Trends and Challenges in Satisfiability Modulo Theories

Cesare Tinelli

The University of Iowa

Intro: Validity Modulo Theories

In a number of CS applications one is interested in

determining the validity of a first-order sencence wrt a *background theory*, a distinguished set $\mathcal{T}$ of first-order models

A formula φ is *$\mathcal{T}$-valid* if it is satisfied by every model of $\mathcal{T}$

Example

$$x + y > 0 \land y < 0 \rightarrow x > 0$$

is $\mathcal{T}$-valid if $\mathcal{T}$ is the set of all expansion of $\mathbb{Z}$ to the free constants x, y, z

Validity Modulo Theories in a Nutshell

Distinguishing Feature

$\mathcal{T}$-validity may be determined more efficiently using specialized methods on $\mathcal{T}$ as opposed to general-purpose first-order reasoning

Validity Modulo Theories in a Nutshell

Distinguishing Feature

$\mathcal{T}$-validity may be determined more efficiently using specialized methods on $\mathcal{T}$ as opposed to general-purpose first-order reasoning

Main Issue

Tension between the scope of background theories and the efficiency of their validity checkers

Validity Modulo Theories in a Nutshell

Distinguishing Feature

$\mathcal{T}$-validity may be determined more efficiently using specialized methods on $\mathcal{T}$ as opposed to general-purpose first-order reasoning

Main Issue

Tension between the scope of background theories and the efficiency of their validity checkers

A lot of theoretical and practical work in

1. identifying *fragments* of theories with efficient checkers
2. enlarging theories and/or their fragments by using several specialized checkers in cooperation

Theory fragments with efficient checkers

Examples

- Universal fragment of theory of equality (over some signature Σ)
- Universal, linear fragment of theory of $\mathbb{R}$
- Universal, difference constraints fragment of theory of $\mathbb{N}$
- Universal fragment of theory of arrays with extensionality
- Universal fragment of theories of inductive data types

Why Satisfiability Modulo Theories?

- Validity Modulo Theories' dual problem

- More popular setting because most validity checkers are refutation-based
  (and so are actually unsatisfiability checkers)

- Terminology originated with SMT-LIB initiative in 2003

- SMT acronym caught on and is now widely used

Goals of This Talk

- Give an overview of SMT and its applications
- Present main approaches and issues
- Discuss some long-standing challenges
- Highlight some new challenges for the field

Applications of SMT

Applications of SMT

- Type checking
  - statically verifying the well-typedness of programs
- Model checking of reactive (in)finite state systems
  - verifying safety properties
  - abstraction/refinement
- Model-based test-case generation
  - generating better test sets
- Specification checking
  - checking the consistency of formal specifications

Applications of SMT

- Extended static checking/static analysis
  - verifying the absence of certain run-time errors
- Optimizing/certifying compilers
  - verifying correctness of optimizations
  - verifying PCC
- Full functional verification
  - supporting proofs of inductive invariants
  - supporting interactive proofs

Main SMT Approaches

Main SMT Approaches

Small engines approaches

- *Eager* encodings to propositional logic
  Typically relying on fast SAT solvers
- *Lazy* encodings to propositional logic
  Typically relying on DPLL solvers + theory solvers (decision procedures)
- *Hybrid* encodings, i.e., eager encodings to other decidable logics:
  - QF fragment of bit vectors
  - QF fragment of linear arithmetic with free symbols

Main SMT Approaches

Big engines approaches

- Eager, specialized encodings to FOL=

  Relying on superposition engine + proper reduction orderings

Claim of the Day

All these approaches can be seen as different instances of a common logical abstraction/refinement framework

Claim of the Day

All these approaches can be seen as different instances of a common logical abstraction/refinement framework

Let's see that

Claim of the Day

All these approaches can be seen as different instances of a common logical abstraction/refinement framework

Let's see that

Diclaimer
The following formal presentation of the framework is somewhat wishy-washy

A proper treatment can be given, using, e.g., the theory of institutions or similar theoretical tools

A Few Technicalities: Logics in Abstract

A logic $\mathcal{L}$ is tuple $(Lan_{\mathcal{L}}, Mod_{\mathcal{L}}, \models_{\mathcal{L}}, Ref_{\mathcal{L}})$ where

- $Lan_{\mathcal{L}}$ is a set of *formulas*
- $Mod_{\mathcal{L}}$ is a set of *models*
- $\models_{\mathcal{L}}$ is a *satisfiability relation* $\subseteq Mod_{\mathcal{L}} \times Lan_{\mathcal{L}}$
- $Ref_{\mathcal{L}}$ is a *refutation system*

A Few Technicalities: Logics in Abstract

A logic $\mathcal{L}$ is tuple $(Lan_{\mathcal{L}}, Mod_{\mathcal{L}}, \models_{\mathcal{L}}, Ref_{\mathcal{L}})$ where

- $Lan_{\mathcal{L}}$ is a set of *formulas*
- $Mod_{\mathcal{L}}$ is a set of *models*
- $\models_{\mathcal{L}}$ is a *satisfiability relation* $\subseteq Mod_{\mathcal{L}} \times Lan_{\mathcal{L}}$
- $Ref_{\mathcal{L}}$ is a *refutation system*

A formula φ is *$\mathcal{L}$-(un)satisfiable* if there is some (no) $\mathcal{A} \in Mod_{\mathcal{L}}$ s.t. $\mathcal{A} \models_{\mathcal{L}} \varphi$

A Few Technicalities: Logics in Abstract

A logic $\mathcal{L}$ is tuple $(Lan_{\mathcal{L}}, Mod_{\mathcal{L}}, \models_{\mathcal{L}}, Ref_{\mathcal{L}})$ where

- $Lan_{\mathcal{L}}$ is a set of *formulas*
- $Mod_{\mathcal{L}}$ is a set of *models*
- $\models_{\mathcal{L}}$ is a *satisfiability relation* $\subseteq Mod_{\mathcal{L}} \times Lan_{\mathcal{L}}$
- $Ref_{\mathcal{L}}$ is a *refutation system*

Typically,

- $Lan_{\mathcal{L}}$ is closed under negation ($\neg$) and conjunction ($\wedge$)
- $Ref_{\mathcal{L}}$ is a (semi)-decision procedure for $\mathcal{L}$-unsatisfiability
  we write $\varphi \vdash_{\mathcal{L}} \bot$ if $Ref_{\mathcal{L}}$ returns "unsatisfiable" for φ

A Few Technicalities: Logics in Abstract

A logic $\mathcal{L}$ is tuple $(Lan_{\mathcal{L}}, Mod_{\mathcal{L}}, \models_{\mathcal{L}}, Ref_{\mathcal{L}})$ where

- $Lan_{\mathcal{L}}$ is a set of *formulas*
- $Mod_{\mathcal{L}}$ is a set of *models*
- $\models_{\mathcal{L}}$ is a *satisfiability relation* $\subseteq Mod_{\mathcal{L}} \times Lan_{\mathcal{L}}$
- $Ref_{\mathcal{L}}$ is a *refutation system*

SMT works with logics where

- $Lan_{\mathcal{L}}$ is a fragment of FOL
- $Mod_{\mathcal{L}}$ is the set of models of some FOL theory $\mathcal{T}$
- $\models_{\mathcal{L}}$ is often decidable, and by efficient methods

Efficiency Abstractions

- For efficiency, SMT methods universally resort to

  some reduction of $\mathcal{L}$-satisfiability to satisfiability in one or more simpler, and more efficient, logics

- The reduction is achieved by a (possibly incremental) abstraction/refinement process

Logic Abstractions

A logic $\hat{\mathcal{L}}$ *effectively abstracts* a logic $\mathcal{L}$ if there are computable mappings

$$(\_)^{\mathrm{a}} : Lan_{\mathcal{L}} \longrightarrow Lan_{\hat{\mathcal{L}}} \qquad (\_)^{\mathrm{a}} : Mod_{\mathcal{L}} \longrightarrow Mod_{\hat{\mathcal{L}}}$$
$$(\_)^{\mathrm{c}} : Lan_{\hat{\mathcal{L}}} \longrightarrow Lan_{\mathcal{L}}$$

s.t.

1. $(\varphi^{\mathrm{a}})^{\mathrm{c}}$ is *equisatisfiable* with φ in $\mathcal{L}$
2. $(\_)^{\mathrm{a}} : Mod_{\mathcal{L}} \longrightarrow Mod_{\hat{\mathcal{L}}}$ is surjective
3. if $\mathcal{A} \models_{\mathcal{L}} \varphi$ then $\mathcal{A}^{\mathrm{a}} \models_{\hat{\mathcal{L}}} \varphi^{\mathrm{a}}$

$\mathcal{L}$*-satisfiability by Abstraction Refinement*

Proposition $\varphi^{\mathrm{a}} \vdash_{\hat{\mathcal{L}}} \perp \quad \Rightarrow \quad \varphi^{\mathrm{a}}$ is $\hat{\mathcal{L}}$-unsat $\quad \Rightarrow \quad \varphi$ is $\mathcal{L}$-unsat

- So, if we abstract φ and $\hat{\mathcal{L}}$'s inference systems finds φ^{a} unsatisfiable, we are done

$\mathcal{L}$-satisfiability by Abstraction Refinement

Proposition $\varphi^{\mathrm{a}} \vdash_{\hat{\mathcal{L}}} \perp \quad \Rightarrow \quad \varphi^{\mathrm{a}}$ is $\hat{\mathcal{L}}$-unsat $\quad \Rightarrow \quad \varphi$ is $\mathcal{L}$-unsat

- So, if we abstract φ and $\hat{\mathcal{L}}$'s inference systems finds φ^{a} unsatisfiable, we are done
- If $\mathcal{A} \models_{\hat{\mathcal{L}}} \varphi^{\mathrm{a}}$ for some $\mathcal{A}$, φ may still be $\mathcal{L}$-unsatisfiable

$\mathcal{L}$-satisfiability by Abstraction Refinement

Proposition $\varphi^{\mathrm{a}} \vdash_{\hat{\mathcal{L}}} \bot \quad \Rightarrow \quad \varphi^{\mathrm{a}}$ is $\hat{\mathcal{L}}$-unsat $\quad \Rightarrow \quad \varphi$ is $\mathcal{L}$-unsat

- So, if we abstract φ and $\hat{\mathcal{L}}$'s inference systems finds φ^{a} unsatisfiable, we are done
- If $\mathcal{A} \models_{\hat{\mathcal{L}}} \varphi^{\mathrm{a}}$ for some $\mathcal{A}$, φ may still be $\mathcal{L}$-unsatisfiable
- In that case, we

$\mathcal{L}$-satisfiability by Abstraction Refinement

Proposition $\varphi^{\mathrm{a}} \vdash_{\hat{\mathcal{L}}} \bot \quad \Rightarrow \quad \varphi^{\mathrm{a}}$ is $\hat{\mathcal{L}}$-unsat $\quad \Rightarrow \quad \varphi$ is $\mathcal{L}$-unsat

- So, if we abstract φ and $\hat{\mathcal{L}}$'s inference systems finds φ^{a} unsatisfiable, we are done
- If $\mathcal{A} \models_{\hat{\mathcal{L}}} \varphi^{\mathrm{a}}$ for some $\mathcal{A}$, φ may still be $\mathcal{L}$-unsatisfiable
- In that case, we

  1. compute some ψ such that $\varphi \models_{\mathcal{L}} \psi$ but $\mathcal{A} \not\models_{\hat{\mathcal{L}}} \psi^{\mathrm{a}}$

$\mathcal{L}$-satisfiability by Abstraction Refinement

Proposition $\varphi^{\mathrm{a}} \vdash_{\hat{\mathcal{L}}} \bot \quad \Rightarrow \quad \varphi^{\mathrm{a}}$ is $\hat{\mathcal{L}}$-unsat $\quad \Rightarrow \quad \varphi$ is $\mathcal{L}$-unsat

- So, if we abstract φ and $\hat{\mathcal{L}}$'s inference systems finds φ^{a} unsatisfiable, we are done
- If $\mathcal{A} \models_{\hat{\mathcal{L}}} \varphi^{\mathrm{a}}$ for some $\mathcal{A}$, φ may still be $\mathcal{L}$-unsatisfiable
- In that case, we
  1. compute some ψ such that $\varphi \models_{\mathcal{L}} \psi$ but $\mathcal{A} \not\models_{\hat{\mathcal{L}}} \psi^{\mathrm{a}}$
  2. check the $\hat{\mathcal{L}}$-satisfiability of $\varphi^{\mathrm{a}} \wedge \psi^{\mathrm{a}}$

$\mathcal{L}$-satisfiability by Abstraction Refinement

Proposition $\varphi^{\mathrm{a}} \vdash_{\hat{\mathcal{L}}} \bot \quad \Rightarrow \quad \varphi^{\mathrm{a}}$ is $\hat{\mathcal{L}}$-unsat $\quad \Rightarrow \quad \varphi$ is $\mathcal{L}$-unsat

- So, if we abstract φ and $\hat{\mathcal{L}}$'s inference systems finds φ^{a} unsatisfiable, we are done
- If $\mathcal{A} \models_{\hat{\mathcal{L}}} \varphi^{\mathrm{a}}$ for some $\mathcal{A}$, φ may still be $\mathcal{L}$-unsatisfiable
- In that case, we
  1. compute some ψ such that $\varphi \models_{\mathcal{L}} \psi$ but $\mathcal{A} \not\models_{\hat{\mathcal{L}}} \psi^{\mathrm{a}}$
  2. check the $\hat{\mathcal{L}}$-satisfiability of $\varphi^{\mathrm{a}} \wedge \psi^{\mathrm{a}}$
- Key to efficiency is how and when to compute and add the *refinement formula* ψ

Why we abstract

Typically,

- we have an efficient, sound and complete $Ref_{\hat{\mathcal{L}}}$ and
- we also have an efficient, sound but incomplete $Ref_{\mathcal{L}}$
- $Ref_{\mathcal{L}}$ is complete for a subset of $Lan_{\mathcal{L}}$

Why we abstract

Typically,

- we have an efficient, sound and complete $Ref_{\hat{\mathcal{L}}}$ and
- we also have an efficient, sound but incomplete $Ref_{\mathcal{L}}$
- $Ref_{\mathcal{L}}$ is complete for a subset of $Lan_{\mathcal{L}}$

A good abstraction and a proper refinement strategy can yield an efficient and complete refutation system for $\mathcal{L}$ thorough the cooperation of $Ref_{\hat{\mathcal{L}}}$ and $Ref_{\mathcal{L}}$

Why we abstract

Typically,

- we have an efficient, sound and complete $Ref_{\hat{\mathcal{L}}}$ and
- we also have an efficient, sound but incomplete $Ref_{\mathcal{L}}$
- $Ref_{\mathcal{L}}$ is complete for a subset of $Lan_{\mathcal{L}}$

Even when completeness is out of reach, abstraction/refinement is still useful to improve accuracy, i.e., a higher number of correctly classified unsat queries

Prototypical Refutation System $Ref_{\hat{\mathcal{L}}}$

Expansion Rules

$$\frac{\Gamma, \Delta}{\Gamma, \Delta, \Delta'} \quad \text{(*)}$$

Contraction Rules

$$\frac{\Gamma, \Delta}{\Gamma} \quad \text{(*)}$$

Splitting Rules

$$\frac{\Gamma, \Delta}{\Gamma, \Delta, \Delta_1 \mid \cdots \mid \Gamma, \Delta, \Delta_n} \quad \text{(*)}, n \geq 2$$

Closing Rules

$$\frac{\Gamma, \Delta}{\bot} \quad \text{(*)}$$

(*) some condition on Δ

$\Gamma, \Delta_{(i)}$ sets of $\hat{\mathcal{L}}$-formulas

$Ref_{\hat{\mathcal{L}}}$ ***with*** $\mathcal{L}$ ***Refinement***

Expansion Rules

$$\frac{\Gamma, \Delta}{\Gamma, \Delta, \Delta'} \quad (*)$$

Contraction Rules

$$\frac{\Gamma, \Delta}{\Gamma} \quad (*)$$

Splitting Rules

$$\frac{\Gamma, \Delta}{\Gamma, \Delta, \Delta_1 \mid \cdots \mid \Gamma, \Delta, \Delta_n} \quad (*),\ n \geq 2$$

Closing Rules

$$\frac{\Gamma, \Delta}{\perp} \quad (*)$$

Refinement Rules

$$\frac{\Gamma, \Delta}{\Gamma, \Delta, \varphi^{\mathrm{a}}} \quad \text{if } (*),\ \Delta^{\mathrm{c}} \models_{\mathcal{L}} \varphi$$

(*) some condition on Δ

Example: Eager Reduction to SAT

$$
\begin{array}{lcl}
\mathcal{L} & = & \text{Integer Difference Logic} \\
Lan_{\mathcal{L}} & = & \text{Boolean combinations of } x - y < \pm n \text{ atoms} \\
Mod_{\mathcal{L}} & = & \text{expansions of } \mathbb{Z} \text{ to free constants}
\end{array}
$$

$$
\begin{array}{lcl}
\hat{\mathcal{L}} & = & \text{propositional logic} \\
Lan_{\hat{\mathcal{L}}} & = & \text{CNF formulas} \\
Mod_{\hat{\mathcal{L}}} & = & \text{propositional models} \\
Ref_{\hat{\mathcal{L}}} & = & \text{any SAT solver}
\end{array}
$$

Example: Eager Reduction to SAT

$\mathcal{L}$	=	Integer Difference Logic
$Lan_{\mathcal{L}}$	=	Boolean combinations of $x - y < \pm n$ atoms
$Mod_{\mathcal{L}}$	=	expansions of $\mathbb{Z}$ to free constants
$\hat{\mathcal{L}}$	=	propositional logic
$Lan_{\hat{\mathcal{L}}}$	=	CNF formulas
$Mod_{\hat{\mathcal{L}}}$	=	propositional models
$Ref_{\hat{\mathcal{L}}}$	=	any SAT solver

Abstraction:

φ^{a} is a Boolean abstraction of φ's CNF

Refinement:

Selected ground instances of IDL axioms over constants in φ
(more or less . . .)

Example: Eager Reduction to SAT

$\mathcal{L}$	=	Integer Difference Logic
$Lan_{\mathcal{L}}$	=	Boolean combinations of $x - y < \pm n$ atoms
$Mod_{\mathcal{L}}$	=	expansions of $\mathbb{Z}$ to free constants
$\hat{\mathcal{L}}$	=	propositional logic
$Lan_{\hat{\mathcal{L}}}$	=	CNF formulas
$Mod_{\hat{\mathcal{L}}}$	=	propositional models
$Ref_{\hat{\mathcal{L}}}$	=	any SAT solver

Proof strategy:

1. Start with $\Gamma = \{\varphi^{\mathrm{a}}\}$
2. Apply refinement rules so that Γ^{c} becomes equisat with φ
3. Apply other rules to Γ (i.e., give Γ to SAT solver)

Example: Eager Reduction to FOL$_=$

$\mathcal{L}$	=	Arrays with extensionality
$Lan_{\mathcal{L}}$	=	Boolean combinations of read/write atoms
$Mod_{\mathcal{L}}$	=	expansions of array models to free constants
$\hat{\mathcal{L}}$	=	FOL with equality
$Lan_{\hat{\mathcal{L}}}$	=	FOL clauses
$Mod_{\hat{\mathcal{L}}}$	=	models of FOL with equality
$Ref_{\hat{\mathcal{L}}}$	=	any superposition-based prover

Example: Eager Reduction to FOL$_=$

$\mathcal{L}$	=	Arrays with extensionality
$Lan_{\mathcal{L}}$	=	Boolean combinations of read/write atoms
$Mod_{\mathcal{L}}$	=	expansions of array models to free constants
$\hat{\mathcal{L}}$	=	FOL with equality
$Lan_{\hat{\mathcal{L}}}$	=	FOL clauses
$Mod_{\hat{\mathcal{L}}}$	=	models of FOL with equality
$Ref_{\hat{\mathcal{L}}}$	=	any superposition-based prover

Abstraction:

φ^{a} is a certain *flat form* of φ's CNF

Refinement:

Array axioms

Example: Eager Reduction to FOL$_=$

$\mathcal{L}$	=	Arrays with extensionality
$Lan_{\mathcal{L}}$	=	Boolean combinations of read/write atoms
$Mod_{\mathcal{L}}$	=	expansions of array models to free constants
$\hat{\mathcal{L}}$	=	FOL with equality
$Lan_{\hat{\mathcal{L}}}$	=	FOL clauses
$Mod_{\hat{\mathcal{L}}}$	=	models of FOL with equality
$Ref_{\hat{\mathcal{L}}}$	=	any superposition-based prover

Proof strategy:

1. Start with $\Gamma = \{\varphi^{\mathrm{a}}\}$
2. Apply refinement rules to add array axioms
3. Apply other rules to Γ (i.e., give Γ to superposition prover)

Example: Eager Reduction to FOL$_=$

$\mathcal{L}$	=	Arrays with extensionality
$Lan_{\mathcal{L}}$	=	Boolean combinations of read/write atoms
$Mod_{\mathcal{L}}$	=	expansions of array models to free constants
$\hat{\mathcal{L}}$	=	FOL with equality
$Lan_{\hat{\mathcal{L}}}$	=	FOL clauses
$Mod_{\hat{\mathcal{L}}}$	=	models of FOL with equality
$Ref_{\hat{\mathcal{L}}}$	=	any superposition-based prover

Termination Conditions:

- $\Gamma = \{\bot\}$ or
- Γ is saturated (*)

(*) Termination is guaranteed with proper reduction ordering

Superposition Rules

Expansion Rules

Superposition Right

$$\frac{\Gamma, C \vee s[u] = t, C' \vee u' = v'}{\Gamma, C \vee s[u] = t, C' \vee u' = v', \mu(C \vee C' \vee s[v'] = t)} \quad \text{if} \begin{cases} \mu = mgu(u', u), \\ \ldots \end{cases}$$

...

Splitting Rules

None

Closing Rules

Fail $\dfrac{\Gamma, \Box}{\bot}$

Superposition Rules

Contraction Rules

Subsumption $\dfrac{\Gamma, C, C'}{\Gamma, C}$ if $\sigma(C) \subseteq C$ for some σ, ...

Deletion $\dfrac{\Gamma, C \vee t = t}{\Gamma}$

...

Refinement Rules

$\mathcal{T}$-Axiom $\dfrac{\Gamma}{\Gamma, C}$ if C is an axiom of $\mathcal{T}$

Example: Lazy Reduction to SAT (DPLL($\mathcal{T}$))

$$
\begin{array}{lcl}
\mathcal{L} & = & \text{QF fragment of some theory } \mathcal{T} \\
Lan_{\mathcal{L}} & = & \text{Boolean combinations of } \mathcal{T}\text{-atoms} \\
Mod_{\mathcal{L}} & = & \text{expansions of models of } \mathcal{T} \text{ to free constants}
\end{array}
$$

$$
\begin{array}{lcl}
\hat{\mathcal{L}} & = & \text{propositional logic} \\
Lan_{\hat{\mathcal{L}}} & = & \text{CNF formulas} \\
Mod_{\hat{\mathcal{L}}} & = & \text{propositional models} \\
Ref_{\hat{\mathcal{L}}} & = & \text{DPLL-based solver}
\end{array}
$$

Example: Lazy Reduction to SAT (DPLL($\mathcal{T}$))

$\mathcal{L}$ = QF fragment of some theory $\mathcal{T}$

$Lan_{\mathcal{L}}$ = Boolean combinations of $\mathcal{T}$-atoms

$Mod_{\mathcal{L}}$ = expansions of models of $\mathcal{T}$ to free constants

$\hat{\mathcal{L}}$ = propositional logic

$Lan_{\hat{\mathcal{L}}}$ = CNF formulas

$Mod_{\hat{\mathcal{L}}}$ = propositional models

$Ref_{\hat{\mathcal{L}}}$ = DPLL-based solver

Abstraction:

φ^{a} is a Boolean abstraction of φ's CNF

Refinement:

Selected ground theorems or unit consequences of Γ^{c} in $\mathcal{T}$

Example: Lazy Reduction to SAT (DPLL($\mathcal{T}$))

$$
\begin{array}{lcl}
\mathcal{L} & = & \text{QF fragment of some theory } \mathcal{T} \\
Lan_{\mathcal{L}} & = & \text{Boolean combinations of } \mathcal{T}\text{-atoms} \\
Mod_{\mathcal{L}} & = & \text{expansions of models of } \mathcal{T} \text{ to free constants}
\end{array}
$$

$$
\begin{array}{lcl}
\hat{\mathcal{L}} & = & \text{propositional logic} \\
Lan_{\hat{\mathcal{L}}} & = & \text{CNF formulas} \\
Mod_{\hat{\mathcal{L}}} & = & \text{propositional models} \\
Ref_{\hat{\mathcal{L}}} & = & \text{DPLL-based solver}
\end{array}
$$

Proof strategy:

1. Start with $\Gamma = \{\varphi^{\mathrm{a}}\}$
2. Apply a mix of DPLL and refinement rules (*)

(*) Termination guaranteed by mild restrictions on rule application mix

Example: Lazy Reduction to SAT (DPLL($\mathcal{T}$))

$\mathcal{L}$	=	QF fragment of some theory $\mathcal{T}$
$Lan_{\mathcal{L}}$	=	Boolean combinations of $\mathcal{T}$-atoms
$Mod_{\mathcal{L}}$	=	expansions of models of $\mathcal{T}$ to free constants
$\hat{\mathcal{L}}$	=	propositional logic
$Lan_{\hat{\mathcal{L}}}$	=	CNF formulas
$Mod_{\hat{\mathcal{L}}}$	=	propositional models
$Ref_{\hat{\mathcal{L}}}$	=	DPLL-based solver

Termination Conditions:

- $\Gamma = \{\bot\}$ (on all branches) or
- Δ^{c} is $\mathcal{T}$-consistent and $\Delta \models \varphi^{a}$, where $\Delta = \{$ literals of Γ $\}$

DPLL($\mathcal{T}$) Rules

Expansion Rules

Unit Propagation $$\frac{\Gamma, l_1, \ldots, l_n, \bar{l}_1 \vee \cdots \vee \bar{l}_n \vee l}{\Gamma, l_1, \ldots, l_n, \bar{l}_1 \vee \cdots \vee \bar{l}_n \vee l, l}$$

Learn $$\frac{\Gamma}{\Gamma, C} \quad \text{if } \Gamma \models C$$

Splitting Rules

Split $$\frac{\Gamma}{\Gamma, l \;\mid\; \Gamma, \bar{l}} \quad \text{if neither } l \text{ nor } \bar{l} \text{ is in } \Gamma$$

Closing Rules

Fail $$\frac{\Gamma, l_1, \ldots, l_n, \bar{l}_1 \vee \cdots \vee \bar{l}_n}{\bot}$$

DPLL($\mathcal{T}$) Rules

Contraction Rules

Forget $\dfrac{\Gamma, C}{\Gamma}$ if $\Gamma \models C$

Restart $\dfrac{\Gamma, \Delta}{\Gamma}$ if $\Delta =$ propagated and splitting literals

Refinement Rules

$\mathcal{T}$-Learn $\dfrac{\Gamma}{\Gamma, C}$ if $\models_{\mathcal{T}} C^{\mathrm{c}}$, atoms of C from Γ

$\mathcal{T}$-Propagate $\dfrac{\Gamma, \Delta}{\Gamma, \Delta, l}$ if Δ set of literals, atom of l from Γ, $\Delta^{\mathrm{c}} \models_{\mathcal{T}} l^{\mathrm{c}}$

Example: Eager Reduction to LRA+UF

$\mathcal{L}$ = finite multisets

$Lan_{\mathcal{L}}$ = Boolean combinations of multiset atoms

$Mod_{\mathcal{L}}$ = expansions of multiset models to free constants

$\hat{\mathcal{L}}$ = linear real arithmetic with uninterpreted symbols

$Lan_{\hat{\mathcal{L}}}$ = Boolean combination of linear expressions

$Mod_{\hat{\mathcal{L}}}$ = expansions of Reals to free symbols

$Ref_{\hat{\mathcal{L}}}$ = DPLL(LRA+UF) solver

Proof strategy:

1. Start with $\Gamma = \{\varphi^{\mathrm{a}}\}$
2. Apply refinement rules until Γ^{c} is equisat with φ
3. Apply other rules to Γ (i.e., give Γ to DPLL(T) solver)

Theory Combinations as Refinement

When $\mathcal{T} = \mathcal{T}_1 + \ldots + \mathcal{T}_n$ combination methods apply

Theory Combinations as Refinement

When $\mathcal{T} = \mathcal{T}_1 + \ldots + \mathcal{T}_n$ combination methods apply

Theory Combinations as Refinement

When $\mathcal{T} = \mathcal{T}_1 + \ldots + \mathcal{T}_n$ combination methods apply

Eager approaches
Reduce $\mathcal{T}_1 + \ldots + \mathcal{T}_n$ to some theory $\mathcal{T}_0$

Theory Combinations as Refinement

When $\mathcal{T} = \mathcal{T}_1 + \ldots + \mathcal{T}_n$ combination methods apply

Eager approaches
Reduce $\mathcal{T}_1 + \ldots + \mathcal{T}_n$ to some theory $\mathcal{T}_0$

Lazy approaches
DPLL($\mathcal{T}_1, \ldots, \mathcal{T}_n$) with Nelson-Oppen combination

Query abstraction involves purification

Refinement is done per theory, with refinement formulas including *shared* equalities

Long-standing Issue in SMT: Quantifiers

Long-standing Issue in SMT: Quantifiers

- Most SMT solvers accept only ground formulas
- In most cases, queries are ground formulas

Long-standing Issue in SMT: Quantifiers

- Most SMT solvers accept only ground formulas
- In most cases, queries are ground formulas
- Dealing with quantified formula is however a real and frequent need

Long-standing Issue in SMT: Quantifiers

- Most SMT solvers accept only ground formulas
- In most cases, queries are ground formulas
- Dealing with quantified formula is however a real and frequent need
- Here is why

Creeping Quantifiers

Often, we want ground satisfiability in a theory $\mathcal{T}_{\mathrm{Full}}$

Creeping Quantifiers

Often, we want ground satisfiability in a theory $\mathcal{T}_{\mathrm{Full}}$

However, we only have a solver for ground satisfiability in a *subtheory* $\mathcal{T}$ of $\mathcal{T}_f$ with

- $\mathcal{T}$'s signature $\subseteq \mathcal{T}_{\mathrm{Full}}$'s signature
- $\mathcal{T}$'s theorems $\subseteq \mathcal{T}_{\mathrm{Full}}$'s theorems

Creeping Quantifiers

Often, we want ground satisfiability in a theory $\mathcal{T}_{\mathrm{Full}}$

However, we only have a solver for ground satisfiability in a *subtheory* $\mathcal{T}$ of $\mathcal{T}_f$ with

- $\mathcal{T}$'s signature $\subseteq$ $\mathcal{T}_{\mathrm{Full}}$'s signature
- $\mathcal{T}$'s theorems $\subseteq$ $\mathcal{T}_{\mathrm{Full}}$'s theorems

We then approximate $\mathcal{T}_{\mathrm{Full}}$-satisfiability with $\mathcal{T}$-satisfiability of $\Gamma \cup \Phi$ where

- Φ is the original ground query and
- Γ is a fixed, selected set of quantified axioms of $\mathcal{T}_{\mathrm{Full}}$ that are not theorems of $\mathcal{T}$

Creeping Quantifiers: Example

$\mathcal{T}$: Theory of integers and lists (with only $\mathsf{cons}, \mathsf{nil}, \mathsf{head}, \mathsf{tail}$)

$\mathcal{T}_{\mathrm{Full}}$: Theory of integers and lists with length function

Γ : $\{\mathsf{len}(\mathsf{nil}) = 0,\ \forall x, y.\ \mathsf{len}(\mathsf{cons}(x, y)) = \mathsf{len}(y) + 1\}$

Creeping Quantifiers: Example

$\mathcal{T}$: Theory of integers and lists (with only $\mathsf{cons}, \mathsf{nil}, \mathsf{head}, \mathsf{tail}$)

$\mathcal{T}_{\mathrm{Full}}$: Theory of integers and lists with length function

Γ : $\{\mathsf{len}(\mathsf{nil}) = 0,\ \forall x, y.\ \mathsf{len}(\mathsf{cons}(x, y)) = \mathsf{len}(y) + 1\}$

Note $\mathcal{T} \cup \Gamma$ is weaker (strictly in this example) than $\mathcal{T}_{\mathrm{Full}}$ but stronger than $\mathcal{T}$

But we can catch more $\mathcal{T}_{\mathrm{Full}}$-unsatisfiable formulas if we check the $\mathcal{T}$-satisfiability of $\Gamma \cup \Phi$ instead of just Φ

Creeping Quantifiers: Example

$\mathcal{T}$: Theory of integers and lists (with only $\mathsf{cons}, \mathsf{nil}, \mathsf{head}, \mathsf{tail}$)

$\mathcal{T}_{\mathrm{Full}}$: Theory of integers and lists with length function

Γ : $\{\mathsf{len}(\mathsf{nil}) = 0,\ \forall x, y.\ \mathsf{len}(\mathsf{cons}(x, y)) = \mathsf{len}(y) + 1\}$

Note $\mathcal{T} \cup \Gamma$ is weaker (strictly in this example) than $\mathcal{T}_{\mathrm{Full}}$ but stronger than $\mathcal{T}$

But we can catch more $\mathcal{T}_{\mathrm{Full}}$-unsatisfiable formulas if we check the $\mathcal{T}$-satisfiability of $\Gamma \cup \Phi$ instead of just Φ

Problem How to deal with quantifiers in Γ?

Creeping Quantifiers: Example

$\mathcal{T}$: Theory of integers and lists (with only $\mathsf{cons}, \mathsf{nil}, \mathsf{head}, \mathsf{tail}$)

$\mathcal{T}_{\mathrm{Full}}$: Theory of integers and lists with length function

Γ : $\{\mathsf{len}(\mathsf{nil}) = 0,\ \forall x, y.\ \mathsf{len}(\mathsf{cons}(x, y)) = \mathsf{len}(y) + 1\}$

Note $\mathcal{T} \cup \Gamma$ is weaker (strictly in this example) than $\mathcal{T}_{\mathrm{Full}}$ but stronger than $\mathcal{T}$

But we can catch more $\mathcal{T}_{\mathrm{Full}}$-unsatisfiable formulas if we check the $\mathcal{T}$-satisfiability of $\Gamma \cup \Phi$ instead of just Φ

Problem How to deal with quantifiers in Γ?

(Still) Current Solution Logical abstraction and then refinement via heuristic quantifier instantiation

Heuristic Instantiation as Generic Refinement

The case of DPLL($\mathcal{T}$) systems

1. Abstract each quantified subformula $Q\mathbf{x}.\ \varphi(\mathbf{x})$ in the query by a fresh Boolean predicate P
2. If P gets ever asserted, refine it by adding one or more instances of $\varphi(\mathbf{x})$ as needed

Heuristic Instantiation as Generic Refinement

The case of DPLL($\mathcal{T}$) systems

1. Abstract each quantified subformula $Q\mathbf{x}.\ \varphi(\mathbf{x})$ in the query by a fresh Boolean predicate P
2. If P gets ever asserted, refine it by adding one or more instances of $\varphi(\mathbf{x})$ as needed

Main Challenges When, how and how much to instantiate

State of the art Patterns, (incomplete) $\mathcal{T}$-matching
See Wednesday morning's talks

Beyond Decision Procedures

As SMT solvers get be embedded in more and different tools more complex forms of outputs are being asked

E.g.

- Unsatisfiable cores
- Proofs
- Interpolants
- Models

Each of these introduces challenges of its own

Unsatisfiable Cores

When Γ is $\mathcal{T}$-unsatisfiable, return minimally $\mathcal{T}$-unsatisfiable subsets of Γ

Uses Conflict analysis, intelligent backtracking

Challenges Minimization is a hard problem, even for simple theories

Approaches Compute *almost minimal* sets

Proofs

When Γ is $\mathcal{T}$-unsatisfiable, produce a proof in some suitable proof system

Uses Embedding in untrusting tools, interpolant generation

Challenges Minimization of overhead, tradeoff between proof size and rule granularity, choice of the proof system

Approaches Several, no unifying themes yet

Interpolants

When $\Gamma_1 \cup \Gamma_2$ is $\mathcal{T}$-unsatisfiable, return a $\mathcal{T}$-interpolant of Γ_1 and Γ_2
(a formula I whose free symbols occur in Γ_1 and Γ_2, and s.t. $\Gamma_1 \models_{\mathcal{T}} I$ and $\Gamma_2, I \models_{\mathcal{T}} \perp$)

Uses Model checking

Challenges New topic, few known interpolating procedures, tricky combination issues

Approaches Eager reduction to LRA+UF

Models

When Γ is satisfiable, return a concrete assignment of values to its free-symbols

Uses Counter-example generation in model checking/ESC/verification, test-case generation

Challenges Potential exponential overhead (difference between sat-checking and constraint solving), compact representation of solutions, combination of solutions

Approaches Mining constraint solving research, more work needed on model generation modulo theories

Foundational Issues

Which version of FOL= is best for SMT?

More concretely, which type system?

Foundational Issues

Which version of FOL= is best for SMT?

More concretely, which type system?

- Unsorted
- Many-sorted
- Order-subsorted
- With predicate subtyping
- With parametrized types
- With dependent types

Foundational Issues

Which version of FOL= is best for SMT?

More concretely, which type system?

Current trend Towards more sophisticated type systems

Rationale Simplifies combination/refinement issues

Challenges Increases complexity of refutation systems, persistent belief that types are mostly a nuisance

Practical Issues

- Embedding of SMT solvers into other tools
- Interoperability of SMT solvers
- Standardization of API's and input/output formats
- Availability of benchmarks
- Comparative experimental evaluations

Practical Issues

- Embedding of SMT solvers into other tools
- Interoperability of SMT solvers
- Standardization of API's and input/output formats
- Availability of benchmarks
- Comparative experimental evaluations

Being addressed by the SMT-LIB initiative

More info at `www.smt-lib.org`

Thank you