Metodos Formais em Engenharia de Software (2022/2023)

SAT solving

Comece por instalar um SAT solver. Recomendamos que instale o MiniSat.

O formato DIMACS CNF

A fórmula proposicional $A_1 \wedge (A_1 \vee P) \wedge (\neg A_1 \vee \neg P \vee A_2) \wedge (A_1 \vee \neg A_2)$
encontra-se já em CNF e pode ser escrita no formato DIMACS como se segue (`example.cnf`):

```
p cnf 3 4
1 0
1 3 0
-1 -3 2 0
1 -2 0
```

Exemplifica-se a invocação de um solver com o MiniSat:

```
$ minisat example.cnf OUT
```

A solução calculada é:

```
SAT
 1 -2 -3 0
```

ou seja, $A_1 = 1$, $A_2 = 0$ e $P = 0$.

Exercício 1 (Experimentando um SAT solver) Invoque um SAT solver com os ficheiros (em formato DIMACS CNF) `meeting.cnf`, `sat-100v429c.cnf` e `unsat-175v753c.cnf` e analise a resposta do solver.

Exercício 2 (Classificação de fórmulas) Com a ajuda de um SAT solver, responda às seguintes questões:

1. A fórmula $A \vee B \to A \vee C$ é satisfazível? É uma contradição? É refutável? É válida?
2. A fórmula $(A \to B \vee C) \wedge \neg(A \wedge \neg B \to C)$ é satisfazível? É válida? É uma contradição? É refutável?
3. A fórmula $(\neg A \to \neg B) \to (\neg A \to B) \to A$ é válida? É refutável? É uma contradição? É satisfazível?

Exercício 3 (Puzzle do unicórnio) Considere o seguinte enigma:

- *If the unicorn is mythical, then it is immortal.*
- *If the unicorn is not mythical, then it is a mortal mammal.*
- *If the unicorn is either immortal or a mammal, then it is horned.*
- *The unicorn is magical if it is horned.*

- *Is the unicorn magical? Is it horned? Is it mythical?*

1. Para o resolver considere 5 variáveis proposicionais, correspondentes a 5 propriedades dos unicórnios, e começe por completar o seguinte ficheiro no formato DIMACS, `unicornpuzzle.cnf`, com a descrição das restrições do puzzle.

   ```
   c The Unicorn puzzle
   c
   c  1 mythical?
   c  2 immortal?
   c  3 mammal?
   c  4 horned?
   c  5 magical?
   c
   p cnf 5 ???
   (...)
   ```

   Depois invoque depois o seu SAT solver para confirmar que o puzzle é satisfazível.

2. Como o puzzle é satisfazível, o SAT solver irá indicar um modelo (i.e., uma atribuição de valores às variáveis que tornam todas as restrições do puzzle verdadeiras). Suponha por exemplo que na invocação acima o solver encontrou a solução seguinte:

   ```
   SAT
   -1 -2 3 4 5 0
   ```

   Considerada como solução (modelo), o significado da última linha é:

   $$\neg mythical \land \neg immortal \land mammal \land horned \land magical$$

   Para obtermos uma nova solução basta incluir no ficheiro `unicornpuzzle.cnf` a negação desta fórmula como restrição:

   ```
   1 2 -3 -4 -5 0
   ```

   Interpretada como restrição, o seu significado exprime que pelo menos um dos valores lógicos atribuídos pelo modelo anterior terá agora de ser diferente.

   $$mythical \lor immortal \lor \neg mammal \lor \neg horned \lor \neg magical$$

   Tendo em conta isto, quantos modelos existem para este problema?

3. Comente as restrições que acrescentou para responder à alínea anterior e use agora o SAT solver para responder às perguntas do enigma. Lembre-se do seguinte resultado fundamental:

   $$\Gamma \models F \qquad \text{sse} \qquad \Gamma, \neg F \;\; \text{UNSAT}$$

   (a) *Is the unicorn magical?*

   (b) *Is the unicorn horned?*

   (c) *Is the unicorn mythical?*

4. Com a ajuda do solver, responda ainda às seguintes questões:
   (a) Será que é possível a um unicórnio ser simultanemente mítico e imortal?
   (b) Poderá existir um unicórnio mortal que não seja mamífero?

Exercício 4 (Configuração de produtos) Certos produtos, como é o caso dos automóveis, são altamente personalizáveis. Mas pode haver dependências intrincadas entre configurações. Os clientes podem não estar cientes de todas essas dependências, e poderão escolher opções de configuração inconsistentes.

Como são muitas configurações e muitas dependências, podemos usar um SAT solver para verificar se o cliente escolhe opções de configuração consistentes. Para isso, podemos seguir os seguintes passos:

- Codificar as dependências entre configurações como uma fórmula proposicional ψ.
- Codificar as opções selecionadas pelo cliente como uma fórmula proposicional ϕ.
- Usar o SAT solver para verificar se $\psi \wedge \phi$ não é contraditório.

Considere agora a seguinte dependência entre as configurações disponíveis para a personalização de um automóvel:

> *"O ar condicionado* Thermotronic comfort *requer uma bateria de alta capacidade, exceto quando combinado com motores a gasolina de 3,2 litros."*

Será que um cliente pode escolher o ar condicionado Thermotronic comfort, uma bateria de pequena capacidade, mas não escolher o motor de 3,2 litros? Responda a esta pergunta com a aajuda de um SAT solver.

SAT solvers API

Diversos SAT solvers possuem APIs de interface para diferentes linguagem de programação que permitem uma utilização incremental do solver. Por exemplo, a biblioteca **PySAT** (`https://pysathq.github.io`) para Python que fornece uma interface simples para vários SAT solvers.

Exercício 5 (Schedule a meeting) Recorde o problema codificado no ficheiro notebook Python `Meeting.ipynb`.

> – *Anne cannot meet on Friday.*
> – *Peter can only meet either on Monday, Wednesday or Thursday.*
> – *Mike cannot meet neither on Tuesday nor on Thursday.*
>
> – *When can the meeting take place?*

1. Comece por analisar o ficheiro e a resposta do solver. Depois altere o programa de forma a que sejam apresentadas todas as soluções possiveis para o problema.
2. A modelação que fizemos deste problema não está a exigir que a reunião se tem que realizar uma única vez na semana. Acrescente restrições que modelem esta exigência.

Mais alguns problemas

Exercício 6 (Sentando os convidados) Temos 3 cadeiras numa fila (*esquerda*, *meio*, *direita*), e precisamos de distribuir por elas 3 convidados (*Ana*, *Susana* e *Pedro*), com as seguintes restrições:

- *A Ana não quer ficar sentada à beira do Pedro.*
- *A Ana não quer ficar na cadeira da esquerda.*
- *A Susana não se quer sentar à esquerda do Pedro.*

- *Será possível sentar os convidados? Como?*

Para formular o problema em lógica proposicional, podemos considerar a seguinte indexação de pessoas e cadeiras:

$$Ana = 1,\ Susana = 2,\ Pedro = 3$$
$$esquerda = 1,\ meio = 2,\ direita = 3$$

Introduzimos depois variáveis proposicionais x_{ij} para $i \in \{1,2,3\}$ e $j \in \{1,2,3\}$, sendo que

$x_{ij} = 1$ *sse* a pessoa i ficar sentada na cadeira j

As restrições a escrever pertencem a várias categorias:

- Todas as pessoas devem estar sentadas numa cadeira.
- Não se poderá sentar mais do que uma pessoa em cada cadeira.
- Restrições correspondentes aos requisitos de cada pessoa.

1. Escreva todas as restrições necessárias para resolver o problema e converta-as em CNF.
2. Crie o ficheiro DIMACS CNF correspondente e invoque o SAT solver.

   Sugestão: Pode implementar um pequeno programa (por exemplo, em C ou em Phyton) para gerar o ficheiro DIMACS CNF para enviar ao SAT solver. Note que pode criar uma matriz ou um dicionário, `x`, de forma a fazer o mapeamento entre cada variável proposicional `x[i][j]` e o valor inteiro que lhe corresponde no formato DIMACS CNF.
3. Desenvolva um programa em Python para resolver este problema, recorrendo ao PySAT.

Exercício 7 (Equivalência de cadeias *if-then-else*) Considere os programas

(1)
```
if (!a && !b) h();
else if (!a) g();
     else f();
```

(2)
```
if (a) f();
      else if (b) g();
           else h();
```

É possível determinar se eles são equivalentes com a ajuda de um SAT solver. Para isso codificamos logicamente cada um deles, usando a seguinte regra de compilação:

$$\mathsf{compile}(\textbf{if } x \textbf{ then } y \textbf{ else } z) \;=\; (x \wedge y) \vee (\neg x \wedge z)$$

Basta depois decidir se as fórmulas $\mathsf{compile}(\mathbf{1})$ e $\mathsf{compile}(\mathbf{2})$ são **equivalentes**. Para isso teremos que resolver um problema de **validade** da fórmula

$$\mathsf{compile}(\mathbf{1}) \leftrightarrow \mathsf{compile}(\mathbf{2})$$

.

Para resolver um problema de validade com um SAT solver, há que negar a fórmula:

$$\begin{array}{l} \neg(\mathsf{compile}(1) \leftrightarrow \mathsf{compile}(2)) \\ \equiv \\ \neg((\mathsf{compile}(1) \wedge \mathsf{compile}(2)) \vee (\neg\mathsf{compile}(1) \wedge \neg\mathsf{compile}(2))) \\ \equiv \\ (\neg\mathsf{compile}(1) \vee \neg\mathsf{compile}(2)) \wedge (\mathsf{compile}(1) \vee \mathsf{compile}(2)) \end{array}$$

Se esta fórmula negada for UNSAT, então os programas serão equivalentes.

Considerando variáveis a, b, f, g, h, codifique os programas **(1)** e **(2)** acima e determine, convertendo a fórmula para CNF e usando o SAT solver, se eles são ou não equivalentes.

Exercício 8 (Sudoku)

Os puzzles Sudoku são problemas de colocação de números inteiros entre 1 e N^2 numa matriz quadrada de dimensão N^2, por forma a que cada coluna e cada linha contenha todos os números, sem repetições. Além disso, cada matriz contém N^2 sub-matrizes quadradas disjuntas, de dimensão N, que deverão também elas conter os números entre 1 e N^2.

Cada problema é dado por uma matriz parcialmente preenchida, cabendo ao jogador completá-la.

O problema pode ser codificado em lógica proposicional criando uma variável proposicional para cada triplo (l, c, n), onde l é uma linha, c é uma coluna, e n é um número. $x_{l,c,n} = 1$ se na linha l, coluna c, estiver o número n, caso contrário será 0.

Tendo em conta o exposto:

1. Modele o problema do Sudoku como um problema SAT, escrevendo as restrições correspondentes às regras do puzzle.

2. Desenvolva um pequeno programa em Python que recebe um tabuleiro Sudoku (por exemplo, a partir de um ficheiro de texto no formato que entender) e imprime o tabuleiro resolvido.