FLIA - 2023.2

Planning Sudoku

Bruno Ribeiro - Danilo Carvalho - Igor Penha - Lucas Gobbi

ÍNDICE

01

INTRODUÇÃO

Introdução

O objetivo geral do trabalho era desenvolver um domínio de tema livre na linguagem de planejamento HDDL.

O foco principal do uso de HDDL é a hierarquia entre “tasks”, ou seja, ações possuem ordens totais ou parciais para ocorrerem.

Sob essa ótica, o sudoku é um jogo que se enquadra bem nessa perspectiva, pois, para resolvê-lo, faz-se necessária uma checagem de linha, coluna e “sub-grid” onde o número está sendo colocado.

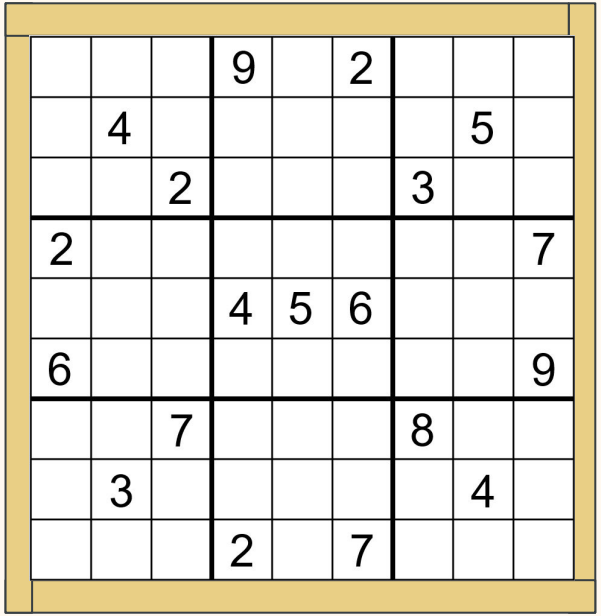
02

O JOGO

O JOGO

O sudoku consiste em um “grid” 9x9 no qual alguns de seus quadrados estão preenchidos com números de 1 a 9. O objetivo do jogo é completar o grid inteiro sem repetir números na mesma linha, coluna ou “sub-grid” / caixa.

Uma parte importante do sudoku é, ele possui apenas uma solução, dessa forma, existe um número mínimo de quadrados vazios em um mapa de forma que ele ainda seja um sudoku. Este número pode variar de acordo com quais quadrados estão preenchidos.



CONCEITOS BÁSICOS

Quadrado

É a célula do jogo, o espaço onde o número é colocado

Grid

É o mapa do jogo, sempre 9x9, e é dividido em 9 sub-grids de 3x3.

Sub-grid / Box

Conhecido como sub-grid, box ou caixa, possuem tamanho 3x3.

Sudoku

Jogo derivado do “number game” de solução única e mapa 9x9.

03

O Domínio

O Domínio

Types

Foram criados os tipos row e col para representar linha e coluna, digit para representar os números e box para representar os sub-grids.

Constants

Todo jogo possui um número fixo de números, linhas, colunas e sub-grids, as quais são instanciadas como constantes.

```
(define (domain sudoku)
1     (:requirements :hierarchy :typing :strips
:universal-preconditions)
2     (:types row col - position digit box)
3     (:predicates
4          (cell-at-box ?b - box ?c - col ?r - row)
5          (digit-at ?d - digit ?c - col ?r - row)
6          (digit-at-box ?d - digit ?b - box)
7          (filled ?c - col ?r - row)
8          (inc ?a ?b - position))
9     (:constants
10          d1 d2 d3 d4 d5 d6 d7 d8 d9 - digit
11          b0 b1 b2 b3 b4 b5 b6 b7 b8 - box
12          c0 c1 c2 c3 c4 c5 c6 c7 c8 - col
13          r0 r1 r2 r3 r4 r5 r6 r7 r8 - row)
```

O Domínio

All-check

m-all-check e all-check são responsáveis por finalizar o problema, checando se todas as células estão preenchidas.

m-fill-cell

Método principal do código, responsável por chamar todas as outras tasks que servem para colocar um número em um quadrado.

```
        (:task all-check :parameters())
1       (:task fill-cell :parameters (?b - box ?c - col ?r - row ?d - digit))
2       (:task check-box :parameters (?b - box ?d - digit))
3       (:task check-row :parameters (?r - row ?d - digit))
4       (:task check-column :parameters (?c - col ?d - digit))
5       (:method m-all-check
6           :parameters ()
7           :task (all-check)
8           :precondition (and (forall (?r - row ?c - col) (filled ?c ?r))))
9       (:method m-fill-cell
10          :parameters (?b - box ?c - col ?r - row ?d - digit)
11          :task (fill-cell ?b ?c ?r ?d)
12          :precondition (and (not (filled ?c ?r)))
13          :ordered-subtasks (and (check-box ?b ?d) (check-row ?r ?d)
(check-column ?c ?d) (FILL-CELL ?b ?c ?r ?d)))
```

O Domínio

m-check-row

M-check-row e m-check-column utilizam do mesmo princípio, a partir de uma linha ou coluna, checar todas as outras colunas, ou linhas, para checar todas as células da linha, ou coluna.

FILL-CELL

Action responsável por efetivamente colocar o número, utilizando os efeitos de filled, para preencher a célula, digit-at para dizer qual dígito está na célula e digit-at-box para dizer que tal dígito já está na box, bloqueando outras células da box de colocarem ele.

```
(:method m-check-row
1    :parameters (?c0 ?c1 ?c2 ?c3 ?c4 ?c5 ?c6 ?c7 ?c8 - col ?r - row ?d - digit)
2    :task (check-row ?r ?d)
3    :precondition (and (inc ?c0 ?c1) (inc ?c1 ?c2) (inc ?c2 ?c3) (inc ?c3 ?c4) (inc ?c4 ?c5)
(inc ?c5 ?c6) (inc ?c6 ?c7) (inc ?c7 ?c8)
4        (forall (?c - col) (not (digit-at ?d ?c ?r)))))
5  (:method m-check-column
6    :parameters (?c - col ?r0 ?r1 ?r2 ?r3 ?r4 ?r5 ?r6 ?r7 ?r8 - row ?d - digit)
7    :task (check-column ?c ?d)
8    :precondition (and (inc ?r0 ?r1) (inc ?r1 ?r2) (inc ?r2 ?r3) (inc ?r3 ?r4) (inc ?r4 ?r5) (inc
?r5 ?r6) (inc ?r6 ?r7) (inc ?r7 ?r8)
9        (forall (?r - row) (not (digit-at ?d ?c ?r)))))
10  (:method m-check-box
11    :parameters (?b - box ?d - digit)
12    :task (check-box ?b ?d)
13    :precondition (and (not (digit-at-box ?d ?b))))
14  (:action FILL-CELL
15    :parameters (?b - box ?c - col ?r - row ?d - digit)
16    :precondition (and (cell-at-box ?b ?c ?r))
17    :effect (and (digit-at ?d ?c ?r) (digit-at-box ?d ?b) (filled ?c ?r)))
18 )
```

04

O PROBLEMA

O Problema

inc

Predicado responsável por declarar qual linha é adjacente a qual linha, e o mesmo para as colunas.

digit-at

Como o mapa não começa vazio, precisamos instanciar os números que estão presentes, utilizando digit-at e digit-at-box.

```
(define (problem sudoku)
1     (:domain sudoku)
2     (:objects)
3     (:init
4          (inc r0 r1) (inc r1 r2) (inc r2 r3) (inc r3 r4) (inc r4 r5)
(inc r5 r6) (inc r6 r7) (inc r7 r8)
5          (inc c0 c1) (inc c1 c2) (inc c2 c3) (inc c3 c4) (inc c4 c5)
(inc c5 c6) (inc c6 c7) (inc c7 c8)
6          (cell-at-box r0 c0 b0)
7          (cell-at-box r0 c1 b0)
8          (digit-at-box d7 b0)
9          (digit-at d7 r0 c1)
10          (filled r0 c1)
11         (cell-at-box r0 c2 b0)
12          (cell-at-box r0 c3 b1)
13          (digit-at-box d5 b1)
14          (digit-at d5 r0 c3)
15          (filled r0 c3)
```

O Problema

cell-at-box

Maneira utilizada para declarar quais células pertencem a quais box.

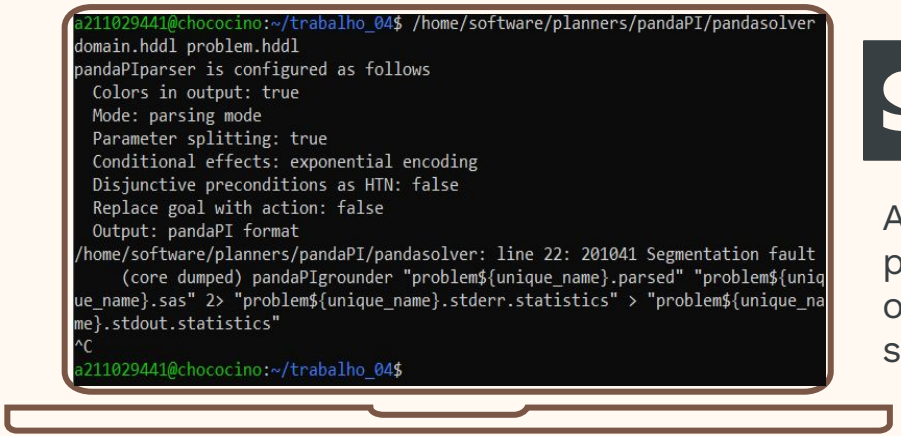
htn

A task na qual precisamos chegar é o all-check.

```
(cell-at-box r8 c7 b8)
29          (cell-at-box r8 c8 b8)
30          (digit-at-box d5 b8)
31          (digit-at d5 r8 c8)
32          (filled r8 c8)
33      )
34      (:htn :tasks (and (all-check)))
```

05

BENCHMARK



Segmentation Fault

Ao tentar executar o código no planejador pandaPI, ocorre um segmentation fault, onde ainda não foi encontrado uma solução para o problema.

06

Parser

O Parser

parser.sh

Este script em Bash tem a função de gerar um arquivo de descrição de problema no formato HDDL (Hierarchical Domain Description Language) para um quebra-cabeça de Sudoku. O processo começa pela criação do arquivo "problem.hddl" e a adição dos cabeçalhos necessários para um arquivo HDDL. Em seguida, o script percorre cada célula do Sudoku, calculando a caixa à qual ela pertence com base nas coordenadas da linha e coluna. O usuário é então solicitado a fornecer os dígitos para preencher cada célula individualmente, e, se um dígito diferente de zero for inserido, o script adiciona as declarações associadas para indicar que a célula foi preenchida. Finalmente, o script fecha as seções iniciadas anteriormente e acrescenta a parte final do arquivo HDDL. O arquivo resultante, "problem.hddl", pode ser utilizado como entrada para um planejador que compreenda a linguagem HDDL, permitindo a resolução hierárquica do problema de Sudoku.

O Parser

```
#!/bin/bash

get_box() {
    local row=$1
    local col=$2
    local box_row=$((row / 3))
    local box_col=$((col / 3))
    local box=$((3*box_row + box_col))
    echo $box
}
> problem.hddl
echo "(define (problem sudoku)" >> problem.hddl
echo -e "\t(:domain sudoku)" >> problem.hddl
echo -e "\t(:objects)" >> problem.hddl
echo -e "\t(:init" >> problem.hddl
echo -e "\t\t(inc r0 r1) (inc r1 r2) (inc r2 r3) (inc r3 r4) (inc r4 r5) (inc r5 r6) (inc r6 r7) (inc r7 r8)" >> problem.hddl
echo -e "\t\t(inc c0 c1) (inc c1 c2) (inc c2 c3) (inc c3 c4) (inc c4 c5) (inc c5 c6) (inc c6 c7) (inc c7 c8)" >> problem.hddl
```

0 Parser

```
for ((i=0; i<9; i++)); do
    for ((j=0; j<9; j++)); do
        box=$(get_box $i $j)
        echo -e "\t\t(cell-at-box r$i c$j b$box)" >> problem.hddl
        read -n 1 digit
        if [[ $digit -ne 0 ]]; then
            echo -e "\t\t(digit-at-box d$digit b$box)" >> problem.hddl
            echo -e "\t\t(digit-at d$digit r$i c$j)" >> problem.hddl
            echo -e "\t\t(filled r$i c$j)" >> problem.hddl
        fi
    done
done
echo -e "\t)" >> problem.hddl
echo -e "\t(:htn :tasks (and (all-check)))" >> problem.hddl
echo ")" >> problem.hddl
```

07

PROBLEM GENERATOR

O Gerador

generator.c

O arquivo "generator.c" é um código em linguagem C é um gerador de problemas de Sudoku. Utilizando um algoritmo de resolução baseado em backtracking, o programa preenche uma grade vazia para criar um desafio de Sudoku. O nível de dificuldade do problema gerado é determinado pelo usuário. O código inclui também a manipulação de bits para otimizar a verificação de números possíveis em cada célula. O programa gera um arquivo de saída contendo o problema de Sudoku resultante.

sudoku.h

O arquivo "sudoku.h" desempenha um papel fundamental no gerador de problemas de Sudoku em linguagem C. Ele define constantes essenciais, como o tamanho da grade do Sudoku e o número de soluções pré-definidas disponíveis. Além disso, o arquivo apresenta soluções válidas de Sudoku, representadas por matrizes como '*solved_sudoku0*' a '*solved_sudoku9*'. Essas soluções são utilizadas pelo programa principal para criar problemas variados. A matriz sudoku dentro do arquivo representa o estado atual do Sudoku a ser gerado.

generator.c

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <time.h>
#include "sudoku.h"

#define ALL_BITS 1022 // 1022 = 1111111110 binary
#define SET_BIT(num, bit)  (num |= (1<<(bit)))
#define CHECK_BIT(num, bit) (num & (1<<(bit)))

static int is_solved;
int solved_sudoku[SIZE][SIZE]=
{
        0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,
};
```

```
int count_set_bits(int num) {
        int count = 0;
        for (int i = 0; i < 32; i++)
                if (CHECK_BIT(num, i)) count++;
        return count;
}

void display_sudoku(const char *filename) {
        FILE *file = fopen(filename, "w");
        if (!filename) {fprintf(stderr, "Error to open file %s\n", filename); return;}
        for (int i = 0; i < SIZE; i++) {
                for (int j = 0; j < SIZE; j++)
                        fprintf(file, "%d", sudoku[i][j]);
                fprintf(file, "\n");
    }
        fclose(file);
}
```

generator.c

```
int solve_sudoku(int level, int x, int y, int val) {
    int best_x = -1, best_y, best_mask, best_bits = 0, bits;
    if (is_solved) return 0;
    if (val >= 1 && val <= 9) sudoku[x][y] = val; // Make the
move
    is_solved = 1;
    for (int i = 0; i < SIZE; i++)
        for (int j = 0; j < SIZE; j++)
            // For each empty cell on the board
            if (sudoku[i][j] == 0) {
                int mask = 0;
                is_solved = 0;
                // Check row
                for (int k = 0; k < SIZE; k++)
                    if (sudoku[i][k] != 0)
                    SET_BIT(mask, sudoku[i][k]);
                // Check column
                for (int k = 0; k < SIZE; k++)
                    if (sudoku[k][j] != 0)
                        SET_BIT(mask, sudoku[k][j]);
                // Check box
                int box_i = (i/3)*3, box_j = (j/3)*3;
                for (int k = box_i; k < box_i+3; k++)
                        for (int m = box_j; m < box_j+3; m++)
                            if (sudoku[k][m] != 0)
                                SET_BIT(mask, sudoku[k][m]);
                    if (mask == ALL_BITS) goto unsolvable;
                    if ((bits = count_set_bits(mask)) > best_bits)
                        best_bits = bits, best_x = i, best_y = j, best_mask = mask;
                }
    if (is_solved) return 0;
    // Make the best move found...
    if (best_x != -1)
        for (int k = 1; k <= 9; k++)
            if (CHECK_BIT(best_mask, k) == 0)
                solve_sudoku(level+1, best_x, best_y, k);
unsolvable:
    if (val >= 1 && val <= 9)
        sudoku[x][y] = 0; // Undo the move
    return 1;
}
```

generator.c

```
void generate_sudoku() {
    int k = 0, level;
    char letter;
    printf("Enter the level of the sudoku: (0-easy 1-medium 2-hard 3-crazy)\n");
    scanf("%c", &letter);
    switch (letter) {
        case '0': level = 51; break;
        case '1': level = 41; break;
        case '2': level = 31; break;
        case '3': level = 23; break;
        default: level = 31; break;
    } do {
        memset(sudoku, 0, sizeof(int)*SIZE*SIZE);
        for (int i = 0; i < SIZE; i++)
            for (int j = 0; j < SIZE; j++)
                if (rand() % 100 < level+k)
                    sudoku[i][j] = solved_sudoku[i][j];
        k++;
    } while (solve_sudoku(0, 0, 0, 0));
    display_sudoku("sudoku");
}
```

```
void initialize_sudoku() {
    int index = rand() % NUM_SUDOKUS;
    switch (index) {
        case 0:
            memcpy(solved_sudoku, solved_sudoku0, sizeof(solved_sudoku)); break;
        case 1:
            memcpy(solved_sudoku, solved_sudoku1, sizeof(solved_sudoku)); break;
        case 2:
            memcpy(solved_sudoku, solved_sudoku2, sizeof(solved_sudoku)); break;
        case 3:
            memcpy(solved_sudoku, solved_sudoku3, sizeof(solved_sudoku)); break;
        case 4:
            memcpy(solved_sudoku, solved_sudoku4, sizeof(solved_sudoku)); break;
        case 5:
            memcpy(solved_sudoku, solved_sudoku5, sizeof(solved_sudoku)); break;
        case 6:
            memcpy(solved_sudoku, solved_sudoku6, sizeof(solved_sudoku)); break;
        case 7:
            memcpy(solved_sudoku, solved_sudoku7, sizeof(solved_sudoku)); break;
        case 8:
            memcpy(solved_sudoku, solved_sudoku8, sizeof(solved_sudoku)); break;
        case 9:
            memcpy(solved_sudoku, solved_sudoku9, sizeof(solved_sudoku)); break;
        default:
            fprintf(stderr, "Error: Invalid sudoku index\n");
            exit(EXIT_FAILURE);
    }
}
```

generator.c

```
int main() {
        srand(time(NULL));
        initialize_sudoku();
        generate_sudoku();
        return 0;
}
```

OBRIGADO!

Bruno Ribeiro - Danilo Carvalho - Igor Penha - Lucas Gobbi