Primeiro EP de MAC239
Resolvedor de Sudoku

Ruan Costa nº USP 7990929 e
Vinícius Silva nº USP 7557626

21 de setembro de 2013

Sumário

1 A Lógica do Sudoku

O primeiro desafio do EP era codificar as regras do Sudoku em cláusulas lógicas. Para isso, usamos o modelo do problema das n rainhas. Naquele caso as regras eram:

- Tem de haver pelo menos uma rainha em todas as linhas.
- Pode haver apenas uma rainha em cada linha.
- Pode haver apenas uma rainha em cada coluna.
- Pode haver apenas uma rainha em cada diagonal.

No caso do Sudoku há a complicação de não podermos trabalhar com apenas dois estados em cada "espaço" (combinação de linha e coluna). A solução foi tratar cada espaço como um vetor de 9 casos. Desta forma trabalhamos o sudoku em três dimensões: linha x coluna x número. Portanto o conjunto dos átomos das cláusulas é definido da seguinte forma:

$$C = \{\vec{x} \in N \times N \times N : 1 \leq x_i \leq 9\}$$

Podemos também definir a função "ligado":

$$l : N \times N \times N \longrightarrow \{0, 1\}$$

$$l(\vec{x}) = \begin{cases} 1, & \texttt{se o espaço } (x_1, x_2) \texttt{ contém o número em } x_3 \\ 0, & \texttt{se o espaço } (x_1, x_2) \texttt{ não contém o número em } x_3 \end{cases}$$

Todos os vetores tais que $l(\vec{x}) = 1$, chamaremos de ligado, e os outros chamaremos de desligado. Assim, podemos traduzir as regras do Sudoku sa guinte forma:

- Tem de haver pelo menos um vetor ligado em cada espaço.
- Pode haver apenas um vetor ligado em cada espaço.
- Pode haver apenas um vetor ligado em cada coluna com o mesmo número.
- Pode haver apenas um vetor ligado em cada linha com o mesmo número.
- Pode haver apenas um vetor ligado em cada "grid" com o mesmo número.

Se considerarmos cada vetor $\vec{x} \in C$ um átomo, poderiamos escrever essas regras na forma clausal com as seguintes clausulas:

- $\bigwedge_{k=1}^{9} \bigwedge_{j=1}^{9} \bigvee_{i=1}^{9} (k, j, i)$
- $\bigwedge_{r=1}^{9} \bigwedge_{k=1}^{9} \bigwedge_{j=1}^{9} \bigwedge_{i=j+1}^{9} (\neg(r, k, j) \vee \neg(r, k, i))$
- $\bigwedge_{r=1}^{9} \bigwedge_{k=1}^{9} \bigwedge_{j=1}^{9} \bigwedge_{i=j+1}^{9} (\neg(r, j, k) \vee \neg(r, i, k))$
- $\bigwedge_{r=1}^{9} \bigwedge_{k=1}^{9} \bigwedge_{j=1}^{9} \bigwedge_{i=j+1}^{9} (\neg(j, r, k) \vee \neg(i, r, k))$
- Para cada grid de centro (r,s): $\bigwedge_{k=1}^{9} \bigwedge_{j=1}^{9} \bigwedge_{i=j+1}^{9} (\neg(j', j'', k) \vee \neg(i', i'', k))$

  tais que $\begin{cases} j' = (j-1)/3 + r - 1 \\ j'' = (j-1)\texttt{mod(3)} + s - 1 \\ i' = (i-1)/3 + r - 1 \\ i'' = (i-1)\texttt{mod(3)} + s - 1 \end{cases}$

2 Implementação

Com as cláusulas definidas, o próximo passo foi implementar um programa que escreve o arquivo de entrada para o Minisat. Escolhemos a linguagem Perl devido à facilidade em manipular strings e arquivos. Caso o Minisat não esteja na pasta do programa, é preciso mudar a variável "$minisat_path" com o caminho correto. A lógica utilizada foi bem simples, o programa segue os seguntes passos:

1. Abre o arquivo de entrada.
2. Cria uma matriz $9 \times 9 \times 9$ enumerada de 1 a 729.
3. Imprime em um arquivo .cnf os valores correspondentes aos campos fixos do Sudoku.
4. Imprime no mesmo arquivo .cnf as clausulas mapeadas na matriz, usando a lógica da sessão anterior.
5. Chama o Bash para executar o minisat.
6. Imprime a saída:
    a) Caso não haja solução, imprime "Insatizfazível".
    b) Caso haja solução, imprime "Satizfazível"e uma solução.