# 形式化方法导引
## 第 4 章 逻辑问题求解——一种通用求解方法:SAT/SMT 求解
## 4.1 应用

黄文超

https://faculty.ustc.edu.cn/huangwenchao
⟶ 教学课程 ⟶ 形式化方法导引

# 课程回顾

- 第 1 章: 自动机、可计算性、复杂度理论
  - 问题是什么？可以解么? 有多难?
- 第 2 章: 怎样用逻辑来定义一个验证器问题?
  - Propositional logic, first-order logic, higher-order logic
- 第 3 章: 怎样进一步定义一种演算规则rules 来降低求解难度?
  - The proof calculus of natural deduction
- 本章: 如何针对上述的 rules求解?

# 本章内容

1. 应用：如何用工具解决经典逻辑相关的问题？
   - SAT,SMT 问题
   - 问题求解工具 Z3
   - 案例实现
     - Satisfiability
     - Validity
     - Numbers and inequalities
     - Eight Queens problem
     - Binary Arithmetic
     - Rectangle fitting
     - Solving Sudoku
   - 其它应用：Symbolic execution
2. 理论：这些工具的核心算法?

**定义**: Propositional Logic in *BNF*

$$\phi ::= p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi)$$

where $p$ stands for any atomic proposition and each occurrence of $\phi$ to the right of ::= stands for any already constructed formula.

**定义**: Verification in Logics

Most logics used in the design, specification and verification of computer systems fundamentally deal with a *satisfaction relation*:

$$\mathcal{M} \vDash \phi$$

问题: $\mathcal{M} \vDash \phi$ 在命题逻辑中**更简单**的表达是什么？

问题：$\mathcal{M} \vDash \phi$ 在命题逻辑中**更简单**的表达是什么？

## 引理: (去除 $\mathcal{M}$)

Given formulas $\phi_1, \phi_2, \ldots, \phi_n$ and $\psi$ of propositional logic,
$\phi_1, \phi_2, \ldots, \phi_n \vDash \psi$ holds iff $\vDash \phi_1 \to (\phi_2 \to (\phi_3 \to \cdots \to (\phi_n \to \psi)))$
holds.

答：**去除 $\mathcal{M}$ 后，求解** validity (见如下定义)

## 定义: Validity

We call $\phi$ *valid*, if $\vDash \phi$ holds.

- We also call $\phi$ as a *tautology* (**重言式**), if $\phi$ is valid.

**下一个问题**：**怎么求解** validity？
答：**换一个问题求解**: Satisfiability (见下页)

## 定义: Satisfiability

Given a formula $\phi$ in propositional logic, we say that $\phi$ is *satisfiable* if it has a valuation in which is evaluates to **T**.

## 例子:

$p \vee q \rightarrow p$ is *satisfiable*, since it computes **T** if we assign **T** to $p$.
Note that $p \vee q \rightarrow p$ is *not valid*.

## 定理:

Let $\phi$ be a formula of propositional logic. Then $\phi$ is *satisfiable* iff $\neg\phi$ is *not valid*.
In other words, $\phi$ is valid iff $\neg\phi$ is not satisfiable.

**总结**: **求解** $\mathcal{M} \vDash \phi$ ——**求解** Validity ——**求解** Satisfiability.

## 定义: SAT 问题

SAT is the *decision* problem: given a propositional formula, is it *satisfiable*?

SAT 问题可用于模型的验证!

## 回顾: 问题可以解么? – 问题 4

Given a set $A \subseteq S$, and $x \in S$, *whether there is a machine* that can compute whether $x \in A$.

- Define a new machine, named *Turing machine,* 图灵机.
- If yes, i.e., there is a Turing machine $M$ for $A$, language $A$ is *decidable*.
- If no, but there is a Turing machine $M$ that can only accept $s$, if $s \in A$, language $A$ is *still Turing-recognizable*.

SAT 是可计算的 (见下页)

## 定义: SAT 问题

SAT is the *decision* problem: given a propositional formula, is it *satisfiable*?

SAT 是可计算的:

- Essentially, this consists of computing the values of the formula for all $2^n$ ways to choose **T** or **F** for the $n$ variables.

问: SAT 属于哪一类问题? (**复杂度**)

答: SAT 属于经典的 NP-Complete 问题 (1970 **年开始研究**)(Bad news)。

## 回顾: 定义: NP-complete and NP-hard

A language $B$ is *NP-complete* if it satisfies two conditions:

1. $B$ is in NP, and

2. every $A$ in NP is polynomial time *reducible* to $B$.

Here, $B$ is *NP-hard* if it satisfies condition 2.

Classical *NP-complete* problem: *SAT* (**形式化方法的重要问题之一**).

Good news: Current SAT solvers are successful for several big formulas.

- 例子: solving the n-queens problem for n=100 yields a 50Mb formula over 10000 variables, but is solved in 10 seconds by the SAT solver *Z3*.

---

### 定义: SMT problem

Extension of SAT, to deal with *numbers* and *inequalities*.

---

SAT, SMT 求解工具:

- Z3, YICES, CVC4
- For non-commercial use they are free to download and to use

Z3

- Z3 is a theorem prover from Microsoft Research.

Z3 interfaces

- Default input format is SMTLIB2
- Other native foreign function interfaces:
  - C++ API
  - .NET API
  - Java API
  - Python API
  - C
  - OCaml
  - Julia

**参考阅读**

- SAT/SMT by Example

### 问题: Is $\phi$ satisfiable?

$\phi = (p \to q) \land (r \leftrightarrow \neg q) \land (\neg p \lor r)$

```python
from z3 import *
p = Bool('p')
q = Bool('q')
r = Bool('r')
solve(Implies(p, q), r == Not(q), Or(Not(p), r))
```

运行结果:
  $python3 z3-1-sat.py
    [q = True, p = False, r = False]
解析:
存在一个解, 满足 $\vDash \phi$。解为 $q = \textbf{T} \land p = \textbf{F} \land r = \textbf{F}$

```python
from z3 import *
p, q = Bools('p q')
demorgan = \
    And(p, q) == \
        Not(Or(Not(p), Not(q)))
def prove(f):
    s = Solver()
    s.add(Not(f))
    if s.check() == unsat:
        print("proved")
    else:
        print("failed to prove")
prove(demorgan)
```

问题: Is $\phi$ valid ? (i.e., $\models \phi$?)

$\phi = (p \wedge q) \leftrightarrow \neg(\neg p \vee \neg q)$

运行结果:
  $python3 z3-2-valid.py
    proved
解析:
  $\phi$ is valid,
    iff $\neg\phi$ is not satisfiable.
  So, $\phi$ is valid.

> 问题: Solve the following system of constraints
>
> $x > 2 \wedge y < 10 \wedge x + 2y = 7$, where $x, y$ are integers.

```python
from z3 import *
x = Int('x')
y = Int('y')
solve(x > 2, y < 10, x + 2*y == 7)
```

**运行结果**:
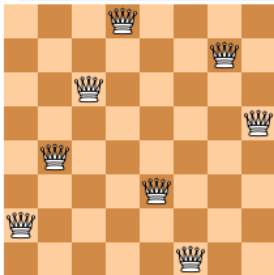    $python3 z3-3-inequalities.py
      [y = 0, x = 7]
**解析**:
    该问题为 SMT 问题

## 问题: Eight-Queens

The eight queens puzzle is the problem of placing eight chess queens on an $8 \times 8$ chessboard so that *no two queens attack each other*. Thus, a solution requires that *no two queens* share the *same row, column, or diagonal*.



As usual in SAT/SMT, don't think about how to solve it, but *only specify the problem*.

- Pure SAT: only boolean variables, no numbers, no inequalities.
- For every position $(i, j)$ on the board: boolean variable $p_{ij}$ expresses whether there is a queen or not.

| $p_{11}$ | $p_{12}$ | $p_{13}$ | $p_{14}$ | $p_{15}$ | $p_{16}$ | $p_{17}$ | $p_{18}$ |
|---|---|---|---|---|---|---|---|
| $p_{21}$ | $p_{22}$ | $p_{23}$ | $p_{24}$ | $p_{25}$ | $p_{26}$ | $p_{27}$ | $p_{28}$ |
| $p_{31}$ | $p_{32}$ | $p_{33}$ | $p_{34}$ | $p_{35}$ | $p_{36}$ | $p_{37}$ | $p_{38}$ |
| $p_{41}$ | $p_{42}$ | $p_{43}$ | $p_{44}$ | $p_{45}$ | $p_{46}$ | $p_{47}$ | $p_{48}$ |
| $p_{51}$ | $p_{52}$ | $p_{53}$ | $p_{54}$ | $p_{55}$ | $p_{56}$ | $p_{57}$ | $p_{58}$ |
| $p_{61}$ | $p_{62}$ | $p_{63}$ | $p_{64}$ | $p_{65}$ | $p_{66}$ | $p_{67}$ | $p_{68}$ |
| $p_{71}$ | $p_{72}$ | $p_{73}$ | $p_{74}$ | $p_{75}$ | $p_{76}$ | $p_{77}$ | $p_{78}$ |
| $p_{81}$ | $p_{82}$ | $p_{83}$ | $p_{84}$ | $p_{85}$ | $p_{86}$ | $p_{87}$ | $p_{88}$ |

(1) At least one queen on row $i$:

- $p_{i1} \vee p_{i2} \vee p_{i3} \vee p_{i4} \vee p_{i5} \vee p_{i6} \vee p_{i7} \vee p_{i8}$

$$\bigvee_{j=1}^{8} p_{ij}$$

(2) At most one queen on row $i$:

- For every $j < k$ not both $p_{ij}$ and $p_{ik}$ are true

$$\bigwedge_{0 < j < k \leq 8} (\neg p_{ij} \vee \neg p_{ik})$$

Requirements until now (on *every* row):

$$\bigwedge_{i=1}^{8} \bigvee_{j=1}^{8} p_{ij} \quad \wedge \quad \bigwedge_{i=1}^{8} \bigwedge_{0 < j < k \leq 8} (\neg p_{ij} \vee \neg p_{ik})$$

| $p_{11}$ | $p_{12}$ | $p_{13}$ | $p_{14}$ | $p_{15}$ | $p_{16}$ | $p_{17}$ | $p_{18}$ |
|---|---|---|---|---|---|---|---|
| $p_{21}$ | $p_{22}$ | $p_{23}$ | $p_{24}$ | $p_{25}$ | $p_{26}$ | $p_{27}$ | $p_{28}$ |
| $p_{31}$ | $p_{32}$ | $p_{33}$ | $p_{34}$ | $p_{35}$ | $p_{36}$ | $p_{37}$ | $p_{38}$ |
| $p_{41}$ | $p_{42}$ | $p_{43}$ | $p_{44}$ | $p_{45}$ | $p_{46}$ | $p_{47}$ | $p_{48}$ |
| $p_{51}$ | $p_{52}$ | $p_{53}$ | $p_{54}$ | $p_{55}$ | $p_{56}$ | $p_{57}$ | $p_{58}$ |
| $p_{61}$ | $p_{62}$ | $p_{63}$ | $p_{64}$ | $p_{65}$ | $p_{66}$ | $p_{67}$ | $p_{68}$ |
| $p_{71}$ | $p_{72}$ | $p_{73}$ | $p_{74}$ | $p_{75}$ | $p_{76}$ | $p_{77}$ | $p_{78}$ |
| $p_{81}$ | $p_{82}$ | $p_{83}$ | $p_{84}$ | $p_{85}$ | $p_{86}$ | $p_{87}$ | $p_{88}$ |

DIY

Similarly, on every *column*:

$$\bigwedge_{j=1}^{8} \bigvee_{i=1}^{8} p_{ij} \quad \wedge \quad \bigwedge_{j=1}^{8} \bigwedge_{0<i<k\leq 8} (\neg p_{ij} \vee \neg p_{kj})$$

| $p_{11}$ | $p_{12}$ | $p_{13}$ | $p_{14}$ | $p_{15}$ | $p_{16}$ | $p_{17}$ | $p_{18}$ |
|---|---|---|---|---|---|---|---|
| $p_{21}$ | $p_{22}$ | $p_{23}$ | $p_{24}$ | $p_{25}$ | $p_{26}$ | $p_{27}$ | $p_{28}$ |
| $p_{31}$ | $p_{32}$ | $p_{33}$ | $p_{34}$ | $p_{35}$ | $p_{36}$ | $p_{37}$ | $p_{38}$ |
| $p_{41}$ | $p_{42}$ | $p_{43}$ | $p_{44}$ | $p_{45}$ | $p_{46}$ | $p_{47}$ | $p_{48}$ |
| $p_{51}$ | $p_{52}$ | $p_{53}$ | $p_{54}$ | $p_{55}$ | $p_{56}$ | $p_{57}$ | $p_{58}$ |
| $p_{61}$ | $p_{62}$ | $p_{63}$ | $p_{64}$ | $p_{65}$ | $p_{66}$ | $p_{67}$ | $p_{68}$ |
| $p_{71}$ | $p_{72}$ | $p_{73}$ | $p_{74}$ | $p_{75}$ | $p_{76}$ | $p_{77}$ | $p_{78}$ |
| $p_{81}$ | $p_{82}$ | $p_{83}$ | $p_{84}$ | $p_{85}$ | $p_{86}$ | $p_{87}$ | $p_{88}$ |

$p_{ij}$ and $p_{i'j'}$ on such a diagnoal

$$i - j = i' - j'$$

| $p_{11}$ | $p_{12}$ | $p_{13}$ | $p_{14}$ | $p_{15}$ | $p_{16}$ | $p_{17}$ | $p_{18}$ |
|---|---|---|---|---|---|---|---|
| $p_{21}$ | $p_{22}$ | $p_{23}$ | $p_{24}$ | $p_{25}$ | $p_{26}$ | $p_{27}$ | $p_{28}$ |
| $p_{31}$ | $p_{32}$ | $p_{33}$ | $p_{34}$ | $p_{35}$ | $p_{36}$ | $p_{37}$ | $p_{38}$ |
| $p_{41}$ | $p_{42}$ | $p_{43}$ | $p_{44}$ | $p_{45}$ | $p_{46}$ | $p_{47}$ | $p_{48}$ |
| $p_{51}$ | $p_{52}$ | $p_{53}$ | $p_{54}$ | $p_{55}$ | $p_{56}$ | $p_{57}$ | $p_{58}$ |
| $p_{61}$ | $p_{62}$ | $p_{63}$ | $p_{64}$ | $p_{65}$ | $p_{66}$ | $p_{67}$ | $p_{68}$ |
| $p_{71}$ | $p_{72}$ | $p_{73}$ | $p_{74}$ | $p_{75}$ | $p_{76}$ | $p_{77}$ | $p_{78}$ |
| $p_{81}$ | $p_{82}$ | $p_{83}$ | $p_{84}$ | $p_{85}$ | $p_{86}$ | $p_{87}$ | $p_{88}$ |

$p_{ij}$ and $p_{i'j'}$ on such a diagonal

$$i + j = i' + j'$$

So far all $i, j, i', j'$ with $(i, j) \neq (i', j')$ satisfying $i + j = i' + j'$ or $i - j = i' - j'$:

$$\neg p_{ij} \vee \neg p_{i'j'}$$

stating that on $(i, j)$ and $(i', j')$ being two distinct positions on a diagonal, no two queens are allowed.

We may restrict to $i < i'$, yielding

$$\bigwedge_{0 < i < i' \leq 8} ( \bigwedge_{j, j' : i+j = i'+j' \vee i-j = i'-j'} \neg p_{ij} \vee \neg p_{i'j'} )$$

$$\bigwedge_{i=1}^{8} \bigvee_{j=1}^{8} p_{ij} \wedge \bigwedge_{i=1}^{8} \bigwedge_{0<j<k\leq 8} (\neg p_{ij} \vee \neg p_{ik}) \wedge \bigwedge_{j=1}^{8} \bigvee_{i=1}^{8} p_{ij} \wedge \bigwedge_{j=1}^{8} \bigwedge_{0<i<k\leq 8} (\neg p_{ij} \vee \neg p_{kj})$$

$$\bigwedge_{0<i<i'\leq 8} \left( \bigwedge_{j,j':i+j=i'+j' \vee i-j=i'-j'} \neg p_{ij} \vee \neg p_{i'j'} \right)$$

```python
from z3 import *
Q = [ Int('Q_%i' % (i + 1)) for i in range(8) ]
val_c = [ And(1 <= Q[i], Q[i] <= 8) for i in range(8) ]
col_c = [ Distinct(Q) ]
diag_c = [ If(i == j, True,
              And(i+Q[i]!=j+Q[j], i+Q[j]!=j+Q[i]))
           for i in range(8) for j in range(i) ]
solve(val_c + col_c + diag_c)
```

|   |   |   | o |   |   |   |   |
|---|---|---|---|---|---|---|---|
|   | o |   |   |   |   |   |   |
|   |   |   |   |   |   |   | o |
|   |   |   |   |   | o |   |   |
| o |   |   |   |   |   |   |   |
|   |   | o |   |   |   |   |   |
|   |   |   |   | o |   |   |   |
|   |   |   |   |   |   | o |   |

运行结果:
  $python3 z3-4-queens.py
    [$Q\_5 = 1$, $Q\_8 = 7$, $Q\_3 = 8$, $Q\_2 = 2$,
    $Q\_6 = 3$, $Q\_4 = 6$, $Q\_7 = 5$, $Q\_1 = 4$]
解析:
  From Eight-Queens to N-Queens?
    26s for 40 queens
思考: 不使用 SMT, 而是换之前解释的方法（pure SAT）, 效率会不会更高?
    看下面讲解

Arithmetic: addition, subtraction, multiplication of integers

## 问题
Compute 13+7?

**解析**: Covered by SMT, why do it in pure SAT?

- Interesting how to express a non-SAT looking problem in SAT
- Introduces flavor of *bounded model checking*
  - BMC, **有界模型检测**, 见第 5 章
  - one of the most important applications of SAT/SMT
- Often (e.g., in hardware verification), SAT encoding of arithmetic *outperforms* SMT

# 1. 应用
## 1.3 案例实现 | Arithmetic in pure SAT

In SAT we only have *Boolean variables*.

### 问题: Binary representation

How to Express a number by a sequence of Boolean values?

**解**: Binary representation

$$a_1 a_2 \cdots a_n$$

of number $a$:

$$a_i \in \{0, 1\} \text{ and}$$

$$a = a_n + 2a_{n-1} + 4a_{n-2} + \cdots = \sum_{i=1}^{n} a_i * 2^{n-i}$$

**例**:

- 01101 represents 8+4+1=13
- 00111 represents 4+2+1=7

## 问题: Add

How to compute $d = a + b$?

- Basic rules: Take care of *carry* $c$

- $0 + 0 + 0 = 0$, carry $= 0$
- $0 + 0 + 1 = 1$, carry $= 0$
- $0 + 1 + 1 = 0$, carry $= 1$
- $1 + 1 + 1 = 1$, carry $= 1$

Start by rightmost carry $= 0$, compute from right to left

| | | | | | | |
|---|---|---|---|---|---|---|
| carries $c$: | 0 | 1 | 1 | 1 | 1 | 0 |
| number $a$=13: | | 0 | 1 | 1 | 0 | 1 |
| number $b$=7: | | 0 | 0 | 1 | 1 | 1 |
| result $d$ | | 1 | 0 | 1 | 0 | 0 |

indeed yielding 10100 representing 20

问题: $d = a + b$?
Result $d_i$ in a formula for $i = 1, \ldots, n$ :

$$d_i \leftrightarrow (a_i \leftrightarrow (b_i \leftrightarrow c_i)) \tag{1}$$

correct, since $(a_i \leftrightarrow (b_i \leftrightarrow c_i))$ yields true if and only if 1 or 3 among $\{a_i, b_i, c_i\}$ yield true

Carry $c_{i-1}$ in a formula for $i = 1, \ldots, n$:

$$c_{i-1} \leftrightarrow ((a_i \wedge b_i) \vee (a_i \wedge c_i) \vee (b_i \wedge c_i)) \tag{2}$$

correct, since $(a_i \wedge b_i) \vee (a_i \wedge c_i) \vee (b_i \wedge c_i)$ yields true if and only at least 2 among $\{a_i, b_i, c_i\}$ yield true.

To express that we start by rightmost carry $= 0$, we state

$$\neg c_n \tag{3}$$

To express that the result should fit in $n$ bits, at the end we should not have a carry left, and we state

$$\neg c_0 \tag{4}$$

Let $\phi$ be the conjunction of all these requirements; this expresses the *correctness* of the corresponding binary addition

To compute $a + b$ for $a = 13, b = 7$, we apply a SAT solver to

$$\phi \wedge \underbrace{\neg a_1 \wedge a_2 \wedge a_3 \wedge \neg a_4 \wedge a_5}_{a=13=01101} \wedge \underbrace{\neg b_1 \wedge \neg b_2 \wedge b_3 \wedge b_4 \wedge b_5}_{b=7=00111}$$

The resulting satisfying assignment will contain $d_1, \neg d_2, d_3, \neg d_4, \neg d_5$ representing the result $d = 20$

Concluding,

In this way computing $d = a + b$ can be done by SAT solving for any binary numbers $a, b$.

By adding given values for $a_i, d_i$ to the formula $\phi$ expressing $d = a + b$, and reading $b_i$ from resulting satisfying assignment, we can compute $b = d - a$ by exploiting the same formula.

## 问题: Rectangle fitting

Given *a big rectangle* and *a number of small rectangles*, can you fit the small rectangles in the big one such that *no two overlap*.
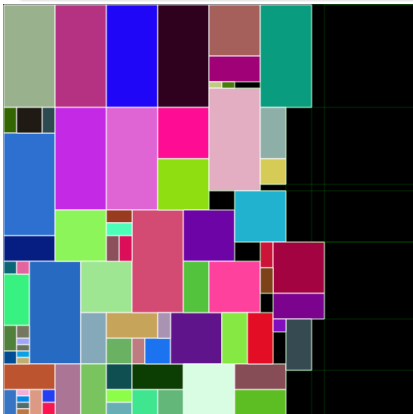


How to specify this problem?

- Number rectangles from $1$ to $n$
- for $i = 1 \ldots n$ introduce variables:
    - $w_i$ is the width of rectangle $i$
    - $h_i$ is the height of rectangle $i$
    - $x_i$ is the $x$-coordinate of the left lower corner of rectangle $i$
    - $y_i$ is the $y$-coordinate of the left lower corner of rectangle $i$

How to specify this problem?

- Configuration of small rectangles:
  - 例子: First rectangle has width 4 and height 6:
  - $(w_1 = 4 \land h_1 = 6) \lor (w_1 = 6 \land h_1 = 4)$
- Configuration of the big rectangle
  - $(0,0) =$ lower left corner of big rectangle.
  - $W =$ width of big rectangle.
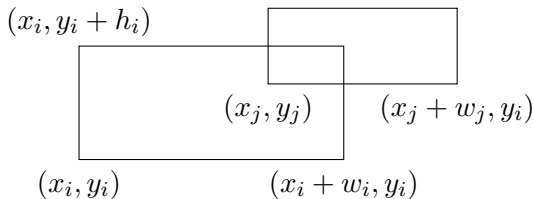  - $H =$ height of big rectangle.
- Requirements:

$$x_i \geq 0 \land x_i + w_i \leq W$$

$$y_i \geq 0 \land y_i + h_i \leq H$$

for all $i = 1, \ldots, n$

Rectangles $i$ and $j$ overlap if

- $x_j < x_i + w_i$
- and $x_i < x_j + w_j$
- and $y_j < y_i + h_i$
- and $y_i < y_j + h_j$

So for all $i, j = 1, \ldots, n, i < j$, we should add the negation of this overlappingness:

$$\neg(x_j < x_i + w_i \wedge x_i < x_j + w_j \wedge y_j < y_i + h_i \wedge y_i < y_j + h_j)$$

or, equivalently

$$x_j \geq x_i + w_i \vee x_i \geq x_j + w_j \vee y_j \geq y_i + h_i \vee y_i \geq y_j + h_j$$

The following formula is *satisfiable* iff the fitting problem *has a solution*

$$\bigwedge_{i=1}^{n} ((w_i = W_i \wedge h_i = H_i) \vee (w_i = H_i \wedge h_i = W_i))$$

$$\wedge \bigwedge_{i=1}^{n} (x_i \geq 0 \wedge x_i + w_i \leq W \wedge y_i \geq 0 \wedge y_i + h_i \leq H)$$

$$\wedge \bigwedge_{1 \leq i < j \leq n} (x_j \geq x_i + w_i \vee x_i \geq x_j + w_j \vee y_j \geq y_i + h_i \vee y_i \geq y_j + h_j)$$

If the formula is satisfiable, then the SMT solver yields a satisfying assignment, that is, the corresponding values of $x_i, y_i, w_i, h_i$

Applying a standard SMT solver like Z3, Yices, or CVC4: feasible for rectangle fitting problems up to *20 or 25 rectangles*

## 问题: Sudoku（数独游戏）

Fill the blank cells in such a way that

- every row, and
- every column, and
- every fat 3 block

contains the numbers 1 to 9, all occurring exactly once

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

**最强大脑**？: The puzzle may by very hard, and *backtracking* and/or *advanced solving techniques* are required.

For SAT/SMT it is peanuts: *just* specify the problem.

How?

Several approaches, all working well

- Pure SAT: for every cell and every number 1 to 9, introduce boolean variable describing whether that number is on that position, so $9^3 = 729$ boolean variables.
- SMT: for every cell define an integer variable for the corresponding number.

We elaborate the latter.

问题: How to specify that every row (and column, and $3 \times 3$ block) contains the numbers 1 to 9, all occurring once?

# 1. 应用
## 1.3 案例实现 | Suduku

Define $9 \times 9$ matrix of integer variables.
- Each cell contains a value in 1, ..., 9

```
X = [ [ Int("x_%s_%s" % (i+1, j+1)) for j in range(9) ]
        for i in range(9) ]
cells_c = [ And(1 <= X[i][j], X[i][j] <= 9)
                for i in range(9) for j in range(9) ]
```

Each row/column/fat 3 block contains a number at most once.

```
rows_c=[ Distinct(X[i]) for i in range(9) ]
cols_c=[ Distinct([ X[i][j] for i in range(9) ])
            for j in range(9) ]
sq_c  =[ Distinct([ X[3*i0 + i][3*j0 + j]
                    for i in range(3) for j in range(3) ])
            for i0 in range(3) for j0 in range(3) ]
```

```
sudoku_c = cells_c + rows_c + cols_c + sq_c
instance = ((0,0,0,0,9,4,0,3,0),
            (0,0,0,5,1,0,0,0,7),
            (0,8,9,0,0,0,0,4,0),
            (0,0,0,0,0,0,2,0,8),
            (0,6,0,2,0,1,0,5,0),
            (1,0,2,0,0,0,0,0,0),
            (0,7,0,0,0,0,5,2,0),
            (9,0,0,0,6,5,0,0,0),
            (0,4,0,9,7,0,0,0,0))
instance_c = [ If(instance[i][j] == 0,
                  True,
                  X[i][j] == instance[i][j])
               for i in range(9) for j in range(9) ]
```

```python
s = Solver()
s.add(sudoku_c + instance_c)
if s.check() == sat:
    m=s.model()
    r=[ [m.evaluate(X[i][j])
        for j in range(9)]
        for i in range(9) ]
    print_matrix(r)
else:
    print("failed to solve")
```

运行结果:
```
$python3 z3-5-sudoku.py
    [[7, 1, 5, 8, 9, 4, 6, 3, 2],
    [2, 3, 4, 5, 1, 6, 8, 9, 7],
    [6, 8, 9, 7, 2, 3, 1, 4, 5],
    [4, 9, 3, 6, 5, 7, 2, 1, 8],
    [8, 6, 7, 2, 3, 1, 9, 5, 4],
    [1, 5, 2, 4, 8, 9, 7, 6, 3],
    [3, 7, 6, 1, 4, 8, 5, 2, 9],
    [9, 2, 8, 3, 6, 5, 4, 7, 1],
    [5, 4, 1, 9, 7, 2, 3, 8, 6]]
```

- Solutions of sudoku puzzles are quickly found by just specifying the rules of the game in SMT format, and apply an SMT solver.
- For several other types of puzzles (kakuro, killer sudoku, binario, …) the SAT/SMT approach to solve or generate them works well too.

SMT 在软件测试中的一个重要应用: 符号执行 (*Symbolic Execution*)

- 文献: *Symbolic Execution for Software Testing: Three Decades Later*

问题: How to explore different program paths and for each path to

- *generate* a set of concrete *input* values exercising that path

- *check* for the presence of *various kinds* of errors

```
int main() {
        x = sym_input();
        y = sym_input();
        testme(x, y);
        return 0;
}
```

```
int twice (int v) {
        return 2*v;
}

void testme (int x, int y) {
        z = twice (y);
        if (z == x) {
                if (x > y+10)
                        ERROR;
                }
        }
}
```
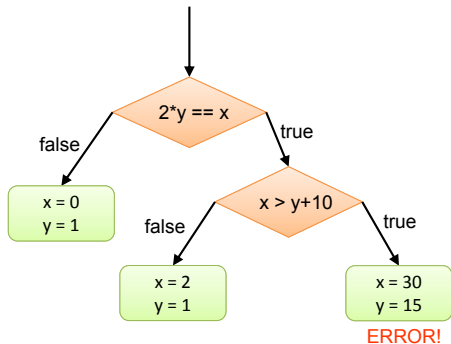
```
int twice (int v) {
        return 2*v;
}

void testme (int x, int y) {
        z = twice (y);
        if (z == x) {
                if (x > y+10)
                        ERROR;
                }
        }
}
```



**问题**: How to reach code ERROR?
SMT problem: Solve $x, y$ satisfying

$$x = 2y \wedge x > y + 10$$

# 作业

实验小作业 1: 使用 pure SAT 求解 N-Queen 问题，并对比 PPT 中 SMT 的实现的效率。要求：
- 代码: 使用 SMT 实现代码，和 PureSAT 实现代码
- 文档: 写出实验记录，要求对比 N 取值不同时，两者的效率

实验小作业 2(二选一): 使用 pure SAT 求解 d=a+b 或 d=a-b，其中,a,b 为正整数。要求：
- 加法和减法问题仅需做一题，减法的实现分数更高
- 代码
- 文档: 简要写出编码思路，代码使用文档和实验结果

实验大作业 (可选): 分别用 Z3 和自己设计的算法求解 rectangle fitting。要求：
- 代码: (1) 使用 Z3 实现 PPT 中的设计 (2) 自己设计算法（使用 C、C++ 语言等）
- 文档: (1) 解释自己的算法思路 (2) 设计测试集，对比两个方法的效率

# 本章大作业参考论文

大作业可参考论文 (但不限于下列论文):

- 应用
  - Notary: A device for secure transaction approval
- 工具实现
  - Z3: An Efficient SMT Solver
  - Deep Cooperation of CDCL and Local Search for SAT