Advanced Software Testing and Debugging (CS598)

Formal Methods Basics

Fall 2020

Lingming Zhang

Some slides borrowed from UW CSE 507: Computer-Aided Reasoning for Software

Topics

- Propositional logic review
- Boolean satisfiability problem (SAT)
- Satisfiability Modulo Theories (SMT)

Topics

- Propositional logic review
- Boolean satisfiability problem (SAT)
- Satisfiability Modulo Theories (SMT)

Syntax of propositional logic

$$(\neg \boldsymbol{p} \wedge \top) \vee (\boldsymbol{q} \rightarrow \bot)$$

Atom truth symbols**:** ⊤ **("true"),** ⊥ **("false")**

propositional variables: ***p, q, r***

Literal an atom **α** or its negation **¬α**

Formula an atom or the application of a **logical connective** to formulas $\boldsymbol{F_1}$, $\boldsymbol{F_2}$:

$\neg F_1$	"not"	(negation)
$F_1 \wedge F_2$	"and"	(conjunction)
$F_1 \vee F_2$	"or"	(disjunction)
$F_1 \longrightarrow F_2$	"implies"	(implication)
$F_1 \longleftrightarrow F_2$	"if and only if"	(iff)

Semantics of propositional logic: interpretations

- An **interpretation *I*** for a propositional formula ***F*** maps every variable in ***F*** to a truth value:

$$I :\{ p \mapsto \text{true}, q \mapsto \text{false}, \ldots\}$$

- ***I*** is a **satisfying interpretation** of ***F***, written as $I \models F$, if ***F*** evaluates to true under ***I***
  - A satisfying interpretation is also called a **model**
- ***I*** is a **falsifying interpretation** of ***F***, written as $I \not\models F$, if ***F*** evaluates to false under ***I***

Semantics of propositional logic: definition

- **Base cases:**

$I \models \top$	
$I \not\models \bot$	
$I \models p$	iff $I[p] \models \text{true}$
$I \not\models p$	iff $I[p] \models \text{false}$

- **Inductive cases:**

$I \models \neg F$	iff $I \not\models F$
$I \models F_1 \wedge F_2$	iff $I \models F_1$ and $I \models F_2$
$I \models F_1 \vee F_2$	iff $I \models F_1$ or $I \models F_2$
$I \models F_1 \longrightarrow F_2$	iff $I \not\models F_1$ or $I \models F_2$
$I \models F_1 \longleftrightarrow F_2$	iff $I \models F_1$ and $I \models F_2$, OR $I \not\models F_1$ and $I \not\models F_2$

Semantics of propositional logic: example

F: $(p \wedge q) \rightarrow (p \vee \neg q)$
I: $\{p \mapsto \text{true}, q \mapsto \text{false}\}$

$\boldsymbol{I} \models \boldsymbol{F}$

Topics

- Propositional logic review
- Boolean satisfiability problem (SAT)
- Satisfiability Modulo Theories (SMT)

Satisfiability & validity of propositional formulas

- ***F*** is satisfiable iff $I \models F$ for some ***I***
- ***F*** is valid iff $I \models F$ for all ***I***
- **Duality** of satisfiability and validity:
  - ***F*** is valid iff $\neg F$ is unsatisfiable.

Techniques for deciding satisfiability & validity

SAT Solver

Search

Enumerate all interpretations (i.e., build a truth table), and check that they satisfy the formula.

Deduction

Assume the formula is invalid, apply proof rules, and check for contradiction in every branch of the proof tree.

Proof by search: enumerating interpretations

- **F**: $(p \wedge q) \rightarrow (p \vee \neg q)$

p	q	$p \wedge q$	$\neg q$	$p \vee \neg q$	F
0	0	0	1	1	1
0	1	0	0	0	1
1	0	0	1	1	1
1	1	1	0	1	1

Proof by deduction: semantic arguments

$$\frac{I \models \neg F}{I \not\models F}$$

Premise

Conclution

$$\frac{I \not\models \neg F}{I \models F}$$

$$\frac{I \models F_1 \wedge F_2}{I \models F_1, I \models F_2}$$

$$\frac{I \not\models F_1 \wedge F_2}{I \not\models F_1 \mid I \not\models F_2}$$

$$\frac{I \models F_1 \vee F_2}{I \models F_1 \mid I \models F_2}$$

$$\frac{I \not\models F_1 \vee F_2}{I \not\models F_1, I \not\models F_2}$$

$$\frac{I \models F_1 \rightarrow F_2}{I \not\models F_1 \mid I \models F_2}$$

$$\frac{I \not\models F_1 \rightarrow F_2}{I \models F_1, I \not\models F_2}$$

$$\frac{I \models F_1 \leftrightarrow F_2}{I \not\models F_1 \vee F_2 \mid I \models F_1 \wedge F_2}$$

$$\frac{I \not\models F_1 \leftrightarrow F_2}{I \models \neg F_1 \wedge F_2 \mid I \models F_1 \wedge \neg F_2}$$

- A **proof rule** consists of
  - **Premise**: facts that have to hold to apply the rule
  - **Conclusion**: facts derived from applying the rule
- Where
  - **Commas** indicate derivation of multiple facts
  - **Pipes** indicate alternative facts (branches in the proof)

Proof by deduction: semantic arguments

$$\frac{I \models \neg F}{I \not\models F} \qquad \frac{I \not\models \neg F}{I \models F}$$

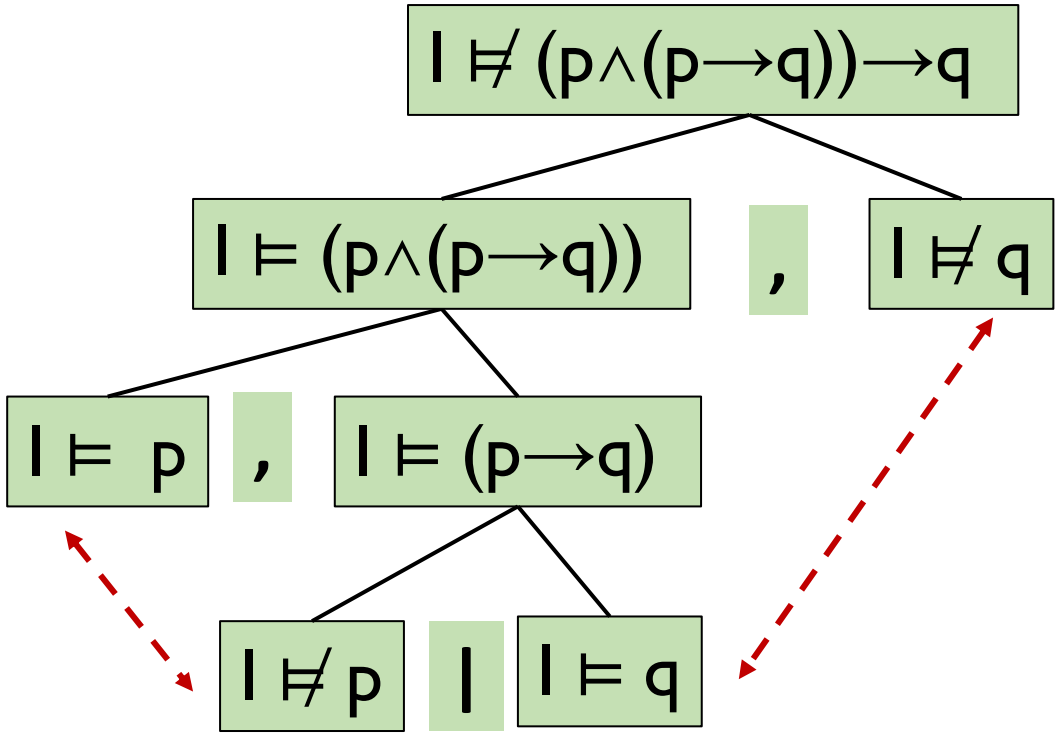
$$\frac{I \models F_1 \wedge F_2}{I \models F_1,\ I \models F_2} \qquad \frac{I \not\models F_1 \wedge F_2}{I \not\models F_1 \mid I \not\models F_2}$$

$$\frac{I \models F_1 \vee F_2}{I \models F_1 \mid I \models F_2} \qquad \frac{I \not\models F_1 \vee F_2}{I \not\models F_1,\ I \not\models F_2}$$

$$\frac{I \models F_1 \rightarrow F_2}{I \not\models F_1 \mid I \models F_2} \qquad \frac{I \not\models F_1 \rightarrow F_2}{I \models F_1,\ I \not\models F_2}$$

$$\frac{I \models F_1 \leftrightarrow F_2}{I \not\models F_1 \vee F_2 \mid I \models F_1 \wedge F_2} \qquad \frac{I \not\models F_1 \leftrightarrow F_2}{I \models \neg F_1 \wedge F_2 \mid I \models F_1 \wedge \neg F_2}$$

Proving $(p \wedge (p \rightarrow q)) \rightarrow q$:



Contradiction!

So the formula is valid.

Getting ready for SAT solving with normal forms

- Arbitrary formula can be hard to solve!
- **Normal form:** a syntactic restriction such that every formula in the logic has an equivalent formula in the normal form
- Three important normal forms for propositional logic:
  - Negation Normal Form (NNF)
  - Disjunctive Normal Form (DNF)
  - Conjunctive Normal Form (CNF)

Negation Normal Form (NNF)

Atom := Variable | $\top$ | $\bot$
Literal := Atom | $\neg$Atom
Formula := Literal | Formula op Formula
op := $\wedge$ | $\vee$

- The only allowed connectives are $\wedge$, $\vee$, and $\neg$.
- $\neg$ can appear only in literals

Conversion to NNF performed using **De Morgan's Laws**:

$\neg(F \wedge G) \Leftrightarrow \neg F \vee \neg G$ $\qquad$ $\neg(F \vee G) \Leftrightarrow \neg F \wedge \neg G$

Disjunctive Normal Form (DNF)

Atom := Variable | $\top$ | $\bot$
Literal := Atom | $\neg$Atom
Formula := Clause $\vee$ Formula
Clause := Literal | Literal $\wedge$ Clause

- Disjunction of conjunction of literals
- Deciding satisfiability of a DNF formula is trivial
- However, may incur exponential increase in formula size

To convert to DNF, convert to NNF and distribute $\wedge$ over $\vee$:

$$(F\wedge(G\vee H)) \Leftrightarrow (F\wedge G)\vee(F\wedge H)$$

$$((G\vee H)\wedge F) \Leftrightarrow (G\wedge F)\vee(H\wedge F)$$

Conjunctive Normal Form (CNF)

Atom := Variable | $\top$ | $\bot$
Literal := Atom | $\neg$Atom
Formula := Clause $\wedge$ Formula
Clause := Literal | Literal $\vee$ Clause

- Conjunction of disjunction of literals
- Deciding the satisfiability of a CNF formula is hard
- **SAT solvers use CNF as their input language**
  - Linear increase in formula size

To convert to CNF, convert to NNF and distribute $\vee$ over $\wedge$:

$$(F \vee (G \wedge H)) \Leftrightarrow (F \vee G) \wedge (F \vee H)$$

$$((G \wedge H) \vee F) \Leftrightarrow (G \vee F) \wedge (H \vee F)$$

Propositional formula to CNF: Tseitin's transformation

- Key idea: introduce **auxiliary variables** to represent the output of subformulas, and constrain those variables using CNF clauses

$x \rightarrow (y \wedge z)$

$a1$
$a1 \leftrightarrow (x \rightarrow a2)$
$a2 \leftrightarrow (y \wedge z)$

$a1$
$a1 \rightarrow (x \rightarrow a2)$
$(x \rightarrow a2) \rightarrow a1$
$a2 \leftrightarrow (y \wedge z)$

$a1$
$\neg a1 \vee (\neg x \vee a2)$
$(x \rightarrow a2) \rightarrow a1$
$a2 \leftrightarrow (y \wedge z)$

$a1$
$\neg a1 \vee (\neg x \vee a2)$
$(x \wedge \neg a2) \vee a1$
$a2 \leftrightarrow (y \wedge z)$

$a1$
$\neg a1 \vee (\neg x \vee a2)$
$x \vee a1$
$\neg a2 \vee a1$
$a2 \leftrightarrow (y \wedge z)$

$a1$
$\neg a1 \vee (\neg x \vee a2)$
$x \vee a1$
$\neg a2 \vee a1$
$\neg a2 \vee y$
$\neg a2 \vee z$
$\neg y \vee \neg z \vee a2$

Solving CNF: Proof by resolution

Resolution rule

$$\frac{a_1 \vee .. \vee an \vee \boldsymbol{\beta} \qquad b_1 \vee .. \vee b_m \vee \neg\boldsymbol{\beta}}{a_1 \vee .. \vee an \vee b_1 \vee .. \vee bm}$$

Unit resolution rule

$$\frac{\boldsymbol{\beta} \qquad b_1 \vee .. \vee b_m \vee \neg\boldsymbol{\beta}}{b_1 \vee .. \vee bm}$$

- Proving that a CNF formula is valid can be done using just this one proof rule!
- Apply the rule until a contradiction (empty clause) is derived, or no more applications are possible

- Unit resolution specializes the resolution rule to the case where one of the clauses is **unit** (a single literal)

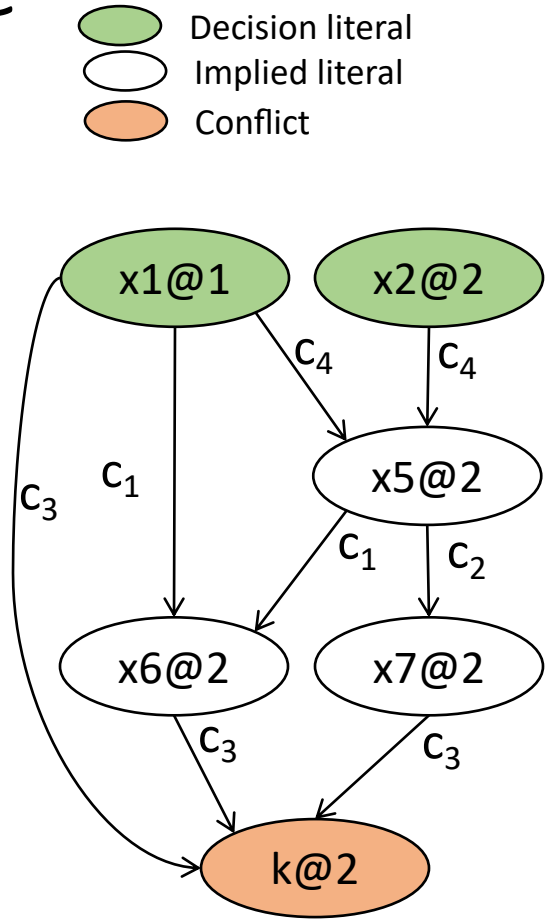
A basic solver: Davis-Putnam-Logemann-Loveland (DPLL, 1962)

```
// Returns true if the CNF formula F is satisfiable; otherwise returns false.
DPLL(F):
G ← BCP(F)
if G = ⊤ then return true
if G = ⊥ then return false
p ← choose(vars(G))
return DPLL(G{p ↦ ⊤}) ||
       DPLL(G{p ↦ ⊥})
```

- **Boolean Constraint Propagation** (BCP) applies unit resolution until fixed point
- If BCP cannot reduce ***F*** to a constant, we choose an unassigned variable and recurse assuming that the variable is either true or false
- If the formula is satisfiable under either assumption, then it has a satisfying assignment (expressed in assumptions). Otherwise, it's unsatisfiable.

DPLL: example

c_1: $\neg x_1 \vee \neg x_5 \vee x_6$
c_2: $\neg x_5 \vee x_7$
c_3: $\neg x_1 \vee \neg x_6 \vee \neg x_7$
c_4: $\neg x_1 \vee \neg x_2 \vee x_5$
c_5: $\neg x_1 \vee \neg x_3 \vee x_5$
c_6: $\neg x_1 \neg x_4 \vee x_5$

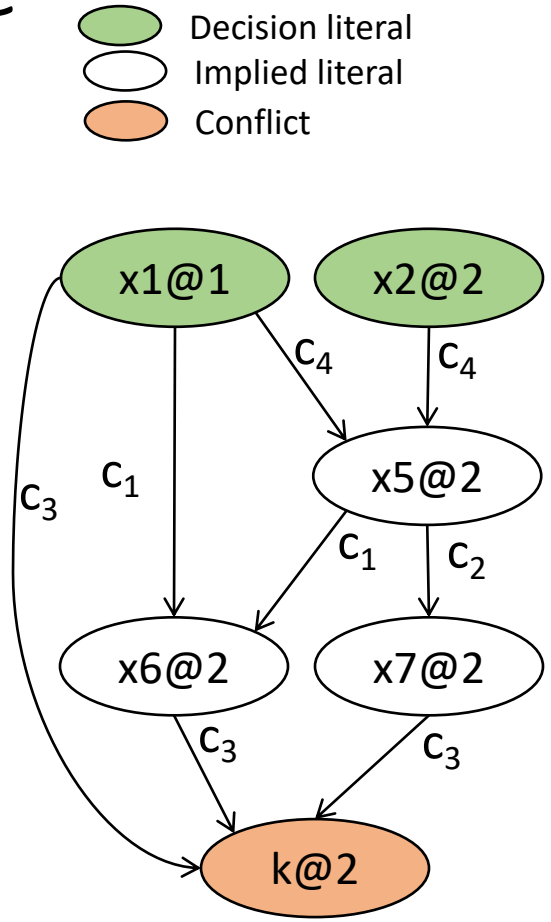


Implication graph

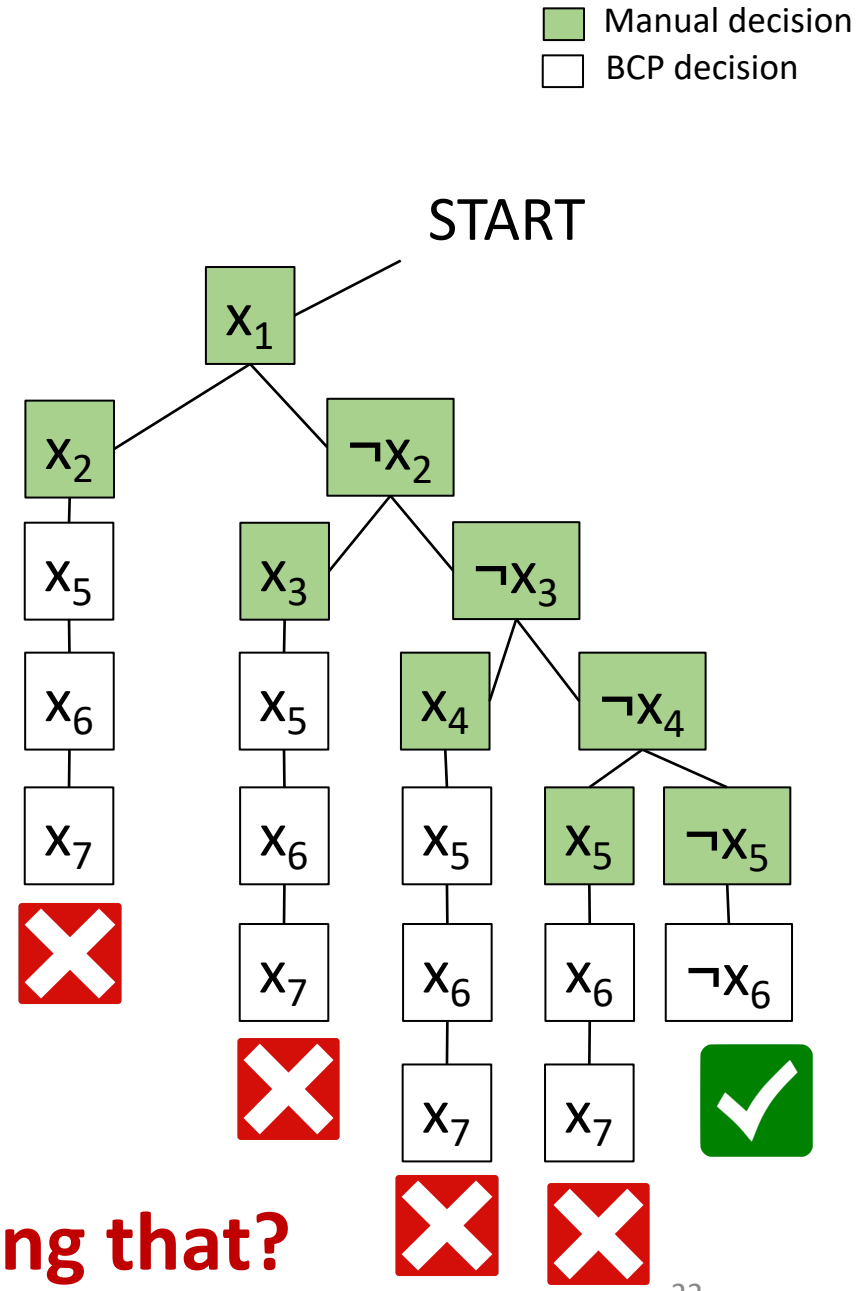
- An **implication graph G = (V, E)** is a DAG recording the history of decisions and the resulting BCP deductions
  - **v∈V** is a literal and the decision level it got decided
  - **⟨v, w⟩ ∈ E** iff **v ≠ w, ¬v ∈ antecedent(w)**, and **⟨v, w⟩** is labeled with **antecedent(w)**
  - **antecedent(v):** the clause from which **v** got decided
- A unit clause **c** is the antecedent of its sole unassigned literal

DPLL: example

c_1: $\neg x_1 \vee \neg x_5 \vee x_6$
c_2: $\neg x_5 \vee x_7$
c_3: $\neg x_1 \vee \neg x_6 \vee \neg x_7$
c_4: $\neg x_1 \vee \neg x_2 \vee x_5$
c_5: $\neg x_1 \vee \neg x_3 \vee x_5$
c_6: $\neg x_1 \neg x_4 \vee x_5$



Implication graph



Decision tree

Can we learn from conflicts and avoid repeating that?

Conflict-Driven Clause Learning (CDCL)

```
ANALYZECONFLICT():
d ← level(conflict)
if d=0 then return -1
c ← antecedent(conflict)
while !oneLitAtLevel(c, d)
    t ← lastAssignedLitAtLevel(c, d)
    v ← varOfLit(t)
    a ← antecedent(t)
    c ← resolve(a, c, v)
b ← assertingLevel(c)
return ⟨b, c⟩
```

Start from the direct antecedent for conflict, traverse back until there is only one literal decided/implied at the current (highest) decision level in **c**

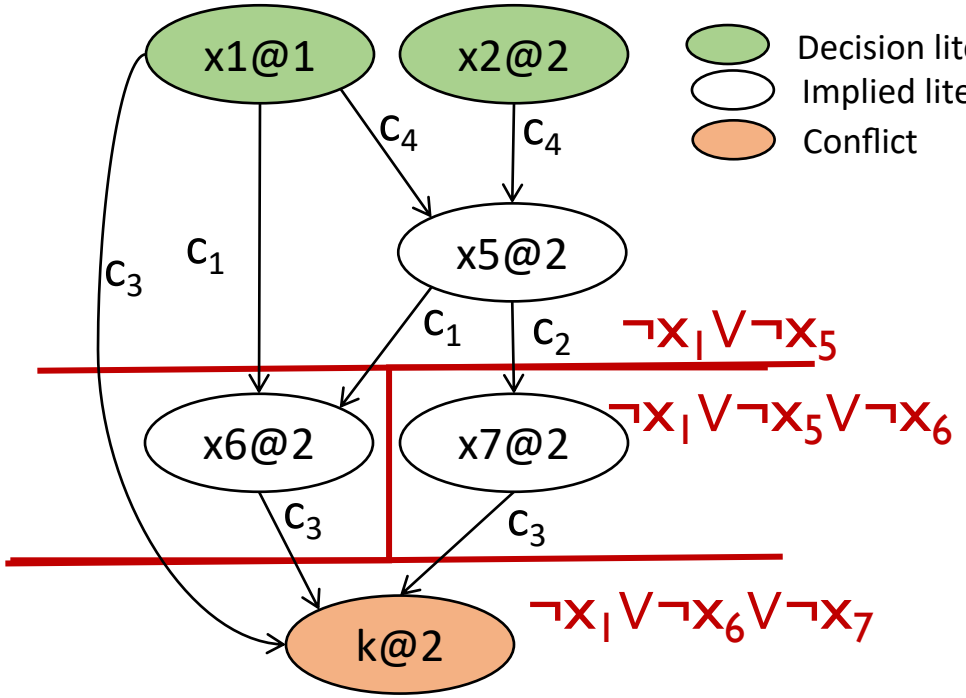
Apply resolution rule to **a** and **c** with respect to variable **v**

Backtrack to the second highest decision level in the newly derived constraint **c**

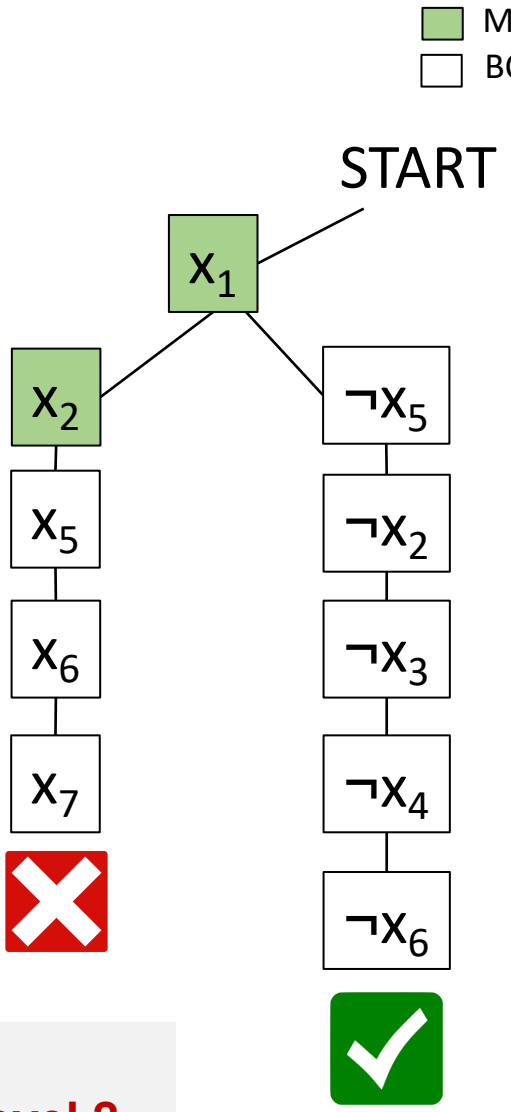
- Backtrack to level **b**
- Add **c** into the original formula

CDCL: example

c_1: $\neg x_1 \vee \neg x_5 \vee x_6$
c_2: $\neg x_5 \vee x_7$
c_3: $\neg x_1 \vee \neg x_6 \vee \neg x_7$
c_4: $\neg x_1 \vee \neg x_2 \vee x_5$
c_5: $\neg x_1 \vee \neg x_3 \vee x_5$
c_6: $\neg x_1 \vee \neg x_4 \vee x_5$
c_7: $\neg x_1 \vee \neg x_5$



Implication graph



Decision tree

t ← lastAssignedLitAtLevel(c, d)
v ← varOfLit(t)
a ← antecedent(t)
c ← resolve(a, c, v)

c: c_3
t: x_7
v: x_7
a: c_2

→

c: $\neg x_1 \vee \neg x_5 \vee \neg x_6$
t: x_6
v: x_6
a: c_1

→

c: $\neg x_1 \vee \neg x_5$
Only x_5 at level 2
Done!

Topics

- Propositional logic review
- Boolean satisfiability problem (SAT)
- Satisfiability Modulo Theories (SMT)

Satisfiability Modulo Theories (SMT)

- Some problems are more naturally expressed in other logics than propositional logic, e.g.:
  - Software verification needs reasoning about equality, arithmetic, data structures, ...
- SMT consists in deciding the satisfiability of a (quantifier-free) first-order formula with respect to a background theory
- Example:
  - Equality with Uninterpreted Functions (EUF)

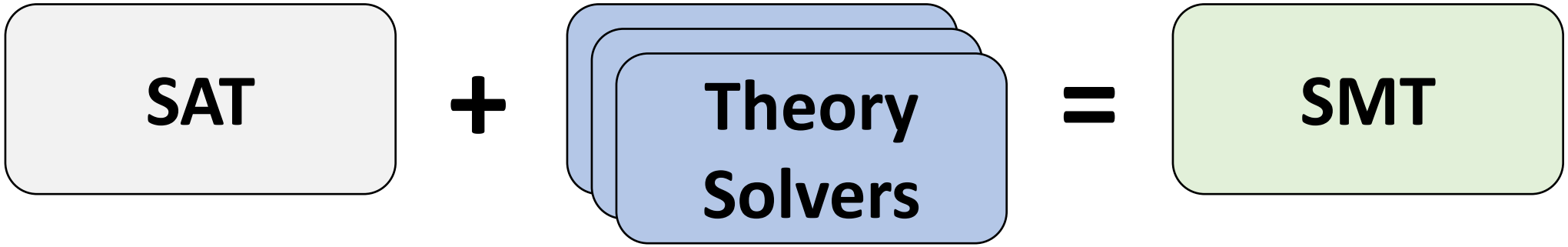
$$g(a)=c \land (f(g(a))\neq f(c) \lor g(a)=d) \land c \neq d$$

Syntax of first-order logic (FOL)

- Logical symbols
  - Connectives: **¬,∧,∨, →, ↔**
  - Parentheses: **(,)**
  - ~~Quantifiers: **∃, ∀**~~
- Non-logical symbols
  - Constants: **x, y, z**
  - N-ary functions: **f(x), x+y**
  - N-ary predicates: **p(x), x>y**
  - ~~Variables: **u, v, w**~~

Usually only consider quantifier-free ground formulas

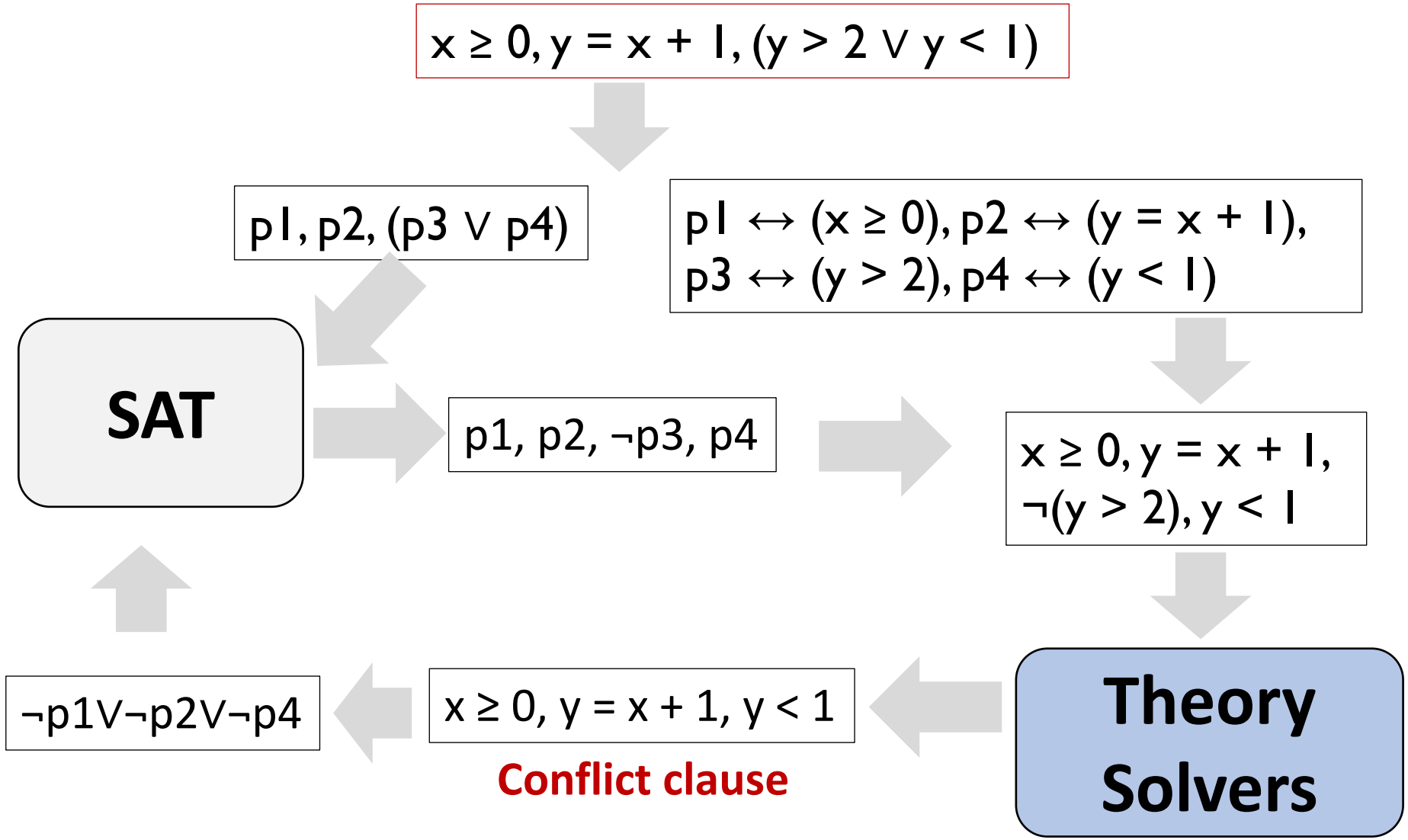
SMT: basic architecture



- Equality + UF
- Arithmetic
- Bit-vectors

…

SMT: basic idea



Common theories

- Equality (and uninterpreted functions)
  - $x = g(y)$
- Fixed-width bitvectors
  - $(b >> 1) = c$
- Linear arithmetic (over R and Z)
  - $2x + y \leq 5$
- Arrays
  - $a[i] = x$

Theories of linear integer and real arithmetic

- Signature
  - Integers (or reals)
  - Arithmetic operations: multiplication by an integer (or real) number, **+, -.**
  - Predicates: **=, ≤**.
  - Expanded with all constant symbols: **x, y, z, ...**
- Deciding TLIA and TLRA
  - Polynomial time for linear real arithmetic (LRA)
  - NP-complete for linear integer arithmetic (LIA)

LIA example: compiler optimization

```
for (i=1; i<=10; i++) {
  a[j+i] = a[j];
}
```

A LIA formula that is unsatisfiable iff this transformation is valid:

$(i \geq 1) \wedge (i \leq 10) \wedge (j + i = j)$

```
int v = a[j];
for (i=1; i<=10; i++) {
  a[j+i] = v;
}
```

Theory of arrays

- Signature
  - Array operations: **read, write**
  - Equality: **=**
  - Expanded with all constant symbols: **x, y, z, ...**
- Axioms
  - **∀a, i, v. read(write(a, i, v), i) = v**
  - **∀a, i, j, v. ¬(i = j) → (read(write(a, i, v), j) = read(a, j))**
  - **∀a, b. (∀i. read(a, i) = read(b, i)) → a = b**
- Deciding T_A
  - Satisfiability problem: NP-complete
  - Used in many software verification tools to model memory

SMT tools

- Z3: https://github.com/Z3Prover/z3
  - **Supported theories:** empty theory, linear arithmetic, nonlinear arithmetic, bitvectors, arrays, datatypes, quantifiers, strings
- CVC4: https://cvc4.github.io/
  - **Supported theories:** rational and integer linear arithmetic, arrays, tuples, records, inductive data types, bitvectors, strings, and equality over uninterpreted function symbols
- STP: https://github.com/stp/stp
  - **Supported theories:** bitvectors, arrays
- Boolector: https://github.com/Boolector/boolector
  - **Supported theories:** bitvectors, arrays, and uninterpreted functions
- …

Further readings

- https://rise4fun.com/z3/tutorial
- https://www.cs.princeton.edu/~zkincaid/courses/fall18/readings/SATHandbook-CDCL.pdf
- http://satsmt2013.ics.aalto.fi/slides/Marques-Silva.pdf
- https://cse442-17f.github.io/Conflict-Driven-Clause-Learning/
- https://homes.cs.washington.edu/~emina/blog/2017-06-23-a-primer-on-sat.html

Thanks and stay safe!