UNIVERSIDADE FEDERAL DO AMAZONAS - UFAM
INSTITUTO DE COMPUTAÇÃO - ICOMP
DEPTO. DE ENGENHARIA DE SOFTWARE

MUNDO DOS BLOCOS VARIÁVEL EM MODEL CHECKING

FERNANDA SOUZA DE FREITAS

GABRIEL DA SILVA GLÓRIA

JULIO CESAR CERRATE CASTRO

LUANE DOS SANTOS LOPES

MUSKAAN RAMCHANDANI

PEDRO HENRIQUE BARROS MENDONÇA

RUAN COSTA DE MAGALHÃES

Manaus - AM

2025

FERNANDA SOUZA DE FREITAS

GABRIEL DA SILVA GLÓRIA

JULIO CESAR CERRATE CASTRO

LUANE DOS SANTOS LOPES

MUSKAAN RAMCHANDANI

PEDRO HENRIQUE BARROS MENDONÇA

RUAN COSTA DE MAGALHÃES

MUNDO DOS BLOCOS VARIÁVEL EM MODEL CHECKING

Trabalho Prático de Fundamentos de Inteligência Artificial-2025-2, como requisito para obtenção de nota parcial, ministrada pelo Prof.: Edjard Mota

Manaus - AM

2025

Sumário

Análise do Código SMV em Lógica de Primeira Ordem Usando Prolog

O código SMV modela um mundo dos blocos com tamanhos variáveis (a, b, c tamanho 1; d tamanho 2), em uma grade de posições 0 a 6 na mesa, com restrições de vacância, estabilidade (centro de massa), níveis de pilha e movimentos. Em FOL/Prolog, isso pode ser representado como um sistema de planejamento baseado em STRIPS (como no Capítulo 17 de Bratko), onde:

- Estados são conjuntos de fatos atômicos (predicados como on/2, position/2, level/2, clear/1, occupied/1).
- Ações são transições (move/3), com pré-condições (can/2), adições (adds/2) e deleções (deletes/2).
- Invariantes (INVAR) são axiomas que devem ser verdadeiros em todos os estados.
- Objetivo (goal) é um conjunto de fatos a serem satisfeitos.
- Verificação (CTLSPEC EF goal) corresponde a uma busca para encontrar um plano que alcance o objetivo (usando recursão em Prolog para simular busca em profundidade ou largura).

A análise foca em correção lógica, completude e restrições físicas (vacância horizontal/vertical, estabilidade).

1. **Representação de Estados (VAR e DEFINE em SMV → Predicados em Prolog)**

No SMV, variáveis como on_a, position_a, level_a representam o estado de cada bloco. Definições como clear_a e occupied_table_0 são fórmulas computadas.

Em Prolog (FOL), estados são listas de fatos [Fact1, Fact2, ...], onde:

- Predicados: on(Bloco, Sobre) (Sobre pode ser table ou outro bloco), position(Bloco, Pos), level(Bloco, Nível), clear(Bloco), occupied(Pos), size(Bloco, Tamanho).
- Justificativa: Isso estende a representação abstrata de Bratko (apenas on/2 e clear/1) para incluir posições espaciais (grade 0..6) e níveis (para estabilidade vertical), como exigido no Ponto 1 do assignment (comparando com Fig. 17.1: lugares abstratos viram posições numéricas; blocos ganham tamanho e centro de massa para vacância/estabilidade).

Código Prolog para representação:

```
% Predicados para fatos atômicos (FOL: ∀ Bloco, Pos, etc.)
on(Bloco, Sobre).  % Ex.: on(a, table) ou on(b, a)
position(Bloco, Pos).  % Pos in 0..6
level(Bloco, Nivel).  % Nivel in 0..10
clear(Bloco).  % Nada sobre o Bloco
occupied(Pos).  % Pos ocupada por algum bloco na mesa
size(Bloco, Tam).  % Tam fixo: size(a,1), etc.

% Estado é uma lista de fatos
estado(S) :- is_list(S).

% Exemplo de estado inicial (do INIT no SMV)
initial_state([on(a, table), position(a, 2), level(a, 0),
               on(b, d), position(b, 3), level(b, 2),
               on(c, table), position(c, 0), level(c, 0),
               on(d, a), position(d, 2), level(d, 1),
               size(a,1), size(b,1), size(c,1), size(d,2)]).

% Definições computadas (como DEFINE no SMV)
clear(Bloco, S) :- \+ member(on(_, Bloco), S).  % FOL: ¬∃ X on(X, Bloco)

occupied(Pos, S) :-
    member(on(Bloco, table), S),
    member(position(Bloco, BPos), S),
    member(size(Bloco, Tam), S),
    BPos =< Pos, Pos =< BPos + Tam - 1.

% Objetivo (goal no SMV)
goal_state([on(a, table), position(a, 0),
            on(c, a), position(c, 0),
            on(d, c), position(d, 0),
            on(b, d), position(b, 1)]).
```

Essa representação é completa para FOL, pois quantificadores (∀/∃) são implícitos em queries Prolog (ex.: clear(a, S) usa negação como falha para ¬∃). Comparado a Bratko: Foi adicionado position/2 e occupied/1 para vacância horizontal (grade onde menor bloco cabe em 1 espaço, como na dica do assignment); level/2 para vacância vertical e estabilidade (centro de massa calculado em ações).

2. **Ações e Transições (TRANS e ASSIGN em SMV → can/adds/deletes em Prolog)**

No SMV, move é uma variável enum com guards em case para pré-condições; next(...) atualiza estados.

Em Prolog, ações são move(Bloco, De, Para), onde De/Para pode ser table_Pos ou bloco. Use can(Acao, Estado) para pré-condições (clear origem/destino, vacância, estabilidade); adds/deletes para efeitos; apply(Acao, Estado, NovoEstado) para transição.

Justificativa: Foi modificado o planner de Fig. 17.6 (Ponto 2 do assignment): foi adicionado variáveis para goals (subset check) e ações (incluindo table_Pos para grade, estabilidade via centro de massa). Explicação da mudança: Em Bratko, can(move(X,Y,Z), [on(X,Y), clear(X), clear(Z)]); aqui, foi colado checks espaciais (vacant_horizontal/3, stable/3).

Código Prolog para ações:

```
% Ações possíveis (enum move no SMV)
action(move(Bloco, De, Para)) :- bloco(Bloco), pos_ou_bloco(De), pos_ou_bloco(Para), De \= Para.

bloco(B) :- member(B, [a,b,c,d]).
pos_ou_bloco(table_Pos) :- between(0,6,Pos); bloco(Pos).

% Pré-condições: can(Acao, Estado) - com vacância e estabilidade
can(move(Bloco, De, Para), S) :-
    member(on(Bloco, De), S),
    clear(Bloco, S),
    clear_destino(Para, S),
    vacant_horizontal(Bloco, Para, S),
```

```
    stable(Bloco, Para, S).

clear_destino(table_Pos, S) :- \+ occupied(Pos, S).  % Para table_Pos
clear_destino(DestBloco, S) :- clear(DestBloco, S).  % Para outro bloco

vacant_horizontal(Bloco, Para, S) :-
    member(size(Bloco, Tam), S),
    (Para = table_Pos ->  % Move para mesa
        forall(between(0, Tam-1, Offset), \+ occupied(Pos + Offset, S))
    ; Para = DestBloco ->  % Move para bloco (vacância horizontal implícita via position)
        member(position(DestBloco, DPos), S),
        member(size(DestBloco, DTam), S),
        member(position(Bloco, BPos), S),
        BPos >= DPos, BPos + Tam -1 =< DPos + DTam -1
    ).

stable(Bloco, Para, S) :-
    Para = DestBloco,  % Só para pilhas
    member(size(Bloco, BTam), S),
    member(size(DestBloco, DTam), S),
    BTam =< DTam,  % Bloco menor ou igual sobre maior
    member(position(DestBloco, DPos), S),
    Delta = DTam - BTam,
    Offset = case(Delta >= 0, Delta / 2, (Delta -1) / 2),  % Centro de massa (como no SMV)
    Offset >= 0, Offset + BTam -1 =< DTam -1.  % Estável se centro alinhado

% Adições e Deleções (adds/deletes no SMV via next)
adds(move(Bloco, De, Para), [on(Bloco, Para), clear(De), position(Bloco, NewPos),
level(Bloco, NewLevel)]) :-
    % Calcula NewPos e NewLevel baseado em Para (similar a next no SMV)
    (Para = table_Pos -> NewPos = Pos, NewLevel = 0
     ; Para = DestBloco -> member(position(DestBloco, DPos), S), NewPos = DPos + Offset,
member(level(DestBloco, DLevel), S), NewLevel = DLevel +1).
```

deletes(move(Bloco, De, Para), [on(Bloco, De), clear(Para), position(Bloco, OldPos), level(Bloco, OldLevel)]).

% Aplica ação: apply(Acao, Estado, NovoEstado)
apply(A, S, S1) :- deletes(A, D), subtract(S, D, S2), adds(A, A1), append(S2, A1, S1).

Em FOL, pré-condições são conjunções (& no SMV vira , em Prolog). A mudança para variáveis em goals (sessão 17.5 de Bratko) é manejada por subset(Goal, S) no planejador abaixo. Estabilidade é nova restrição: FOL ¬(size(Bloco) > size(Dest) ∧ ¬alinhado_centro), evitando instabilidade.

3. Invariantes e Restrições (INVAR em SMV → Axiomas em Prolog)

No SMV, INVAR previne colisões em níveis/posições e ciclos em pilhas. Em Prolog, invariantes são checados em estados válidos (valid_state(S)), usando negação para ¬.

Código Prolog:

```
% Invariante: Sem colisões em mesmo nível e posição sobreposta
valid_state(S) :-
    forall((bloco(B1), bloco(B2), B1 \= B2),
        (member(level(B1, L), S), member(level(B2, L), S) ->
         \+ (member(position(B1, P1), S), member(size(B1, S1), S),
             member(position(B2, P2), S), member(size(B2, S2), S),
             P1 =< P2 + S2 -1, P2 =< P1 + S1 -1))),
    % Sem ciclos (exemplos do INVAR)
    \+ (member(on(a,b), S), member(on(b,a), S)),
    \+ (member(on(a,c), S), member(on(c,a), S)),
    % ... (todos os ! ciclos do SMV)
```

```
% Limites de posição e nível
 forall(bloco(B), (member(position(B, P), S), member(size(B, Tam), S), P >= 0, P + Tam -1 =< 6)),
 forall(bloco(B), (member(level(B, N), S), N =< 10)).
```

Invariantes são fórmulas FOL universais (∀ B1,B2 ¬colisão(L,P)), garantindo consistência. Sem eles, planos poderiam violar física (ex.: bloco maior sobre menor sem estabilidade).

4. Planejador e Verificação (CTLSPEC em SMV → plano/3 em Prolog)

No SMV, EF goal verifica existência de plano via model checking. Em Prolog, plano/3 gera o plano recursivamente (como no diagrama da página 2: base se goal satisfeito, recursivo via ação aplicável).

Código Prolog completo para planejador:

```
% Planejador recursivo (plano(Estado, Goal, Plano))
plano(S, Goal, []) :- subset(Goal, S), valid_state(S).  % Base: goal satisfeito e estado válido
plano(S, Goal, [A|Plan]) :-
    action(A),
    can(A, S),
    apply(A, S, S1),
    valid_state(S1),  % Checa invariantes no novo estado
    plano(S1, Goal, Plan).
% Consulta exemplo: encontre plano do inicial ao goal
consulta_plan :- initial_state(Init), goal_state(Goal), plano(Init, Goal, Plan), write(Plan), nl.
```

Isso é a formulação recursiva do diagrama (plano de S1 a Sgoal via ações A1..Ai). Em FOL, é equivalente a $\exists$ Plano $\forall$ i (can(Ai, Si) $\wedge$ apply(Ai, Si, Si+1) $\wedge$ subset(Goal, Sfinal)). Para o SMV, EF goal é true se existe tal Plano.

Conclusão Geral da Análise: O código SMV é uma representação precisa do mundo dos blocos com restrições físicas, mas em Prolog/FOL, torna-se mais declarativo e fácil para planejamento regressivo.