An Introduction to SAT Solving

Marc Moreno Maza

University of Western Ontario, November 15, 2017
Plan

The Boolean satisfiability problem

Tseytin’s transformation

How SAT solvers work
Plan

The Boolean satisfiability problem

Tseytin’s transformation

How SAT solvers work
Propositional formula

Definition
Let $V$ be a finite set of Boolean valued variables. A \textit{propositional formula} on $V$ is defined inductively as follows:

- each of the constants false, true is a propositional formula on $V$,
- any element of $V$ is a propositional formula on $V$,
- if $\phi$ and $\phi'$ are propositional formulas on $V$ then $\neg \phi$, $(\phi)$, $\phi \land \phi'$, $\phi \lor \phi'$, $\phi \rightarrow \phi'$, $\phi \leftrightarrow \phi'$ are propositional formulas on $V$ as well.

Examples and counter-examples

- $p \lor \neg q$ is a propositional formula on $V = \{p, q\}$,
- $p + \neg q$ is \textbf{not} a propositional formula on $V = \{p, q\}$,
- $p \lor \neg q$ is \textbf{not} a propositional formula on $V = \{p\}$,
- $(\ )$ is \textbf{not} a propositional formula on $V = \{p, q\}$.
Assignment

Definition
Let again $V$ be a finite set of Boolean valued variables.

- An assignment on $V$ is any map from $V$ to \{false, true\}.
- Any assignment $v$ on $V$ induces an assignment on the propositional formulas on $V$ by applying the following rules: if $\phi$ and $\phi'$ are propositional formulas on $V$ then we have:
  1. $v(\neg \phi) = \neg v(\phi)$,
  2. $v(\phi \land \phi') = v(\phi) \land v(\phi')$,
  3. $v(\phi \lor \phi') = v(\phi) \lor v(\phi')$,
  4. $v(\phi \rightarrow \phi') = v(\phi) \rightarrow v(\phi')$,
  5. $v(\phi \leftrightarrow \phi') = v(\phi) \leftrightarrow v(\phi')$.

Example
For the set of Boolean variables $V = \{p, q\}$. define $v(p) = false$ and $v(q) = true$. Then, we have:

- $v(p \rightarrow q) = true$.
- $v(p \leftrightarrow q) = false$. 
Satisfiability (1/3)

Definition
Let again $V$ be a finite set of Boolean valued variables and let $\phi$ propositional formula on $V$.

- An assignment $v$ on $V$ a *satisfying assignment* for $\phi$ if we have $v(\phi) = \text{true}$.
- The propositional formula $\phi$ is said *satisfiable* if there exists a satisfying assignment for $\phi$.

Deciding whether or not a propositional formula is satisfiable is called the *Boolean satisfiability problem*, denoted by SAT.

Examples

- $p \land (q \lor \neg p) \land (\neg q \lor \neg r)$ is satisfiable for $v(p) = \text{true}$, $v(q) = \text{true}$, $v(r) = \text{false}$.
- $p \land (q \lor \neg p) \land (\neg q \lor \neg p)$ is unsatisfiable.
Remarks

- Simple method for checking satisfiability: the *truth table* method, that is, check all $2^n$ possibilities for $v$ where $n$ is the number of variables in $V$.
- This always works, but has a running time growing exponentially with $n$.
- Practical algorithms and software solving SAT problems also run in time $O(2^n)$ but plays many tricks to terminate computations as early as possible.
Satisfiability (2/3)

Eight queens puzzle

- For $n = 4$, there are two solutions.
- **Exercise**: how to phrase the search for those solutions into a SAT problem?
- **Hints**:
  - What should the Boolean variables represent?
  - What should the propositional formula represent?
- Remember the rules:
  - at most one (and at least one) queen in every row,
  - at most one (and at least one) queen in every column,
  - at most one queen in every diagonal.
SAT solvers

- Modern SAT solvers are based on resolution, and only apply on conjunctive normal form (CNF)
- A *conjunctive normal* form (CNF) is a conjunction of clauses
- A *clause* is a disjunction of literals
- A *literal* is a variable or the negation of a variable
- Hence a CNF is of the shape

\[ \wedge_{1 \leq i \leq s} \left( \vee_{1 \leq j \leq t} \ell_{i,j} \right) \]

SAT solvers have many applications. Problems involving

- binary arithmetic
- program correctness
- termination of rewriting
- puzzles like Sudoku

can be encoded as SAT problems
The yices SAT solver (1/2)

Calling sequence

- The yices is simply used by calling
  
  ```
  yices -e -smt test.smt
  ```

  where test.smt contains the formula to test.

- The input file uses a Lisp-like syntax where and and and or can any number of arguments.

Availability

- The yices SAT solver is publicly available at http://yices.csl.sri.com/old/download-yices1-full.html for Linux, MacOS and Windows.

- yices requires the GMP library https://gmplib.org/.

- **Important**: Here, we use the version 1 of yices.
The yices 1 SAT solver (2/2)

Example

(benchmark test.smt
 :extrapreds ((A) (B) (C) (D))
 :formula (and
 (iff A (and D B))
 (implies C B)
 (not (or A B (not D)))
 (or (and (not A) C) D)
 ))

produces

sat
 (= A false)
 (= B false)
 (= D true)
 (= C false)
Plan

The Boolean satisfiability problem

Tseytin’s transformation

How SAT solvers work
Conversion to CNF: bad news

Proposition
An arbitrary propositional logic can be reduced to an equisatisfiable CNF formula.

Proof
This is done by applying the rules of Boolean algebra. See previous lectures or the monotone laws in https://en.wikipedia.org/wiki/Boolean_algebra.
Proceeding as in the proof blows up the size of the formula. Consider:

\[(p_1 \land \cdots \land p_n) \lor (q_1 \land \cdots \land q_m)\]

will produce \(nm\) clauses. In fact, one can prove the following result, given here without proof.

Proposition
Every CNF equivalent to the formula below has at least \(2^n - 1\) clauses;

\[\cdots((p_1 \leftrightarrow p_2) \leftrightarrow p_3)\cdots \leftrightarrow p_n\]
Conversion to CNF: good news

Definition
Propositional formulas $\alpha$ and $\beta$ are equisatisfiable if: one is satisfiable iff the other is satisfiable.

Proposition
An arbitrary propositional logic can be reduced to an equisatisfiable (not equivalent) CNF formula in polynomial time and polynomial space.

Sketch of proof and pseudo-algorithm
Apply the following rules in order to obtain a CNF or a just a constant:
- eliminate constants; for instance replace $p \land \text{true}$ with $p$,
- apply the rules of Boolean algebra to simplify; for instance, replace $p \land p$ with $p$,
- add new propositional variables, which act as names for sub-formulas of the original formula; see the example on the next slide.
Tseytins transformation: example (1/2)
Consider the following formula $\phi := ((p \lor q) \land r) \rightarrow (\neg s)$
Consider all sub-formulas (except the variables themselves):

\[
\begin{align*}
\neg s & \quad (1) \\
p \lor q & \quad (2) \\
(p \lor q) \land r & \quad (3) \\
((p \lor q) \land r) \rightarrow (\neg s) & \quad (4)
\end{align*}
\]

Introduce a new variable for each sub-formula:

\[
\begin{align*}
x_1 & \leftrightarrow \neg s & \quad (5) \\
x_2 & \leftrightarrow p \lor q & \quad (6) \\
x_3 & \leftrightarrow x_2 \land r & \quad (7) \\
x_4 & \leftrightarrow x_3 \rightarrow x_1 & \quad (8)
\end{align*}
\]

Conjunct all substitutions and the substitution for $\phi$:

\[
T(\phi) := x_4 \land (x_4 \leftrightarrow x_3 \rightarrow x_1) \land (x_3 \leftrightarrow x_2 \land r) \land (x_2 \leftrightarrow p \lor q) \land (x_1 \leftrightarrow \neg s)
\]
Tseytins transformation: example (1/2)

All substitutions can be transformed into CNF, e.g.

\[ x_2 \leftrightarrow p \lor q \equiv x_2 \rightarrow (p \lor q) \land ((p \lor q) \rightarrow x_2) \quad (9) \]
\[ \equiv (\neg x_2 \lor p \lor q) \land (\neg (p \lor q) \lor x_2) \quad (10) \]
\[ \equiv (\neg x_2 \lor p \lor q) \land ((\neg p \land \neg q) \lor x_2) \quad (11) \]
\[ \equiv (\neg x_2 \lor p \lor q) \land (\neg p \lor x_2) \land (\neg q \lor x_2) \quad (12) \]

See https://en.wikipedia.org/wiki/Tseytin_transformation for more details on Tseytins transformation.
Plan

The Boolean satisfiability problem

Tseytin’s transformation

How SAT solvers work


**Naive solver**

**Input:** A propositional formula $\phi$ on a finite set $V$ and a partially defined assignment $v$ on $V$.

**Output:** true if $\phi$ is satisfiable, false otherwise.

- if every clause of $\phi$ has a true literal, return true;
- if any clause of $\phi$ has all false literals, return false;
- choose an $x \in V$ that is unassigned in $v$ and choose $b \in \{\text{true, false}\}$;
  - let $v(x) = b$;
  - if $\phi$ is satisfiable, return true;  
  - let $v(x) = \neg b$;
  - if $\phi$ is satisfiable, return true;
- unassign $v(x)$; # backtracking takes place here
  return false;

This procedure can terminate early if:

- the formula is satisfied before all truth assignments are tested,
- all clauses are false before all variables have been assigned.
The pure literal rule

Proposition
If a variable is always positive (that is, never appears with a $\neg$) or always negative (that is, always appears with a $\neg$) in a CNF formula, you only need to set it to one value true for positive variables, false for negative variables. Note that such variables are called pure.

Proof
- Suppose $x$ occurs only as positive literals in $\phi$,
- If $\phi$ is satisfied by $v$ and $v(x) = false$, then $\phi$ is also satisfied by $v'$ which is identical to $v$ except that $v'(x) = true$.
- so dont bother trying $v(x) = false$.

Note that literals may become pure as variables are assigned, because all clauses in which a variable has one value may become true because of other variables.
Unit propagation

Principle

- Unit propagation (a.k.a Boolean constraint propagation or BCP) is one key component to fast SAT solving.
- Whenever all the literals in a clause are false except one, the remaining literal must be true in any satisfying assignment (such a clause is called a unit clause).
- Therefore, the algorithm can assign it to true immediately.
- After choosing a variable there are often many unit clauses.
- Setting a literal in a unit clause often creates other unit clauses, leading to a cascade
- Based on those observations, we define a sub-algorithm BCP as follows

BCP

- Repeatedly search for unit clauses, and set unassigned literal to required value.
- If a literal is assigned conflicting values, return false else return true.
**Davis-Putnam-Logemann-Loveland Algorithm**

**Input:** A propositional formula \( \phi \) on a finite set \( V \) and a partially defined assignment \( \mathbf{v} \) on \( V \)

**Output:** true if \( \phi \) is satisfiable, false otherwise.

- if every clause of \( \phi \) has a true literal, return true;
- if any clause of \( \phi \) has all false literals, return false;
- if the sub-algorithm BCP returns false, then return false;
- choose an \( x \in V \) that is unassigned in \( \mathbf{v} \) and choose \( b \in \{\text{true, false}\} \);
  - let \( \mathbf{v}(x) = b \);
  - if \( \phi \) is satisfiable, return true; \# this is a recursive call
  - let \( \mathbf{v}(x) = \neg b \);
  - if \( \phi \) is satisfiable, return true; \# this is a recursive call
- unassign \( \mathbf{v}(x) \); \# backtracking takes place here
  - return false;

- Note that recursive calls and calls to BCP modify \( \phi \) and \( \mathbf{v} \).
References

- http://yices.csl.sri.com(old/download-yices1-full.html (The yices software)
- http://0a.io/ boolean-satisfiability-problem-or-sat-in-5-minutes/

This lecture follows partly a presentation by Hans Zantema (Eindhoven University of Technology) and another by David L. Dill (Stanford University).