Solving Constraint Satisfaction Problems by a SAT Solver

Naoyuki Tamura, Tomoya Tanjo, and Mutsunori Banbara

Kobe University, JAPAN

Osaka Workshop for Verification and Validation
Feb 28, 2011 at AIST Amagasaki
Contents

1. SAT problems and SAT solvers
2. SAT encodings of Constraint Satisfaction Problems (CSP)
   - Order Encoding
3. SAT-based Systems using Order Encoding
   - SAT-based Constraint Solver Sugar
     Results of the 2009 International CSP Solver Competition
   - Shop Scheduling
   - Two-Dimentional Strip Packing
   - Combinatorial Test Case Generation
   - Sokoban Puzzle
4. Summary
   - CSPSAT Project

Naoyuki Tamura, Tomoya Tanjo, and Mutsunori Banbara
Solving Constraint Satisfaction Problems by a SAT Solver
SAT problems and SAT solvers
SAT (Boolean satisfiability testing) is a problem to decide whether a given Boolean formula has any satisfying truth assignment.

- SAT is a central problem in Computer Science both theoretically and practically.
- SAT was the first NP-complete problem [Cook 1971].
- SAT has very efficient implementation (MiniSat, etc.)
- SAT-based approach is becoming popular in many areas.
SAT instances

SAT instances are given in the conjunctive normal form (CNF).

**CNF formula**

- A **CNF formula** is a conjunction of clauses.
- A **clause** is a disjunction of literals.
- A **literal** is either a Boolean variable or its negation.

DIMACS CNF is used as the standard format for CNF files.

```
p cnf 9 7 ; Number of variables and clauses
1 2 0 ; a ∨ b
9 3 0 ; c ∨ d
1 8 4 0 ; a ∨ e ∨ f
-2 -4 5 0 ; ¬b ∨ ¬f ∨ g
-4 6 0 ; ¬f ∨ h
-2 -6 7 0 ; ¬b ∨ ¬h ∨ i
-5 -7 0 ; ¬g ∨ ¬i
```
**SAT Solver**

**SAT solver** is a program to decide whether a given SAT instance is satisfiable (SAT) or unsatisfiable (UNSAT). Usually, it also returns a truth assignment as a solution when the instance is SAT.

- Systematic (complete) SAT solver answers SAT or UNSAT.
  - Most of them are based on the DPLL algorithm.
- Stochastic (incomplete) SAT solver only answers SAT (no answers for UNSAT).
  - Local search algorithms are used.
Modern SAT solvers

- The following techniques have been introduced to DPLL and they drastically improved the performance of modern SAT solvers.
  - **CDCL** (Conflict Driven Clause Learning) [Silva 1996]
  - Non-chronological Backtracking [Silva 1996]
  - Random Restarts [Gomes 1998]
  - Watched Literals [Moskewicz & Zhang 2001]
  - Variable Selection Heuristics [Moskewicz & Zhang 2001]

- **Chaff** and **zChaff** solvers made one to two orders magnitude improvement [2001].

- SAT competitions and SAT races since 2002 contribute to the progress of SAT solver implementation techniques.

- **MiniSat** solver showed its good performance in the 2005 SAT competition with about 2000 lines of code in C++.

- Modern SAT solvers can handle instances with more than $10^6$ variables and $10^7$ clauses.
SAT-based Approach

SAT-based approach is becoming popular for solving hard combinatorial problems.

- Planning (SATPLAN, Blackbox) [Kautz & Selman 1992]
- Automatic Test Pattern Generation [Larrabee 1992]
- Job-shop Scheduling [Crawford & Baker 1994]
- Software Specification (Alloy) [1998]
- Bounded Model Checking [Biere 1999]
- Software Package Dependency Analysis (SATURN)
  - SAT4J is used in Eclipse 3.4.
- Rewriting Systems (AProVE, Jambox)
- Answer Set Programming (clasp, Cmodels-2)
- FOL Theorem Prover (iProver, Darwin)
- First Order Model Finder (Paradox)
- Constraint Satisfaction Problems (Sugar) [Tamura et al. 2006]
Why SAT-based? (personal opinions)

SAT solvers are very fast.

- Clever implementation techniques, such as two literal watching.
  - It minimizes house-keeping informations for backtracking.
- Cache-aware implementation [Zhang & Malik 2003]
  - For example, a SAT-encoded Open-shop Scheduling problem instance gp10-10 is solved within 4 seconds with more than 99% cache hit rate by MiniSat.

```
$ valgrind --tool=cachegrind minisat gp10-10-1091.cnf
L2 refs: 42,842,531 ( 31,633,380 rd +11,209,151 wr)
L2 misses: 25,674,308 ( 19,729,255 rd + 5,945,053 wr)
L2 miss rate: 0.4% ( 0.4% + 1.0% )
```
Why SAT-based? (personal opinions)

SAT-based approach is similar to RISC approach in ’80s by Patterson.

- **RISC**: Reduced Instruction Set Computer

  Patterson claimed a computer of a “reduced” and fast instruction set with an efficient optimizing compiler can be faster than a “complex” computer (**CISC**).

SAT Solver \(\iff\) RISC

SAT Encoder \(\iff\) Optimizing Compiler

- In that sense, study of both SAT solvers and SAT encodings are important and interesting topics.
SAT encodings of Constraint Satisfaction Problems
Finite linear CSP

- **Variables**
  - **Integer variables** with finite domains
    - \( \ell(x) \): the lower bound of \( x \)
    - \( u(x) \): the upper bound of \( x \)
  - **Boolean variables**

- **Constraints**
  - **Arithmetic operators**: +, −, constant multiplication, etc.
  - **Comparison operators**: =, ≠, ≥, >, ≤, <
  - **Logical operators**: ¬, ∧, ∨, ⇒

- A CSP is called **satisfiable** when there exists an assignment which satisfies all given constraints. Otherwise, it is called **unsatisfiable**.
SAT encodings

There have been several methods proposed to encode CSP into SAT.

- **Direct encoding** is the most widely used one [de Kleer 1989].
- **Order encoding** is a new encoding showing a good performance for a wide variety of problems [Tamura et al. 2006].
  - It is first used to encode job-shop scheduling problems by [Crawford & Baker 1994].
  - It succeeded to solve previously undecided problems in open-shop scheduling, job-shop scheduling, two-dimensional strip packing, etc.
- Other encodings:
  - **Multivalued encoding** [Selman 1992]
  - **Support encoding** [Kasif 1990]
  - **Log encoding** [Iwama 1994]
  - **Log-support encoding** [Gavanelli 2007]
Order encoding

In order encoding [Tamura et al. 2006], a Boolean variable $p(x \leq i)$ is defined as true iff the integer variable $x$ is less than or equal to the domain value $i$, that is, $x \leq i$.

**Boolean variables for each integer variable $x$**

\[
p(x \leq i) \quad (\ell(x) \leq i < u(x))
\]

For example, the following four Boolean variables are used to encode an integer variable $x \in \{2, 3, 4, 5, 6\}$,

**4 Boolean variables for $x \in \{2, 3, 4, 5, 6\}$**

\[
p(x \leq 2) \quad p(x \leq 3) \quad p(x \leq 4) \quad p(x \leq 5)
\]

Boolean variable $p(x \leq 6)$ is unnecessary since $x \leq 6$ is always true.
The following clauses are required to make $p(x \leq i)$ be true iff $x \leq i$.

**Clauses for each integer variable $x$**

\[-p(x \leq i - 1) \lor p(x \leq i) \quad (\ell(x) < i < u(x))\]

For example, 3 clauses are required for $x \in \{2, 3, 4, 5, 6\}$.

**3 clauses for $x \in \{2, 3, 4, 5, 6\}$**

\[-p(x \leq 2) \lor p(x \leq 3)\]
\[-p(x \leq 3) \lor p(x \leq 4)\]
\[-p(x \leq 4) \lor p(x \leq 5)\]
The following table shows possible satisfiable assignments for the given clauses.

\[ \neg p(x \leq 2) \lor p(x \leq 3) \]
\[ \neg p(x \leq 3) \lor p(x \leq 4) \]
\[ \neg p(x \leq 4) \lor p(x \leq 5) \]

<table>
<thead>
<tr>
<th>Interpretation</th>
<th>( p(x \leq 2) )</th>
<th>( p(x \leq 3) )</th>
<th>( p(x \leq 4) )</th>
<th>( p(x \leq 5) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = 2 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( x = 3 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( x = 4 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( x = 5 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( x = 6 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
A constraint is encoded by enumerating its conflict regions instead of conflict points.

**Constraint clauses**

When all points \((x_1, \ldots, x_k)\) in the region \(i_1 < x_1 \leq j_1, \ldots, i_k < x_k \leq j_k\) violate the constraint, the following clause is added.

\[
p(x_1 \leq i_1) \lor \neg p(x_1 \leq j_1) \lor \cdots \lor p(x_k \leq i_k) \lor \neg p(x_k \leq j_k)
\]
Encoding a constraint $x + y \leq 7$
Encoding a constraint \( x + y \leq 7 \)

\( \neg(y \geq 6) \)
Encoding a constraint $x + y \leq 7$

$p(y \leq 5)$
Encoding a constraint $x + y \leq 7$

$p(y \leq 5) 
\neg(x \geq 3 \land y \geq 5)$
Encoding a constraint $x + y \leq 7$

$p(y \leq 5)$
$p(x \leq 2) \lor p(y \leq 4)$
Encoding a constraint $x + y \leq 7$

- $p(y \leq 5)$
- $p(x \leq 2) \lor p(y \leq 4)$
- $\neg (x \geq 4 \land y \geq 4)$
Encoding a constraint $x + y \leq 7$

$p(y \leq 5)$
$p(x \leq 2) \lor p(y \leq 4)$
$p(x \leq 3) \lor p(y \leq 3)$
Order encoding (cont.)

Encoding a constraint $x + y \leq 7$

\begin{align*}
p(y \leq 5) \\
p(x \leq 2) \lor p(y \leq 4) \\
p(x \leq 3) \lor p(y \leq 3) \\
\neg(x \geq 5 \land y \geq 3)
\end{align*}
Order encoding (cont.)

Encoding a constraint \( x + y \leq 7 \)

\[
\begin{align*}
  p(y \leq 5) \\
  p(x \leq 2) \lor p(y \leq 4) \\
  p(x \leq 3) \lor p(y \leq 3) \\
  p(x \leq 4) \lor p(y \leq 2)
\end{align*}
\]
Encoding a constraint $x + y \leq 7$

\[ p(y \leq 5) \]
\[ p(x \leq 2) \lor p(y \leq 4) \]
\[ p(x \leq 3) \lor p(y \leq 3) \]
\[ p(x \leq 4) \lor p(y \leq 2) \]
\[ \neg(x \geq 6) \]
Encoding a constraint $x + y \leq 7$

$$
p(y \leq 5)
$$

$$
p(x \leq 2) \lor p(y \leq 4)
$$

$$
p(x \leq 3) \lor p(y \leq 3)
$$

$$
p(x \leq 4) \lor p(y \leq 2)
$$

$$
p(x \leq 5)
$$
Bound propagation in order encoding

**Encoding a constraint** $x + y \leq 7$

- $C_1 : p(y \leq 5)$
- $C_2 : p(x \leq 2) \lor p(y \leq 4)$
- $C_3 : p(x \leq 3) \lor p(y \leq 3)$
- $C_4 : p(x \leq 4) \lor p(y \leq 2)$
- $C_5 : p(x \leq 5)$

- When $p(x \leq 3)$ becomes false (i.e. $x \geq 4$), $p(y \leq 3)$ becomes true (i.e. $y \leq 3$) by **unit propagation** on $C_3$.
- This corresponds to the **bound propagation** in CSP solvers.
SAT-based Systems using Order Encoding
Sugar: a SAT-based Constraint Solver

Sugar is a constraint solver based on the order encoding.

In the 2008 CSP solver competition, Sugar became the winner in GLOBAL category.

In the 2008 Max-CSP solver competition, Sugar became the winner in three categories of INTENSIONAL and GLOBAL constraints.

In the 2009 CSP solver competition (CSC’09), Sugar became the winner in three categories of GLOBAL constraints.
### Results of GLOBAL in CSC’09 (556 instances)

#### GLOBAL in CSC’09: Comparison of Sugar with top solvers

<table>
<thead>
<tr>
<th>Series</th>
<th>Sugar+m</th>
<th>Sugar+p</th>
<th>Mistral</th>
<th>Choco</th>
<th>bpsolver</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIBD</td>
<td>(83)</td>
<td>76</td>
<td>77</td>
<td>76</td>
<td>58</td>
</tr>
<tr>
<td>Costas Array</td>
<td>(11)</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Latin Square</td>
<td>(10)</td>
<td>10</td>
<td>9</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Magic Square</td>
<td>(18)</td>
<td>8</td>
<td>8</td>
<td>13</td>
<td>15</td>
</tr>
<tr>
<td>NengFa</td>
<td>(3)</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Orthogonal Latin Square</td>
<td>(9)</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Perfect Square Packing</td>
<td>(74)</td>
<td>54</td>
<td>53</td>
<td>40</td>
<td>47</td>
</tr>
<tr>
<td>Pigeons</td>
<td>(19)</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>Quasigroup Existence</td>
<td>(35)</td>
<td>30</td>
<td>29</td>
<td>29</td>
<td>28</td>
</tr>
<tr>
<td>Pseudo-Boolean</td>
<td>(100)</td>
<td>68</td>
<td>75</td>
<td>59</td>
<td>53</td>
</tr>
<tr>
<td>BQWH</td>
<td>(20)</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Cumulative Job-Shop</td>
<td>(10)</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>RCPSP</td>
<td>(78)</td>
<td>78</td>
<td>78</td>
<td>78</td>
<td>77</td>
</tr>
<tr>
<td>Cabinet</td>
<td>(40)</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>Timetabling</td>
<td>(46)</td>
<td>25</td>
<td>42</td>
<td>39</td>
<td>14</td>
</tr>
<tr>
<td>Total</td>
<td>(556)</td>
<td>446</td>
<td>468</td>
<td>435</td>
<td>391</td>
</tr>
</tbody>
</table>

- **Sugar+m**: Sugar with MiniSat backend
- **Sugar+p**: Sugar with PicoSAT backend

Naoyuki Tamura, Tomoya Tanjo, and Mutsunori Banbara

Solving Constraint Satisfaction Problems by a SAT Solver
GLOBAL: Time vs Number of solved instances

Naoyuki Tamura, Tomoya Tanjo, and Mutsunori Banbara
Solving Constraint Satisfaction Problems by a SAT Solver
Optimal values for 3 instances out of 192 benchmark instances were unknown until 2005 [Blum 2005, Laborie 2005].

Remaining instances were closed by Sugar [Tamura et al., CP-2006]

<table>
<thead>
<tr>
<th>OSS Instance</th>
<th>Opt.</th>
<th>Previously known bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Lower bound</td>
</tr>
<tr>
<td>j7-per0-0</td>
<td>1048</td>
<td>1039</td>
</tr>
<tr>
<td>j8-per0-1</td>
<td>1039</td>
<td>1018</td>
</tr>
<tr>
<td>j8-per10-2</td>
<td>1002</td>
<td>1000</td>
</tr>
</tbody>
</table>
SATSHOP (Nabeshima, Koshimura et al.)
SAT-based Job-Shop Scheduling Problem Solver

Optimum Scheduling of abz9 found by SATSHOP

- “Lemma Reusing for SAT based Planning and Scheduling” [Nabeshima et al., ICAPS2006]
- Closed two instances (abz9 [Adams et al., 1988], yn1) [Koshimura et al., IEICE Trans. 2010]
2D Strip Regular Packing (Soh, Inoue et al.)

- Find a minimum length to pack all given rectangles in a fixed width strip.
- Closed 29 instances [Soh et al., RCRA2008]
New and Better Packing of Fu instance

- Find a minimum length to pack all given polygons in a fixed width strip.
- New solution of Fu instance with the length of 31 was found by Sugar three days ago!
Generating test cases for combinatorial testing is equivalent to finding a covering array (CA).

Combination of the order encoding and Hnich’s encoding is used.

Lower-bounds are updated for six instances and the optimum size are decided for two instances [Banbara et al., LPAR2010].
Solving Sokoban Puzzle by Sugar (Tamura)

Sugar can find a solution as a satisfiable \( \Phi(s) \) with minimal \( s \).

\[
\Phi(s) = I(S_0) \land \bigvee_{i=0}^{s} G(S_t) \land \bigwedge_{t=0}^{s-1} T(S_t, S_{t+1})
\]

State of \( t \)-th step is represented by a set of variables \( S_t \). Initial and Goal conditions are represented by formulas \( I(S_0) \) and \( G(S_t) \)'s respectively. Each transition is represented by \( T(S_t, S_{t+1}) \).
We presented Order encoding and SAT-based systems using the order encoding including Sugar constraint solver.

SAT solver can be used as an engine for solving hard combinatorial problems. However, modeling and encoding are also important research topic.

Sugar is developed as a software of the following project.

- http://bach.istc.kobe-u.ac.jp/sugar/
Objective and Research Topics

R&D of efficient and practical SAT-based CSP solvers

- SAT encodings
  - CSP, Dynamic CSP, Temporal Logic, Distributed CSP
- Parallel SAT solvers
  - Multi-core, PC Cluster

Teams and Professors

- Kobe University (3)
- National Institute of Informatics (1)
- University of Yamanashi (3)
- Kyushu University (4)
- Waseda University (1)
References

- SAT and SAT solvers
  - International Conference on Theory and Applications of Satisfiability Testing (SAT)
  - Journal on Satisfiability, Boolean Modeling and Computation

- Papers on Sugar