Universidade Federal do Amazonas
Instituto de Computação - IComp

Mundo dos blocos variável em model checking

Estefany Licinha Mendes
Karen Juliana Baéz González
Luna Veiga Horta Braga
Rebeca Martins Xavier
Samuel Davi Chagas

Manaus
2025

Metodologia

Para encontrar a solução das três situações do problema do Mundo dos Blocos com tamanhos variáveis, foi utilizada a metodologia de verificação formal por Model Checking, aplicando os conceitos estudados em aula e as orientações contidas no material Planning as Model Checking. O processo teve início com a modelagem do domínio do problema na linguagem SMV (Symbolic Model Verification), compatível com o verificador NuSMV/nuXmv, que permite representar sistemas de estados finitos e verificar propriedades temporais.

Primeiramente, foi feita a abstração do mundo físico em um conjunto de variáveis simbólicas que representam as condições relevantes do problema: qual bloco está sobre qual suporte (support_X) e qual a posição do bloco na mesa (slot_X). Essa abstração permitiu representar todas as possíveis configurações do mundo de forma formal e manipulável. Em seguida, foram definidos predicados auxiliares — como clear_X (indicando que um bloco está livre), size_ok_X_on_Y (verificando estabilidade conforme tamanhos) e slot_free (para checar disponibilidade de espaço na mesa) — que servem como condições de transição e asseguram a consistência física do modelo (sem sobreposições ou blocos "flutuando").

A partir dessas definições, cada estado inicial (S_0) e cada estado objetivo (Sf_1–Sf_4, S_5, S_7) foram formalizados conforme as figuras apresentadas no enunciado. Com isso, o sistema foi descrito em termos de transições de estados, permitindo que o verificador explore automaticamente todas as combinações possíveis de ações válidas (como mover um bloco da mesa para outro bloco, ou para uma posição livre na mesa).

As propriedades a serem verificadas foram expressas em lógica temporal CTL (Computation Tree Logic). Para gerar os planos de ação, utilizou-se a forma negada da propriedade de alcançabilidade:

CTLSPEC !EF(goal)

Essa expressão questiona se "não é possível" alcançar o estado objetivo; quando o verificador retorna falso, ele exibe o contraexemplo, que é precisamente a sequência de estados (ou ações) que levam do estado inicial ao estado final — ou seja, o plano de solução. Esse procedimento foi aplicado para as três situações propostas:

1. De S_0 até qualquer um dos estados finais Sf_1–Sf_4;
2. De S_0 até o estado S_5;
3. De S_0 até o estado S_7.

Com base nesses resultados, o sistema foi capaz de gerar automaticamente os planos válidos que levam cada configuração inicial à sua meta, comprovando a correção das soluções por meio de um raciocínio formal e exaustivo. O método utilizado garante não apenas a validade dos resultados, mas também a ausência de soluções incorretas, pois o verificador analisa todas as trajetórias possíveis dentro do espaço de estados.

Possíveis Melhorias e Refinamentos

Embora o modelo proposto tenha atingido o objetivo de validar as três situações, algumas melhorias e refinamentos poderiam ser implementados para aprimorar tanto a eficiência quanto a clareza dos resultados.

Em primeiro lugar, a modelagem das transições pode ser detalhada de forma mais realista, incorporando explicitamente as pré-condições físicas de cada movimento (por exemplo, restrições de estabilidade e disponibilidade de espaço). Isso tornaria o modelo mais fiel ao comportamento real do mundo dos blocos e reduziria a quantidade de estados redundantes explorados pelo verificador.

Outra possibilidade é o uso de abstração hierárquica ou simbólica, técnica que agrupa estados semelhantes e reduz o tamanho do espaço de busca, permitindo uma verificação mais rápida e escalável. Essa abordagem é especialmente útil se o número de blocos ou posições fosse ampliado em versões futuras do problema. Do ponto de vista metodológico, poderia ser incorporada a análise de propriedades LTL (Linear Temporal Logic), que, ao contrário da CTL, permite expressar sequências lineares de ações e restrições temporais (“primeiro mover X, depois mover Y”). Isso enriqueceria a análise com propriedades de ordem e dependência temporal.

Por fim, uma melhoria prática seria integrar o modelo com ferramentas de geração automática de planos, como NuSMV-P, ou criar um script de pós-processamento que traduza automaticamente os contraexemplos gerados pelo verificador em uma sequência legível de comandos do tipo move(bloco, destino). Isso tornaria o processo mais interpretável e aproximaria o resultado formal de uma aplicação concreta em planejamento de ações. Em síntese, o trabalho alcançou os objetivos propostos, demonstrando a viabilidade de usar model checking para o planejamento automático de ações em um domínio físico simplificado. Entretanto, há espaço para aprimorar a representação das regras físicas e a automação da extração de planos, de modo a tornar o processo mais eficiente, expressivo e generalizável para domínios mais complexos.

Tabelas Comparativas

Situação 1

Tipo de restrição	STRIPS	Destino	Regra em Linguagem Natural	Prolog estendido	Implementação NuSMV	Justificativa / Equivalência

Block Properties	block(a).
	block(b).
	block(c).
	block(d).
		-	Cada bloco tem tamanho fixo	size(a,1).
				size(b,1).
				size(c,2).
				size(d,2).
					DEFINE
					size_a := 1;
					size_b := 1;
					size_c := 2;
					size_d := 2;
						Ambos representam o tamanho fixo como constantes globais.

Mobility	-	Bloco a	Um bloco só pode ser movido se não houver outro bloco em cima.	clear(Block, State) :- \+ member(pos(_, on(Block)), State).	DEFINE clear_a := (support_b != on_a) & (support_c != on_a) & (support_d != on_a);	O predicado clear/2 em Prolog equivale às definições clear_x no NuSMV, que verificam se nada está apoiado sobre o bloco.
Target Accessibility	-	Bloco b ou c	Só é possível empilhar um bloco sobre outro se ambos estiverem livres.	can(move(X, on(Y)), S) :- X \= Y, clear(X,S), clear(Y,S), stable_on(X,Y).	TRANS next(support_x) in {table, on_y, ...} (com clear_y incluído nas condições de transição)	As pré-condições de empilhamento no Prolog correspondem às restrições de transição do NuSMV: só ocorre mudança se ambos estiverem livres.
Stability	-	Empilhamento	Um bloco só pode ser colocado sobre outro de tamanho maior ou igual.	stable_on(X,Y) :- size(X,SX), size(Y,SY), SX =< SY.	DEFINE stable_a_on_b := size_a <= size_b;	O predicado stable_on/2 é diretamente traduzido em condições lógicas size_x <= size_y.
Spatial Occupancy	-	Slots da mesa	Para colocar um bloco sobre a mesa, os slots necessários	space_check(B,S,State) :- size(B,SZ), End is S+SZ-1, forall(between	slot_x : 0..6; TRANS next(slot_x) in 0..6;	No Prolog, a verificação é feita por iteração dos slots; no NuSMV, é representada por variáveis de faixa

			devem estar livres.	(S,End,I),(table_slot(I), occupied_slots(State,O), \+member(I,O))).		(0..6) que controlam posições possíveis.
Logical Validity	-	-	Um bloco não pode ser colocado sobre si mesmo.	can(move(X,on(Y)),_) :- X \= Y.	Condição implícita (não há on_self nos estados possíveis).	Essa restrição é tratada logicamente em ambas as linguagens.

Situação 2

Tipo de restrição	**STRIPS**	**Destino**	**Regra em Linguagem Natural**	**Prolog estendido**	**Implementação NuSMV**	**Justificativa / Equivalência**
Block Properties	block(a). block(b). block(c). block(d).	-	Cada bloco tem tamanho fixo	size(a,1). size(b,1). size(c,2). size(d,2).	DEFINE size_a := 1; size_b := 1; size_c := 2; size_d := 2;	Ambos representam o tamanho fixo como constantes globais.
Mobility	-	Bloco a, c	Um bloco só pode se mover se estiver livre.	clear(c, S). clear(a, S).	TRANS move_c_d -> clear_c; TRANS move_a_table 1 -> clear_a;	Correspondência direta entre o predicado clear/2 e as condições clear_x nas transições.
Target Accessibility	-	Bloco b, d	Só é possível empilhar um bloco sobre outro se ambos estiverem livres.	can(move(X, on(Y)), S) :- X \= Y, clear(X,S), clear(Y,S), stable_on(X,Y).	TRANS next(support_x) in {table, on_y, ...} (com clear_y incluído nas condições de transição)	As pré-condições de empilhamento no Prolog correspondem às restrições de transição do NuSMV: só ocorre mudança se ambos estiverem livres.
Stability	-	Empilhamento	Só é permitido empilhar c	stable_on(c,d).	TRANS move_c_d ->	A regra de estabilidade é expressa em ambas as

			sobre d se size(c) ≤ size(d).		size_c <= size_d;	linguagens da mesma forma.
Spatial Occupancy	-	Slots da mesa	Para colocar d na mesa, os slots necessários devem estar livres.	space_check(d,3,S).	TRANS move_d_table3 -> free(3) & free(4);	Em Prolog, o predicado space_check/3 faz a checagem iterativa; em SMV, a condição é explicitada como free(3)&free(4).
Logical Validity	-	-	Nenhum bloco pode ser movido sobre si mesmo.	can(move(X,on(Y)),S) :- X \= Y.	Implícito no domínio (sem “on_self”).	Mesmo conceito do modelo 1.

Situação 3

Tipo de restrição	**STRIPS**	**Destino**	**Regra em Linguagem Natural**	**Prolog estendido**	**Implementação NuSMV**	**Justificativa / Equivalência**

Block Properties	block(a).
	block(b).
	block(c).
	block(d).
		-	Cada bloco tem tamanho fixo	size(a,1).
				size(b,1).
				size(c,2).
				size(d,2).
					DEFINE
					size_a := 1;
					size_b := 1;
					size_c := 2;
					size_d := 2;
						Ambos representam o tamanho fixo como constantes globais.

Mobility	-	Bloco c	Apenas o topo (c) pode ser movido inicialmente.	clear(c,S).	TRANS move_c_a -> clear_c;	No início, apenas c é “clear”.
Target Accessibility	-	Bloco a ou b	Só é possível empilhar sobre	can(move(B,on(A)),S) :- clear(B,S), clear(A,S).	TRANS move_b_table2 -> clear_b;	Correspondência direta entre pré-condições de empilhamento.

			blocos livres.			
Stability	-	Empilhamento	d só pode ser colocado sobre c se size(d) ≤ size(c).	stable_on(d,c).	TRANS move_d_c -> size_d <= size_c;	Equivalência direta entre regra de estabilidade e condição SMV.
Spatial Occupancy	-	Slots da mesa	Deve haver espaço livre na mesa para desmontar a torre.	space_check(b,2,S).	TRANS move_b_table 2 -> free(2);	Mesma correspondência de espaço físico (slots).
Logical Validity	-	-	Nenhum bloco pode ser empilhado sobre si mesmo.	can(move(X,on(Y)),S) :- X \= Y.	Implícito	Restrição fundamental do domínio.