Survey of Satisfiability Modulo Theories (SMT)

Albert Oliveras
Technical University of Catalonia (BarcelonaTech)

Theoretical Foundations of Applied SAT Solving
BIRS Workshop
Banff, Canada
January 22nd, 2014
Overview of the talk

- Motivation
- SMT
- Theories of Interest
- History of SMT
- Eager approach
- Lazy approach
  - Optimizations
  - Theory propagation
  - Conflict analysis in DPLL($T$)
  - Combining Theory Solvers
  - Eager vs Lazy
Introduction

- Historically, automated reasoning $\equiv$ uniform proof-search procedures for FO logic

- Limited success: is FO logic the best compromise between expressivity and efficiency?

- Current trend [Sha02] is to gain efficiency by:
  - addressing only (expressive enough) decidable fragments of a certain logic
  - incorporate domain-specific reasoning, e.g:
    - arithmetic reasoning
    - equality
    - data structures (arrays, lists, stacks, ...)

Survey of Satisfiability Modulo Theories (SMT) – p. 3
Examples of this recent trend:

- **SAT**: use propositional logic as the formalization language
  - + high degree of efficiency
  - - expressive (all NP-complete) but involved encodings

- **SMT**: propositional logic + domain-specific reasoning
  - + improves the expressivity
  - - certain (but acceptable) loss of efficiency

**GOAL OF THIS TALK:**
introduce **SMT**, with its main techniques
Overview of the talk

- Motivation
- **SMT**
  - Theories of Interest
  - History of SMT
  - Eager approach
  - Lazy approach
    - Optimizations
    - Theory propagation
    - Conflict analysis in DPLL($T$)
    - Combining Theory Solvers
    - Eager vs Lazy
Need and Applications of SMT

Some problems are more naturally expressed in other logics than propositional logic, e.g:

- Software verification needs reasoning about equality, arithmetic, data structures, pointers, functions calls, ...

SMT consists of deciding the satisfiability of a (ground) FO formula with respect to a background theory

Example ( Equality with Uninterpreted Functions – EUF ):

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

Wide range of applications:

- Predicate abstraction [LNO06]
- Model checking [AMP06]
- Scheduling [BNO+08b]
- Test generation [TdH08]
- ...
Overview of the talk

- Motivation
- SMT

**Theories of Interest**

- History of SMT
- Eager approach
- Lazy approach
  - Optimizations
  - Theory propagation
  - Conflict analysis in DPLL($T$)
  - Combining Theory Solvers
  - Eager vs Lazy
Equality with Uninterpreted Functions, i.e. “=” is equality

If background logic is FO with equality, EUF is empty theory

Consider formula

\[ a \ast (f(b) + f(c)) = d \land b \ast (f(a) + f(c)) \neq d \land a = b \]
Theories of Interest - EUF [BD94, NO80, NO07]

- Equality with Uninterpreted Functions, i.e. “=” is equality

- If background logic is FO with equality, EUF is empty theory

- Consider formula

\[ a \ast (f(b) + f(c)) = d \land b \ast (f(a) + f(c)) \neq d \land a = b \]

- Formula is UNSAT, but no arithmetic reasoning is needed

- If we abstract the formula into

\[ h(a, g(f(b), f(c))) = d \land h(b, g(f(a), f(c))) \neq d \land a = b \]

it is still UNSAT

- EUF is used to abstract non-supported constructions, e.g:
  - Non-linear multiplication
  - ALUs in circuits
Theories of Interest - Arithmetic

- Very **useful** for **obvious reasons**

- **Restricted** fragments support **more efficient** methods:
  - **Bounds**: \( x \bowtie k \) with \( \bowtie \in \{ <, >, \leq, \geq, = \} \)
  - **Difference logic**: \( x - y \bowtie k \), with \( \bowtie \in \{ <, >, \leq, \geq, = \} \)
    - \([\text{NO05}, \text{WIGG05}, \text{SM06}]\)
  - **UTVPI**: \( \pm x \pm y \bowtie k \), with \( \bowtie \in \{ <, >, \leq, \geq, = \} \)
    - \([\text{LM05}]\)
  - **Linear arithmetic**, e.g: \( 2x - 3y + 4z \leq 5 \)
    - \([\text{DdM06}]\)
  - **Non-linear arithmetic**, e.g: \( 2xy + 4xz^2 - 5y \leq 10 \)
    - \([\text{BLNM}^{+09}, \text{ZM10}]\)
  - Variables are either **reals** or **integers**
Two interpreted function symbols \( \text{read} \) and \( \text{write} \).

Theory is axiomatized by:
- \( \forall a \forall i \forall v \ (\text{read}(\text{write}(a, i, v), i) = v) \)
- \( \forall a \forall i \forall j \forall v \ (i \neq j \rightarrow \text{read}(\text{write}(a, i, v), j) = \text{read}(a, j)) \)

Sometimes extensionality is added:
- \( \forall a \forall b \ ((\forall i (\text{read}(a, i) = \text{read}(b, i))) \rightarrow a = b) \)

Is the following set of literals satisfiable?
\[
\begin{align*}
\text{write}(a, i, x) & \neq b \\
\text{read}(b, i) & = y \\
\text{read}(\text{write}(b, i, x), j) & = y \\
 a & = b \\
i & = j
\end{align*}
\]

Used for:
- Software verification
- Hardware verification (memories)
Constants represent vectors of bits

Useful both for hardware and software verification

Different type of operations:
- **String**-like operations: concat, extract, ...
- **Logical** operations: bit-wise not, or, and, ...
- **Arithmetic** operations: add, subtract, multiply, ...

Assume bit-vectors have size 3. Is the formula SAT?

\[
\begin{align*}
a[0 : 1] & \neq b[0 : 1] \land (a|b) = c \land c[0] = 0 \land a[1] + b[1] = 0
\end{align*}
\]
In practice, theories are not isolated

Software verifications needs arithmetic, arrays, bitvectors, ...

Formulas of the following form usually arise:

\[ a = b + 2 \land A = \text{write}(B, a + 1, 4) \land (\text{read}(A, b + 3) = 2 \lor f(a - 1) \neq f(b + 1)) \]

The goal is to combine decision procedures for each theory
GOOD NEWS: efficient decision procedures for sets of ground literals exist for various theories of interest

PROBLEM: in practice, we need to deal with:

1. arbitrary Boolean combinations of literals ($\land, \lor, \neg$)
   (DNF conversion is not a solution in practice)
2. multiple theories
3. quantifiers

We will only focus on (1) and (2), but techniques for (3) exist.
SMT in Practice (2)

- **SMT-LIB**: language, benchmarks, tutorials, ...

- **SMT-COMP**: performance and capabilities of tools

- **SMT Workshop**: held annually, collocated with CADE, CAV, SAT.
Overview of the talk

- Motivation
- SMT
- Theories of Interest

**History of SMT**

- Eager approach
- Lazy approach
  - Optimizations
  - Theory propagation
  - Conflict analysis in DPLL($T$)
  - Combining Theory Solvers
  - Eager vs Lazy
SMT Prehistory - Late 70’s and 80’s

Pioneers:

Influential results:
- Nelson-Oppen congruence closure procedure [NO80]
- Nelson-Oppen combination method [NO79]
- Shostak combination method [Sho84]

Influential systems:
- Nqthm prover [BM90] [Boyer, Moore]
- Simplify [DNS05] [Detlefs, Nelson, Saxe]
Beginnings of SMT - Early 2000s

**KEY FACT:** SAT solvers improved performance

Two ways of exploiting this fact:

- **Eager approach:** encode SMT into SAT
  
  [Bryant, Lahiri, Pnueli, Seshia, Strichman, Velev, ...]
  
  [PRSS99, SSB02, SLB03, BGV01, BV02]
  
  First systems: UCLID [LS04]

- **Lazy approach:** plug SAT solver with a decision procedure
  
  [Armando, Barrett, Castellini, Cimatti, Dill, Giunchiglia, deMoura, Ruess, Sebastiani, Stump,...]
  
  [ACG00, dMR02, BDS02a, ABC+02]
  
  First systems: TSAT [ACG00], ICS [FORS01], CVC [BDS02b], MathSAT [ABC+02]
Overview of the talk

- Motivation
- SMT
- Theories of Interest

**Eager approach**

- Lazy approach
  - Optimizations
  - Theory propagation
  - Conflict analysis in DPLL($T$)
  - Combining Theory Solvers
  - Eager vs Lazy
Eager approach

- **Methodology**: translate problem into equisatisfiable propositional formula and use off-the-shelf SAT solver

- Why “eager”? Search uses all theory information from the beginning

- **Characteristics**:
  - Can use best available SAT solver
  - Sophisticated encodings are needed for each theory

- **Tools**: UCLID, Beaver, Boolector, STP, SONOLAR, Spear, SWORD
Eager approach – Example

Let us consider an EUF formula:

- **First step:** remove function/predicate symbols. Assume we have terms $f(a)$, $f(b)$ and $f(c)$.

  - **Ackermann** reduction:
    - Replace them by fresh constants $A$, $B$ and $C$
    - Add clauses:
      - $a = b \rightarrow A = B$
      - $a = c \rightarrow A = C$
      - $b = c \rightarrow B = C$

  - **Bryant** reduction:
    - Replace $f(a)$ by $A$
    - Replace $f(b)$ by $\text{ite}(b = a, A, B)$
    - Replace $f(c)$ by $\text{ite}(c = a, A, \text{ite}(c = b, B, C))$

Now, atoms are *equalities* between **constants**
Second step: encode formula into propositional logic

- **Small-domain** encoding:
  - If there are \( n \) different constants, there is a model with size at most \( n \log n \)
  - \( \log n \) bits to encode the value of each constant
  - \( a = b \) translated using the bits for \( a \) and \( b \)

- **Per-constraint** encoding:
  - Each atom \( a = b \) is replaced by \( \text{var } P_{a,b} \)
  - Transitivity constraints are added (e.g. \( P_{a,b} \land P_{b,c} \rightarrow P_{a,c} \))

This is a **very rough** overview of an encoding from EUF to SAT.

See [PRSS99, SSB02, SLB03, BGV01, BV02] for details.
Overview of the talk

- Motivation
- SMT
- Theories of Interest
- Eager approach

Lazy approach

- Optimizations
- Theory propagation
- Conflict analysis in DPLL($T$)
- Combining Theory Solvers
- Eager vs Lazy
Lazy approach

Methodology:
Example: consider EUF and the CNF

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

- SAT solver returns model \([1, 2, 4]\)
Lazy approach

Methodology:
Example: consider **EUF** and the CNF

\[
g(a) = c \quad \land \quad \left( f(g(a)) \neq f(c) \quad \lor \quad g(a) = d \right) \quad \land \quad c \neq d
\]

- **SAT solver** returns model \([1, \overline{2}, \overline{4}]\)
- **Theory solver** says **T-inconsistent**
Lazy approach

Methodology:
Example: consider EUF and the CNF

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

- **SAT solver** returns model \([1, \overline{2}, \overline{4}]\)
- **Theory solver** says *T*-inconsistent
- Send \(\{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2 \lor 4\}\) to **SAT solver**
Lazy approach

Methodology:
Example: consider EUF and the CNF

\[
g(a) = c \quad \land \quad (f(g(a)) \neq f(c) \quad \lor \quad g(a) = d) \quad \land \quad c \neq d
\]

- SAT solver returns model \([1, 2, -4]\)
- Theory solver says \(T\)-inconsistent
- Send \(\{1, 2 \lor 3, 4, 1 \lor 2 \lor 4\}\) to SAT solver
- SAT solver returns model \([1, 2, 3, -4]\)
Lazy approach

Methodology:
Example: consider EUF and the CNF

\[ g(a) = c \quad \land \quad \left( f(g(a)) \neq f(c) \quad \lor \quad g(a) = d \right) \quad \land \quad c \neq d \]

- SAT solver returns model \([1, \overline{2}, \overline{4}]\)
- Theory solver says \(T\)-inconsistent
- Send \(\{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2 \lor 4\}\) to SAT solver
- SAT solver returns model \([1, 2, 3, \overline{4}]\)
- Theory solver says \(T\)-inconsistent
Lazy approach

Methodology:
Example: consider EUF and the CNF

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

- SAT solver returns model \([1, \overline{2}, \overline{4}]\)
- Theory solver says \(T\)-inconsistent
- Send \(\{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2 \lor 4\}\) to SAT solver
- SAT solver returns model \([1, 2, 3, \overline{4}]\)
- Theory solver says \(T\)-inconsistent
- SAT solver detects \(\{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2 \lor 4, \overline{1} \lor \overline{2} \lor \overline{3} \lor 4\}\)

UNSATISFIABLE
Lazy approach (2)

- Why “lazy”?  
  Theory information used lazily when checking $T$-consistency of propositional models

- Characteristics:
  + Modular and flexible
  - Theory information does not guide the search

- Tools:
  Alt-Ergo, ArgoLib, Ario, Barcelogic, CVC, DTP, ICS, MathSAT, OpenSMT, Sateen, SVC, Simplify, tSAT, veriT, Yices, Z3, etc...
Overview of the talk

- Motivation
- SMT
- Theories of Interest
- Eager approach
- Lazy approach
  - Optimizations
    - Theory propagation
    - Conflict analysis in DPLL($T$)
    - Combining Theory Solvers
    - Eager vs Lazy
Lazy approach - Optimizations

Several optimizations for enhancing efficiency:

- Check $T$-consistency only of full propositional models
Lazy approach - Optimizations

Several optimizations for enhancing efficiency:

- Check $T$-consistency only of full propositional models.
- Check $T$-consistency of partial assignment while being built.
Lazy approach - Optimizations

Several optimizations for enhancing efficiency:

- Check $T$-consistency only of full propositional models
- Check $T$-consistency of partial assignment while being built
- Given a $T$-inconsistent assignment $M$, add $\neg M$ as a clause
Several optimizations for enhancing efficiency:

- Check $T$-consistency only of full propositional models
- Check $T$-consistency of partial assignment while being built
- Given a $T$-inconsistent assignment $M$, add $\neg M$ as a clause
- Given a $T$-inconsistent assignment $M$, identify a $T$-inconsistent subset $M_0 \subseteq M$ and add $\neg M_0$ as a clause
Several optimizations for enhancing efficiency:

- Check $T$-consistency only of full propositional models.
- Check $T$-consistency of partial assignment while being built.
- Given a $T$-inconsistent assignment $M$, add $\neg M$ as a clause.
- Given a $T$-inconsistent assignment $M$, identify a $T$-inconsistent subset $M_0 \subseteq M$ and add $\neg M_0$ as a clause.
- Upon a $T$-inconsistency, add clause and restart.
Lazy approach - Optimizations

Several optimizations for enhancing efficiency:

- Check $T$-consistency only of full propositional models-
- Check $T$-consistency of partial assignment while being built

- Given a $T$-inconsistent assignment $M$, add $\neg M$ as a clause-
- Given a $T$-inconsistent assignment $M$, identify a $T$-inconsistent subset $M_0 \subseteq M$ and add $\neg M_0$ as a clause

- Upon a $T$-inconsistency, add clause and restart-
- Upon a $T$-inconsistency, bactrack to some point where the assignment was still $T$-consistent
Lazy approach - Important points

Important and beneficial aspects of the lazy approach:
(even with the optimizations)

- **Everyone does what he/she is good at:**
  - **SAT solver** takes care of **Boolean information**
  - **Theory solver** takes care of **theory information**

- **Theory solver only receives** conjunctions of literals

- **Modular approach:**
  - **SAT solver and T-solver communicate** via a **simple API**
  - **SMT for a new theory** only requires **new T-solver**
  - **SAT solver** can be **embedded** in a lazy SMT system with very few new lines of code
Overview of the talk

- Motivation
- SMT
- Theories of Interest
- Eager approach
- Lazy approach
  - Optimizations
  - Theory propagation
    - Conflict analysis in DPLL($T$)
    - Combining Theory Solvers
    - Eager vs Lazy
Lazy approach - $T$-propagation

- As pointed out the lazy approach has one drawback:
  - Theory information does not guide the search (too lazy)

- How can we improve that?
  
  \[ T \text{-Propagate :} \]

\[
M \parallel F \quad \Rightarrow \quad M l \parallel F \quad \text{if} \quad \begin{cases} 
M \models_T l \\
\text{if } l \text{ or } \neg l \text{ occurs in } F \text{ and not in } M 
\end{cases}
\]

- Search guided by $T$-Solver by finding $T$-consequences, instead of only validating it as in basic lazy approach.

- Naive implementation:::
  Add $\neg l$. If $T$-inconsistent then infer $l$ \[\text{[ACG00]}\]
  But for efficient Theory Propagation we need:
  - $T$-Solvers specialized and fast in it.
  - fully exploited in conflict analysis

- This approach has been named \textbf{DPLL}(T) \[\text{[NOT06]}\]
In a nutshell:

\[ \text{DPLL}(T) = \text{DPLL}(X) + T\text{-Solver} \]

- **DPLL(\(X\)):**
  - Very similar to a SAT solver, enumerates Boolean models
  - Not allowed: pure literal, blocked clause elimination, ....
  - Required: incremental addition of clauses
  - Desirable: partial model detection

- **\(T\text{-Solver}\):**
  - Checks *consistency* of conjunctions of literals
  - Computes *theory propagations*
  - Produces explanations of inconsistency/\(T\)-propagation
  - Should be *incremental* and *backtrackable*
Consider again EUF and the formula:

\[
g(a) = c \quad \land \quad (f(g(a)) \neq f(c) \lor g(a) = d) \quad \land \quad c \neq d
\]

\[
0 \parallel 1, \overline{2} \lor 3, \overline{4} \quad \Rightarrow \quad (\text{UnitPropagate})
\]
Consider again EUF and the formula:

\[
\begin{align*}
&g(a) = c \quad \land \quad \left( f(g(a)) \neq f(c) \quad \lor \quad g(a) = d \right) \quad \land \quad c \neq d \\
&\quad \quad \downarrow 1 \quad \downarrow 2 \quad \downarrow 3 \quad \downarrow 4 \\
&0 \quad \| \quad 1, \quad 2 \lor 3, \quad 4 \quad \Rightarrow \quad (\text{UnitPropagate}) \\
&1 \quad \| \quad 1, \quad 2 \lor 3, \quad 4 \quad \Rightarrow \quad (\text{UnitPropagate})
\end{align*}
\]
Consider again EUF and the formula:

\[
\begin{align*}
g(a) = c & \quad \land \quad \left( f(g(a)) \neq f(c) \lor g(a) = d \right) \quad \land \quad c \neq d \\
1 & \quad \land \quad \left( 2 \lor 3 \right) \quad \land \quad 4
\end{align*}
\]

\[
\begin{align*}
0 \quad \parallel 1, \quad 2 \lor 3, \quad 4 & \quad \Rightarrow \quad (\text{UnitPropagate}) \\
1 \quad \parallel 1, \quad 2 \lor 3, \quad 4 & \quad \Rightarrow \quad (\text{UnitPropagate}) \\
1 \quad 4 \quad \parallel 1, \quad 2 \lor 3, \quad 4 & \quad \Rightarrow \quad (T-\text{Propagate})
\end{align*}
\]
Consider again EUF and the formula:

\[ g(a) = c \quad \land \quad (f(g(a)) \neq f(c) \quad \lor \quad g(a) = d) \quad \land \quad c \neq d \]

\[ 0 \quad || \quad 1, \quad \overline{2} \lor 3, \quad \overline{4} \quad \Rightarrow \quad \text{(UnitPropagate)} \]

\[ 1 \quad || \quad 1, \quad \overline{2} \lor 3, \quad \overline{4} \quad \Rightarrow \quad \text{(UnitPropagate)} \]

\[ 1 \quad \overline{4} \quad || \quad 1, \quad \overline{2} \lor 3, \quad \overline{4} \quad \Rightarrow \quad \text{(T-Propagate)} \]

\[ 1 \quad \overline{4} \quad 2 \quad || \quad 1, \quad \overline{2} \lor 3, \quad \overline{4} \quad \Rightarrow \quad \text{(T-Propagate)} \]
Consider again EUF and the formula:

\[
g(a) = c \quad \land \quad \left( (f(g(a)) \neq f(c) \lor g(a) = d \right) \quad \land \quad c \neq d
\]

\[
\begin{align*}
0 \quad \parallel \quad &1, \quad \overline{2} \lor 3, \quad \overline{4} \quad \Rightarrow \quad (\text{UnitPropagate}) \\
1 \quad \parallel \quad &1, \quad \overline{2} \lor 3, \quad \overline{4} \quad \Rightarrow \quad (\text{UnitPropagate}) \\
1 \overline{4} \quad \parallel \quad &1, \quad \overline{2} \lor 3, \quad \overline{4} \quad \Rightarrow \quad (T-\text{Propagate}) \\
1 \overline{4} 2 \quad \parallel \quad &1, \quad \overline{2} \lor 3, \quad \overline{4} \quad \Rightarrow \quad (T-\text{Propagate}) \\
1 \overline{4} 2 \overline{3} \quad \parallel \quad &1, \quad \overline{2} \lor 3, \quad \overline{4} \quad \Rightarrow \quad (\text{Fail})
\end{align*}
\]
Consider again EUF and the formula:

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

\[
\begin{align*}
\emptyset \parallel 1, \overline{2} \lor 3, \overline{4} & \Rightarrow (\text{UnitPropagate}) \\
1 \parallel 1, \overline{2} \lor 3, \overline{4} & \Rightarrow (\text{UnitPropagate}) \\
1 \overline{4} \parallel 1, \overline{2} \lor 3, \overline{4} & \Rightarrow (T\text{-Propagate}) \\
1 \overline{4} \overline{2} \parallel 1, \overline{2} \lor 3, \overline{4} & \Rightarrow (T\text{-Propagate}) \\
1 \overline{4} \overline{2} \overline{3} \parallel 1, \overline{2} \lor 3, \overline{4} & \Rightarrow (\text{Fail})
\end{align*}
\]

UNSAT
DPLL(T) - Overall algorithm

High-level view gives the same algorithm as a CDCL SAT solver:

```java
while (true) {
    while (propagate_gives_conflict()) {
        if (decision_level == 0) return UNSAT;
        else analyze_conflict();
    }
    restart_if_applicable();
    simplify_clause_database_if_applicable();
    if (!decide()) returns SAT; // All vars assigned
}
```

Differences are in:
- `propagate_gives_conflict`
- `analyze_conflict`
propagate_gives_conflict() returns Bool

do {

    // unit propagate
    if ( unit_prop_gives_conflict() ) then return true

    // check T-consistency of the model
    if ( solver.is_model_inconsistent() ) then return true

    // theory propagate
    solver.theory_propagate()

} while (someTheoryPropagation)

return false
DPLL\( (T) \) - Propagation (2)

- Three operations:
  - Unit propagation (SAT solver)
  - Consistency checks (\( T \)-solver)
  - Theory propagation (\( T \)-solver)

- Cheap operations are computed first

- If theory is expensive, calls to \( T \)-solver are sometimes skipped

- For completeness, only necessary to call \( T \)-solver at the leaves (i.e. when we have a full propositional model)

- Theory propagation is not necessary for completeness
Case Reasoning in Theory Solvers

For certain theories, consistency checking requires case reasoning.

**Example:** consider the theory of arrays and the set of literals

\[
\text{read}(\text{write}(A, i, x), j) \neq x \quad \text{read}(\text{write}(A, i, x), j) \neq \text{read}(A, j)
\]
For certain theories, consistency checking requires case reasoning.

Example: consider the theory of arrays and the set of literals

\[ read(write(A, i, x), j) \neq x \quad read(write(A, i, x), j) \neq read(A, j) \]

Two cases:

- \( i = j \). LHS rewrites into \( x \neq x \) !!
Case Reasoning in Theory Solvers

For certain theories, consistency checking requires case reasoning.

Example: consider the theory of arrays and the set of literals

\[ \text{read}(\text{write}(A, i, x), j) \neq x \quad \text{read}(\text{write}(A, i, x), j) \neq \text{read}(A, j) \]

Two cases:

- \( i = j \). LHS rewrites into \( x \neq x \) !!!
- \( i \neq j \). RHS rewrites into \( \text{read}(A, j) \neq \text{read}(A, j) \) !!!
Case Reasoning in Theory Solvers

For certain theories, consistency checking requires case reasoning.

Example: consider the theory of arrays and the set of literals

$$read(write(A, i, x), j) \neq x \quad read(write(A, i, x), j) \neq read(A, j)$$

Two cases:

- $i = j$. LHS rewrites into $x \neq x$ !!!
- $i \neq j$. RHS rewrites into $read(A, j) \neq read(A, j)$ !!!

CONCLUSION: $T$-inconsistent
A complete T-solver might need to reason by cases via internal case splitting and backtracking mechanisms.

An alternative is to lift case splitting and backtracking from the T-Solver to the SAT engine.

Basic idea: encode case splits as sets of clauses and send them as needed to the SAT engine for it to split on them.

Possible benefits:
- All case-splitting is coordinated by the SAT engine
- Only have to implement case-splitting infrastructure in one place
- Can learn a wider class of lemmas
Case Reasoning in Theory Solvers (3)

- **Basic idea:** encode case splits as a set of clauses and send them as needed to the SAT engine.

- **Example:**
  - Assume model contains literal \( s = \text{read}(\text{write}(A, i, t), j) \) if \( s' \) and \( i = j \) or \( s = \text{read}(A, j) \) if \( s' \) and \( i \neq j \).

- **DPLL(\( X \))** asks: “is it \( T \)-satisfiable”?

- **\( T \)-solver** says: “I do not know yet, but it will be helpful that you consider these theory lemmas:”

\[
\begin{align*}
  s = s' \land i = j & \rightarrow s = t \\
  s = s' \land i \neq j & \rightarrow s = \text{read}(A, j)
\end{align*}
\]

- We need certain **completeness conditions** (e.g. once all lits from a certain subset \( \mathcal{L} \) has been decided, the \( T \)-solver should answer YES/NO).
Overview of the talk

- Motivation
- SMT
- Theories of Interest
- Eager approach
- Lazy approach
  - Optimizations
  - Theory propagation
  - Conflict analysis in DPLL(T)
  - Combining Theory Solvers
  - Eager vs Lazy
DPLL\(T\) - Conflict Analysis

Remember conflict analysis in SAT solvers:

\[ C := \text{conflicting clause} \]

\textbf{while} \( C \) contains more than one lit of last DL

\[ l := \text{last literal assigned in } C \]
\[ C := \text{Resolution}(C, \text{reason}(l)) \]

\textbf{end while}

\[
// \text{let } C = C' \lor l \text{ where } l \text{ is UIP backjump(maxDL}(C'))
\]

\text{add } l \text{ to the model with reason } C
\text{learn}(C)
Conflict analysis in DPLL(T):

```latex
\textbf{if} boolean conflict \textbf{then} \ C := conflicting clause \\
\textbf{else} \ C := \neg(\text{solver.explain_inconsistency}()) \\
\textbf{while} \ C \text{ contains more than one lit of last DL} \\
\quad l := \text{last literal assigned in} \ C \\
\quad C := \text{Resolution}(C, \text{reason}(l)) \\
\textbf{end while} \\

// let $C = C' \lor l$ where $l$ is UIP \\
\text{backjump(maxDL}(C')) \\
\text{add } l \text{ to the model with reason } C \\
\text{learn}(C)
```
What does `explain_inconsistency` return?

- A (small) conjunction of literals \( l_1 \land \ldots \land l_n \) such that:
  - They were in the model when \( T \)-inconsistency was found
  - It is \( T \)-inconsistent

What is now `reason(l)`?

- If \( l \) was unit propagated, reason is the clause that propagated it
- If \( l \) was \( T \)-propagated?
  - \( T \)-solver has to provide an explanation for \( l \), i.e.
    a (small) set of literals \( l_1, \ldots, l_n \) such that:
      - They were in the model when \( l \) was \( T \)-propagated
      - \( l_1 \land \ldots \land l_n \models_T l \)
    - Then `reason(l)` is \( \neg l_1 \lor \ldots \lor \neg l_n \lor l \)
DPLL(\(T\)) - Conflict Analysis (4)

Let \(M\) be of the form \(\ldots, c = b, \ldots\) and let \(F\) contain
\[
h(a) = h(c) \lor p \quad a = b \lor \neg p \lor a = d \quad a \neq d \lor a = b
\]

Take the following sequence:

1. **Decide** \(h(a) \neq h(c)\)
2. **UnitPropagate** \(p\) (due to clause \(h(a) = h(c) \lor p\))
3. **T-Propagate** \(a \neq b\) (since \(h(a) \neq h(c)\) and \(c = b\))
4. **UnitPropagate** \(a = d\) (due to clause \(a = b \lor \neg p \lor a = d\))
5. **Conflicting clause** \(a \neq d \lor a = b\)

**Explain** \((a \neq b)\) is \(\{h(a) \neq h(c), c = b\}\)

\[
\begin{align*}
&h(a) = h(c) \lor c \neq b \lor a \neq b \\
&\quad \quad \downarrow \\
&h(a) = h(c) \lor p \\
\end{align*}
\]

\[
\begin{align*}
&h(a) = h(c) \lor c \neq b \lor \neg p \\
&\quad \quad \downarrow \\
&h(a) = h(c) \lor c \neq b
\end{align*}
\]

\[
\begin{align*}
&h(a) = h(c) \lor p \\
\end{align*}
\]

\[
\begin{align*}
&h(a) = h(c) \lor c \neq b \\
&\quad \quad \downarrow \\
&h(a) = h(c) \lor c \neq b
\end{align*}
\]
Unsatisfiability Proofs in SAT

Unsatisfiability is detected when, at decision level zero, unit propagation on input clauses and lemmas finds a falsified clause.

Resolution refutation obtained via:

- Derivation of the empty clause via unit resolution from input clauses and lemmas
- Resolution derivations of lemmas (trivial since lemmas were generated via resolution steps)
Unsatisfiability Proofs in SMT

Unsatisfiability is detected when, at decision level zero, propagation detects a conflict.

Propagation = unit propagation (UP) + $T$-propagation

But $T$-propagation can be seen as UP on a $T$-lemma:

\[
\begin{align*}
    a = b \land b = c &\models_T a = c \text{ is a UP on the } T\text{-lemma} \\
    a \neq b \lor b \neq c \lor a = c
\end{align*}
\]

Conflict = falsified clause or $T$-inconstency.

But a $T$-inconsistency can be seen as a falsified $T$-lemma.

Refutation obtained via:

- Derivation of the empty clause via unit resolution from input clauses, lemmas and $T$-lemmas.
- Resolution derivations of lemmas (trivial since were generated via resolution using maybe $T$-lemmas).
- Proofs of validity of $T$-lemmas (this is $T$-specific).
Overview of the talk

- Motivation
- SMT
- Theories of Interest
- Eager approach
- Lazy approach
  - Optimizations
  - Theory propagation
  - Conflict analysis in DPLL($T$)

**Combining Theory Solvers**

- Eager vs Lazy
Need for combination

- In software verification, formulas like the following one arise:

\[ a = b + 2 \land A = \text{write}(B, a + 1, 4) \land (\text{read}(A, b + 3) = 2 \lor f(a - 1) \neq f(b + 1)) \]

- Here reasoning is needed over
  - The theory of linear arithmetic (\(T_{LA}\))
  - The theory of arrays (\(T_A\))
  - The theory of uninterpreted functions (\(T_{EUF}\))

- Remember that \(T\)-solvers only deal with conjunctions of lits.

- Given \(T\)-solvers for the three individual theories, can we combine them to obtain one for \((T_{LA} \cup T_A \cup T_{EUF})\)?

- Under certain conditions the Nelson-Oppen combination method gives a positive answer
Motivating example - Convex case

Consider the following set of literals:

\[ f(f(x) - f(y)) = a \]
\[ f(0) = a + 2 \]
\[ x = y \]

There are two theories involved: \( \mathbb{T}_{LA(\mathbb{R})} \) and \( \mathbb{T}_{EUF} \)

**FIRST STEP:** purify each literal so that it belongs to a single theory

\[ f(f(x) - f(y)) = a \implies f(e_1) = a \implies f(e_1) = a \]
\[ e_1 = f(x) - f(y) \]
\[ e_1 = e_2 - e_3 \]
\[ e_2 = f(x) \]
\[ e_3 = f(y) \]
Consider the following set of literals:

\[
\begin{align*}
f(f(x) - f(y)) &= a \\
f(0) &= a + 2 \\
x &= y
\end{align*}
\]

There are two theories involved: \( \mathbb{T}_{LA(\mathbb{R})} \) and \( \mathbb{T}_{EUF} \)

**FIRST STEP:** purify each literal so that it belongs to a single theory

\[
\begin{align*}
f(0) &= a + 2 \implies f(e_4) &= a + 2 \implies f(e_4) &= e_5 \\
e_4 &= 0 \\
e_5 &= a + 2
\end{align*}
\]
SECOND STEP: check satisfiability and exchange entailed equalities

<table>
<thead>
<tr>
<th>EUF</th>
<th>Arithmetic</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(e_1) = a$</td>
<td>$e_2 - e_3 = e_1$</td>
</tr>
<tr>
<td>$f(x) = e_2$</td>
<td>$e_4 = 0$</td>
</tr>
<tr>
<td>$f(y) = e_3$</td>
<td>$e_5 = a + 2$</td>
</tr>
<tr>
<td>$f(e_4) = e_5$</td>
<td></td>
</tr>
<tr>
<td>$x = y$</td>
<td></td>
</tr>
</tbody>
</table>

The two solvers only share constants: $e_1, e_2, e_3, e_4, e_5, a$

To merge the two models into a single one, the solvers have to agree on equalities between shared constants (interface equalities)

This can be done by exchanging entailed interface equalities
SECOND STEP: check satisfiability and exchange entailed equalities

\[
\begin{align*}
EUF & \\
\quad f(e_1) &= a \\
\quad f(x) &= e_2 \\
\quad f(y) &= e_3 \\
\quad f(e_4) &= e_5 \\
\quad x &= y
\end{align*}
\]

\[
\begin{align*}
\text{Arithmetic} & \\
\quad e_2 - e_3 &= e_1 \\
\quad e_4 &= 0 \\
\quad e_5 &= a + 2 \\
\quad e_2 &= e_3
\end{align*}
\]

The two solvers only share constants: \( e_1, e_2, e_3, e_4, e_5, a \)

- \( \text{EUF-Solver says SAT} \)
- \( \text{Ari-Solver says SAT} \)
- \( \text{EUF } \models e_2 = e_3 \)
Motivating example - Convex case (2)

SECOND STEP: check satisfiability and exchange entailed equalities

<table>
<thead>
<tr>
<th>EUF</th>
<th>Arithmetic</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(e_1) = a$</td>
<td>$e_2 - e_3 = e_1$</td>
</tr>
<tr>
<td>$f(x) = e_2$</td>
<td>$e_4 = 0$</td>
</tr>
<tr>
<td>$f(y) = e_3$</td>
<td>$e_5 = a + 2$</td>
</tr>
<tr>
<td>$f(e_4) = e_5$</td>
<td>$e_2 = e_3$</td>
</tr>
<tr>
<td>$x = y$</td>
<td></td>
</tr>
<tr>
<td>$e_1 = e_4$</td>
<td></td>
</tr>
</tbody>
</table>

The two solvers only share constants: $e_1, e_2, e_3, e_4, e_5, a$

- **EUF-Solver** says SAT
- **Ari-Solver** says SAT
- **Ari** $\models e_1 = e_4$
Motivating example - Convex case (2)

SECOND STEP: check satisifiability and exchange entailed equalities

<table>
<thead>
<tr>
<th>EUF</th>
<th>Arithmetic</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(e_1) = a$</td>
<td>$e_2 - e_3 = e_1$</td>
</tr>
<tr>
<td>$f(x) = e_2$</td>
<td>$e_4 = 0$</td>
</tr>
<tr>
<td>$f(y) = e_3$</td>
<td>$e_5 = a + 2$</td>
</tr>
<tr>
<td>$f(e_4) = e_5$</td>
<td>$e_2 = e_3$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>$a = e_5$</td>
</tr>
<tr>
<td>$e_1 = e_4$</td>
<td></td>
</tr>
</tbody>
</table>

The two solvers only share constants: $e_1, e_2, e_3, e_4, e_5, a$

- **EUF**-Solver says SAT
- **Ari**-Solver says SAT
- **EUF** $\models a = e_5$
Motivating example - Convex case (2)

SECOND STEP: check satisfiability and exchange entailed equalities

\[
\begin{align*}
EUF & \\
\quad f(e_1) & = a \\
\quad f(x) & = e_2 \\
\quad f(y) & = e_3 \\
\quad f(e_4) & = e_5 \\
\quad x & = y \\
\quad e_1 & = e_4 \\
\end{align*}
\]

\[
\begin{align*}
\quad Arithmetic & \\
\quad e_2 - e_3 & = e_1 \\
\quad e_4 & = 0 \\
\quad e_5 & = a + 2 \\
\quad e_2 & = e_3 \\
\quad a & = e_5 \\
\end{align*}
\]

The two solvers only share constants: \(e_1, e_2, e_3, e_4, e_5, a\)

- \(EUF\)-Solver says SAT
- Ari-Solver says \textbf{UNSAT}
- Hence the original set of lits was \textbf{UNSAT}
Nelson-Oppen – The convex case

A theory $T$ is **stably-infinite** iff every $T$-satisfiable quantifier-free formula has an infinite model.

A theory $T$ is **convex** iff

$$S \models_T a_1 = b_1 \lor \ldots \lor a_n = b_n \implies S \models a_i = b_i \text{ for some } i$$

**Deterministic Nelson-Oppen:** [NO79, TH96, MZ02]

- Given two signature-disjoint, stably-infinite and convex theories $T_1$ and $T_2$
- Given a set of literals $S$ over the signature of $T_1 \cup T_2$
- The $(T_1 \cup T_2)$-satisfiability of $S$ can be checked with the following algorithm:
Deterministic Nelson-Oppen

1. **Purify** $S$ and split it into $S_1 \cup S_2$.
   
   Let $E$ the set of interface equalities between $S_1$ and $S_2$

2. If $S_1$ is $T_1$-unsatisfiable then **UNSAT**

3. If $S_2$ is $T_2$-unsatisfiable then **UNSAT**

4. If $S_1 \models_{T_1} x = y$ with $x = y \in E \setminus S_2$ **then**
   
   $S_2 := S_2 \cup \{x = y\}$ and **goto 3**

5. If $S_2 \models_{T_2} x = y$ with $x = y \in E \setminus S_1$ **then**
   
   $S_1 := S_1 \cup \{x = y\}$ and **goto 2**

6. Report **SAT**
Consider the following **UNSATISFIABLE** set of literals:

\[
\begin{align*}
1 \leq x & \leq 2 \\
f(1) &= a \\
f(x) &= b \\
a &= b + 2 \\
f(2) &= f(1) + 3
\end{align*}
\]

There are **two theories** involved: \( \mathbb{T}_{LA(\mathbb{Z})} \) and \( \mathbb{T}_{EUF} \)

**FIRST STEP:** *purify* each literal so that it belongs to a single theory

\[
f(1) = a \implies f(e_1) = a \\
e_1 = 1
\]
Motivating example – Non-convex case

Consider the following **UNSATISFIABLE** set of literals:

\[
\begin{align*}
1 \leq x &\leq 2 \\
f(1) &= a \\
f(x) &= b \\
a &= b + 2 \\
f(2) &= f(1) + 3
\end{align*}
\]

There are **two theories** involved: \(\mathbb{T}_{LA(\mathbb{Z})}\) and \(\mathbb{T}_{EUF}\)

**FIRST STEP:** purify each literal so that it belongs to a single theory

\[
\begin{align*}
f(2) &= f(1) + 3 &\implies e_2 &= 2 \\
f(e_2) &= e_3 \\
f(e_1) &= e_4 \\
e_3 &= e_4 + 3
\end{align*}
\]
Motivating example – Non-convex case (2)

SECOND STEP: check satisfiability and exchange entailed equalities

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>EUF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ≤ x</td>
<td>f(e₁) = a</td>
</tr>
<tr>
<td>x ≤ 2</td>
<td>f(x) = b</td>
</tr>
<tr>
<td>e₁ = 1</td>
<td>f(e₂) = e₃</td>
</tr>
<tr>
<td>a = b + 2</td>
<td>f(e₁) = e₄</td>
</tr>
<tr>
<td>e₂ = 2</td>
<td></td>
</tr>
<tr>
<td>e₃ = e₄ + 3</td>
<td></td>
</tr>
<tr>
<td>a = e₄</td>
<td></td>
</tr>
</tbody>
</table>

The two solvers only share constants: x, e₁, a, b, e₂, e₃, e₄

- Ari-Solver says SAT
- EUF-Solver says SAT
- EUF ⊨ a = e₄
Motivating example – Non-convex case (2)

SECOND STEP: check satisfiability and exchange entailed equalities

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>EUF</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \leq x$</td>
<td>$f(e_1) = a$</td>
</tr>
<tr>
<td>$x \leq 2$</td>
<td>$f(x) = b$</td>
</tr>
<tr>
<td>$e_1 = 1$</td>
<td>$f(e_2) = e_3$</td>
</tr>
<tr>
<td>$a = b + 2$</td>
<td>$f(e_1) = e_4$</td>
</tr>
<tr>
<td>$e_2 = 2$</td>
<td></td>
</tr>
<tr>
<td>$e_3 = e_4 + 3$</td>
<td></td>
</tr>
<tr>
<td>$a = e_4$</td>
<td></td>
</tr>
</tbody>
</table>

The two solvers only share constants: $x, e_1, a, b, e_2, e_3, e_4$

- *Ari*-Solver says SAT
- *EUF*-Solver says SAT
- No theory entails any other interface equality, but...
Motivating example – Non-convex case(2)

SECOND STEP: check satisfiability and exchange entailed equalities

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>EUF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ≤ x</td>
<td>( f(e_1) = a )</td>
</tr>
<tr>
<td>x ≤ 2</td>
<td>( f(x) = b )</td>
</tr>
<tr>
<td>( e_1 = 1 )</td>
<td>( f(e_2) = e_3 )</td>
</tr>
<tr>
<td>( a = b + 2 )</td>
<td>( f(e_1) = e_4 )</td>
</tr>
<tr>
<td>( e_2 = 2 )</td>
<td></td>
</tr>
<tr>
<td>( e_3 = e_4 + 3 )</td>
<td></td>
</tr>
<tr>
<td>( a = e_4 )</td>
<td></td>
</tr>
</tbody>
</table>

The two solvers only share constants: \( x, e_1, a, b, e_2, e_3, e_4 \)

- Ari-Solver says SAT
- EUF-Solver says SAT
- Ari \( \models_T x = e_1 \lor x = e_2 \). Let’s consider both cases.
Motivating example – Non-convex case(2)

SECOND STEP: check satisfiability and exchange entailed equalities

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>EUF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 \leq x</td>
<td>( f(e_1) = a )</td>
</tr>
<tr>
<td>x \leq 2</td>
<td>( f(x) = b )</td>
</tr>
<tr>
<td>( e_1 = 1 )</td>
<td>( f(e_2) = e_3 )</td>
</tr>
<tr>
<td>( a = b + 2 )</td>
<td>( f(e_1) = e_4 )</td>
</tr>
<tr>
<td>( e_2 = 2 )</td>
<td>( x = e_1 )</td>
</tr>
<tr>
<td>( e_3 = e_4 + 3 )</td>
<td></td>
</tr>
<tr>
<td>( a = e_4 )</td>
<td></td>
</tr>
<tr>
<td>x = e_1</td>
<td></td>
</tr>
</tbody>
</table>

- Ari-Solver says SAT
- EUF-Solver says SAT
- EUF \( \models_T a = b \), that when sent to Ari makes it UNSAT
Motivating example – Non-convex case (2)

SECOND STEP: check satisfiability and exchange entailed equalities

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>EUF</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \leq x$</td>
<td>$f(e_1) = a$</td>
</tr>
<tr>
<td>$x \leq 2$</td>
<td>$f(x) = b$</td>
</tr>
<tr>
<td>$e_1 = 1$</td>
<td>$f(e_2) = e_3$</td>
</tr>
<tr>
<td>$a = b + 2$</td>
<td>$f(e_1) = e_4$</td>
</tr>
<tr>
<td>$e_2 = 2$</td>
<td></td>
</tr>
<tr>
<td>$e_3 = e_4 + 3$</td>
<td></td>
</tr>
<tr>
<td>$a = e_4$</td>
<td></td>
</tr>
</tbody>
</table>

Let’s try now with $x = e_2$
Motivating example – Non-convex case(2)

SECOND STEP: check satisfiability and exchange entailed equalities

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>EUF</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \leq x$</td>
<td>$f(e_1) = a$</td>
</tr>
<tr>
<td>$x \leq 2$</td>
<td>$f(x) = b$</td>
</tr>
<tr>
<td>$e_1 = 1$</td>
<td>$f(e_2) = e_3$</td>
</tr>
<tr>
<td>$a = b + 2$</td>
<td>$f(e_1) = e_4$</td>
</tr>
<tr>
<td>$e_2 = 2$</td>
<td>$x = e_2$</td>
</tr>
<tr>
<td>$e_3 = e_4 + 3$</td>
<td></td>
</tr>
<tr>
<td>$a = e_4$</td>
<td></td>
</tr>
<tr>
<td>$x = e_2$</td>
<td></td>
</tr>
</tbody>
</table>

- *Ari*-Solver says SAT
- *EUF*-Solver says SAT
- $EUF \models_T b = e_3$, that when sent to *Ari* makes it **UNSAT**
SECOND STEP: check satisfiability and exchange entailed equalities

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>EUF</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \leq x$</td>
<td>$f(e_1) = a$</td>
</tr>
<tr>
<td>$x \leq 2$</td>
<td>$f(x) = b$</td>
</tr>
<tr>
<td>$e_1 = 1$</td>
<td>$f(e_2) = e_3$</td>
</tr>
<tr>
<td>$a = b+2$</td>
<td>$f(e_1) = e_4$</td>
</tr>
<tr>
<td>$e_2 = 2$</td>
<td>$x = e_2$</td>
</tr>
<tr>
<td>$e_3 = e_4 + 3$</td>
<td></td>
</tr>
<tr>
<td>$a = e_4$</td>
<td></td>
</tr>
<tr>
<td>$x = e_2$</td>
<td></td>
</tr>
</tbody>
</table>

Since both $x = e_1$ and $x = e_2$ are UNSAT, the set of literals is UNSAT
In the previous example Deterministic NO does not work

This was because $T_{LA}(Z)$ is not convex:

$$S_{LA}(Z) \models T_{LA}(Z) \ x = e_1 \lor x = e_2,$$

but

$$S_{LA}(Z) \not\models T_{LA}(Z) \ x = e_1 \text{ and }$$

$$S_{LA}(Z) \not\models T_{LA}(Z) \ x = e_2$$

However, there is a version of NO for non-convex theories

Given a set constants $C$, an arrangement $\mathcal{A}$ over $C$ is:

- A set of equalities and disequalities between constants in $C$
- For each $x, y \in C$ either $x = y \in \mathcal{A}$ or $x \neq y \in \mathcal{A}$
Nelson-Oppen – The non-convex case (2)

Non-deterministic Nelson-Oppen: [NO79, TH96, MZ02]

- Given two signature-disjoint, stably-infinite theories $T_1$ and $T_2$
- Given a set of literals $S$ over the signature of $T_1 \cup T_2$
- The $(T_1 \cup T_2)$-satisfiability of $S$ can be checked via:

1. **Purify** $S$ and split it into $S_1 \cup S_2$
   
   Let $C$ be the set of shared constants

2. **For every** arrangement $A$ over $C$ **do**
   
   If $(S_1 \cup A)$ is $T_1$-satisfiable and $(S_2 \cup A)$ is $T_2$-satisfiable
   
   report **SAT**

3. Report **UNSAT**

This is another example of Case Reasoning inside a $T$-Solver
Overview of the talk

- Motivation
- SMT
- Theories of Interest
- Eager approach
- Lazy approach
  - Optimizations
  - Theory propagation
  - Conflict analysis in DPLL($T$)
  - Combining Theory Solvers

Eager vs Lazy
REMEMBER....

Important and beneficial aspects of the lazy approach: (even with the optimizations)

- Everyone does what he/she is good at:
  - SAT solver takes care of Boolean information
  - Theory solver takes care of theory information

- Theory solver only receives conjunctions of literals

- Modular approach:
  - SAT solver and $T$-solver communicate via a simple API
  - SMT for a new theory only requires new $T$-solver
  - SAT solver can be embedded in a lazy SMT system with very few new lines of code
The **Lazy Approach** idea (**SAT Solver + Theory Reasoner**) has been applied to other **extensions of SAT** ($x_i$’s are Boolean):

- Cardinality constraints (e.g. $x_1 + x_2 + \ldots + x_7 \leq 4$)
- Pseudo-Boolean constraints (e.g. $7x_1 + 4x_2 + 3x_3 + 5x_4 \leq 10$)
- ...

Also sophisticated **encodings exist** for these constraints (**Eager Approach**)

**Lazy approach** extremely simple to implement, but is it **always** competitive w.r.t. an encoding?
Eager vs Lazy Approach (3)

Consider the problem with no SAT clauses and two constraints:

\[ x_1 + \ldots + x_n \leq n/2 \]
\[ x_1 + \ldots + x_n > n/2 \]

Let us see how a (very) Lazy Approach would behave:

- Problem is obviously unsatisfiable
- Inconsistency explanations are of the form:
Consider the problem with no SAT clauses and two constraints:

\[ x_1 + \ldots + x_n \leq n / 2 \]
\[ x_1 + \ldots + x_n > n / 2 \]

Let us see how a (very) Lazy Approach would behave:

- Problem is obviously unsatisfiable
- Inconsistency explanations are of the form:

\[ \neg x_{i_1} \lor \ldots \lor \neg x_{i_{n/2}+1} \]
\[ x_{i_1} \lor \ldots \lor x_{i_{n/2}} \]
Consider the problem with no SAT clauses and two constraints:

\[ x_1 + \ldots + x_n \leq n/2 \]
\[ x_1 + \ldots + x_n > n/2 \]

Let us see how a (very) Lazy Approach would behave:

- Problem is obviously unsatisfiable
- Inconsistency explanations are of the form:
  \[ \neg x_{i_1} \lor \ldots \lor \neg x_{i_{n/2+1}} \]
  \[ x_{i_1} \lor \ldots \lor x_{i_{n/2}} \]
- All \( \binom{n}{\frac{n}{2}+1} + \binom{n}{n/2} \) explanations are needed to produce an unsatisfiable subset of clauses
Consider the problem with no SAT clauses and two constraints:

\[ x_1 + \ldots + x_n \leq n/2 \]
\[ x_1 + \ldots + x_n > n/2 \]

Let us see how a (very) Lazy Approach would behave:

- **Problem is obviously unsatisfiable**
- Inconsistency explanations are of the form:
  \[ \neg x_{i_1} \lor \ldots \lor \neg x_{i_{n/2+1}} \]
  \[ x_{i_1} \lor \ldots \lor x_{i_{n/2}} \]
- **All** \((\binom{n}{\frac{n}{2}+1}) + (\binom{n}{n/2})\) explanations are needed to produce an unsatisfiable subset of clauses
- Hence, runtime is exponential in \(n\).
What has happened?

- **Lazy approach** = lazily encoding (parts of) the theory into SAT
- Sometimes, **only parts** of the theory need to be encoded
- But in this example the **whole constraint** is encoded into SAT...
- ...and the encoding used is a **very naive** one
Lazy approach = lazily encoding (parts of) the theory into SAT
Sometimes, only parts of the theory need to be encoded
But in this example the whole constraint is encoded into SAT...
...and the encoding used is a very naive one
Best here is a good SAT encoding with auxiliary variables
Robert Nieuwenhuis, Albert Oliveras, Cesare Tinelli. *Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T).*  


References


References


References


References


References


References


References


References


