PCS 3838 - Inteligência Artificial
Exercício Prático

Victoria Tanaka - 8988806

Escola Politécnica - Universidade de São Paulo

Neste trabalho foi proposto o uso da linguagem Prolog [1] para desenvolver um programa que utiliza a abordagem de satisfação de restrições para resolver o Sudoku.

I. Introdução

Sudoku é um jogo baseado na colocação lógica de números. O objetivo do jogo é a colocação de números de 1 a 9 em cada uma das células vazias numa grade de 9x9, constituída por 3x3 subgrades chamadas regiões. O quebra-cabeça contém algumas pistas iniciais, que são números inseridos em algumas células, de maneira a permitir uma indução ou dedução dos números em células que estejam vazias. Cada coluna, linha e região só pode ter um número de cada um dos 1 a 9. [2]

II. Abordagem proposta e Estrutura do software

Foi usado como referência um tutorial encontrado no site do SWI (ambiente de execução utilizado no trabalho) [3], que descreve o uso da biblioteca clpfd (Constraint Logic Programming over Finite Domains).

O software é composto pelo arquivo *sudokuswi.pl*, mostrado na íntegra na listagem 1. Apenas uma modificação foi feita no código original: foi adicionada a linha 12, como será explicado a seguir.

```
1  :-use_module(library(clpfd)).
2  sudoku(Rows) :-
3          length(Rows, 9), maplist(same_length(Rows), Rows),
4          append(Rows, Vs), Vs ins 1..9,
5          maplist(all_distinct, Rows),
6          transpose(Rows, Columns),
7          maplist(all_distinct, Columns),
8          Rows = [As,Bs,Cs,Ds,Es,Fs,Gs,Hs,Is],
9          blocks(As, Bs, Cs),
10         blocks(Ds, Es, Fs),
11         blocks(Gs, Hs, Is),
12         maplist(label, Rows).
13
14 blocks([], [], []).
15 blocks([N1,N2,N3|Ns1], [N4,N5,N6|Ns2], [N7,N8,N9|Ns3]) :-
16         all_distinct([N1,N2,N3,N4,N5,N6,N7,N8,N9]),
17         blocks(Ns1, Ns2, Ns3).
18
19 problem(1, [[_,_,3,_,9,_,_,_,1],
20              [4,2,_,3,5,_,_,_,_],
21              [6,_,_,_,_,_,2,5,3,_],
22              [_,_,_,6,_,_,_,2,_],
23              [_,_,_,5,8,4,_,_,_],
24              [_,3,_,_,_,1,_,_,_],
25              [_,5,8,4,_,_,_,_,6],
26              [_,_,_,_,_,1,3,_,5,4],
27              [3,_,_,_,_,6,_,8,_,_]]).
28
29 % Outros problemas...
```

Listing 1: Algoritmo em Prolog do Sudoku.

A linha 1 apenas importa a biblioteca clpfd necessária. As linhas $2-12$ definem o predicado $sudoku(Rows)$, que recebe uma lista de listas (as linhas do puzzle). Definimos o tamanho dessas linhas como 9 (poderíamos resolver Sudokus de tamanhos diferentes, mas esse não é o caso), e todas devem ter o mesmo comprimento. Depois define-se o domínio das variáveis: números de 1 a 9. Através de $maplist/2$ e $all\_distinct/1$ estabelecemos que todas as linhas devem ser diferentes. Transpondo as linhas, obtemos as colunas, e novamente declaramos que as colunas são distintas. Por último, transforma-se as linhas em blocos de 3x3, como definido no predicado $blocks$. O predicado $label/1$ é chamado para todas as linhas, o que garante que todas as variáveis tem um valor único.
As linhas 14-17 definem os blocos 3x3 de forma recursiva (com as listas vazias sendo a base da recursão), conferindo se todos são dinstintos.
As funções $ins/2$, $all\_distinct/1$ e $transpose/2$ são da biblioteca clpfd.

III. Descrição dos experimentos

Foram escolhidos os seguintes exemplos, retirados de [4] e [5], com suas respectivas soluções.

		3		9				1
4	2		3	5				
6					2	5	3	
			6				2	
			5	8	4			
	3				1			
	5	8	4					6
				1	3		5	4
3				6		8		

(a)

5	8	3	7	9	6	2	4	1
4	2	1	3	5	8	6	7	9
6	9	7	1	4	2	5	3	8
8	4	5	6	3	9	1	2	7
7	1	2	5	8	4	9	6	3
9	3	6	2	7	1	4	8	5
1	5	8	4	2	7	3	9	6
2	6	9	8	1	3	7	5	4
3	7	4	9	6	5	8	1	2

(b)

Figura 1: Primeiro exemplo proposto e sua solução.

7			3					
	6				4	9		
		8	9	7	6	3		
	9				8		2	
		1				8		
	5		2				3	
		7	4	8	2	1		
		5	1				9	
					9			7

(a)

7	4	9	3	2	5	6	1	8
5	6	3	8	1	4	9	7	2
1	2	8	9	7	6	3	5	4
3	9	6	7	4	8	5	2	1
2	7	1	6	5	3	8	4	9
8	5	4	2	9	1	7	3	6
9	3	7	4	8	2	1	6	5
4	8	5	1	6	7	2	9	3
6	1	2	5	3	9	4	8	7

(b)

Figura 2: Segundo exemplo proposto e sua solução.

Foi testado também um terceiro exemplo, retirado da seção "muito difícil"do site fonte [6]. O porquê desse exemplo extra será explicado na seção IV.

8				1		9		
			4		3			1
							7	4
	6		2				5	
						2		
	3				8		6	
		5						
2		4	1					
		8	9	4				3

(a)

8	4	6	5	1	7	9	3	2
9	5	7	4	2	3	6	8	1
1	2	3	6	8	9	5	7	4
7	6	1	2	9	4	3	5	8
5	8	9	3	6	1	2	4	7
4	3	2	7	5	8	1	6	9
3	9	5	8	7	2	4	1	6
2	7	4	1	3	6	8	9	5
6	1	8	9	4	5	7	2	3

(b)

Figura 3: Exemplo muito difícil proposto e sua solução.

Os programas foram executados num sistema Windows 10 Pro, versão 1803. O tempo de execução foi rápido, todos os exemplos foram resolvidos sem nenhum tempo de espera perceptível.

IV. Análise dos resultados

A figura 4 mostra os resultados dos exemplos mostrados na seção III.

```
?- problem(1, Rows), sudoku(Rows), maplist(portray_clause, Rows).
[5, 8, 3, 7, 9, 6, 2, 4, 1].
[4, 2, 1, 3, 5, 8, 6, 7, 9].
[6, 9, 7, 1, 4, 2, 5, 3, 8].
[8, 4, 5, 6, 3, 9, 1, 2, 7].
[7, 1, 2, 5, 8, 4, 9, 6, 3].
[9, 3, 6, 2, 7, 1, 4, 8, 5].
[1, 5, 8, 4, 2, 7, 3, 9, 6].
[2, 6, 9, 8, 1, 3, 7, 5, 4].
[3, 7, 4, 9, 6, 5, 8, 1, 2].
Rows = [[5, 8, 3, 7, 9, 6, 2, 4|...], [4, 2, 1, 3, 5, 8, 6|...], [6, 9,

?- problem(2, Rows), sudoku(Rows), maplist(portray_clause, Rows).
[7, 4, 9, 3, 2, 5, 6, 1, 8].
[5, 6, 3, 8, 1, 4, 9, 7, 2].
[1, 2, 8, 9, 7, 6, 3, 5, 4].
[3, 9, 6, 7, 4, 8, 5, 2, 1].
[2, 7, 1, 6, 5, 3, 8, 4, 9].
[8, 5, 4, 2, 9, 1, 7, 3, 6].
[9, 3, 7, 4, 8, 2, 1, 6, 5].
[4, 8, 5, 1, 6, 7, 2, 9, 3].
[6, 1, 2, 5, 3, 9, 4, 8, 7].
Rows = [[7, 4, 9, 3, 2, 5, 6, 1|...], [5, 6, 3, 8, 1, 4, 9|...], [1, 2,

?- problem(3, Rows), sudoku(Rows), maplist(portray_clause, Rows).
[8, 4, 6, 5, 1, 7, 9, 3, 2].
[9, 5, 7, 4, 2, 3, 6, 8, 1].
[1, 2, 3, 6, 8, 9, 5, 7, 4].
[7, 6, 1, 2, 9, 4, 3, 5, 8].
[5, 8, 9, 3, 6, 1, 2, 4, 7].
[4, 3, 2, 7, 5, 8, 1, 6, 9].
[3, 9, 5, 8, 7, 2, 4, 1, 6].
[2, 7, 4, 1, 3, 6, 8, 9, 5].
[6, 1, 8, 9, 4, 5, 7, 2, 3].
```

Figura 4: Resultado das consultas para os 3 problemas propostos.

Como podemos ver das figuras 1b, 2b e 3b, as respostas encontradas batem com a solução.

Para obter os resultados, foi usada a mesma consulta apresentada em [3]:

$$\boxed{problem(3, Rows), sudoku(Rows), maplist(portray_clause, Rows).}$$

$problem(3, Rows)$ indica o que o problema a ser resolvido é o exemplo 3 (definido no código fonte);
$sudoku(Rows)$ resolve o problema de restrições de domínio do Sudoku; e
$maplist(portray_clause, Rows)$ apenas imprime as soluções de uma forma legível.

O exemplo 3 foi incluído para demonstrar o motivo da alteração do código original. A figura 5 mostra parte do output da consulta para o exemplo 3 sem a mudança.

```
?- problem(3, Rows), sudoku(Rows), maplist(portray_clause, Rows).
[8, 4, _, _, 1, _, 9, 3, _].
[_, _, _, 4, _, 3, _, _, 1].
[1, _, 3, _, _, 9, _, 7, 4].
[_, 6, 1, 2, _, 4, 3, 5, 8].
[5, 8, _, 3, 6, 1, 2, 4, _].
[4, 3, 2, _, _, 8, 1, 6, _].
[3, _, 5, _, _, _, 4, _, _].
[2, _, 4, 1, 3, _, _, _, _].
[6, _, 8, 9, 4, _, _, _, 3].
Rows = [[8, 4, _76050, _76056, 1, _76068, 9, 3|...], [_76098, _76104, _76110, 4, _7
0, 3|...], [4, 3, 2|...], [3, _76404|...], [2|...], [...|...]],
_76050 in 6..7,
all_distinct([8, 4, _76050, _76098, _76104, _76110, 1, _76164|...]),
all_distinct([_76050, _76110, 3, 1, _76290, 2, 5, 4|...]),
all_distinct([8, 4, _76050, _76056, 1, _76068, 9, 3|...]),
_76098 in 7\/9,
all_distinct([8, _76098, 1, _76218, 5, 4, 3, 2|...]),
all_distinct([_76098, _76104, _76110, 4, _76122, 3, _76134, _76140|...]),
_76104 in 2\/5,
all_distinct([4, _76104, _76164, 6, 8, 3, _76404, _76464|...]),
_76110 in 6..7\/9,
_76164 in 2\/5,
all_distinct([1, _76164, 3, _76176, _76182, 9, _76194, 7|...]),
_76404 in 1\/9,
all_distinct([3, _76404, 5, 2, _76464, 4, 6, _76524|...]),
all_distinct([3, _76404, 5, _76416, _76422, _76428, 4, _76440|...]),
_76464 in 7\/9,
all_distinct([2, _76464, 4, 1, 3, _76488, _76494, _76500|...]),
_76524 in 1\/7,
all_distinct([6, _76524, 8, 9, 4, _76548, _76554, _76560|...]),
_76548 in 2\/5\/7,
all_distinct([_76416, _76422, _76428, 1, 3, _76488, 9, 4|...]),
all_distinct([_76068, 3, 9, 4, 1, 8, _76428, _76488|...]),
_76488 in 5..7,
_76428 in 2\/6..7,
_76422 in 2\/7..8,
all_distinct([1, _76122, _76182, _76242, 6, _76362, _76422, 3|...]),
_76416 in 6..8,
all_distinct([_76056, 4, _76176, 2, 3, _76356, _76416, 1|...]),
_76356 in 5\/7,
all_distinct([2, _76242, 4, 3, 6, 1, _76356, _76362|...]),
all_distinct([4, 3, 2, _76356, _76362, 8, 1, 6|...]),
_76362 in 5\/7\/9,
_76242 in 7\/9,
all_distinct([_76218, 6, 1, 2, _76242, 4, 3, 5|...]),
_76182 in 2\/5\/8,
all_distinct([_76056, 1, _76068, 4, _76122, 3, _76176, _76182|...]),
_76176 in 5..6\/8,
_76122 in 2\/5\/7..8,
_76068 in 2\/5..7,
```

Figura 5: Resultado da consulta para o exemplo muito difícil com o código original. Note que as variáveis não são instanciadas, e o programa só encontra algumas posições e restringe outras.

Como discutido na seção II, o comando $maplist(label, Rows)$ força que todas as variáveis tenham um único valor concreto (ou seja, podemos resolver até Sudokus com mais de uma solução). É provado que problemas de Sudoku devem ter pelo menos 17 dicas para haver uma única solução (apesar de existirem problemas com mais de 17 que também possuem múltiplas soluções) [7]. Então, para testar o efeito de um Sudoku com mais de uma solução no nosso código, basta criar um problema com menos de 17 dicas, ou usar qualquer problema mal formulado. Como exemplo

de execução, a figura 6 mostra várias soluções encontradas para o puzzle retirado de [8]. O output do código original para este exemplo é semelhante à figura 5.

```
?- problem(4, Rows), sudoku(Rows), maplist(portray_clause, Rows).
[2, 8, 6, 1, 5, 9, 7, 4, 3].
[3, 5, 4, 7, 6, 8, 9, 1, 2].
[7, 1, 9, 2, 4, 3, 5, 6, 8].
[8, 2, 3, 6, 1, 5, 4, 9, 7].
[6, 9, 7, 8, 2, 4, 1, 3, 5].
[1, 4, 5, 3, 9, 7, 8, 2, 6].
[5, 6, 8, 9, 3, 1, 2, 7, 4].
[4, 3, 1, 5, 7, 2, 6, 8, 9].
[9, 7, 2, 4, 8, 6, 3, 5, 1].
Rows = [[2, 8, 6, 1, 5, 9, 7, 4|...], [3, 5, 4, 7, 6, 8, 9|...], [7, 1, 9, 2, 4, 3|...], [
[2, 8, 6, 1, 5, 9, 7, 4, 3].
[3, 5, 4, 7, 6, 8, 9, 1, 2].
[7, 1, 9, 2, 4, 3, 5, 6, 8].
[8, 9, 3, 6, 1, 5, 4, 2, 7].
[6, 2, 7, 8, 9, 4, 1, 3, 5].
[1, 4, 5, 3, 2, 7, 8, 9, 6].
[5, 6, 8, 9, 3, 1, 2, 7, 4].
[4, 3, 1, 5, 7, 2, 6, 8, 9].
[9, 7, 2, 4, 8, 6, 3, 5, 1].
Rows = [[2, 8, 6, 1, 5, 9, 7, 4|...], [3, 5, 4, 7, 6, 8, 9|...], [7, 1, 9, 2, 4, 3|...], [
[2, 8, 6, 1, 5, 9, 7, 4, 3].
[3, 5, 4, 7, 6, 8, 9, 1, 2].
[7, 1, 9, 2, 4, 3, 5, 6, 8].
[8, 9, 3, 6, 1, 5, 4, 2, 7].
[6, 2, 7, 8, 9, 4, 1, 3, 5].
[1, 4, 5, 3, 7, 2, 8, 9, 6].
[5, 6, 8, 9, 3, 1, 2, 7, 4].
[4, 3, 1, 5, 2, 7, 6, 8, 9].
[9, 7, 2, 4, 8, 6, 3, 5, 1].
Rows = [[2, 8, 6, 1, 5, 9, 7, 4|...], [3, 5, 4, 7, 6, 8, 9|...], [7, 1, 9, 2, 4, 3|...], [
[2, 8, 6, 1, 5, 9, 7, 4, 3].
[3, 5, 7, 6, 4, 8, 2, 1, 9].
[4, 1, 9, 7, 2, 3, 5, 6, 8].
[8, 2, 1, 9, 6, 5, 4, 3, 7].
[6, 9, 3, 8, 7, 4, 1, 2, 5].
[7, 4, 5, 3, 1, 2, 8, 9, 6].
[5, 6, 8, 2, 3, 1, 9, 7, 4].
[1, 3, 4, 5, 9, 7, 6, 8, 2].
[9, 7, 2, 4, 8, 6, 3, 5, 1].
Rows = [[2, 8, 6, 1, 5, 9, 7, 4|...], [3, 5, 7, 6, 4, 8, 2|...], [4, 1, 9, 7, 2, 3|...], [
```

Figura 6: Resultado da consulta para o problema com muitas soluções (para este caso em particular, foram encontradas 8 soluções diferentes). Apenas algumas soluções foram mostrada por questões de legibilidade.

Na verdade, poderíamos usar o código original, porém a consulta deveria ser modificada para incluir essa única instanciação:

$$\boxed{problem(3, Rows), sudoku(Rows), \mathbf{maplist(label,\ Rows)}, maplist(portray_clause, Rows).}$$

Faz muito mais sentido, entretanto, colocar essa restrição dentro do algoritmo, para a string de consulta ficar mais simples e inteligível.

V. Conclusões e trabalhos futuros

O uso da linguagem Prolog para a resolução deste problema específico mostrou-se muito oportuna, pois Sudoku é fundamentalmente um problema de lógica e facilmente descrito por satisfação de restrições. Com apenas algumas linhas de código, conseguimos resolver os mais variados exemplos de Sudoku que achamos na Internet. Outra vantagem do Prolog neste caso é a sua clareza e facilidade de entendimento do algoritmo.
Para problemas mais complexos, teremos resoluções mais complexas. Mas casos como este, onde o paradigma lógico se aplica, o uso de linguagens como Prolog é muito mais produtivo do que o uso de linguagens procedurais.

Referências

[1] "Swi prolog." http://www.swi-prolog.org/. [Online; accessed 2018-11-20].
[2] "Sudoku." https://pt.wikipedia.org/wiki/Sudoku. [Online; accessed 2018-11-20].
[3] "Biblioteca clpfd." http://www.swi-prolog.org/pldoc/man?section=clpfd. [Online; accessed 2018-11-20].
[4] "Sudoku exemplo 1." http://www.7sudoku.com/view-puzzle?date=20181006. [Online; accessed 2018-11-20].
[5] "Sudoku exemplo 2." http://www.7sudoku.com/view-puzzle?date=20181010. [Online; accessed 2018-11-20].
[6] "Sudoku exemplo 3." http://www.7sudoku.com/view-puzzle?date=20181010. [Online; accessed 2018-11-20].
[7] "17 and sudoku clues - numberphile." https://www.youtube.com/watch?ab_channel=Numberphile&v=MlyTq-xVkQE. [Online; accessed 2018-12-05].
[8] "Unsolvable sudoku puzzles." http://www.sudokudragon.com/unsolvable.htm. [Online; accessed 2018-12-05].