
1

2ITX0 Applied Logic, fall 2019

Topic **SAT solving**

Teacher: Hans Zantema

We follow a big part of the MOOC (massive open online course)
https://www.coursera.org/learn/automated-reasoning-sat
more precisely: the first three lectures of both week (= module) 1 and week 2, the first four lectures of week 3 and the first two lectures of week 4

First we define SAT(isfiability) and give some basic observations

---

2

A **propositional formula** is composed from Boolean variables and the operators

- $\neg$ (negation, 'not')
- $\vee$ (disjunction, 'or')
- $\wedge$ (conjunction, 'and')
- $\rightarrow$ (implication, $p \rightarrow q \equiv \neg p \vee q$)
- $\leftrightarrow$ (bi-implication, $p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$)

3

A propositional formula is **satisfiable, SAT** if it is possible to give values to the variables such that the formula yields true

The mapping of the variables to these values is the **satisfying assignment**

4

Example:

$$(p \vee q) \wedge (\neg p \vee \neg q) \wedge (p \vee \neg q)$$

is satisfiable

Choose the satisfying assignment $p =$ true and $q =$ false, for which indeed all three parts

$$p \vee q,\ \neg p \vee \neg q,\ p \vee \neg q$$

yield true

5

Example:

$$(p \vee q) \ \wedge \ (\neg p \vee \neg q) \wedge$$

$$(\neg p \vee q) \ \wedge \ (p \vee \neg q)$$

is **unsatisfiable** since for every choice of the variables, one of the four parts

$$p \vee q, \ \neg p \vee \neg q, \ p \vee \neg q, \ p \vee \neg q$$

yields false

6

SAT is the decision problem:

given a propositional formula, is it satisfiable?

Basic method for SAT: **truth tables**

Essentially this consists of computing the values of the formula for all 2^n ways to choose values $1 =$ true or $0 =$ false for the n variables

The formula is satisfiable if and only if at least one of these 2^n cases yields $1 =$ true

---

7

Filling the truth table of any formula is based on the truth tables of the building blocks $\neg$, $\vee$, $\wedge$, $\rightarrow$, $\leftrightarrow$:

p	$\neg p$
0	1
1	0

p	q	$p \vee q$
0	0	0
0	1	1
1	0	1
1	1	1

p	q	$p \wedge q$
0	0	0
0	1	0
1	0	0
1	1	1

p	q	$p \rightarrow q$
0	0	1
0	1	1
1	0	0
1	1	1

p	q	$p \leftrightarrow q$
0	0	1
0	1	0
1	0	0
1	1	1

8

Example: $\neg((p \wedge q) \rightarrow (p \vee r))$

p	q	r	$p \wedge q$	$p \vee r$	$(p \wedge q) \rightarrow (p \vee r)$	$\neg((p \wedge q) \rightarrow (p \vee r))$
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

9

Example: $\neg((p \wedge q) \rightarrow (p \vee r))$

p	q	r	$p \wedge q$	$p \vee r$	$(p \wedge q) \rightarrow (p \vee r)$	$\neg((p \wedge q) \rightarrow (p \vee r))$
0	0	0	0			
0	0	1	0			
0	1	0	0			
0	1	1	0			
1	0	0	0			
1	0	1	0			
1	1	0	1			
1	1	1	1			

10

Example: $\neg((p \wedge q) \rightarrow (p \vee r))$

p	q	r	$p \wedge q$	$p \vee r$	$(p \wedge q) \rightarrow (p \vee r)$	$\neg((p \wedge q) \rightarrow (p \vee r))$
0	0	0	0	0		
0	0	1	0	1		
0	1	0	0	0		
0	1	1	0	1		
1	0	0	0	1		
1	0	1	0	1		
1	1	0	1	1		
1	1	1	1	1		

11

Example: $\neg((p \wedge q) \rightarrow (p \vee r))$

p	q	r	$p \wedge q$	$p \vee r$	$(p \wedge q) \rightarrow (p \vee r)$	$\neg((p \wedge q) \rightarrow (p \vee r))$
0	0	0	0	0	1	
0	0	1	0	1	1	
0	1	0	0	0	1	
0	1	1	0	1	1	
1	0	0	0	1	1	
1	0	1	0	1	1	
1	1	0	1	1	1	
1	1	1	1	1	1	

12

Example: $\neg((p \wedge q) \rightarrow (p \vee r))$

p	q	r	$p \wedge q$	$p \vee r$	$(p \wedge q) \rightarrow (p \vee r)$	$\neg((p \wedge q) \rightarrow (p \vee r))$
0	0	0	0	0	1	0
0	0	1	0	1	1	0
0	1	0	0	0	1	0
0	1	1	0	1	1	0
1	0	0	0	1	1	0
1	0	1	0	1	1	0
1	1	0	1	1	1	0
1	1	1	1	1	1	0

Only zeros in final column $\Rightarrow \neg((p \wedge q) \rightarrow (p \vee r))$ is **unsatisfiable**

13

This truth table method always works, but is **exponential** in the number of variables

Infeasible for big n, so we need something better

Bad news: SAT is NP-complete
(being a theoretical standard for 'hard' problems)

So it is unlikely that a SAT solver exists that is efficient for all formulas

In fact, SAT was the starting point of research in NP-completeness (1970)

14

Good news, big surprise:

Current SAT solvers are successful for several big formulas

In a later lecture we will see that solving the n-queens problem for $n = 100$ yields a 50 Mb formula over 10,000 variables, but is solved in 10 seconds by the SAT solver Z3

Success stories are from a wide range of application areas

So:
NP-hard is not always hard

15

In the rest of these first 4 lectures we will see

- Extension to SMT, to deal with numbers and inequalities

- Tools for SAT/SMT, the SMT-LIB syntax
- Examples, applications
- Underlying mechanisms / theory

16

SMT solving, syntax and tools

We saw that **SAT** = satisfiability is about **propositional formulas**, that is, composed from Boolean variables and the operators $\neg$, $\vee$, $\wedge$, $\rightarrow$ and $\leftrightarrow$

SMT means **satisfiability modulo theories**, and is about SAT extended by extra theory

Here we focus on the theory of **linear inequalities**, either on real numbers or integers

---

17

Again a formula is **satisfiable, SAT** if it is possible to give values to the variables such that the formula yields true

The mapping of the variables to these values is the **satisfying assignment**

Example:

For integer variables a, b, c, d the SMT formula

$$2a > b + c \ \wedge \ 2b > c + d \wedge$$

$$2c > 3d \ \wedge \ 3d > a + c$$

is satisfiable with satisfying assignment

$$a = 30, b = 27, c = 32, d = 21$$

18

Tools

For checking satisfiability of SMT formulas and finding satisfying assignments there are several powerful tools, like **Z3**, **YICES** and **CVC4**

We will use **Z3**, to be downloaded from `https://github.com/Z3Prover`, or to be used on-line via `https://rise4fun.com/z3`

For non-commercial use they are free to download and to use

Most of them have their own syntax, but all accept the syntax of the SMT-LIB standard

Now we show how to use the syntax of SMT-LIB version 2

19

The tools allow to be used interactively, but we want to apply them on big formulas, typically generated by some program and stored in a file

If the file is called `file`, then we will call the tool by typing, e.g.,

```
z3 -smt2 file
```

in a command-line interface

Alternatively, go to `https://rise4fun.com/z3`

Then by cut-and-paste you may enter the content of your file, and let Z3 run online, giving the same output without downloading Z3

Also the output can be stored in a file by doing cut-and-paste

---

20

General ingredients of the file:

- `set-logic` and/or `set-options` (often redundant)
- declarations: `declare-const`, `declare-fun`
- `(assert` $\cdots$ `)` containing the actual formula
- `(check-sat)` to do the actual sat solving
- `(get-model)` to show the satisfying assignment in case of satisfiability

---

21

All operators `>`, `<`, `=`, `>=`, `<=`, `+`, `*`, `and`, `or`, `not`, ... are written in **prefix** notation: always first `(`, then the operator, then the arguments, then `)`

So in this notation the expressions $2a$ and $b + c$ are written as

`(* 2 a)` and `(+ b c)`

Hence the requirement $2a > b + c$ is written as

```
(> (* 2 a) (+ b c))
```

Operators

`+`, `and`, `or`

may have any number of arguments, not only two

For linearity, `*` always has two arguments, one of which is a number

---

22

So our problem of solving

$$2a > b + c \ \wedge \ 2b > c + d \ \wedge \ 2c > 3d \ \wedge \ 3d > a + c$$

can be expressed by

```
(declare-const a Int)
(declare-const b Int)
(declare-const c Int)
(declare-const d Int)
(assert
(and (> (* 2 a) (+ b c)) (> (* 2 b) (+ c d))
     (> (* 2 c) (* 3 d)) (> (* 3 d) (+ a c))))
(check-sat)
(get-model)
```

23

Put this text in a file called `file`, e.g., by doing cut and paste from these slides, or the MOOC examples document

Then calling `z3 -smt2 file` yields as output

```
sat
(model
(define-fun b () Int 27)
(define-fun a () Int 30)
(define-fun c () Int 32)
(define-fun d () Int 21)
)
```

in a fraction of a second, indeed containing the resulting values

$$a = 30, b = 27, c = 32, d = 21$$

24

The same can be done by one function f instead of four separate variables a, b, c, d, where a, b, c, d are replaced by $f(1), f(2), f(3), f(4)$

In SMT syntax $f(1)$ is written as `(f 1)`

```
(declare-fun f (Int) Int)
(assert
(and
(> (* 2 (f 1)) (+ (f 2) (f 3)))
(> (* 2 (f 2)) (+ (f 3) (f 4)))
(> (* 2 (f 3)) (* 3 (f 4)))
(> (* 3 (f 4)) (+ (f 1) (f 3)))))
```

```
(check-sat)
(get-model)
```

25

An example in propositional SAT: check satisfiability of

$$(A \leftrightarrow (D \wedge B)) \wedge (C \rightarrow B) \wedge \neg(A \vee B \vee \neg D) \wedge ((\neg A \wedge C) \vee D)$$

use `implies` for implication ($\rightarrow$), and `iff` for bi-implication ($\leftrightarrow$)

```
(declare-const A Bool)
(declare-const B Bool)
(declare-const C Bool)
(declare-const D Bool)
(assert
(and
(iff A (and D B))
(implies C B)
(not (or A B (not D)))
(or (and (not A) C) D)
))
(check-sat)
(get-model)
```

26

SMT solvers can deal with big numbers: compute

$$C = \frac{87 * 98798798987987987987923423 + 93 * 763429999988888888887364578645}{2}$$

```
(declare-const A Int)
(declare-const B Int)
(declare-const C Int)
(assert (and
(= A 98798798987987987987923423)
(= B 763429999988888888887364578645)
(= (+ (* 87 A) (* 93 B)) (+ C C))))
(check-sat)
(get-model)
```

yielding the value $C = 39797242755460810810739927575893$

27

A convenient operation is **if-then-else**, abbreviated to `ite`

It has always three arguments:

- the first is the **condition**, which is boolean,
- the second is the result in case this condition is true, and
- the third is the result in case this condition is false

28

Example:

```
(ite (< a b) 13 (* 3 a))
```

yields the value 13 in case $a < b$, otherwise it yields $3a$

Example:

```
(+
(ite p 1 0)
(ite q 1 0)
(ite r 1 0))
```

yields the number of variables that are true among the boolean variables p, q, r

---

29

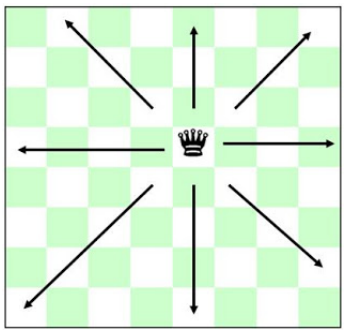
Summarizing,

- We saw how SAT can be extended to SMT, to deal with linear inequalities
- SMT solvers like **Z3**, **YICES** and **CVC4** accept formulas in the SMT-LIB syntax
- Next we will give several applications, but you may start by playing around right now

---

30

Eight Queens Problem



Can we put 8 queens on the chess board in such a way that no two may hit each other?

31

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

Here it can be done in 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

---

32

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}

33

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 $p_{i'j'}$ in same row: $i = i'$

34

Row i:

$$p_{i1}, p_{i2}, p_{i3}, p_{i4}, p_{i5}, p_{i6}, p_{i7}, p_{i8}$$

At **least** one queen on row i:

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

Shorthand:

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

35

Row i:

$$p_{i1}, p_{i2}, p_{i3}, p_{i4}, p_{i5}, p_{i6}, p_{i7}, p_{i8}$$

At **most** one queen on row i?

That is: for every $j < k$ not both p_{ij} and p_{ik} are true

So $\neg p_{ij} \vee \neg p_{ik}$ for all $j < k$

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

36

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}

Column requirements for p_{ij}: same as for rows with i, j swapped

---

37

Requirements until now:

At least one queen on every row:

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

At most one queen on every row:

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

38

And similar for the columns:

At least one queen on every column:

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

At most one queen on every column:

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

39

Remains to express:

At most one queen on every diagonal

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

$$\Longleftrightarrow$$

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

40

Diagonal in other direction:

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

$$\iff$$

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

41

So for 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

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

42

Total formula:

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

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

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

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

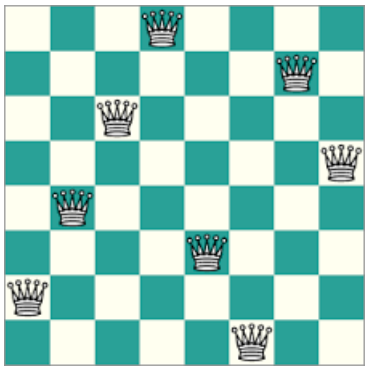
$$\bigwedge_{0<i<i'\leq 8} \Big(\bigwedge_{j,j' : i+j=i'+j' \vee i-j=i'-j'} \neg p_{ij} \vee \neg p_{i'j'} \Big)$$

43

The resulting formula

- looks complicated, but is easily generated in SAT/SMT syntax by some *for* loops
- consists of 740 requirements
- is easily solved by current SAT solver; resulting true values easily give the queen positions

44



45

Trick to find all 92 solutions:

Add negation of found solution to formula, and repeat this until formula is unsatisfiable

Generalizes to n queens on $n \times n$ board, e.g., for $n = 100$ Z3 finds a satisfying assignment of the 50Mb formula within 10 seconds

Same approach applicable to extending a partially filled board, which is an NP-complete problem

46

To conclude: we expressed a chess board problem in a pure SAT problem

Generate this formula in the appropriate syntax by a small program

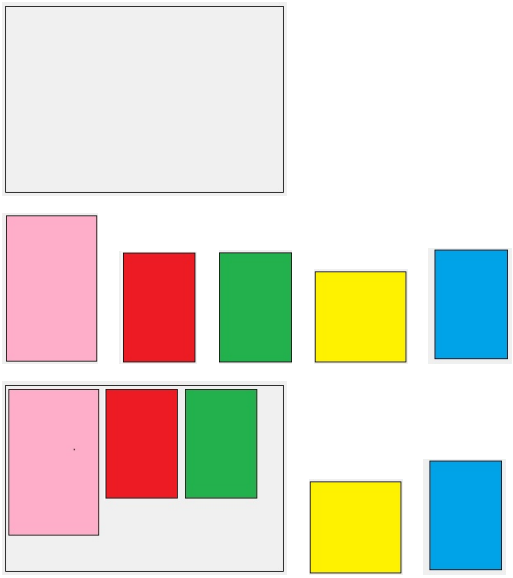
Then a SAT solver immediately finds a solution

47

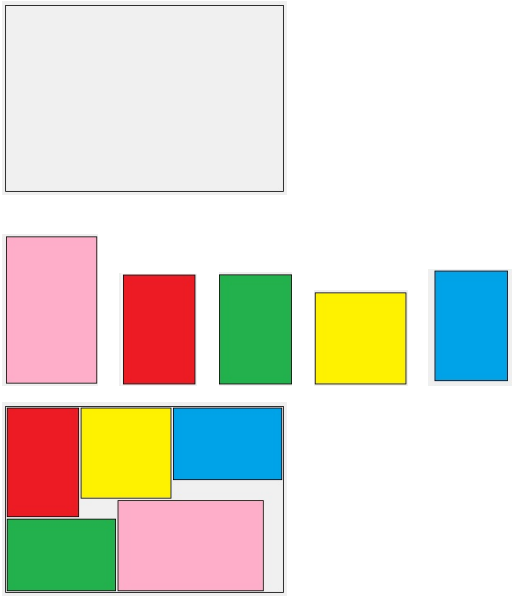
Rectangle fitting

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?

48



49



50

How to specify this problem?

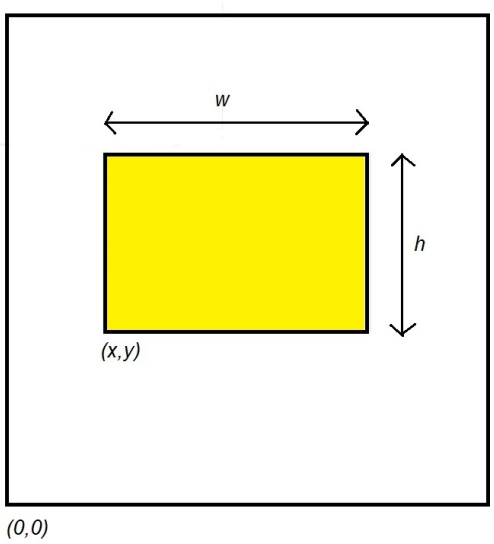
Number rectangles from 1 to n

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

---

51



52

Collect all requirements in a formula

width / height requirements:

First rectangle has width 4 and height 6:

$$(w_1 = 4 \wedge h_1 = 6) \vee (w_1 = 6 \wedge h_1 = 4)$$

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

53

Fit in big rectangle:

Let

- $(0, 0)$ = lower left corner of big rectangle
- W = width of big rectangle
- H = height of big rectangle

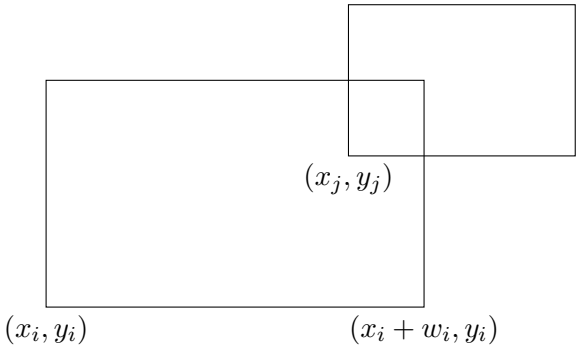
Requirements:

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

and $y_i \geq 0 \ \wedge \ y_i + h_i \leq H$
for all $i = 1, \ldots, n$

54

No overlap?



If overlap, then $x_i + w_i > x_j$

55

Overlap:

Rectangles i and j overlap if

$$x_i + w_i > x_j$$

(right side of rectangle i is right from left side of rectangle j)

and $x_i < x_j + w_j$ (left side of rectangle i is left from right side of rectangle j)

and $y_i + h_i > y_j$ (top of rectangle i is above bottom of rectangle j)

and $y_i < y_j + h_j$ (bottom of rectangle i is below top of rectangle j)

56

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

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

or, equivalently

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

57

Summarizing, the full formula reads

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

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

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

58

The resulting formula

- exactly describes all requirements of our rectangle fitting problem
- is composed from linear inequalities and propositional operations $\neg, \wedge, \vee$
- hence is perfectly suitable for SMT solvers

The formula is satisfiable **if and only if** our fitting problem has a solution

---

59

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

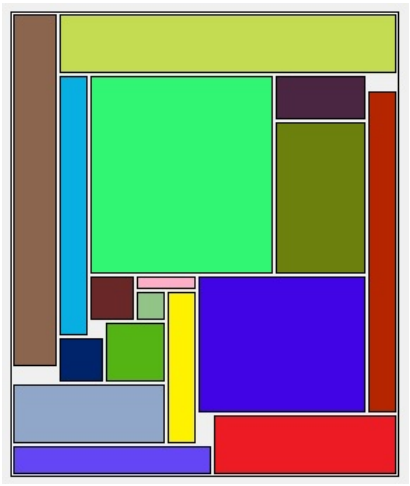
From these values the intended solution is easily obtained

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

For generating the formula in SMT syntax (typically hundreds of lines):

better generate by a script/program than edit yourself

60



61

Solving a **Sudoku** by SAT/SMT

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

62

Rules of the game:

Fill the blank cells in such a way that

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

contains the numbers 1 to 9, all occurring exactly once

Doing by hand the first steps are easy

63

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

64

After a few steps this particular puzzle becomes very hard, and backtracking and/or advanced solving techniques are required

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

How?

65

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

66

How to specify that every row (and column, and 3×3 block) contains the numbers 1 to 9, all occurring once?

Several ways

We choose to specify that these numbers are ≥ 1 and ≤ 9, and are all distinct

We could work out being distinct, but the SMT syntax has the special construct `(distinct .... )`

67

Very straightforward:

Declaration

```
(declare-fun A (Int Int) Int)
```

Requirements:

```
(<= 1 (A 1 1)) (>= 9 (A 1 1))
```

and similar for all other $i, j = 1, \ldots, 9$

Elements in row 1 are distinct:

```
(distinct (A 1 1) (A 1 2) (A 1 3) (A 1 4)
(A 1 5) (A 1 6) (A 1 7) (A 1 8) (A 1 9))
```

and similar for rows 2 to 9

68

Similar for every column and every 3×3 block, like

```
(distinct
(A 1 1) (A 1 2) (A 1 3)
(A 2 1) (A 2 2) (A 2 3)
(A 3 1) (A 3 2) (A 3 3))
```

The full formula is concluded by the given numbers in the sudoku puzzle:

```
(= 9 (A 1 3))
(= 8 (A 1 4))
(= 5 (A 1 5))
(= 6 (A 1 6))
(= 8 (A 2 2))
...
```

69

Applying Z3 (or any other SMT solver) on the resulting formula yields a satisfying assignment giving values for all 81 variables, being the solution of the sudoku puzzle

The computation only takes a fraction of a second

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

70

Great part of the formula is the same for all sudoku puzzles:

- ≥ 1 and ≤ 9 for all 81 cells
- distinctness for all 9 rows, 9 columns and 9 blocks

Rest of the formula specifies the values given in the sudoku puzzle

---

71

Method to generate Sudoku puzzles:

- Start by a full sudoku = solution of any sudoku puzzle
- Remove given numbers one by one until the formula extended by the **negation** of the known solution is satisfiable
- Then it has another solution than the known one, but one step before it had not, so the puzzle of one step before is one with a unique solution

---

72

Concluding:

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

---

73

Planning / scheduling by SAT/SMT

Projects to plan / schedule, typically split up into parts, let's call them **jobs**

Assume running times of the jobs are known / estimated, and it has to be decided which jobs start at which moment

Typical requirements:

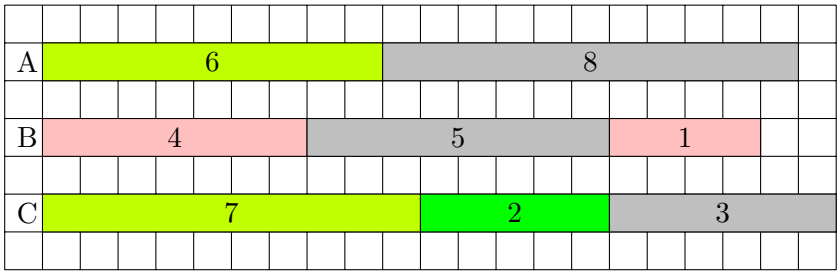
- some jobs should run before others
- people are assigned to jobs
- jobs may need particular resources with limited availability
- jobs may run in parallel or not
- ...

74

Example

- Jobs 1 to 8, having running times 4,5,6,7,8,9,10,11
- Each job should be executed by one of three persons A, B, C
- Jobs 1 and 4 need the specialism of B
- Job 2 should run after jobs 6 and 7 have finished

Solution in total running time 21:



75

How to solve this?

Specify the problem and call an SMT-solver

For every job i typically introduce

- start time $S(i)$
- end time $E(i)$
- person $P(i)$ to execute it

76

Specify running time of job i:

$$E(i) - S(i) = \cdots$$

Job j should run after job i has finished:

$$S(j) \geq E(i)$$

Jobs i and j should not run in parallel (e.g., need a shared resource)

$$S(j) \geq E(i) \ \vee \ S(i) \geq E(j)$$

Number n of available persons:

$$1 \leq P(i) \leq n$$

for all jobs i

Job i should be executed by person p:

$$P(i) = p$$

77

At every moment every person is involved in at most one job:

$$(P(i) = P(j)) \rightarrow (S(j) \geq E(i) \vee S(i) \geq E(j))$$

for all jobs $i < j$

Minimize total running time?

Introduce variable T for total running time and add requirements

$$S(i) \geq 0 \ \wedge \ E(i) \leq T$$

for all jobs i

Now we want to find the **smallest** value of T such that the total formula is satisfiable

---

78

How?

Two approaches:

- Add $T = v$ for several values of v until a value v is found for which it is satisfiable and for which for $v - 1$ it is unsatisfiable

  (quickly by binary search)

- Use built in feature `(minimize T)` as is available in some SMT solvers

---

79

Concluding

By just introducing variables for starting times of the jobs and specifying all requirements in these variables, SMT solvers succeed in solving scheduling problems

Several variants are possible, like finding class schedules for schools

Does not always scale up

80

Resolution

Basic method for satisfiability of propositional formulas

Still the basis of current SAT solvers

A **propositional formula** is composed from Boolean variables and the operators

- $\neg$ (negation, 'not')
- $\vee$ (disjunction, 'or')
- $\wedge$ (conjunction, 'and')
- $\rightarrow$ (implication, $p \rightarrow q \equiv \neg p \vee q$)
- $\leftrightarrow$ (bi-implication, $p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$)

81

A propositional formula is **satisfiable, SAT** if it is possible to give values to the variables such that the formula yields true

Resolution is only applicable to formulas of a particular shape, namely **conjunctive normal form (CNF)**

A **conjunctive normal form (CNF)** is a conjunction of clauses

A **clause** is a disjunction of literals

A **literal** is either a variable or the negation of a variable

Hence a CNF is of the shape

$$\bigwedge_i (\bigvee_j \ell_{ij})$$

where ℓ_{ij} are literals

82

Example:

$$(p \vee q) \ \wedge \ (\neg p \vee \neg q) \wedge$$

$$(\neg p \vee q) \ \wedge \ (p \vee \neg q)$$

is an example of a CNF

It is unsatisfiable since for every choice of the variables, one of the four clauses

$$p \vee q, \ \neg p \vee \neg q, \ p \vee \neg q, \ p \vee \neg q$$

yields false

83

Arbitrary formulas can be transformed to CNFs in a clever way by the **Tseitin transformation**, maintaining satisfiability, by introducing extra variables for subformulas

This makes resolution applicable to arbitrary propositional formulas

---

84

Idea of checking SAT of CNF:

Clauses are properties that we know to be true

From these clauses new clauses are derived, trying to derive a contradiction: the empty clause $\bot$

If this succeeds, we know that the CNF is unsatisfiable

Surprisingly, we need only one rule, the **resolution rule**

85

Resolution rule

This rule states that if there are clauses of the shape $p \vee V$ and $\neg p \vee W$, then the new clause $V \vee W$ may be added

This is correct by case analysis on p and $\neg p$

Formal notation of resolution rule:

$$\frac{p \vee V, \quad \neg p \vee W}{V \vee W}$$

86

Order of literals in a clause does not play a role since

$$p \vee q \equiv q \vee p$$

Double occurrences of literals may be removed since

$$p \vee p \equiv p$$

Think of a clause as a **set** of literals

Think of a CNF as a **set** of clauses

87

Example

We prove that the CNF consisting of the following clauses 1 to 5 is unsatisfiable

1	$p \vee q$	
2	$\neg r \vee s$	
3	$\neg q \vee r$	
4	$\neg r \vee \neg s$	
5	$\neg p \vee r$	

6	$p \vee r$	$(1, 3, q)$
7	r	$(5, 6, p)$
8	s	$(2, 7, r)$
9	$\neg r$	$(4, 8, s)$
10	$\bot$	$(7, 9, r)$

88

Remarks:

- Lot of freedom in choice: several other sequences of resolutions steps will lead to $\bot$ too
- Resolution steps on p in which V contains q and W contains $\neg q$ for some q (or conversely) are allowed but useless

  In that case the new clause $V \vee W$ is of the shape $q \vee \neg q \vee \cdots$ and hence equivalent to *true*, not containing fruitful information
- If a clause consists of a single literal ℓ (a **unit clause**) then the resolution rule allows to remove the literal $\neg\ell$ from a clause containing $\neg\ell$

  This is called **unit resolution**

89

Unit resolution corresponds to the instances of the general resolution rule

$$\frac{p \vee V, \quad \neg p \vee W}{V \vee W}$$

in which either V or W is the empty clause

So

$$\frac{p, \quad \neg p \vee W}{W} \qquad\qquad \frac{p \vee V, \quad \neg p}{V}$$

90

General format of **proof rules**:

$$\frac{A_1, A_2, \ldots, A_n}{C}$$

This means: assuming that assumptions $A_1, A_2, \ldots, A_n$ all hold, then we may conclude that conclusion C holds

Flat notation:

$$A_1, A_2, \ldots, A_n \vdash C$$

or

$$A_1, A_2, \ldots, A_n \models C$$

Most times $\vdash$ is used to apply **syntactical** rules, while $\models$ refers to **semantics**, that is, meaning, value

---

91

We have seen this before in the course **Logic and Set Theory**

To emphasize this **value**, this was denoted as $\overset{val}{\models}$, like in

$$P \wedge Q \overset{val}{\models} P$$

The meaning is: assume that the properties left from the $\overset{val}{\models}$ symbol hold, then we may conclude that the property right from it holds too

---

92

Proof systems

A set of proof rules of this shape (in one of these notations) is called a **proof system**

A proof system is called **sound** if all its rules are valid

A proof system is called **complete** if every property in the corresponding format that holds, can be proved by only using the proof rules

---

93

The goal of resolution is to prove that a CNF is **unsatisfiable**, in the format of the rules corresponding to deriving the empty clause $\perp$

So the proof system only consisting of the resolution rule is called **sound** if from deriving the empty clause $\bot$ we can conclude that the CNF is unsatisfiable

Conversely, it is called **complete** if for every unsatisfiable CNF, the empty clause $\bot$ can be derived by only applying the rule

Soundness follows from correctness of the rule, shown on the next slide, completeness we will show later

Combining these results yields:
a CNF is unsatisfiable if and only if $\bot$ can be derived by only using the resolution rule

---

94

Soundness of resolution

$$\frac{p \vee V, \quad \neg p \vee W}{V \vee W}$$

Assume that $p \vee V$ and $\neg p \vee W$ both hold

Case analysis on p:

It p is true, then from $\neg p \vee W$ we conclude W, so also $V \vee W$

It p is false, then from $p \vee V$ we conclude V, so also $V \vee W$

So in all cases the desired property $V \vee W$ holds

95

How to use resolution?

To prove that Φ is a **tautology** = always true:

prove that $\neg\Phi$ is unsatisfiable

To prove that Φ **implies** Ψ:

prove that $\Phi \wedge \neg\Psi$ is unsatisfiable

To prove that Φ and Ψ are **equivalent**:

prove that $\neg(\Phi \leftrightarrow \Psi)$ (or, equivalently $(\Phi \vee \Psi) \wedge (\neg\Phi \vee \neg\Psi)$) is unsatisfiable

---

96

Example: (free after Lewis Carroll)

1. Good-natured tenured professors are dynamic
2. Grumpy student advisors play slot machines
3. Smokers wearing a cap are phlegmatic

4. Comical student advisors are professors
5. Smoking untenured members are nervous
6. Phlegmatic tenured members wearing caps are comical
7. Student advisors who are not stock market players are scholars
8. Relaxed student advisors are creative
9. Creative scholars who do not play slot machines wear caps
10. Nervous smokers play slot machines
11. Student advisors who play slot machines do not smoke
12. Creative good-natured stock market players wear caps

Then we have to prove that no student advisor is smoking

97

The first step is giving names to every notion to be formalized

name	meaning	opposite
A	good-natured	grumpy
B	tenured	
C	professor	
D	dynamic	phlegmatic
E	wearing a cap	
F	smoke	
G	comical	
H	relaxed	nervous
I	play stock market	
J	scholar	
K	creative	
L	plays slot machine	
M	student advisor	

98

So we have to prove that assuming properties 1 to 12, we can conclude $\neg(M \wedge F)$, stating that no student advisor is smoking

So we have to prove that

$$1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \wedge 6 \wedge 7 \wedge 8 \wedge 9 \wedge 10 \wedge 11 \wedge 12 \wedge M \wedge F$$

is unsatisfiable

Each of the properties 1, 2, 3, ... can be written as a clause, for instance

$$1 : \underbrace{\text{Good-natured}}_{A} \underbrace{\text{tenured}}_{B} \underbrace{\text{professors}}_{C} \text{ are } \underbrace{\text{dynamic}}_{D}$$

reads $(A \wedge B \wedge C) \rightarrow D$

99

This property $(A \wedge B \wedge C) \rightarrow D$ is equivalent to a clause:

$$\begin{aligned} (A \wedge B \wedge C) \rightarrow D &\equiv \\ \neg(A \wedge B \wedge C) \vee D &\equiv \\ \neg A \vee \neg B \vee \neg C \vee D & \end{aligned}$$

Each of the other properties 2 to 12 is transformed to a clause similarly

---

100

We have to prove **unsat** of

1. $\neg A \vee \neg B \vee \neg C \vee D$
2. $A \vee \neg M \vee L$
3. $\neg F \vee \neg E \vee \neg D$
4. $\neg G \vee \neg M \vee C$
5. $\neg F \vee B \vee \neg H$
6. $D \vee \neg B \vee \neg E \vee G$
7. $I \vee \neg M \vee J$
8. $\neg H \vee \neg M \vee K$
9. $\neg K \vee \neg J \vee L \vee E$
10. $H \vee \neg F \vee L$
11. $\neg L \vee \neg M \vee \neg F$
12. $\neg K \vee \neg A \vee \neg I \vee E$
13. M
14. F

101

Unit resolution on M and F: remove $\neg M$ and $\neg F$ everywhere:

1. $\neg A \vee \neg B \vee \neg C \vee D$
2. $A \vee L$
3. $\neg F \vee \neg E \vee \neg D$
4. $\neg G \vee C$
5. $\vee B \vee \neg H$
6. $D \vee \neg B \vee \neg E \vee G$
7. $I \vee J$
8. $\neg H \vee K$
9. $\neg K \vee \neg J \vee L \vee E$
10. $H \vee L$
11. $\neg L$
12. $\neg K \vee \neg A \vee \neg I \vee E$

102

New unit clause $\neg L$ has been created; unit resolution on $\neg L$ = remove literal L everywhere:

- $\neg A \vee \neg B \vee \neg C \vee D$
- A
- $\neg E \vee \neg D$
- $\neg G \vee C$
- $B \vee \neg H$
- $D \vee \neg B \vee \neg E \vee G$
- $I \vee J$
- $\neg H \vee K$
- $\neg K \vee \neg J \vee E$
- H
- $\neg K \vee \neg A \vee \neg I \vee E$

103

Unit resolution on new unit clauses A and H: remove $\neg A$ and $\neg H$ everywhere:

- $\neg B \vee \neg C \vee D$
- $\neg E \vee \neg D$
- $\neg G \vee C$
- B
- $D \vee \neg B \vee \neg E \vee G$
- $I \vee J$
- K
- $\neg K \vee \neg J \vee E$
- $\neg K \vee \neg I \vee E$

104

Unit resolution on new unit clauses B and K: remove $\neg B$ and $\neg K$ everywhere:

1. $\neg C \vee D$
2. $\neg E \vee \neg D$
3. $\neg G \vee C$
4. $D \vee \neg E \vee G$
5. $I \vee J$
6. $\neg J \vee E$
7. $\neg I \vee E$

Now no unit resolution possible any more, but by resolution we continue:

8: $J \vee E$ (5,7,I)

9: E (6,8,J)

creating new unit clause E, and all $\neg E$ may be removed by unit resolution

---

105

1. $\neg C \vee D$

2. $\neg D$

3. $\neg G \vee C$

4. $D \vee G$

5. $\neg C$ (1,2,D)

6. G (2,4,D)

7. $\neg G$ (3,5,C)

8. $\perp$ (6,7,G)

concluding the proof

106

We saw

- how tautologies, implications and equivalences can be proved by proving unsatisfiability of modified formulas
- how this can be combined with resolution to solve a classical puzzle, much more efficient than using truth tables

Resolution strategy: apply unit resolution as long as possible

Internally in Z3 and other tools, a combination of unit resolution and case analysis is used: DPLL

107

DPLL

Algorithm to establish whether a CNF is satisfiable

Originates from paper appeared in 1962 by Davis, Logemann and Loveland, based on earlier paper by Davis and Putnam, justifying 'P' in 'DPLL'

Basis of current SAT/SMT solvers

108

Idea of DPLL:

- First apply **unit resolution** as long as possible

- If you can not proceed by unit resolution or trivial observations then choose a variable p, introduce the cases p and $\neg p$, and for both cases go on recursively

---

109

Here *apply unit resolution as long as possible* means

As long as a clause occurs consisting of one literal ℓ (a unit clause):

- remove $\neg\ell$ from all clauses containing $\neg\ell$ (that's the unit resolution), and
- remove all clauses containing ℓ (since they are redundant)
- also remove ℓ itself, but remember it for constructing a satisfying assignment

In the algorithm we denote this by **unit-resol**

110

The algorithm:

DPLL(X):
$X :=$ unit-resol(X)

if $X = \emptyset$ then return(sat)

if $\bot \notin X$ then

choose variable p in X
DPLL($X \cup \{p\}$)
DPLL($X \cup \{\neg p\}$)

111

- Terminates since every recursive call decreases number of variables: after applying unit-resol on $X \cup \{p\}$ or $X \cup \{\neg p\}$, the variable p does not occur at all
- If 'sat' is returned then all involved (remembered) unit clauses yield a satisfying assignment
- Otherwise, it is a big case analysis yielding $\bot$ for all cases, so unsat

So DPLL is a complete method to establish satisfiability:

it always terminates, and at the end states whether the CNF is sat or unsat

---

112

Example

Consider the CNF consisting of the following nine clauses

1.	$\neg p \vee \neg s$	4.	$p \vee r$	7.	$\neg s \vee t$
2.	$\neg p \vee \neg r$	5.	$p \vee s$	8.	$q \vee s$
3.	$\neg q \vee \neg t$	6.	$r \vee t$	9.	$q \vee \neg r$

No unit resolution possible: choose variable p

Add p, unit resolution:	Add $\neg p$, unit resolution:
$\neg s$ (1), $\neg r$ (2)	r (4), s (5)
q (use $\neg s$, 8), t (use $\neg r$, 6)	q (use r, 9), t (use s, 7)
$\neg t$ (use q, 3)	$\neg t$ (use q, 3)
$\perp$ (use t, $\neg t$)	$\perp$ (use t, $\neg t$)

Both branches yield $\perp$, so original CNF is unsatisfiable

---

113

Example

Consider the CNF consisting of the following eight clauses

1.	$\neg p \vee \neg s$	4.	$p \vee r$	7.	$\neg s \vee t$
2.	$\neg p \vee \neg r$	5.	$p \vee s$	8.	$q \vee s$
3.	$\neg q \vee \neg t$	6.	$r \vee t$		

No unit resolution possible: choose variable p

Add p, unit resolution:	Add $\neg p$, unit resolution:
$\neg s$ (1), $\neg r$ (2)	r (4)
q (use $\neg s$, 8), t (use $\neg r$, 6)	s (5)
$\neg t$ (use q, 3)	t (use s, 7)
$\perp$ (use t, $\neg t$)	$\neg q$ (use t, 3)

Yields satisfying assignment $p = q = \mathit{false}$, $r = s = t = \mathit{true}$

114

Let's recall the example of the non-smoking student advisors

We already saw that after doing unit-resol we arrive at

1. $\neg C \vee D$
2. $\neg E \vee \neg D$
3. $\neg G \vee C$
4. $D \vee \neg E \vee G$
5. $I \vee J$
6. $\neg J \vee E$
7. $\neg I \vee E$

Now we have to choose a variable for case analysis: all choices are correct, but we will show that one choice may be more efficient than another

---

115

1. $\neg C \vee D$
2. $\neg E \vee \neg D$
3. $\neg G \vee C$
4. $D \vee \neg E \vee G$
5. $I \vee J$
6. $\neg J \vee E$
7. $\neg I \vee E$

Choose variable C for case analysis:

Add C, unit resolution:
D (1)
$\neg E$ (use D, 2)
$\neg J$ (use $\neg E$, 6)
$\neg I$ (use $\neg E$, 7)
J (use $\neg I$, 5)
$\perp$ (use $\neg J$, J)

Add $\neg C$, unit resolution:
$\neg G$ (3)
$D \vee \neg E$ (4)
No further unit-resol possible, so we should start a second case analysis on remaining CNF

2. $\neg E \vee \neg D$
4. $D \vee \neg E$
5. $I \vee J$
6. $\neg J \vee E$
7. $\neg I \vee E$

116

2. $\neg E \vee \neg D$
4. $D \vee \neg E$
5. $I \vee J$
6. $\neg J \vee E$
7. $\neg I \vee E$

Add D, unit resolution:
$\neg E$ (2)
$\neg J$ (use $\neg E$, 6)
$\neg I$ (use $\neg E$, 7)
J (use $\neg I$, 5)
$\perp$ (use $\neg J$, J)

Add $\neg D$, unit resolution:
$\neg E$ (4)
$\neg J$ (use $\neg E$, 6)
$\neg I$ (use $\neg E$, 7)
J (use $\neg I$, 5)
$\perp$ (use $\neg J$, J)

So case analysis on C succeeds, but needs a nested case analysis

---

117

Instead we could start by case analysis on E, by which no nested case analysis is needed:

1. $\neg C \vee D$
2. $\neg E \vee \neg D$
3. $\neg G \vee C$
4. $D \vee \neg E \vee G$
5. $I \vee J$
6. $\neg J \vee E$
7. $\neg I \vee E$

Add E, unit resolution:
$\neg D$ (2)
G (use E, $\neg D$, 4)
C (use G, 3)
D (use C, 1)
$\perp$ (use $\neg D$, D)

Add $\neg E$, unit resolution:
$\neg J$ (use $\neg E$, 6)
$\neg I$ (use $\neg E$, 7)
J (use $\neg I$, 5)
$\perp$ (use $\neg J$, J)

118

Concluding,

- DPLL is a complete method for satisfiability, based on unit resolution and case analysis
- Efficiency strongly depends on the choice of the variable

We gave an example that twice a case analysis is needed for one choice, and only one case analysis for another choice, but in bigger formulas one choice may lead to 10,000 times a case analysis, while another choice may lead to only 1000 times a case analysis

- Current SAT solvers follow this scheme, combined with good heuristics for variable choice and several optimizations

---

119

DPLL transforms to resolution

More precisely: if we have a DPLL proof that a CNF X is unsat, then it can be transformed to a sequence of resolution steps starting in X and ending in the empty clause $\bot$

As a consequence, resolution is **complete**:

For every CNF that is unsat by only using resolution we can derive $\bot$

120

Recall DPLL on unsat CNF X

DPLL(X):
$X :=$ unit-resol(X)
if $\bot \notin X$ then
    choose variable p in X
    DPLL($X \cup \{p\}$)
    DPLL($X \cup \{\neg p\}$)

Observation:
By $X :=$ unit-resol(X) unsat is maintained, so both $X \cup \{p\}$ and $X \cup \{\neg p\}$ are unsat too

$\Longrightarrow$ apply transformation recursively

121

Write $X \vdash C$ if by only resolution the clause C can be derived from the CNF X

So we have to show that if

$$X \cup \{p\} \vdash \bot$$

and

$$X \cup \{\neg p\} \vdash \bot$$

then

$$X \vdash \bot$$

122

Assume $X \cup \{p\} \vdash \bot$

Using unit clause p means: remove $\neg p$ from a clause

If resolution is applied on a clause from which $\neg p$ was removed, it also applies if $\neg p$ was not removed:

adding $\neg p$ to V or W or both in

$$\frac{q \vee V, \quad \neg q \vee W}{V \vee W}$$

yields result $V \vee W$ to which $\neg p$ may be added

Applying this on $X \cup \{p\} \vdash \bot$ yields either $X \vdash \neg p$ or $X \vdash \bot$

123

So $X \cup \{p\} \vdash \bot$ yields either $X \vdash \neg p$ or $X \vdash \bot$

Similarly, $X \cup \{\neg p\} \vdash \bot$ yields either $X \vdash p$ or $X \vdash \bot$

If one of the cases yields $X \vdash \bot$ we are done

Otherwise, both p and $\neg p$ are derived from X, from which by one extra resolution step also $\bot$ is derived

So in all cases the DPLL proof of unsat is transformed to a resolution proof $X \vdash \bot$

We illustrate this by the example of the non-smoking student advisor

---

124

Unit-resolution yields the CNF X

$1 : \neg C \vee D$	$5 : I \vee J$
$2 : \neg E \vee \neg D$	$6 : \neg J \vee E$
$3 : \neg G \vee C$	$7 : \neg I \vee E$
$4 : D \vee \neg E \vee G$	

Unit resolution on $X \cup \{E\}$ first removes all $\neg E$: put on **red glasses** 

Continue by unit resolution

8	$\neg C$	$(1, 2)$
9	$\neg G$	$(3, 8)$
10	D	$(4, 9)$
11	$\bot$	$(2, 10)$

Doing the same without red glasses:

8	$\neg C \vee \neg E$	$(1, 2)$
9	$\neg G \vee \neg E$	$(3, 8)$
10	$D \vee \neg E$	$(4, 9)$
11	$\neg E$	$(2, 10)$

125

Similarly, for the same CNF X

$$
\begin{array}{ll}
1 : \neg C \vee D & 5 : I \vee J \\
2 : \neg E \vee \neg D & 6 : \neg J \vee E \\
3 : \neg G \vee C & 7 : \neg I \vee E \\
4 : D \vee \neg E \vee G &
\end{array}
$$

unit resolution on $X \cup \{\neg E\}$ removes all E

yields $\bot$ with **red glasses**,  and E without

The first branch yields $\neg E$, this second E, one more resolution step yields $\bot$

The same approach applies for more complicated nested examples

---

126

Concluding,

We sketched how to transform a DPLL proof of unsat to a resolution proof yielding $\bot$

As a consequence, resolution is **complete**:
every unsat CNF admits a resolution proof yielding $\bot$

To be really sure, this sketch should be extended to a **formal proof**

Typically, it will use **induction** and separate **lemma**s stating the properties that are needed

The size of the resulting resolution proof coincides with the size of the DPLL proof

---

127

How to establish satisfiability of an arbitrary propositional formula that is not a CNF?

Transform to CNF, preserving satisfiability

Then after this transformation a DPLL based SAT solver may be applied that needs a CNF as input

Current SAT solvers do this automatically

128

First approach:

Find a CNF that is logically equivalent to the given formula

 always possible


resulting CNF is often unacceptably large: the size can be exponential in the size of the original formula

First we show how it is possible, and why it may blow up

For any formula Φ we can make its truth table

129

For any 0 in this truth table we can make a corresponding clause

p	q	r	Φ	$p \vee \neg q \vee r$	$\neg p \vee q \vee \neg r$	$\neg p \vee \neg q \vee \neg r$
0	0	0	1	1	1	1
0	0	1	1	1	1	1
0	1	0	0	0	1	1
0	1	1	1	1	1	1
1	0	0	1	1	1	1
1	0	1	0	1	0	1
1	1	0	1	1	1	1
1	1	1	0	1	1	0

Now the conjunction $(p \vee \neg q \vee r) \wedge (\neg p \vee q \vee \neg r) \wedge (\neg p \vee \neg q \vee \neg r)$ of these clauses has the same truth table as Φ, so it is logically equivalent to Φ

130

This approach always works: if the truth table of Φ contains k 0's, then we obtain a CNF consisting of k clauses

Drawback: this k may be very large:

if there are n variables, then the truth table has 2^n rows: **exponential** in n

All of these clauses have exactly n literals

A smaller CNF logically equivalent to Φ may exist, having clauses of $< n$ literals

Example: $p \wedge (\neg q \vee r)$ is a CNF with 2 clauses, having 5 0's in truth table of 8 rows

For some formulas the exponential number of clauses is **unavoidable**

131

Example: $\Phi : (\cdots((p_1 \leftrightarrow p_2) \leftrightarrow p_3) \cdots \leftrightarrow p_n)$

This formula yields true if and only if an even number of p_i's has the value false

Claim
Let X be a CNF satisfying $\Phi \equiv X$
Then every clause C in X contains exactly n literals

Proof:

Assume not, then some p_i does not occur in a clause C of X

Then you can give values to the remaining variables such that C is false, and X is false too, independent of the value of p_i

Swapping values of p_i does not swap values of X, contradicting $\Phi \equiv X$ $\quad \square$

132

The truth table of Φ contains 2^n rows, half of which containing 0

So exactly 2^{n-1} zeros

Every clause of exactly n literals has one 0 in its truth table

So we need 2^{n-1} such clauses to obtain the truth table of Φ

So for this Φ the exponential size is unavoidable

So we need something else: the **Tseitin transformation**

133

Tseitin transformation

Linear transformation of arbitrary propositional formula to CNF, preserving satisfiability

Originates from 1966 paper by Tseitin

Applied in current SAT solvers

Key idea:

Give a name to every subformula (except literals) and use this name as a **fresh** variable

For every formula Φ on ≤ 3 variables there is a small CNF $\mathit{cnf}(\Phi) \equiv \Phi$

Transform a big formula Φ to the conjunction of $\mathit{cnf}(\Phi_i)$ for many small formulas Φ_i obtained from Φ, one for each subformula

---

134

More precisely, for every subformula Ψ we define

- $n_\Psi = \Psi$ if Ψ is a literal
- $n_\Psi =$ the name of Ψ, otherwise

The **Tseitin transformation** $T(\Phi)$ of Φ, is defined to be the CNF consisting of:

- n_Φ
- $\mathit{cnf}(q \leftrightarrow \neg n_\Psi)$ for every non-literal subformula of the shape $\neg\Psi$ having name q

- $cnf(q \leftrightarrow (n_{\Psi_1} \diamond n_{\Psi_2}))$ for every subformula of the shape $\Psi_1 \diamond \Psi_2$ having name q, for $\diamond \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$

135

Example Φ:

$$\underbrace{(\underbrace{\neg s \wedge p}_{B}) \leftrightarrow (\underbrace{(\underbrace{q \rightarrow r}_{D}) \vee \neg p}_{C})}$$

yields $T(\Phi)$, writing $n_\Phi = A$:

$A \wedge$

$cnf(A \leftrightarrow (B \leftrightarrow C)) \wedge$

$cnf(B \leftrightarrow (\neg s \wedge p)) \wedge$

$cnf(C \leftrightarrow (D \vee \neg p)) \wedge$

$cnf(D \leftrightarrow (q \rightarrow r))$

136

Preservation of satisfiability

Let Φ is satisfiable, then it admits a satisfying assignment

Extend this to n_Ψ for subformulas Ψ: n_Ψ gets the value of the subformula Ψ

Then by construction this yields a satisfying assignment for $T(\Phi)$:

- n_Φ yields true
- $q \leftrightarrow \neg n_\Psi$ yields true for subformula $\neg\Psi$ with name q, so does $cnf(q \leftrightarrow \neg n_\Psi)$
- $q \leftrightarrow (n_{\Psi_1} \diamond n_{\Psi_2})$ yields true for subformula $\Psi_1 \diamond \Psi_2$ having name q, for $\diamond \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$, so does $cnf(q \leftrightarrow (n_{\Psi_1} \diamond n_{\Psi_2}))$

So satisfiability of Φ implies satisfiability of $T(\Phi)$

137

Conversely, assume $T(\Phi)$ is satisfiable = admits a satisfying assignment

Apply this same satisfying assignment to the original formula Φ

Since $q \leftrightarrow (n_{\Psi_1} \diamond n_{\Psi_2})$ yields true for every subformula $\Psi_1 \diamond \Psi_2$ having name q (and similar for $\neg n_\Psi$), we obtain that every subformula Ψ of Φ gets the value of n_Ψ

Since n_Φ yields true (as part of $T(\Phi)$), we obtain that the original formula Φ yields true, so Φ is satisfiable

Combining both directions:
Theorem Φ is satisfiable if and only if $T(\Phi)$ is satisfiable

---

138

In our example $\Phi = (\underbrace{\neg s \wedge p}_{B}) \leftrightarrow (\underbrace{(\underbrace{q \rightarrow r}_{D}) \vee \neg p}_{C})$ with $n_\Phi = A$:

Φ is satisfiable with satisfying assignment $p = q = s = \text{true}$, $r = \text{false}$

Then $B = D = C = \text{false}$, $A = \text{true}$

Using these values for A, B, C, D yields a satisfying assignment for
$A \wedge$
$A \leftrightarrow (B \leftrightarrow C) \wedge$
$B \leftrightarrow (\neg s \wedge p) \wedge$
$C \leftrightarrow (D \vee \neg p) \wedge$
$D \leftrightarrow (q \rightarrow r)$

so also of the equivalent formula $T(\Phi)$

139

Conversely, a satisfying assignment for $T(\Phi)$ yields true for

$A \wedge$
$A \leftrightarrow (B \leftrightarrow C) \wedge$
$B \leftrightarrow (\neg s \wedge p) \wedge$
$C \leftrightarrow (D \vee \neg p) \wedge$
$D \leftrightarrow (q \rightarrow r)$

so $B \leftrightarrow C$ is true,
in which we may replace B by $\neg s \wedge p$,
and C by $D \vee \neg p$,
and D by $q \rightarrow r$

The resulting formula is Φ, and still yields true

140

In order to get a real CNF we still need to compute the formulas $\mathit{cnf}(n_\Psi \leftrightarrow \cdots)$, being a CNF equivalent to $n_\Psi \leftrightarrow \cdots$

These are simple:

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

$$cnf(p \leftrightarrow (q \vee r)) = (\neg p \vee q \vee r) \wedge (p \vee \neg q) \wedge (p \vee \neg r)$$

$$cnf(p \leftrightarrow (q \wedge r)) = (p \vee \neg q \vee \neg r) \wedge (\neg p \vee q) \wedge (\neg p \vee r)$$

$$cnf(p \leftrightarrow (q \leftrightarrow r)) = (p \vee q \vee r) \wedge (p \vee \neg q \vee \neg r) \wedge (\neg p \vee q \vee \neg r) \wedge (\neg p \vee \neg q \vee r)$$

141

Easy to compute, e.g.,

$$p \leftrightarrow (q \vee r) \equiv (p \rightarrow (q \vee r)) \wedge ((q \vee r) \rightarrow p)$$

$$p \rightarrow (q \vee r) \equiv \neg p \vee q \vee r$$

$$(q \vee r) \rightarrow p \equiv \neg(q \vee r) \vee p \equiv (\neg q \wedge \neg r) \vee p \equiv (\neg q \vee p) \wedge (\neg r \vee p)$$

So

$$cnf(p \leftrightarrow (q \vee r)) = (\neg p \vee q \vee r) \wedge (p \vee \neg q) \wedge (p \vee \neg r)$$

142

Concluding,

- For every propositional formula Φ its Tseitin transformation $T(\Phi)$ is easily computed
- Size of $T(\Phi)$ is linear in size of Φ
- Preserves satisfiability
- Not only CNF, even 3-CNF
- Satisfying assignment for $T(\Phi)$ easily gives a satisfying assignment for Φ by simply ignoring the fresh variables

---

143

In this way the whole DPLL machinery is available for arbitrary propositional formulas: first apply Tseitin transformation and then apply DPLL

This is exactly what happens internally when applying Z3 on a propositional formula

For dealing with linear inequalities more techniques are needed, in particular the **simplex algorithm**, which is integrated in the underlying DPLL mechanism

Not presented here; this concludes our theoretical basis of SAT/SMT

Next two weeks = four lectures will be about **information theory**

The last part of the course will be about **program correctness** in which SAT/SMT comes back as an important technique