Software Analysis & Verification

Ruzica Piskac

Fall 2025

DPLL Procedure

Davis Putnam Logemann Loveland

Review: Syntax of Propositional Logic (PL)

Truth_symbol ::= ⊤(true), ⊥ (false)

variable ::= p, q, r,…

atom ::= truth_symbol | variable

literal ::= atom | ¬atom

formula ::=literal |
¬formula |
formula ∧ formula |
formula ∨ formula |
formula → formula |
formula ↔ formula

Terminology

- Literal ℓ
  - a propositional variable P or its negation $\neg$P
- Clause C
  - a disjunction of literals $\ell_1 \vee \ldots \vee \ell_n$
  - if n=1, C is called a *unit clause*
- Conjunctive (or clausal) normal form (CNF) F
  - a formula that is a conjunction of clauses $C_1 \wedge \ldots \wedge C_m$

Semantics of Propositional Logic

- The meaning (value) of $\top$ is always *True*. The meaning of $\bot$ is always *False*.
- The meaning of the other formulas depends on the meaning of the propositional variables.
  - Base cases: Truth Tables

P	Q	$\neg$P	P $\wedge$ Q	P $\vee$ Q	P $\rightarrow$ Q	P $\leftrightarrow$ Q
False	*False*	*True*	*False*	*False*	*True*	*True*
False	*True*	*True*	*False*	*True*	*True*	*False*
True	*False*	*False*	*False*	*True*	*False*	*False*
True	*True*	*False*	*True*	*True*	*True*	*True*

  - Non-base cases: Given by reduction to the base cases
    Example: the meaning of (P $\vee$ Q) $\wedge$ R is the same as the meaning of A $\wedge$ R where A has the same meaning as P $\vee$ Q.

Semantics of Propositional Logic

- An assignment of Boolean values to the propositional variables of a formula is an interpretation of the formula.

P	Q	P ∨ Q	(P ∨ Q) ∧ ¬Q	(P ∨ Q) ∧ ¬Q → P
False	*False*	*False*	*False*	*True*
False	*True*	*True*	*False*	*True*
True	*False*	*True*	*True*	*True*
True	*True*	*True*	*False*	*True*

- Interpretations:
  {P ↦ *False*, Q ↦ *False*}, {P ↦ *False*, Q ↦ *True*}, . . .

- The semantics of Propositional logic is compositional: the meaning of a formula is defined recursively in terms of the meaning of the formula's components.

Semantics of Propositional Logic

- An assignment of Boolean values to the propositional variables of a formula is an interpretation of the formula.

P	Q	P ∨ Q	(P ∨ Q) ∧ ¬Q	(P ∨ Q) ∧ ¬Q → P
False	*False*	*False*	*False*	*True*
False	*True*	*True*	*False*	*True*
True	*False*	*True*	*True*	*True*
True	*True*	*True*	*False*	*True*

- A formula is
  - (un)satisfiable if it is true in some (no) interpretation,
  - valid if it is true in every possible interpretation
- A formula that is valid or unsatisfiable is called tautology

The SAT Problem

- The satisfiability problem for propositional logic (SAT) asks whether a given formula F is satisfiable.
- SAT is decidable.
- Hence, so is validity of propositional formulas.
- However, SAT is NP-complete
- Hence, checking validity is co-NP-complete.

The SAT Problem

- Many problems in formal verification can be reduced to checking the satisfiability of a formula in some logic.
- In practice, NP-completeness means the time needed to solve a SAT problem grows exponentially with the number of propositional variables in the formula.
- Despite NP-completeness, many realistic instances (in the order of 100,000 variables) can be checked very efficiently by state-of-the-art SAT solvers.

Computer generated math proof is largest ever at 200 terabytes

May 30, 2016 by Bob Yirka report

Credit: Victorgrigas/Wikideia/ CC BY-SA 3.0

(Phys.org)—A trio of researchers has solved a single math problem by using a supercomputer to grind through over a trillion color combination possibilities, and in the process has generated the largest math proof ever—the text of it is 200 terabytes in size. In their paper uploaded to the preprint server *arXiv*, Marijn Heule with the University of Texas, Oliver Kullmann with Swansea University and Victor Marek with the University of Kentucky outline the math problem, the means by which a supercomputer was programmed to solve it, and the answer which the proof was asked to provide.

- Two-Hundred-Terabyte Maths Proof Is Largest Ever

Decision Procedures for SAT

- Naive model enumeration
  - guess candidate model and check
  - what if formula is unsatisfiable?
- Naive logical deduction
  - draw consequences from given facts to obtain contradiction
  - what if formula is satisfiable?
- Modern SAT solvers
  - combine enumeration and deduction in a clever way
    - DPLL algorithm: Davis, Putnam, Logemann, and Loveland (1962)
  - operate on formulas in Conjunctive Normal Form (CNF)
- Some alternatives to DPLL
  - (Ordered) Binary Decision Diagrams (BDDs)
  - Stalmarcks method
  - local stochastic search

Conjuctive Normal Form

- $\varphi \leftrightarrow \varphi'$ $\Rightarrow_{CNF}$ $\varphi \rightarrow \varphi' \wedge \varphi' \rightarrow \varphi$
- $\varphi \rightarrow \varphi'$ $\Rightarrow_{CNF}$ $\neg\varphi \vee \varphi'$
- $\neg(\varphi \vee \varphi')$ $\Rightarrow_{CNF}$ $\neg\varphi \wedge \neg\varphi'$
- $\neg(\varphi \wedge \varphi')$ $\Rightarrow_{CNF}$ $\neg\varphi \vee \neg\varphi'$
- $\neg\neg\varphi$ $\Rightarrow_{CNF}$ φ
- $(\varphi \wedge \varphi') \vee \varphi''$ $\Rightarrow_{CNF}$ $(\varphi \vee \varphi'') \wedge (\varphi' \vee \varphi'')$

- PROBLEM: (potential) exponential blowup of the resulting formula

The original DPLL procedure

- Tries to build incrementally a satisfying truth assignment M for a CNF formula F
- M is grown by
  - deducing the truth value of a literal from M and F, or
  - guessing a truth value
- If a wrong guess for a literal leads to an inconsistency, the procedure backtracks and tries the opposite value

The Original DPLL Procedure – Example

Comment:

To simplify notation, we denote formula

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

with

$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

The Original DPLL Procedure – Example

assign	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,$	1

Deduce 1

1	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,$	$\neg 1 \vee \neg 2,$	$\neg 1 \vee \neg 3 \vee \neg 4,\ 1$

Deduce $\neg 2$

$1, 2$	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

Guess 3

$1, 2, 3$	$1 \vee 2,$	$2 \vee \neg 3 \vee 4,$	$\neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

Deduce 4

$1, 2, 3, 4$	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

Conflict

The Original DPLL Procedure – Example

assign	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,$	1

Deduce 1

1	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,$	$\neg 1 \vee \neg 2,$	$\neg 1 \vee \neg 3 \vee \neg 4,\ 1$

Deduce $\neg 2$

$1, 2$	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

Guess 3

$1, 2, 3$	$1 \vee 2,$	$2 \vee \neg 3 \vee 4,$	$\neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

Deduce 4

$1, 2, 3, 4$	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

Undo 3

The Original DPLL Procedure – Example

assign	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

Deduce 1

1	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

Deduce $\neg 2$

1, 2	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

Deduce $\neg 3$

1, 2, 3	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

Model Found

An Abstract Framework for DPLL

- The DPLL procedure can be described declaratively by simple sequent-style calculi
- Such calculi however cannot model meta-logical features such as backtracking, learning and restarts
- We model DPLL and its enhancements as transition systems instead
- A transition system is a binary relation over states, induced by a set of conditional transition rules

An Abstract Framework for DPLL

- State
  - **fail** or M ‖ F
  - where
    - F is a CNF formula, a set of clauses, and
    - M is a sequence of annotated literals denoting a partial truth assignment
- Initial State
  - ∅ ‖ F, where F is to be checked for satisfiability
- Expected final states:
  - **fail** if F is unsatisfiable
  - M ‖ G
    where
    - M is a model of G
    - G is logically equivalent to F

Transition Rules for the Original DPLL

- Extending the assignment:

UnitProp $M \parallel F, C \vee l \rightarrow M\ l \parallel F, C \vee l$

- $M \vDash \neg C$
- l is undefined in M

Decide $M \parallel F, C \rightarrow M\ l^d \parallel F, C$

- l or $\neg l$ occur in C
- l is undefined in M

Notation: l^d is a decision literal

Transition Rules for the Original DPLL

- Repairing the assignment:

Fail $\quad M \parallel F, C \rightarrow \text{fail}$

- $M \models \neg C$
- M does not contain decision literals

Backtrack $\quad M\ l^d\ N \parallel F, C \rightarrow M\ \neg l \parallel F, C$

- $M\ l^d\ N \models \neg C$
- l is the last decision literal

Transition Rules DPLL – Example

$\varnothing \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg\, 1 \vee \neg\, 2,\ \neg\, 1 \vee \neg\, 3 \vee \neg\, 4,\ 1$

UnitProp 1

$1 \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg\, 1 \vee \neg\, 2,\ \neg\, 1 \vee \neg\, 3 \vee \neg\, 4,\ 1$

UnitProp –¬2

$1,\ 2 \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg\, 1 \vee \neg\, 2,\ \neg\, 1 \vee \neg\, 3 \vee \neg\, 4,\ 1$

Decide 3

$1,\ 2,\ 3^{d} \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg\, 1 \vee \neg\, 2,\ \neg\, 1 \vee \neg\, 3 \vee \neg\, 4,\ 1$

UnitProp 4

$1,\ 2,\ 3^{d},\ 4 \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg\, 1 \vee \neg\, 2,\ \neg\, 1 \vee \neg\, 3 \vee \neg\, 4,\ 1$

Backtrack 3

Transition Rules DPLL – Example

$\varnothing \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg\, 1 \vee \neg\, 2,\ \neg\, 1 \vee \neg\, 3 \vee \neg\, 4,\ 1$

UnitProp 1

$1 \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg\, 1 \vee \neg\, 2,\ \neg\, 1 \vee \neg\, 3 \vee \neg\, 4,\ 1$

UnitProp –2

$1,\ 2 \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg\, 1 \vee \neg\, 2,\ \neg\, 1 \vee \neg\, 3 \vee \neg\, 4,\ 1$

Decide 3

$1,\ 2,\ 3^{d} \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg\, 1 \vee \neg\, 2,\ \neg\, 1 \vee \neg\, 3 \vee \neg\, 4,\ 1$

UnitProp 4

$1,\ 2,\ 3^{d},\ 4 \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg\, 1 \vee \neg\, 2,\ \neg\, 1 \vee \neg\, 3 \vee \neg\, 4,\ 1$

Backtrack 3

$1,\ 2,\ 3 \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg\, 1 \vee \neg\, 2,\ \neg\, 1 \vee \neg\, 3 \vee \neg\, 4,\ 1$

Transition Rules for the Original DPLL

UnitProp

$$M \parallel F, C \vee l \rightarrow M\ l \parallel F, C \vee l$$

- $M \vDash \neg C$
- l is undefined in M

Decide

$$M \parallel F, C \rightarrow M\ l^{d} \parallel F, C$$

- l or $\neg l$ occur in C
- l is undefined in M

Fail

$$M \parallel F, C \vee l \rightarrow \text{fail}$$

- $M \vDash \neg C$
- M does not contain decision literals

Backtrack

$$M\ l^{d}\ N \parallel F, C \rightarrow M \neg l \parallel F, C$$

- $M\ l^{d}\ N \vDash \neg C$
- l is the last decision literal

The Basic DPLL System – Correctness

- Some terminology
  - Irreducible state: state to which no transition rule applies.
  - Execution: sequence of transitions allowed by the rules and starting with states of the form $\emptyset \parallel F$.
  - Exhausted execution: execution ending in an irreducible state
- **Proposition** (Strong Termination) Every execution in Basic DPLL is finite
- **Proposition** (Soundness) For every exhausted execution starting with $\emptyset \parallel F$ and ending in $M \parallel F$, $M \models F$
- **Proposition** (Completeness) If F is unsatisfiable, every exhausted execution starting with $\emptyset \parallel F$ ends with **fail**
- Maintained in more general rules + theories

Extensions to DPLL

- Clausal learning
- Two watched literals
- Periodically restarting backtrack search

- More details at
- https://www.cs.princeton.edu/~zkincaid/courses/fall18/readings/SATHandbook-CDCL.pdf

Conjuctive Normal Form

- $\varphi \leftrightarrow \varphi'$ $\Rightarrow_{CNF}$ $\varphi \rightarrow \varphi' \wedge \varphi' \rightarrow \varphi$
- $\varphi \rightarrow \varphi'$ $\Rightarrow_{CNF}$ $\neg\varphi \vee \varphi'$
- $\neg(\varphi \vee \varphi')$ $\Rightarrow_{CNF}$ $\neg\varphi \wedge \neg\varphi'$
- $\neg(\varphi \wedge \varphi')$ $\Rightarrow_{CNF}$ $\neg\varphi \vee \neg\varphi'$
- $\neg\neg\varphi$ $\Rightarrow_{CNF}$ φ
- $(\varphi \wedge \varphi') \vee \varphi''$ $\Rightarrow_{CNF}$ $(\varphi \vee \varphi'') \wedge (\varphi' \vee \varphi'')$

- PROBLEM: (potential) exponential blowup of the resulting formula

Tseitin Transformation [1968]

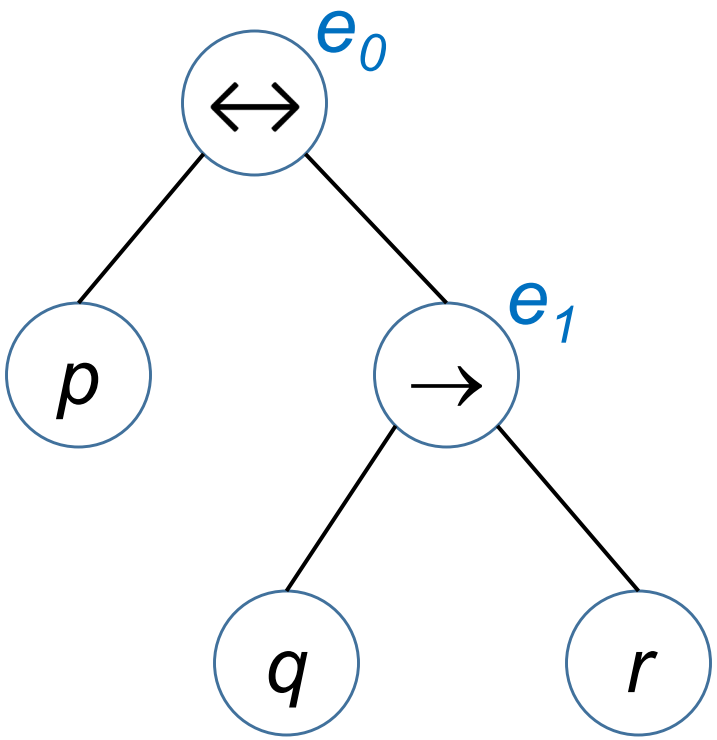
- Used in practice
  - No exponential blow-up
  - CNF formula size is linear wrt original formula
- Does not produce an equivalent CNF
- However, given F, the following holds for the computed CNF F':
  - F' is equisatisfiable to F
  - Every model of F' can be translated (i.e., projected) to a model of F
  - Every model of F can be translated (i.e., completed) to a model of F'
- No model is lost or added in the conversion

Tseitin Transformation – Main Idea

- Introduce a fresh variable e_i for every subformula G_i of F
  - e_i represents the truth value of G_i
- Assert that every e_i and G_i pair are equivalent
  - Assertions expressed as CNF
- Conjoin all such assertions in the end

Example

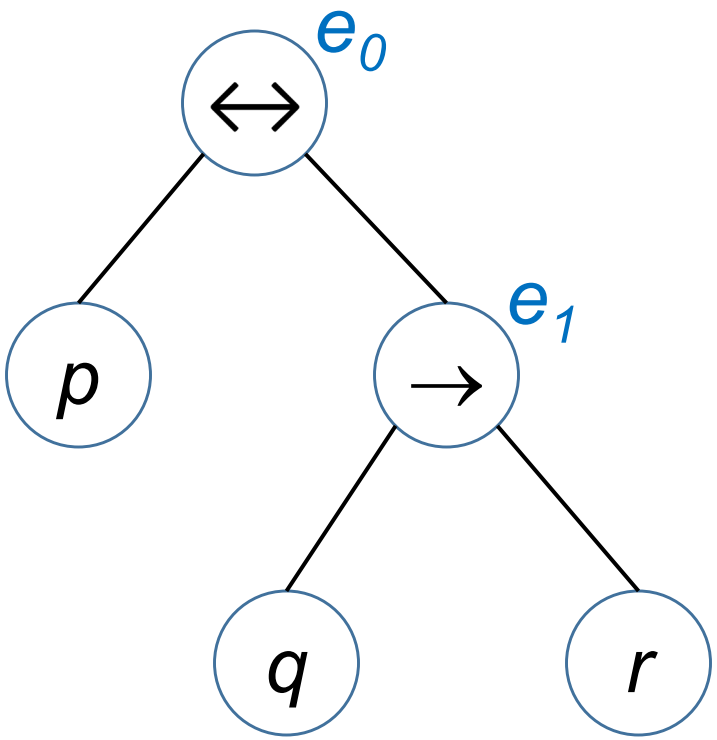
F: $p \leftrightarrow (q \rightarrow r)$



F: $e_0 \wedge (e_0 \leftrightarrow (p \leftrightarrow e_1)) \wedge (e_1 \leftrightarrow (q \rightarrow r))$

Example

$F: p \leftrightarrow (q \rightarrow r)$

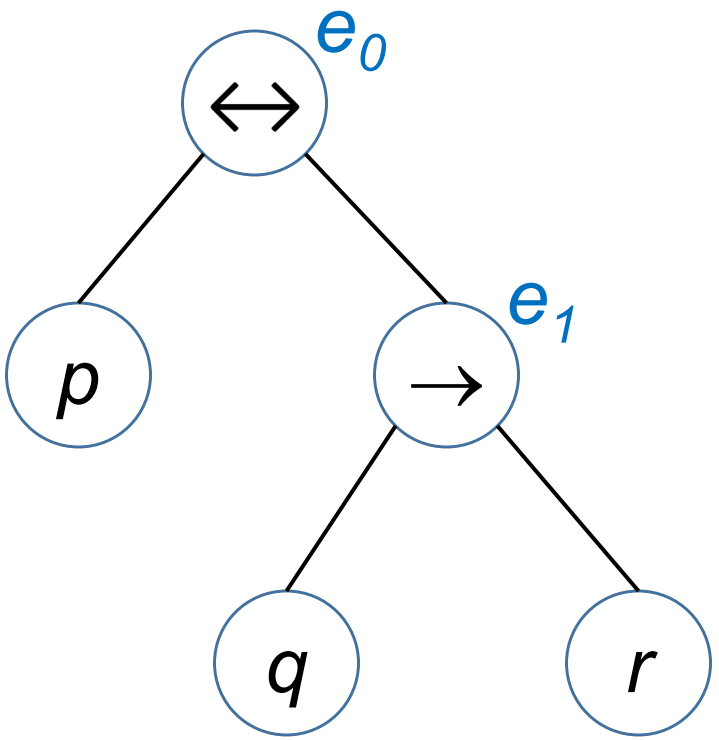


$F: e_0 \wedge (e_0 \leftrightarrow (p \leftrightarrow e_1)) \wedge (e_1 \leftrightarrow (q \rightarrow r))$

$F: e_0 \wedge (\neg e_0 \vee \neg p \vee e_1) \wedge (\neg e_0 \vee p \vee \neg e_1) \wedge (e_0 \vee p \vee e_1) \wedge (e_0 \vee \neg p \vee \neg e_1) \wedge (e_1 \leftrightarrow (q \rightarrow r))$

Example

$F: p \leftrightarrow (q \rightarrow r)$



$F: e_0 \wedge (e_0 \leftrightarrow (p \leftrightarrow e_1)) \wedge (e_1 \leftrightarrow (q \rightarrow r))$

$F: e_0 \wedge (\neg e_0 \vee \neg p \vee e_1) \wedge (\neg e_0 \vee p \vee \neg e_1) \wedge (e_0 \vee p \vee e_1) \wedge (e_0 \vee \neg p \vee \neg e_1) \wedge (e_1 \leftrightarrow (q \rightarrow r))$

$F: e_0 \wedge (\neg e_0 \vee \neg p \vee e_1) \wedge (\neg e_0 \vee p \vee \neg e_1) \wedge (e_0 \vee p \vee e_1) \wedge (e_0 \vee \neg p \vee \neg e_1) \wedge (\neg e_1 \vee \neg q \vee r) \wedge (e_1 \vee q) \wedge (e_1 \vee \neg r)$

How to Verify Programs when a SAT Solver is all you have

How to convert a program into a SAT formula?

Original code

```
x=x+y;
if (x!=1)
  x=2;
else
  x++;
assert(x<=3);
```

How to convert a program into a SAT formula?

Original code

```
x=x+y;
if (x!=1)
  x=2;
else
  x++;
assert(x<=3);
```

Convert to static single assignment

```
x1=x0+y0;
if (x1!=1)
  x2=2;
else
  x3=x1+1;
x4=(x1!=1)?x2:x3;
assert(x4<=3);
```

Generate constraints

$C \equiv x_1=x_0+y_0 \wedge x_2=2 \wedge x_3=x_1+1 \wedge (x_1!=1 \wedge x_4=x_2 \vee x_1=1 \wedge x_4=x_3)$

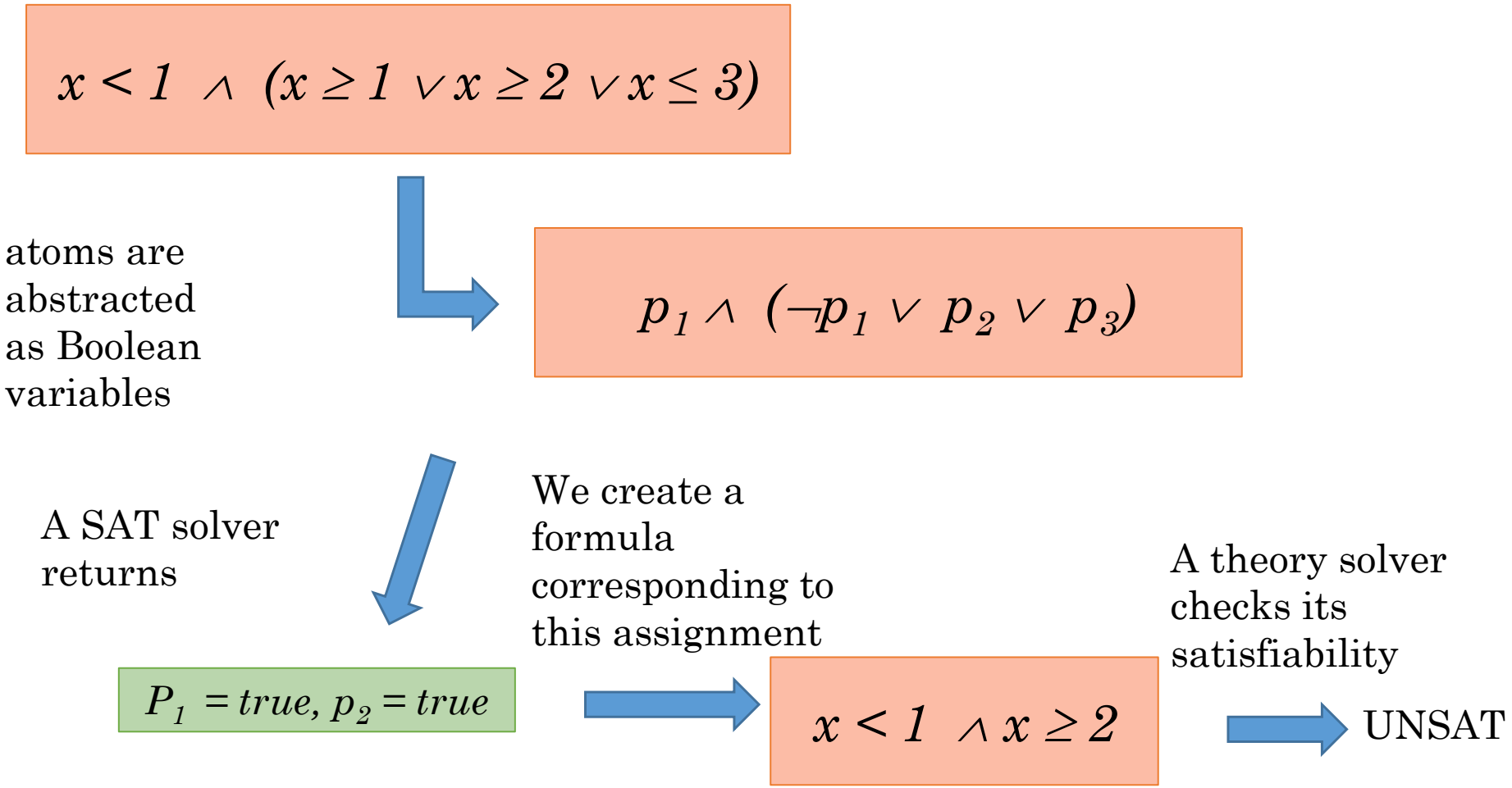
$P \equiv x_4 <= 3$

Check if $C \wedge \neg P$ is satisfiable, if it is then the assertion is violated

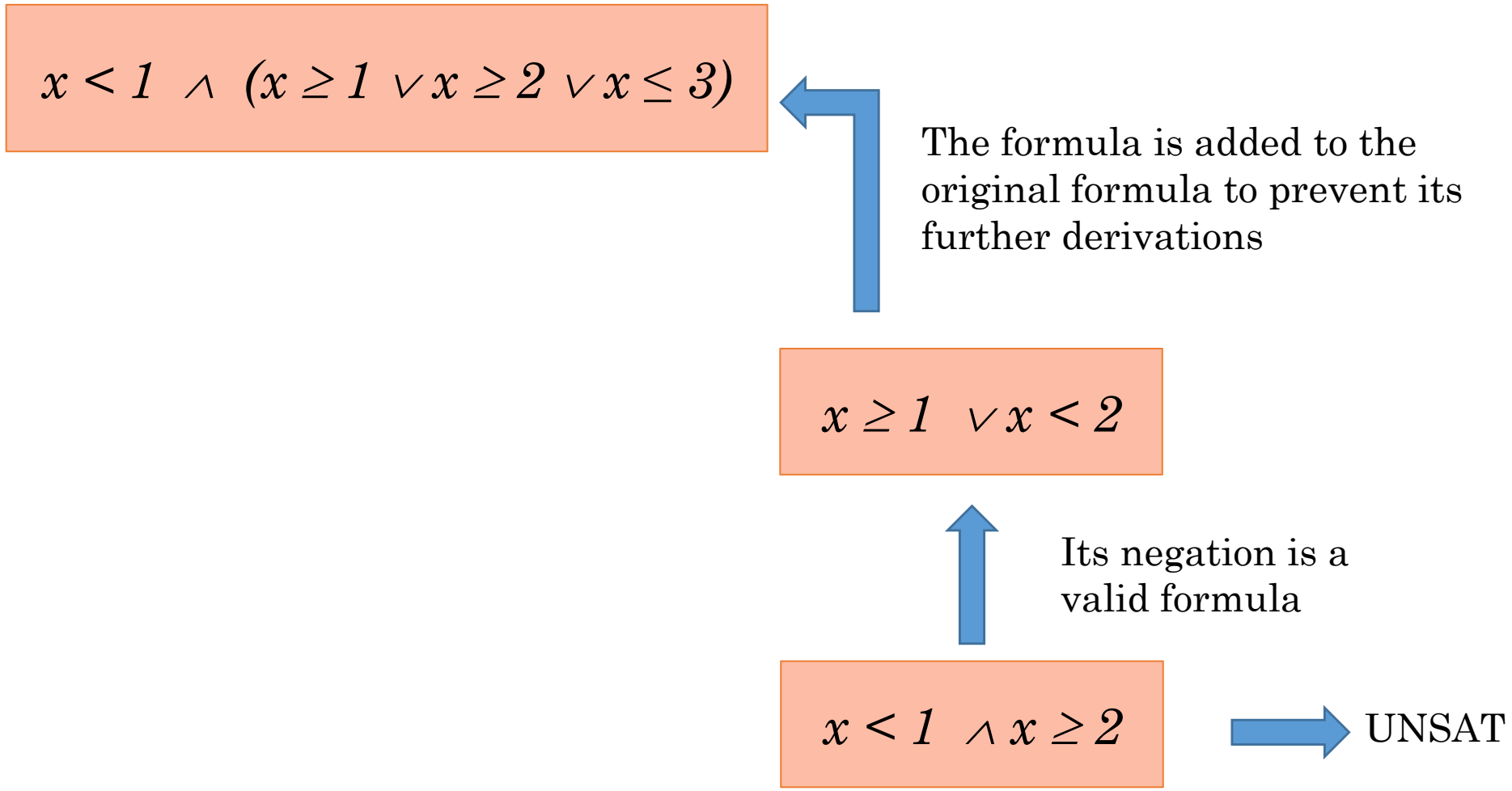
$C \wedge \neg P$ is converted to boolean logic using a bit vector representation for the integer variables $y_0, x_0, x_1, x_2, x_3, x_4$

SAT Solving is a Basis for SMT solvers

Lazy approach to SMT



Lazy approach to SMT



Lazy approach to SMT