ECE750T-28:
Computer-aided Reasoning for Software Engineering

Lecture 17: SMT Solvers and the DPPL($\mathcal{T}$) Framework

Vijay Ganesh
(Original notes from Isil Dillig)
An SMT (satisfiability modulo theory) solver is a tool that decides satisfiability of formulas in combination of various first-order theories.
SMT Solvers

- An **SMT (satisfiability modulo theory) solver** is a tool that decides satisfiability of formulas in combination of various first-order theories.

- SMT solvers are generalizations of SAT solvers.
SMT Solvers

- An SMT (satisfiability modulo theory) solver is a tool that decides satisfiability of formulas in combination of various first-order theories.

- SMT solvers are generalizations of SAT solvers.

- Can think of SMT formula as SAT formula where propositional variables are replaced with formulas in first-order theories.
SMT Solvers

- An SMT (satisfiability modulo theory) solver is a tool that decides satisfiability of formulas in combination of various first-order theories.

- SMT solvers are generalizations of SAT solvers.

- Can think of SMT formula as SAT formula where propositional variables are replaced with formulas in first-order theories.

- Common first-order theories SMT solvers reason about:
SMT Solvers

- An SMT (satisfiability modulo theory) solver is a tool that decides satisfiability of formulas in combination of various first-order theories.

- SMT solvers are generalizations of SAT solvers.

- Can think of SMT formula as SAT formula where propositional variables are replaced with formulas in first-order theories.

- Common first-order theories SMT solvers reason about:
  - Theory of equality.
SMT Solvers

- An SMT (satisfiability modulo theory) solver is a tool that decides satisfiability of formulas in combination of various first-order theories.

- SMT solvers are generalizations of SAT solvers.

- Can think of SMT formula as SAT formula where propositional variables are replaced with formulas in first-order theories.

- Common first-order theories SMT solvers reason about:
  - Theory of equality
  - Theory of rationals
SMT Solvers

- An SMT (satisfiability modulo theory) solver is a tool that decides satisfiability of formulas in combination of various first-order theories.

- SMT solvers are generalizations of SAT solvers.

- Can think of SMT formula as SAT formula where propositional variables are replaced with formulas in first-order theories.

- Common first-order theories SMT solvers reason about:
  - Theory of equality
  - Theory of rationals
  - Theory of integers
SMT Solvers

➤ An SMT (satisfiability modulo theory) solver is a tool that decides satisfiability of formulas in combination of various first-order theories

➤ SMT solvers are generalizations of SAT solvers

➤ Can think of SMT formula as SAT formula where propositional variables are replaced with formulas in first-order theories

➤ Common first-order theories SMT solvers reason about:
  ➤ Theory of equality
  ➤ Theory of bitvectors
  ➤ Theory of rationals
  ➤ Theory of integers
SMT Solvers

- An SMT (satisfiability modulo theory) solver is a tool that decides satisfiability of formulas in combination of various first-order theories.

- SMT solvers are generalizations of SAT solvers.

- Can think of SMT formula as SAT formula where propositional variables are replaced with formulas in first-order theories.

- Common first-order theories SMT solvers reason about:
  - Theory of equality
  - Theory of rationals
  - Theory of integers
  - Theory of bitvectors
  - Theory of arrays
SMT Solvers

- An SMT (satisfiability modulo theory) solver is a tool that decides satisfiability of formulas in combination of various first-order theories.

- SMT solvers are generalizations of SAT solvers.

- Can think of SMT formula as SAT formula where propositional variables are replaced with formulas in first-order theories.

- Common first-order theories SMT solvers reason about:
  - Theory of equality
  - Theory of rationals
  - Theory of integers
  - Theory of bitvectors
  - Theory of arrays
  - Difference logic
Applications of SMT Solvers

- SMT solvers have gained enormous popularity over the last several years
Applications of SMT Solvers

- SMT solvers have gained enormous popularity over the last several years
- SMT solving is active research topic today
Applications of SMT Solvers

- SMT solvers have gained enormous popularity over the last several years
- SMT solving is active research topic today
- Many applications: software verification, programming languages, test case generation, planning and scheduling, . . .
Applications of SMT Solvers

- SMT solvers have gained enormous popularity over the last several years
- SMT solving is active research topic today
- Many applications: software verification, programming languages, test case generation, planning and scheduling, . . .
- Slogan: “Whatever SAT solvers can do, SMT solvers can do better!”
Applications of SMT Solvers

- SMT solvers have gained enormous popularity over the last several years

- SMT solving is an active research topic today

- Many applications: software verification, programming languages, test case generation, planning and scheduling, ...

- Slogan: “Whatever SAT solvers can do, SMT solvers can do better!”

- This is the case because SMT solvers generalize SAT solvers; they can handle much richer theories than propositional logic
Existing SMT Solvers

- Many existing off-the-shelf SMT solvers:
Existing SMT Solvers

- Many existing off-the-shelf SMT solvers:
  - Yices (SRI)
  - Z3 (Microsoft Research)
  - CVC3 (NYU, U Iowa)
  - STP (Stanford)
  - MathSAT (U Trento, Italy)
  - Barcelogic (Catalonia, Spain)
Existing SMT Solvers

- Many existing off-the-shelf SMT solvers:
  - Yices (SRI)
  - Z3 (Microsoft Research)
  - CVC3 (NYU, U Iowa)
  - STP (Stanford)
  - MathSAT (U Trento, Italy)
  - Barcelogic (Catalonia, Spain)

- Annual competition SMT-COMP between solvers; tools ranked in various categories
Existing SMT Solvers

- Many existing off-the-shelf SMT solvers:
  - Yices (SRI)
  - Z3 (Microsoft Research)
  - CVC3 (NYU, U Iowa)
  - STP (Stanford)
  - MathSAT (U Trento, Italy)
  - Barcelogic (Catalonia, Spain)

- Annual competition SMT-COMP between solvers; tools ranked in various categories

- All of these SMT solvers have many users
Existing SMT Solvers

- Many existing off-the-shelf SMT solvers:
  - Yices (SRI)
  - Z3 (Microsoft Research)
  - CVC3 (NYU, U Iowa)
  - STP (Stanford)
  - MathSAT (U Trento, Italy)
  - Barcelogic (Catalonia, Spain)

- Annual competition SMT-COMP between solvers; tools ranked in various categories

- All of these SMT solvers have many users

- For instance, at Microsoft, there are at least two dozen projects that rely on the Z3 SMT solver
Overview

▶ Plan for today: Get the complete picture of how SMT solvers work
Overview

- **Plan for today**: Get the complete picture of how SMT solvers work

- We’ve already learned about some aspects of SMT solvers
Overview

▶ Plan for today: Get the complete picture of how SMT solvers work

▶ We’ve already learned about some aspects of SMT solvers

▶ Already know how to decide satisfiability of several qff first-order theories (theory of equality, theory of rationals, theory of integers)
Overview

- **Plan for today:** Get the complete picture of how SMT solvers work

- We’ve already learned about some aspects of SMT solvers

- Already know how to decide satisfiability of several qff first-order theories (theory of equality, theory of rationals, theory of integers)

- Also already know how to combine these theories using Nelson-Oppen technique
Overview

▶ **Plan for today:** Get the complete picture of how SMT solvers work

▶ We’ve already learned about some aspects of SMT solvers

▶ Already know how to decide satisfiability of several qff first-order theories (theory of equality, theory of rationals, theory of integers)

▶ Also already know how to combine these theories using Nelson-Oppen technique

▶ **Big missing piece:** How to handle boolean structure of SMT formulas including disjunctions
Motivation for DPLL($\mathcal{T}$)

- So far, decided satisfiability of first-order theories by converting to DNF
Motivation for DPLL(\(\mathcal{T}\))

- So far, decided satisfiability of first-order theories by converting to DNF
- In reality, this is completely impractical: DNF conversion can yield exponentially larger formula
Motivation for DPLL(\(\mathcal{T}\))

- So far, decided satisfiability of first-order theories by converting to DNF
- In reality, this is completely impractical: DNF conversion can yield exponentially larger formula
- For many real problems, DNF conversion is prohibitively expensive
Motivation for DPLL($\mathcal{T}$)

- So far, decided satisfiability of first-order theories by converting to DNF

- In reality, this is completely impractical: DNF conversion can yield exponentially larger formula

- For many real problems, DNF conversion is prohibitively expensive

- Thus, we need a way to decide satisfiability of SMT formulas without expensive conversion to DNF
DPLL(\(T\)) Overview

- **Key idea underlying SMT solvers**: Combine DPLL algorithm for SAT solving with theory solvers
DPLL(\(\mathcal{T}\)) Overview

- **Key idea underlying SMT solvers**: Combine DPLL algorithm for SAT solving with theory solvers

- **Theory solver**: Decision procedure for checking satisfiability in conjunctive fragment
DPLL(\(T\)) Overview

- **Key idea underlying SMT solvers:** Combine DPLL algorithm for SAT solving with theory solvers

- **Theory solver:** Decision procedure for checking satisfiability in conjunctive fragment

- **This architecture where we combine DPLL-based SAT solvers with theory solvers is known as DPLL(\(T\)) framework**
**DPLL(\(\mathcal{T}\)) Overview**

- **Key idea underlying SMT solvers:** Combine DPLL algorithm for SAT solving with theory solvers

- **Theory solver:** Decision procedure for checking satisfiability in conjunctive fragment

- This architecture where we combine DPLL-based SAT solvers with theory solvers is known as **DPLL(\(\mathcal{T}\)) framework**

- Called DPLL(\(\mathcal{T}\)) because we combine DPLL algorithm with solver for theory \(\mathcal{T}\)
DPLL(\(\mathcal{T}\)) Overview

- Key idea underlying SMT solvers: Combine DPLL algorithm for SAT solving with theory solvers
- Theory solver: Decision procedure for checking satisfiability in conjunctive fragment
- This architecture where we combine DPLL-based SAT solvers with theory solvers is known as DPLL(\(\mathcal{T}\)) framework
- Called DPLL(\(\mathcal{T}\)) because we combine DPLL algorithm with solver for theory \(\mathcal{T}\)
- However, \(\mathcal{T}\) can be a combination theory, such as \(\mathcal{T}_{\leq} \cup \mathcal{T}_{\mathbb{Z}}\)
**DPLL(\(\mathcal{T}\)) Overview**

- **Key idea underlying SMT solvers:** Combine DPLL algorithm for SAT solving with theory solvers

- **Theory solver:** Decision procedure for checking satisfiability in conjunctive fragment

- This architecture where we combine DPLL-based SAT solvers with theory solvers is known as **DPLL(\(\mathcal{T}\)) framework**

- Called **DPLL(\(\mathcal{T}\))** because we combine DPLL algorithm with solver for theory \(\mathcal{T}\)

- However, \(\mathcal{T}\) can be a **combination theory**, such as \(\mathcal{T}_{\leq} \cup \mathcal{T}_{\mathbb{Z}}\)

- As before, solver for \(\mathcal{T}_{\leq} \cup \mathcal{T}_{\mathbb{Z}}\) is obtained by using Nelson-Oppen technique
Main Idea of DPLL(\(\mathcal{T}\))

- In the DPLL(\(\mathcal{T}\)) framework, SAT solver handles boolean structure of formula
Main Idea of DPLL(T)

- In the DPLL(T) framework, SAT solver handles boolean structure of formula

- For this, treat each atomic formula as a propositional variable $\Rightarrow$ resulting formula called **boolean abstraction**
Main Idea of DPLL($\mathcal{T}$)

- In the DPLL($\mathcal{T}$) framework, SAT solver handles boolean structure of formula

- For this, treat each atomic formula as a propositional variable $\Rightarrow$ resulting formula called **boolean abstraction**

- Now, use SAT solver to decide satisfiability of boolean abstraction
Main Idea of DPPL($\mathcal{T}$), cont.

- If there is no satisfying assignment to boolean abstraction, formula is $\text{UNSAT}$.
Main Idea of DPPL($\mathcal{T}$), cont.

- If there is no satisfying assignment to boolean abstraction, formula is **UNSAT**

- If there is satisfying assignment to boolean abstraction, formula may not be **SAT**
Main Idea of DPPL(\(\mathcal{T}\)), cont.

- If there is no satisfying assignment to boolean abstraction, formula is \textbf{UNSAT}

- If there is satisfying assignment to boolean abstraction, formula may not be \textbf{SAT}

- Main job of the theory solver is to check whether assignments made by SAT solver is \textit{Satisfiable Modulo Theory (SMT)}
Main Idea of DPPL(\(T\)), cont.

- If there is no satisfying assignment to boolean abstraction, formula is \textbf{UNSAT}.

- If there is satisfying assignment to boolean abstraction, formula may not be SAT.

- Main job of the theory solver is to check whether assignments made by SAT solver is \textbf{Satisfiable Modulo Theory (SMT)}.

- If SAT solver finds assignment that is consistent with theory, then SMT formula is satisfiable.
SMT Formulas and Boolean Abstraction

- SMT formula in theory $\mathcal{T}$ formed as usual (structural induction):

$$F := a^i_{\mathcal{T}} \mid F_1 \land F_2 \mid F_1 \lor F_2 \mid \neg F$$
SMT Formulas and Boolean Abstraction

- SMT formula in theory $\mathcal{T}$ formed as usual (structural induction):
  $$F := a^i_\mathcal{T} \mid F_1 \land F_2 \mid F_1 \lor F_2 \mid \neg F$$

- For each SMT formula, define a bijective function $B$, called boolean abstraction function, that maps SMT formula to overapproximate SAT formula
SMT Formulas and Boolean Abstraction

- SMT formula in theory $\mathcal{T}$ formed as usual (structural induction):
  $$ F := a^i_{\mathcal{T}} \mid F_1 \land F_2 \mid F_1 \lor F_2 \mid \neg F $$

- For each SMT formula, define a bijective function $B$, called boolean abstraction function, that maps SMT formula to overapproximate SAT formula

- Function $B$ defined inductively as follows:
SMT Formulas and Boolean Abstraction

- SMT formula in theory $\mathcal{T}$ formed as usual (structural induction):
  \[ F ::= a_\mathcal{T}^i \mid F_1 \land F_2 \mid F_1 \lor F_2 \mid \neg F \]

- For each SMT formula, define a bijective function $B$, called boolean abstraction function, that maps SMT formula to overapproximate SAT formula

- Function $B$ defined inductively as follows:
  \[ B(a_\mathcal{T}^i) = b_i \]
SMT Formulas and Boolean Abstraction

- SMT formula in theory $T$ formed as usual (structural induction):
  \[ F := a^i_T \mid F_1 \land F_2 \mid F_1 \lor F_2 \mid \neg F \]

- For each SMT formula, define a bijective function $B$, called boolean abstraction function, that maps SMT formula to overapproximate SAT formula

- Function $B$ defined inductively as follows:
  \[ B(a^i_T) = b_i \]
  \[ B(F_1 \land F_2) = B(F_1) \land B(F_2) \]
SMT Formulas and Boolean Abstraction

- SMT formula in theory $\mathcal{T}$ formed as usual (structural induction):
  \[
  F := a^i_\mathcal{T} \mid F_1 \land F_2 \mid F_1 \lor F_2 \mid \neg F
  \]

- For each SMT formula, define a bijective function $B$, called boolean abstraction function, that maps SMT formula to overapproximate SAT formula

- Function $B$ defined inductively as follows:
  \[
  \begin{align*}
  B(a^i_\mathcal{T}) &= b_i \\
  B(F_1 \land F_2) &= B(F_1) \land B(F_2) \\
  B(F_1 \lor F_2) &= B(F_1) \lor B(F_2)
  \end{align*}
  \]
SMT Formulas and Boolean Abstraction

- SMT formula in theory $\mathcal{T}$ formed as usual (structural induction):
  \[ F ::= a^i_{\mathcal{T}} | F_1 \land F_2 | F_1 \lor F_2 | \lnot F \]

- For each SMT formula, define a bijective function $\mathcal{B}$, called boolean abstraction function, that maps SMT formula to overapproximate SAT formula

- Function $\mathcal{B}$ defined inductively as follows:
  \[
  \begin{align*}
  \mathcal{B}(a^i_{\mathcal{T}}) &= b_i \\
  \mathcal{B}(F_1 \land F_2) &= \mathcal{B}(F_1) \land \mathcal{B}(F_2) \\
  \mathcal{B}(F_1 \lor F_2) &= \mathcal{B}(F_1) \lor \mathcal{B}(F_2) \\
  \mathcal{B}(\lnot F) &= \lnot \mathcal{B}(F_1)
  \end{align*}
  \]
What is the boolean abstraction of this formula?

\[ F : x = z \land ((y = z \land x \neq z) \lor \neg(x = z)) \]
Example

- What is the boolean abstraction of this formula?

\[ F : \quad x = z \land ((y = z \land x \neq z) \lor \lnot(x = z)) \]

- \[ B(F) = b_1 \land ((b_2 \land b_3) \lor \lnot b_1) \]
Example

What is the boolean abstraction of this formula?

\[ F : x = z \land ((y = z \land x \neq z) \lor \neg(x = z)) \]

\[ B(F) = b_1 \land ((b_2 \land b_3) \lor \neg b_1) \]

Boolean abstraction is also called \textit{boolean skeleton}
What is the boolean abstraction of this formula?

\[ F : x = z \land ((y = z \land x \neq z) \lor \neg(x = z)) \]

\[ B(F) = b_1 \land ((b_2 \land b_3) \lor \neg b_1) \]

Boolean abstraction is also called boolean skeleton

Since \( B \) is a bijective function, \( B^{-1} \) also exists
Example

What is the boolean abstraction of this formula?

\[ F : x = z \land ((y = z \land x \neq z) \lor \neg(x = z)) \]

\[ B(F) = b_1 \land ((b_2 \land b_3) \lor \neg b_1) \]

Boolean abstraction is also called boolean skeleton

Since \( B \) is a bijective function, \( B^{-1} \) also exists

What is \( B^{-1}(b_2 \lor \neg b_1) \)?
Example

What is the boolean abstraction of this formula?

\[ F : x = z \land ((y = z \land x \neq z) \lor \neg(x = z)) \]

\[ B(F) = b_1 \land ((b_2 \land b_3) \lor \neg b_1) \]

Boolean abstraction is also called boolean skeleton

Since \( B \) is a bijective function, \( B^{-1} \) also exists

What is \( B^{-1}(b_2 \lor \neg b_1) \)? \[ y = z \lor \neg(x = z) \]
Boolean Abstraction as Overapproximation

- **Observe:** The boolean abstraction constructed this way overapproximates satisfiability of the formula.
Boolean Abstraction as Overapproximation

- **Observe:** The boolean abstraction constructed this way overapproximates satisfiability of the formula

- Is this formula satisfiable?

\[ F : x = z \land ((y = z \land x \neq z) \lor \neg(x = z)) \]
Boolean Abstraction as Overapproximation

- **Observe:** The boolean abstraction constructed this way overapproximates satisfiability of the formula

- Is this formula satisfiable? No

\[ F : x = z \land ((y = z \land x \neq z) \lor \neg(x = z)) \]
Boolean Abstraction as Overapproximation

- **Observe:** The boolean abstraction constructed this way overapproximates satisfiability of the formula

- Is this formula satisfiable? **No**

\[
F : \quad x = z \land ((y = z \land x \neq z) \lor \neg(x = z))
\]

- Boolean abstraction: \( B(F) = b_1 \land ((b_2 \land b_3) \lor \neg b_1) \)
Boolean Abstraction as Overapproximation

- **Observe:** The boolean abstraction constructed this way overapproximates satisfiability of the formula

- Is this formula satisfiable? No

\[ F : x = z \land ((y = z \land x \neq z) \lor \neg(x = z)) \]

- Boolean abstraction: \( B(F) = b_1 \land ((b_2 \land b_3) \lor \neg b_1) \)

- Is this satisfiable?
Boolean Abstraction as Overapproximation

- **Observe:** The boolean abstraction constructed this way overapproximates satisfiability of the formula

- Is this formula satisfiable? No

\[ \begin{align*}
F : & \quad x = z \land ((y = z \land x \neq z) \lor \neg(x = z)) \\
\end{align*} \]

- Boolean abstraction: \( B(F) = b_1 \land ((b_2 \land b_3) \lor \neg b_1) \)

- Is this satisfiable? Yes
Boolean Abstraction as Overapproximation

- **Observe:** The boolean abstraction constructed this way overapproximates satisfiability of the formula

- Is this formula satisfiable? **No**

\[ F : x = z \land ((y = z \land x \neq z) \lor \neg(x = z)) \]

- Boolean abstraction: \( \mathcal{B}(F) = b_1 \land ((b_2 \land b_3) \lor \neg b_1) \)

- Is this satisfiable? **Yes**

- What is a sat assignment?
Boolean Abstraction as Overapproximation

- **Observe:** The boolean abstraction constructed this way overapproximates satisfiability of the formula

- Is this formula satisfiable? **No**
  
  $$F : \ x = z \land ((y = z \land x \neq z) \lor \neg(x = z))$$

- Boolean abstraction: $$\mathcal{B}(F) = b_1 \land ((b_2 \land b_3) \lor \neg b_1)$$

- Is this satisfiable? **Yes**

- What is a sat assignment? $$A = b_1 \land b_2 \land b_3$$
Boolean Abstraction as Overapproximation

- **Observe**: The boolean abstraction constructed this way overapproximates satisfiability of the formula

- Is this formula satisfiable? No

  \[ F : x = z \land ((y = z \land x \neq z) \lor \neg(x = z)) \]

- Boolean abstraction: \( B(F) = b_1 \land ((b_2 \land b_3) \lor \neg b_1) \)

- Is this satisfiable? Yes

- What is a sat assignment? \( A = b_1 \land b_2 \land b_3 \)

- What is \( B^{-1}(A) \)?
Boolean Abstraction as Overapproximation

- **Observe:** The boolean abstraction constructed this way overapproximates satisfiability of the formula

- Is this formula satisfiable? No

  \[ F : x = z \land ((y = z \land x \neq z) \lor \neg(x = z)) \]

- Boolean abstraction: \[ B(F) = b_1 \land ((b_2 \land b_3) \lor \neg b_1) \]

- Is this satisfiable? Yes

- What is a sat assignment? \( A = b_1 \land b_2 \land b_3 \)

- What is \( B^{-1}(A) \)? \( x = y \land y = z \land x \neq z \)
Boolean Abstraction as Overapproximation

- **Observe**: The boolean abstraction constructed this way overapproximates satisfiability of the formula

- Is this formula satisfiable? No

  \[ F : x = z \land ((y = z \land x \neq z) \lor \neg (x = z)) \]

- Boolean abstraction: \( B(F) = b_1 \land ((b_2 \land b_3) \lor \neg b_1) \)

- Is this satisfiable? Yes

- What is a sat assignment? \( A = b_1 \land b_2 \land b_3 \)

- What is \( B^{-1}(A) \)? \( x = y \land y = z \land x \neq z \)

- Is \( B^{-1}(A) \) satisfiable?
Boolean Abstraction as Overapproximation

- **Observe:** The boolean abstraction constructed this way overapproximates satisfiability of the formula

- Is this formula satisfiable? **No**

\[ F : x = z \land ((y = z \land x \neq z) \lor \neg(x = z)) \]

- Boolean abstraction: \( B(F) = b_1 \land ((b_2 \land b_3) \lor \neg b_1) \)

- Is this satisfiable? **Yes**

- What is a sat assignment? \( A = b_1 \land b_2 \land b_3 \)

- What is \( B^{-1}(A) \)? \( x = y \land y = z \land x \neq z \)

- Is \( B^{-1}(A) \) satisfiable? **No**
SMT Solving: Simplest Version

- In simplest version of SMT solvers, construct boolean abstraction \( B(F) \) of SMT formula \( F \)
SMT Solving: Simplest Version

- In simplest version of SMT solvers, construct boolean abstraction $B(F)$ of SMT formula $F$

- If $B(F)$ is unsat, return unsat
SMT Solving: Simplest Version

- In simplest version of SMT solvers, construct boolean abstraction $B(F)$ of SMT formula $F$

- If $B(F)$ is unsat, return unsat

- If $B(F)$ is sat, get sat assignment $A$ (conjunction of propositional literals)
SMT Solving: Simplest Version

- In simplest version of SMT solvers, construct boolean abstraction $B(F)$ of SMT formula $F$

- If $B(F)$ is unsat, return unsat

- If $B(F)$ is sat, get sat assignment $A$ (conjunction of propositional literals)

- Construct $B^{-1}(A)$; this is conjunction of atomic $T$-formulas
SMT Solving: Simplest Version

- In simplest version of SMT solvers, construct boolean abstraction $B(F)$ of SMT formula $F$
- If $B(F)$ is unsat, return unsat
- If $B(F)$ is sat, get sat assignment $A$ (conjunction of propositional literals)
- Construct $B^{-1}(A)$; this is conjunction of atomic $T$-formulas
- Query $T$-solver for satisfiability of $B^{-1}(A)$
If $\mathcal{T}$-solver decides $B^{-1}(A)$ is sat, return SAT.

Why? Because we found an assignment that (i) both satisfies boolean structure, and (ii) consistent with theory axioms.

If $B^{-1}(A)$ is unsat, does this mean original formula is UNSAT? No b/c might be other ways of satisfying boolean structure.

In this case, construct new boolean abstraction $B(F) \land \neg A$.

Repeat until we find assignment consistent with theory or until boolean abstraction is unsat.
If $\mathcal{T}$-solver decides $B^{-1}(A)$ is sat, return SAT

Why? Because we found an assignment that (i) both satisfies boolean structure, and (ii) consistent with theory axioms
If $T$-solver decides $B^{-1}(A)$ is sat, return SAT

Why? Because we found an assignment that (i) both satisfies boolean structure, and (ii) consistent with theory axioms

If $B^{-1}(A)$ is unsat, does this mean original formula is UNSAT?
If $\mathcal{T}$-solver decides $B^{-1}(A)$ is sat, return SAT

Why? Because we found an assignment that (i) both satisfies boolean structure, and (ii) consistent with theory axioms

If $B^{-1}(A)$ is unsat, does this mean original formula is UNSAT?

No b/c might be other ways of satisfying boolean structure
SMT Solving: Simplest Version, cont

- If $\mathcal{T}$-solver decides $B^{-1}(A)$ is sat, return $\text{SAT}$

- Why? Because we found an assignment that (i) both satisfies boolean structure, and (ii) consistent with theory axioms

- If $B^{-1}(A)$ is unsat, does this mean original formula is $\text{UNSAT}$?

- No b/c might be other ways of satisfying boolean structure

- In this case, construct new boolean abstraction $B(F) \land \neg A$
SMT Solving: Simplest Version, cont

- If $\mathcal{T}$-solver decides $B^{-1}(A)$ is sat, return SAT

- Why? Because we found an assignment that (i) both satisfies boolean structure, and (ii) consistent with theory axioms

- If $B^{-1}(A)$ is unsat, does this mean original formula is UNSAT?

- No b/c might be other ways of satisfying boolean structure

- In this case, construct new boolean abstraction $B(F) \land \neg A$

- Repeat until we find assignment consistent with theory or until boolean abstraction is unsat
SMT Solving, Simplest Version: Correctness

- If $B^{-1}(A)$ is unsat, construct new abstraction as $B(F) \land \neg A$
SMT Solving, Simplest Version: Correctness

- If $B^{-1}(A)$ is unsat, construct new abstraction as $B(F) \land \neg A$

- Does $B(F) \land \neg A$ still overapproximate satisfiability?
If $B^{-1}(A)$ is unsat, construct new abstraction as $B(F) \land \neg A$

Does $B(F) \land \neg A$ still overapproximate satisfiability?
SMT Solving, Simplest Version: Correctness

- If $B^{-1}(A)$ is unsat, construct new abstraction as $B(F) \land \neg A$

- Does $B(F) \land \neg A$ still overapproximate satisfiability?

- Yes because since $B^{-1}(A)$ is unsat $B^{-1}(\neg A)$ is valid
SMT Solving, Simplest Version: Correctness

- If $B^{-1}(A)$ is unsat, construct new abstraction as $B(F) \land \neg A$

- Does $B(F) \land \neg A$ still overapproximate satisfiability?

- Yes because since $B^{-1}(A)$ is unsat $B^{-1}(-A)$ is valid

- Thus, $F \land B^{-1}(-A)$ is equivalent to $F$
SMT Solving, Simplest Version: Correctness

- If $B^{-1}(A)$ is unsat, construct new abstraction as $B(F) \land \neg A$

- Does $B(F) \land \neg A$ still overapproximate satisfiability?

- Yes because since $B^{-1}(A)$ is unsat $B^{-1}(\neg A)$ is valid

- Thus, $F \land B^{-1}(\neg A)$ is equivalent to $F$

- Hence, $B(F \land B^{-1}(\neg A))$ (i.e., $B(F) \land \neg A$) still overapproximates satisfiability
SMT Solving, Simplest Version: Correctness

- If $B^{-1}(A)$ is unsat, construct new abstraction as $B(F) \land \neg A$

- Does $B(F) \land \neg A$ still overapproximate satisfiability?

- Yes because since $B^{-1}(A)$ is unsat $B^{-1}(\neg A)$ is valid

- Thus, $F \land B^{-1}(\neg A)$ is equivalent to $F$

- Hence, $B(F \land B^{-1}(\neg A))$ (i.e., $B(F) \land \neg A$) still overapproximates satisfiability

- Formulas such as $\neg A$ that are $\mathcal{T}$-valid are called theory conflict clauses
SMT Solving, Simplest Version: Termination

▶ Approach is sound, but is it guaranteed to terminate?
SMT Solving, Simplest Version: Termination

- Approach is sound, but is it guaranteed to terminate? Yes

- Suppose SAT solver gives assignment $A$ s.t. $\mathcal{B}^{-1}(A)$ is unsat
Approach is sound, but is it guaranteed to terminate? Yes

Suppose SAT solver gives assignment $A$ s.t. $B^{-1}(A)$ is unsat

We’ll never obtain same assignment $A$ again because formula next time is $B(F) \land \neg A$
SMT Solving, Simplest Version: Termination

- Approach is sound, but is it guaranteed to terminate? Yes
- Suppose SAT solver gives assignment $A$ s.t. $B^{-1}(A)$ is unsat
- We’ll never obtain same assignment $A$ again because formula next time is $B(F') \land \neg A$
- There are finitely many satisfying assignments to boolean abstraction, and we get different sat assignment every time
SMT Solving, Simplest Version: Termination

- Approach is sound, but is it guaranteed to terminate? Yes

- Suppose SAT solver gives assignment $A$ s.t. $B^{-1}(A)$ is unsat

- We’ll never obtain same assignment $A$ again because formula next time is $B(F) \land \neg A$

- There are finitely many satisfying assignments to boolean abstraction, and we get different sat assignment every time

- Thus, we’ll eventually either find assignment consistent with theory $\Rightarrow$ SAT
SMT Solving, Simplest Version: Termination

- Approach is sound, but is it guaranteed to terminate? Yes

- Suppose SAT solver gives assignment \( A \) s.t. \( B^{-1}(A) \) is unsat

- We’ll never obtain same assignment \( A \) again because formula next time is \( B(F') \land \neg A \)

- There are finitely many satisfying assignments to boolean abstraction, and we get different sat assignment every time

- Thus, we’ll eventually either find assignment consistent with theory \( \Rightarrow SAT \)

- Or all satisfying assignments contradict theory axioms \( \Rightarrow UNSAT \)
Example

- Consider example from before:

\[ F : \quad x = z \land ((y = z \land x \neq z) \lor \neg(x = z)) \]
Example

- Consider example from before:
  \[
  F : \quad x = z \land ((y = z \land x \neq z) \lor \neg(x = z))
  \]
  \[
  B(F) : b_1 \land ((b_2 \land b_3) \lor \neg b_1)
  \]
Consider example from before:

\[ F : x = z \land ((y = z \land x \neq z) \lor \neg(x = z)) \]

\[ B(F) : b_1 \land ((b_2 \land b_3) \lor \neg b_1) \]

Sat assignment to \( B(F) \) \( A : b_1 \land b_2 \land b_3 \)
Example

Consider example from before:

\[ F : \quad x = z \land ((y = z \land x \neq z) \lor \neg(x = z)) \]

\[ B(F) : b_1 \land ((b_2 \land b_3) \lor \neg b_1) \]

Sat assignment to \( B(F) \) \( A : b_1 \land b_2 \land b_3 \)

\( B^{-1}(A) \) is unsat
Consider example from before:

\[ F : x = z \land ((y = z \land x \neq z) \lor \neg(x = z)) \]

\[ B(F) : b_1 \land ((b_2 \land b_3) \lor \neg b_1) \]

Sat assignment to \( B(F) \) \( A : b_1 \land b_2 \land b_3 \)

\( B^{-1}(A) \) is unsat

What is new boolean abstraction?

\[ (b_1 \land ((b_2 \land b_3) \lor \neg b_1)) \land \neg(b_1 \land b_2 \land b_3) \]
Example

- Consider example from before:

  \[ F : x = z \land ((y = z \land x \neq z) \lor \neg(x = z)) \]

- \[ B(F) : b_1 \land ((b_2 \land b_3) \lor \neg b_1) \]

- Sat assignment to \( B(F) \) \( A : b_1 \land b_2 \land b_3 \)

- \( B^{-1}(A) \) is unsat

- What is new boolean abstraction?

  \[ (b_1 \land ((b_2 \land b_3) \lor \neg b_1)) \land \neg(b_1 \land b_2 \land b_3) \]

- Is this formula SAT?
Example

- Consider example from before:

\[ F : x = z \land ((y = z \land x \neq z) \lor \neg(x = z)) \]

- \( \mathcal{B}(F) : b_1 \land ((b_2 \land b_3) \lor \neg b_1) \)

- Sat assignment to \( \mathcal{B}(F) \ A : b_1 \land b_2 \land b_3 \)

- \( \mathcal{B}^{-1}(A) \) is unsat

- What is new boolean abstraction?

\[ (b_1 \land ((b_2 \land b_3) \lor \neg b_1)) \land \neg(b_1 \land b_2 \land b_3) \]

- Is this formula SAT? No, thus original formula UNSAT
Shortcoming of This Approach

- So far, we just add negation of current assignment as theory conflict clause
Shortcoming of This Approach

- So far, we just add negation of current assignment as theory conflict clause
- Unfortunately, conflict clauses obtained this way are too weak
Shortcoming of This Approach

- So far, we just add negation of current assignment as theory conflict clause
- Unfortunately, conflict clauses obtained this way are too weak
- Suppose $A$ is a conjunction of 100 literals such that

\[ B^{-1}(A) = x = y \land x \neq y \land a_1 \land a_2 \land \ldots \land a_{98} \]
Shortcoming of This Approach

- So far, we just add negation of current assignment as theory conflict clause
- Unfortunately, conflict clauses obtained this way are too weak
- Suppose $A$ is a conjunction of 100 literals such that
  \[ B^{-1}(A) = x = y \land x \neq y \land a_1 \land a_2 \land \ldots \land a_{98} \]
- Theory conflict clause $\neg A$ prevents exact same assignment
Shortcoming of This Approach

- So far, we just add negation of current assignment as theory conflict clause

- Unfortunately, conflict clauses obtained this way are too weak

- Suppose $A$ is a conjunction of 100 literals such that

  \[ B^{-1}(A) = x = y \land x \neq y \land a_1 \land a_2 \land \ldots \land a_{98} \]

- Theory conflict clause $\neg A$ prevents exact same assignment

- But it doesn’t prevent many other bad assignments involving $x = y \land x \neq y$ such as:

  \[ B^{-1}(A) = x = y \land x \neq y \land a_1 \land a_2 \land \ldots \land \neg a_{98} \]
Shortcoming of This Approach

- So far, we just add negation of current assignment as theory conflict clause
- Unfortunately, conflict clauses obtained this way are too weak
- Suppose \( A \) is a conjunction of 100 literals such that
  \[
  B^{-1}(A) = x = y \land x \neq y \land a_1 \land a_2 \land \ldots \land a_{98}
  \]
- Theory conflict clause \( \neg A \) prevents exact same assignment
- But it doesn’t prevent many other bad assignments involving \( x = y \land x \neq y \) such as:
  \[
  B^{-1}(A) = x = y \land x \neq y \land a_1 \land a_2 \land \ldots \land \neg a_{98}
  \]
- In fact, there are \( 2^{98} \) unsat assignments containing \( x = y \land x \neq y \) but \( \neg A \) prevents only one of them!
SMT solving, Improvement #1

- Suppose SAT solver makes assignment $A$ s.t. $B^{-1}(A)$ is unsat.
SMT solving, Improvement #1

- Suppose SAT solver makes assignment $A$ s.t. $B^{-1}(A)$ is unsat

- Rather than adding $\neg A$ as a conflict clause, better idea is to find an unsatisfiable core of $B^{-1}(A)$
SMT solving, Improvement #1

- Suppose SAT solver makes assignment $A$ s.t. $B^{-1}(A)$ is unsat.

- Rather than adding $\neg A$ as a conflict clause, better idea is to find an unsatisfiable core of $B^{-1}(A)$.

- An unsatisfiable core $C$ of $A$ contains a subset of atoms in $A$ and $B^{-1}(C)$ is still unsatisfiable.
Suppose SAT solver makes assignment $A$ s.t. $B^{-1}(A)$ is unsat.

Rather than adding $\neg A$ as a conflict clause, better idea is to find an unsatisfiable core of $B^{-1}(A)$.

An unsatisfiable core $C$ of $A$ contains a subset of atoms in $A$ and $B^{-1}(C)$ is still unsatisfiable.

Ideally, we would like to find the minimal unsatisfiable core.
SMT solving, Improvement #1

- Suppose SAT solver makes assignment $A$ s.t. $B^{-1}(A)$ is unsat

- Rather than adding $\neg A$ as a conflict clause, better idea is to find an unsatisfiable core of $B^{-1}(A)$

- An unsatisfiable core $C$ of $A$ contains a subset of atoms in $A$ and $B^{-1}(C)$ is still unsatisfiable.

- Ideally, we would like to find the minimal unsatisfiable core

- Minimal unsatisfiable core $C^*$ has property that if you drop any single atom of $C^*$, result is satisfiable
SMT solving, Improvement #1

▶ Suppose SAT solver makes assignment $A$ s.t. $B^{-1}(A)$ is unsat

▶ Rather than adding $\neg A$ as a conflict clause, better idea is to find an unsatisfiable core of $B^{-1}(A)$

▶ An unsatisfiable core $C$ of $A$ contains a subset of atoms in $A$ and $B^{-1}(C)$ is still unsatisfiable.

▶ Ideally, we would like to find the minimal unsatisfiable core

▶ Minimal unsatisfiable core $C^*$ has property that if you drop any single atom of $C^*$, result is satisfiable

▶ What is a minimal unsat core of $x = y \land x \neq y \land a_1 \land a_2 \land \ldots \land a_{98}$?
SMT solving, Improvement #1

- Suppose SAT solver makes assignment $A$ s.t. $B^{-1}(A)$ is unsat.

- Rather than adding $\neg A$ as a conflict clause, better idea is to find an unsatisfiable core of $B^{-1}(A)$.

- An unsatisfiable core $C$ of $A$ contains a subset of atoms in $A$ and $B^{-1}(C)$ is still unsatisfiable.

- Ideally, we would like to find the minimal unsatisfiable core.

- Minimal unsatisfiable core $C^*$ has property that if you drop any single atom of $C^*$, result is satisfiable.

- What is a minimal unsat core of $x = y \land x \neq y \land a_1 \land a_2 \land \ldots \land a_{98}$?
  
  $x = y \land x \neq y$
Computing Minimal Unsat Core

How can we compute minimal unsat core of conjunctive $T$ formula without modifying theory solver?
Computing Minimal Unsat Core

- How can we compute minimal unsat core of conjunctive $\mathcal{T}$ formula without modifying theory solver?

- Let $\phi$ be original unsatisfiable conjunct
Computing Minimal Unsat Core

- How can we compute minimal unsat core of conjunctive $\mathcal{T}$ formula without modifying theory solver?

- Let $\phi$ be original unsatisfiable conjunct

- Drop one atom from $\phi$, call this $\phi'$
Computing Minimal Unsat Core

- How can we compute minimal unsat core of conjunctive $\mathcal{T}$ formula without modifying theory solver?

- Let $\phi$ be original unsatisfiable conjunct

- Drop one atom from $\phi$, call this $\phi'$

- If $\phi'$ is still unsat, $\phi := \phi'$
Computing Minimal Unsat Core

- How can we compute minimal unsat core of conjunctive $\mathcal{T}$ formula without modifying theory solver?
- Let $\phi$ be original unsatisfiable conjunct
- Drop one atom from $\phi$, call this $\phi'$
- If $\phi'$ is still unsat, $\phi := \phi'$
- Repeat this for every atom in $\phi$
Computing Minimal Unsat Core

- How can we compute minimal unsat core of conjunctive $\mathcal{T}$ formula without modifying theory solver?

- Let $\phi$ be original unsatisfiable conjunct

- Drop one atom from $\phi$, call this $\phi'$

- If $\phi'$ is still unsat, $\phi := \phi'$

- Repeat this for every atom in $\phi$

- Clearly, resulting $\phi$ is minimal unsat core of original formula
Example

- Let’s compute minimal unsat core of

\[ \phi : x = y \land f(x) + z = 5 \land f(x) \neq f(y) \land y \leq 3 \]
Example

Let's compute minimal unsat core of

$$\phi: x = y \land f(x) + z = 5 \land f(x) \neq f(y) \land y \leq 3$$

Drop $x = y$ from $\phi$. Is result unsat?
Example

Let’s compute minimal unsat core of

\[ \phi : x = y \land f(x) + z = 5 \land f(x) \neq f(y) \land y \leq 3 \]

Drop \( x = y \) from \( \phi \). Is result unsat? no, so keep \( x = y \)
Example

Let’s compute minimal unsat core of

\[ \phi : x = y \land f(x) + z = 5 \land f(x) \neq f(y) \land y \leq 3 \]

Drop \( x = y \) from \( \phi \). Is result unsat? no, so keep \( x = y \)

Drop \( f(x) + z = 5 \). Is result unsat? yes, so drop \( f(x) + z = 5 \)

Finally, drop \( y \leq 3 \). Is result unsat? yes, drop \( y \leq 3 \)
Example

Let’s compute minimal unsat core of

$$\phi: x = y \land f(x) + z = 5 \land f(x) \neq f(y) \land y \leq 3$$

Drop $x = y$ from $\phi$. Is result unsat? no, so keep $x = y$

Drop $f(x) + z = 5$. Is result unsat? yes, so drop $f(x) + z = 5$

New formula: $\phi: x = y \land f(x) \neq f(y) \land y \leq 3$
Example

Let’s compute minimal unsat core of

\[ \phi : \ x = y \land f(x) + z = 5 \land f(x) \neq f(y) \land y \leq 3 \]

Drop \( x = y \) from \( \phi \). Is result unsat? no, so keep \( x = y \)

Drop \( f(x) + z = 5 \). Is result unsat? yes, so drop \( f(x) + z = 5 \)

New formula: \( \phi : \ x = y \land f(x) \neq f(y) \land y \leq 3 \)

Drop \( f(x) \neq f(y) \). Is result unsat?
Example

Let’s compute minimal unsat core of

\[ \phi : x = y \land f(x) + z = 5 \land f(x) \neq f(y) \land y \leq 3 \]

Drop \( x = y \) from \( \phi \). Is result unsat? no, so keep \( x = y \)

Drop \( f(x) + z = 5 \). Is result unsat? yes, so drop \( f(x) + z = 5 \)

New formula: \( \phi : x = y \land f(x) \neq f(y) \land y \leq 3 \)

Drop \( f(x) \neq f(y) \). Is result unsat? no, keep \( f(x) \neq f(y) \)
Example

Let’s compute minimal unsat core of

\[ \phi : x = y \land f(x) + z = 5 \land f(x) \neq f(y) \land y \leq 3 \]

- Drop \( x = y \) from \( \phi \). Is result unsat? no, so keep \( x = y \)

- Drop \( f(x) + z = 5 \). Is result unsat? yes, so drop \( f(x) + z = 5 \)

- New formula: \( \phi : x = y \land f(x) \neq f(y) \land y \leq 3 \)

- Drop \( f(x) \neq f(y) \). Is result unsat? no, keep \( f(x) \neq f(y) \)

- Finally, drop \( y \leq 3 \). Is result unsat?
Example

Let’s compute minimal unsat core of

\[ \phi : x = y \land f(x) + z = 5 \land f(x) \neq f(y) \land y \leq 3 \]

Drop \( x = y \) from \( \phi \). Is result unsat? \( \text{no, so keep } x = y \)

Drop \( f(x) + z = 5 \). Is result unsat? \( \text{yes, so drop } f(x) + z = 5 \)

New formula: \( \phi : x = y \land f(x) \neq f(y) \land y \leq 3 \)

Drop \( f(x) \neq f(y) \). Is result unsat? \( \text{no, keep } f(x) \neq f(y) \)

Finally, drop \( y \leq 3 \). Is result unsat? \( \text{yes, drop } y \leq 3 \)
Example

Let’s compute minimal unsat core of

\[ \phi : x = y \land f(x) + z = 5 \land f(x) \neq f(y) \land y \leq 3 \]

• Drop \( x = y \) from \( \phi \). Is result unsat? no, so keep \( x = y \)

• Drop \( f(x) + z = 5 \). Is result unsat? yes, so drop \( f(x) + z = 5 \)

• New formula: \( \phi : x = y \land f(x) \neq f(y) \land y \leq 3 \)

• Drop \( f(x) \neq f(y) \). Is result unsat? no, keep \( f(x) \neq f(y) \)

• Finally, drop \( y \leq 3 \). Is result unsat? yes, drop \( y \leq 3 \)

• What is minimal unsat core?
Example

Let’s compute minimal unsat core of

$$\phi : x = y \land f(x) + z = 5 \land f(x) \neq f(y) \land y \leq 3$$

- Drop $x = y$ from $\phi$. Is result unsat? no, so keep $x = y$
- Drop $f(x) + z = 5$. Is result unsat? yes, so drop $f(x) + z = 5$

New formula: $\phi : x = y \land f(x) \neq f(y) \land y \leq 3$

- Drop $f(x) \neq f(y)$. Is result unsat? no, keep $f(x) \neq f(y)$
- Finally, drop $y \leq 3$. Is result unsat? yes, drop $y \leq 3$

- What is minimal unsat core? $x = y \land f(x) \neq f(y)$
SMT Solving Using Unsat Cores

- Given formula $F$, construct boolean abstraction $B(F)$
SMT Solving Using Unsat Cores

- Given formula $F$, construct boolean abstraction $B(F)$
- Use SAT solver to decide if $B(F)$ is unsat; if so $F$ also unsat
SMT Solving Using Unsat Cores

- Given formula $F$, construct boolean abstraction $B(F)$
- Use SAT solver to decide if $B(F)$ is unsat; if so $F$ also unsat
- Otherwise, get satisfying assignment $A$ to $B(F)$
SMT Solving Using Unsat Cores

- Given formula $F$, construct boolean abstraction $\mathcal{B}(F)$
- Use SAT solver to decide if $\mathcal{B}(F)$ is unsat; if so $F$ also unsat
- Otherwise, get satisfying assignment $A$ to $\mathcal{B}(F)$
- Query theory solver if $\mathcal{B}^{-1}(A)$ is sat; if so $F$ is sat
SMT Solving Using Unsat Cores

- Given formula $F$, construct boolean abstraction $B(F)$
- Use SAT solver to decide if $B(F)$ is unsat; if so $F$ also unsat
- Otherwise, get satisfying assignment $A$ to $B(F)$
- Query theory solver if $B^{-1}(A)$ is sat; if so $F$ is sat
- Otherwise, compute minimal unsat core $C$ of $B^{-1}(A)$
SMT Solving Using Unsat Cores

- Given formula $F$, construct boolean abstraction $B(F)$
- Use SAT solver to decide if $B(F)$ is unsat; if so $F$ also unsat
- Otherwise, get satisfying assignment $A$ to $B(F)$
- Query theory solver if $B^{-1}(A)$ is sat; if so $F$ is sat
- Otherwise, compute minimal unsat core $C$ of $B^{-1}(A)$
- Use $\neg C$ as theory conflict clause
SMT Solving Using Unsat Cores

- Given formula $F$, construct boolean abstraction $B(F)$
- Use SAT solver to decide if $B(F)$ is unsat; if so $F$ also unsat
- Otherwise, get satisfying assignment $A$ to $B(F)$
- Query theory solver if $B^{-1}(A)$ is sat; if so $F$ is sat
- Otherwise, compute minimal unsat core $C$ of $B^{-1}(A)$
- Use $\neg C$ as theory conflict clause
- i.e., construct new boolean abstraction as $B(F \land \neg C)$
SMT Solving Using Unsat Cores

- Given formula $F$, construct boolean abstraction $\mathcal{B}(F)$
- Use SAT solver to decide if $\mathcal{B}(F)$ is unsat; if so $F$ also unsat
- Otherwise, get satisfying assignment $A$ to $\mathcal{B}(F)$
- Query theory solver if $\mathcal{B}^{-1}(A)$ is sat; if so $F$ is sat
- Otherwise, compute minimal unsat core $C$ of $\mathcal{B}^{-1}(A)$
- Use $\neg C$ as theory conflict clause
- i.e., construct new boolean abstraction as $\mathcal{B}(F \land \neg C)$
- Repeat until we decide sat or unsat
This strategy is much better than simplest strategy where we add $B^{-1}(A)$ as theory conflict clause.
Discussion

- This strategy is much better than simplest strategy where we add $B^{-1}(A)$ as theory conflict clause.
- Using simple strategy, we block just one assignment
Discussion

- This strategy is much better than simplest strategy where we add $B^{-1}(A)$ as theory conflict clause.

- Using simple strategy, we block just one assignment

- Using minimal unsat cores, we block many assignments using one theory conflict clause
Discussion

- This strategy is much better than simplest strategy where we add $B^{-1}(A)$ as theory conflict clause.

- Using simple strategy, we block just one assignment

- Using minimal unsat cores, we block many assignments using one theory conflict clause

- However, our strategy still not ideal because it waits for full assignment to boolean abstraction to generate conflict clause
Motivation for Integration with DPLL

Consider very large formula $F$ containing $x = y$ and $x \neq y$ with corresponding boolean variables $b_1$ and $b_2$.
Motivation for Integration with DPLL

- Consider very large formula $F$ containing $x = y$ and $x \neq y$ with corresponding boolean variables $b_1$ and $b_2$

- Also, suppose $\mathcal{B}(F)$ contains hundreds of boolean variables
Motivation for Integration with DPLL

- Consider very large formula $F$ containing $x = y$ and $x \neq y$ with corresponding boolean variables $b_1$ and $b_2$

- Also, suppose $\mathcal{B}(F)$ contains hundreds of boolean variables

- As soon as sat solver makes assignment $b_1 = \top$, $b_2 = \top$, we are doomed because this is unsatisfiable in theory
Motivation for Integration with DPLL

- Consider very large formula $F$ containing $x = y$ and $x \neq y$ with corresponding boolean variables $b_1$ and $b_2$

- Also, suppose $B(F)$ contains hundreds of boolean variables

- As soon as sat solver makes assignment $b_1 = \top, b_2 = \top$, we are doomed because this is unsatisfiable in theory

- Thus, no need to continue with SAT solving after this bad partial assignment
Motivation for Integration with DPLL

- Consider very large formula $F$ containing $x = y$ and $x \neq y$ with corresponding boolean variables $b_1$ and $b_2$

- Also, suppose $B(F)$ contains hundreds of boolean variables

- As soon as sat solver makes assignment $b_1 = \top$, $b_2 = \top$, we are doomed because this is unsatisfiable in theory

- Thus, no need to continue with SAT solving after this bad partial assignment

- Idea: Don’t use SAT solver as “blackbox”
Motivation for Integration with DPLL

- Consider very large formula \( F \) containing \( x = y \) and \( x \neq y \) with corresponding boolean variables \( b_1 \) and \( b_2 \)

- Also, suppose \( \mathcal{B}(F) \) contains hundreds of boolean variables

- As soon as sat solver makes assignment \( b_1 = \top, b_2 = \top \), we are doomed because this is unsatisfiable in theory

- Thus, no need to continue with SAT solving after this bad partial assignment

- **Idea:** Don’t use SAT solver as “blackbox”

- Instead, integrate theory solver right into the DPLL algorithm
DPLL-Based SAT Solver Architecture

Idea: Integrate theory solver right into this SAT solving loop!

Vijay Ganesh (Original notes from Isil Dillig), ECE750T-28: Computer-aided Reasoning for Software Engineering Lecture 17: SMT Solvers and the DPPL(\text{T}) Framework
DPLL-Based SAT Solver Architecture

Decide

BCP

SAT

no conflict

backtrack
if \( d > 0 \)

Analyze Conflict

UNSAT

▶ Idea: Integrate theory solver right into this SAT solving loop!
DPLL(\(T\)) Framework

- Decide
- BCP
- Analyze Conflict
- Theory Solve

No conflict, theory propagation lemma(s)

SAT

Backtrack if \(d > 0\)

Conflict

C(A)

Conflict clause

UNSAT
DPLL(\(T\)) Framework

- Combination of DPLL-based SAT solver and decision procedure for conjunctive \(T\) formula called **DPLL(\(T\)) framework**
DPLL(\(\mathcal{T}\)) Framework

- Suppose SAT solver has made assignment in Decide step and performed BCP
DPLL(\(\mathcal{T}\)) Framework

- Suppose SAT solver has made assignment in Decide step and performed BCP
- If no conflict detected, immediately invoke theory solver
DPLL(\(T\)) Framework

- Suppose SAT solver has made assignment in Decide step and performed BCP
- If no conflict detected, immediately invoke theory solver
- Specifically, suppose \(A\) is current partial assignment to boolean abstraction
DPLL(\(\mathcal{T}\)) Framework

- Suppose SAT solver has made assignment in Decide step and performed BCP
- If no conflict detected, immediately invoke theory solver
- Specifically, suppose \(A\) is current partial assignment to boolean abstraction
- Use theory solver to decide if \(B^{-1}(A)\) is unsat
DPLL(\(\mathcal{T}\)) Framework

- Suppose SAT solver has made assignment in Decide step and performed BCP

- If no conflict detected, immediately invoke \textit{theory solver}

- Specifically, suppose \(A\) is current \textit{partial assignment} to boolean abstraction

- Use theory solver to decide if \(B^{-1}(A)\) is unsat

- If \(B^{-1}(A)\) unsat, add theory conflict clause \(\neg A\) to clause database
DPLL(\mathcal{T}) Framework

- Suppose SAT solver has made assignment in Decide step and performed BCP

- If no conflict detected, immediately invoke theory solver

- Specifically, suppose $A$ is current partial assignment to boolean abstraction

- Use theory solver to decide if $B^{-1}(A)$ is unsat

- If $B^{-1}(A)$ unsat, add theory conflict clause $\neg A$ to clause database

- Or better, add negation of unsat core of $A$ to clause database
DPLL(\(\mathcal{T}\)) Framework

- Decide
  - SAT
  - Conflict
    - no conflict, theory propagation lemma(s)
    - Conflict
      - Analyze Conflict
        - backtrack if \(d > 0\)
        - UNSAT
          - Add theory conflict clause and continue doing BCP, which will detect conflict
    - backtrack
  - Theory Solve
    - C(A)
      - Conflict clause

Vijay Ganesh (Original notes from Isil Dillig), ECE750T-28: Computer-aided Reasoning for Software Engineering Lecture 17: SMT Solvers and the DPLL(\(\mathcal{T}\)) Framework
DPLL(\(\mathcal{T}\)) Framework

- Add theory conflict clause and continue doing BCP, which will detect conflict
- As before, AnalyzeConflict decides what level to backtrack to
Theory Propagation

- What we described so far is sufficient to solve SMT formulas, but we can much better!
Theory Propagation

▶ What we described so far is sufficient to solve SMT formulas, but we can much better!

▶ Suppose original formula contains literals $x = y, y = z, x \neq z$ with corresponding boolean variables $b_1, b_2, b_3$
What we described so far is sufficient to solve SMT formulas, but we can much better!

Suppose original formula contains literals $x = y$, $y = z$, $x \neq z$ with corresponding boolean variables $b_1$, $b_2$, $b_3$

Suppose SAT solver makes partial assignment $b_1 : \top$, $b_2 : \top$
Theory Propagation

► What we described so far is sufficient to solve SMT formulas, but we can much better!

► Suppose original formula contains literals $x = y$, $y = z$, $x \neq z$ with corresponding boolean variables $b_1, b_2, b_3$

► Suppose SAT solver makes partial assignment $b_1 : \top$, $b_2 : \top$

► In next Decide step, free to assign $b_3 : \top$ or $b_3 : \bot$
Theory Propagation

- What we described so far is sufficient to solve SMT formulas, but we can much better!

- Suppose original formula contains literals $x = y, y = z, x \neq z$ with corresponding boolean variables $b_1, b_2, b_3$

- Suppose SAT solver makes partial assignment $b_1 : \top, b_2 : \top$

- In next Decide step, free to assign $b_3 : \top$ or $b_3 : \bot$

- But assignment $b_3 : \top$ is sub-optimal, b/c will lead to conflict in $T$
Theory Propagation Lemma, cont

- **Idea:** Theory solver can communicate which literals are implied by current partial assignment.

In our example, \(\neg x \neq z\) implied by current partial assignment \(x = y \land y = z\). Thus, can safely add \(b_1 \land b_2 \rightarrow b_3\) to clause database.

These kinds of clauses implied by theory are called theory propagation lemmas.
Theory Propagation Lemma, cont

- **Idea**: Theory solver can communicate which literals are implied by current partial assignment

- In our example, \( \neg x \neq z \) implied by current partial assignment
  
  \[ x = y \land y = z \]

Thus, can safely add \( b_1 \land b_2 \rightarrow b_3 \) to clause database

These kinds of clauses implied by theory are called theory propagation lemmas
Theory Propagation Lemma, cont

➤ **Idea:** Theory solver can communicate which literals are implied by current partial assignment

➤ In our example, $\neg x \neq z$ implied by current partial assignment

\[ x = y \land y = z \]

➤ Thus, can safely add $b_1 \land b_2 \rightarrow b_3$ to clause database
Theory Propagation Lemma, cont

- **Idea:** Theory solver can communicate which literals are implied by current partial assignment

- In our example, \( \neg x \neq z \) implied by current partial assignment
  
  \[ x = y \land y = z \]

- Thus, can safely add \( b_1 \land b_2 \rightarrow b_3 \) to clause database

- These kinds of clauses implied by theory are called theory propagation lemmas
DPLL(\mathcal{T}) Framework

- After adding theory propagation lemma, continue doing BCP
DPLL(\(\mathcal{T}\)) Framework

- After adding theory propagation lemma, continue doing BCP
- Adding theory propagation lemmas prevents bad assignments to boolean abstraction
Inferring Theory Propagation Lemmas

- How do we obtain theory propagation lemmas?
Inferring Theory Propagation Lemmas

- How do we obtain theory propagation lemmas?

- **Option #1:** Treat theory solver as blackbox, query whether particular literal $a$ is implied by current partial assignment?
Inferring Theory Propagation Lemmas

- How do we obtain theory propagation lemmas?

- **Option #1**: Treat theory solver as blackbox, query whether particular literal \( a \) is implied by current partial assignment?

- **Option #2**: Modify theory solver so that it can figure out implied literals
Inferring Theory Propagation Lemmas

How do we obtain theory propagation lemmas?

Option #1: Treat theory solver as blackbox, query whether particular literal \( a \) is implied by current partial assignment?

Option #2: Modify theory solver so that it can figure out implied literals

Second option is considered more efficient, but have to figure out how to do this for each different theory
Which Theory Propagation Lemmas to Add

- Which theory propagation lemmas do we add?

Option #1: Figure out and add all literals implied by current partial assignment; called exhaustive theory propagation.

Option #2: Only figure out literals "obviously" implied by current partial assignment.

Exhaustive theory propagation can be very expensive; second option considered preferable.

There isn't much of a science behind which literals are "obviously" implied.

Solvers use different strategies to obtain simple-to-find implications.
Which Theory Propagation Lemmas to Add

► Which theory propagation lemmas do we add?

► **Option #1**: Figure out and add all literals implied by current partial assignment; called **exhaustive theory propagation**
Which Theory Propagation Lemmas to Add

- Which theory propagation lemmas do we add?

- **Option #1:** Figure out and add all literals implied by current partial assignment; called exhaustive theory propagation

- **Option #2:** Only figure out literals "obviously" implied by current partial assignment

Exhaustive theory propagation can be very expensive; second option considered preferable. There isn't much of a science behind which literals are "obviously" implied. Solvers use different strategies to obtain simple-to-find implications.
Which Theory Propagation Lemmas to Add

- Which theory propagation lemmas do we add?
  - **Option #1**: Figure out and add all literals implied by current partial assignment; called exhaustive theory propagation
  - **Option #2**: Only figure out literals “obviously” implied by current partial assignment

- Exhaustive theory propagation can be very expensive; second option considered preferable
Which Theory Propagation Lemmas to Add

- Which theory propagation lemmas do we add?

- **Option #1:** Figure out and add all literals implied by current partial assignment; called **exhaustive theory propagation**

- **Option #2:** Only figure out literals “obviously” implied by current partial assignment

- Exhaustive theory propagation can be very expensive; second option considered preferable

- There isn’t much of a science behind which literals are “obviously” implied
Which Theory Propagation Lemmas to Add

- Which theory propagation lemmas do we add?

- **Option #1:** Figure out and add all literals implied by current partial assignment; called exhaustive theory propagation

- **Option #2:** Only figure out literals “obviously” implied by current partial assignment

- Exhaustive theory propagation can be very expensive; second option considered preferable

- There isn’t much of a science behind which literals are “obviously” implied

- Solvers use different strategies to obtain simple-to-find implications
Summary

- SMT solvers decide satisfiability in boolean combinations of different theories

Instead of converting to DNF, they handle boolean structure using SAT solving techniques. The most common approach is to construct boolean abstraction and lazily infer theory conflict clauses. To do this, one can either consider SAT solver as blackbox or can integrate with it. The latter strategy is considered superior and known as DPLL($T$) framework.
Summary

- SMT solvers decide satisfiability in boolean combinations of different theories
- Instead of converting to DNF, they handle boolean structure using SAT solving techniques
Summary

- SMT solvers decide satisfiability in boolean combinations of different theories

- Instead of converting to DNF, they handle boolean structure using SAT solving techniques

- Most common approach is to construct boolean abstraction and lazily infer theory conflict clauses
Summary

- SMT solvers decide satisfiability in boolean combinations of different theories

- Instead of converting to DNF, they handle boolean structure using SAT solving techniques

- Most common approach is to construct boolean abstraction and lazily infer theory conflict clauses

- To do this, can either consider SAT solver as blackbox or can integrate with it
Summary

- SMT solvers decide satisfiability in boolean combinations of different theories

- Instead of converting to DNF, they handle boolean structure using SAT solving techniques

- Most common approach is to construct boolean abstraction and lazily infer theory conflict clauses

- To do this, can either consider SAT solver as blackbox or can integrate with it

- Latter strategy considered superior and known as DPLL(\(\mathcal{T}\)) framework