[PG50327] Diogo da Silva Rebelo, Mestrado em Engenharia Informática

Metodos Formais em Engenharia de Software (2022/2023)

SAT solving - Exercício de avaliação

0. Dados do Problema:

- Personalização entre:
  - 2 modelos de CPU;
  - 2 modelos de RAM;
  - 2 modelos de Motherboards;
  - 3 modelos de Placa Gráfica;
  - 3 modelos de Monitor;
- Um computador tem de possuir:
  - 1 CPU;
  - 1 RAM;
  - 1 Motherboard;
  - 1 Placa Gráfica;
  - Pode possuir ou não monitor(es);
- A personalização obedece a um conjunto de regras definido no enunciado.

1. Modelação do Problema:

1.1 Conjunto de Variáveis Proposicionais:

Para o nome de cada variável utilizou-se a abreviatura do componente seguida do seu tipo.

Componente	Tipo			Variável Proposicional
CPU	1	2		CPU[1/2]
RAM	1	2		RAM[1/2]
MB	1	2		MB[1/2]
PG	1	2	3	PG[1/2/3]
MON	1	2	3	MON[1/2/3]

1.2 Descrição do Problema com VP e conversão das fórmulas para CNF:

- ***Cada PC deve ter um CPU.***

  $(CPU1 \lor CPU2) \land (\neg CPU1 \lor \neg CPU2)$
- ***Cada PC deve ter uma RAM.***

  $(RAM1 \lor RAM2) \land (\neg RAM1 \lor \neg RAM2)$
- ***Cada PC deve ter uma MB.***

  $(MB1 \lor MB2) \land (\neg MB1 \lor \neg MB2)$
- ***Cada PC deve ter uma PG.***

  Para este caso, decidi, utilizar uma propriedade, já que é uma fórmula que lida com mais de duas variáveis proposicionais. Então:

  Seja t cada tipo de placa gráfica. Para este caso, podemos até utilizar a regra seguinte:

  - Pelo menos 1 PG: $\bigvee_{t=1}^{3} PGt$
  - No máximo 1 PG: $\bigwedge_{a=1}^{K-1} \bigwedge_{b=a+1}^{K} (\neg PGt_a \lor \neg PGt_b)$
  - O que equivale a:

  $(PG1 \land \neg PG2 \land \neg PG3) \lor (PG2 \land \neg PG1 \land \neg PG3) \lor (PG3 \land \neg PG1 \land \neg PG2)$

  $\equiv$

  $(\neg PG1 \lor \neg PG2) \land (\neg PG1 \lor \neg PG3) \land (PG1 \lor PG2 \lor PG3) \land (\neg PG2 \lor \neg PG3)$
- ***Cada PC deve ter 0 ou mais monitores.***

  $MON1 \lor MON2 \lor MON3 \lor (\neg MON1 \land \neg MON2 \land \neg MON3) \equiv \top$
- ***A motherboard MB1 quando combinada com a placa gráfica PG1, obriga à utilização da RAM1.***

$$(MB1 \wedge PG1) \rightarrow RAM1 \equiv \neg(MB1 \wedge PG1) \vee RAM1 \equiv \neg MB1 \vee \neg PG1 \vee RAM1$$

- ***A placa gráfica PG1 precisa do CPU1, excepto quando combinada com uma memória RAM2.***

$$(PG1 \wedge \neg RAM2) \rightarrow CPU1 \equiv \neg(PG1 \wedge \neg RAM2) \vee CPU1 \equiv \neg PG1 \vee RAM2 \vee CPU1$$

- ***O CPU2 só pode ser instalado na motherboard MB2.***

$$CPU2 \rightarrow MB2 \equiv \neg CPU2 \vee MB2$$

- ***O monitor MON1 para poder funcionar precisa da placa gráfica PG1 e da memória RAM2.***

$$MON1 \rightarrow (PG1 \wedge RAM2) \equiv \neg MON1 \vee (PG1 \wedge RAM2) \equiv (\neg MON1 \vee PG1) \wedge (\neg MON1 \vee RAM2)$$

- ***O monitor MON2 precisa da memória RAM2 para poder trabalhar com a placa gráfica PG3.***

$$(MON2 \wedge PG3) \rightarrow RAM2 \equiv \neg(MON2 \wedge PG3) \vee RAM2 \equiv \neg MON2 \vee \neg PG3 \vee RAM2$$

---

2. Codificação em SAT solver e prova de consistência:

Para codificar cada variável proposicional em formato suportado pelo solver, optou-se por identificar cada componente por um inteiro maior que 0, como se mostra abaixo.

```
!pip install python-sat[pblib,aiger]
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting python-sat[aiger,pblib]
  Downloading python_sat-0.1.7.dev19-cp37-cp37m-manylinux2010_x86_64.whl (1.8 MB)
     |████████████████████████████████| 1.8 MB 5.8 MB/s
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from python-sat[aiger,pblib]) (1.15.0)
Collecting py-aiger-cnf>=2.0.0
  Downloading py_aiger_cnf-5.0.4-py3-none-any.whl (5.2 kB)
Collecting pypblib>=0.0.3
  Downloading pypblib-0.0.4-cp37-cp37m-manylinux2014_x86_64.whl (3.4 MB)
     |████████████████████████████████| 3.4 MB 48.0 MB/s
Collecting bidict<0.22.0,>=0.21.0
  Downloading bidict-0.21.4-py3-none-any.whl (36 kB)
Collecting funcy<2.0,>=1.12
  Downloading funcy-1.17-py2.py3-none-any.whl (33 kB)
Collecting py-aiger<7.0.0,>=6.0.0
  Downloading py_aiger-6.1.25-py3-none-any.whl (18 kB)
Collecting parsimonious<0.9.0,>=0.8.1
  Downloading parsimonious-0.8.1.tar.gz (45 kB)
     |████████████████████████████████| 45 kB 4.1 MB/s
Collecting toposort<2.0,>=1.5
  Downloading toposort-1.7-py2.py3-none-any.whl (9.0 kB)
Requirement already satisfied: pyrsistent<0.19.0,>=0.18.0 in /usr/local/lib/python3.7/dist-packages (from py-aiger<7.0.
Collecting attrs<22,>=21
  Downloading attrs-21.4.0-py2.py3-none-any.whl (60 kB)
     |████████████████████████████████| 60 kB 9.7 MB/s
Requirement already satisfied: sortedcontainers<3.0.0,>=2.3.0 in /usr/local/lib/python3.7/dist-packages (from py-aiger<
Building wheels for collected packages: parsimonious
  Building wheel for parsimonious (setup.py) ... done
  Created wheel for parsimonious: filename=parsimonious-0.8.1-py3-none-any.whl size=42723 sha256=bddc13d633e9458fdcff8c
  Stored in directory: /root/.cache/pip/wheels/88/5d/ba/f27d8af07306b65ee44f9d3f9cadea1db749a421a6db8a99bf
Successfully built parsimonious
Installing collected packages: toposort, parsimonious, funcy, bidict, attrs, py-aiger, python-sat, pypblib, py-aiger-cn
  Attempting uninstall: attrs
    Found existing installation: attrs 22.1.0
    Uninstalling attrs-22.1.0:
      Successfully uninstalled attrs-22.1.0
Successfully installed attrs-21.4.0 bidict-0.21.4 funcy-1.17 parsimonious-0.8.1 py-aiger-6.1.25 py-aiger-cnf-5.0.4 pypbl
```

```
from pysat.solvers import Minisat22

s = Minisat22()

components = ['CPU1','CPU2','RAM1','RAM2','MB1','MB2','PG1','PG2','PG3','MON1','MON2','MON3']

# Codificação dos componentes e seus tipos
  # CPU1 = 1  | CPU2 = 2
  # RAM1 = 3  | RAM2 = 4
  # MB1  = 5  | MB2  = 6
  # PG1  = 7  | PG2  = 8  | PG3  = 9
  # MON1 = 10 | MON2 = 11 | MON3 = 12

x = {}
i = 1

for c in components:
  x[c] = i
  i += 1

# Unicidade
```

```
# Unicidade
s.add_clause([x['CPU1'], x['CPU2']])
s.add_clause([-x['CPU1'], -x['CPU2']])

s.add_clause([x['RAM1'], x['RAM2']])
s.add_clause([-x['RAM1'], -x['RAM2']])

s.add_clause([x['MB1'], x['MB2']])
s.add_clause([-x['MB1'], -x['MB2']])

# Exatamente 1 PG
# Pelo menos 1 PG
s.add_clause([x[t] for t in components[6:9]])

# No máximo 1 PG
t = 1
for a in components[6:8]:
  for b in components[6+t:9]:
    s.add_clause([-x[a], -x[b]])
  t = t+1

# para que o monitor 3 seja considerado no modelo e não dar erro de índice
s.add_clause([-x['MON3'],x['MON3']])

# Rstantes regras
s.add_clause([-x['MB1'], -x['PG1'], x['RAM1']])
s.add_clause([-x['PG1'], x['RAM2'], x['CPU1']])
s.add_clause([-x['CPU2'], x['MB2']])
s.add_clause([-x['MON1'], x['PG1']])
s.add_clause([-x['MON1'], x['RAM2']])
s.add_clause([-x['MON2'], -x['PG3'], x['RAM2']])

if s.solve():
    print("SAT")
    m = s.get_model()
    print(f"{m}\n")
    print("O computador pode ser formado pelos componentes: ")
    for d in components:
      if m[x[d]-1] > 0:
        print("[%s] " %d)

else:
    print("UNSAT")
    print("O computador não pode ser produzido.")

s.delete()
```

```
SAT
[1, -2, 3, -4, 5, -6, 7, -8, -9, -10, -11, -12]

O computador pode ser formado pelos componentes:
[CPU1]
[RAM1]
[MB1]
[PG1]
```

Por definição, um conjunto de fórmulas é consistente ou satisfazível se **existir uma atribuição que sirva de modelo a essa conjunto de fórmulas.** Então, como verificamos no solver, o problema é SAT e, por isso, é satisfazível, sendo consistente. O modelo encontrado pode ser obtido executando o código acima.

3. Resposta às questões com SAT solver:

(a) ***O monitor MON1 só poderá ser usado com uma motherboard MB1?***

Para responder a esta questão, temos sempre presente as seguintes propriedades:

$$\mathcal{T} \mid= F \quad iff \quad \mathcal{T}, \neg F \quad UNSAT$$

$$F \text{ é } satisfazível \; se \; e \; só \; se \; \neg F \text{ não é } válido$$

Trata-se de um problema de validez.

Coloque-se a fórmula inicial em CNF:

$$MON1 \rightarrow MB1 \equiv \neg MON1 \vee MB1$$

Negando a fórmula: $\neg(\neg MON1 \vee MB1) \equiv MON1 \wedge \neg MB1$

Então, verifiquemos se:

$T \wedge \neg(\neg MON1 \vee MB1) \equiv \mathbf{T \wedge (MON1 \wedge \neg MB1)}$ **é insatisfazível.**

```
from pysat.solvers import Minisat22

s = Minisat22()

components = ['CPU1','CPU2','RAM1','RAM2','MB1','MB2','PG1','PG2','PG3','MON1','MON2','MON3']

# Codificação dos componentes e seus tipos
  # CPU1 = 1  | CPU2 = 2
  # RAM1 = 3  | RAM2 = 4
  # MB1  = 5  | MB2  = 6
  # PG1  = 7  | PG2  = 8  | PG3  = 9
  # MON1 = 10 | MON2 = 11 | MON3 = 12

x = {}
i = 1

for c in components:
  x[c] = i
  i += 1

# Unicicidade
s.add_clause([x['CPU1'], x['CPU2']])
s.add_clause([-x['CPU1'], -x['CPU2']])

s.add_clause([x['RAM1'], x['RAM2']])
s.add_clause([-x['RAM1'], -x['RAM2']])

s.add_clause([x['MB1'], x['MB2']])
s.add_clause([-x['MB1'], -x['MB2']])

# Exatamente 1 PG
# Pelo menos 1 PG
s.add_clause([x[t] for t in components[6:9]])

# No máximo 1 PG
t = 1
for a in components[6:8]:
  for b in components[6+t:9]:
    s.add_clause([-x[a], -x[b]])
  t = t+1

# para que o monitor 3 seja considerado no modelo e não dar erro de índice
s.add_clause([-x['MON3'],x['MON3']])

# Rstantes regras
s.add_clause([-x['MB1'], -x['PG1'], x['RAM1']])
s.add_clause([-x['PG1'], x['RAM2'], x['CPU1']])
s.add_clause([-x['CPU2'], x['MB2']])
s.add_clause([-x['MON1'], x['PG1']])
s.add_clause([-x['MON1'], x['RAM2']])
s.add_clause([-x['MON2'], -x['PG3'], x['RAM2']])

# 1. MON1 só poderá ser usado com uma motherboard MB1?
# MON1∧¬MB1
s.add_clause([x['MON1']])
s.add_clause([-x['MB1']])

if s.solve():
    print("SAT")
    m = s.get_model()
    print(f"{m}\n")
    print("O computador pode ser formado pelos componentes: ")
    for d in components:
      if m[x[d]-1] > 0:
        print("[%s] " %d)

else:
    print("UNSAT")
    print("O computador não pode ser produzido.")

s.delete()
```

```
SAT
[1, -2, -3, 4, -5, 6, 7, -8, -9, 10, -11, -12]

O computador pode ser formado pelos componentes:
[CPU1]
[RAM2]
[MB2]
[PG1]
[MON1]
```

R: Executando a porção de código acima, verificamos que deu SAT e que na solução encontrada se utilizam os componentes CPU1, RAM2, MB2, PG1 e MON1. Então, o MON1 está a ser utilizado com a MB2. De facto, não estamos a dizer que o MON1 não pode ser usado com uma MB1, apenas estamos a dizer que essa combinação não é obrigatória. A negação da proposição é satisfazível, logo, a afirmação inicial não é válida.

(b) ***Um cliente pode personalizar o seu computador da seguinte forma: uma motherboard MB1, o CPU1, a placa gráfica PG2 e a memória RAM1?***

Trata-se de um problema de satisfação.

Coloque-se a fórmula incial em CNF:

$$MB1 \wedge CPU1 \wedge PG2 \wedge RAM1$$

```
from pysat.solvers import Minisat22

s = Minisat22()

components = ['CPU1','CPU2','RAM1','RAM2','MB1','MB2','PG1','PG2','PG3','MON1','MON2','MON3']

# Codificação dos componentes e seus tipos
  # CPU1 = 1  | CPU2 = 2
  # RAM1 = 3  | RAM2 = 4
  # MB1  = 5  | MB2  = 6
  # PG1  = 7  | PG2  = 8  | PG3  = 9
  # MON1 = 10 | MON2 = 11 | MON3 = 12

x = {}
i = 1

for c in components:
  x[c] = i
  i += 1

# Unicicidade
s.add_clause([x['CPU1'], x['CPU2']])
s.add_clause([-x['CPU1'], -x['CPU2']])

s.add_clause([x['RAM1'], x['RAM2']])
s.add_clause([-x['RAM1'], -x['RAM2']])

s.add_clause([x['MB1'], x['MB2']])
s.add_clause([-x['MB1'], -x['MB2']])

# Exatamente 1 PG
# Pelo menos 1 PG
s.add_clause([x[t] for t in components[6:9]])

# No máximo 1 PG
t = 1
for a in components[6:8]:
  for b in components[6+t:9]:
    s.add_clause([-x[a], -x[b]])
  t = t+1

# para que o monitor 3 seja considerado no modelo e não dar erro de índice
s.add_clause([-x['MON3'],x['MON3']])

# Rstantes regras
s.add_clause([-x['MB1'], -x['PG1'], x['RAM1']])
s.add_clause([-x['PG1'], x['RAM2'], x['CPU1']])
s.add_clause([-x['CPU2'], x['MB2']])
s.add_clause([-x['MON1'], x['PG1']])
s.add_clause([-x['MON1'], x['RAM2']])
s.add_clause([-x['MON2'], -x['PG3'], x['RAM2']])

# 2. Um cliente pode personalizar o seu computador da seguinte forma:
# uma motherboard MB1, o CPU1, a placa gráfica PG2 e a memória RAM1?
# MB1∧CPU1∧PG2∧RAM1
s.add_clause([x['MB1']])
s.add_clause([x['CPU1']])
s.add_clause([x['PG2']])
s.add_clause([x['RAM1']])


if s.solve():
    print("SAT")
    m = s.get_model()
    print(f"{m}\n")
    print("O computador pode ser formado pelos componentes: ")
```

```
    for d in components:
      if m[x[d]-1] > 0:
        print("[%s] " %d)

else:
    print("UNSAT")
    print("O computador não pode ser produzido.")

s.delete()
```

```
SAT
[1, -2, 3, -4, 5, -6, -7, 8, -9, -10, -11, -12]

O computador pode ser formado pelos componentes:
[CPU1]
[RAM1]
[MB1]
[PG2]
```

R: Executando a porção de código acima, verificamos que deu SAT e que na solução encontrada se utilizam os componentes CPU1, RAM1, MB1 e PG2. Há esta combinação de valores que torna a proposição é verdadeira, logo, é satisfazível.

(c) ***É possivel combinar a motherboard MB2, a placa gráfica PG3 e a RAM1 num mesmo computador?***

Trata-se de um problema de satisfação.

Coloque-se a fórmula inicial em CNF:

$MB2 \wedge PG3 \wedge RAM1$

```
from pysat.solvers import Minisat22

s = Minisat22()

components = ['CPU1','CPU2','RAM1','RAM2','MB1','MB2','PG1','PG2','PG3','MON1','MON2','MON3']

# Codificação dos componentes e seus tipos
  # CPU1 = 1  | CPU2 = 2
  # RAM1 = 3  | RAM2 = 4
  # MB1  = 5  | MB2  = 6
  # PG1  = 7  | PG2  = 8  | PG3  = 9
  # MON1 = 10 | MON2 = 11 | MON3 = 12

x = {}
i = 1

for c in components:
  x[c] = i
  i += 1

# Unicicidade
s.add_clause([x['CPU1'], x['CPU2']])
s.add_clause([-x['CPU1'], -x['CPU2']])

s.add_clause([x['RAM1'], x['RAM2']])
s.add_clause([-x['RAM1'], -x['RAM2']])

s.add_clause([x['MB1'], x['MB2']])
s.add_clause([-x['MB1'], -x['MB2']])

# Exatamente 1 PG
# Pelo menos 1 PG
s.add_clause([x[t] for t in components[6:9]])

# No máximo 1 PG
t = 1
for a in components[6:8]:
  for b in components[6+t:9]:
    s.add_clause([-x[a], -x[b]])
  t = t+1

# para que o monitor 3 seja considerado no modelo e não dar erro de índice
s.add_clause([-x['MON3'],x['MON3']])

# Rstantes regras
s.add_clause([-x['MB1'], -x['PG1'], x['RAM1']])
s.add_clause([-x['PG1'], x['RAM2'], x['CPU1']])
s.add_clause([-x['CPU2'], x['MB2']])
s.add_clause([-x['MON1'], x['PG1']])
s.add_clause([-x['MON1'], x['RAM2']])
s.add_clause([-x['MON2'], -x['PG3'], x['RAM2']])
```

```
# 3. É possivel combinar a motherboard MB2, a placa gráfica PG3 e a RAM1 num ´
# mesmo computador?
# MB2∧PG3∧RAM1
s.add_clause([x['MB2']])
s.add_clause([x['PG3']])
s.add_clause([x['RAM1']])

if s.solve():
    print("SAT")
    m = s.get_model()
    print(f"{m}\n")
    print("O computador pode ser formado pelos componentes: ")
    for d in components:
      if m[x[d]-1] > 0:
        print("[%s] " %d)

else:
    print("UNSAT")
    print("O computador não pode ser produzido.")

s.delete()
```

```
SAT
[1, -2, 3, -4, -5, 6, -7, -8, 9, -10, -11, -12]

O computador pode ser formado pelos componentes:
[CPU1]
[RAM1]
[MB2]
[PG3]
```

R: Executando a porção de código acima, verificamos que deu SAT e que na solução encontrada se utilizam os componentes CPU1, RAM1, MB2 e PG3, e, por isso, uma solução com a combinação de componentes pretendida. Há esta combinação de valores que torna a proposição é verdadeira, logo, é satisfazível.

(d) ***Para combinarmos a placa gráfica PG2 e a RAM1 temos que usar o CPU2?***

Para responder a esta questão, temos sempre presente as seguintes propriedades:

$\mathcal{T} \mid= F \quad iff \quad \mathcal{T}, \neg F \quad UNSAT$

$F \text{ é } satisfazível \; se \; e \; só \; se \; \neg F \; não \text{ é } válido$

Trata-se de um problema de validez.

Coloque-se a fórmula inicial em CNF:

$(PG2 \wedge RAM1) \rightarrow CPU2 \equiv \neg PG2 \vee \neg RAM1 \vee CPU2$

Negando a fórmula: $\neg(\neg PG2 \vee \neg RAM1 \vee CPU2) \equiv PG2 \wedge RAM1 \wedge \neg CPU2$

Então, verifiquemos se: $T \wedge \neg(\neg PG2 \vee \neg RAM1 \vee CPU2) \equiv \mathbf{T} \wedge \mathbf{(PG2 \wedge RAM1 \wedge \neg CPU2)}$ **é insatisfazível.**

```
from pysat.solvers import Minisat22

s = Minisat22()

components = ['CPU1','CPU2','RAM1','RAM2','MB1','MB2','PG1','PG2','PG3','MON1','MON2','MON3']

# Codificação dos componentes e seus tipos
  # CPU1 = 1  | CPU2 = 2
  # RAM1 = 3  | RAM2 = 4
  # MB1  = 5  | MB2  = 6
  # PG1  = 7  | PG2  = 8  | PG3  = 9
  # MON1 = 10 | MON2 = 11 | MON3 = 12

x = {}
i = 1

for c in components:
  x[c] = i
  i += 1

# Unicicidade
s.add_clause([x['CPU1'], x['CPU2']])
s.add_clause([-x['CPU1'], -x['CPU2']])

s.add_clause([x['RAM1'], x['RAM2']])
s.add_clause([-x['RAM1'], -x['RAM2']])

s.add_clause([x['MB1'], x['MB2']])
```

```
s.add_clause([-x['MB1'], -x['MB2']])

# Exatamente 1 PG
# Pelo menos 1 PG
s.add_clause([x[t] for t in components[6:9]])

# No máximo 1 PG
t = 1
for a in components[6:8]:
  for b in components[6+t:9]:
    s.add_clause([-x[a], -x[b]])
  t = t+1

# para que o monitor 3 seja considerado no modelo e não dar erro de índice
s.add_clause([-x['MON3'],x['MON3']])

# Rstantes regras
s.add_clause([-x['MB1'], -x['PG1'], x['RAM1']])
s.add_clause([-x['PG1'], x['RAM2'], x['CPU1']])
s.add_clause([-x['CPU2'], x['MB2']])
s.add_clause([-x['MON1'], x['PG1']])
s.add_clause([-x['MON1'], x['RAM2']])
s.add_clause([-x['MON2'], -x['PG3'], x['RAM2']])

# 4. Para combinarmos a placa gráfica PG2 e a RAM1 temos que usar o CPU2?
# PG2∧RAM1∧¬CPU2 ?
s.add_clause([x['PG2']])
s.add_clause([x['RAM1']])
s.add_clause([-x['CPU2']])

if s.solve():
    print("SAT")
    m = s.get_model()
    print(f"{m}\n")
    print("O computador pode ser formado pelos componentes: ")
    for d in components:
      if m[x[d]-1] > 0:
        print("[%s] " %d)

else:
    print("UNSAT")
    print("O computador não pode ser produzido.")

s.delete()
```

```
SAT
[1, -2, 3, -4, 5, -6, -7, 8, -9, -10, -11, -12]

O computador pode ser formado pelos componentes:
[CPU1]
[RAM1]
[MB1]
[PG2]
```

R: Executando a porção de código acima, verificamos que deu SAT e que na solução encontrada se utilizam os componentes CPU1, RAM1, MB1 e PG2, e, por isso, uma solução em que se utilizam PG2 e RAM1 sem o CPU2. Ou seja, para combinarmos a PG2 e a RAM1, pode não ser necessário o CPU2. A negação da proposição é satisfazível, logo, a afirmação inicial não é válida.
