Sat Solving

Pedro Costa

March 2021

Introdução

Senti que seria possível ter uma apresentação mais clara das minhas resoluções utilizando LaTeX do que enviando imagens do caderno e como me pareceu haver liberdade no que escolher (pelo que li na BB), farei então a explicação dos exercícios ao longo deste relatório.

1 - Puzzle

1 - Variáveis proposicionais e conversão para CNF

Como variáveis proposicionais, assumi:
A_1 - ir ao sábado
A_2 - usar chapéu
A_3 - usar anel
A_4 - ser adulto
A_5 - ser loiro

Assim, podemos então exprimir cada regra à custa das variáveis anteriores, mostrando já o seu valor em CNF:

- Sócios loiros não podem ir ao sábado:

$$A_5 \Rightarrow \neg A_1 \equiv \neg A_5 \vee \neg A_1$$

- Quem não for adulto têm que usar chapéu:

$$\neg A_5 \Rightarrow A_2 \equiv A_5 \vee A_2$$

- Cada sócio usa anel ou não usa chapéu:

$$A_2 \vee \neg A_3$$

- Um sócio vai ao sábado se e só se é adulto:

$$A_1 \Leftrightarrow A_4 \equiv \neg A_1 \vee A_4 \wedge \neg A_4 \vee A_1$$

- Todos os sócios adultos têm que usar anel:

$$A_4 \Rightarrow A_3 \equiv \neg A_4 \vee A_3$$

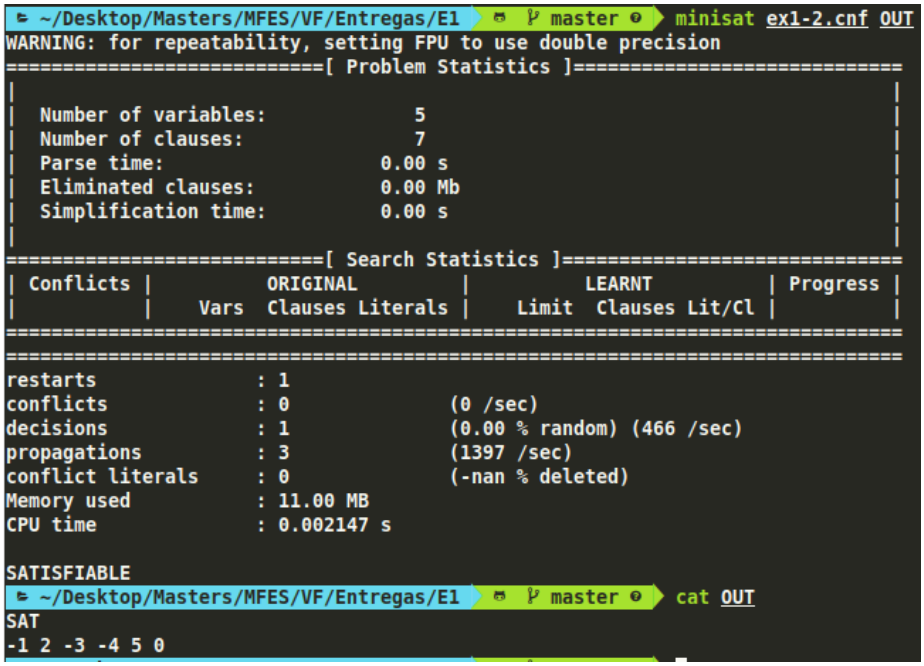
- Quem usa anel tem que ser loiro:

$$A_3 \Rightarrow A_5 \equiv \neg A_3 \vee A_5$$

2 - Comprovar que o problema é persistente

Recorrendo a um SAT solver, como pedido, obtemos uma possível solução para o problema, o que confirma que o mesmo é consistente.

Note-se ainda que a solução obtida indica que é possível um sócio usar chapéu e ser loiro!



3 - Utilizar o SAT solver para responder a questões

O método utilizado aqui passa por adicionar cláusulas à formulação anterior de forma a verificar se o problema em questão é verdadeiro ou não. Assim, indico então o que foi adicionado para perceber cada pergunta.

- Quem usa anel não pode ir ao sábado?
  A cláusula a adicionar aqui é a negação do que queremos saber, ou seja:

$$\neg(A_3 \Rightarrow \neg A_1) \equiv \neg(\neg A_3 \vee \neg A_1) \equiv A_3 \wedge A_1$$

O resultado obtido é *UNSAT* pelo que podemos então afirmar que a afirmação é verdadeira.

- Pode um sócio de chapéu ser loiro?
  Ainda que se possa efetivamente testar esta pergunta, realço que na resposta à pergunta 1.2) já obtemos um cenário que confirma que um sócio de chapeú pode ser loiro. No entanto, se quissesemos testar isto de qualquer forma, bastava adicionar as seguintes condições:

$$A_2 \wedge A_5$$

- Afinal a associação não pode ter sócios adultos?
  A cláusula a adicionar aqui é, mais uma vez, a negação do que queremos saber, ou seja:

$$\neg(\neg A4) \equiv A4$$

  O resultado obtido é *UNSAT* pelo que podemos então afirmar que a afirmação é verdadeira, ou seja, não pode haver sócios adultos.

2 - Sudoku

1 - Modelação do problema (já em CNF)

Para modelar o jogo de Sudoku como um problema SAT, recorri a 4 conjuntos de restrições, sendo estes:

- Não haver repetidos por linha:
  Para resolver esta restrição fixamos cada linha e cada número, dizendo que o número deve existir em pelo menos uma das colunas, ou seja:

$$\bigwedge l \in \{0..N^2\}, n \in \{0..N^2\} :: (x_{l1n} \vee x_{l2n} \vee ... \vee x_{lcn})$$

- Não haver repetidos por coluna:
  Identicamente ao caso anterior, fixamos cada número e, desta vez, cada coluna, dizendo que o número deve existir em pelo uma das linhas, ou seja:

$$\bigwedge c \in \{0..N^2\}, n \in \{0..N^2\} :: (x_{1cn} \vee x_{2cn} \vee ... \vee x_{lcn})$$

- Não haver repetidos por submatriz:
  Este caso acaba por ser mais complexo que os anteriores, mas pode também ser resolvido forçando apenas cada número a cada submatriz. Ou seja, de forma pouco rigorosa:

$$\bigwedge S \in Submatrizes :: \forall (l, c) \in S :: (x_{lc1} \vee x_{lc2} \vee ... \vee x_{lcn})$$

- Todas as células terem um, e apenas um, número:
  Cada entrada tem de ter pelo menos um valor, pode ser expresso com:

$$\bigwedge l \in \{0..N^2\}, c \in \{0..N^2\} :: (x_{lc1} \vee x_{lc2} \vee ... \vee x_{lcn})$$

  Cada entrada pode ter no máximo um valor, pode ser expresso com:

$$\bigwedge l \in \{0..N^2\}, c \in \{0..N^2\}, n \in \{0..N^2\}, n' \in \{0..N^2\} - \{n\} :: \neg x_{lcn} \vee \neg x_{lcn'}$$

- Preencher os espaços já ocupados:
  Este conjunto é absolutamente trivial, bastando adicionar como restrição cada casa que já se encontra ocupada com o valor em causa, ou seja, para qualquer (l,c,n):

$$x_{lcn}$$

3 - Resolução do problema com um SAT Solver

Para gerar o ficheiro DIMACS CNF criei, como sugerido, um programa. A linguagem que escolhi foi Haskell, tendo recorrido maioritariamente a listas de compreensão para criar as várias cláusulas.

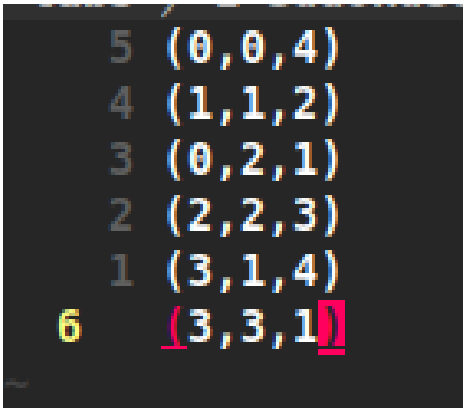
```haskell
module SudokuSolver where

import Data.List
import Control.Monad
import Control.Arrow

type Var = (Int,Int,Int,Int) -- (l,c,n,sign)
data Setup = S { ls :: [Int], cols :: [Int], range :: [Int], mat :: [Var] } deriving (Show)
newtype SatSol = SS { unSS :: [[Int]] }

instance Show SatSol where
    show (SS m) = unlines $ map ((++ "0"). concatMap ((++ " ") . show)) m

-- Ensure that there are no repeated per line, column or submatrix and exactly 1 number per square
allFilled, noRepLine, noRepCol, noRepSquare, noRepSub :: Setup -> [[Var]]
allFilled (S l c r _) = [ [ (l',c',n',1) | n' <- r ] | l' <- l, c' <- c ]
noRepLine (S l c n _) = [ [ (l',c',n',1) | c' <- c ] | l' <- l, n' <- n ]
noRepCol  (S l c n _) = [ [ (l',c',n',1) | l' <- l ] | c' <- c, n' <- n ]

noRepSquare (S l c n _) = concat [ concat [ perNumber (l',c',n') | n' <- n ] | l' <- l, c' <- c] where
    perNumber t@(l',c',n') = [ [(l',c',n',-1), (l',c',n'',-1)] | n'' <- n \\ [n'] ]

noRepSub (S l c n _) = concat [  [ map (\(a,b) -> (a,b,n',1) ) r' | n' <- n ] | r' <- r ] where
    r = map createPairs . (uncurry zip . (id &&& tail)) $ enumFromThenTo 0 (isqrt $ length l) (length l)
    isqrt = floor . sqrt . fromIntegral
    createPairs (lb,ub) = [(a,b) | a <- [lb..ub-1], b <- [lb..ub-1]]

identifyVar :: (Int,Int,Int) -> Var -> Int
identifyVar (ln, cn, nn) (l,c,n,s) = s * ((l * (cn*nn)) + (c * cn) + n)

reConvert :: Int -> Int -> (Int,Int,Int)
reConvert n i = (line,col,number) where
    (line,lm) = ((i-1) `div` (n*n), (i-1) `mod` (n*n))
    (col, cm) = (max (lm `div` n) 0, lm `mod` n)
    number = cm + 1

readSol :: String -> IO ()
readSol = readFile >=> mapM_ (putStrLn . show) . map (reConvert 4) . filter (>0) . init . map (\x -> read x :: Int) . words . (!!1) . lines

main n file = do
    baseSetup <- map (\(a,b,c) -> (a,b,c,1)) . map read . lines <$> readFile file
    let setup = S [0..n-1] [0..n-1] [1..n] baseSetup
    let conds = concatMap ($ setup) [allFilled, noRepSub, noRepSquare, noRepCol, noRepLine, map (:[]) . mat]
    let satReady = SS $ nubBy (\l1 l2 -> l1 == l2 || l1 == reverse l2) $ map (map (identifyVar (n,n,n))) conds
    let (nvars, nclauses) = (n*n*n, length $ unSS satReady)
    writeFile "sudoku.cnf" ("p cnf " ++ show nvars ++ " " ++ show nclauses ++ "\n" ++ show satReady)
```

Figure 1: SAT Model Generator

O programa é capaz de ler uma solução de um ficheiro de configuração composto por trios com o formato das variáveis proposicionais, gerando um

ficheiro **sudoku.cnf** com o formato DIMACS CNF pronto a ser utilizado num SAT Solver.

Para o exemplo disponibilizado junto deste problema, obtém-se exatamente a mesma solução que a professora obteu, como se pode ver abaixo:



Ficheiro de configuração usado

```
*SudokuSolver Control.Monad> readSol "sudoku.sol"
(0,0,4)
(0,1,3)
(0,2,1)
(0,3,2)
(1,0,1)
(1,1,2)
(1,2,4)
(1,3,3)
(2,0,2)
(2,1,1)
(2,2,3)
(2,3,4)
(3,0,3)
(3,1,4)
(3,2,2)
(3,3,1)
```

Figure 2: Solução obtida (interpretada do output do SAT solver)