Relatório Exercício Programa I - Sudoku

Fellipe Souto Sampaio *
Gervásio Protásio dos Santos Neto †

22 de setembro de 2013

MAC 0239 Métodos Formais em Programação
Prof. Marcelo Finger

Instituto de Matemática e Estatística - IME USP
Rua do Matão 1010
05311-970 Cidade Universitária, São Paulo - SP

*Número USP: 7990422 e-mail: fellipe.sampaio@usp.com
†Número USP: 7990996 e-mail: gervasio.neto@usp.br

1 Introdução

Este relatório pretende explicar a implementação do exercício programa I, explicando desde modelagem do problema a implementação do programa o descreve. A proposta do exercício é desenvolver um programa que descreva um jogo sudoku através de uma *Formula Normal Conjuntiva* e resolver um dado jogo fornecido pelo usuário, devolvendo uma resposta que o satisfaz.

2 Modelo do Sudoku

O sudoku modelado é de ordem 3 (9x9), começemos delimitando quais são as regras que cada quadrado da grade deve respeitar, que são as seguintes:

- Na linha cada quadrado deve possuir apenas 1 número entre 1 a 9, de forma que todos os números da linha sejam distintos entre sí.
- Na coluna cada quadrado deve possuir apenas 1 número entre 1 a 9, de forma que todos os números da coluna sejam distintos entre sí.
- Em cada região 3x3 todo quadrado deve possuir apenas 1 número entre 1 a 9, de forma que todos os números da linha sejam distintos entre sí.
- Em cada quadrado pode-se colocar apenas 1 número, sendo este de 0 a 9.

Admitindo as regras enumeradas acima como "axiomas"para construção de um sudoku, podemos enumerar algumas consequências dessas definições:

- Os números estarão dispostos em linhas, colunas e regiões através da representação de apenas um elemento por quadrado.
- Um quadrado nunca pode receber mais que um elemento.
- Um sudoku com dicas restringe a quantidade de candidatos possíveis para um dado quadrado.

Com todas essas regras de construção a modelagem do problema através de uma *Formula Normal Conjuntiva* torna-se extremamente útil. Na modelagem proposta teremos as seguintes definições:

- Existem 729 variáveis, sendo um grupo de nove valores possíveis para cada um dos 81 quadrados.
- Admitiu-se contar de 1 a 729 as variáveis, sempre no sentido da esquerda para direita e de cima para baixo.
- Um sudoku com dicas restringe a quantidade de candidatos possíveis para um dado quadrado.

3 Implementação do Modelador

O modelador foi escrito na linguagem de programação Perl, por sua praticidade em lidar com listas e tabelas de hashing. O arquivo principal que modela o sudoku é o módulo nxnArray.pm. Neste módulo está declarado a classe de mesmo nome do arquivo, a qual possui como atributos :

- Sudoku - Uma matriz 9x9 que guarda o sudoku não resolvido.
- Input - Um array com a entrada fornecida pelo usuário.
- FNC - Uma tabela de hashing, no qual suas chaves são todas as cláusulas que compõem a FNC.
- QntClausules - Um contador de quantas cláusulas são criadas ao longo do programa.
- Centros - A coordenada do quadrado central de todas as 9 regiões do sudoku.

As cláusulas serão escritas no arquivo de saida utilizando 8 métodos diferentes, nos quais cada um irá modelar um tipo de restrição.

Nos métodos *highlanderColumn, highlanderLine e subSquare* cada variável atômica será combinada dois-a-dois com todas as outras representante de um dado número. Esta combinação impede que exista mais de um representante de um número em uma mesma região, linha e coluna. Tem-se o seguinte exemplo para ilustrar a ideia:

Seja $X_{1,1} \ldots X_{1,9}$ todas as variáveis que representam um X $\in [1 \ldots 9]$, as cláusulas serão da forma:

$\neg X_{1,1}$ V $\neg X_{1,2}$

$\neg X_{1,1}$ V $\neg X_{1,3}$

$\vdots$

$\neg X_{1,1}$ V $\neg X_{1,9}$

$\neg X_{1,2}$ V $\neg X_{1,3}$

$\vdots \neg X_{1,2}$ V $\neg X_{1,9}$

$\vdots$

$\neg X_{1,8}$ V $\neg X_{1,9}$

$\neg X_{2,1}$ V $\neg X_{2,2}$

$\vdots$

$\neg X_{2,1}$ V $\neg X_{2,9}$

$\vdots$

$\neg X_{9,1}$ V $\neg X_{9,2}$

$\vdots$

$\neg X_{9,1}$ V $\neg X_{9,9}$

$\vdots$

$\neg X_{9,8}$ V $\neg X_{9,9}$

Cada combinação em um dado X $\in [1 \ldots 9]$ gera um total de 36 cláusulas e 324 para cada número, repetindo isso para cada linha, coluna ou subquadrado tem-se um total de 2916, por fim juntando os três casos, 8748 cláusulas.

3.1 Algoritmo Para Linhas e Colunas

O algoritmo para escrita das cláusulas de linhas e colunas funciona sobre uma matriz unidade (9x9) A*z, onde z varia em $\in [1 \ldots 9]$. A permutação citada é realizada fixando um elemento $X_{i,j}$ e interando outro $Y_{i+k,j+l}$ dependendo do caso. Para recuperar o átomo correspondente da cláusula é utilizada a seguinte relação:

$$9 \times j + 81 \times i + z$$

, onde o par (i,j) será a coordenada da matriz e z o coeficiente multiplicatório de A. Tem-se como exemplo:
¬1 V ¬10
¬1 V ¬19
⋮
¬1 V ¬73
¬2 V ¬11
⋮
¬2 V ¬74
⋮

3.2 Algoritmo Para Região

O método *subsquate* é responsável por construir as cláusulas que forçam a unicidade dos números nas regiões 3x3.

O que fazemos é que usamos os centros das regiões (armazenados previamente como um atributo da classe) para percorrer a região. Já que sabiamos o centro *(l,k)*, os pontos da região seriam { ($l \pm q$, $k \pm p$) | $q, p \in$ [-1,…,1] }.

De posse de tal informação e da fórmula mencionada na Seção 3.1 fomos capazes de construir, de forma similar a como fizemos para linhas e colunas, as cláusulas das regiões.

Por exemplo, na primeira região, as clásulas que forçam que cada região possa conter o número 1 apenas umas vez seriam as seguintes:

¬1 V ¬82
¬1 V ¬163
¬1 V ¬10
¬1 V ¬91
¬1 V ¬172
¬1 V ¬19
¬1 V ¬100
¬1 V ¬181

3.3 Algoritmo da Permutação dos candidatos

Cada quadrado pode conter um X $\in [1 \ldots 9]$, e caso este X tenha um valor delimitado todos os outros candidatos serão falsos. Para isso utiliza-se a mesma ideia de permutação dois-a-dois, só que neste caso entre os candidatos de um dado quadrado. Por exemplo:

Seja $Q_{1,1}$ o quadrado de coordenada (1,1), as cláusulas de seus prováveis candidatos são da forma:
$\neg 1 \lor \neg 2$
$\neg 1 \lor \neg 3$
$\vdots$
$\neg 1 \lor \neg 9$
$\neg 2 \lor \neg 3$
$\vdots$
$\neg 2 \lor \neg 9$
$\vdots$
$\neg 8 \lor \neg 9$

Pelas cláusulas acima se $Q_{1,1}$ receber o valor 1, todos os outros candidatos devem receber 0, para que a cláusula continue sendo verdadeira.

3.4 Algoritmos para Existência de Números

Os algoritmos descritos até este pontos garantem apenas que exitem *no máximo* um número de cada tipo em cada linha, região e coluna e que não se atribua a um mesmo quadrado números distintos. Contudo, se tivessemos apenas essas cláusulas, um sudoku vazio seria uma resposta válida.

Isso ocorre pois essas cláusulas não forçam que haja *pelo menos um* de cada número em cada linha, coluna ou região. E foi para contornar este problema, desenvolvemos os métodos *columnExistence, lineExistence* e *regionExistence.*

Seu funcionamento é similar ao dos algoritmos que garantem a unicidade, contudo, ao invés de construírmos fórmular dois a dois, criamos, cláusulas que forçam o aparecimento de um número.

Por exemplo, o método *lineExistence* teria como uma das cláusulas de seu output a seguinte fórmula:

$1 \lor 10 \lor 19 \lor 28 \lor \lor 37 \lor 46 \lor 55 \lor 64 \lor 73$

Que significa que ao menos um destes átomos precisa ser verdade. Como estes átomos representam a existência do número 1 na primeira linha, o que essa fórmula diz essencialemte é que a primeira linha deve conter o número 1 ao menos uma vez.

As cláusulas geradas por estes métodos, combinadas com as geradas pelos métodos descritos anteriormente garantem que cada número aparece uma vez, e somente uma vez.

3.5 Inserção das Dicas

Após a construção do modelo para um sudoku abstrato o método *insertTips* realiza a tarefa de inserir as dicas do sudoku fornecido pelo usuário. Através da inserção das dicas o resolvedor SAT é forçado a fornecer uma valoração que respeite estas delimitações

4 Interface Modelador-minisat

O módulo *nxnArray.pm* contém os métodos geradores das cláusulas em formato CNF DIMACS que, quando processados pelo SAT-SOLVER (no nosso caso, foi usado o minisat), produzirá uma uma resposta para o Sudoku.

Além deste módulo, utilizamos mais dois arquivos: *main.pl* e *filtro.pl*.

O aqruivo *filtro.pl* recebe com entrada um arquivo que contém a saída do minisat (ou seja, um arquivo que contém a valoração que constitui uma resposta para o problema dado), processa este arquivo e imprime na saída padrão o sudoku resolvido com a valoração produzida pelo minisat.

Lembramos que para gerar um arquivo com a valoração utilizando o minisat basta executar:

```
./minisat <arquivoCNF> <arquivoDeSaida>
```

Já *main.pl* é arquivo principal do EP, sendo o programa que dá o output *de facto* do programa e é melhor descrito na Seção 5 - Integração e Funcionamento.

5 Integração e Funcionamento

Todos os códigos-fonte em Perl são conectados por meio do arquivo *main.pl*. Ele deve ser invocado da seguinte forma:

```
./main.pl <arquivoComSudoku> <nomeDoCNF> <nomeDaSaidaDoMinisat>
```

O arquivo com a configuração do sudoku já deve existir no diretorio em que *main.pl* está sendo invocado. O arquivo .cnf, bem como o arquivo de saída do minisat, serão criados durante a execução de *main.pl* e continuarão no diretório, podendo ser reutilizados. Vale lembrar que *main.pl* e o executável do minisat (que deve chamar-se minisat) devem estar no mesmo diretório para que a execução aconteça corretamente.

Nele instanciamos o módulo *nxnArray.pm* e utilizamos os métodos deste para ler a entrada contendo a configuração do Sudoku e escrever as cláusulas pertinentes (descritas em seções anteriores) em um arquivo .cnf . Para esta escrita utilizamos dois *Filehandles*, INPUT e OUTPUT.

INPUT é o arquivo de entrada, que será lido e modelado. Já OUTPUT referencia o arquivo .cnf, no qual serão escritas as cláusulas.

Uma vez que terminamos de escrever as clásulas, usamos a funcionalidade de aspas invertidas do Perl (“) para executarmos um comando bash. Nesse caso, o comando executado é:

```
./minisat $output $answer
```

Onde $output guarda o nome do arquivo .cnf e $answer é o nomoe do arquivo em que o minisat escreverá uma valoração que satisfaz as cláusulas para o Sudoku modelado.

Por fim, usamos o comando system() do Perl que, similarmente as aspas invertidas, executa um comando bash, mas que permite envio de informação para a saída padrão. Utilizando esse comando, invocamos o seguinte script:

```
./filtro.pl $answer
```

Ou seja, passamos como argumento de *filtro.pl* o arquivo de saída do minisat. O resultado é que no STDOUT é então impresso o Sudoku resolvido, conforme o funcionamento de *filtro.pl*.

O resultado final da evocação de *main.pl* é um arquivo .cnf que contém as cláusulas que modelam o Sudoku de entrada, um arquivo contendo a saída do minisat e, na sáida padrão, o Sudoku resolvido.

6 Conclusão e Exemplos Testados

Através da modelagem do sudoku via FNC foi possível criar um método para simular seu comportamento e uma maneira viável e rápida para resolver um dado sudoku. Em testes realizados a performace do programa demonstrou-se muito satisfatória. Podemos citar como exemplos testados os seguintes:

Entrada

5 3 0 0 7 0 0 0 0
6 0 0 1 9 5 0 0 0
0 9 8 0 0 0 0 6 0
8 0 0 0 6 0 0 0 3
4 0 0 8 0 3 0 0 1
7 0 0 0 2 0 0 0 6
0 6 0 0 0 0 2 8 0
0 0 0 4 1 9 0 0 5
0 0 0 0 8 0 0 7 9

Problem Statistics

Number of variables: 729
Number of clauses: 5759
Parse time: 0.01 s
restarts : 1 conflicts : 0 (0 /sec)
decisions : 1 (0.00 % random) (125 /sec)
propagations : 729 (91125 /sec)
conflict literals : 0 (-nan % deleted)
Memory used : 21.00 MB
CPU time : 0.008 s
SATISFIABLE

Saida

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

Entrada

8 0 0 0 0 0 0 0 0
0 0 3 6 0 0 0 0 0
0 7 0 0 9 0 2 0 0
0 5 0 0 0 7 0 0 0
0 0 0 0 4 5 7 0 0
0 0 0 1 0 0 0 3 0
0 0 1 0 0 0 0 6 8
0 0 8 5 0 0 0 1 0
0 9 0 0 0 0 4 0 0

Problem Statistics

Number of variables: 729
Number of clauses: 7421
Parse time: 0.01 s
Simplification time: 0.00 s restarts : 1 conflicts : 93 (7750 /sec)
decisions : 113 (0.00 % random) (9417 /sec)
propagations : 7123 (593583 /sec)
conflict literals : 722 (12.17 % deleted)
Memory used : 22.00 MB
CPU time : 0.012 s
SATISFIABLE

Saida

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

Obs: Na maioria dos sudokus presente em jornais e revistas é atribuido um nível de dificuldade de 1 até 5 estrela, ou seja, do mais fácil ao mais difícil. O sudoku acima foi criado pelo matemático finlandês Arto Inkala, e pela classificação usual teria onze estrelas.