On deciding satisfiability by DPLL(Γ+Σ) and unsound theorem proving

Maria Paola Bonacina

Dipartimento di Informatica
Università degli Studi di Verona
Verona, Italy

August 4, 2009

1Joint work with Chris Lynch (Department of Mathematics and Computer Science, Clarkson University, NY, USA) and Leonardo de Moura (Microsoft Research, Redmond, WA, USA)
Motivation: reasoning for SW verification

Idea: Unsound theorem proving to get decision procedures

$\text{DPLL}(\Gamma + T)$ with $\text{UTP}$: SMT-solver $+$ Superposition $+$ UTP

Decision procedures for type systems

Discussion
Problem statement

- Decide *satisfiability* of first-order formulæ generated by SW verification tools
- Satisfiability w.r.t. *background theories* (e.g., linear arithmetic, bitvectors)
- With *quantifiers* to write, e.g.,
  - frame conditions over loops
  - auxiliary invariants over heaps
  - axioms of *type systems* and
  - *application-specific theories* without decision procedure
Shape of problem

- Background theory $\mathcal{T}$
  - $\mathcal{T} = \bigcup_{i=1}^{n} \mathcal{T}_i$, e.g., linear arithmetic, bit-vectors
- Set of formulæ: $\mathcal{R} \cup P$
  - $\mathcal{R}$: set of non-ground clauses without $\mathcal{T}$-symbols
  - $P$: large ground formula (set of ground clauses) may contain $\mathcal{T}$-symbols
- Determine whether $\mathcal{R} \cup P$ is satisfiable modulo $\mathcal{T}$
  (Equivalently: determine whether $\mathcal{T} \cup \mathcal{R} \cup P$ is satisfiable)
Tools

- Davis-Putnam-Logemann-Loveland (DPLL) procedure for SAT
- $\mathcal{T}_i$-solvers: Satisfiability procedures for the $\mathcal{T}_i$’s
- DPLL($\mathcal{T}$)-based SMT-solver: Decision procedure for $\mathcal{T}$ with Nelson-Oppen combination of the $\mathcal{T}_i$-sat procedures
- First-order engine $\Gamma$ to handle $\mathcal{R}$ (additional theory): Resolution+Rewriting+Superposition: Superposition-based
Combining strengths of different tools

- DPLL: SAT-problems; large non-Horn clauses
- Theory solvers: linear arithmetic, bitvectors
- DPLL($\Gamma$)-based SMT-solver: efficient, scalable, integrated theory reasoning
- Superposition-based inference system $\Gamma$:
  - equalities, Horn clauses, universal quantifiers
  - known to be a sat-procedure for several theories of data structures
How to get decision procedures?

- During SW development conjectures are usually false due to mistakes in implementation or specification.
- Need theorem prover that terminates on satisfiable inputs.
- Not possible in general:
  - FOL is only semi-decidable.
  - First-order formulae of linear arithmetic with uninterpreted functions: not even semi-decidable.

However we need less than a general solution.
Problematic axioms do occur in relevant inputs

\( \sqsubseteq \): subtype relation
\( f \): type constructor (e.g., Array-of)

- **Transitivity**
  \( \neg (x \sqsubseteq y) \lor \neg (y \sqsubseteq z) \lor x \sqsubseteq z \)

- **Monotonicity**
  \( \neg (x \sqsubseteq y) \lor f(x) \sqsubseteq f(y) \)

Resolution generates unbounded number of clauses (even with negative selection)
In practice we need finitely many

Example:

1. \( \neg (x \sqsubseteq y) \lor f(x) \sqsubseteq f(y) \)
2. \( a \sqsubseteq b \) generate
3. \( \{ f^i(a) \sqsubseteq f^i(b) \}_{i \geq 0} \)

In practice \( f(a) \sqsubseteq f(b) \) or \( f^2(a) \sqsubseteq f^2(b) \) often suffice to show satisfiability
Idea: Unsound theorem proving

- TP applied to maths: most conjectures are *true*
- Sacrifice *completeness* for efficiency
  Retain *soundness*: if proof found, input *unsatisfiable*
Idea: Unsound theorem proving

- TP applied to maths: most conjectures are true
- Sacrifice completeness for efficiency
  Retain soundness: if proof found, input unsatisfiable
- TP applied to verification: most conjectures are false
- Sacrifice soundness for termination
  Retain completeness: if no proof, input satisfiable
Idea: Unsound theorem proving

- TP applied to maths: most conjectures are *true*
- Sacrifice *completeness* for efficiency
  Retain *soundness*: if proof found, input *unsatisfiable*
- TP applied to verification: most conjectures are *false*
- Sacrifice *soundness* for termination
  Retain *completeness*: if no proof, input *satisfiable*
- How do we do it: Additional axioms to enforce termination
- Detect *unsoundness* as conflict + Recover by *backtracking* (DPLL framework)
Example

1. $\neg (x \sqsubseteq y) \lor f(x) \sqsubseteq f(y)$
2. $a \sqsubseteq b$
3. $a \sqsubseteq f(c)$
4. $\neg (a \sqsubseteq c)$
Example

1. \( \neg (x \sqsubseteq y) \lor f(x) \sqsubseteq f(y) \)
2. \( a \sqsubseteq b \)
3. \( a \sqsubseteq f(c) \)
4. \( \neg (a \sqsubseteq c) \)

1. Add \( f(x) \simeq x \)
2. Rewrite \( a \sqsubseteq f(c) \) into \( a \sqsubseteq c \) and get \( \Box \): backtrack!
Example

1. \( \neg(x \sqsubseteq y) \lor f(x) \sqsubseteq f(y) \)
2. \( a \sqsubseteq b \)
3. \( a \sqsubseteq f(c) \)
4. \( \neg(a \sqsubseteq c) \)

1. Add \( f(x) \simeq x \)
2. Rewrite \( a \sqsubseteq f(c) \) into \( a \sqsubseteq c \) and get \( \Box \): backtrack!
3. Add \( f(f(x)) \simeq x \)
4. \( a \sqsubseteq b \) yields only \( f(a) \sqsubseteq f(b) \)
5. \( a \sqsubseteq f(c) \) yields only \( f(a) \sqsubseteq c \)
6. Reach saturated state and detect satisfiability
DPLL

State of derivation: $M \parallel F$

- **Decide**: guess $L$ is true, add it to $M$ (decided literals)
- **UnitPropagate**: propagate consequences of assignment (implied literals)
- **Conflict**: detect $L_1 \lor \ldots \lor L_n$ all false
- **Explain**: unfold implied literals and detect decided $L_i$ in conflict clause
- **Learn**: may learn conflict clause
- **Backjump**: undo assignment for $L_i$
- **Unsat**: conflict clause is $\Box$ (nothing else to try)
DPLL($\mathcal{T}$)

State of derivation: $M \parallel F$

- $\mathcal{T}$-Propagate: add to $M$ an $L$ that is $\mathcal{T}$-consequence of $M$
- $\mathcal{T}$-Conflict: detect that $L_1, \ldots, L_n$ in $M$ are $\mathcal{T}$-inconsistent

Since $\mathcal{T}_i$-solvers build $\mathcal{T}$-model:

- PropagateEq: add to $M$ a ground $s \simeq t$ true in $\mathcal{T}$-model
DPLL($\Gamma + \mathcal{T}$): integrate $\Gamma$ in DPLL($\mathcal{T}$)

- **Idea:** literals in $M$ can be premises of $\Gamma$-inferences
- Stored as *hypotheses* in inferred clause
- *Hypothetical clause:* $H \triangleright C$ (equivalent to $\neg H \lor C$)
- Inferred clauses inherit hypotheses from premises

- **Note:** don’t need $\Gamma$ for ground inferences
- Use each engine for what is best for:
  - $\Gamma$ works on non-ground clauses and ground unit clauses
  - DPLL($\mathcal{T}$) works on all and only ground clauses
State of derivation: $M \parallel F$

$F$: set of hypothetical clauses

- **Deduce**: $\Gamma$-inference, e.g., superposition, using *non-ground* clauses in $F$ and literals in $M$

- **Backjump**: remove hypothetical clauses depending on undone assignments
Unsound inferences

- Single unsound inference rule: add *arbitrary* clause $C$
- Simulate many:
  - Suppress literals in long clause $C \lor D$:
    add $C$ and subsume
  - Replace deep term $t$ by constant $a$:
    add $t \simeq a$ and rewrite
Controlling unsound inferences

- Unsound inferences to induce termination on sat input
- What if the unsound inference makes problem unsat?!
- Detect conflict and backjump:
  - Keep track by adding $\lceil C \rceil \models C$
  - $\lceil C \rceil$: new propositional variable (a “name” for $C$)
  - Treat “unnatural failure” like “natural failure”
- Thus unsound inferences are \textit{reversible}
Unsound theorem proving in DPLL(Γ+𝕋)

State of derivation: $M \parallel F$

Inference rule:

- *UnsoundIntro*: add $\llbracket C \rrbracket \triangleright C$ to $F$ and $\llbracket C \rrbracket$ to $M$
Example as done by system

1. \( \neg (x \sqsubseteq y) \lor f(x) \sqsubseteq f(y) \)
2. \( a \sqsubseteq b \)
3. \( a \sqsubseteq f(c) \)
4. \( \neg (a \sqsubseteq c) \)
Example as done by system

1. \( \neg(x \sqsubseteq y) \lor f(x) \sqsubseteq f(y) \)
2. \( a \sqsubseteq b \)
3. \( a \sqsubseteq f(c) \)
4. \( \neg(a \sqsubseteq c) \)

1. Add \([f(x) \simeq x] \triangleright f(x) \simeq x\)
2. Rewrite \( a \sqsubseteq f(c) \) into \([f(x) \simeq x] \triangleright a \sqsubseteq c\)
Example as done by system

1. \( \neg(x \sqsubseteq y) \lor f(x) \sqsubseteq f(y) \)
2. \( a \sqsubseteq b \)
3. \( a \sqsubseteq f(c) \)
4. \( \neg(a \sqsubseteq c) \)

1. Add \([f(x) \simeq x] \triangleright f(x) \simeq x\)
2. Rewrite \( a \sqsubseteq f(c) \) into \([f(x) \simeq x] \triangleright a \sqsubseteq c\)
3. Generate \([f(x) \simeq x] \triangleright \Box; \text{ Backtrack, learn } \neg[f(x) \simeq x]\)
Example as done by system

1. \( \neg(x \sqsubseteq y) \lor f(x) \sqsubseteq f(y) \)
2. \( a \sqsubseteq b \)
3. \( a \sqsubseteq f(c) \)
4. \( \neg(a \sqsubseteq c) \)

1. Add \([f(x) \simeq x] \triangleright f(x) \simeq x\)
2. Rewrite \( a \sqsubseteq f(c) \) into \([f(x) \simeq x] \triangleright a \sqsubseteq c\)
3. Generate \([f(x) \simeq x] \triangleright \Box; \) Backtrack, learn \( \neg[f(x) \simeq x]\)
4. Add \([f(f(x)) \simeq x] \triangleright f(f(x)) \simeq x\)
5. \( a \sqsubseteq b \) yields only \( f(a) \sqsubseteq f(b) \)
6. \( a \sqsubseteq f(c) \) yields only \( f(a) \sqsubseteq f(f(c)) \)
    rewritten to \([f(f(x))) = x] \triangleright f(a) \sqsubseteq c\)
7. Reach saturated state and detect satisfiability
Issues about completeness

- Γ is refutationally complete
- Since Γ does not see all the clauses, DPLL(Γ + T) does not inherit refutational completeness trivially
Issues about completeness

- $\Gamma$ is refutationally complete
- Since $\Gamma$ does not see all the clauses, \( \text{DPLL}(\Gamma + \mathcal{T}) \) does not inherit refutational completeness trivially
- \( \text{DPLL}(\mathcal{T}) \) has depth-first search: complete for ground SMT problems, not when injecting non-ground inferences
- Solution: *iterative deepening* on inference depth
Issues about completeness

- \( \Gamma \) is refutationally complete
- Since \( \Gamma \) does not see all the clauses, DPLL(\( \Gamma + T \)) does not inherit refutational completeness trivially
- DPLL(\( T \)) has depth-first search: complete for ground SMT problems, not when injecting non-ground inferences
- Solution: *iterative deepening* on inference depth
- However refutationally complete only for \( T \) empty
  
  Example: \( \mathcal{R} = \{ x = a \lor x = b \} \), \( P = \emptyset \), \( T \) is arithmetic
  
  Unsat but can’t tell!
Solution

- Sufficient condition for refutational completeness with $\mathcal{T} \neq \emptyset$: $\mathcal{R}$ be *variable-inactive* (tested automatically by $\Gamma$)
  - it implies stable-infiniteness
    (needed for completeness of Nelson-Oppen combination)
  - it excludes cardinality constraints (e.g., $x = a \lor x = b$)
Solution

- Sufficient condition for refutational completeness with $\mathcal{T} \neq \emptyset$: $\mathcal{R}$ be variable-inactive (tested automatically by $\Gamma$)
  - it implies stable-infiniteness
    (needed for completeness of Nelson-Oppen combination)
  - it excludes cardinality constraints (e.g., $x = a \lor x = b$)

- Use iterative deepening on both Deduce and UnsoundIntro to impose also termination: $\text{DPLL}(\Gamma+\mathcal{T})$ gets “stuck” at $k$
How to get decision procedures

To decide satisfiability modulo $\mathcal{T}$ of $\mathcal{R} \cup P$:

- Find sequence of “unsound axioms” $U$
- Show that there exists $k$ s.t. $k$-bounded DPLL($\Gamma + \mathcal{T}$) is guaranteed to terminate
  - with $Unsat$ if $\mathcal{R} \cup P$ is $\mathcal{T}$-unsat
  - in a state which is not stuck at $k$ if $\mathcal{R} \cup P$ is $\mathcal{T}$-sat
Decision procedures

- \( \mathcal{R} \) has single monadic function symbol \( f \)
- *Essentially finite*: if \( \mathcal{R} \cup P \) is sat, has model where range of \( f \) is *finite*
- Such a model satisfies \( f^j(x) \simeq f^k(x) \) for some \( j \neq k \)
Decision procedures

- $R$ has single monadic function symbol $f$
- **Essentially finite**: if $R \cup P$ is sat, has model where range of $f$ is finite
- Such a model satisfies $f^j(x) \simeq f^k(x)$ for some $j \neq k$
- *UnsoundIntro* adds “pseudo-axioms” $f^j(x) \simeq f^k(x)$ for $j > k$
- Use $f^j(x) \simeq f^k(x)$ as rewrite rule to limit term depth
Decision procedures

- $\mathcal{R}$ has single monadic function symbol $f$
- *Essentially finite*: if $\mathcal{R} \cup P$ is sat, has model where range of $f$ is finite
- Such a model satisfies $f^j(x) \simeq f^k(x)$ for some $j \neq k$
- *UnsoundIntro* adds “pseudo-axioms” $f^j(x) \simeq f^k(x)$ for $j > k$
- Use $f^j(x) \simeq f^k(x)$ as rewrite rule to limit term depth
- Clause length limited by properties of $\Gamma$ and $\mathcal{R}$
- Only finitely many clauses generated: termination without getting stuck
Situations where clause length is limited

$\Gamma$: Superposition, Hyperresolution, Simplification

Negative selection: only positive literals in positive clauses are active

- $\mathcal{R}$ is Horn
- $\mathcal{R}$ is \textit{ground-preserving}: variables in positive literals appear also in negative literals; the only positive clauses are ground
Concrete examples of essentially finite theories

Axiomatizations of type systems:

- **Reflexivity** \( x \sqsubseteq x \) (1)
- **Transitivity** \( \neg(x \sqsubseteq y) \lor \neg(y \sqsubseteq z) \lor x \sqsubseteq z \) (2)
- **Anti-Symmetry** \( \neg(x \sqsubseteq y) \lor \neg(y \sqsubseteq x) \lor x \simeq y \) (3)
- **Monotonicity** \( \neg(x \sqsubseteq y) \lor f(x) \sqsubseteq f(y) \) (4)
- **Tree-Property** \( \neg(z \sqsubseteq x) \lor \neg(z \sqsubseteq y) \lor x \sqsubseteq y \lor y \sqsubseteq x \) (5)

\[ \text{MI} = \{(1), (2), (3), (4)\}: \text{type system with multiple inheritance} \]
\[ \text{SI} = \text{MI} \cup \{(5)\}: \text{type system with single inheritance} \]
Concrete examples of decision procedures

\[ \text{DPLL}(\Gamma + \mathcal{T}) \text{ with } \text{UnsoundIntro adding } f^j(x) \simeq f^k(x) \text{ for } j > k \]
decides the satisfiability modulo \( \mathcal{T} \) of problems

- \( \text{MI} \cup P \) (MI is Horn)
- \( \text{SI} \cup P \) (all ground-preserving except Reflexivity)
- \( \text{MI} \cup \text{TR} \cup P \) and \( \text{SI} \cup \text{TR} \cup P \) (by combination)

\[ \text{TR} = \{ \neg(g(x) \simeq \text{null}), \ h(g(x)) \simeq x \} \]
where \( g \) represents the type representative of a type.
Summary of contributions and directions for future work

- DPLL(Γ+Τ) + unsound TP: termination
- Decision procedures for type systems with multiple/single inheritance used in ESC/Java and Spec#
- DPLL(Γ+Τ) + variable-inactivity: completeness for Τ ≠ ∅ and combination of both built-in and axiomatized theories
- Extension to more presentations
  (e.g., \( y \sqsubseteq x \land u \sqsubseteq v \supset map(x, u) \sqsubseteq map(y, v) \))
- Avoid duplication of reasoning on ground unit clauses