a;
        esac;

-- Regras de transição para o bloco B
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 & clear_b & fits_b_1 & clear_1 : 1;
        move = move_b_2 & clear_b & fits_b_2 & clear_2 : 2;
        move = move_b_3 & clear_b & fits_b_3 & clear_3 : 3;
        move = move_b_4 & clear_b & fits_b_4 & clear_4 : 4;

        TRUE : on_b;
        esac;

--  Regras de transição para o bloco C
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 & clear_c & fits_c_1 & clear_1 & clear_2 & clear_3 :
1;
        move = move_c_2 & clear_c & fits_c_2 & clear_2 & clear_3 & clear_4 :
2;

        TRUE : on_c;
        esac;

--   Propriedade CTL: não é possível atingir o objetivo (usado para verificar
bloqueios)
CTLSPEC !EF goal;
```

-- Justiça: considera apenas execuções onde o objetivo é eventualmente alcançado
FAIRNESS goal;

-- Propriedades de segurança

-- Evita ciclos de empilhamento (ex.: A sobre B e B sobre A)
CTLSPEC AG !(on_a = b & on_b = a);

-- Impede que o bloco A seja colocado na célula 4 (por instabilidade ou regra de domínio)
CTLSPEC AG !(on_a = 4);

**Exemplo de Trace Gerado pelo Código**

| **Etapa** | **on_a** | **on_b** | **on_c** | **Ação Executada** |
|---|---|---|---|---|
| 1 | b | c | 4 | move_a_1 |
| 2 | 1 | c | 4 | –objetivo |