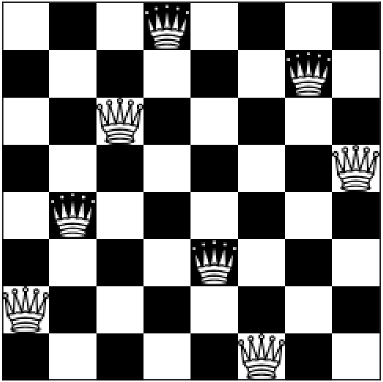
Introduction to SAT (constraint) solving

Justyna Petke

SAT, SMT and CSP solvers are used for solving problems involving constraints.

The term “constraint solver”, however, usually refers to a CSP solver.

The 8-queens problem



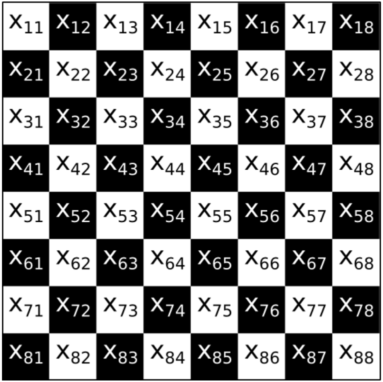
The Boolean satisfiability problem (SAT)
is the problem of deciding whether there is a variable
assignment that satisfies a given propositional formula.

SAT example

$$x_1 \lor x_2 \lor \neg x_4$$
$$\neg x_2 \lor \neg x_3$$

- x_i : a Boolean variable
- x_i, $\neg x_i$: a literal
- $\neg x_2 \lor \neg x_3$: a clause

The 8-queens problem



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

The 8-queens problem : SAT model

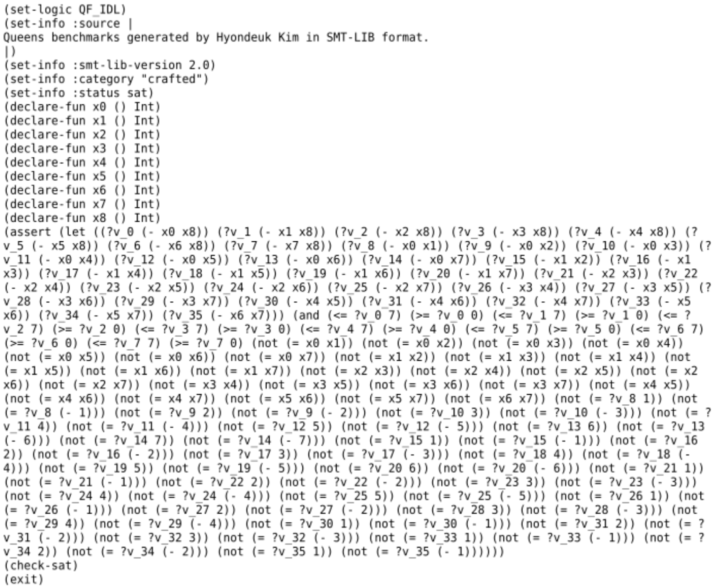
```
p cnf 64 744
1 2 3 4 5 6 7 8 0
-1 -2 0
-1 -3 0
-1 -4 0
-1 -5 0
-1 -6 0
-1 -7 0
-1 -8 0
-2 -3 0
-2 -4 0
-2 -5 0
-2 -6 0
-2 -7 0
-2 -8 0
-3 -4 0
-3 -5 0
-3 -6 ..
```

The Satisfiability Modulo Theories (SMT)
is the problem of deciding whether there is a variable
assignment that satisfies a given formula in first order logic with
respect to a background theory.

Example background theories for SMT

- Equality with Uninterpreted Functions
  (e.g. $f(x) = y \land f(x) \neq y$ is UNSAT)
- Non-linear arithmetic (e.g. $x^2 + yz \leq 10$)
  : variables can be reals
- Arrays (e.g. $write(a, x, 3) = b$, $read(a, x) = b$)
- Bit vectors (e.g. $x[0:1] \neq y[0:1]$)

The 8-queens problem : SMT model



```
(set-logic QF_IDL)
(set-info :source |
Queens benchmarks generated by Hyondeuk Kim in SMT-LIB format.
|)
(set-info :smt-lib-version 2.0)
(set-info :category "crafted")
(set-info :status sat)
(declare-fun x0 () Int)
(declare-fun x1 () Int)
(declare-fun x2 () Int)
(declare-fun x3 () Int)
(declare-fun x4 () Int)
(declare-fun x5 () Int)
(declare-fun x6 () Int)
(declare-fun x7 () Int)
(declare-fun x8 () Int)
(assert (let ((?v_0 (- x0 x8)) (?v_1 (- x1 x8)) (?v_2 (- x2 x8)) (?v_3 (- x3 x8)) (?v_4 (- x4 x8)) (?
v_5 (- x5 x8)) (?v_6 (- x6 x8)) (?v_7 (- x7 x8)) (?v_8 (- x0 x1)) (?v_9 (- x0 x2)) (?v_10 (- x0 x3)) (?
v_11 (- x0 x4)) (?v_12 (- x0 x5)) (?v_13 (- x0 x6)) (?v_14 (- x0 x7)) (?v_15 (- x1 x2)) (?v_16 (- x1
x3)) (?v_17 (- x1 x4)) (?v_18 (- x1 x5)) (?v_19 (- x1 x6)) (?v_20 (- x1 x7)) (?v_21 (- x2 x3)) (?v_22
(- x2 x4)) (?v_23 (- x2 x5)) (?v_24 (- x2 x6)) (?v_25 (- x2 x7)) (?v_26 (- x3 x4)) (?v_27 (- x3 x5)) (?
v_28 (- x3 x6)) (?v_29 (- x3 x7)) (?v_30 (- x4 x5)) (?v_31 (- x4 x6)) (?v_32 (- x4 x7)) (?v_33 (- x5
x6)) (?v_34 (- x5 x7)) (?v_35 (- x6 x7))) (and (<= ?v_0 7) (>= ?v_0 0) (<= ?v_1 7) (>= ?v_1 0) (<= ?
v_2 7) (>= ?v_2 0) (<= ?v_3 7) (>= ?v_3 0) (<= ?v_4 7) (>= ?v_4 0) (<= ?v_5 7) (>= ?v_5 0) (<= ?v_6 7)
(>= ?v_6 0) (<= ?v_7 7) (>= ?v_7 0) (not (= x0 x1)) (not (= x0 x2)) (not (= x0 x3)) (not (= x0 x4))
(not (= x0 x5)) (not (= x0 x6)) (not (= x0 x7)) (not (= x1 x2)) (not (= x1 x3)) (not (= x1 x4)) (not
(= x1 x5)) (not (= x1 x6)) (not (= x1 x7)) (not (= x2 x3)) (not (= x2 x4)) (not (= x2 x5)) (not (= x2
x6)) (not (= x2 x7)) (not (= x3 x4)) (not (= x3 x5)) (not (= x3 x6)) (not (= x3 x7)) (not (= x4 x5))
(not (= x4 x6)) (not (= x4 x7)) (not (= x5 x6)) (not (= x5 x7)) (not (= x6 x7)) (not (= ?v_8 1)) (not
(= ?v_8 (- 1))) (not (= ?v_9 2)) (not (= ?v_9 (- 2))) (not (= ?v_10 3)) (not (= ?v_10 (- 3))) (not (= ?
v_11 4)) (not (= ?v_11 (- 4))) (not (= ?v_12 5)) (not (= ?v_12 (- 5))) (not (= ?v_13 6)) (not (= ?v_13
(- 6))) (not (= ?v_14 7)) (not (= ?v_14 (- 7))) (not (= ?v_15 1)) (not (= ?v_15 (- 1))) (not (= ?v_16
2)) (not (= ?v_16 (- 2))) (not (= ?v_17 3)) (not (= ?v_17 (- 3))) (not (= ?v_18 4)) (not (= ?v_18 (-
4))) (not (= ?v_19 5)) (not (= ?v_19 (- 5))) (not (= ?v_20 6)) (not (= ?v_20 (- 6))) (not (= ?v_21 1))
(not (= ?v_21 (- 1))) (not (= ?v_22 2)) (not (= ?v_22 (- 2))) (not (= ?v_23 3)) (not (= ?v_23 (- 3)))
(not (= ?v_24 4)) (not (= ?v_24 (- 4))) (not (= ?v_25 5)) (not (= ?v_25 (- 5))) (not (= ?v_26 1)) (not
(= ?v_26 (- 1))) (not (= ?v_27 2)) (not (= ?v_27 (- 2))) (not (= ?v_28 3)) (not (= ?v_28 (- 3))) (not
(= ?v_29 4)) (not (= ?v_29 (- 4))) (not (= ?v_30 1)) (not (= ?v_30 (- 1))) (not (= ?v_31 2)) (not (= ?
v_31 (- 2))) (not (= ?v_32 3)) (not (= ?v_32 (- 3))) (not (= ?v_33 1)) (not (= ?v_33 (- 1))) (not (= ?
v_34 2)) (not (= ?v_34 (- 2))) (not (= ?v_35 1)) (not (= ?v_35 (- 1))))))
(check-sat)
(exit)
```

The Constraint Satisfaction Problem (CSP)
is the problem of deciding whether there is a variable
assignment that satisfies a given set of constraints.

The 8-queens problem : CSP model

```
ESSENCE′ 1.0

given n : 8
letting queens_n be domain int(0..n − 1)

find queens : matrix indexed by [ queens_n ] of queens_n

such that

 alldifferent(queens),
 forall i, j : queens_n .
  (i > j) =>    ((queens[i] - i ! = queens[j] - j)
               /\ (queens[i] + i ! = queens[j] + j))
```

SAT, SMT or CSP?

- SAT:
  + extremely efficient
  - problem with expressivity
- SMT:
  + better expressivity, incorporates domain-specific reasoning
  - some loss of efficiency
- CSP:
  + very expressive, uses domain-specific reasoning
  - some loss of efficiency

SAT, SMT or CSP?

- SAT:
  + extremely efficient
  - problem with expressivity
- SMT:
  + better expressivity, incorporates domain-specific reasoning
  - some loss of efficiency
- CSP:
  + very expressive, uses domain-specific reasoning
  - some loss of efficiency

highly problem-dependent though..

A short introduction to SAT solving

SAT solver classification

- complete SAT solvers
  : based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm
- incomplete SAT solvers
  : based on local search
- hybrid SAT solvers

SAT solver classification

Most widely used SAT solvers:

- Conflict-Driven Clause Learning (CDCL) SAT solvers

Why use SAT solvers ?

SAT solvers are extremely efficient

Year	Variables	Clauses
60s - 70s	tens	hundreds

SAT solvers are extremely efficient

Year	Variables	Clauses
60s - 70s	tens	hundreds
80s - early 90s	hundreds	thousands

SAT solvers are extremely efficient

Year	Variables	Clauses
60s - 70s	tens	hundreds
80s - early 90s	hundreds	thousands
late 90s	thousands	hundreds of thousands

SAT solvers are extremely efficient

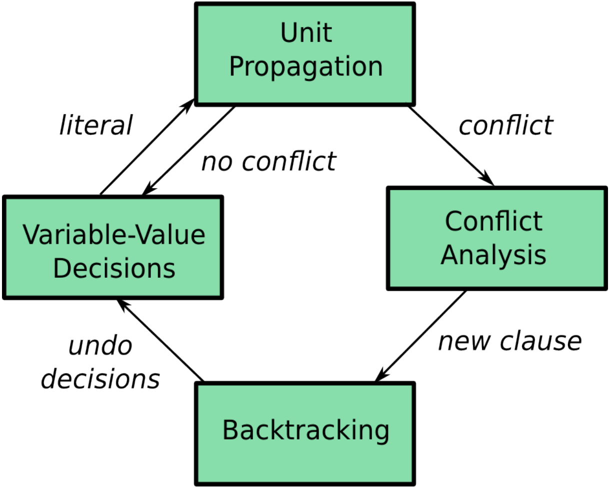
Year	Variables	Clauses
60s - 70s	tens	hundreds
80s - early 90s	hundreds	thousands
late 90s	thousands	hundreds of thousands
00s - now	hundreds of thousands and more	millions

SAT Applications

- Bounded Model Checking
- Planning
- Software Verification
- Automatic Test Pattern Generation
- Combinational Equivalence Checking
- Combinatorial Interaction Testing
- and many others..

How do CDCL SAT solvers work?

CDCL SAT solver



Unit Propagation

Variable-Value Decisions

Conflict Analysis

Backtracking

literal

no conflict

conflict

new clause

undo decisions

Unit propagation

$$x_0 \lor x_2 \lor x_3$$
$$\neg x_2 \lor \neg x_3$$

decision: $x_3 = 0$
decision level: 1

Unit propagation

$$x_0 \lor x_2 \lor x_3$$
$$\neg x_2 \lor \neg x_3$$

decision: $x_3 = 0$

decision level: 1

Unit propagation

$$x_0 \lor x_2$$
$$\neg x_2 \lor \neg x_3 \ : \ \text{satisfied}$$

decision: $x_3 = 0$
decision level: 1

Unit propagation

$$x_0 \vee x_2$$
$$\neg x_2 \vee \neg x_3 : \text{ satisfied}$$

decision: $x_3 = 0$ (d.l. 1), $x_2 = 0$
decision level: 2

Unit propagation

$$x_0 \vee x_2$$

$$\neg x_2 \vee \neg x_3 : \text{ satisfied}$$

decision: $x_3 = 0$ (d.l. 1), $x_2 = 0$

decision level: 2

Unit propagation

x_0

$\neg x_2 \vee \neg x_3$: satisfied

decision: $x_3 = 0$ (d.l. 1), $x_2 = 0$

decision level: 2

Unit propagation

x_0

$\neg x_2 \vee \neg x_3$: satisfied

decision: $x_3 = 0$ (d.l. 1), $x_2 = 0$ (d.l. 2), $x_0 = 1$ (cl. 1)

decision level: 2

Unit propagation

$$x_0 : \text{satisfied}$$
$$\neg x_2 \vee \neg x_3 : \text{satisfied}$$

decision: $x_3 = 0$ (d.l. 1), $x_2 = 0$ (d.l. 2), $x_0 = 1$ (cl. 1)
decision level: 2

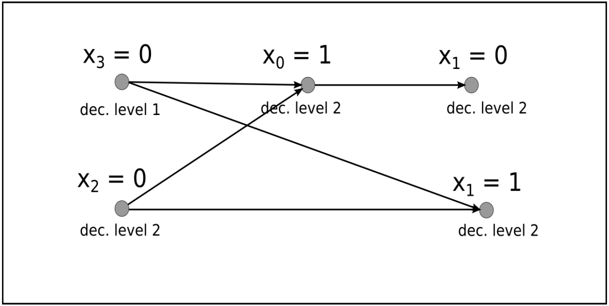
SAT solver (60s)

```
$F = \mathrm{BCP}(F)$ : unit propagation (Boolean constraint propagation)
if $F = \mathrm{True}$ : return satisfiable
if empty clause $\in F$ : return unsatisfiable
pick remaining variable $x$ and literal $l \in \{x, \neg x\}$
if $\mathrm{DPLL}(F \wedge \{l\})$ returns satisfiable : return satisfiable
return $\mathrm{DPLL}(F \wedge \{\neg l\})$
```

Conflict Analysis (late 90s)

MiniSAT demo
http : //minisat.se/Papers.html

Conflict Analysis - implication graph



Conflict Analysis - conflict clause

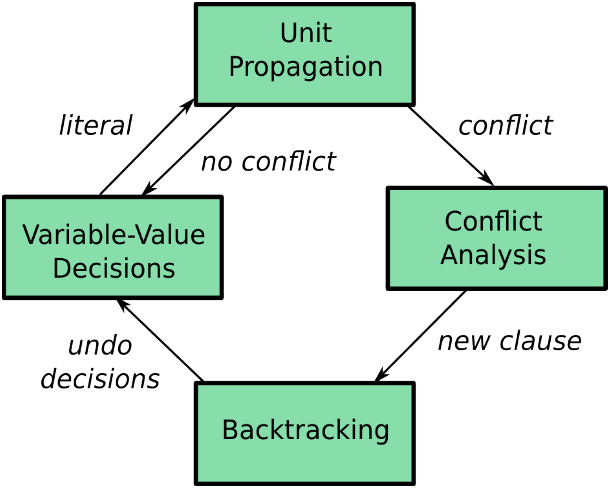
Candidate conflict clauses:
$\neg(x_3 = 0 \wedge x_2 = 0 \wedge x_0 = 1) \leftrightarrow x_3 \vee x_2 \vee \neg x_0$
or
$\neg(x_3 = 0 \wedge x_2 = 0) \leftrightarrow x_3 \vee x_2$
or
$\neg(x_2 = 0 \wedge x_0 = 1) \leftrightarrow x_2 \vee \neg x_0$

First Unit Implication Point (First UIP) scheme adds a conflict clause that contains only one variable that is assigned at the current decision level (a so-called asserting clause).

Conclusions

SAT, SMT and CSP solvers are used for solving problems involving constraints.

CDCL SAT solver



SAT solvers are extremely efficient

Year	Variables	Clauses
60s - 70s	few tens	hundreds
80s - early 90s	few hundreds	few thousands
late 90s	(tens of) thousands	hundreds of thousands
00s - now	hundreds of thousands and more	millions

SAT Applications

- Bounded Model Checking
- Planning
- Software Verification
- Automatic Test Pattern Generation
- Combinational Equivalence Checking
- Combinatorial Interaction Testing
- and many others..

References

- Armin Biere, ed. *Handbook of satisfiability.* Vol. 185. IOS PressInc, 2009.
- Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, Cesare Tinelli. *Satisfiability Modulo Theories. Handbook of Satisfiability* 2009: 825-885
- Francesca Rossi, Peter Van Beek, and Toby Walsh, eds. *Handbook of constraint programming.* Vol. 2. Elsevier Science, 2006.
- MiniSAT demo ("Practical SAT - a tutorial on applied satisfiability solving") http : //minisat.se/Papers.html

Example solvers

- SAT: MiniSAT, Glucose, CryptoMiniSAT, SAT4J
- SMT: Z3, Yices, CVC4
- CSP: Minion, Gecode, G12, ILOG, JaCoP
- SAT- and SMT-based constraint solvers: Sugar, Fzn2smt
- hybrid solvers: Chuffed
- and many others (see SAT/SMT/CSP solver competitions and MiniZinc challenge)