Practical Applications of SAT

Carsten Sinz

Institute for Formal Models and Verification
Johannes Kepler University Linz
Linz, Austria
Motivation

- \((x \cdot y = x + y + 674) \land (x - 6 = 4 \cdot (y - 6))\)
- Solution for \(x, y\) in \(\mathbb{Z}\)?
- SW-Verification: Solution in \(\mathbb{Z}\) mod \(2^{32}\)?
- Demo: c32sat
  - SAT-based solver / tautology checker for C-expressions
  - Just checked \(2^{3 \cdot 32} \approx 7.9 \cdot 10^{28}\) variable assignments using a state-of-the-art SAT-solver!
Part 1: Industrial Applications
Application 1: Product Configuration

- Configurable products, model lines
  - Products assembled out of standardized components
  - E.g. computers, cars, telecommunication equipment
- Dependencies between components
  - Specified using logical formalism („product overview“)
- Automatic (rule-based) order processing system
  - Checks customer’s order, transforms it into a parts list
- Computational problems:
  1. Determine valid (constructible) product instance satisfying
     - component dependencies
     - customer’s restrictions
  2. Check consistency of product overview
Case Study: Configuration of DC’s Mercedes Cars

Options available for Mercedes-Benz’s C class: (excerpt, total: 692)

- 231 garage door opener integrated into interior mirror
- 280 steering wheel in leather design (two-colored) with chrome clip
- 550 trailer appliance
- 581 comfort air-conditioning THERMOTRONIC
- 671 light metal wheels 4x, 7 spoke design

Restrictions for Mercedes-Benz’s C class: (excerpt, total: 952)

- AMG styling (772) cannot be combined with trailer appliance (550).
- Comfort air-conditioning (581) requires high-capacity battery (673), except when combined with gasoline engines with 2.6 or 3.2 liter cylinder capacity.
Order Processing Schema for Mercedes Cars

1. Order completion („supplementation“)
2. Consistency check
3. Generation of parts list
DaimlerChrysler: Batch Configuration Algorithm

\[
\text{do} \quad \text{Supplementation} \\
\quad \text{if } Z_1 \text{ then add code } c_1 \\
\quad \backslash \ldots \backslash \\
\quad \text{if } Z_n \text{ then add code } c_n \\
\text{until no further changes result} \\
\text{for } i=1 \text{ to } n \text{ do} \quad \text{Constructability check} \\
\quad \text{if } \neg (c_i \Rightarrow B_i) \text{ then “error”} \\
\text{for } j=1 \text{ to } k \text{ do} \quad \text{Parts list generation} \\
\quad \text{if } T_j \text{ then select part } p_j
\]
Batch Configuration Algorithm: Translation to SAT

- **Typical formula** $B_i$ in constructability check:

$$((-L(M112+M28+M001+M112+M28+M113)+\ldots$$
$$+(220/248/289/331/480/481/500/540/611/656/657+956/819/875+\ldots$$
$$+(460/M113)/882/W10/Y94/Y95/X35/X59/X62))+\ldots$$
$$-(L/M113-X62/M112+M28+\ldots$$
$$+(772/774/X62)/M111+M23+M001+\ldots$$
$$+(280+\ldots$$
$$+460/772/774/X62))-\ldots$$
$$+(L/M112+M28+222+223+231+$$
$$254+292+423+(460/249+461+551+810)+(524+668+634+636/820)+543+581+679+(955+265+657$$
$$+(140A/200A)/956+570+(201A/208A)))+809/M112+M28+221+222+231+254+292+(349/460)+423+$$
$$+(460/249+461+551+810)+(524+668+634+636/820)+543+581+679+955+265+657+(140A/200A)+$$
$$800/M112+M28+221+222+231+254+292+(349/460)+423+(460/249+461+551+810)+(524+668+6$$
$$34+636/820)+543+581+679+956+570+(201A/208A)+800/M113+231+249+254+265+441+(460/46$$
$$1)+\ldots$$
$$+(551/460)+(810/460)+(524+668+634+636/820)+543+580A+809/M113+231+249+254+265+(34$$
$$9/460)+441+(460/461)+(551/460)+(810/460)+(524+668+634+636/820)+543+580A+800/M111+M2$$
$$3+M001+221+231+249+254+292+423+(524+634$$
$$\ldots$$
$$+X34/X51/X52/X54/X55/X57/X58/X60/X61/X63/X64))$$

- **Translation to SAT:**
  1. Propositional Dynamic Logic (PDL)
  2. Consistency conditions as SAT problems (monotonicity of supplementation)
Correctness Conditions

- Conditions: $B \land E$ with
  
  $B := (Z_1 \Rightarrow c_1) \land \cdots \land (Z_n \Rightarrow c_n) \land$

  $(c_1 \Rightarrow B_1) \land \cdots \land (c_n \Rightarrow B_n)$

- $B$ characterizes all constructible, extended orders
- $E$ is a (small) test condition

Correctness conditions include:

- For each part there is be at least one constructible order
- For each equipment option there is be at least one constructible order with and one constructible order without it
Demo
Application 2: Hardware Verification

- Correctness of HW-designs
  - At gate-level
  - Properties specified in temporal logic
Model Checking (MC)

- Given: hardware description $M$ (finite transition system, model), property $P$ (in temporal logic, e.g. LTL, CTL)
- Check whether property $P$ holds in $M$, i.e. whether $M$ is a model of $P$ ($M \models P$)
- Hardware description $M$: set of initial states plus transition relation
- Typical properties $P$:
  - Safety properties: “$x$ always holds” (i.e. in every state reachable from some distinguished initial states)
  - Liveness properties: “there will be a point in time when $x$ holds” (e.g. a request is answered)
  - In “$x$ always holds”: $x$ typically a propositional formula
Bounded Model Checking (BMC)

- **Original MC Question:**
  - Show that “always $p$” holds (i.e. holds in all reachable states)

- **BMC Question:**
  - Show that “always $p$” holds on all runs of length $\leq k$ (for some $k$), or formulated (negatedly) as a SAT problem:
    - Is there a path of length $\leq k$ from an initial state to a state where $p$ does not hold?
  - **Initial states:** given as predicate $l(s)$ over the state variables $s = (x_1, \ldots, x_n)$
  - **Transition relation:** given as predicate $\tau(s, s')$ of state $s$ and successor state $s'$
BMC as SAT

- Formula to check for satisfiability:

$$I(s_0) \land \bigwedge_{i=0}^{k-1} \tau(s_i, s_{i+1}) \land \bigvee_{i=0}^{k} \neg p(s_i)$$

Is there a path of length \(\leq k\) from an initial state to a state where \(p\) does not hold?

- If such a path exists, we have found a counter-example for “always \(p\)”
- Otherwise, we know that no such path of length \(\leq k\) exists; we then may increase \(k\) and check again
Consider a 2-bit counter, counting repeatedly from \( c=0 \) to \( c=2 \). Prove that when initially \( c \neq 3 \), then always \( c \neq 3 \).

- **2 state bits:** \( s_i = (x_0^i, x_1^i) \), counter \( c_i = 2 \cdot x_1^i + x_0^i \)
- **Initial state condition:** \( I(s_0) = \neg (x_0^0 \wedge x_1^0) \) (i.e., \( c_0 \neq 3 \))

**Transition relation:**

\[
x'_1 = x_0, \quad x'_0 = \neg (x_0 \lor x_1)
\]

**Circuit:**
BMC Example (cont’d)

- **Transition relation** in form \( \tau(s, s') \):
  \[
  \tau(s_i, s_{i+1}) = (x_1^{i+1} \iff x_0^i) \land (x_0^{i+1} \iff \neg(x_1^i \lor x_0^i))
  \]

- **Property** \( p(s) \):
  \[
  p(s_i) = \neg(x_1^i \land x_0^i)
  \]

- **SAT-Solver** will confirm that property holds for all \( k \).
BMC in the Industry

- BMC and SAT techniques widely accepted nowadays:
  - Intel, AMD, IBM, Infineon, ...
  - Cadence, ...
- Fully-automated tools: „push-button technology“
- Also used in conjunction with ATP methods (e.g. FP verification at Intel)
Further Applications

- (Hardware) Equivalence Checking
- Asynchronous circuit synthesis (IBM)
- Software-Verification
- Expert system verification
- Planning (air-traffic control, telegraph routing)
- Scheduling (sport tournaments)
- Finite mathematics (quasigroups)
- Cryptanalysis
Part 2: Explaining the Success of SAT
Well-known: (3-)SAT is NP-complete
Best known theoretical upper bound (for 3-SAT): $1.473^n$ (Brueggemann, Kern; 2004)
- 100 vars in 1 sec $\Rightarrow$ 1000 vars in $3.41 \cdot 10^{15}$ secs
Largest BMC-instance solved at SAT Competition:
$>$370,000 variables, $>$7 mio. clauses ($<$ 200 min.)
$\Rightarrow$ Large gap between theoretical and empirical results. So why this?
DPLL-Algorithm

```java
boolean DPLL(ClauseSet S)
{
    while ( S contains a unit clause \( \{L\} \) ) {
        delete from S clauses containing \( L \); // unit-subsumption
        delete \( \neg L \) from all clauses in S; // unit-resolution
    }
    if ( \( \Box \in S \) ) return false; // empty clause?
    while ( S contains a pure literal \( L \) )
        delete from S all clauses containing \( L \);
    if ( S = \( \emptyset \) ) return true; // no clauses?
    choose a literal \( L \) occurring in S; // case-splitting
    if ( DPLL( S \cup \{ \{L\} \} ) ) return true; // first branch
    else if ( DPLL( S \cup \{ \{\neg L\} \} ) ) return true; // second branch
    else return false;
}
```
Why is the DPLL-Algorithm so Successful?

- Highly optimized implementations
  - Clause learning (no-good learning)
  - Fast Boolean constraint propagation (*watched literals* data structure)
  - Improved (dynamic) variable selection heuristics (VSIDS, locality considered)
  - Rapid random restarts (to overcome heavy-tail behavior)
Tractable SAT Instances

- Tractable subclasses:
  - 2-SAT, Horn-SAT, q-Horn-SAT, ... (syntactical)
  - Bounded (hyper-)tree-width (structural)
  - Do not occur frequently in practice

- Fraction of 2-clauses (2+p-SAT) in Random-3-SAT

- „Structure“ in problem instances
  - Implication chains (of 2-clauses)
  - Independent components
  - Other, graph-based notions of structure
SAT Instances as Graphs

- Interaction graph [Rish&Dechter 2000]
  (variables as nodes, clauses as edges)
- Factor graph [Kschischang et al. '98, Braunstein et al. '05]
  (bi-partite graph including variable- and clause-vertices)
- Implication graph [Aspvall et al. '79]
  (implicational structure, for 2-clauses only)
- Slight variants of these graph representations
  (e.g. co-occurrence of literals)
Visualization of SAT Instances

- Variables are nodes, clauses are (sets of) edges
- Visual emphasis on 2-clauses:
  - Use graph layout algorithms
longmult8-B
random100
Demo
Summary

- Two industrial applications of SAT:
  - Bounded model checking (BMC)
  - Product configuration

- Structural analysis:
  - Why are SAT-Solvers so successful?

- Future:
  - New applications (e.g. SW verification), improved implementations
  - Better understanding of what the really hard problems are