Introduction to Satisfiability Modulo Theories

Combinatorial Problem Solving (CPS)

Albert Oliveras    Enric Rodríguez-Carbonell

May 24, 2019
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 in deciding the satisfiability of a (quantifier-free) first-order formula with respect to a background theory.

Example (Equality with Uninterpreted Functions – EUF):

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

SMT is widely applied in hardware/software verification.

Theories of interest here:

- EUF, arithmetic, arrays, bit vectors, combinations of these.

With these and other theories, SMT methods can also be used to solve combinatorial problems.
Lazy Approach to SMT

Methodology:

Example: consider EUF and

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

Methodology:

Example: consider EUF and

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

• Send \{1, 2 \lor 3, 4\} to SAT solver
  SAT solver returns model [1, 2, 4]
  Theory solver says \text{T}-inconsistent
Lazy Approach to SMT

Methodology:

Example: consider EUF and

\[ g(a) = c \quad \land \quad \left( \underbrace{f(g(a)) \neq f(c)}_{1} \quad \lor \quad \underbrace{g(a) = d}_{3} \right) \quad \land \quad c \neq d \]

- Send \( \{1, \overline{2} \lor 3, \overline{4}\} \) to SAT solver
  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 to SMT

Methodology:
Example: consider EUF and

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

- Send \(\{1, \overline{2} \lor 3, \overline{4}\}\) to SAT solver
  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

- Send \(\{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2 \lor 4, \overline{1} \lor \overline{2} \lor \overline{3} \lor 4\}\) to SAT solver
  SAT solver says UNSATISFIABLE
Lazy Approach to SMT

- Why “lazy”? Theory information used lazily when checking $T$-consistency of propositional models (cf. eagerly encoding into SAT upfront)

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

- (Early) Tools:
  - Barcelogic (UPC)
  - CVC (Uni. NY + Iowa)
  - DPT (Intel)
  - MathSAT (Univ. Trento)
  - Yices (SRI)
  - Z3 (Microsoft)
  - ...
Optimizations

Several optimizations for enhancing efficiency:

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

Several optimizations for enhancing efficiency:

- Check $T$-consistency only of full propositional models
- Check $T$-consistency of partial assignment while being built
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
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
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
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, do conflict analysis and backjump
Important Points

Advantages of the lazy approach:

- Everyone does what it 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 extended to a lazy SMT system with very few new lines of code (40?)
Theory propagation

- As pointed out, the lazy approach has a drawback:
  - Theory information does not guide the search

- How can we improve that? Theory propagation
  
  \[ M \parallel F \Rightarrow M l \parallel F \text{ if } \begin{cases} M \models_T l \\ l \text{ or } \neg l \text{ occurs in } F \text{ and not in } M \end{cases} \]

- 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 \( l \).
  But for efficient \( T \)-Propagate we need specialized \( T \)-Solvers

- This approach has been named \( \text{DPLL}(T) \)
Example

Consider again EUF and the formula:

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

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

Consider again EUF and the formula:

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

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

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

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

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

Consider again EUF and the formula:

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

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

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

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

Consider again **EUF** and the formula:

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

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

Consider again EUF and the formula:

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

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

Consider again EUF and the formula:

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

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

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

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

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

\[
1234 \parallel 1, \overline{2} \lor 3, \overline{4} \quad \Rightarrow \quad \text{(Fail)}
\]

\[
\text{fail}
\]

No search!
Overall algorithm

High-level view gives the same algorithm as in 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()) returns SAT; // All vars assigned
}
```

Differences are in:

- `propagate_gives_conflict`
- `analyze_conflict`
DPLL($T$) - Propagation

propagate_gives_conflict() returns Bool

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

    return false
propagate_gives_conflict() returns Bool

\[
do \{

\hspace{1cm} \text{// unit propagate}
\hspace{1cm} \text{if} \ ( \text{unit_prop_gives_conflict()} ) \ \text{then return true}

\hspace{1cm} \text{// check T-consistency of the model}
\hspace{1cm} \text{if} \ ( \text{solver.is_model_inconsistent()} ) \ \text{then return true}

\hspace{1cm} \text{// theory propagate}
\hspace{1cm} \text{solver.theory_propagate()}

\}\ \text{while (doneSomeTheoryPropagation)}

\text{return false}
\]
DPLL($T$) - Propagation

- 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
  - Only strictly necessary to call $T$-consistency at the leaves (i.e. when we have a full propositional model)
  - $T$-propagation is not necessary for correctness
DPLL(\(T\)) - Conflict Analysis

Remember conflict analysis in SAT solvers:

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

\[ \text{while } C \text{ contains more than one lit of last DL} \]
\[ \quad l := \text{last literal assigned in } C \]
\[ \quad C := \text{Resolution}(C, \text{reason}(l)) \]

\[ \text{end while} \]

// let \( C = C' \lor l \) where \( l \) is the only lit of last DL
\[ \text{backjump(maxDL}(C')) \]
\[ \text{add } l \text{ to the model with reason } C \]
\[ \text{learn}(C) \]
DPLL($T$) - Conflict Analysis

Conflict analysis in DPLL($T$):

if boolean conflict then $C :=$ conflicting clause
else $C := \neg (\text{solver.explain.inconsistency}())$

while $C$ contains more than one lit of last DL

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

end while

// let $C = C' \lor l$ where $l$ is the only lit of last DL
backjump(maxDL($C'$))
add $l$ to the model with reason $C$
learn($C$)
DPLL($T$) - Conflict Analysis

What does `explain_inconsistency` return?

- An explanation of the inconsistency:
  A (small) conjunction of literals $l_1 \land \ldots \land l_n$ such that:
    - It is $T$-inconsistent

What is now `reason(l)`?

- If $l$ was unit propagated: clause that propagated it
- If $l$ was $T$-propagated:
  - An explanation of the propagation:
    A (small) clause $\neg l_1 \lor \ldots \lor \neg l_n \lor l$ such that:
      - $l_1 \land \ldots \land l_n \models_T l$
      - $l_1, \ldots, l_n$ were in the model when $l$ was $T$-propagated
DPLL(\(T\)) - Conflict Analysis

Let \( M \) be \( c=b \) and let \( F \) contain

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

Take the following sequence:

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

\[\text{Explain}(a \neq b) \text{ is } \{ h(a) \neq h(c), c = b \} \]

\[
\begin{align*}
    h(a) &= h(c) \lor p, \\
    g(a) &= g(b) \lor \neg p \\
    a &= b \lor g(a) \neq g(b) \\
    h(a) &= h(c) \lor a = b \\
    h(a) &= h(c) \lor c \neq b
\end{align*}
\]
**DPLL(T) – T-Solver API**

What does DPLL(T) need from T-Solver?

- **T-consistency check** of a set of literals $M$, with:
  - **Explain of T-inconsistency:**
    find small $T$-inconsistent subset of $M$
  - **Incrementality:** if $l$ is added to $M$, check for $M \cup l$ faster than reprocessing $M \cup l$ from scratch.

- **Theory propagation:** find input $T$-consequences of $M$, with:
  - **Explain $T$-Propagate of $l$:**
    find (small) subset of $M$ that $T$-entails $l$.

- **Backtrack $n$:** undo last $n$ literals added
Bibliography - Further reading


