DPLL($\Gamma+\mathcal{T}$): a new style of reasoning for program checking

Maria Paola Bonacina

Dipartimento di Informatica
Università degli Studi di Verona
Verona, Italy, EU

Talk given at the Institute of Software, Chinese Academy of Sciences
Beijing, P.R. China

9 June 2011

Motivation: reasoning for program checking

A new style of reasoning: DPLL($\Gamma+\mathcal{T}$)

Speculative inferences for decision procedures

Current work: interpolation

Motivation: reasoning for program checking

Program checking and automated reasoning

- **Program checking**:
  Design computer programs that (help to) check whether computer programs satisfy desired properties
- **Automated reasoning**:
  Design computer programs that (help to) check whether formulæ follow from other formulæ: *theorem proving* and *model building*

Some motivation for program checking

- Software is everywhere
- Needed: *Reliability*
- Difficult goal: Software may be
  - Artful
  - Complex
  - Huge
  - Varied
  - Old (and undocumented)
  - Less standardized than hardware

Many approaches to program checking

- *Testing*: automated test case generation, (semi-)automated testing ...
- *Static analysis*: type systems, data-flow analysis, control-flow analysis, pointer analysis, symbolic execution, abstract interpretation ...
- *Dynamic analysis*: traces, abstract interpretation ...
- *Software model checking*: BMC, CEGAR, SMT-MC ...
- *Deductive verification*: weakest precondition calculi, verification conditions generation and proof ...

None of them suffices alone

- A *pipeline of tools* for program checking, where
  - Problems of increasing difficulty are attacked by
  - Approaches of increasing power (and cost)
- Most methods for program checking apply *logic*
- Most can benefit from automated reasoning
- Automated reasoning *is* artificial intelligence
- Automated reasoning for program checking *is* artificial intelligence

Problem statement

- Decide *satisfiability* of first-order formulæ generated by SW verification tools (verifying compiler, static analyzer, test generator, synthesizer, model checker)
- Satisfiability w.r.t. *background theories*
- With *quantifiers* to write
  - invariants about loops, heaps, data structures ...
  - axioms of *application-specific theories* without decision procedure (*type systems*)
- Emphasis on *automation*: prover called by other tools

Shape of problem

- Background theory $\mathcal{T}$
  - $\mathcal{T} = \bigcup_{i=1}^{n} \mathcal{T}_i$, e.g., linear arithmetic
- 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) typically with $\mathcal{T}$-symbols
- Determine whether $\mathcal{R} \cup P$ is *satisfiable* modulo $\mathcal{T}$ (Equivalently: determine whether $\mathcal{T} \cup \mathcal{R} \cup P$ is *satisfiable*)

Some key state-of-the-art reasoning methods

- 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 combination by *equality sharing* of the $\mathcal{T}_i$-sat procedures
- First-order engine Γ to handle $\mathcal{R}$ (additional theory): Resolution+Rewriting+Superposition: *Superposition-based*

A new style of reasoning: DPLL($\Gamma+\mathcal{T}$)

How to combine their strengths?

- DPLL: SAT-problems; large non-Horn clauses
- Theory solvers: e.g., ground equality, linear arithmetic
- DPLL($\mathcal{T}$)-based SMT-solver: efficient, scalable, integrated theory reasoning
- Superposition-based inference system Γ:
  - FOL+= clauses with *universally quantified variables* (*automated* instantiation)
  - Sat-procedure for several theories of data structures (e.g., lists, arrays, records)

Superposition-based inference system Γ

- Generic, FOL$+=$, axiomatized theories
- Deduce clauses from clauses (*expansion*)
- Remove redundant clauses (*contraction*)
- Well-founded *ordering* $\succ$ on terms and literals to restrict expansion and define contraction
- Semi-decision procedure: empty clause (contradiction) generated, return *unsat*
- No backtracking

Ordering-based inferences

Ordering $\succ$ on terms and literals to

- restrict *expansion inferences*
- define *contraction inferences*

Complete Simplification Ordering:

- *stable*: if $s \succ t$ then $s\sigma \succ t\sigma$
- *monotone*: if $s \succ t$ then $l[s] \succ l[t]$
- *subterm property*: $l[t] \succeq t$
- *total* on ground terms and literals

Inference system Γ

State of derivation: set of clauses F

- Expansion rules:
  - *Resolution*: resolve maximal complementary literals
  - *Paramodulation/Superposition*: resolution with equality built-in: superpose maximal side of maximal equation into maximal literal/side
- Contraction rules:
  - *Simplification*: by well-founded rewriting
  - *Subsumption*: eliminate less general clauses

Superposition-based satisfiability procedures

- *Termination* results by analysis of inferences: Γ is $\mathcal{T}$-satisfiability procedure
- Covered theories include: *lists*, *arrays* and *records* with or without extensionality, *recursive data structures*

Joint works with Alessandro Armando, Mnacho Echenim, Michaël Rusinowitch, Silvio Ranise and Stephan Schulz

Combination of theories

- If Γ terminates on $\mathcal{R}_i$-sat problems, it terminates on $\mathcal{R}$-sat problems for $\mathcal{R} = \bigcup_{i=1}^{n} \mathcal{R}_i$, if $\mathcal{R}_i$'s *disjoint* and *variable-inactive*
- Variable-inactivity: no maximal literal $t \simeq x$ where $x \notin Var(t)$ (no superposition from variables)
- Only inferences across theories: *superpositions from shared constants*
- Variable inactivity implies stable infiniteness:
  Γ reveals lack of stable infiniteness by generating *cardinality constraint* (e.g., $y \simeq x \vee y \simeq z$) not variable-inactive

Joint works with Alessandro Armando, Silvio Ghilardi, Enrica Nicolini, Silvio Ranise, Stephan Schulz and Daniele Zucchelli

DPLL and DPLL($\mathcal{T}$)

- Propositional logic, ground problems in built-in theories
- Build candidate model M
- Decision procedure:
  model found: return *sat*;
  failure: return *unsat*
- Backtracking

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

If $\mathcal{T}_i$-solver builds $\mathcal{T}_i$-model (*model-based theory combination*):

- *PropagateEq*: add to M a ground $s \simeq t$ true in $\mathcal{T}_i$-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*: $(L_1 \wedge \ldots \wedge L_n) \triangleright (L'_1 \vee \ldots L'_m)$ interpreted as $\neg L_1 \vee \ldots \vee \neg L_n \vee L'_1 \vee \ldots \vee L'_m$
- Inferred clauses inherit hypotheses from premises

Joint work with Leonardo de Moura and Chris Lynch
on top of work by Nikolaj Bjørner and Leonardo de Moura

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

State of derivation: $M \parallel F$

- *Expansion*: take as premises *non-ground* clauses from F and $\mathcal{R}$-literals (unit clauses) from M and add result to F
- *Backjump*: remove hypothetical clauses depending on undone assignments
- *Contraction*: as above + *scope level* to prevent situation where clause is deleted, but clauses that make it redundant are gone because of backjumping

DPLL($\Gamma+\mathcal{T}$): expansion inferences

- *Deduce*: Γ-rule γ, e.g., superposition, using *non-ground* clauses $\{H_1 \triangleright C_1, \ldots, H_m \triangleright C_m\}$ in F and ground $\mathcal{R}$-literals $\{L_{m+1}, \ldots, L_n\}$ in M

  $$M \parallel F \quad \Longrightarrow \quad M \parallel F, H \triangleright C$$

  where $H = H_1 \cup \ldots \cup H_m \cup \{L_{m+1}, \ldots, L_n\}$
  and γ infers C from $\{C_1, \ldots, C_m, L_{m+1}, \ldots, L_n\}$
- Only $\mathcal{R}$-literals: Γ-inferences ignore $\mathcal{T}$-literals
- Take ground unit $\mathcal{R}$-clauses from M as *PropagateEq* puts them there

DPLL($\Gamma+\mathcal{T}$): contraction inferences

- Single premise $H \triangleright C$: apply to C (e.g., *tautology deletion*)
- Multiple premises (e.g., *subsumption*, *simplification*): prevent situation where clause is deleted, but clauses that make it redundant are gone because of backjumping
- *Scope level*:
  - $level(L)$ in $M\ L\ M'$: number of decided literals in $M\ L$
  - $level(H) = max\{level(L) \mid L \in H\}$ and 0 for $\emptyset$

DPLL($\Gamma+\mathcal{T}$): contraction inferences

- Say we have $H \triangleright C$, $H_2 \triangleright C_2, \ldots, H_m \triangleright C_m$, and $L_{m+1}, \ldots, L_n$
- $C_2, \ldots, C_m, L_{m+1}, \ldots, L_n$ simplify C to C' or subsume it
- Let $H' = H_2 \cup \ldots \cup H_m \cup \{L_{m+1}, \ldots, L_n\}$
- Simplification: replace $H \triangleright C$ by $(H \cup H') \triangleright C'$
- Both simplification and subsumption:
  - if $level(H) \geq level(H')$: delete
  - if $level(H) < level(H')$: disable (re-enable when backjumping $level(H')$)

DPLL($\Gamma+\mathcal{T}$) as a transition system

- Search mode: State of derivation $M \parallel F$
  - M sequence of *assigned ground literals*: partial model
  - F set of *hypothetical clauses*
- Conflict resolution mode: State of derivation $M \parallel F \parallel C$
  - C ground conflict clause

Initial state: M empty, F is $\{\emptyset \triangleright C \mid C \in \mathcal{R} \uplus P\}$

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

- *Refutational completeness* of the inference system:
  - from that of Γ, DPLL($\mathcal{T}$) and equality sharing
  - made combinable by variable-inactivity
- *Fairness* of the search plan:
  - depth-first search fair only for ground SMT problems;
  - add *iterative deepening* on *inference depth*

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

Use each engine for what is best at:

- DPLL($\mathcal{T}$) works on ground clauses
- Γ not involved with ground inferences and built-in theory
- Γ works on non-ground clauses and ground unit clauses taken from M: inferences guided by current partial model
- Γ works on $\mathcal{R}$-sat problem

Speculative inferences for decision procedures

How to get decision procedures?

- SW development: **false** conjectures 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 formulæ 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

Example:

1. $\neg(x \sqsubseteq y) \vee f(x) \sqsubseteq f(y)$ (*Monotonicity*)
2. $a \sqsubseteq b$ generates by resolution
3. $\{f^i(a) \sqsubseteq f^i(b)\}_{i \geq 0}$

E.g. $f(a) \sqsubseteq f(b)$ or $f^2(a) \sqsubseteq f^2(b)$ often suffice to show satisfiability

Idea: Allow speculative inferences

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

Idea: Allow speculative inferences

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!

Idea: Allow speculative inferences

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. Terminate and detect satisfiability

Speculative inferences in DPLL($\Gamma+\mathcal{T}$)

- Speculative inference: add *arbitrary* clause C
- To induce termination on sat input
- What if it 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)
  - Speculative inferences are *reversible*

Speculative inferences in DPLL($\Gamma+\mathcal{T}$)

State of derivation: $M \parallel F$

Inference rule:

- *SpeculativeIntro*: add $\lceil C \rceil \triangleright C$ to F and $\lceil C \rceil$ to M
- Rule *SpeculativeIntro* also bounded by iterative deepening

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 $\lceil f(x) \simeq x \rceil \triangleright \Box$; 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 $f(a) \sqsubseteq f(f(c))$
   rewritten to $\lceil f(f(x)) = x \rceil \triangleright f(a) \sqsubseteq c$
7. Terminate and detect satisfiability

Decision procedures with speculative inferences

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

- Find sequence of "speculative 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 $j \neq k$
- *SpeculativeIntro* adds "pseudo-axioms" $f^j(x) \simeq f^k(x)$, $j > k$
- 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 $j \neq k$
- *SpeculativeIntro* adds "pseudo-axioms" $f^j(x) \simeq f^k(x)$, $j > k$
- 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

Situations where clause length is limited

Γ: Superposition, Resolution + negative selection, Simplification

Negative selection: only positive literals in positive clauses are active

- $\mathcal{R}$ is Horn
- $\mathcal{R}$ is *ground-preserving*: variables in positive literals appear also in negative literals;
  the only positive clauses are ground

Axiomatizations of type systems

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

Multiple inheritance: $\mathsf{MI} = \{(1),(2),(3),(4)\}$
Single inheritance: $\mathsf{SI} = \mathsf{MI} \cup \{(5)\}$

Concrete examples of decision procedures

DPLL($\Gamma+\mathcal{T}$) with *SpeculativeIntro* adding $f^j(x) \simeq f^k(x)$ for $j > k$ decides the satisfiability modulo $\mathcal{T}$ of problems

- $\mathrm{MI} \cup P$
- $\mathrm{SI} \cup P$
- $\mathrm{MI} \cup \mathrm{TR} \cup P$ and $\mathrm{SI} \cup \mathrm{TR} \cup P$

where $\mathrm{TR} = \{\neg(g(x) \simeq null),\ h(g(x)) \simeq x\}$

Joint work with Leonardo de Moura and Chris Lynch

Current work: interpolation

What is interpolation?

Given closed formulæ A and B, such that $A \vdash B$,
an *interpolant* is a closed formula I such that

- $A \vdash I$
- $I \vdash B$ and
- I is made of symbols common to A and B.

Craig's interpolation lemma: interpolants for closed formulæ do exist (non-constructive proof)

What is interpolation?

Given closed formulæ A and B, such that $A, B \vdash \bot$,
a *reverse interpolant* is a closed formula I such that

- $A \vdash I$
- $I, B \vdash \bot$ and
- I is made of symbols common to A and B.

Reasoning modulo theories: $\vdash_{\mathcal{T}}$
$\mathcal{T}$-symbols are regared as common

Why interpolation?

Several applications in SW verification, e.g.:

- *Abstraction refinement* in software model checking
- *Invariant generation*
- Annotation improvement

Intuition: a formula in between formulæ: information on intermediate states

Interpolation system

- Given: proof (refutation) of $A \cup B$ (A and B sets of clauses)
- Terminology: *A-colored*, *B-colored*, *transparent*
- Interpolation system: extracts interpolant of (A, B) from proof
- How? Attaching to each clause in the proof a *partial interpolant*
- The partial interpolant of $\Box$ is the interpolant of (A, B)

Partial interpolant

- *Partial interpolant* $PI(C)$ of clause C in refutation of $A \cup B$: interpolant of $g_A(C) = A \wedge \neg(C|_A)$ and $g_B(C) = B \wedge \neg(C|_B)$.
- If C is $\Box$: $PI(C)$ is an interpolant of (A, B).
- Requirements:
  - $g_A(C) \vdash PI(C)$ or $A \wedge \neg(C|_A) \vdash PI(C)$
  - $g_B(C) \wedge PI(C) \vdash \perp$ or $B \wedge \neg(C|_B) \wedge PI(C) \vdash \perp$, and
  - $PI(C)$ is transparent.
- Complete interpolation system

State of the art

- Interpolation systems for resolution proofs in propositional logic: HKPYM, MM (DPLL)
- Interpolation systems for theories: equality, linear rational arithmetic, linear integer arithmetic, arrays without extensionality
- Interpolation system for equality sharing
- Putting them all together: interpolation system for DPLL($\mathcal{T}$)

Current work

- Interpolation system for Γ
  - Ground proofs
  - Non-ground proofs: investigating restrictions
- Interpolation system for DPLL($\Gamma+\mathcal{T}$)

Joint work with Moa Johansson

Acknowledgements

Thanks to all my co-authors

and

Thank you!