SMT Solving

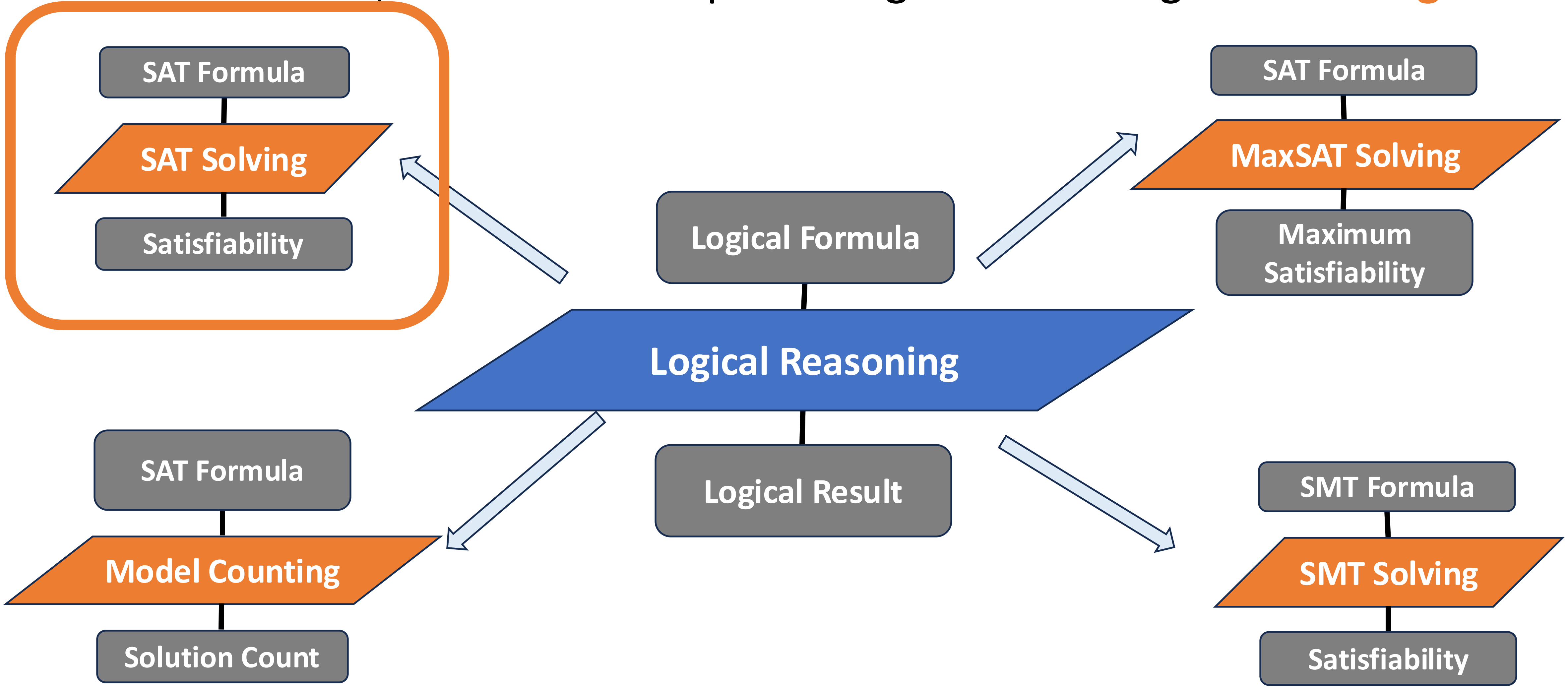
Wenxi Wang

University of Virginia

wenxiw@virginia.edu

Logical Reasoning

In this lecture, we focus on a specific logical reasoning- **SAT solving**



Recap

Logical Reasoning

Propositional Logic

Variables: Boolean

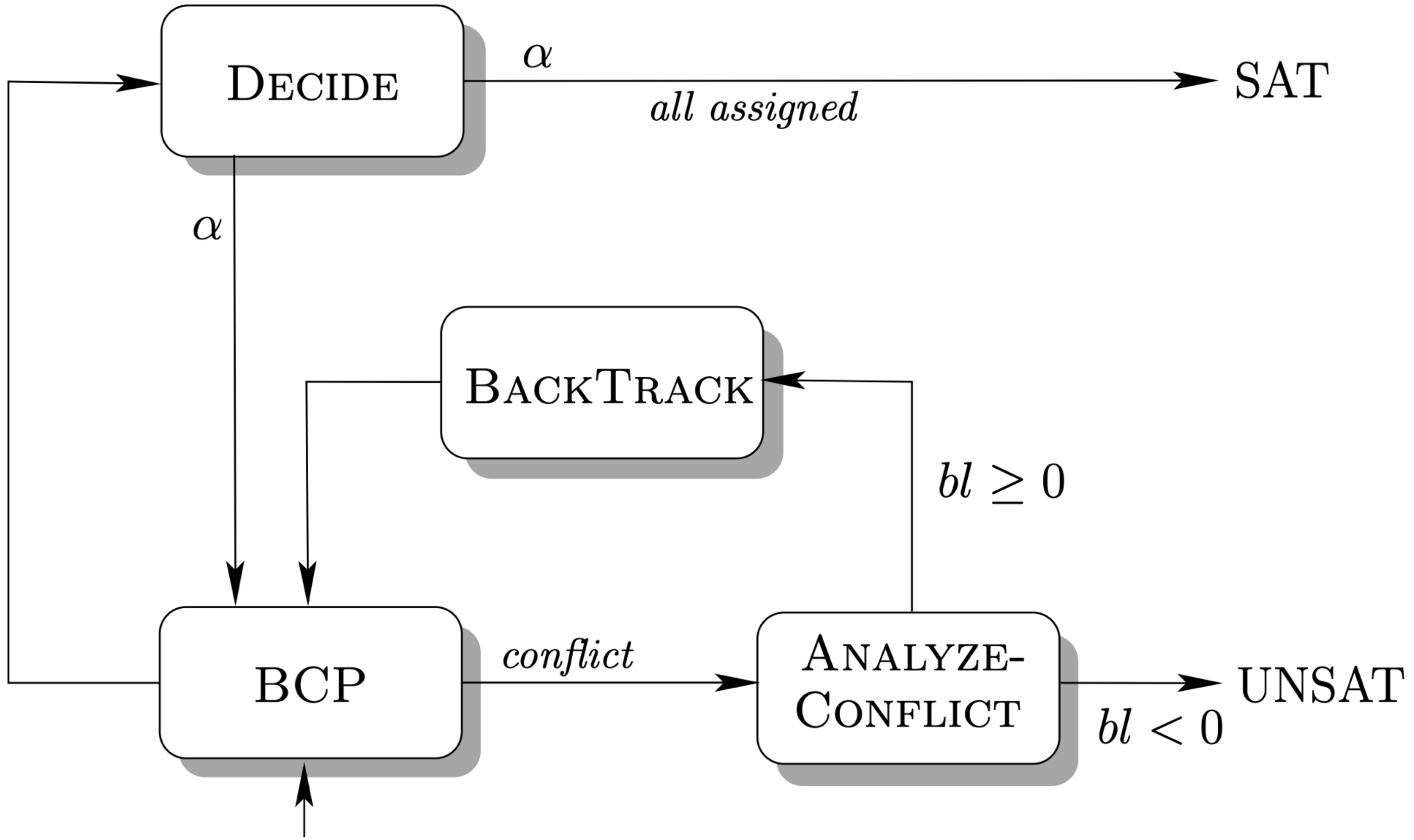
Operators sorted by precedence:

$\neg$ >> $\wedge$ >> $\vee$ >> $\Rightarrow$ >> $\Leftrightarrow$

Recap

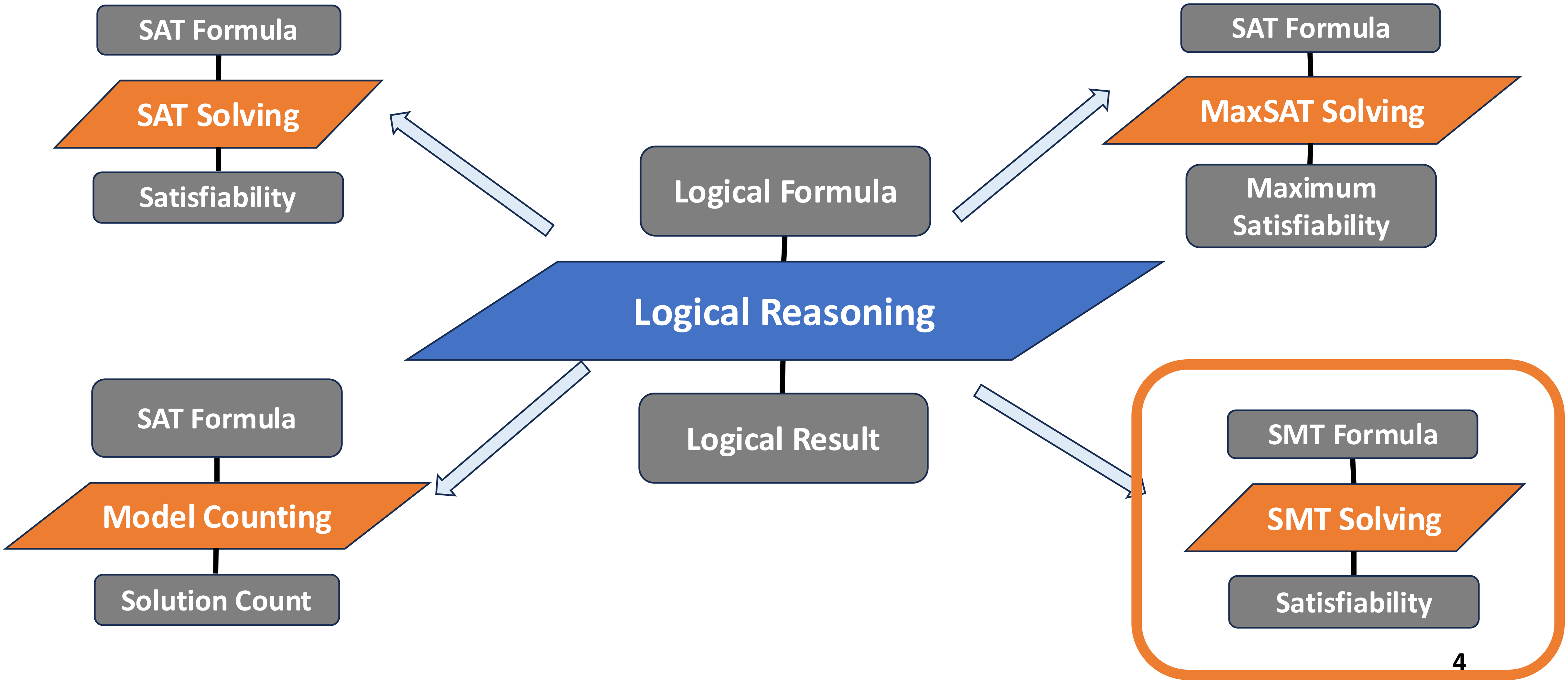
CDCL SAT solving

General Workflow



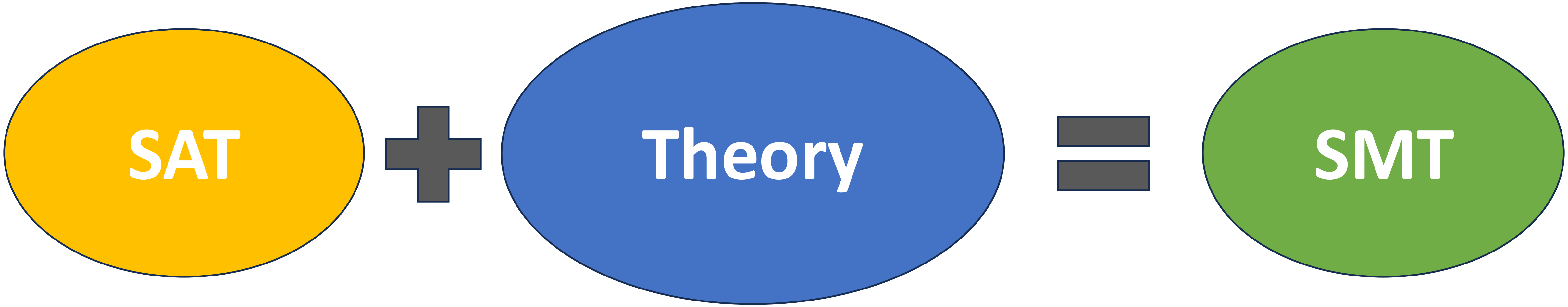
Logical Reasoning

In this lecture, we focus on **SMT solving**



SMT Solving

SMT: Satisfiability Modulo Theory



Bit vectors: ((a>> b) & c) < c

Arithmetic: $(2x + 3y \leq 5) \vee (y + 5y - 10z \geq 6)$

Equality: $y1 = y2 \wedge \neg(y1 = y3) \Rightarrow \neg(y2 = y3)$

Arrays: $(i = j \wedge a[j] = 1) \Rightarrow a[i] = 1$

… …

SMT Solving

First-order logic

First-order logic extends propositional logic with **quantifiers** and the **nonlogical symbols,** representing all kinds of **theories**.

Bit vectors: ((a>> b) & c) < c

Arithmetic: $(2x + 3y \leq 5) \vee (y + 5y - 10z \geq 6)$

Equality: $y1 = y2 \wedge \neg(y1 = y3) \Rightarrow \neg(y2 = y3)$

Arrays: $(i = j \wedge a[j] = 1) \Rightarrow a[i] = 1$

… …

Example:

- $\exists y \in Z.\ \forall x \in Z.\ x > y$,
- $\forall n \in N.\ \exists p \in N.\ n > 1 \Rightarrow (\mathit{isprime}(p) \wedge n < p < 2n)$,

where “>”, “isprime” are nonlogical symbols

SMT Solving

Let’s focus on a simple case:

From Propositional to Quantifier-Free Theories

Example:

$F = \underbrace{x1 + x2 > 9}_{\text{b1}} \vee x2 \neq x3 \wedge \underbrace{x2 = x3}_{\text{b2}} \Leftrightarrow \underbrace{x4 > x5}_{\text{b3}} \wedge A \wedge \neg B$

Propositional Skeleton PS_F = $(b1 \vee \neg b2) \wedge b2 \Leftrightarrow b3 \wedge A \wedge \neg B$

$b1$: $x1 + x2 > 9$

can be denoted as **e(x1+x2>9)**

$b2$: $x2 = x3$

can also be denoted as **e(x2 = x3)**

$b3$: $x4 > x5$

can also be denoted as **e($x4 > x5$)**

SMT Solving

Interpretation of symbols is important!

Syntax to Semantics

Example

F : x + y > z => y > z − x

We construct a "standard" interpretation **I**

The domain is the integers, $\mathbb{Z}$:D_I =$\mathbb{Z}$={…,−2,−1,0,1,2,…}

α_I:{ + $\longmapsto$+$\mathbb{Z}$, – $\longmapsto$−$\mathbb{Z}$, > $\longmapsto$>$\mathbb{Z}$, $x$$\longmapsto$10, $y$$\longmapsto$39, $x$$\longmapsto$-2}

SMT Solving

T-satisfiability

Let T be a Σ-theory.

A **Σ-**formula $\boldsymbol{\varphi}$ is T-satisfiable if there exists an interpretation ***I*** such that the interpretation satisfies $\boldsymbol{\varphi}$, $\boldsymbol{I \models \varphi}$.

SMT Solving

Two main approaches for SMT

Lazy Approach:
Integrate a theory solver with a CDCL solver for SAT

Eager Approach
Encode the SMT formula to a equi-satisfiable SAT formula

Lazy SMT Solving

General Algorithm

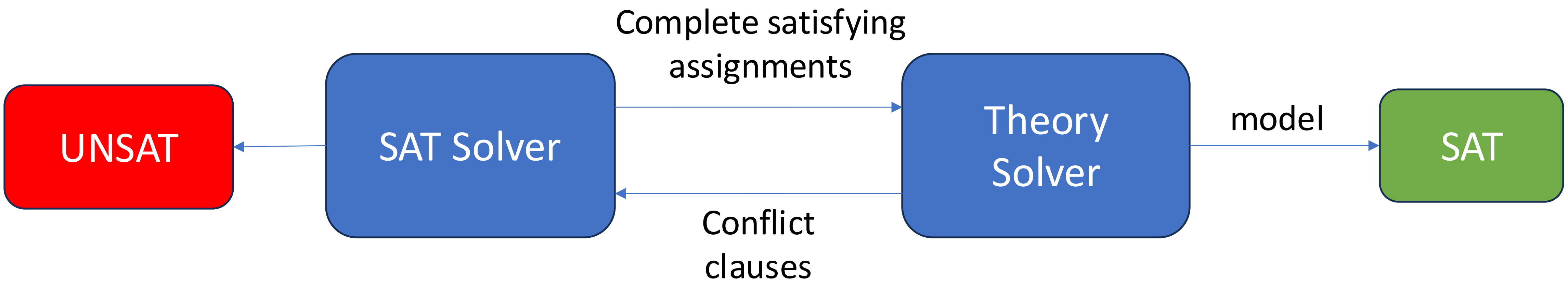
Algorithm LAZY-CDCL

Input: A formula φ
Output: "Satisfiable" if the formula is satisfiable, and "Unsatisfiable" otherwise

```
function LAZY-CDCL
    ADDCLAUSES(cnf(e(φ)));
    while (TRUE) do
        while (BCP() = "conflict") do
            backtrack-level := ANALYZE-CONFLICT();
            if backtrack-level < 0 then return "Unsatisfiable";
            else BackTrack(backtrack-level);
        if ¬DECIDE() then                                  ▷ Full assignment
            ⟨t, res⟩:=DEDUCTION(T̂h(α));                    ▷ α is the assignment
            if res="Satisfiable" then return "Satisfiable";
            ADDCLAUSES(e(t));
```

Lazy SMT Solving

General framework

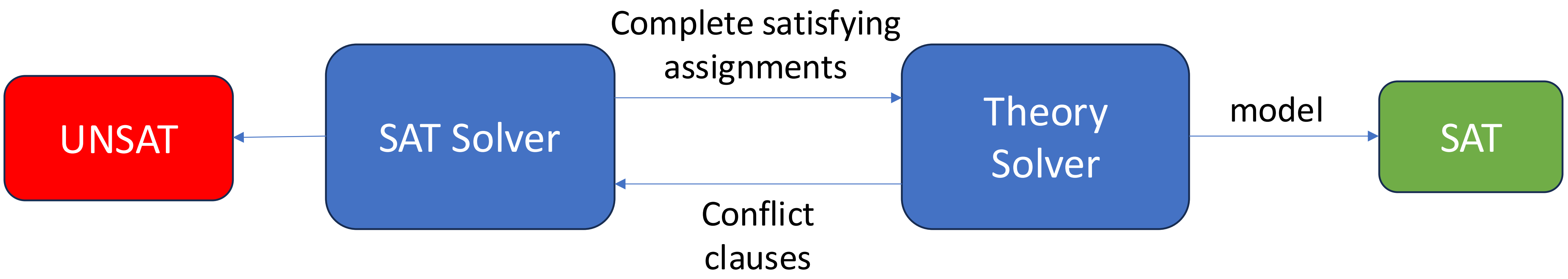


Lazy SMT Solving

Running Example

$$(x = y) \wedge (y = z) \wedge (x \neq z)$$

Propositional skeleton **PS:** **b1** $\wedge$ **b2** $\wedge$ **b3**

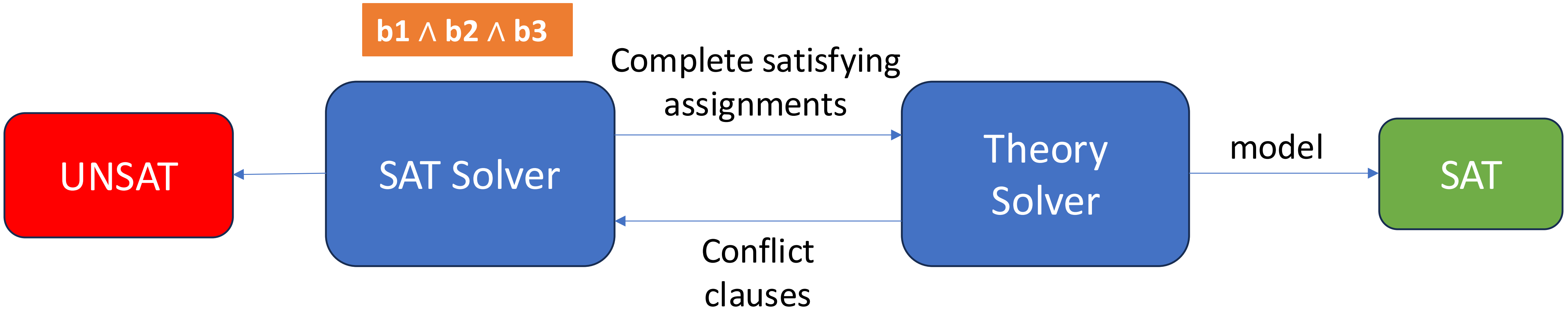


Lazy SMT Solving

Running Example

(x = y) ∧ (y = z) ∧ (x ≠ z)

Propositional skeleton **PS: b1 ∧ b2 ∧ b3**

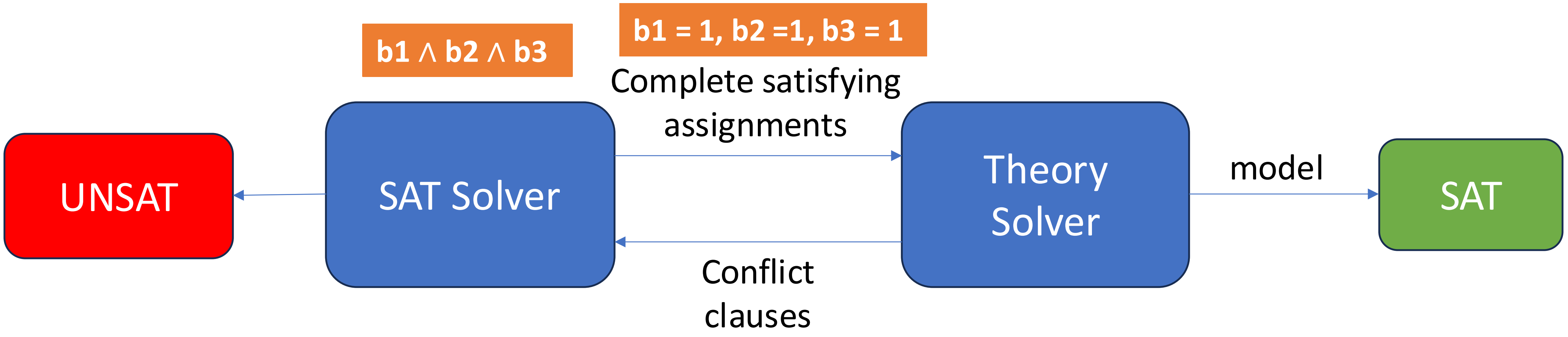


Lazy SMT Solving

Running Example

(x = y) ∧ (y = z) ∧ (x ≠ z)

Propositional skeleton **PS: b1 ∧ b2 ∧ b3**

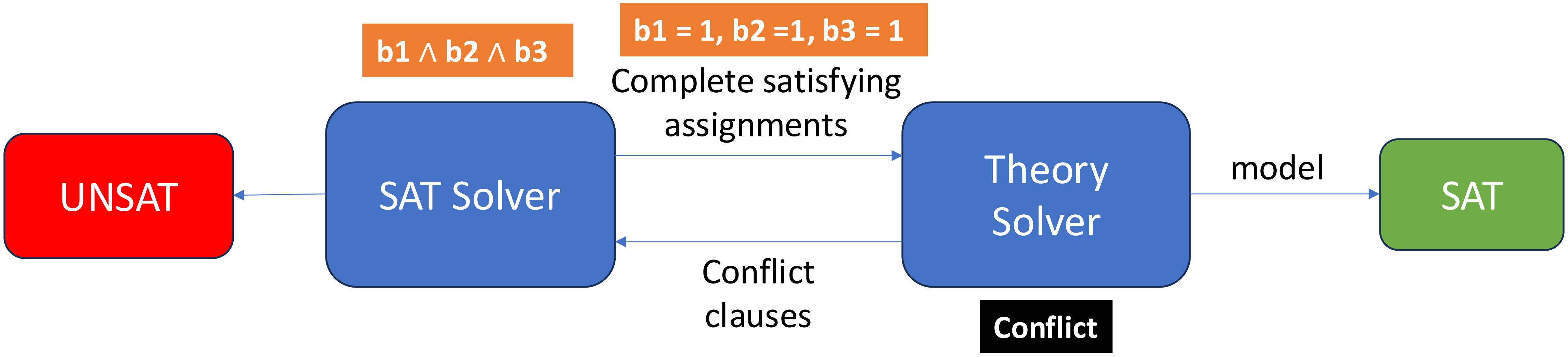


Lazy SMT Solving

Running Example

$$(x = y) \land (y = z) \land (x \neq z)$$

Propositional skeleton **PS:** **b1** $\land$ **b2** $\land$ **b3**



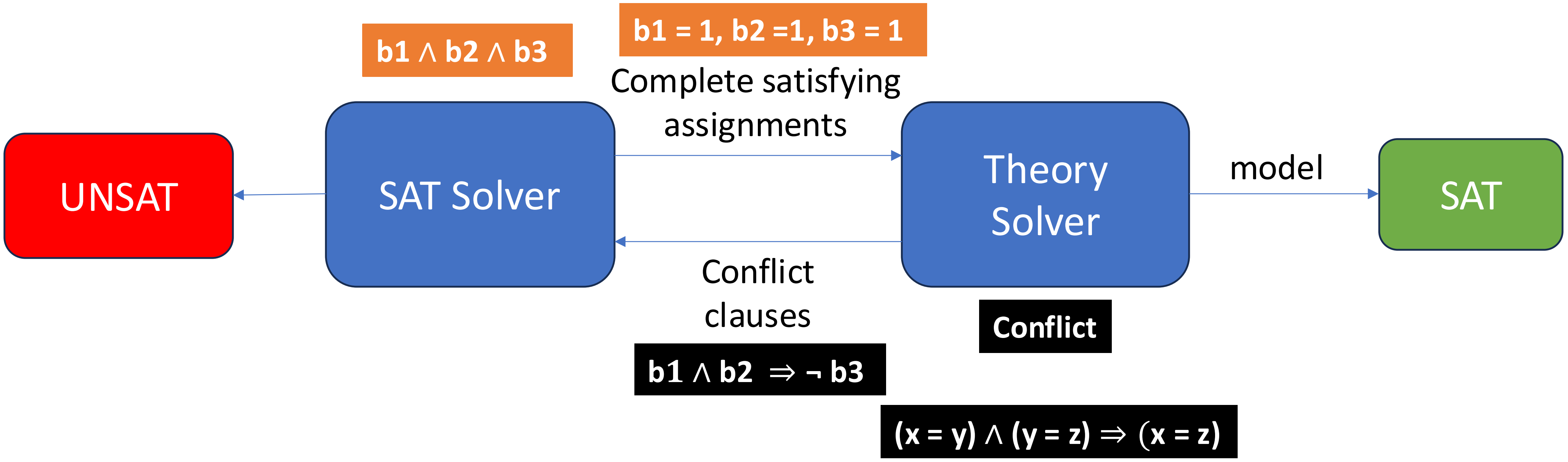
$$(x = y) \land (y = z) \Rightarrow (x = z)$$

Lazy SMT Solving

Running Example

$$(x = y) \wedge (y = z) \wedge (x \neq z)$$

Propositional skeleton **PS:** **b1 ∧ b2 ∧ b3**

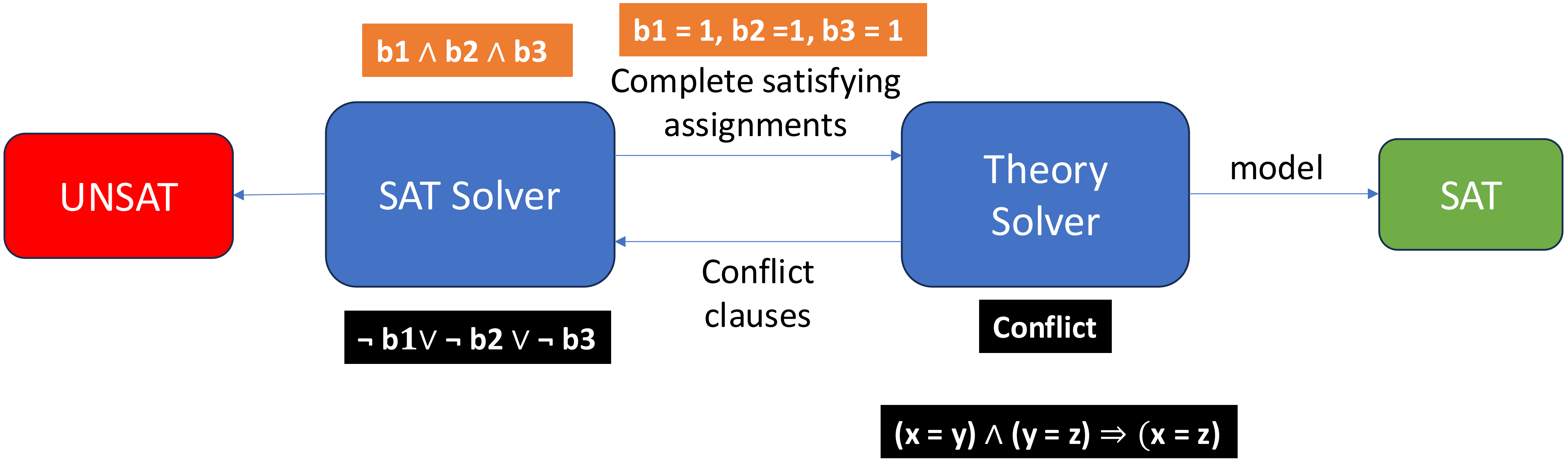


Lazy SMT Solving

Running Example

$$(x = y) \wedge (y = z) \wedge (x \neq z)$$

Propositional skeleton **PS:** $\mathbf{b1 \quad \wedge \quad b2 \quad \wedge \quad b3}$

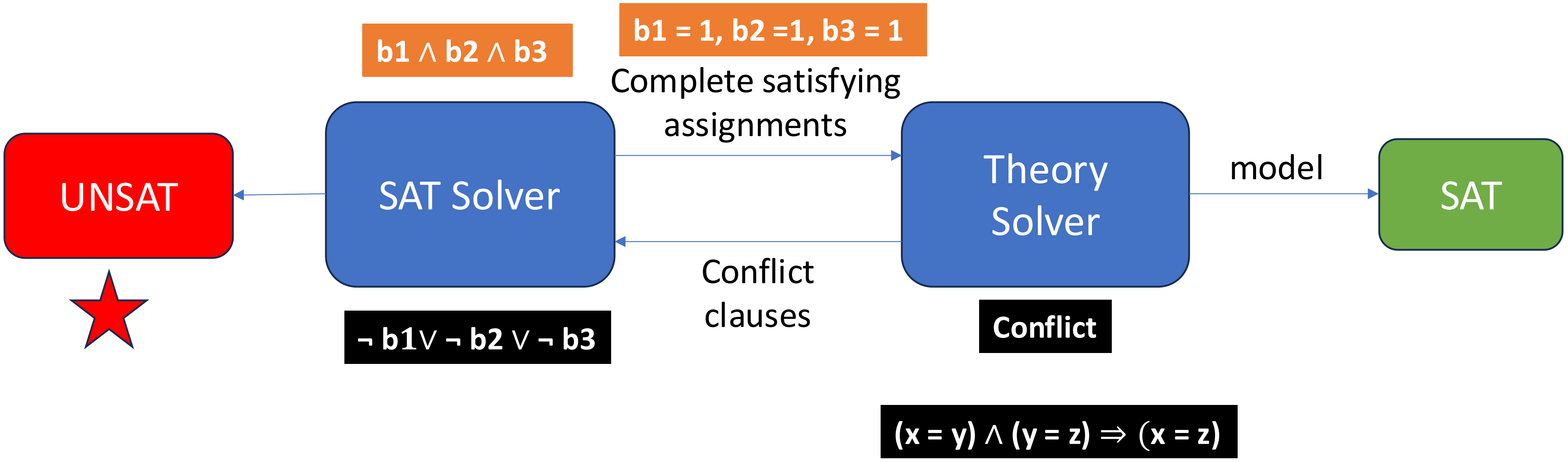


Lazy SMT Solving

Running Example

$$(x = y) \land (y = z) \land (x \neq z)$$

Propositional skeleton **PS:** **b1 ∧ b2 ∧ b3**



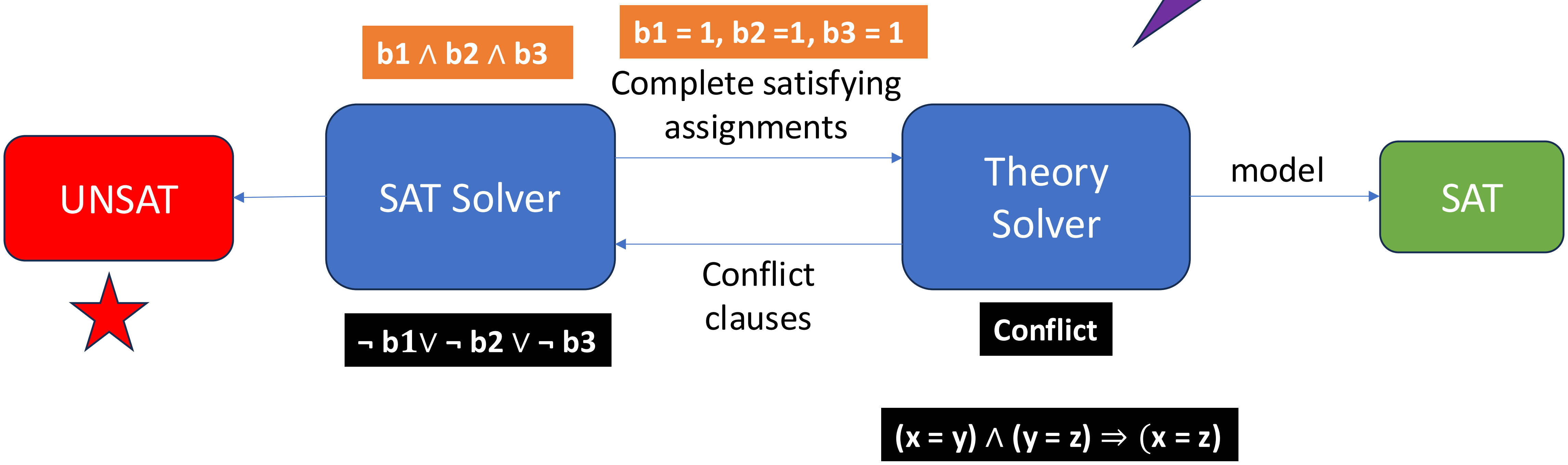
Lazy SMT Solving

Running Example

(x = y) ∧ (y = z) ∧ (x ≠ z)

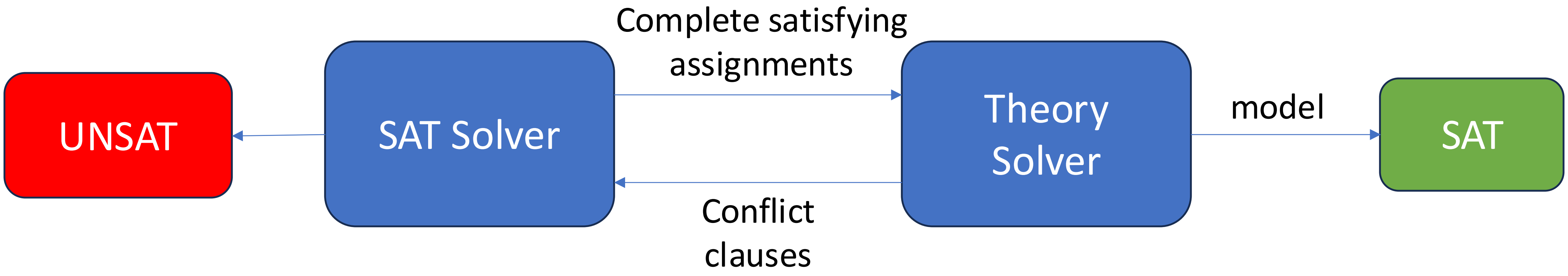
Propositional skeleton **PS: b1 ∧ b2 ∧ b3**

Lazy SMT!



Lazy SMT Solving

Still not efficient enough …



Example, a formula that contains literals

$x1 \geq 12$ and $x1 < 2$

$x1$ is integer

Assume SAT assigns $e(x1 \geq 12) \mapsto$ true and $e(\ x1 < 2) \mapsto$ true.

Lazy SMT solving does not call Theory solver until a full satisfying assignment is found.

Any call to Theory solver results in a contradiction between these two facts.

waste time to compute the complete SAT assignment

Improvements: DPLL(T)

Do Theory Propagation when the SAT assignment is still partial!

Example

a formula that contains literals $x1 \geq 12$ and $x1 < 2$

Assume SAT makes $e(x1 \geq 12)$ **true**,

Theory solver detects that $\neg(x1 < 2)$ is implied, that is

$e(x1 \geq 12) \Rightarrow \neg e(x1 < 2)$

Then $\boldsymbol{\neg e(x1 \geq 12)} \vee \boldsymbol{\neg e(x1 < 2)}$ is added to SAT solver

Improvements: DPLL(T)

Call theory solver to do Theory Propagation
when the SAT assignment is still partial!

Algorithm LAZY-CDCL

Input: A formula φ
Output: "Satisfiable" if the formula is satisfiable, and "Unsat
otherwise

```
function LAZY-CDCL
    ADDCLAUSES(cnf(e(φ)));
    while (TRUE) do
        while (BCP() = "conflict") do
            backtrack-level := ANALYZE-CONFLICT();
            if backtrack-level < 0 then return "Unsatisfi
            else BackTrack(backtrack-level);
        if ¬DECIDE() then                                    ▷
            ⟨t, res⟩:=DEDUCTION(T̂h(α));                      ▷ α is
            if res="Satisfiable" then return "Satisfiable
            ADDCLAUSES(e(t));
```

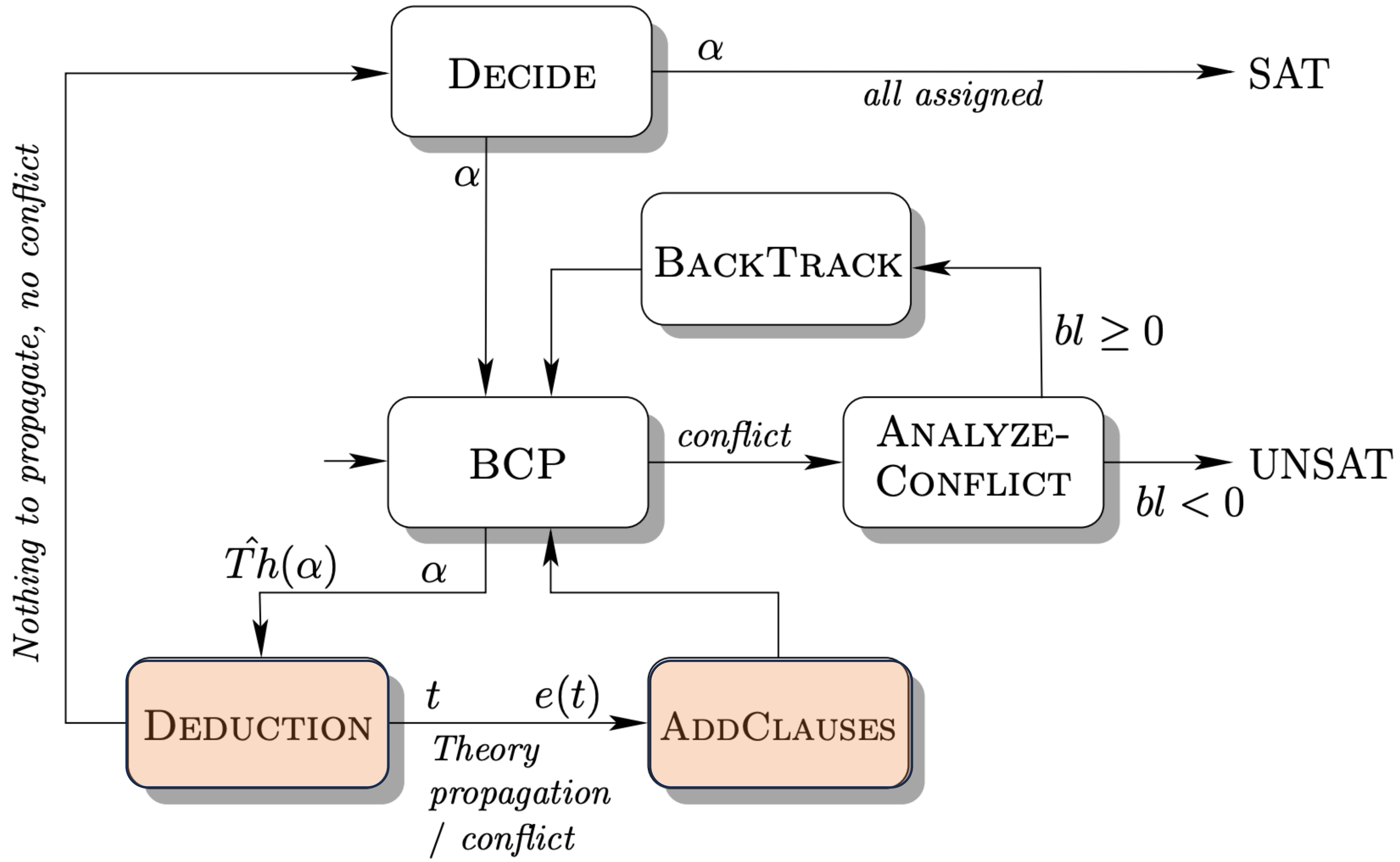
Algorithm DPLL(T)

Input: A formula φ
Output: "Satisfiable" if the formula is satisfiable, and "Unsatisfiable" otherwise

```
function DPLL(T)
    ADDCLAUSES(cnf(e(φ)));
    while (TRUE) do
        repeat
            while (BCP() = "conflict") do
                backtrack-level := ANALYZE-CONFLICT();
                if backtrack-level < 0 then return "Unsatisfiable";
                else BackTrack(backtrack-level);
            ⟨t, res⟩:=DEDUCTION(T̂h(α));
            ADDCLAUSES(e(t));
        until t ≡ TRUE;
        if α is a full assignment then return "Satisfiable";
        DECIDE();
```

Improvements: DPLL(T)

Call theory solver to do Theory Propagation
when the SAT assignment is still partial!



Improvements: DPLL(T)

Call theory solver to do Theory Propagation
when the SAT assignment is still partial!