Decision procedures with unsound theorem proving for software verification

Maria Paola Bonacina[1]

Dipartimento di Informatica
Università degli Studi di Verona
Verona, Italy

September 2009

[1] Joint work with Chris Lynch (Clarkson U., NY) and Leonardo de Moura (MSR, Redmond, WA)

Motivation: reasoning about software

- Problem statement
- Combining strengths of different reasoners

Unsound theorem proving

DPLL($\Gamma + \mathcal{T}$): Superposition within SMT-solver

Decision procedures for type systems

Discussion

Motivation

- Software is everywhere
- Needed: *Reliability*
- Difficult goal: Software may be
  - Artful
  - Complex
  - Huge
  - Varied
  - Old (and undocumented)
- Software/hardware border: blurred, evolving

Some approaches to software reliability

- Testing (test case generation ...)
- Programmer assistants
- Program analyzers
- Static analysis (types, extended static checking, abstract interpretations ...)
- Dynamic analysis (traces ...)
- Software model checkers (+ *theorem proving*, e.g., SMT-BMC, CEGAR-SMC)

Reasoning about software

Systems with reasoning about software

Typical architecture:

- *Front-end*: interface, problem modelling, compiling
  - From programs to formulæ
    (via specifications, annotations, intermediate languages)
- *Back-end*: problem solving by **reasoning engine**
  - **Problem**: determine *satisfiability* of formulæ
  - **Objective**: decision procedures

Ingredients of formulæ

- Propositional logic (PL): $\vee$, $\neg$, $\wedge$
- Equality: $\simeq$, $\not\simeq$, free constant and function symbols
- Theories of *data structures*, e.g.:
  - Lists, recursive data structures: constructors (*cons*), selectors (*car*, *cdr*)
  - Arrays, records: *select*, *store*
  - Bitvectors
- Linear arithmetic: $\leq, +, -, \ldots -2, -1, 0, 1, 2, \ldots$
- Formalizations of type systems, e.g.: subtype relation $\sqsubseteq$, type constructor `Array-of` (monadic function f)
- First-order logic (FOL): $\forall$, $\exists$, free predicate symbols

Why quantifiers

- To capture frame conditions over loops
- To summarize auxiliary invariants over heaps
- To axiomatize *type systems*
- To supply *axioms of theories without decision procedure* for ground formulæ

Shape of problems

- 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*)

What do we need

- $\mathcal{T}_i$-solvers: *Satisfiability procedures* for the $\mathcal{T}_i$'s, e.g. *Simplex method* for linear arithmetic
- Davis-Putnam-Logemann-Loveland (DPLL) procedure for SAT
- 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 to handle $\mathcal{R}$ (additional theory): Resolution+Rewriting+Superposition: *Superposition-based*

Combining strengths of different reasoning engines

- DPLL: SAT-problems; large non-Horn clauses
- Theory solvers: linear arithmetic, bitvectors
- DPLL($\mathcal{T}$)-based SMT solvers: efficient, scalable, integrated theory reasoning
- Superposition-based inference system Γ:
  - equalities, Horn clauses, universal quantifiers
  - known to be a sat-procedure for several theories of data structures

How about termination?

- During development conjectures are usually **false** because of mistakes in implementation or specification
- Need a theorem prover that **terminates on satisfiable** inputs
- Not possible in general:
  - FOL is only semi-decidable
  - First-order formulæ of linear arithmetic with uninterpreted function: not even semi-decidable

However we need less than a general solution!

Problematic axioms do occur in relevant inputs

Let $\sqsubseteq$ be a subtype relation and f a type constructor

- *Transitivity*
  $\neg(x \sqsubseteq y) \vee \neg(y \sqsubseteq z) \vee x \sqsubseteq z$
- *Monotonicity*
  $\neg(x \sqsubseteq y) \vee f(x) \sqsubseteq f(y)$

Resolution generates unbounded number of clauses

In practice we need finitely many

Example:

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

Example

1. $\neg(x \sqsubseteq y) \vee f(x) \sqsubseteq f(y)$
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 $\square$: backtrack!

Example

1. $\neg(x \sqsubseteq y) \vee f(x) \sqsubseteq f(y)$
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
- *UnitPropagate*: propagate consequences of assignment
- *Conflict*: detect $L_1 \vee \ldots \vee L_n$ all false
- *Learn*: detect by resolution L_i made false by assignment (not propagation)
- *Backjump*: undo assignment for L_i
- *Unsat*: conflict clause is $\square$ (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 Γ in DPLL($\mathcal{T}$)

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

- **Note**: don't need Γ for ground inferences
- Use each engine for what is best for:
  - non-ground clauses: seen only by Γ
  - ground non-unit clauses: seen only by DPLL($\mathcal{T}$)
  - ground unit clauses: seen by both

DPLL($\Gamma + \mathcal{T}$)

State of derivation: $M \parallel F$

- *Deduce*: Γ-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 $C \vee D$:
    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 $\lceil C \rceil \triangleright C$
  - $\lceil C \rceil$: new propositional variable (a "name" for C)
  - Treat "unnatural failure" like "natural failure"

Unsound theorem proving in DPLL($\Gamma + \mathcal{T}$)

State of derivation: $M \parallel F$

Inference rule:

- *UnsoundIntro*: add $\lceil C \rceil \triangleright C$ to F and $\lceil C \rceil$ to M

Example as done by system

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

Example as done by system

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

1. Add $\lceil f(x) \simeq x \rceil \triangleright f(x) \simeq x$
2. Rewrite $a \sqsubseteq f(c)$ into $\lceil f(x) \simeq x \rceil \triangleright a \sqsubseteq c$

Example as done by system

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

1. Add $\lceil f(x) \simeq x \rceil \triangleright f(x) \simeq x$
2. Rewrite $a \sqsubseteq f(c)$ into $\lceil f(x) \simeq x \rceil \triangleright a \sqsubseteq c$
3. Generate $\lceil f(x) \simeq x \rceil \triangleright \Box$; Backtrack, learn $\neg \lceil f(x) \simeq x \rceil$

Example as done by system

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

1. Add $\lceil f(x) \simeq x \rceil \triangleright f(x) \simeq x$
2. Rewrite $a \sqsubseteq f(c)$ into $\lceil f(x) \simeq x \rceil \triangleright a \sqsubseteq c$
3. Generate $\lceil f(x) \simeq x \rceil \triangleright \square$; Backtrack, learn $\neg \lceil f(x) \simeq x \rceil$
4. Add $\lceil f(f(x)) \simeq x \rceil \triangleright f(f(x)) \simeq x$
5. $a \sqsubseteq b$ yields only $f(a) \sqsubseteq f(b)$
6. $a \sqsubseteq f(c)$ yields only $f(a) \sqsubseteq f(f(c))$
   hence $\lceil f(f(x)) = x \rceil \triangleright f(a) \sqsubseteq c$
7. Reach saturated state and detect satisfiability

Issues about completeness

- Γ is refutationally complete
- Since Γ sees only non-ground clauses, DPLL($\Gamma + \mathcal{T}$) does not inherit refutational completeness trivially

Issues about completeness

- Γ is refutationally complete
- Since Γ sees only non-ground clauses, DPLL($\Gamma + \mathcal{T}$) does not inherit refutational completeness trivially
- 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

- Γ is refutationally complete
- Since Γ sees only non-ground clauses, DPLL($\Gamma + \mathcal{T}$) does not inherit refutational completeness trivially
- DPLL($\mathcal{T}$) 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 $\mathcal{T}$ empty
  Example: $\mathcal{R} = \{x = a \vee x = b\}$, $P = \emptyset$, $\mathcal{T}$ is arithmetic
  Unsat but can't tell!

Issues about completeness (continued)

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

Issues about completeness (continued)

- Sufficient condition for refutational completeness with $\mathcal{T} \neq \emptyset$:
- $\mathcal{R}$ be *variable inactive* (a technical but simple condition)
  - it implies stable-infiniteness
    (needed for completeness of Nelson-Oppen combination)
  - it excludes cardinality constraints (e.g., $x = a \vee x = b$)
- Use *iterative deepening* on both *Deduce* and *UnsoundIntro* to impose also termination: 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

- $\mathcal{R}$ 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

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 Γ and $\mathcal{R}$
- Only finitely many clauses generated: termination without getting stuck

Concrete examples of essentially finite theories

Decision procedures for axiomatizations of type systems:

$$
\begin{array}{rll}
\text{Reflexivity} & x \sqsubseteq x & (1) \\
\text{Transitivity} & \neg(x \sqsubseteq y) \vee \neg(y \sqsubseteq z) \vee x \sqsubseteq z & (2) \\
\text{Anti-Symmetry} & \neg(x \sqsubseteq y) \vee \neg(y \sqsubseteq x) \vee x \simeq y & (3) \\
\text{Monotonicity} & \neg(x \sqsubseteq y) \vee f(x) \sqsubseteq f(y) & (4) \\
\text{Tree-Property} & \neg(z \sqsubseteq x) \vee \neg(z \sqsubseteq y) \vee x \sqsubseteq y \vee y \sqsubseteq x & (5)
\end{array}
$$

$\mathsf{MI} = \{(1), (2), (3), (4)\}$: type system with *multiple inheritance*
$\mathsf{SI} = \mathsf{MI} \uplus \{(5)\}$: type system with *single inheritance*

Summary of contributions

- Unsound theorem proving [Lynch at CSL 2004]
- Variable-inactivity [Bonacina et al. at IJCAR 2006]
- DPLL(Γ) [de Moura, Bjorner at IJCAR 2008]

- DPLL($\Gamma + \mathcal{T}$) + variable-inactivity: extend completeness ($\mathcal{T} \neq \emptyset$)
- DPLL($\Gamma + \mathcal{T}$) + unsound TP: termination
- Decision procedures for type systems with multiple/single inheritance used in ESC/Java and Spec#

[Bonacina, Lynch, de Moura at CADE 2009]