Introduction to SMT

Albert Oliveras
Technical University of Catalonia

8th International SAT/SMT/AR Summer School 2019
Lisbon, Portugal
July 3rd, 2019
Overview of the talk

- Motivation
  - SMT
  - Theories of Interest
  - History of SMT
  - Eager approach
  - Lazy approach
    - Optimizations and DPLL(T)
    - Theory solvers: difference logic and case splitting
    - Combining Theory Solvers
  - Limitations and Other Approaches
Introduction

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

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

Another 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, ...)

Examples of this alternative 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 and DPLL($T$)
  - Theory solvers: difference logic and case splitting
  - Combining Theory Solvers

- Limitations and Other Approaches
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, ...

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 and DPLL($T$)
    - Theory solvers: difference logic and case splitting
    - Combining Theory Solvers
  - Limitations and Other Approaches
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
  \[ a \times (f(b) + f(c)) = d \land b \times (f(a) + f(c)) \neq d \land a = b \]
  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**: $x \preceq k$ with $\preceq \in \{<,>,\leq,\geq,=\}$
  - **Difference logic**: $x - y \preceq k$, with $\preceq \in \{<,>,\leq,\geq,=\}$
    - [NO05, WIGG05, SM06]
  - **UTVPI**: $\pm x \pm y \preceq k$, with $\preceq \in \{<,>,\leq,\geq,=\}$
    - [LM05]
  - **Linear arithmetic**, e.g: $2x - 3y + 4z \leq 5$
    - [DdM06]
  - **Non-linear arithmetic**, e.g: $2xy + 4xz^2 - 5y \leq 10$
    - [BLNM+09, ZM10]

- Variables are either **reals or integers**

- **Machine-inspired arithmetic**: **floating-point arithmetic**
Two interpreted function symbols \( \text{read} \) and \( \text{write} \)

Theory is axiomatized by:

\[
\forall a \forall i \forall v \left( \text{read}(\text{write}(a, i, v), i) = v \right)
\]

\[
\forall a \forall i \forall j \forall v \left( i \neq j \rightarrow \text{read}(\text{write}(a, i, v), j) = \text{read}(a, j) \right)
\]

Sometimes extensionality is added:

\[
\forall a \forall b \left( \left( \forall i \left( \text{read}(a, i) = \text{read}(b, i) \right) \right) \rightarrow a = b \right)
\]

Is the following set of literals satisfiable?

\[
\text{write}(a, i, x) \neq b \quad \text{read}(b, i) = y \quad \text{read}(\text{write}(b, i, x), j) = y \\
\]

\[
a = b \quad i = j
\]

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, substract, 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.
Overview of the talk

- Motivation
- SMT
- Theories of Interest

**History of SMT**

- Eager approach
- Lazy approach
  - Optimizations and DPLL($T$)
  - Theory solvers: difference logic and case splitting
  - Combining Theory Solvers
- Limitations and Other Approaches
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]
**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]

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

- Motivation
- SMT
- Theories of Interest

Eager approach

- Lazy approach
  - Optimizations and DPLL($T$)
  - Theory solvers: difference logic and case splitting
  - Combining Theory Solvers

- Limitations and Other Approaches
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
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:
    \[
    \begin{align*}
    a = b & \rightarrow A = B \\
    a = c & \rightarrow A = C \\
    b = c & \rightarrow B = C
    \end{align*}
    \]

- **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**
**Eager approach – Example (2)**

- **Second step:** encode formula into propositional logic
  - **Small-domain** encoding:
    - If there are $n$ different constants, there is a model with size at most $n$
    - $\log n$ 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 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 and DPLL(T)

Theory solvers: difference logic and case splitting

Combining Theory Solvers

Limitations and Other Approaches
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 \([1, \overline{2}, \overline{4}]\)
Lazy approach

Methodology:
Example: consider EUF and the CNF

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

- SAT solver returns model \([1, \bar{2}, \bar{4}]\)
- 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

\[
\begin{align*}
g(a) = c & \quad \land \quad ( f(g(a)) \neq f(c) \quad \lor \quad g(a) = d ) \quad \land \quad c \neq d \\
\end{align*}
\]

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

- Motivation
- SMT
- Theories of Interest
- Eager approach
- Lazy approach

Optimizations and DPLL(\(T\))
- Theory solvers: difference logic and case splitting
- Combining Theory Solvers

Limitations and Other Approaches
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
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
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.
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 - $T$-propagation

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

- How can we improve that? For example:
  - Assume that $a < b$, $b < c$ are in our partial assignment $M$.
  - If the formula contains $a < c$ we would like to add it to $M$

- 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 $\text{DPLL}(T)$ \[\text{NOT06}\]
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 relatively little effort
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 literal detection, ...
  - Desirable: partial model detection

- **T-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:

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

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

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

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

\[
1 \ || \ 1, \ 2 \lor 3, \ 4 \ \Rightarrow \ (\text{UnitPropagate})
\]
Consider again \textbf{EUF} and the formula:

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

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

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

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

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

\[
\begin{align*}
\emptyset \ | \ 1, \overline{2} \lor 3, \overline{4} & \implies \text{(UnitPropagate)} \\
1 \ | \ 1, \overline{2} \lor 3, \overline{4} & \implies \text{(UnitPropagate)} \\
1 \overline{4} \ | \ 1, \overline{2} \lor 3, \overline{4} & \implies \text{(T-Propagate)} \\
1 \overline{4} \ 2 \ | \ 1, \overline{2} \lor 3, \overline{4} & \implies \text{(T-Propagate)}
\end{align*}
\]
Consider again EUF and the formula:

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

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

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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ∥</td>
<td>1, 2 ∨ 3, 4</td>
<td>⇒</td>
<td>(UnitPropagate)</td>
<td></td>
</tr>
<tr>
<td>1 ∥</td>
<td>1, 2 ∨ 3, 4</td>
<td>⇒</td>
<td>(UnitPropagate)</td>
<td></td>
</tr>
<tr>
<td>1 4 ∥</td>
<td>1, 2 ∨ 3, 4</td>
<td>⇒</td>
<td>(T-Propagate)</td>
<td></td>
</tr>
<tr>
<td>1 4 2 ∥</td>
<td>1, 2 ∨ 3, 4</td>
<td>⇒</td>
<td>(T-Propagate)</td>
<td></td>
</tr>
<tr>
<td>1 4 2 3 ∥</td>
<td>1, 2 ∨ 3, 4</td>
<td>⇒</td>
<td>(Fail)</td>
<td></td>
</tr>
</tbody>
</table>

**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();
    remove_lemmas_if_applicable();
    if (!decide()) return 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
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
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}

// let \( C = C' \lor l \) where \( l \) is UIP
\text{backjump(maxDL(C'))}
add \( l \) to the model with reason \( C \)
\text{learn}(C)
Conflict analysis in DPLL($T$):

```plaintext
if boolean conflict then C:= conflicting clause
else C:= ¬( solver.explain_inconsistency() )

while C contains more than one lit of last DL

  l:= last literal assigned in C
  C:= Resolution(C, reason(l))

end while

// let C = C' v l where l is UIP
backjump(maxDL(C'))
add l to the model with reason C
learn(C)
```
DPLL($T$) - Conflict Analysis (3)

What does \texttt{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 \texttt{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 \texttt{reason}(l) is $\neg l_1 \lor \ldots \lor \neg l_n \lor l$
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($a$) $\neq h(c)$, $c = b$\}

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

Introduction to SMT – p. 30
Overview of the talk

- Motivation
- SMT
- Theories of Interest
- Eager approach
- Lazy approach
  - Optimizations and DPLL(T)
  - \textit{T}-solvers: difference logic
    - Combining Theory Solvers
- Limitations and Other Approaches
Difference logic

- Literals in Difference Logic are of the form \( a - b \bowtie k \), where
  - \( \bowtie \in \{ \leq, \geq, <, >, =, \neq \} \)
  - \( a \) and \( b \) are integer/real variables
  - \( k \) is an integer/real

- At the formula level,
  - \( a = b \) is replaced by \( p \) and \( p \leftrightarrow a \leq b \land b \leq a \) is added

- If domain is \( \mathbb{Z} \) then \( a - b < k \) is replaced by \( a - b \leq k - 1 \)

- If domain is \( \mathbb{R} \) then \( a - b < k \) is replaced by \( a - b \leq k - \delta \)
  - \( \delta \) is a sufficiently small real
  - \( \delta \) is not computed but used symbolically
    (i.e. numbers are pairs \((k, \delta)\))

- Hence we can assume all literals are \( a - b \leq k \)
Note that any solution to a set of DL literals can be shifted (i.e. if $\sigma$ is a solution then $\sigma'(x) = \sigma(x) + k$ also is a solution).

This allows one to process bounds $x \leq k$

- Introduce fresh variable $zero$
- Convert all bounds $x \leq k$ into $x - zero \leq k$
- Given a solution $\sigma$, shift it so that $\sigma(zero) = 0$

If we allow (dis)equalities as literals, then:
- If domain is $\mathbb{R}$ consistency check is polynomial
- If domain is $\mathbb{Z}$ consistency check is NP-hard ($k$-colorability)
- $1 \leq c_i \leq k$ with $i = 1 \ldots \#verts$ encodes $k$ colors available
- $c_i \neq c_j$ if $i$ and $j$ adjacents encode proper assignment
Difference Logic as a Graph Problem

Given $M = \{a-b \leq 2, b-c \leq 3, c-a \leq -7\}$, construct weighted graph $G(M)$

**Theorem:**

$M$ is $T$-inconsistent iff $G(M)$ has a negative cycle
Theorem:

\[ M \text{ is } T\text{-inconsistent} \iff G(M) \text{ has a negative cycle} \]

\[ \Leftarrow \]

Any negative cycle \( a_1 \xrightarrow{k_1} a_2 \xrightarrow{k_2} a_3 \rightarrow \ldots \rightarrow a_n \xrightarrow{k_n} a_1 \)
corresponds to a set of literals:

\[
\begin{align*}
    a_1 & - a_2 \leq k_1 \\
    a_2 & - a_3 \leq k_2 \\
    & \vdots \\
    a_n & - a_1 \leq k_n
\end{align*}
\]

If we add them all, we get \( 0 \leq k_1 + k_2 + \ldots + k_n \), which is inconsistent since neg. cycle implies \( k_1 + k_2 + \ldots + k_n < 0 \)
Theorem:

\[ M \text{ is } T\text{-inconsistent iff } G(M) \text{ has a negative cycle} \]

\[ \Rightarrow \]

Let us assume that there is no negative cycle.

1. Consider additional vertex \( o \) with edges \( o \rightarrow v \) to all verts. \( v \)
2. For each variable \( x \), let \( \sigma(x) = -dist(o,x) \)  
   [exists because there is no negative cycle]
3. \( \sigma \) is a model of \( M \)
   - If \( \sigma \not\models x - y \leq k \) then \( -dist(o,x) + dist(o,y) > k \)
   - Hence, \( dist(o,y) > dist(o,x) + k \)
   - But \( k = weight(x \rightarrow y) \)!!!
forall $v \in V$ do $d[v] := \infty$ endfor
forall $i = 1$ to $|V| - 1$ do
forall $(u,v) \in E$ do
forall $(u,v) \in E$ do
  if $d[v] > d[u] + \text{weight}(u,v)$ then
    $d[v] := d[u] + \text{weight}(u,v)$
    $p[v] := u$
  endif
endfor
endfor
forall $(u,v) \in E$ do
  if $d[v] > d[u] + \text{weight}(u,v)$ then
    Negative cycle detected
    Cycle reconstructed following $p$
  endif
endfor
Consistency checks can be performed using Bellman-Ford in time $O(|V| \cdot |E|)$.

Other more efficient variants exist.

Incrementality easy:
- Upon arrival of new literal $a \xrightarrow{k} b$ process graph from $u$.

Solutions can be kept after backtracking.

Inconsistency explanations are negative cycles (irredundant but not minimal explanations).
Theory propagation

- Addition of $a \xrightarrow{k} b$ entails $c - d \leq k'$ only if

$$
c \longrightarrow^* a \xrightarrow{k} b \longrightarrow^* d
$$

- Given a solution $\sigma$, each edge $a \xrightarrow{k} b$ (i.e. $a - b \leq k$) has its reduced cost $k - \sigma(a) + \sigma(b) \geq 0$

- Shortest path computation more efficient using reduced costs, since they are non-negative [Dijkstra’s algorithm]

- Theory propagation $\approx$ shortest-path computations

- Explanations are the shortest paths
Overview of the talk

- Motivation
- SMT
- Theories of Interest
- Eager approach
- Lazy approach
  - Optimizations and DPLL($T$)
    - $T$-solvers: case splitting
  - Combining Theory Solvers
- Limitations and Other Approaches
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 \(\text{read}(A, j) \neq \text{read}(A, j)\) !!!

CONCLUSION: T-inconsistent
Case Reasoning in Theory Solvers (2)

- A complete T-solver reasons 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 (more details later)
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 \[ s = read\left(\text{write}(A, i, t), j\right) \]

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:”

\[
s = s' \land i = j \quad \rightarrow \quad s = t
\]

\[
s = s' \land i \neq j \quad \rightarrow \quad s = \text{read}(A, j)
\]

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

- Motivation
- SMT
- Theories of Interest
- Eager approach
- Lazy approach
  - Optimizations and DPLL($T$)
  - Theory solvers: difference logic and case splitting
- Combining Theory Solvers
- Limitations and Other Approaches
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 (\(T_{\text{LA}}\))
- The theory of arrays (\(T_{\text{A}}\))
- The theory of uninterpreted functions (\(T_{\text{EUF}}\))

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 \(T_{\text{LA}} \cup T_{\text{A}} \cup T_{\text{EUF}}\)?

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: \( T_{LA(\mathbb{R})} \) and \( T_{EUF} \)

**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

**EUF** | **Arithmetic**
--- | ---
\( f(e_1) = a \) | \( e_2 - e_3 = e_1 \)
\( f(x) = e_2 \) | \( e_4 = 0 \)
\( f(y) = e_3 \) | \( e_5 = a + 2 \)
\( f(e_4) = e_5 \)
\( x = y \)

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
Motivating example - Convex case (2)

SECOND STEP: check satisfiability and exchange entailed equalities

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

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

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

SECOND STEP: check satisfiability and exchange entailed equalities

\[
\begin{align*}
EUF & & Arithmetic \\
\text{f}(e_1) & = & a & & e_2 - e_3 & = & e_1 \\
\text{f}(x) & = & e_2 & & e_4 & = & 0 \\
\text{f}(y) & = & e_3 & & e_5 & = & a + 2 \\
\text{f}(e_4) & = & e_5 & & e_2 & = & e_3 \\
x & = & y \\
e_1 & = & e_4
\end{align*}
\]

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

- \( EUF \)-Solver says SAT
- \( Ari\)-Solver says SAT
- \( Ari \models e_1 = e_4 \)
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>( 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
- \( \text{EUF} \models a = e_5 \)
**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>$a = e_5$</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 UNSAT**
- Hence the original set of lits was **UNSAT**

*Introduction to SMT – p. 49*
A theory $T$ is stably-infinite iff every $T$-satisfiable quantifier-free formula has an infinite model.

A theory $T$ is convex iff

$S \models_T a_1 = b_1 \lor \ldots \lor a_n = b_n \implies S \models a_i = b_i$ 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: \( \mathbb{T}_{LA(Z)} \) and \( \mathbb{T}_{EUF} \)

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

\[
f(1) = a \implies f(e_1) = a \\
e_1 = 1
\]
Consider the following **UNSATISFIABLE** set of literals:

\[
1 \leq x \leq 2 \\
f(1) = a \\
f(x) = b \\
a = b + 2 \\
f(2) = f(1) + 3
\]

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

\[
f(2) = f(1) + 3 \implies e_2 = 2 \\
f(e_2) = e_3 \\
f(e_1) = e_4 \\
e_3 = e_4 + 3
\]
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
- $EUF \models a = e_4$
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>\text{ } f(e_1) = a</td>
</tr>
<tr>
<td>x \leq 2</td>
<td>\text{ } f(x) = b</td>
</tr>
<tr>
<td>e_1 = 1</td>
<td>\text{ } f(e_2) = e_3</td>
</tr>
<tr>
<td>a = b + 2</td>
<td>\text{ } f(e_1) = e_4</td>
</tr>
<tr>
<td>e_2 = 2</td>
<td>\text{ } e_3 = e_4 + 3</td>
</tr>
<tr>
<td>e_3 = e_4 + 3</td>
<td>\text{ } a = e_4</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.
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>x = e₁</td>
</tr>
<tr>
<td>e₃ = e₄ + 3</td>
<td></td>
</tr>
<tr>
<td>a = e₄</td>
<td></td>
</tr>
<tr>
<td>x = e₁</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**
SECOND STEP: check satisfiability and exchange entailed equalities

\begin{align*}
\text{Arithmetic} & \quad & \text{EUF} \\
1 & \leq & x & \quad & f(e_1) & = & a \\
x & \leq & 2 & \quad & f(x) & = & b \\
e_1 & = & 1 & \quad & f(e_2) & = & e_3 \\
a & = & b + 2 & \quad & f(e_1) & = & e_4 \\
e_2 & = & 2 \\
e_3 & = & e_4 + 3 \\
a & = & e_4
\end{align*}

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>( f(e_1) = a )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ( \leq ) ( x )</td>
<td>( f(x) = b )</td>
</tr>
<tr>
<td>( x ) ( \leq ) 2</td>
<td>( f(e_2) = e_3 )</td>
</tr>
<tr>
<td>( e_1 = 1 )</td>
<td>( f(e_1) = e_4 )</td>
</tr>
<tr>
<td>( a = b + 2 )</td>
<td>( x = e_2 )</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>
<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:
\[
S_{LA}(Z) \models T_{LA}(Z) \ x = e_1 \lor x = e_2, \text{ but }
\]
\[
S_{LA}(Z) \not\models T_{LA}(Z) \ x = e_1 \text{ and }
\]
\[
S_{LA}(Z) \not\models T_{LA}(Z) \ x = e_2
\]

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 \( x \neq y \in \mathcal{A} \)
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 $\mathcal{A}$ over $C$ do
   If $(S_1 \cup \mathcal{A})$ is $T_1$-satisfiable and $(S_2 \cup \mathcal{A})$ is $T_2$-satisfiable
   report **SAT**
3. Report **UNSAT**
Overview of the talk

- Motivation
- SMT
- Theories of Interest
- Eager approach
- Lazy approach
  - Optimizations and DPLL($T$)
  - Theory solvers: difference logic and case splitting
  - Combining Theory Solvers

Limitations and Other Approaches
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*) can be applied to other extensions of SAT:

- 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** seems to dominate, but can we claim that it is **always** the best option?
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:
  \[ \neg x_{i_1} \lor \ldots \lor \neg x_{i_{n/2+1}} \lor x_{i_1} \lor \ldots \lor x_{i_{n/2}} \]
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 \textit{unsatisfiable}
- Inconsistency \textit{explanations} are of the form:
  \[ \neg x_{i_1} \lor \ldots \lor \neg x_{i_{n/2}+1} \]
  \[ x_{i_1} \lor \ldots \lor x_{i_{n/2}} \]
- \textbf{All} \( \binom{n}{\frac{n}{2}+1} + \binom{n}{n/2} \) explanations are needed to produce an unsatisfiable subset of clauses
- Hence, \textit{runtime} is \textit{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
- Best here is a **good SAT encoding** with auxiliary variables
The diamonds example

\[a_n < a_0 \land \bigwedge_{k=0}^{n-1} (a_k < b_k \land b_k < a_{k+1}) \lor (a_k < c_k \land c_k < a_{k+1}) \]

With these literals, only exponential refutations exist.

Introducing \(a_0 < a_1, \ a_1 < a_2, \ldots\) allows linear refutations.
Other approaches

Previous examples show limitations of (DPLL(T))
There are more technical limitations out of the scope of this talk
Research on model-based procedures tries to address these issues:

- **Linear Real Arithmetic**
  - Generalizing DPLL to Richer Logics [MKS09]
  - Conflict Resolution [KTV09]
  - Natural Domain SMT [Cot10]

- **Linear Integer Arithmetic**
  - Cutting to the Chase [JdM13]

- **Non-Linear Real Arithmetic**
  - Solving Non-Linear Arithmetic [JM12]

- **General Framework**
  - Model-Constructing Satisfiability Calculus [JM13]
  - Satisfiability Modulo Theories and Assignments [BGS17]
References


References


References


References


References


References


References


## References

## References

<table>
<thead>
<tr>
<th>Reference</th>
<th>Authors</th>
<th>Title</th>
<th>Conference/Book Details</th>
</tr>
</thead>
</table>