The Boolean Satisfiability (SAT) Problem, SAT Solver Technology, and Equivalence Verification

Priyank Kalla

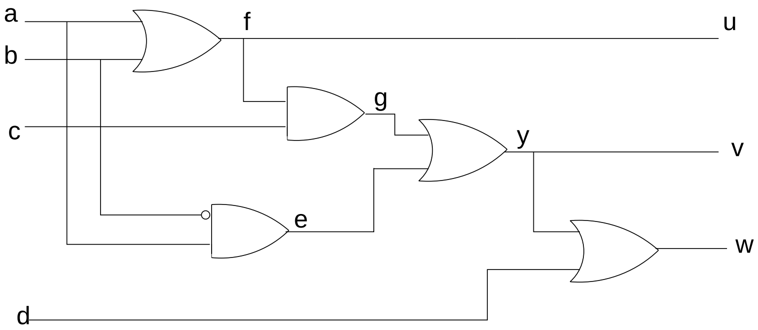
Associate Professor
Electrical and Computer Engineering, University of Utah
kalla@ece.utah.edu
http://www.ece.utah.edu/~kalla

Lectures: Aug 21, 2017 onwards

What is Boolean Satisfiability (SAT)?

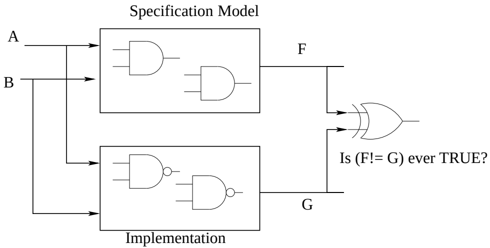
- Given a Boolean formula $f(x_1, \ldots, x_n)$, find an assignment to $x_1, \ldots, x_n$ s.t. $f = 1$
- Otherwise, prove that such an assignment does not exist: problem is infeasible!
- There may be many SAT assignments: find an assignment, or enumerate all assignments (ALL-SAT)
- The formula f is given in conjunctive normal form (CNF), SAT solvers operate CNF representation of f
- Any decidable decision problem can be formulated and solved as SAT
- SAT is fundamental, has wide applications in many areas: hardware & software verification, graph theory, combinatorial optimization, artificial intelligence, VLSI design automation, cryptography/cryptanalysis, planning, scheduling, many more....

- Simulation vector generation: Given the circuit below, find an assignment to primary inputs s.t. $u = 1, v = 1, w = 0$, or prove that one does not exits
- Translate the circuit into CNF, and solve SAT



SAT in Equivalence checking

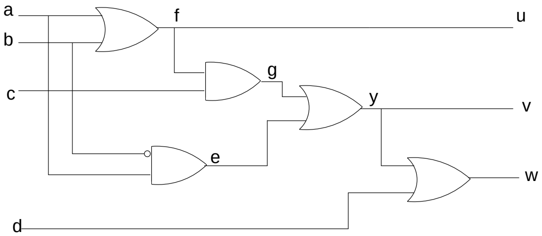
- Prove infeasibility of the miter!
  - Find an assignment to the inputs s.t. $(F \neq G) = 1$ (bug)
  - If no assignment (infeasible), circuits are equivalent
- Model checking: find an assignment s.t. a property is satisfied/falsified



SAT formulation

- A Boolean formula $f(x_1, \ldots, x_n)$ over propositional variables $x_1, \ldots, x_n \in \{0, 1\}$, using propositional connectives $\neg, \vee, \wedge$, parenthesis, and implications $\implies$, $\iff$
  - Example: $f = ((\neg x_1 \wedge x_2) \vee x_3) \wedge (\neg x_2 \vee x_3)$
- A CNF formula representation of f is:
  - a conjunction of clauses
  - each clause is a disjunction of literals
  - each literal is a variable or its negation (complement)
- Example: $f = (\neg x_1 \vee x_2)(\neg x_2 \vee x_3 \vee \neg x_4)(x_1 \vee x_2 \vee x_3 \vee \neg x_4)$
- Alternate notation $f = (x_1' + x_2)(x_2' + x_3 + x_4')(x_1 + x_2 + x_3 + x_4')$
- Any Boolean formula (circuit) can be encoded into CNF

Encode a Circuit to CNF



$$
\begin{aligned}
f &= a \vee b \\
f &\iff a \vee b \quad \text{(equality is a double-implication)} \\
CNF: &\quad (f \implies (a \vee b)) \wedge ((a \vee b) \implies f) \\
&\quad (\neg f \vee (a \vee b)) \wedge (\neg(a \vee b) \vee f) \\
&\quad (\neg f \vee (a \vee b)) \wedge ((\neg a \wedge \neg b) \vee f)) \quad \text{(CNF?)} \\
&\quad (\neg f \vee (a \vee b)) \wedge (\neg a \vee f)(\neg b \vee f)
\end{aligned}
$$

Encode Circuit to CNF

Circuit to CNF: Implication to Clauses

In general, if $f = OP(a, b)$, the CNF representation is:

- $f \iff OP(a, b)$, further simplified as:
- $(f \implies OP(a, b)) \wedge (OP(a, b) \implies f)$
- Translate implication to Boolean formula: $a \implies b$ means $(a' + b)$ is TAUTOLOGY.

- For $f = a \wedge b$, CNF: $(\neg f + a)(\neg f + b)(\neg a + \neg b + f)$
- For $f = a \oplus b$, CNF: $(\neg f + a + b)(f + \neg a + b)(f + a + \neg b)(\neg f + \neg a + \neg b)$
- For the previous circuit, we need to further constrain $u = 1, v = 1, w = 0$ to solve the simulation vector generation problem. Encode constraints $u = 1, v = 1, w = 0$ into CNF as $(u)(v)(w')$
- Conjunct ALL clauses (constraints) and invoke a SAT solver to find a solution

SAT Solving Complexity

- In general, SAT is NP-complete. No polynomial-time algorithm exists to solve SAT (in theory).
- The restricted 2-SAT problem, where every clause contains only 2 literals, can be solved in polynomial time.
- Circuit-to-CNF: Recall, 2-input AND/OR gates need a 3-literal clause for modeling the constraint.
    - Circuit-SAT is therefore also NP-complete.
- However, modern SAT solvers are a success story in Computer Science and Engineering. Efficient heuristics and implementation tricks make SAT solvers very efficient.
- EDA gave a big impetus to SAT solving
- Many large problems can be solved very quickly by SAT solvers.
- So, how is a CNF SAT formula solved?

- An assignment can make a clause satisfied or unsatisfied
- Since $f = C_1 \wedge C_2 \wedge \cdots \wedge C_n$, try to SATISFY each clause C_i
- The first approach by Davis & Putnam [DP 1960]: based on unit clause, pure literal and resolution rules
- Later Davis, Logemann, Loveland [DLL 1962] proposed an alternative backtrack-based search algorithm
- These algorithms are now known as DPLL algorithms
- Modern solvers are highly sophisticated: conclict-driven clause learning (CDCL) and search-space pruning, among many efficient heuristics

Satisfy a clause

A clause is satisfied if any literal is assigned to 1. E.g. for $x_2 = 0$, clause $(x_1 \vee \neg x_2 \vee \neg x_3) = 1$.

Satisfy a clause

A clause is unsatisfied if all literals are assigned to 0. E.g. the assignment of $x_1 = 0, x_2 = x_3 = 1$, makes clause $(x_1 \vee \neg x_2 \vee \neg x_3)$ unsatisfied.

Unit clause

A clause containing a single unassigned literal, and all other literals assigned to 0. E.g., the assignment $x_1 = 0, x_3 = 1$, makes $(x_1 \vee \neg x_2 \vee \neg x_3) = (0 \vee \neg x_2 \vee 0)$ a unit clause. Unit clause forces a necessary assignment $(x_2 = 0)$ for the formula to be TRUE.

- Formula f is satisfied, if all clauses are satisfied; f is unsatisfied, if at least one clause is unsatisfied.

Pure Literals

- A literal is pure if it appears only as a positive literal, or only as a negative literal.
  - $f = (\neg x_1 \vee x_2) \wedge (x_3 \vee \neg x_2) \wedge (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$
  - x_1, x_3 are pure literals.
- Clauses containing pure literals can be easily satisfied.
  - Assign pure literals to the values that satisfy the clauses
  - Pure literals do not cause inconsistent value assignments (or conflicts) to variables.
- Iteratively apply unit clause propagation and pure literal simplifcation on the CNF formula

Resolution

- Resolution Rule: Given clauses $(x \vee \alpha)$ and $(\neg x \vee \beta)$, infer $(\alpha \vee \beta)$
  - $RES(x \vee \alpha, \neg x \vee \beta) = (\alpha \vee \beta)$
- The DP algorithm was resolution-based

Resolution-based SAT

Given CNF formula f, deduce if it is SAT or UNSAT

Resolution-based SAT

Given CNF formula f, deduce if it is SAT or UNSAT

- Complete algorithm: Iterate the following steps

Resolution-based SAT

Given CNF formula f, deduce if it is SAT or UNSAT

- Complete algorithm: Iterate the following steps
  - Select variable x that is not pure (both $x, \neg x$ exist)

Resolution-based SAT

Given CNF formula f, deduce if it is SAT or UNSAT

- Complete algorithm: Iterate the following steps
  - Select variable x that is not pure (both $x, \neg x$ exist)
  - Apply resolution rules between every pair of clauses $(x \vee \alpha)$ and $(\neg x \vee \beta)$; simplify f

Resolution-based SAT

Given CNF formula f, deduce if it is SAT or UNSAT

- Complete algorithm: Iterate the following steps
  - Select variable x that is not pure (both $x, \neg x$ exist)
  - Apply resolution rules between every pair of clauses $(x \vee \alpha)$ and $(\neg x \vee \beta)$; simplify f
  - Remove clauses with pure literals x or $\neg x$

Resolution-based SAT

Given CNF formula f, deduce if it is SAT or UNSAT

- Complete algorithm: Iterate the following steps
  - Select variable x that is not pure (both $x, \neg x$ exist)
  - Apply resolution rules between every pair of clauses $(x \vee \alpha)$ and $(\neg x \vee \beta)$; simplify f
  - Remove clauses with pure literals x or $\neg x$
  - Apply pure literal rules and unit propagation

Given CNF formula f, deduce if it is SAT or UNSAT

- Complete algorithm: Iterate the following steps
    - Select variable x that is not pure (both $x, \neg x$ exist)
    - Apply resolution rules between every pair of clauses $(x \vee \alpha)$ and $(\neg x \vee \beta)$; simplify f
    - Remove clauses with pure literals x or $\neg x$
    - Apply pure literal rules and unit propagation

Terminate when empty clause (UNSAT) or empty formula (SAT)

Deduce SAT/UNSAT by Resolution: Example

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4)$$

Deduce SAT/UNSAT by Resolution: Example

$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4)$

$(\neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4)$

$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4)$

$(\neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4)$

$(\neg x_3 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4)$

Deduce SAT/UNSAT by Resolution: Example

$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4)$

$(\neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4)$

$(\neg x_3 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4)$

(x_3)

Deduce SAT/UNSAT by Resolution: Example

$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4)$

$(\neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4)$

$(\neg x_3 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4)$

(x_3)

Satisfiable!

Backtrack Binary Search for SAT

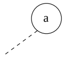
- The [DP 1960] approach using resolution was inefficient
- Then the [DLL 1962] was introduced:
  - Select a variable x, assign either $x = 0$ or $x = 1$ [decision assignment]
  - Simplify formula with unit propagation, pure literal rules [deduce]
  - If conflict, then backtrack [diagnose]
    - If cannot backtrack further, return UNSAT
  - If formula satisfied, return SAT
  - Otherwise, proceed with another decision

DPLL Example

$$f = (a + b' + d)(a + b' + e)(b' + d' + e')(a + b + c + d)(a + b + c + d')(a + b + c' + e)(a + b + c' + e')$$

$$f = (a + b' + d)(a + b' + e)(b' + d' + e')(a + b + c + d)(a + b + c + d')(a + b + c' + e)(a + b + c' + e')$$

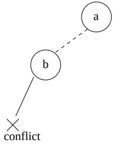
$a = 0$



DPLL Example

$f = (a + b' + d)(a + b' + e)(b' + d' + e')(a + b + c + d)(a + b + c + d')(a + b + c' + e)(a + b + c' + e')$

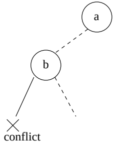
$a = 0$, $b = 1$, conflict, backtrack, change last decision!



DPLL Example

$f = (a + b' + d)(a + b' + e)(b' + d' + e')(a + b + c + d)(a + b + c + d')(a + b + c' + e)(a + b + c' + e')$

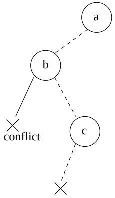
$a = 0, \ b = 0$



DPLL Example

$f = (a + b' + d)(a + b' + e)(b' + d' + e')(a + b + c + d)(a + b + c + d')(a + b + c' + e)(a + b + c' + e')$

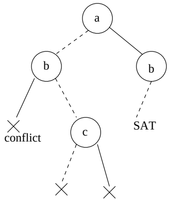
$a = 0,\ b = 0,\ c = 0,$
conflict, backtrack!



DPLL Example

$$f = (a + b' + d)(a + b' + e)(b' + d' + e')(a + b + c + d)(a + b + c + d')(a + b + c' + e)(a + b + c' + e')$$

$$a = 1,\ b = 0$$



Non-chronological Backtracking via CDCL

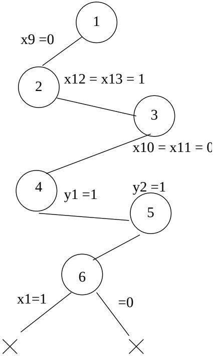
- Previous example shows a chronological backtrack based binary search
- Modern SAT solvers analyze decisions and conflicts to dynamically learn clauses
  - Conflict Driven Clause Learning (CDCL)
  - Solver learns more clauses, and appends them to the original CNF
  - More constraints help to prune the search
  - Results in a non-chronological backtrack-based search
  - The approach is still complete: Will find SAT, or will prove UNSAT
- There are also "incomplete" solvers, that rely on local search
  - Heuristics to guide the search, but search not exhaustive
  - May find a SAT solution if one exists, but cannot prove UNSAT
- There are also SAT pre-processors
  - Input CNF $\mathcal{F}_1$, output CNF $\mathcal{F}_2$, size($\mathcal{F}_1$) $>$ size($\mathcal{F}_2$)

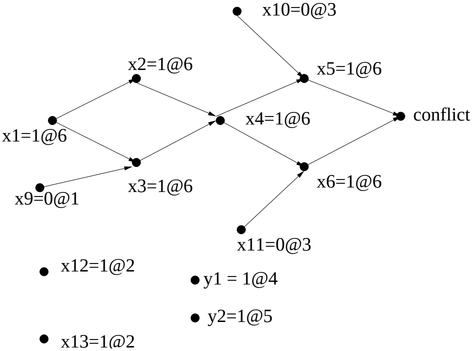
Conflict-Driven Clause Learning (CDCL) solvers

- Modern CDCL-solvers: based on DPLL, but do quite a bit more
  - Learn new constraints while encountering conflicts
  - Enable non-chronological backtracking, thus pruning search-space
  - Branching heuristics: which variable to branch on ($x_i = 0$? or $x_i = 1$?)
  - Heuristics for search re-starts
  - Efficient management of clause-database: minimize learnt clauses, discard unused learnt clauses
- Concept of CDCL from [GRASP, Joao Marques-Silva and Karem Sakallah]
- Read GRASP report on class website

$$(x_1' + x_2)(x_1' + x_3 + x_9)(x_2' + x_3' + x_4)(x_4' + x_5 + x_{10})(x_4' + x_6 + x_{11})$$
$$(x_5' + x_6')(x_1 + x_7 + x_{12}')(x_1 + x_8)(x_7' + x_8' + x_{13}')(y_1 + z_1)(y_2 + z_2)$$

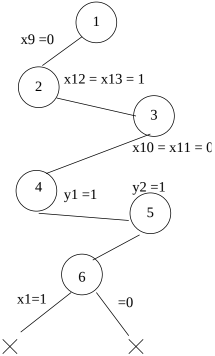
$$(x_1' + x_2)(x_1' + x_3 + {\color{red}x_9})(x_2' + x_3' + x_4)(x_4' + x_5 + x_{10})(x_4' + x_6 + x_{11})$$
$$(x_5' + x_6')(x_1 + x_7 + x_{12}')(x_1 + x_8)(x_7' + x_8' + x_{13}')(y_1 + z_1)(y_2 + z_2)$$

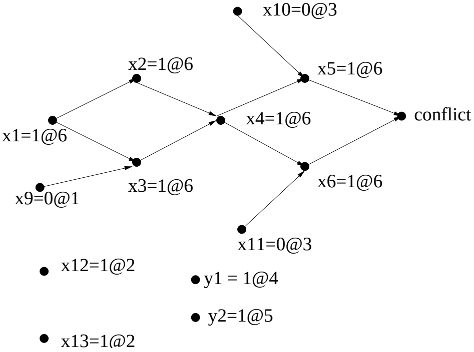




CDCL & Non-Chronological Backtracking [From GRASP]

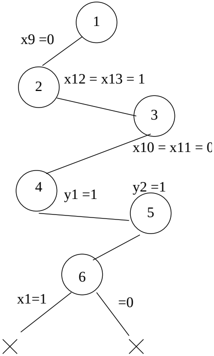
$$(x_1' + x_2)(x_1' + x_3 + x_9)(x_2' + x_3' + x_4)(x_4' + x_5 + x_{10})(x_4' + x_6 + x_{11})$$
$$(x_5' + x_6')(x_1 + x_7 + x_{12}')(x_1 + x_8)(x_7' + x_8' + x_{13}')(y_1 + z_1)(y_2 + z_2)$$

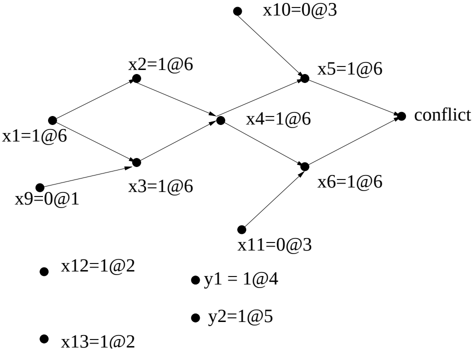




CDCL & Non-Chronological Backtracking [From GRASP]

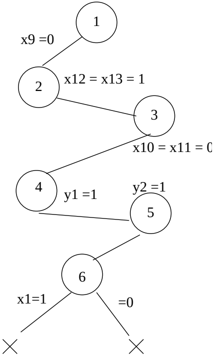
$$(x_1' + x_2)(x_1' + x_3 + {\color{red}x_9})(x_2' + x_3' + x_4)(x_4' + x_5 + {\color{red}x_{10}})(x_4' + x_6 + {\color{red}x_{11}})$$
$$(x_5' + x_6')(x_1 + x_7 + {\color{red}x_{12}'})(x_1 + x_8)(x_7' + x_8' + {\color{red}x_{13}'})({\color{blue}y_1} + z_1)({\color{blue}y_2} + z_2)$$

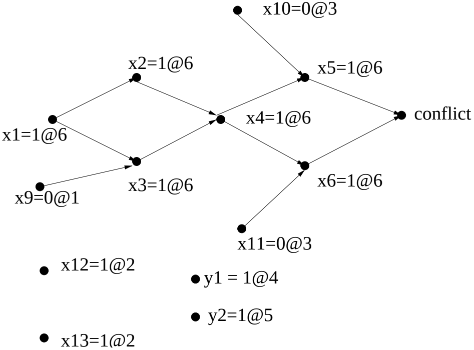




CDCL & Non-Chronological Backtracking [From GRASP]

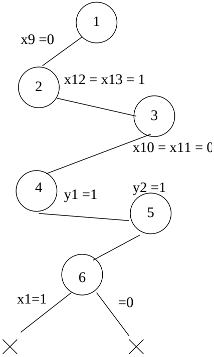
$$(x_1' + x_2)(x_1' + x_3 + x_9)(x_2' + x_3' + x_4)(x_4' + x_5 + x_{10})(x_4' + x_6 + x_{11})$$
$$(x_5' + x_6')(x_1 + x_7 + x_{12}')(x_1 + x_8)(x_7' + x_8' + x_{13}')(y_1 + z_1)(y_2 + z_2)$$

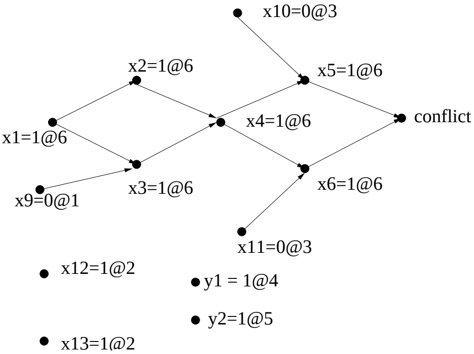




$$(x_1' + x_2)(x_1' + x_3 + x_9)(x_2' + x_3' + x_4)(x_4' + x_5 + x_{10})(x_4' + x_6 + x_{11})$$
$$(x_5' + x_6')(x_1 + x_7 + x_{12}')(x_1 + x_8)(x_7' + x_8' + x_{13}')(y_1 + z_1)(y_2 + z_2)$$

Conflict: $(x_9' \wedge x_{12} \wedge x_{13} \wedge x_{10}' \wedge x_{11}' \wedge y_1 \wedge y_2 \wedge x_1) \implies \text{FALSE}$

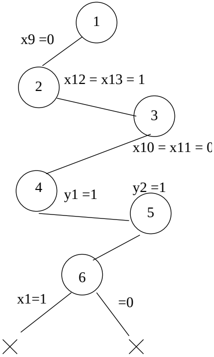


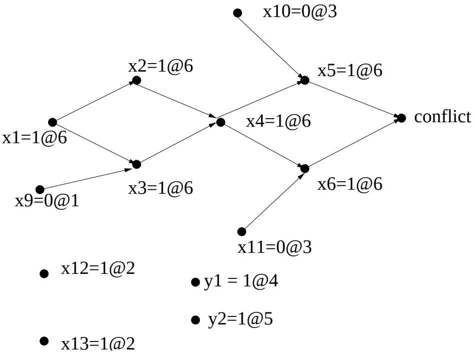


CDCL & Non-Chronological Backtracking [From GRASP]

$$(x_1' + x_2)(x_1' + x_3 + x_9)(x_2' + x_3' + x_4)(x_4' + x_5 + x_{10})(x_4' + x_6 + x_{11})$$
$$(x_5' + x_6')(x_1 + x_7 + x_{12}')(x_1 + x_8)(x_7' + x_8' + x_{13}')(y_1 + z_1)(y_2 + z_2)$$

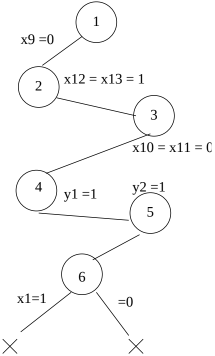
Is the learnt Clause $= (x_9 \vee x_{12}' \vee x_{13}' \vee x_{10} \vee x_{11} \vee y_1' \vee y_2' \vee x_1')$?

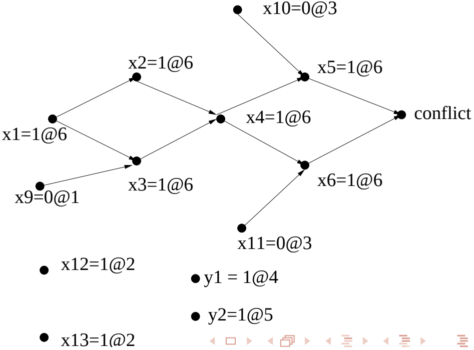




CDCL: Analyze the cause of conflict

- From the conflict-node in the implication graph, traverse back to *antecedents* (or root nodes x_1, x_9, x_{10}, x_{11})
- Note than x_{12}, x_{13}, y_1, y_2 are *unreachable*
- Conflict clause can be simplified:
  - From $(x_9 \vee x_{12}' \vee x_{13}' \vee x_{10} \vee x_{11} \vee y_1' \vee y_2' \vee x_1')$
  - To $(x_9 \vee x_{10} \vee x_{11} \vee x_1')$

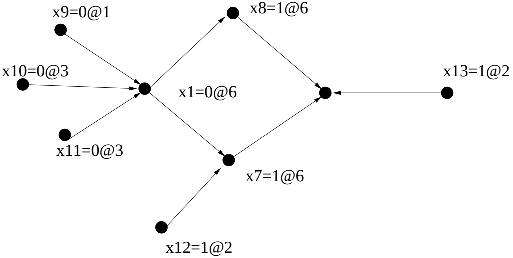




- Add learnt clause to original CNF
- Chronological backtrack: revert last assignment from $x_1 = 1$ to $x_1 = 0$

$$(x_1' + x_2)(x_1' + x_3 + x_9)(x_2' + x_3' + x_4)(x_4' + x_5 + x_{10})(x_4' + x_6 + x_{11})$$
$$(x_5' + x_6')(x_1 + x_7 + x_{12}')(x_1 + x_8)(x_7' + x_8' + x_{13}')(y_1 + z_1)(y_2 + z_2)$$

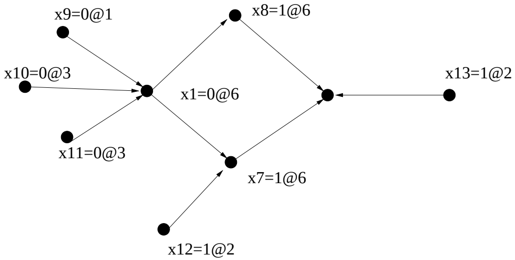
Assignment on Learnt Clause: $(x_9 \vee x_{10} \vee x_{11} \vee x_1')$



- Add learnt clause to original CNF
- Chronological backtrack: revert last assignment from $x_1 = 1$ to $x_1 = 0$

$$(x_1' + x_2)(x_1' + x_3 + x_9)(x_2' + x_3' + x_4)(x_4' + x_5 + x_{10})(x_4' + x_6 + x_{11})$$
$$(x_5' + x_6')(x_1 + x_7 + x_{12}')(x_1 + x_8)(x_7' + x_8' + x_{13}')(y_1 + z_1)(y_2 + z_2)$$

Assignment on Learnt Clause: $(x_9 \vee x_{10} \vee x_{11} \vee x_1')$



$x_1 = 0$ also leads to a conflict. Learn new clause?

Conflict-Driven Clause Learning (CDCL) solvers

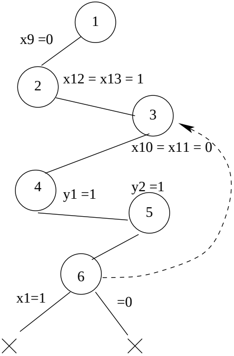
$$(x_1' + x_2)(x_1' + x_3 + x_9)(x_2' + x_3' + x_4)(x_4' + x_5 + x_{10})(x_4' + x_6 + x_{11})$$
$$(x_5' + x_6')(x_1 + x_7 + x_{12}')(x_1 + x_8)(x_7' + x_8' + x_{13}')(y_1 + z_1)(y_2 + z_2)$$

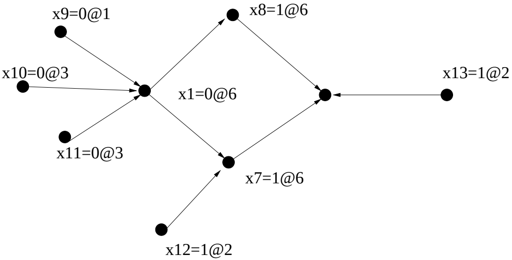
First learnt/conflict clause CC_1: $(x_9 \vee x_{10} \vee x_{11} \vee x_1')$

- New conflict clause also derived from implication graph
- CC_2: $(x_9 \vee x_{12}' \vee x_{13}' \vee x_{10} \vee x_{11})$
- Decision on x_1, y_1, y_2 does not affect the CNF SAT!
- Non-Chronological backtrack:
  - To the MAX decision-level in the conflict clause!
  - Backtrack to Decision-Level 3, undo x_{10} or x_{11}

CDCL search space pruning

CC_2: $(x_9 \vee x'_{12} \vee x'_{13} \vee x_{10} \vee x_{11})$





Conflict-Driven Clause Learning (CDCL) solvers

- Recent techniques can identify more conflict clauses
- Identify unique implication points (UIPs)
- Decision heuristics: Branch on high-activity literals [GRASP]
  - Activity: A score for every literal
  - The number of occurrences of a literal in the formula
- As conflict clauses are added, activity changes
- After n conflicts, multiply activity by $f < 1$, or *rescore*
  - VSIDS heuristic: Variable State Independent Decaying Sum [CHAFF]

A List of CDCL SAT solvers

- GRASP, circa 1996, from Silva and Sakallah
- zCHAFF 2001, from Princeton, Prof. Sharad Malik
- BerkMin 2002
- MiniSAT, 2004 (?) from Cadence Berkeley Labs
- PicoSAT and Lingeling, from Prof. Armin Biere, Univ. Linz
- Please visit `www.satisfiability.org`

Extract UNSAT Cores from UNSAT CNF

- CNF: $\mathcal{F} = (a' + b')(a' + b)(a + b')(a + b)(x + y)(y + z)$
  - Note that $\mathcal{F}$ is UNSAT
  - Identify a minimum number of clauses that make $\mathcal{F}$ UNSAT
  - This subset of clauses is the *UNSAT Core*, or MIN-UNSAT
  - Helps to identify the causes for UNSAT
- $(a' + b')(a' + b)(a + b')(a + b)$ is the UNSAT core in $\mathcal{F}$
- UNSAT core may not be unique
- UNSAT cores have many applications in verification
- Study of UNSAT cores and applications: Good class project option!

- Where does SAT fail?
- For hard UNSAT instances, such as equivalence verification

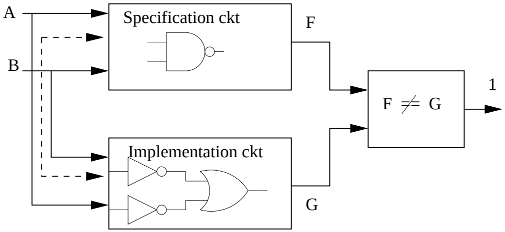


Figure: Miter the circuits F, G

- Prove UNSAT, or find a counter-example
- Limitations: No internal structural equivalences
- EDA-techniques: Circuit-SAT, AIG-reductions, constraint-learning

What Next?

How to improve SAT for Circuit Equivalence Verification?

AND-INVERT-GRAPH (AIG) based Reductions!