# Sudoku em Prolog

**Antônio Rege Lopes dos Santos, Victor Antunes Vieira**

Instituto Federal de Educação, Ciência e Tecnologia do Acre (IFAC)
Rio Branco, AC - Brasil

Universidade Federal Fluminense (UFF)
Niterói, RJ - Brasil

{antonio.rsantos,victor.vieira}@ifac.edu.br

***Abstract.*** *This article describes an algorithm in the Prolog language to solve the game Sudoku.*

***Resumo.*** *Este artigo descreve um algoritmo na linguagem Prolog para resolver o jogo Sudoku.*

## 1. Introdução

Sudoku é um jogo baseado na colocação lógica de números em uma grade. O objetivo consiste em colocar números de 1 a 9 em cada uma das células vazias na grade de 9 linhas e 9 colunas (9x9), constituída por 3x3 subgrades chamadas de regiões. As regras do jogo podem ser encontradas em https://pt.wikipedia.org/wiki/Sudoku [1].

A Figura 1 apresenta uma grade de Sudoku com o jogo pronto para ser iniciado. Para vencer, o jogador precisa preencher os espaços em branco com números de 1 a 9 de maneira que eles não se repitam nem nas linhas, nem nas colunas e nem nas subgrades.

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

**Figura 1. Um típico jogo de Sudoku**

Este artigo descreve um algoritmo em Prolog, linguagem de programação baseada nos paradigmas lógico e declarativo e que é especialmente associada com a Inteligência Artificial, para resolver o jogo Sudoku em qualquer nível. Usou-se SWI-PROLOG [2] e a biblioteca Constraint Logic Programming over Finite Domains (*clpfd*) do SWI [3].

O restante do texto está organizado da seguinte forma: na seção 2, o algoritmo é apresentado detalhadamente; e na seção 3 são descritas as considerações finais sobre o

trabalho, incluindo link para o repositório com o código no GitHub e recomendação de trabalhos futuros.

## 2. Algoritmo

Para facilitar o entendimento da solução, o código-fonte é analisado em partes, com imagens representando os trechos do algoritmo, bem como com a numeração das linhas para referenciá-las no texto. O número das linhas segue a sequência original, conforme o código-fonte completo. A Figura 2 apresenta as primeiras linhas do código.

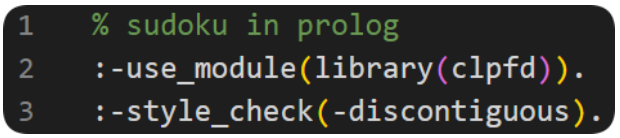


**Figura 2. Início do código de Sudoku em Prolog**

Na linha 1, observa-se um comentário em Prolog. Linhas iniciadas com o caracter "%" são tratadas como comentários e, portanto, não são consideradas na execução do programa. Para comentários com mais de uma linha, podem ser utilizados os caracteres "/*", para abrir, e "*/", para fechar o comentário.

Na linha 2, há a utilização de *use_module()* para carregar a biblioteca *clpfd*. O comando *use_module* é usado no Prolog para carregar uma biblioteca ou módulo de código em um programa. Ele permite que você inclua funções, predicados e outras definições de código em seu programa, de forma a reutilizá-las ou estender as funcionalidades disponíveis.

A biblioteca *clpfd* é uma extensão para o Prolog que permite que você resolva problemas de programação lógica que envolvam restrições em um conjunto finito de valores. Ela fornece uma série de predicados e operadores que permitem expressar essas restrições e também fornece uma série de algoritmos para resolver esses problemas. Nesse trabalho, o principal predicado utilizado foi o *all_different/1*.

A documentação dos predicados segue o padrão utilizado na documentação oficial. Dado um predicado *pred* com n argumentos, é representado no código por *pred/n*. O método *all_different/1* da biblioteca *clpfd* é usado para impor a restrição de que todos os elementos de uma lista devem ser diferentes entre si. Ele é útil para resolver problemas de cálculo em que é necessário garantir que cada elemento da lista é único [4].

Por exemplo, suponha que exista uma lista *P* com alguns elementos e exista, ainda, a necessidade de garantia de que todos esses elementos sejam diferentes. Pode ser utilizado o método *all_different/1* da seguinte maneira:

*all_different(P).*

Isso irá forçar a variável *P* a ser instanciada com valores que são todos diferentes entre si. Se existir um conjunto de restrições adicionais para a lista, pode ser usado o

método *all_different/1* para garantir que essas restrições sejam satisfeitas enquanto ainda se mantém a restrição de que todos os elementos da lista sejam diferentes.

Um exemplo que se relaciona com a solução do Sudoku é o seguinte: suponha que haja uma lista de números inteiros e é necessário garantir que todos esses números são diferentes entre si e também estão no intervalo de 1 a 9. Pode ser utilizado o método *all_different/1* da seguinte maneira:

*P ins 1..9, all_different(P).*

Isso irá forçar a variável *P* a ser instanciada com valores que são todos diferentes entre si e também estão no intervalo de 1 a 9.

Continuando a análise do código-fonte, entre as linhas 4 e 6 observa-se a "função" *solve()*, conforme apresentado na Figura 3. A palavra função aparece entre aspas pois em Prolog chama-se de predicados (que, por sua vez, podem ser fatos, regras ou consultas). Têm esse nome por derivar da lógica de predicados.

```
4   solve(P):-
5       sudoku(P),
6       forall(member(R,P), (print(R),nl)).
```

**Figura 3. Declaração do predicado *solve/1***

Na linha 4, é realizada a definição do predicado *solve/1*, que tem como objetivo resolver o quebra-cabeça de Sudoku dado. Ele possui um único argumento, *P*, que é uma lista representando o quebra-cabeça. O predicado *solve/1* é composto por duas cláusulas:

- *sudoku(P)*, predicado auxiliar que tenta encontrar uma solução para o quebra-cabeça de sudoku dado. Esse predicado é responsável por aplicar as regras do sudoku e verificar se a solução encontrada é válida e será detalhado na sequência deste documento;
- *forall(member(R,P), (print(R),nl))*, que é um predicado que itera sobre cada linha *R* da lista *P* e imprime cada uma delas, seguida de uma quebra de linha (*nl*). Esse predicado é responsável por imprimir a solução encontrada pelo predicado *sudoku/1*. A quebra de linha tem como objetivo facilitar a solução final.

Em resumo, o predicado *solve/1* tenta resolver um quebra-cabeça de Sudoku usando o predicado *sudoku/1* e em seguida imprime a solução encontrada usando o predicado *forall/2*.

A partir da linha 7 é possível observar na Figura 4 o predicado *sudoku()*. Há a declaração de uma lista de variáveis chamada *Vars* atribuindo a ela uma lista de 81 variáveis, representando a grade de Sudoku tradicional, 9x9. Em seguida, é utilizado o predicado *ins* para especificar que cada uma dessas variáveis deve estar dentro do intervalo de 1 a 9. Isso significa que, no final da execução dessa linha, todas as variáveis em *Vars* devem ter um valor entre 1 e 9.

```
7   sudoku([[A1,B1,C1,D1,E1,F1,G1,H1,I1],
8           [A2,B2,C2,D2,E2,F2,G2,H2,I2],
9           [A3,B3,C3,D3,E3,F3,G3,H3,I3],
10          [A4,B4,C4,D4,E4,F4,G4,H4,I4],
11          [A5,B5,C5,D5,E5,F5,G5,H5,I5],
12          [A6,B6,C6,D6,E6,F6,G6,H6,I6],
13          [A7,B7,C7,D7,E7,F7,G7,H7,I7],
14          [A8,B8,C8,D8,E8,F8,G8,H8,I8],
15          [A9,B9,C9,D9,E9,F9,G9,H9,I9]]):-
16      Vars = [A1,B1,C1,D1,E1,F1,G1,H1,I1,
17              A2,B2,C2,D2,E2,F2,G2,H2,I2,
18              A3,B3,C3,D3,E3,F3,G3,H3,I3,
19              A4,B4,C4,D4,E4,F4,G4,H4,I4,
20              A5,B5,C5,D5,E5,F5,G5,H5,I5,
21              A6,B6,C6,D6,E6,F6,G6,H6,I6,
22              A7,B7,C7,D7,E7,F7,G7,H7,I7,
23              A8,B8,C8,D8,E8,F8,G8,H8,I8,
24              A9,B9,C9,D9,E9,F9,G9,H9,I9],
25      Vars ins 1..9,
```

**Figura 4. Declaração do predicado sudoku**

Essa declaração é um tipo de restrição de domínio, que especifica os possíveis valores que cada uma das variáveis pode assumir. No caso do Sudoku, essa restrição é usada para garantir que cada célula da grade tenha um valor válido, ou seja, um número entre 1 e 9. É comum que as restrições de domínio sejam usadas em programação lógica para limitar os possíveis valores que as variáveis podem assumir e, assim, simplificar a resolução de problemas.

## 2.1. Linhas

A linha 27, na Figura 5, inicia a aplicação do predicado *all_different* para garantir que não haja números idênticos na linha 1 da grade do jogo.

```
26      % Rows
27      all_different([A1,B1,C1,D1,E1,F1,G1,H1,I1]),
28      all_different([A2,B2,C2,D2,E2,F2,G2,H2,I2]),
29      all_different([A3,B3,C3,D3,E3,F3,G3,H3,I3]),
30      all_different([A4,B4,C4,D4,E4,F4,G4,H4,I4]),
31      all_different([A5,B5,C5,D5,E5,F5,G5,H5,I5]),
32      all_different([A6,B6,C6,D6,E6,F6,G6,H6,I6]),
33      all_different([A7,B7,C7,D7,E7,F7,G7,H7,I7]),
34      all_different([A8,B8,C8,D8,E8,F8,G8,H8,I8]),
35      all_different([A9,B9,C9,D9,E9,F9,G9,H9,I9]),
```

**Figura 5. Predicado *all_different* garantindo valores diferentes entre os números de cada linha.**

O esquema da Tabela 1 representa visualmente a grade 9x9 do jogo. As linhas 28 a 35 do código da Figura 5 representam as linhas de 2 a 9 do Sudoku.

**Tabela 1. Organização da grade do Sudoku 9x9 com destaque para a primeira linha**

| A1 | B1 | C1 | D1 | E1 | F1 | G1 | H1 | I1 |
|---|---|---|---|---|---|---|---|---|
| A2 | B2 | C2 | D2 | E2 | F2 | G2 | H2 | I2 |
| A3 | B3 | C3 | D3 | E3 | F3 | G3 | H3 | I3 |
| A4 | B4 | C4 | D4 | E4 | F4 | G4 | H4 | I4 |
| A5 | B5 | C5 | D5 | E5 | F5 | G5 | H5 | I5 |
| A6 | B6 | C6 | D6 | E6 | F6 | G6 | H6 | I6 |
| A7 | B7 | C7 | D7 | E7 | F7 | G7 | H7 | I7 |
| A8 | B8 | C8 | D8 | E8 | F8 | G8 | H8 | I8 |
| A9 | B9 | C9 | D9 | E9 | F9 | G9 | H9 | I9 |

## 2.2. Colunas

A partir da linha 37 são aplicados os predicados para garantir a regra de não haver elementos repetidos nas colunas, como pode ser visto na Figura 6.

```
36      % Columns
37      all_different([A1,A2,A3,A4,A5,A6,A7,A8,A9]),
38      all_different([B1,B2,B3,B4,B5,B6,B7,B8,B9]),
39      all_different([C1,C2,C3,C4,C5,C6,C7,C8,C9]),
40      all_different([D1,D2,D3,D4,D5,D6,D7,D8,D9]),
41      all_different([E1,E2,E3,E4,E5,E6,E7,E8,E9]),
42      all_different([F1,F2,F3,F4,F5,F6,F7,F8,F9]),
43      all_different([G1,G2,G3,G4,G5,G6,G7,G8,G9]),
44      all_different([H1,H2,H3,H4,H5,H6,H7,H8,H9]),
45      all_different([I1,I2,I3,I4,I5,I6,I7,I8,I9]),
```

**Figura 6. Garantindo valores diferentes entre os números de cada coluna**

No esquema da Tabela 2, a coluna destacada representa a região considerada na linha 37. As linhas 38 a 45 representam as colunas 2 a 9.

**Tabela 2. Organização da grade do Sudoku 9x9 com destaque para a primeira coluna**

| A1 | B1 | C1 | D1 | E1 | F1 | G1 | H1 | I1 |
|---|---|---|---|---|---|---|---|---|
| A2 | B2 | C2 | D2 | E2 | F2 | G2 | H2 | I2 |
| A3 | B3 | C3 | D3 | E3 | F3 | G3 | H3 | I3 |
| A4 | B4 | C4 | D4 | E4 | F4 | G4 | H4 | I4 |
| A5 | B5 | C5 | D5 | E5 | F5 | G5 | H5 | I5 |
| A6 | B6 | C6 | D6 | E6 | F6 | G6 | H6 | I6 |
| A7 | B7 | C7 | D7 | E7 | F7 | G7 | H7 | I7 |
| A8 | B8 | C8 | D8 | E8 | F8 | G8 | H8 | I8 |
| A9 | B9 | C9 | D9 | E9 | F9 | G9 | H9 | I9 |

## 2.3. Subgrades

A partir da linha 47 o predicado *all_different* é utilizado em cada uma das regiões 3x3 da grade do jogo (também chamadas de subgrades ou regiões), como pode ser visto na Figura 7.

```
46      % Squares
47      all_different([A1,A2,A3,B1,B2,B3,C1,C2,C3]),
48      all_different([D1,D2,D3,E1,E2,E3,F1,F2,F3]),
49      all_different([G1,G2,G3,H1,H2,H3,I1,I2,I3]),
50      all_different([A4,A5,A6,B4,B5,B6,C4,C5,C6]),
51      all_different([D4,D5,D6,E4,E5,E6,F4,F5,F6]),
52      all_different([G4,G5,G6,H4,H5,H6,I4,I5,I6]),
53      all_different([A7,A8,A9,B7,B8,B9,C7,C8,C9]),
54      all_different([D7,D8,D9,E7,E8,E9,F7,F8,F9]),
55      all_different([G7,G8,G9,H7,H8,H9,I7,I8,I9]),
56      label(Vars).
```

**Figura 7. Garantindo valores diferentes entre os números de cada subgrade**

O esquema da Tabela 3 representa a área considerada na linha 47 do código-fonte. As linhas subsequentes, até a 55, consideram todas as subgrades restantes. O predicado *label/1*, na linha 5 da Figura 7, é usado para atribuir valores às variáveis declaradas no seu argumento de modo a satisfazer todas as restrições impostas pelo programa.

**Tabela 3. Organização da grade do Sudoku 9x9 com destaque para a primeira subgrade**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **A1** | **B1** | **C1** | D1 | E1 | F1 | G1 | H1 | I1 |
| **A2** | **B2** | **C2** | D2 | E2 | F2 | G2 | H2 | I2 |
| **A3** | **B3** | **C3** | D3 | E3 | F3 | G3 | H3 | I3 |
| A4 | B4 | C4 | D4 | E4 | F4 | G4 | H4 | I4 |
| A5 | B5 | C5 | D5 | E5 | F5 | G5 | H5 | I5 |
| A6 | B6 | C6 | D6 | E6 | F6 | G6 | H6 | I6 |
| A7 | B7 | C7 | D7 | E7 | F7 | G7 | H7 | I7 |
| A8 | B8 | C8 | D8 | E8 | F8 | G8 | H8 | I8 |
| A9 | B9 | C9 | D9 | E9 | F9 | G9 | H9 | I9 |

## 2.4. Testes

Para testar o funcionamento do algoritmo, pode-se definir uma entrada que representa um preenchimento parcial do esquema do jogo. Por exemplo, pode-se utilizar a seguinte lista (os índices que contém "_" representam as posições que devem ser preenchidas pelo algoritmo):

*[*
*[_,4,7,_,9,_,_,6,_],*
*[9,5,_,_,_,8,_,_,2],*
*[6,_,_,7,_,_,_,_,_],*
*[_,3,_,_,6,_,_,5,_],*
*[_,_,9,_,_,_,6,_,_],*
*[_,8,_,_,4,_,_,2,_],*
*[_,_,_,_,_,5,_,_,6],*
*[4,_,_,9,_,_,_,1,3],*
*[_,9,_,_,1,_,2,_,7]*
*]*

A representação visual dessa lista é apresentada no esquema da Tabela 4.

**Tabela 4. Representação visual de possíveis dados de entrada para o algoritmo do Sudoku em Prolog.**

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

Passa-se a lista acima como parâmetro do predicado *solve()* para que ele possa processar e retornar o resultado correto do jogo.

*solve([[_,4,7,_,9,_,_,6,_],[9,5,_,_,_,8,_,_,2],[6,_,_,7,_,_,_,_,_],[_,3,_,_,6,_,_,5,_],[_,_,9,_,_,_,6,_,_],[_,8,_,_,4,_,_,2,_],[_,_,_,_,_,5,_,_,6],[4,_,_,9,_,_,_,1,3],[_,9,_,_,1,_,2,_,7]]).*

O resultado da chamada de solve com a lista pode ser visto na Figura 8, com as quebras de linha para cada índice, por conta de *nl* na linha 6 do algoritmo. O resultado foi obtido utilizando o SWISH [5].

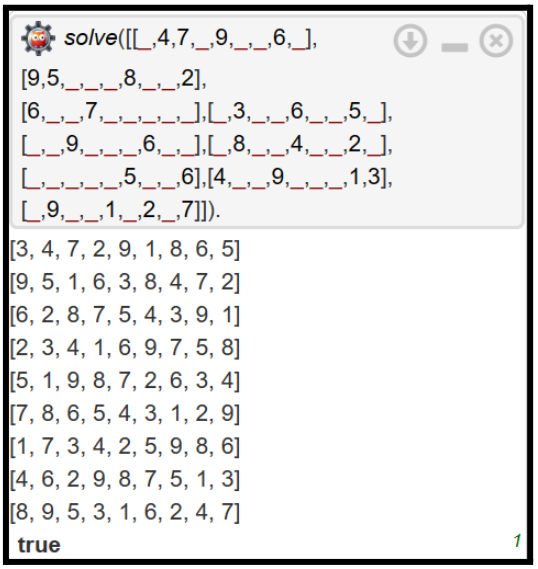


**Figura 8. Resultado do algoritmo para uma entrada específica**

Para facilitar a visualização, o esquema da Tabela 5 apresenta o grid com a solução de Sudoku dada pelo algoritmo. As células em destaque representam os valores que foram preenchidos automaticamente pelo programa.

**Tabela 5. Representação visual de solução do Sudoku, dada uma entrada**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **3** | 4 | 7 | **2** | 9 | **1** | **8** | 6 | **5** |
| 9 | 5 | **1** | **6** | **3** | 8 | **4** | **7** | 2 |
| 6 | **2** | **8** | 7 | **5** | **4** | **3** | **9** | **1** |
| **2** | 3 | **4** | **1** | 6 | **9** | **7** | 5 | **8** |
| **5** | **1** | 9 | **8** | **7** | **2** | 6 | **3** | **4** |
| **7** | 8 | **6** | **5** | 4 | **3** | **1** | 2 | **9** |
| **1** | **7** | **3** | **4** | **2** | 5 | **9** | **8** | 6 |
| 4 | **6** | **2** | 9 | **8** | **7** | **5** | 1 | 3 |
| **8** | 9 | **5** | **3** | 1 | **6** | 2 | **4** | 7 |

## 3. Considerações finais

Este artigo apresentou um algoritmo em Prolog para o jogo Sudoku. O algoritmo utiliza a biblioteca *clpfd* [3] do SWI-Prolog [2] para garantir que não existam números iguais em linhas, colunas e subgrades, obedecendo às regras do jogo. O código está disponível em Repositório do GitHub acessível a partir do endereço https://github.com/vrcvieira/sudoku-prolog.

Os níveis de dificuldades são definidos a partir das entradas. Sugere-se testes com as seguintes entradas [6]:

- Nível fácil:
  - *solve([[8,_,_,_,_,5,_,_,_],[_,7,_,9,_,_,_,4,_],[_,_,9,_,7,8,3,2,5],[3,_,1,_,9,_,_,5,_],[_,_,6,_,_,_,1,_,_],[_,9,_,_,3,_,6,_,2],[2,8,3,6,5,_,7,_,_],[_,1,_,_,2,_,8,_,_],[_,_,_,1,_,_,_,_,9]]).*

- Nível médio:
    - *solve([[1,_,_,8,7,_,_,_,_],[_,6,_,_,9,4,_,_,_],[_,_,9,_,_,6,1,_,_],[6,_,_,7,_,_,2,1,_],[4,9,_,_,_,_,_,5,3],[_,2,1,_,_,5,_,_,6],[_,_,4,5,_,_,8,_,_],[_,_,_,9,6,_,_,7,_],[_,_,_,_,4,8,_,_,1]]).*
- Nível difícil:
    - *solve([[_,_,_,4,_,1,_,_,_],[_,_,2,_,_,_,1,_,_],[_,6,_,_,8,_,_,7,_],[4,_,_,_,_,_,_,_,9],[_,_,8,_,_,_,4,_,_],[7,_,_,_,_,_,_,_,5],[_,7,_,_,4,_,_,5,_],[_,_,9,_,_,_,3,_,_],[_,_,_,1,_,3,_,_,_]]).*

Como trabalhos futuros, sugere-se: a) a classificação de diferentes possibilidades de entradas para o Sudoku em níveis, seguindo métricas bem estabelecidas para a classificação; e b) a construção de grades com valores distintos múltiplos de 9.

## Referências


[1] Sudoku, https://pt.wikipedia.org/wiki/Sudoku, dezembro de 2022.

[2] SWI-Prolog, https://www.swi-prolog.org, dezembro de 2022.

[3] SWI-Prolog Manual: library(clpfd), https://www.swi-prolog.org/man/clpfd.html, dezembro de 2022.

[4] Malheiro, R. P. T. (2010). Resolução do Problema do Escalonamento da Produção de Energia Eléctrica usando Programação Lógica por Restrições (Doctoral dissertation, Instituto Politecnico do Porto (Portugal)).

[5] SWISH, https://swish.swi-prolog.org, dezembro de 2022.

[6] 42 modelos de sudoku para imprimir de todos os níveis, https://www.artesanatopassoapassoja.com.br/sudoku-para-imprimir, dezembro de 2022.