CSE40511: Program Verification

DPLL(T) Framework

2025 Fall

Woosuk Lee

Motivation

- In the previous lectures, we learned decision procedures for various first-order theories (and their combinations)
  - that can handle only quantifier-free conjunctive formulas.
- Questions
  - How can we handle more general forms containing both conjunctions and disjunctions?
  - Can we leverage the previous CDCL algorithm for propositional logic for first-order theories?
- Answer: **DPLL(T)**

The Key Idea

- SAT solver handles boolean structure, and theory solver handles theory-specific reasoning.
- For each atomic formula in a first-order theory, transform it into a fresh propositional variable (called ***Boolean abstraction***)
- If the resulting propositional logic formula is
  - UNSAT: we are done — also UNSAT modulo first-order theory
  - SAT : it doesn't necessarily mean original formula is SAT

    Ask a theory solver to check if the SAT assignment is satisfiable modulo theory
  - If not, add conflict clause to guide the search

Motivating Example

- Suppose we have a formula in the theory of equality (T_E)

$$\underbrace{g(a) = c}_{b_1} \wedge (\neg(\underbrace{f(g(a)) = f(c)}_{b_2}) \vee \underbrace{g(a) = d}_{b_3}) \wedge \neg(\underbrace{c = d}_{b_4})$$

- Represent it as a propositional logic formula

$$b_1 \wedge (\neg b_2 \vee b_3) \wedge \neg b_4$$

Basic SMT Solving (First Iteration)

- Suppose we have a formula in the theory of equality (T_E)

$$\underbrace{g(a) = c}_{b_1} \wedge (\neg(\underbrace{f(g(a)) = f(c)}_{b_2}) \vee \underbrace{g(a) = d}_{b_3}) \wedge \neg(\underbrace{c = d}_{b_4})$$

- Represent it as a propositional logic formula

$$b_1 \wedge (\neg b_2 \vee b_3) \wedge \neg b_4$$

- SAT solver returns a solution $\{b_1 \mapsto \text{true}, b_2 \mapsto \text{false}, b_3 \mapsto \text{false}, b_4 \mapsto \text{false}\}$

Basic SMT Solving (First Iteration)

- Suppose we have a formula in the theory of equality (T_E)

$$\underbrace{g(a) = c}_{b_1} \wedge (\neg(\underbrace{f(g(a)) = f(c)}_{b_2}) \vee \underbrace{g(a) = d}_{b_3}) \wedge \neg(\underbrace{c = d}_{b_4})$$

- Represent it as a propositional logic formula

$$b_1 \wedge (\neg b_2 \vee b_3) \wedge \neg b_4$$

- SAT solver returns a solution $\{b_1 \mapsto \text{true}, b_2 \mapsto \text{false}, b_3 \mapsto \text{false}, b_4 \mapsto \text{false}\}$
- T_E solver says UNSAT for $g(a) = c \wedge f(g(a)) \neq f(c) \wedge g(a) \neq d \wedge c \neq d$

Basic SMT Solving (Second Iteration)

- Suppose we have a formula in the theory of equality (T_E)

$$\underbrace{g(a) = c}_{b_1} \wedge (\neg(\underbrace{f(g(a)) = f(c)}_{b_2}) \vee \underbrace{g(a) = d}_{b_3}) \wedge \neg(\underbrace{c = d}_{b_4})$$

- Add the negation of the current assignment

$$b_1 \wedge (\neg b_2 \vee b_3) \wedge \neg b_4 \wedge (\neg b_1 \vee b_2 \vee b_3 \vee b_4)$$

- SAT solver returns $\{b_1 \mapsto \text{true}, b_2 \mapsto \text{true}, b_3 \mapsto \text{true}, b_4 \mapsto \text{false}\}$
- T_E solver says UNSAT for $g(a) = c \wedge f(g(a) = f(c) \wedge g(a) = d \wedge c \neq d$

Basic SMT Solving (Third Iteration)

- Suppose we have a formula in the theory of equality (T_E)

$$\underbrace{g(a) = c}_{b_1} \wedge (\neg(\underbrace{f(g(a)) = f(c)}_{b_2}) \vee \underbrace{g(a) = d}_{b_3}) \wedge \neg(\underbrace{c = d}_{b_4})$$

- Add the negation of the current assignment

$$b_1 \wedge (\neg b_2 \vee b_3) \wedge \neg b_4 \wedge (\neg b_1 \vee b_2 \vee b_3 \vee b_4) \wedge (\neg b_1 \vee \neg b_2 \vee \neg b_3 \vee b_4)$$

- SAT solver returns $\{b_1 \mapsto \text{true}, b_2 \mapsto \text{false}, b_3 \mapsto \text{true}, b_4 \mapsto \text{false}\}$.
- T_E solver says UNSAT for $g(a) = c \wedge f(g(a) = f(c) \wedge g(a) = d \wedge c \neq d$

Basic SMT Solving (Fourth Iteration)

- Suppose we have a formula in the theory of equality (T_E)

$$\underbrace{g(a) = c}_{b_1} \wedge (\neg(\underbrace{f(g(a)) = f(c)}_{b_2}) \vee \underbrace{g(a) = d}_{b_3}) \wedge \neg(\underbrace{c = d}_{b_4})$$

- Add the negation of the current assignment

$$b_1 \wedge (\neg b_2 \vee b_3) \wedge \neg b_4 \wedge (\neg b_1 \vee b_2 \vee b_3 \vee b_4) \wedge (\neg b_1 \vee \neg b_2 \vee \neg b_3 \vee b_4)$$
$$\wedge (\neg b_1 \vee b_2 \vee \neg b_3 \vee b_4)$$

- SAT solver returns UNSAT
- Therefore, UNSAT

Improved SMT Solving

- Just adding negation of current assignment is **too weak**.
- Recall $\{b_1 \mapsto \text{true}, b_2 \mapsto \text{true}, b_3 \mapsto \text{true}, b_4 \mapsto \text{false}\}$

  in the second iteration and

  $\{b_1 \mapsto \text{true}, b_2 \mapsto \text{false}, b_3 \mapsto \text{true}, b_4 \mapsto \text{false}\}$

  in the third iteration

  lead to T_E UNSAT **for the same reason** $(g(a) = c \wedge g(a) = d \wedge c \neq d)$
- Instead of preventing exact same assignment, we may prevent the major reason for UNSAT.

Improved SMT Solving (First Iteration)

- Same formula

$$\underbrace{g(a) = c}_{b_1} \wedge (\neg(\underbrace{f(g(a)) = f(c)}_{b_2}) \vee \underbrace{g(a) = d}_{b_3}) \wedge \neg(\underbrace{c = d}_{b_4})$$

- SAT solver returns a solution $\{b_1 \mapsto \text{true}, b_2 \mapsto \text{false}, b_3 \mapsto \text{false}, b_4 \mapsto \text{false}\}$
- T_E solver says UNSAT for $g(a) = c \wedge f(g(a)) \neq f(c) \wedge g(a) \neq d \wedge c \neq d$
- Major reason: $b_1 \wedge \neg b_2$

Improved SMT Solving (Second Iteration)

- Same formula

$$\underbrace{g(a) = c}_{b_1} \wedge (\neg(\underbrace{f(g(a)) = f(c)}_{b_2}) \vee \underbrace{g(a) = d}_{b_3}) \wedge \neg(\underbrace{c = d}_{b_4})$$

- Add the negation of $b_1 \wedge \neg b_2$

$$b_1 \wedge (\neg b_2 \vee b_3) \wedge \neg b_4 \wedge (\neg b_1 \vee b_2)$$

- SAT solver returns a solution $\{b_1 \mapsto \text{true}, b_2 \mapsto \text{true}, b_3 \mapsto \text{true}, b_4 \mapsto \text{false}\}$
- T_E solver says UNSAT for $g(a) = c \wedge f(g(a)) = f(c) \wedge g(a) = d \wedge c \neq d$
- Major reason: $b_1 \wedge b_3 \wedge \neg b_4$

Improved SMT Solving (Third Iteration)

- Same formula

$$\underbrace{g(a) = c}_{b_1} \wedge (\neg\underbrace{(f(g(a)) = f(c))}_{b_2} \vee \underbrace{g(a) = d}_{b_3}) \wedge \neg\underbrace{(c = d)}_{b_4}$$

- Add the negation of $b_1 \wedge b_3 \wedge \neg b_4$

$$b_1 \wedge (\neg b_2 \vee b_3) \wedge \neg b_4 \wedge (\neg b_1 \vee b_2) \wedge (\neg b_1 \vee \neg b_3 \vee b_4)$$

- SAT solver returns UNSAT
- Therefore, UNSAT

Further Improved SMT Solving

- The improved version is better than the basic version (4 → 3 iterations until UNSAT).
- But still need to wait for full assignment from the SAT solver, which can be problematic.
- As soon as SAT solver makes assignment b_1 = true (i.e., $g(a) = c$), we can deduce $f(g(a) = f(c)$, which makes b_2 also true (but SAT solver assigned false to b_2 in the first iteration)
- No need to continue with SAT solving after this partial assignment to obtain b_1 = true and b_2 = false.

Further Improved SMT Solving (First Iteration)

- Same formula

$$\underbrace{g(a) = c}_{b_1} \wedge (\neg \underbrace{(f(g(a)) = f(c))}_{b_2} \vee \underbrace{g(a) = d}_{b_3}) \wedge \neg \underbrace{(c = d)}_{b_4}$$

- SAT solver assigns $b_1 \mapsto \text{true}$
- T_E solver says b_2 should also be true ($\because g(a) = c \Rightarrow f(g(a)) = f(c)$)
- Add $b_1 \implies b_2$

$$b_1 \wedge (\neg b_2 \vee b_3) \wedge \neg b_4 \wedge (\neg b_1 \vee b_2)$$

Further Improved SMT Solving (First Iteration)

- Same formula

$$\underbrace{g(a) = c}_{b_1} \wedge (\neg(\underbrace{f(g(a)) = f(c)}_{b_2}) \vee \underbrace{g(a) = d}_{b_3}) \wedge \neg(\underbrace{c = d}_{b_4})$$

- SAT solver assigns $b_1 \mapsto \text{true}, b_2 \mapsto \text{true}, b_3 \mapsto \text{true}$
- T_E solver says b_4 should be true if $b_1 \wedge b_3$
- Add $b_1 \wedge b_3 \implies b_4$

$$b_1 \wedge (\neg b_2 \vee b_3) \wedge \neg b_4 \wedge (\neg b_1 \vee b_2) \wedge (\neg b_1 \vee \neg b_3 \vee b_4)$$

Further Improved SMT Solving (First Iteration)

- Same formula

$$\underbrace{g(a) = c}_{b_1} \wedge (\neg(\underbrace{f(g(a)) = f(c)}_{b_2}) \vee \underbrace{g(a) = d}_{b_3}) \wedge \neg\underbrace{(c = d)}_{b_4}$$

- SAT solver assigns $b_1 \mapsto \text{true}, b_2 \mapsto \text{true}, b_3 \mapsto \text{true}, b_4 \mapsto \text{true}$
- UNSAT ($\because \cdots \wedge \neg b_4$)
- Conflict clause learning ($\neg b_1 \vee \neg b_3$)

$$b_1 \wedge (\neg b_2 \vee b_3) \wedge \neg b_4 \wedge (\neg b_1 \vee b_2) \wedge (\neg b_1 \vee \neg b_3 \vee b_4) \wedge (\neg b_1 \vee \neg b_3)$$

Further Improved SMT Solving (Second Iteration)

- Same formula

$$\underbrace{g(a) = c}_{b_1} \wedge (\neg\underbrace{(f(g(a)) = f(c))}_{b_2} \vee \underbrace{g(a) = d}_{b_3}) \wedge \neg\underbrace{(c = d)}_{b_4}$$

- For $b_1 \wedge (\neg b_2 \vee b_3) \wedge \neg b_4 \wedge (\neg b_1 \vee b_2) \wedge (\neg b_1 \vee \neg b_3 \vee b_4) \wedge (\neg b_1 \vee \neg b_3)$
- SAT solver says UNSAT
- Therefore, UNSAT

Summary

- The basic SMT solving uses a SAT solver as blackbox (needs 4 iterations)
- The improved SMT solving uses a SAT solver as blackbox, but add minimal conflict clauses with aid of a theory solver (needs 3 iterations).
- The further improved SMT solving does NOT a SAT solver as blackbox. It integrates a theory solver right into the SAT solver. Also, the theory solver guides the search of SAT solver (needs 2 iterations).

Names of the Approaches

- The basic approach : **Off-line SMT**
- The improved approach: **Off-line SMT with minimal unsat core**
- The further improved approach: **On-line SMT** (aka **DPLL(T)**)

Details of the Algorithms

Boolean Abstraction

- The encoding function e takes an SMT formula F and return a boolean formula as follows:
  - If F is of form $F_1 \wedge F_2$ (or $F_1 \vee F_2$) for some formula F_1, F_2, then

    $e(F) = e(F_1) \wedge e(F_2)$ (or $e(F_1) \vee e(F_2)$)
  - If F is of form $\neg F'$ for some formula F', then

    $e(F) = \neg e(F')$
  - If F is of form an SMT formula, then

    $e(F) = b$ where b is a fresh propositional variable

Quiz

- What is the boolean abstraction of the formula?

$$x = y \wedge ((y = z \wedge \neg(x > 3)) \vee x + y \leq 2)$$

Boolean Abstraction

- The function e^{-1} also exists (returns the original SMT formula for a given boolean formula)
  - e.g., $e(x = y \wedge ((y = z \wedge \neg(x > 3)) \vee x + y \leq 2)) = b_1 \wedge ((b_2 \wedge \neg b_3) \vee b_4)$
  - $e^{-1}(b_1 \wedge ((b_2 \wedge \neg b_3) \vee b_4)) =$

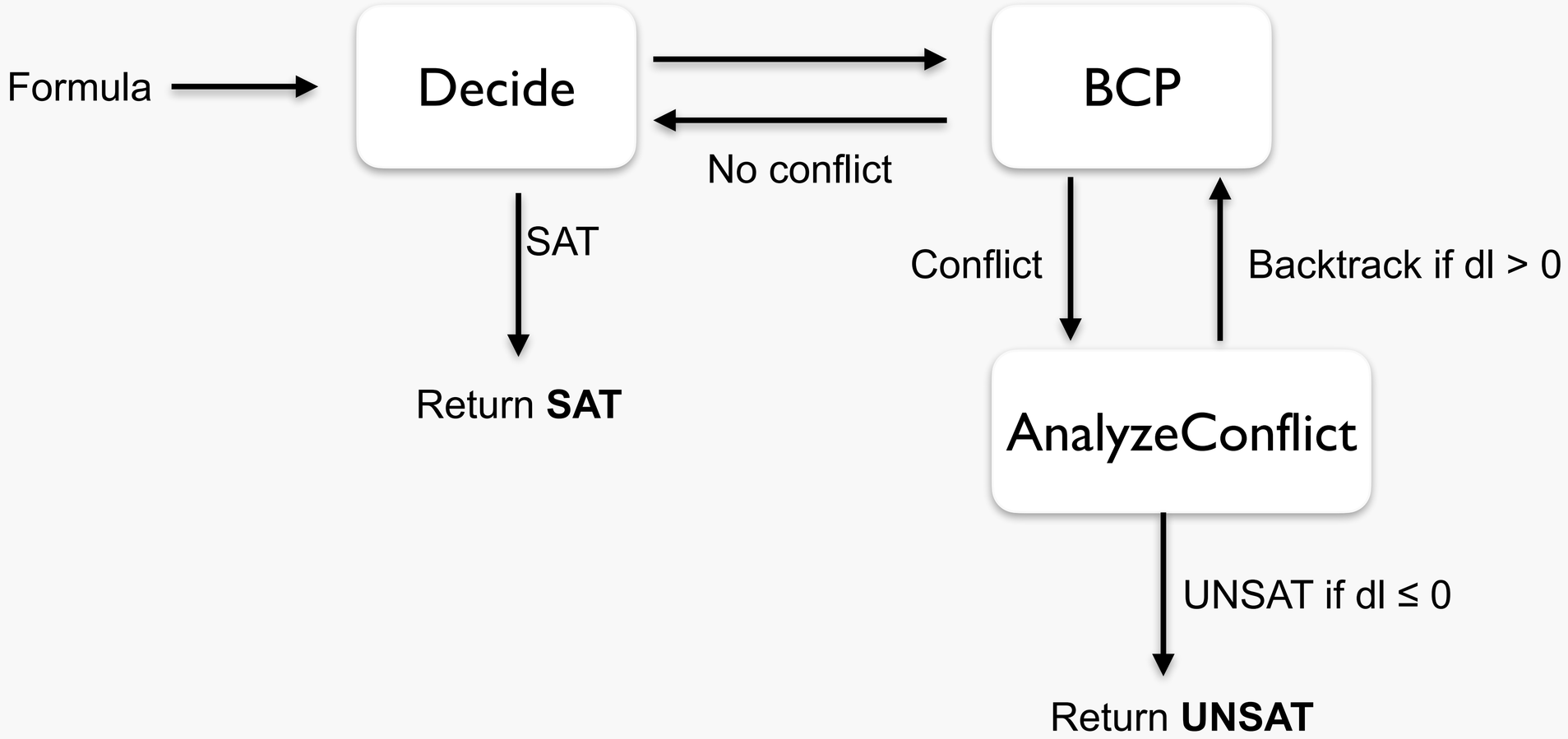
    $x = y \wedge ((y = z \wedge \neg(x > 3)) \vee x + y \leq 2)$

- If an SMT formula F is satisfiable, then $e(F)$ is also satisfiable.
  - Is the opposite also true?

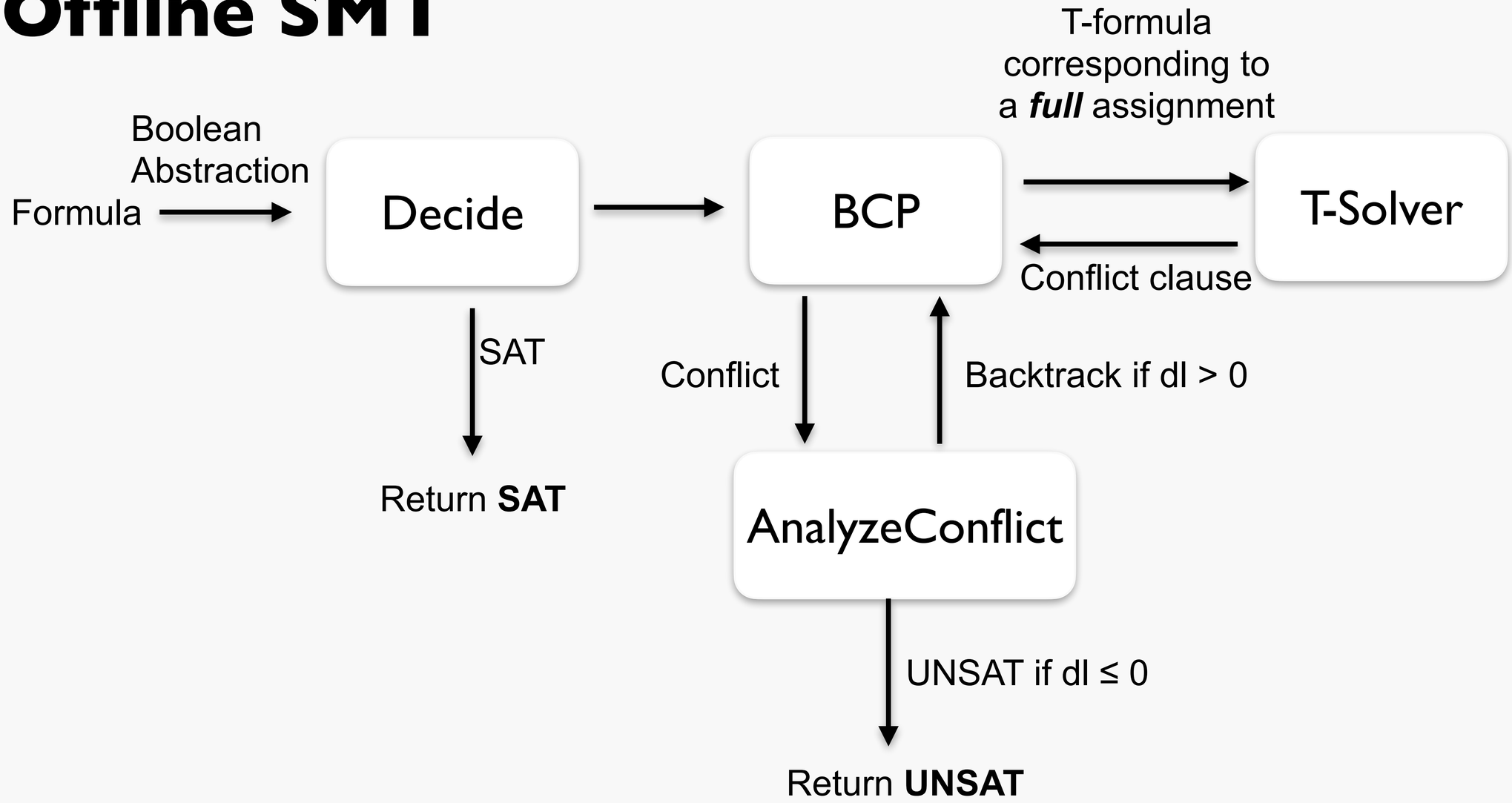
Theory Conflict Clauses

- $e(F)$ is also satisfiable $\not\Rightarrow$ SMT formula F is satisfiable
  - because the boolean abstraction abstracts away theory-related things
- We need to learn theory conflict clauses that prevent solutions of $e(F)$ which are not satisfiable modulo theory.
- Two different approaches
  - Off-line (eager): Use a SAT solver as black-box
  - On-line (lazy): Integrate theory solver into the CDCL algorithm

Review: Overview of CDCL Algorithm



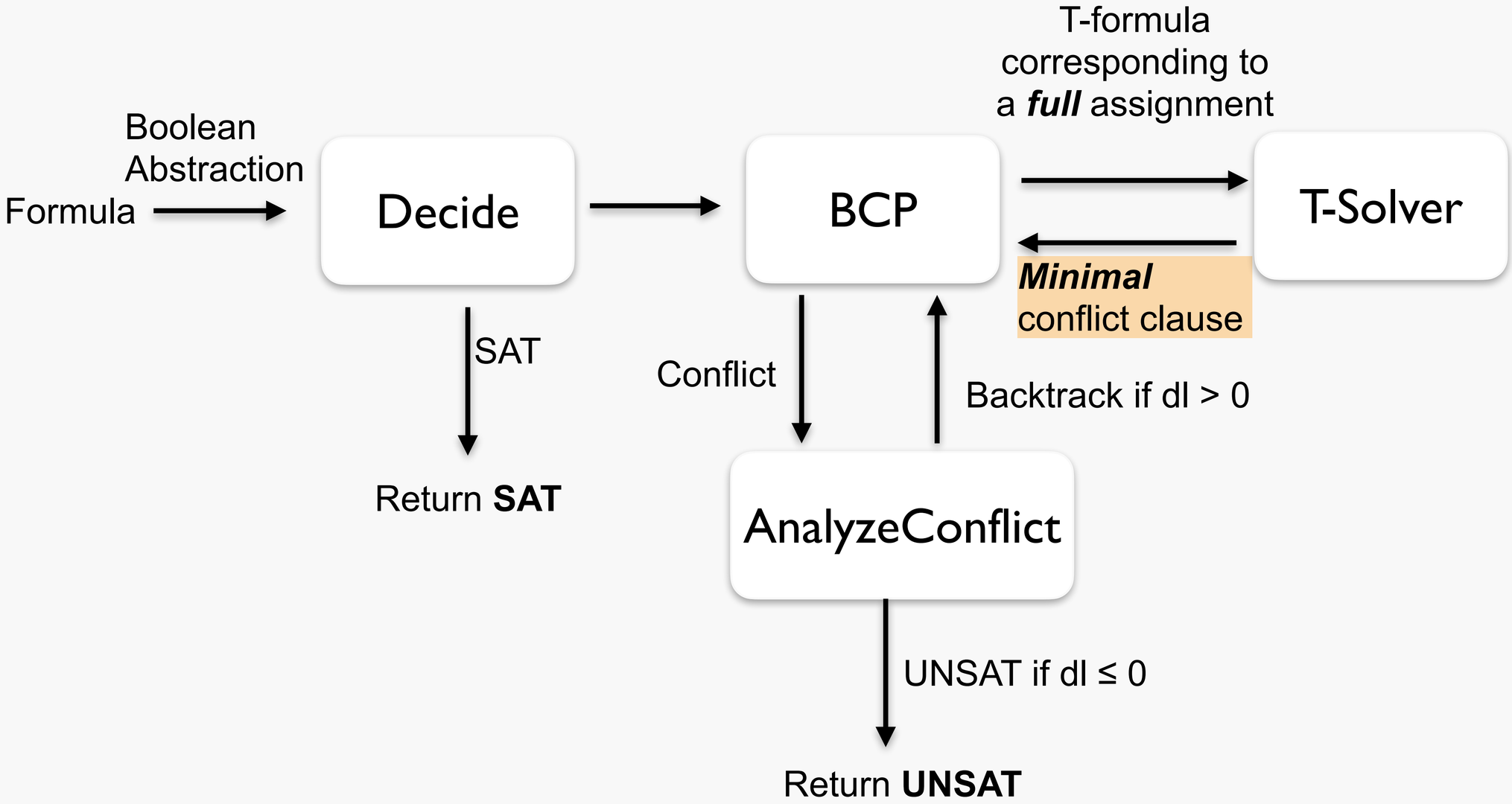
Offline SMT


Boolean
Abstraction
Formula
Decide
BCP
T-formula
corresponding to
a *full* assignment
T-Solver
Conflict clause
SAT
Return **SAT**
Conflict
Backtrack if dl > 0
AnalyzeConflict
UNSAT if dl ≤ 0
Return **UNSAT**

Offline SMT

```
function OfflineSMT (F) {
    B := e(F)
    while (true) {
        A := CDCL(B)
        if (A is UNSAT) return UNSAT
        R := T-Solver(e^-1(A))
        if (R is SAT) return SAT
        B := B ∧ ¬A
    }
}
```

Offline SMT with UNSAT core



UNSAT Core

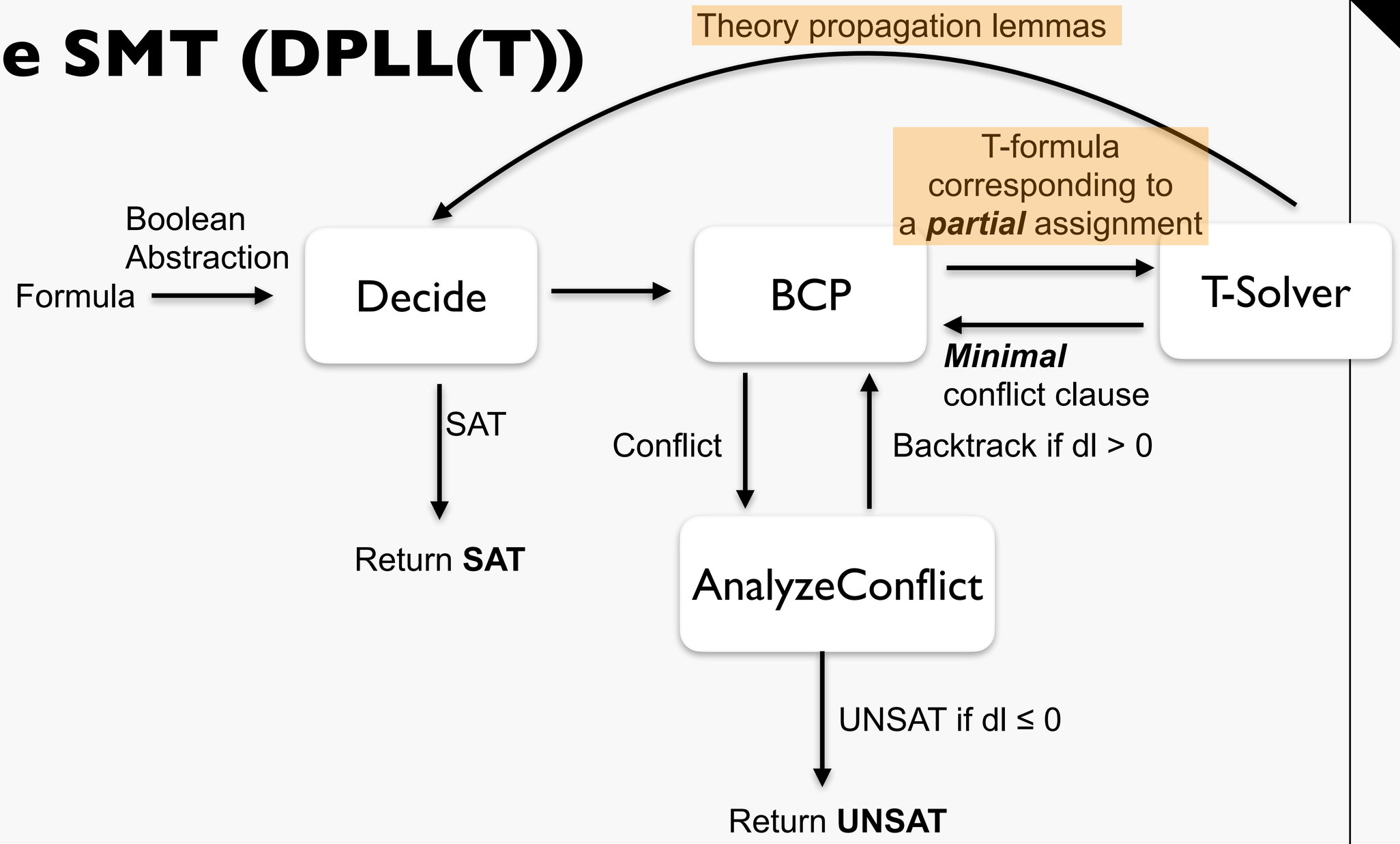
- Given an unsatisfiable formula in CNF, a subset of clauses whose conjunction is still unsatisfiable is called an ***unsatisfiable core*** of the formula.
  - What are all unsat cores of $F : x = y \wedge x < y \wedge x > y \wedge x \neq y$?

- An unsatisfiable core is called a ***minimal unsatisfiable core***, if every proper subset (allowing removal of any arbitrary clauses) of it is satisfiable.
  - What are minimal unset cores of F?

Offline SMT with UNSAT core

```
function OfflineSMT (F) {
    B := e(F)
    while (true) {
        A := CDCL(B)
        if (A is UNSAT) return UNSAT
        R := T-Solver(e^-1(A))
        if (R is SAT) return SAT
        U := MinUnsatCore(T-Solver(e^-1(A)))
        B := B ∧ ¬e(U)
    }
}
```

Online SMT (DPLL(T))



Theory Propagation

- Theory solver can infer which literals are implied by current partial assignment and let SAT solver know
- In the example, $b_1 \implies b_2$ and $b_1 \wedge b_3 \implies b_4$ are added.
- These kinds of clauses implied by theory are called ***theory propagation lemmas***.
- Adding theory propagation lemmas prevents bad assignments to boolean abstraction.

Summary

- Boolean abstraction
- Learning theory conflict clauses
- Off-line (eager) approaches
  - basic
  - with minimal unsat-core
- On-line (lazy) approach (DPLL(T))