🟫 Planejamento no Mundo dos Blocos — Baseado em Bratko

Equipe

- **Disciplina**: Inteligência Artificial IEC034/ICC265 - 2025/1
- **Curso**: Ciência/Engenharia da Computação - Turmas CO01 e CB500
- **Integrantes**:
  - Antonio Mileysson França Bragança - 21850963
  - Jessica de Figueredo Colares - 22060036
  - Lucas Vinícius Gonçalves Gadelha - 22050517

1) Descrição do Problema em Linguagem Natural

O **problema** trata de um mundo de blocos com **larguras diferentes**, mas todos com **mesma altura**. Os blocos devem ser empilhados em posições fixas (chamadas de "places") ou sobre outros blocos, respeitando regras físicas de estabilidade:

- Um bloco pode ser colocado sobre dois blocos se sua **largura for menor ou igual à soma das larguras dos dois blocos mais 1**.
- Um bloco maior pode ser colocado centralizado sobre outro **se for no máximo 2 unidades maior**.
- O objetivo é montar uma estrutura final (meta) a partir de uma configuração inicial.

---

2) Conceitos de `clear` e outros

Estado do Mundo

Representado como uma **lista de relações** do tipo:

- `on(X, Y)`: o bloco X está sobre Y (Y pode ser um bloco ou lugar).
- `clear(X)`: o topo de X está livre.
- `block(X)`: X é um bloco.
- `place(P)`: P é uma posição base.
- `object(X)`: X é um bloco ou uma posição.
- `size(X, N)`: o tamanho (largura) do bloco X é N.

Exemplo:

```
% Estado inicial (exemplo)
state([clear(3), on(c,p([1,2])), on(b,6), on(a,4), on(d,p([a,b]))]).

% Estado final desejado
goal([clear(1), clear(2), clear(3), on(d,p([4,6])), on(c,p([d,d])), on(a,c),
on(b,c)]).
```

3) Representação lógica (sem assign/retract)

```
block(a). block(b). block(c). block(d).
place(1). place(2). place(3). place(4).

size(a, 2).
size(b, 2).
size(c, 1).
size(d, 1).

object(X) :- block(X).
object(X) :- place(X).
```

4) Restrições

4.1 Em linguagem natural

- Um bloco só pode ser movido se estiver livre (`clear`).
- O destino deve estar livre também.
- Um bloco não pode ser empilhado sobre ele mesmo.
- Um bloco pode ser colocado sobre dois blocos se seu tamanho for menor ou igual à soma dos tamanhos desses blocos mais 1.
- Um bloco pode ser centralizado sobre um só bloco se for até 2 unidades maior que ele.

4.2 Código das restrições

```
can(move(Block,p([Oi,Oj]),p([Bi,Bj])),
    [clear(Block),clear(Bi),clear(Bj),on(Block,p([Oi,Oj]))]) :-
    block(Block), block(Bi), block(Bj),
    Bi \== Block, Oi \== Bi, Oj \== Bj, Block \== Oi,
    size(Block,Sb), size(Bi,Si), size(Bj,Sj),
    SizeTo is Si + Sj,
    Sb =< SizeTo + 1.

can(move(Block,p([Oi,Oj]),p(B,B)),
    [clear(Block),clear(B),on(Block,p([Oi,Oj]))]) :-
    block(Block), block(B), B \== Block,
    Oi \== B, Oj \== B,
    size(Block,SizeB), size(B,Sb),
    SizeB =< Sb + 2.

can(move(Block,From,To),
    [clear(Block),clear(To),on(Block,From)]) :-
    block(Block), object(To), object(From),
    To \== Block, From \== To, Block \== From.
```

5) Planejador com Regressão de Metas e Means-Ends

```
plan(State, Goals, []) :-
    satisfied(State, Goals).

plan(State, Goals, Plan) :-
    conc(PrePlan, [Action], Plan),
    select(State, Goals, Goal),
    achieves(Action, Goal),
    can(Action, Condition),
    preserves(Action, Goals),
    regress(Goals, Action, RegressedGoals),
    plan(State, RegressedGoals, PrePlan).
```

Auxiliares:

```
preserves(Action, Goals) :-
    deletes(Action, DelList),
    \+ (member(G, DelList), member(G, Goals)).

regress(Goals, Action, RegressedGoals) :-
    adds(Action, AddList),
    delete_all(Goals, AddList, G1),
    can(Action, Cond),
    addnew(Cond, G1, RegressedGoals).
```

☑ Testando com exemplos do documento

Consulta de teste (baseada no livro):

```
?- plan(
    [clear(2), clear(4), clear(b), clear(c), on(a,1), on(b,3), on(c,a)],
    [on(a,b), on(b,c)],
    Plan).
```

Debug com trace:

```
?- trace, plan(InitialState, Goal, Plan).
```

🧠 Observações Finais

- Blocos possuem larguras diferentes e a estabilidade importa.
- O modelo segue **representação STRIPS-like**, com ações definidas por precondições, efeitos (`adds`, `deletes`) e regressão de metas.
- Não usamos `assign` ou `retract`, apenas **representação declarativa pura em Prolog**.
