SAT and SMT Solvers in Practice

Marijn J.H. Heule and Ruben Martins

Carnegie Mellon University

`http://www.cs.cmu.edu/~mheule/15816-f19/`

Automated Reasoning and Satisfiability, September 12, 2019

DIMACS: SAT solver input format

The DIMACS format for SAT solvers has three types of lines:

- header: `p cnf n m` in which `n` denotes the highest variables index and `m` the number of clauses
- clauses: a sequence of integers ending with `0`
- comments: any line starting with `c`

$$
\begin{aligned}
&(a \vee b \vee \overline{c}) \wedge \\
&(\overline{a} \vee \overline{b} \vee c) \wedge \\
&(b \vee c \vee \overline{d}) \wedge \\
&(\overline{b} \vee \overline{c} \vee d) \wedge \\
&(a \vee c \vee d) \wedge \\
&(\overline{a} \vee \overline{c} \vee \overline{d}) \wedge \\
&(\overline{a} \vee b \vee d)
\end{aligned}
$$

```
c example
p cnf 4 7
 1  2 -3 0
-1 -2  3 0
 2  3 -4 0
-2 -3  4 0
 1  3  4 0
-1 -3 -4 0
-1  2  4 0
```

DIMACS: SAT solver output format

The solution line of a SAT solver starts with "`s `":

- `s SATISFIABLE`: The formula is satisfiable
- `s UNSATISFIABLE`: The formula is unsatisfiable
- `s UNKNOWN`: The solver cannot determine satisfiability

In case the formula is satisfiable, the solver emits a certificate:

- lines starting with "`v `"
- a list of integers ending with `0`
- e.g. `v -1 2 4 0`

In case the formula is unsatisfiable, then most solvers support emitting a proof of unsatisfiability to a separate file

CaDiCaL: download and install

Most SAT solvers are implemented in C/C++

CaDiCaL is one of the strongest SAT solvers. As the name suggests it is based on CDCL. Recommended for Linux and macOS users.

obtain CaDiCaL:

- `git clone https://github.com/arminbiere/cadical.git`
- cd cadical
- ./configure; make

to run: `./build/cadical` formula.cnf

SAT4J: download and install

SAT4J is a SAT solver in Java. It is also based on CDCL. Recommended for windows users.

obtain SAT4J:

- `git clone https://github.com/marijnheule/sat-examples.git`
- cd sat-examples

to run: `java -jar org.sat4j.core-2.3.1.jar` formula.cnf

UBCSAT

UBCSAT is a local search SAT solver.

obtain UBCSAT:

- download and unzip http://ubcsat.dtompkins.com/downloads/ubcsat-beta-12-b18.tar.gz
- cd ubcsat-beta-12-b18
- make clean; make

to run: `./ubcsat -alg ddfw -i` formula.cnf

there are many LS algorithms to choose from (`-alg`)

Many SAT solvers

Many SAT solvers have been developed

Lots of them participate in the annual SAT competition

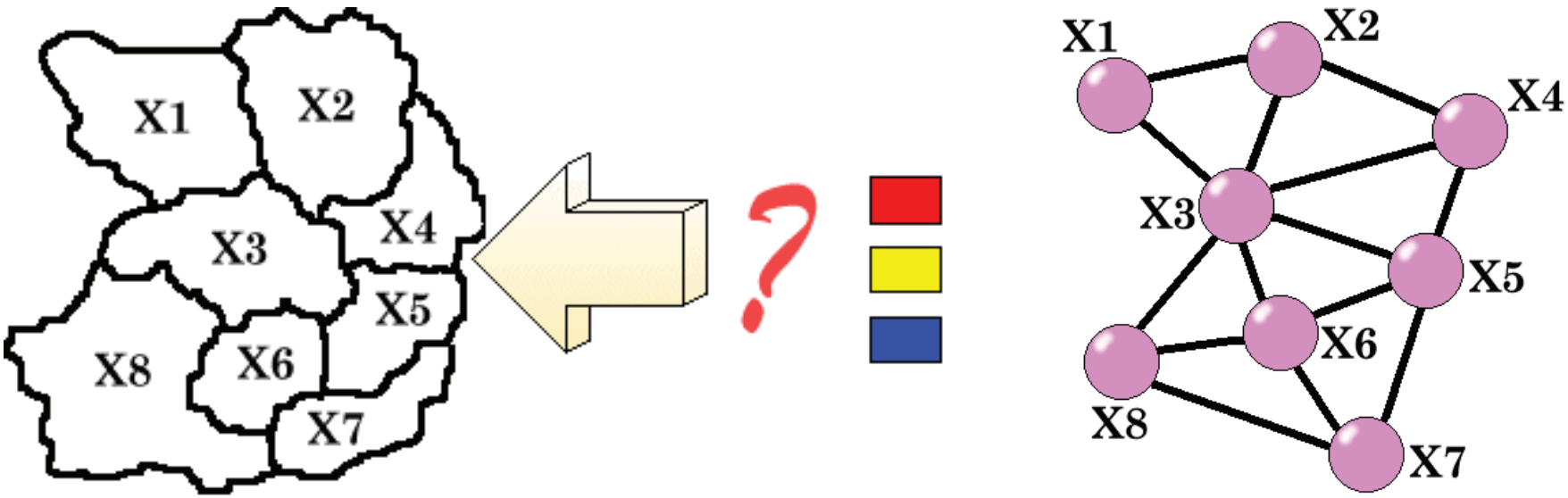
- All code of participants in open source
- Each solver is run on hundreds of benchmarks
- Large timeout 5000 seconds

For details and downloading more solvers visit
http://satcompetition.org/

Demo: SAT Solving

Graph coloring

Given a graph $G(V, E)$, can the vertices be colored with k colors such that for each edge $(v, w) \in E$, the vertices v and w are colored differently.



Graph coloring encoding

Variables	Range	Meaning
$x_{v,i}$	$i \in \{1, \ldots, c\}$ $v \in \{1, \ldots, \lvert V \rvert\}$	node v has color i

Clauses	Range	Meaning
$(x_{v,1} \vee x_{v,2} \vee \cdots \vee x_{v,c})$	$v \in \{1, \ldots, \lvert V \rvert\}$	v is colored
$(\overline{x}_{v,s} \vee \overline{x}_{v,t})$	$s \in \{1, \ldots, c-1\}$ $t \in \{s+1, \ldots, c\}$	v has at most one color
$(\overline{x}_{v,i} \vee \overline{x}_{w,i})$	$(v, w) \in E$	v and w have a different color

Graph coloring encoding code

```c
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char** argv) {
  FILE* graph  = fopen (argv[1], "r");
  int i, j, a, b, nVertex, nEdge, nColor = atoi  (argv[2]);
  fscanf (graph, " p edge %i %i ", &nVertex, &nEdge);

  printf ("p cnf %i %i\n", nVertex * nColor, nVertex + nEdge * nColor);

  for (i = 0; i < nVertex; i++) {
    for (j = 1; j <= nColor; j++)
      printf ("%i ", i * nColor + j);
    printf ("0\n"); }

  while (1) {
    int tmp = fscanf (graph, " e %i %i ", &a, &b);
    if (tmp == 0 || tmp == EOF) break;
    for (j = 1; j <= nColor; j++)
      printf ("-%i -%i 0\n", (a-1) * nColor + j, (b-1) * nColor + j);
  }
}
```

Demo: Encode, Decode

Unsatisfiable cores

An unsatisfiable core of an unsatisfiable formula F is a subset of F that is unsatisfiable.

An minimal unsatisfiable core of an unsatisfiable formula such that the removal of any clause makes the formula satisfiable.

Extracting a minimal unsatisfiable core from a formula has many applications, but the computational costs could be high.

- maxSAT
- diagnosis
- formal verification

Proofs

A proof of unsatisfiability is a certificate that a given formula is unsatisfiable.

Various proof producing methods exists (another lecture).

Proof checking tools cannot only validate a proof but also produce additional information about the formula:

- unsatisfiable core
- optimized proof

DRAT-trim is a tool that validates proofs and produces such information

Demo: Core Extraction

SMT-LIB: SMT solver input format

http://smtlib.cs.uiowa.edu/

Language has similarities with functional languages and it is more readable than CNF. Theories:

- Arrays,
- Bitvectors,
- Boolean predicates,
- Floating point,
- Ints,
- Reals

SMT-LIB: SMT solver input format

http://smtlib.cs.uiowa.edu/

```
; Basic Boolean example
(set-option :print-success false)
(set-logic QF_UF)
(declare-const p Bool)
(assert (and p (not p)))
(check-sat) ; returns 'unsat'
(exit)
```

SMT-LIB: SMT solver input format

http://smtlib.cs.uiowa.edu/

```
; Integer arithmetic
(set-logic QF_LIA)
(declare-const x Int)
(declare-const y Int)
(assert (= (- x y) (+ x (- y) 1)))
(check-sat)
; unsat
(exit)
```

SMT Solvers

- Z3 (Microsoft): https://github.com/Z3Prover/z3/wiki
- CVC4 (Stanford): http://cvc4.cs.stanford.edu/web/
- Yices (SRI): http://yices.csl.sri.com/
- Boolector (JKU Austria): https://boolector.github.io/

SMT Solvers

We recommend the use of Z3:

- Tutorials:
  `https://rise4fun.com/z3/tutorial`
  `https://theory.stanford.edu/~nikolaj/programmingz3.html`
- APIs for Python, C++, Java
- MIT License: `https://github.com/Z3Prover/z3`
- Most used and cited SMT solver (>5,000 citations)

Demo: SMT solving

https://rise4fun.com/z3/tutorial

Proving program equivalence in SMT

```
1 int power3(int in)
2 {
3     int i, out_a;
4     out_a = in;
5     for (i = 0; i < 2; i++)
6       out_a = out_a * in;
7   return out_a;
8 }
```

```
1 int power3_new(int in)
2 {
3     int out_b;
4
5     out_b = (in * in) * in;
6
7     return out_b;
8 }
```

$$\varphi_a \equiv (out0\_a = in0\_a) \wedge (out1\_a = out0\_a \times in0\_a) \wedge (out2\_a = out1\_a \times in0\_a)$$

$$\varphi_b \equiv out0\_b = (in0\_b \times in0\_b) \times in0\_b$$

To show these programs are equivalent, we must show the following formula is valid: $in0\_a = in0\_b \wedge \varphi_a \wedge \varphi_b \implies out2\_a = out0\_b$

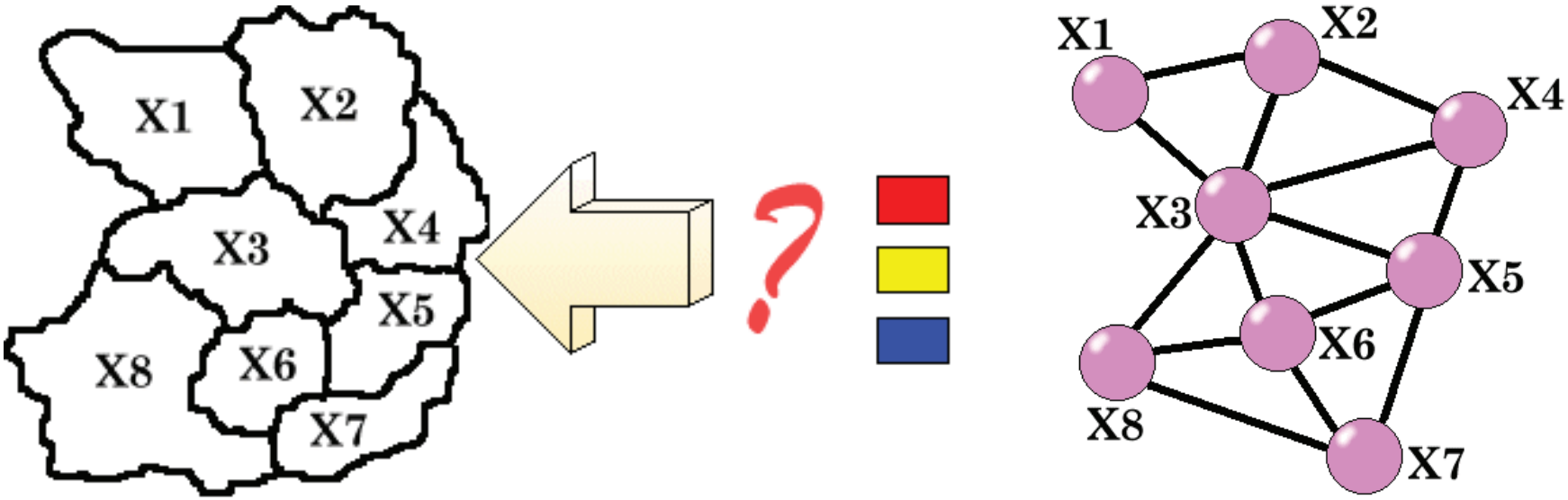
Demo: Program equivalence with SMT solving

Integers as mathematical integers:
`https://rise4fun.com/Z3/BLQpl`

Integers as bit vectors:
`https://rise4fun.com/Z3/ibsw3`

`https://rise4fun.com/Z3/V7Sf`
Using uninterpreted functions:

Graph coloring encoding in SMT



Variables:

- Integer variables x_i for each node

Constraints:

- $1 \leq x_i \leq c$
- $x_i \neq x_j$ for $(x_i, x_j) \in E$

Demo: Encoding in SMT

Unsatisfiable cores in SMT

https://rise4fun.com/Z3/VHDA

```
; Getting unsatisfiable cores
(set-option :produce-unsat-cores true)
(set-logic QF_UF)
(declare-const p Bool) (declare-const q Bool) (declare-const r Bool)
(declare-const s Bool) (declare-const t Bool)
(assert (! (=> p q) :named PQ))
(assert (! (=> q r) :named QR))
(assert (! (=> r s) :named RS))
(assert (! (=> s t) :named ST))
(assert (! (not (=> q s)) :named NQS))
(check-sat)
; unsat
(get-unsat-core)
; (QR RS NQS)
(exit)
```

SAT and SMT Solvers in Practice

Marijn J.H. Heule and Ruben Martins

Carnegie Mellon University

`http://www.cs.cmu.edu/~mheule/15816-f19/`

Automated Reasoning and Satisfiability, September 12, 2019