HI4 実験 SMT ソルバ演習レポート

提出日: 2024.11.29

HI4 45 番 山口惺司

課題 smt-2: 解のない連立方程式

解がないような連立方程式を考え，その解を Z3 に求めさせようとした場合に，UNSAT(充足不能，解なし) となることを確認するプログラムを作成せよ.

解答

次のような連立方程式を考えた.

$$2x + 3y = 6$$
$$2x + 3y = 8$$

この連立方程式は解のない連立方程式である.
解がない理由は x と y の係数が一致するのに対し, 右辺の値が一致しないためである.
この連立方程式の解を Z3 に求めさせようとするとどうなるのか調べる.

プログラム

```
1  from z3 import *
2
3  x = Real('x')
4  y = Real('y')
5
6  s = Solver()
7  s.add(2*x + 3*y == 6)
8  s.add(2*x + 3*y == 8)
9
10 result = s.check() # sat or unsat
11 print(result)
12 if result == sat:
13     print(s.model()) # show a solution
```

実行例

実行例 1

```
$ python3 smt-2.py
unsat
```

課題 smt-5: 魔法陣

3x3 の「魔法陣」の一例を SMT ソルバで見つけさせるプログラムを作成せよ.

解答

魔方陣の条件は 13 20 行目で追加しており, 縦横斜めの合計値が Sum と等しくなるようにしている.
また, 得られた魔方陣は, 各行・各列・各対角線の総計の和がすべて 15 となっているため, 条件を満たしていると言える.

プログラム

```
1  from z3 import *
2  import numpy as np
3
4  n = IntVector('n', 9)
5  Sum = Int('Sum')
6
7  s = Solver()
8
9  for i in range(9):
10         s.add(n[i] >= 1)
11         s.add(n[i] <= 9)
12
13 s.add(Sum == n[0] + n[1] + n[2]);
14 s.add(Sum == n[3] + n[4] + n[5]);
15 s.add(Sum == n[6] + n[7] + n[8]);
16 s.add(Sum == n[0] + n[3] + n[6]);
17 s.add(Sum == n[1] + n[4] + n[7]);
18 s.add(Sum == n[2] + n[5] + n[8]);
19 s.add(Sum == n[0] + n[4] + n[8]);
20 s.add(Sum == n[2] + n[4] + n[6]);
21
22 s.add(Distinct(n))
23
24 result = s.check()
25 print(result)
26 if result == sat:
27         m = s.model()
28
29 data = np.array([])
```

```
for i in range(9):
        data = np.append(data, m[n[i]])

data = data.reshape(3, 3)
print(data)
```

実行例

実行例 1

```
$ python3 smt-5.py
sat
[[2 7 6]
 [9 5 1]
 [4 3 8]]
```

課題 smt-a1: 連立方程式の難問を解かせる

人間が解くには「難問」とされている連立方程式を探し, それを SMT ソルバで解かせてみよう.

解答

次のような連立方程式を考えた.

$$\begin{aligned} y &= 9x^2 + 11z + 3 \\ 5x - y + 2 &= 0 \end{aligned}$$

この連立方程式は x^2 が入っており, 人間が解くには少々難しい. 問題作成に当たって参考にしたサイトは以下のとおりである.

Corbettmaths "Advanced simultaneous equations":

https://corbettmaths.com/wp-content/uploads/2013/02/advanced-simultaneous-equations-p

pdf

この解を Z3 に求めさせる.

実行結果から, 解を得るまでの時間の平均は, 0.168s となった.

プログラム

```
from z3 import *

x = Real('x')
y = Real('y')

```

```
6  s = Solver()
7  s.add(y = 9 * x**2 + 11 * x + 3)
8  s.add(5 * x -y + 2 == 0)
9
10 result = s.check() # sat or unsat
11 print(result)
12 if result == sat:
13     print(s.model()) # show a solution
```

実行例

実行例 1

```
$ time python3 smt-a1.py
sat
[y = 1/3, x = -1/3]

real    0m0.169s
user    0m0.145s
sys     0m0.025s
```

実行例 2

```
$ time python3 smt-a1.py
sat
[y = 1/3, x = -1/3]

real    0m0.168s
user    0m0.156s
sys     0m0.013s
```

実行例 3

```
$ time python3 smt-a1.py
sat
[y = 1/3, x = -1/3]

real    0m0.170s
user    0m0.162s
sys     0m0.009s
```

実行例 4

```
$ time python3 smt-a1.py
sat
[y = 1/3, x = -1/3]
```

```
real    0m0.168s
user    0m0.160s
sys     0m0.008s
```

実行例 5

```
$ time python3 smt-a1.py
sat
[y = 1/3, x = -1/3]

real    0m0.169s
user    0m0.148s
sys     0m0.021s
```

課題 smt-a2: ジョブ・スケジューリング (job scheduling)

次のような条件の 6 つのジョブがあり, 全ジョブを実行するスケジュールを作成したい. 全ジョブの完了が最も短くなるようなスケジュールを求めるプログラムを作成せよ.

条件:

- Job1 から Job6 の 6 つのジョブが存在する.
- 各ジョブの実行に必要な時間は, Job1 は 3 時間, Job2 は 2 時間, Job3 は 3 時間, Job4 は 4 時間, Job5 は 6 時間, Job6 は 2 時間である.
- Job3 は Job2 の後, Job4 は Job1 の後, Job5 は Job2 の後, Job6 は Job1 から Job5 すべてを終了した後に行う必要がある.

解答

7, 8 行目で各 Job の実行に必要な時間を設定している.
12 14 行目で「Job3 は Job2 の後, Job4 は Job1 の後, Job5 は Job2 の後」という条件を追加している.
16 21 行目で「Job6 は Job1 から Job5 すべてを終了した後に行う」という条件を追加している.
31 43 行目は Job の完了を視覚的にわかりやすくするためのコードである.

プログラム

```
1 from z3 import *
2
3 start = IntVector('start', 6)
```

```
end = IntVector('end', 6)
time = [3, 2, 3, 4, 6, 2]

for i in range(6):
        end[i] = start[i] + time[i]

s = Optimize()

s.add(start[2] == end[1])
s.add(start[3] == end[0])
s.add(start[4] == end[1])

for i in range(6):
        s.add(start[i] >= 0)
        s.add(end[i] >= 0)
        if i == 5:
                continue
        s.add(start[5] >= end[i])

s.minimize(sum(end))

result = s.check()
if result == sat:
        m = s.model()

start_time = []

flag = False

for i in range(6):
        print("job" + str(i+1) + ": ", end = "")
        for j in range(m[start[5]].as_long() + time[5]):
                if m[start[i]].as_long() <= j and m[start[i]].as_long() +
    time[i] > j:
                        print("*", end="")
                        flag = True
                else :
                        print(".", end="")
                        flag = False

        print()
```

実行例

実行例 1

```
$ python3 eq.py
job1: ***.......
```

```
job2: **........
job3: ..***.....
job4: ...****...
job5: ..******..
job6: ........**
```

課題 smt-3a: 9x9 の数独ソルバ

一般的な数独である 9x9 の数独の解を求めるプログラムを作成せよ.

解答

使用した数独のデータは以下のサイトから入手した.
ナンプレ 20 - 無料数独問題集: https://numberplace.net/
まず, 9x9 の数独のデータを 2 次元のリストとして data 入れた.
n を 1 次元の配列を宣言し, 24 27 行目ですでに埋まっている数字を n に代入した.
数独の条件として

- 各行に 1〜9 の数字がすべて含まれている.
- 各列に 1〜9 の数字がすべて含まれている.
- 各 3 × 3 のブロックに 1〜9 の数字がすべて含まれている.

というものがある. 29 31 行目では行についての条件,
33 35 行目では列についての条件,
37 40 行目では各 3x3 のブロックについての条件を追加した.
解を得るまでにかかった実行時間を平均すると, 0.3054s となった.
また, 得られた解を見ると数独の条件を満たしていると言える.

プログラム

```
1  from z3 import *
2  import time
3
4  data = [
5  [0, 2, 3, 0, 0, 5, 0, 6, 0],
6  [7, 0, 5, 0, 2, 0, 9, 0, 0],
7  [0, 0, 0, 0, 7, 0, 0, 0, 4],
8  [5, 8, 0, 0, 0, 0, 3, 0, 9],
9  [0, 0 ,0 ,0 ,0, 0, 0, 0, 0],
10 [6, 0, 2, 0, 1, 0, 5, 0, 0],
11 [0, 0, 0, 2, 9, 0, 0, 0, 0],
12 [2, 3, 7, 0, 0, 8, 0, 0, 0],
13 [0, 4, 1, 0, 0, 0, 0, 0, 0]]
14
```

```
n = IntVector('n', 81)
Sum = Int('Sum')

s = Solver()

for i in range(81):
        s.add(n[i] >= 1)
        s.add(n[i] <= 9)

for i in range(9):
        for j in range(9):
                if data[i][j] != 0:
                        s.add(n[i * 9 + j] == data[i][j])

for i in range(0, 81, 9):
        s.add(Sum == sum(n[i:i+9]))
        s.add(Distinct(n[i:i+9]))

for i in range(9):
        s.add(Sum == sum(n[i:81:9]))
        s.add(Distinct(n[i:81:9]))

for i in range(0, 81, 27):
        for j in range(i, i + 9, 3):
                s.add(Sum == sum(n[j:j+3] + n[j+9:j+12] + n[j+18:j+21]))
                s.add(Distinct(n[j:j+3] + n[j+9:j+12] +  n[j+18:j+21]))

result = s.check()
print(result)
if result == sat:
        m = s.model()
        print("Puzzle: ")
        for i in range(9):
                for j in range(9):
                        if data[i][j] == 0:
                                print("-", end = ' ')
                                continue
                        print(data[i][j], end = ' ')
                print()

        print()
        print("Solution: ")
        for i in range(81):
                print(m[n[i]], end = ' ')
                if i % 9 == 8:
                        print()
```

実行例

実行例 1

```
$ python3 smt-a3.py
sat

Puzzle:
- 2 3 - - 5 - 6 -
7 - 5 - 2 - 9 - -
- - - - 7 - - - 4
5 8 - - - - 3 - 9
- - - - - - - - -
6 - 2 - 1 - 5 - -
- - - 2 9 - - - -
2 3 7 - - 8 - - -
- 4 1 - - - - - -

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

real    0m0.301s
user    0m0.284s
sys     0m0.017s
```

実行例 2

```
$ python3 smt-a3.py
sat

Puzzle:
- 2 3 - - 5 - 6 -
7 - 5 - 2 - 9 - -
- - - - 7 - - - 4
5 8 - - - - 3 - 9
- - - - - - - - -
6 - 2 - 1 - 5 - -
- - - 2 9 - - - -
2 3 7 - - 8 - - -
```

```
- 4 1 - - - - - - -

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

real    0m0.300s
user    0m0.271s
sys     0m0.029s
```

実行例 3

```
$ python3 smt-a3.py
sat

Puzzle:
- 2 3 - - 5 - 6 -
7 - 5 - 2 - 9 - -
- - - - 7 - - - 4
5 8 - - - - 3 - 9
- - - - - - - - -
6 - 2 - 1 - 5 - -
- - - 2 9 - - - -
2 3 7 - - 8 - - -
- 4 1 - - - - - -

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

real    0m0.328s
user    0m0.300s
sys     0m0.028s
```

実行例 4

```
$ python3 smt-a3.py
sat

Puzzle:
- 2 3 - - 5 - 6 -
7 - 5 - 2 - 9 - -
- - - - 7 - - - 4
5 8 - - - - 3 - 9
- - - - - - - - -
6 - 2 - 1 - 5 - -
- - - 2 9 - - - -
2 3 7 - - 8 - - -
- 4 1 - - - - - -

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

real    0m0.298s
user    0m0.286s
sys     0m0.012s
```

実行例 5

```
$ python3 smt-a3.py
sat

Puzzle:
- 2 3 - - 5 - 6 -
7 - 5 - 2 - 9 - -
- - - - 7 - - - 4
5 8 - - - - 3 - 9
- - - - - - - - -
6 - 2 - 1 - 5 - -
- - - 2 9 - - - -
2 3 7 - - 8 - - -
- 4 1 - - - - - -
```

```
Solution:
4 2 3 9 8 5 1 6 7
7 6 5 4 2 1 9 8 3
1 9 8 6 7 3 2 5 4
5 8 4 7 6 2 3 1 9
3 1 9 8 5 4 7 2 6
6 7 2 3 1 9 5 4 8
8 5 6 2 9 7 4 3 1
2 3 7 1 4 8 6 9 5
9 4 1 5 3 6 8 7 2

real    0m0.300s
user    0m0.283s
sys     0m0.016s
```

感想

9x9 の数独は人間が解くには, かなり時間がかかるが, プログラムで解かせると 0.3 秒ほどで解いてしまうことに驚いた.
SMT ソルバは初めて触ったので, どういうコードを書けばいいのか, どういう条件を追加すればいいのかわからず, はじめは困惑したが何とかプログラムを作成することができてよかった.