Chapter 9 – Satisfiability of Boolean Expressions

Albert-Ludwigs-Universität Freiburg

Dr. Tobias Schubert, Dr. Ralf Wimmer

Professur für Rechnerarchitektur
WS 2016/17

Literature

- A. Biere, M. J. H. Heule, H. van Maaren, T. Walsh: Handbook of Satisfiability, IOS Press, 2009
- Tons of workshop, conference, and journal papers ...

Propositional logic

Definition (Syntax of Propositional Logic)

Let $x_1, \ldots, x_n$ be a set of variables. A propositional logic formula is defined through the following inductive process:

1. Every variable x_i is a formula.
2. For every formula F_1 and F_2
   - the conjunction $(F_1 \wedge F_2)$ and
   - the disjunction $(F_1 \vee F_2)$ are also formulas.
3. For every formula F, its negation $(\neg F)$ is a formula.
4. Only the formulas which can be built using the above three rules belong to the set of propositional logic formulas.

As a context-free grammar:

$$F ::= x_i \mid \neg F \mid (F \wedge F) \mid (F \vee F)$$

Propositional logic

Definition (Semantics of Propositional Logic)

An assignment $\mathscr{A}_x : \{x_1, \ldots, x_n\} \to \{0, 1\}$ is a mapping that assigns either the value 0 or 1 to each variable of the propositional logic formula.

$\mathscr{A}_x$ is extended to the mapping $\mathscr{A} : \{F \,|\, F \text{ Formula}\} \to \{0, 1\}$, that assigns a truth value to each formula:

1. If x_i is a variable, then:
   - $\mathscr{A}(x_i) = \mathscr{A}_x(x_i)$.
2. If F is a conjunction/disjunction, then:
   - $\mathscr{A}(F_1 \wedge F_2) = 1 \Leftrightarrow \mathscr{A}(F_1) = 1$ and $\mathscr{A}(F_2) = 1$.
   - $\mathscr{A}(F_1 \vee F_2) = 1 \Leftrightarrow \mathscr{A}(F_1) = 1$ or $\mathscr{A}(F_2) = 1$.
3. If F is a negation, then:
   - $\mathscr{A}(\neg F') = 1 \Leftrightarrow \mathscr{A}(F') = 0$.

Propositional logic

Definition (Satisfiability)

- A propositional logic formula F is satisfiable iff there exists an assignment $\mathscr{A}(F) = 1$.
- Such a satisfying assignment is called a model of F, denoted by $\mathscr{A} \models F$.
- On the other hand, if there exist no assignment $\mathscr{A}$ such that $\mathscr{A}(F) = 1$, then F is unsatisfiable. For every such assignment $\mathscr{A}$ we have $\mathscr{A} \not\models F$.

Propositional logic

Typically restrictions regarding the structure of a formula are made:

Definition (Literal)

A literal L is either a variable ($L = x_i$) or the negation of a variable ($L = \neg x_i$).

Definition (Clause)

A formula $C = (L_1 \vee \ldots \vee L_k)$ with the literals $L_1, \ldots, L_k$ is also indicated as Clause.

Propositional logic

Definition (Conjunctive Normal Form, CNF)

A propositional logic formula F is in conjunctive normal form (CNF) iff it is made of a conjunction of clauses:

$$F = \bigwedge_{j=1}^{m} C_j \qquad \text{with } C_1, \ldots, C_m \text{ Clauses}$$

- Example: $(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee x_4)$
- An assignment $\mathscr{A}$ satisfies a CNF formula F iff every clause in F is satisfied.

Notation

- A clause $C = (\ell_1 \vee \ldots \vee \ell_n)$ can also be seen as a set of literals: $C = \{\ell_1, \ldots, \ell_n\}$
- The empty clause, denoted with $\emptyset$, describes the empty set of literals, and it is unsatisfiable by definition
- A formula in CNF can be seen as a set of clauses, i. e., $F = \{C_1, \ldots, C_m\}$.

Satisfiability problem of propositional logic

Definition (SAT-Problem)

Let F be a propositional logic formula. The question to answer is:

$$\text{Is } F \text{ satisfiable?}$$

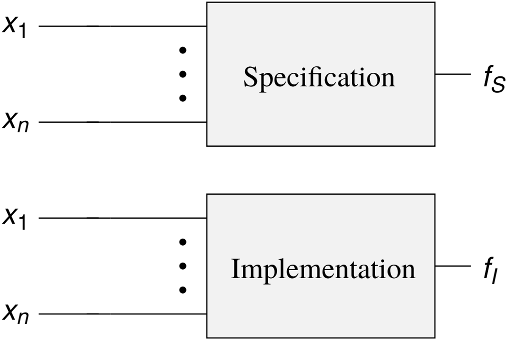
That is, there exists an assignment $\mathscr{A}$ for the variables in F so that $\mathscr{A}(F) = 1$ hold?

- The terms propositional logic formula and Boolean formula have the same meaning.
- Techniques for solving instances of the SAT-problem are called SAT-algorithms or also SAT-solvers

SAT for the verification of combinatorial circuits

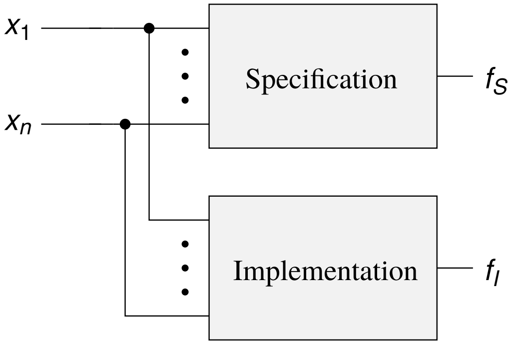
- Given
  - Specification and implementation of a combinatorial circuit
- Question
  - Are the specification and the implementation equivalent?
- Approach for SAT-based equivalence checking
  - Generate a so-called Miter-circuit from specification and implementation
  - Build a Boolean formula from the Miter representation
  - Solve the formula with a SAT algorithm
- The specification and the implementation of a combinatorial circuit are equivalent iff the Boolean formula generated from the Miter is unsatisfiable

Construction of the Miter circuit



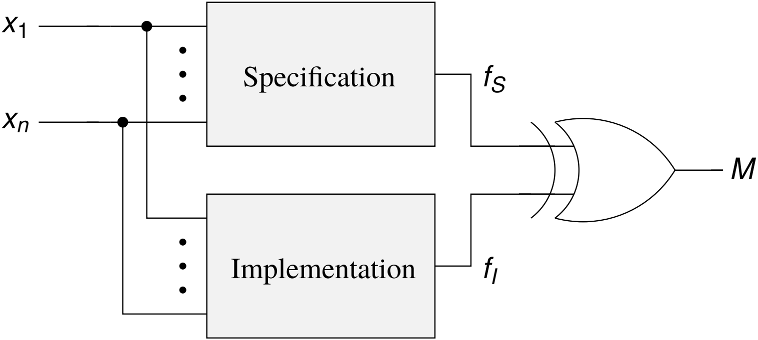
$\Rightarrow$ Connect corresponding inputs

Construction of the Miter circuit



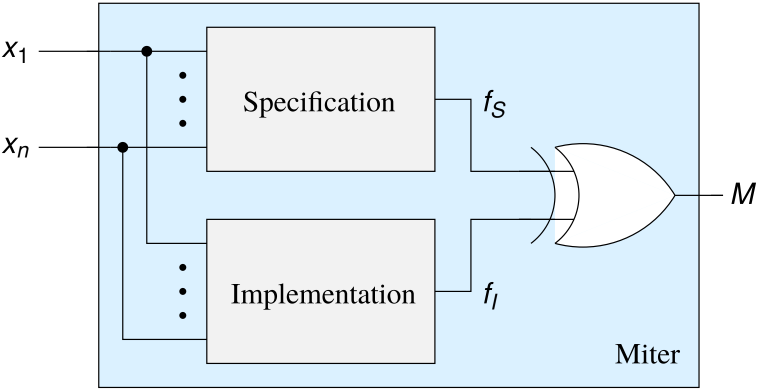
$\Rightarrow$ Link corresponding outputs by EXOR gates

Construction of the Miter circuit



$\Rightarrow$ Miter circuit

Construction of the Miter circuit



$\Rightarrow M = 1 \Leftrightarrow$ Specification & Implementation not equivalent

SAT for the verification of combinatorial circuits

Remarks:

- The drafted method can be extended to combinatorial circuits having multiple outputs
- Usually SAT-algorithms take as input only CNF formulas, that means the Boolean function of the Miter circuit must be translated into a CNF representation
- In BDD-based equivalence checking the limiting resource is the available memory, whereas in the SAT-based approach this is the runtime of the solving algorithm.

SAT for the verification of combinatorial circuits

In the following are discussed:

- conversion into a CNF Boolean formula of the Miter circuit,
- complexity of the SAT problem,
- algorithms for SAT solving.

Conversion of a propositional logic formula into CNF

Definition (Equivalence)

Two propositional logic formulas F and G are equivalent, denoted with $F \equiv G$, iff for every assignment $\mathscr{A}$ suitable for F and G, $\mathscr{A}(F) = \mathscr{A}(G)$ holds.

Theorem

Every propositional logic formula F can be translated into an equivalent CNF formula F'.

Beweis.

This can be shown by induction over the formula composition. □

UNI FREIBURG

Conversion to CNF

- Given: a propositional logic formula F
- Conversion:
  1. Replace in F every sub-formula of the form

     $\neg\neg F_1$ by F_1,

     $\neg(F_1 \wedge F_2)$ by $(\neg F_1 \vee \neg F_2)$,

     $\neg(F_1 \vee F_2)$ by $(\neg F_1 \wedge \neg F_2)$,

     until no more sub-formulas of that kind exist in F.
  2. Replace in F every sub-formula of the form

     $F_1 \vee (F_2 \wedge F_3)$ by $(F_1 \vee F_2) \wedge (F_1 \vee F_3)$,

     $(F_1 \wedge F_2) \vee F_3$ by $(F_1 \vee F_3) \wedge (F_2 \vee F_3)$,

     until no more sub-formulas of that kind occur in F.
- Result: CNF formula F' equivalent to F

Conversion to CNF

Definition (Size of a formula)

The size of a propositional logic formula F, denoted as $|F|$, is defined as the number of the operators $\Diamond$ with $\Diamond \in \{\wedge, \vee, \neg\}$ that occur in F.

Theorem

There exist propositional logic formulas having size $(2 \cdot m - 1)$ *for which every equivalent CNF formula have a size of* $(m \cdot 2^m - 1)$

Conversion to CNF

Proof

Consider formulas built as follows

$$F_m = \bigvee_{j=1}^{m} (x_{j,1} \wedge x_{j,2})$$

with different pairs of literals, in this case only positive $x_{1,1}$, $x_{1,2}$, ..., $x_{m,1}$, $x_{m,2}$. The size of this kind of formulas clearly amounts to $(2 \cdot m - 1)$. A minimal equivalent formula F'_m in CNF has the shape of

$$F'_m = \bigwedge_{k_1,\dots,k_m \in \{1,2\}} (x_{1,k_1} \vee \dots \vee x_{m,k_m})$$

with a total of 2^m clauses. For conjuncting the clauses, $(2^m - 1)$ AND connections are needed. As every clause is made of m literals, that requires $(m-1)$ OR connections for each, $|F'_m|$ is then:

$$|F'_m| = 2^m - 1 + 2^m \cdot (m-1) = m \cdot 2^m - 1.$$

Conversion to CNF

Example 1

- Given
  - $F_1 = x_1 x_2 \vee x_3 x_4$
  - $m = 2$
  - $|F_1| = (2 \cdot m - 1) = 3$
- Conversion of F_1 into the equivalent CNF F' with $F_1 \equiv F'$
  - $F' = (x_1 \vee x_3) \wedge (x_2 \vee x_3) \wedge (x_1 \vee x_4) \wedge (x_2 \vee x_4)$
  - $|F'| = (m \cdot 2^m - 1) = 7$

Conversion to CNF

Example 2

- Given
  - $F_2 = x_1 x_2 \vee x_3 x_4 \vee \ldots \vee x_{37} x_{38} \vee x_{39} x_{40}$
  - $m = 20$
  - $|F_2| = 39 = (2 \cdot m - 1)$
- CNF formula F'', equivalent to F_2, has a size of
  - $|F''| = (m \cdot 2^m - 1) = (20 \cdot 2^{20} - 1) = (20 \cdot 1048576 - 1)$ $= 20\,971\,519$

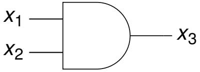
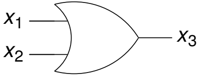
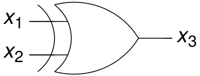
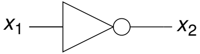
Tseitin transformation

In order to avoid the exponential size of the CNF form obtained from the formula created from the function F of the circuit, some alternative techniques can be applied:

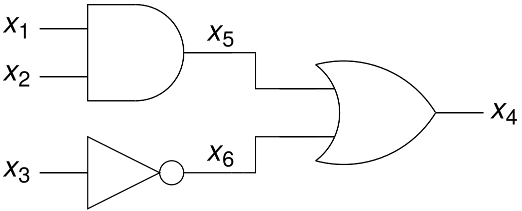
- Define a satisfiability equivalent CNF F' equivalent to F that is satisfiable iff F is satisfiable
- For each gate output insert an additional variable; in general the CNF F' will have variables which do not occur in F
- For each gate realize a “characteristic function” in CNF which evaluates to 1 for every possible consistent signal configuration
- Put together the individual gates using an AND connection to obtain the final CNF formula

$\Rightarrow$ Tseitin transformation

Tseitin transformation

Gates	Function	CNF formula
	$x_3 \equiv x_1 \wedge x_2$	$(\neg x_3 \vee x_1) \wedge (\neg x_3 \vee x_2) \wedge$ $(x_3 \vee \neg x_1 \vee \neg x_2)$
	$x_3 \equiv x_1 \vee x_2$	$(x_3 \vee \neg x_1) \wedge (x_3 \vee \neg x_2) \wedge$ $(\neg x_3 \vee x_1 \vee x_2)$
	$x_3 \equiv x_1 \oplus x_2$	$(\neg x_3 \vee x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_1 \vee \neg x_2) \wedge$ $(x_3 \vee \neg x_1 \vee x_2) \wedge (x_3 \vee x_1 \vee \neg x_2)$
	$x_2 \equiv \neg x_1$	$(x_2 \vee x_1) \wedge (\neg x_2 \vee \neg x_1)$

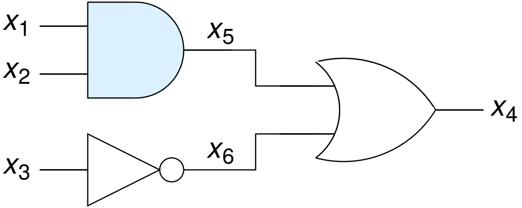
Tseitin transformation



$$F_{SK} = (x_1 \wedge x_2) \vee \neg x_3$$

$$\begin{aligned} F_{SK}^{CNF} = {} & (\neg x_5 \vee x_1) \wedge (\neg x_5 \vee x_2) \wedge (x_5 \vee \neg x_1 \vee \neg x_2) \wedge \\ & (x_6 \vee x_3) \wedge (\neg x_6 \vee \neg x_3) \wedge \\ & (x_4 \vee \neg x_5) \wedge (x_4 \vee \neg x_6) \wedge (\neg x_4 \vee x_5 \vee x_6) \wedge \\ & (x_4) \end{aligned}$$

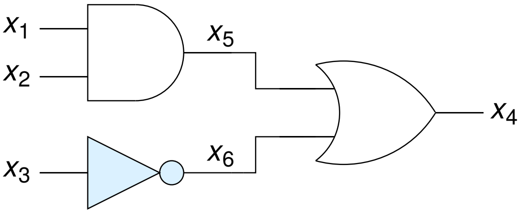
Tseitin transformation



$$F_{SK} = (x_1 \wedge x_2) \vee \neg x_3$$

$$\begin{aligned} F_{SK}^{CNF} = {} & (\neg x_5 \vee x_1) \wedge (\neg x_5 \vee x_2) \wedge (x_5 \vee \neg x_1 \vee \neg x_2) \wedge \\ & (x_6 \vee x_3) \wedge (\neg x_6 \vee \neg x_3) \wedge \\ & (x_4 \vee \neg x_5) \wedge (x_4 \vee \neg x_6) \wedge (\neg x_4 \vee x_5 \vee x_6) \wedge \\ & (x_4) \end{aligned}$$

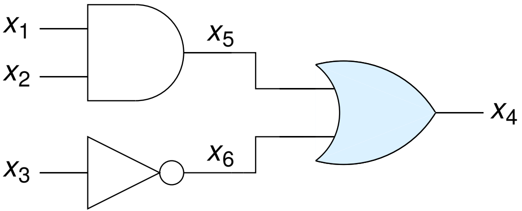
Tseitin transformation



$$F_{SK} = (x_1 \wedge x_2) \vee \neg x_3$$

$$\begin{aligned} F_{SK}^{CNF} = {} & (\neg x_5 \vee x_1) \wedge (\neg x_5 \vee x_2) \wedge (x_5 \vee \neg x_1 \vee \neg x_2) \wedge \\ & (x_6 \vee x_3) \wedge (\neg x_6 \vee \neg x_3) \wedge \\ & (x_4 \vee \neg x_5) \wedge (x_4 \vee \neg x_6) \wedge (\neg x_4 \vee x_5 \vee x_6) \wedge \\ & (x_4) \end{aligned}$$

Tseitin transformation



$$F_{SK} = (x_1 \wedge x_2) \vee \neg x_3$$

$$\begin{aligned} F_{SK}^{CNF} = {} & (\neg x_5 \vee x_1) \wedge (\neg x_5 \vee x_2) \wedge (x_5 \vee \neg x_1 \vee \neg x_2) \wedge \\ & (x_6 \vee x_3) \wedge (\neg x_6 \vee \neg x_3) \wedge \\ & (x_4 \vee \neg x_5) \wedge (x_4 \vee \neg x_6) \wedge (\neg x_4 \vee x_5 \vee x_6) \wedge \\ & (x_4) \end{aligned}$$

Tseitin transformation

As long as for the CNF representation of each single gate only a constant number of clauses is required, the number of clauses in the final CNF will be linear in the number of gates in the circuit (the same holds for the size of the formula)

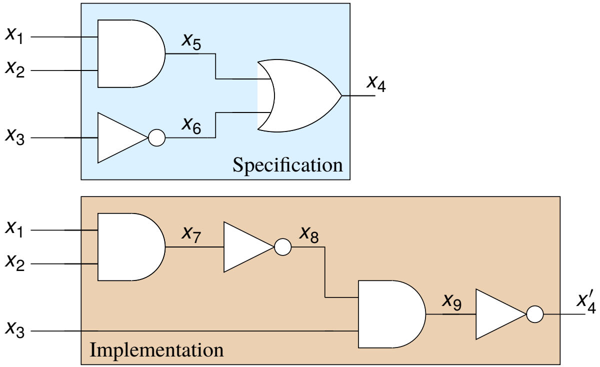
Tseitin transformation

Comparison equivalent / satisfiability equivalent CNF representation

- Given
  - $F = x_1 x_2 \vee x_3 x_4 \vee \ldots \vee x_{37} x_{38} \vee x_{39} x_{40}$
  - $|F| = 39 = (2 \cdot m - 1)$ with $m = 20$
- Conversion of F into equivalent CNF F' with $F \equiv F'$
  - $|F'| = (m \cdot 2^m - 1) = (20 \cdot 2^{20} - 1) = (20 \cdot 1048576 - 1) = 20971519$
- Tseitin transformation of F into satisfiability equivalent CNF F''
  - $|F''| = \underbrace{180}_{\text{20 AND-gates}} + \underbrace{171}_{\text{19 OR-gates}} + \underbrace{38}_{38\ \wedge} = 389$

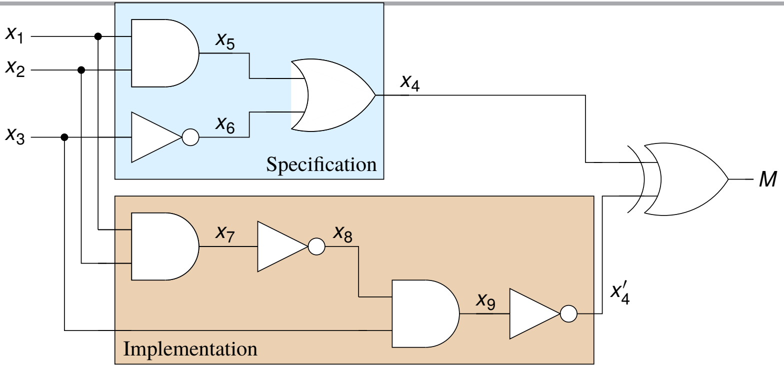
SAT for the verification of combinatorial circuits

Let the specification and the implementation of a combinatorial circuits be defined as follows



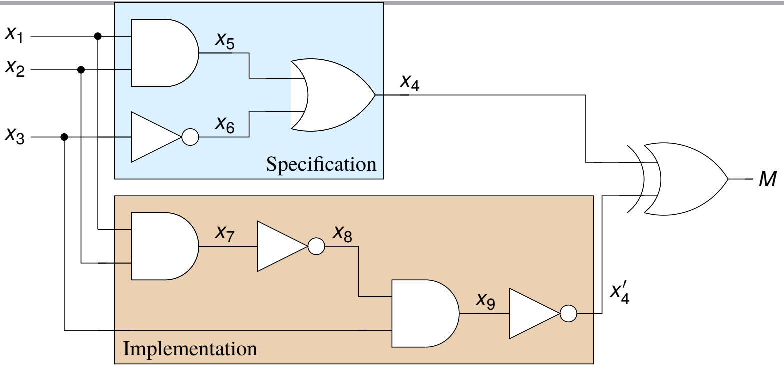
Question: are the specification and the implementation equivalent?

SAT for the verification of combinatorial circuits



$$
\begin{aligned}
F_M = {} & (\neg x_5 \vee x_1) \wedge (\neg x_5 \vee x_2) \wedge (x_5 \vee \neg x_1 \vee \neg x_2) \wedge (x_6 \vee x_3) \wedge (\neg x_6 \vee \neg x_3) \wedge \\
& (x_4 \vee \neg x_5) \wedge (x_4 \vee \neg x_6) \wedge (\neg x_4 \vee x_5 \vee x_6) \wedge (\neg x_7 \vee x_1) \wedge (\neg x_7 \vee x_2) \wedge \\
& (x_7 \vee \neg x_1 \vee \neg x_2) \wedge (x_7 \vee x_8) \wedge (\neg x_7 \vee \neg x_8) \wedge (\neg x_9 \vee x_3) \wedge (\neg x_9 \vee x_8) \wedge \\
& (x_9 \vee \neg x_3 \vee \neg x_8) \wedge (x_9 \vee x_4') \wedge (\neg x_9 \vee \neg x_4') \wedge (\neg M \vee \neg x_4 \vee \neg x_4') \wedge \\
& (\neg M \vee x_4 \vee x_4') \wedge (M \vee \neg x_4 \vee x_4') \wedge (M \vee x_4 \vee \neg x_4') \wedge (M)
\end{aligned}
$$

SAT for the verification of combinatorial circuits



$$
\begin{aligned}
F_M = {} & (\neg x_5 \lor x_1) \land (\neg x_5 \lor x_2) \land (x_5 \lor \neg x_1 \lor \neg x_2) \land (x_6 \lor x_3) \land (\neg x_6 \lor \neg x_3) \land \\
& (x_4 \lor \neg x_5) \land (x_4 \lor \neg x_6) \land (\neg x_4 \lor x_5 \lor x_6) \land (\neg x_7 \lor x_1) \land (\neg x_7 \lor x_2) \land \\
& (x_7 \lor \neg x_1 \lor \neg x_2) \land (x_7 \lor x_8) \land (\neg x_7 \lor \neg x_8) \land (\neg x_9 \lor x_3) \land (\neg x_9 \lor x_8) \land \\
& (x_9 \lor \neg x_3 \lor \neg x_8) \land (x_9 \lor x'_4) \land (\neg x_9 \lor \neg x'_4) \land (\neg M \lor \neg x_4 \lor \neg x'_4) \land \\
& (\neg M \lor x_4 \lor x'_4) \land (M \lor \neg x_4 \lor x'_4) \land (M \lor x_4 \lor \neg x'_4) \land (M)
\end{aligned}
$$

F_M is unsatisfiable $\Rightarrow$ Implementation and specification are equivalent!

Complexity of the SAT problem

- S.A. Cook, 1971: The SAT problem is NP-Complete
- Some special cases can be solved in linear time:
  - 2-SAT (the formula contains only binary clauses)
  - Horn formulas (every clause contains at most one positive literal)

Complexity of the SAT problem

Observation from the practice:

- Nowadays, modern SAT algorithms are able to solve numerous relevant problems in acceptable time
- CNF formulas with several **hundred thousands of variables** and even **millions of clauses** are by now manageable

Applications of SAT algorithms:

- Combinational Equivalence Checking
- Automatic Test Pattern Generation
- Bounded Model Checking
- AI Planning
- Scheduling
- ...

Overview on SAT-algorithms

Complete methods:

- Due to a systematic procedure they can also prove the unsatisfiability of a CNF formula
- DP-Algorithm
  - M. Davis, H. Putnam, 1960
  - based on Resolution
- DLL-Algorithm
  - M. Davis, G. Logemann, D. Loveland, 1962
  - based on depth-first search
- Modern SAT algorithms
  - based on the DLL-algorithm, enriched with efficient data structures and several acceleration and optimization techniques
  - zChaff, MiniSat, MiraXT, precosat, lingeling, antom, Glucose

Overview SAT algorithms

Incomplete methods:

- based on local search
- basic idea:
  - choose an initial valuation to the variables
  - As long as the formula is not satisfied, modify the assignments according to some heuristics (“Flip” single variable valuation)
- GSat, WSat (H.A. Kautz, B. Selman, 1992 & 1996)
- They cannot prove the unsatisfiability of a CNF formula

⇒ Not further considered in the following!

Resolution

Definition (Resolution)

Let C_1 and C_2 be two clauses and L be a literal with the following property: $L \in C_1$ and $\neg L \in C_2$. Then one can compute the clause R

$$R = \left(C_1 \setminus \{L\}\right) \cup \left(C_2 \setminus \{\neg L\}\right)$$

that is denoted as the resolvent of the clauses C_1 and C_2 over L. The notation $R = C_1 \otimes_L C_2$ is commonly used.

Resolution

Example:

- $C_1 = (x_1, x_2, x_3)$, $C_2 = (x_4, \neg x_2)$
  $C_1 \otimes_{x_2} C_2 = (x_1, x_3, x_4)$
- $C_3 = (x_4, x_2, x_3)$, $C_4 = (x_4, \neg x_2)$
  $C_3 \otimes_{x_2} C_4 = (x_3, x_4)$
- $C_5 = (x_4, x_2)$, $C_6 = (\neg x_4, \neg x_2)$
  $C_5 \otimes_{x_2} C_6 = (x_4, \neg x_4)$
- $(x_4, \neg x_4)$ is satisfied for every valuation to x_4, and is called a tautology clause.

Resolution

Lemma (Resolution Lemma)

Let F be a CNF formula and R be the resolvent of two clauses C_1 and C_2 from F. Then F and $F \cup \{R\}$ are equivalent: $F \equiv F \cup \{R\}$.

Resolution

Definition

Let F be a CNF formula. Then $Res(F)$ is defined as

$$Res(F) = F \cup \{R \mid R \text{ is the resolvent of two clauses in } F\}.$$

Moreover is defined:

$$\begin{aligned} Res^0(F) &= F \\ Res^{t+1}(F) &= Res(Res^t(F)) \text{ for } t \geq 0 \\ Res^*(F) &= \lim_{t\geq 0} Res^t(F) \end{aligned}$$

Resolution

Theorem (Resolution Theorem)

A CNF formula F is unsatisfiable if and only if $\emptyset \in Res^*(F)$.

Resolution

A first SAT algorithm can be deduced from Resolution Lemma and Theorem

- Given
  - A CNF formula F
- Procedure
  - Starting from $F = Res^0(F)$, define $F = Res^t(F)$ for increasing $t > 0$ until either the empty clause is derived or Resolution cannot be applied anymore
- Result
  - Case 1. for some $t > 0$: $\emptyset \in Res^t(F) \Rightarrow F$ is unsatisfiable
  - Case 2. for some $t > 0$: $\emptyset \notin Res^t(F) = Res^{t+1}(F) \Rightarrow F$ is satisfiable

UNI FREIBURG

Resolution

Complexity of this naive method

- Since in a clause a variable occurs either as a positive literal, or negative literal, or it does not occur at all, for a formula having n variables the runtime and memory consumption lie in $O(3^n)$ in the worst case.

Resolution

Example:

- Is the following CNF formula F satisfiable?

  $F = (x_1, x_2) \wedge (x_1, \neg x_3) \wedge (\neg x_1, x_3) \wedge (\neg x_1, \neg x_2) \wedge (x_3, \neg x_2) \wedge (\neg x_3, x_2)$

- Procedure based on the algorithm drafted above

  $Res^0(F) = F$

  $Res^1(F) = Res^0(F) \cup \{(x_2, x_3), (x_1, x_3), (\neg x_2, \neg x_3), (x_1, \neg x_2), (\neg x_1, x_2), (\neg x_1, \neg x_3)\}$

  $Res^2(F) = Res^1(F) \cup \{\ldots, (x_1), \ldots, (\neg x_1), \ldots\}$

  $Res^3(F) = Res^2(F) \cup \{\emptyset\}$

Resolution

Example:

- Is the following CNF formula F satisfiable?

  $F = (x_1, x_2) \wedge (x_1, \neg x_3) \wedge (\neg x_1, x_3) \wedge (\neg x_1, \neg x_2) \wedge (x_3, \neg x_2) \wedge (\neg x_3, x_2)$

- Procedure based on the algorithm drafted above

  $Res^0(F) = F$

  $Res^1(F) = Res^0(F) \cup \{(x_2, x_3), (x_1, x_3), (\neg x_2, \neg x_3), (x_1, \neg x_2), (\neg x_1, x_2), (\neg x_1, \neg x_3)\}$

  $Res^2(F) = Res^1(F) \cup \{\dots, (x_1), \dots, (\neg x_1), \dots\}$

  $Res^3(F) = Res^2(F) \cup \{\emptyset\}$

$\Rightarrow$ F is unsatisfiabile!

Resolution

Example:

- Is the following CNF formula F satisfiable?

  $F = (x_1, x_2, x_3) \wedge (x_2, \neg x_3, \neg x_4) \wedge (\neg x_2, x_5)$

- Procedure based on the algorithm drafted above

  $Res^0(F) = F$

  $Res^1(F) = Res^0(F) \cup \{(x_1, x_3, x_5), (\neg x_3, \neg x_4, x_5), (x_1, x_2, \neg x_4)\}$

  $Res^2(F) = Res^1(F) \cup \{(x_1, \neg x_4), (x_1, \neg x_4, x_5), (x_1, \neg x_4, x_2, x_5)\}$

  $Res^3(F) = Res^2(F) = Res^*(F)$

Resolution

Example:

- Is the following CNF formula F satisfiable?

  $F = (x_1, x_2, x_3) \land (x_2, \neg x_3, \neg x_4) \land (\neg x_2, x_5)$

- Procedure based on the algorithm drafted above

  $Res^0(F) = F$

  $Res^1(F) = Res^0(F) \cup \{(x_1, x_3, x_5), (\neg x_3, \neg x_4, x_5), (x_1, x_2, \neg x_4)\}$

  $Res^2(F) = Res^1(F) \cup \{(x_1, \neg x_4), (x_1, \neg x_4, x_5), (x_1, \neg x_4, x_2, x_5)\}$

  $Res^3(F) = Res^2(F) = Res^*(F)$

$\Rightarrow$ F is satisfiable!

To be continued …

There's **much** more to talk about in SAT solving:

- Preprocessing
- Modern efficient algorithms
- Modelling of real-world problems using SAT

Some topic are coming later:

- Complexity $\rightarrow$ Info III
- Reasoning $\rightarrow$ AI / Decision Procedures / Verification
- Modelling $\rightarrow$ AI / Verification