Finite Model Finding in Satisfiability Modulo Theories

Andrew Reynolds
December 3, 2013
University of Iowa
Motivation

• Many aspects of modern life are dependent upon software

• Correctness of software is often highly critical
  – Flight control, Bank records, Medical Devices, ...

• Growing need for automated reasoning
  – For software verification and other applications
Approaches to Automated Reasoning

• **Boolean Satisfiability Solvers**
  – Fast, Handle Decidable Logic
  – Cons: May be difficult to encode problem into SAT

• **Automated First-Order Theorem Provers**
  – Handle problems in an expressive natural encoding
  – Cons: Logic can be Undecidable

• **Alternative**: **Satisfiability Modulo Theories (SMT)**
  – Incorporate specialized procedures for theories
    • Arithmetic, bitvectors, arrays, datatypes, …
  – Many problems can be expressed as SMT problems
SMT Solvers

- **SMT solvers** are powerful tools that
  - Are used in many formal methods applications
  - Have optimized performance due to combination of:
    - Off-the-shelf SAT solver
    - Fast decision procedures for (ground) constraints
  - May generate:
    - **Proofs**
      - Theorem proving, software/hardware verification
    - **Models**
      - Failing instances of aforementioned applications
      - Invariant synthesis, scheduling, test case generation
SMT: Limitations

• Ongoing challenge: *quantified* formulas
  – Are useful for:
    • Frame axioms in software verification
    • Universal safety properties
    • Axiomatization of unsupported theories
    • ...
  – Needed by a growing number of SMT-based applications

• Current methods for handling quantifiers in SMT:
  – **Heuristic** methods for answering “UNSAT”
  – **Limited** capability of answering “SAT”
    • Often will return “UNKNOWN” after some effort
Contributions

• **Finite Model Finding in SMT**
  – New approach for handling quantifiers in SMT
  – Different from **ATP** finite model finders:
    • Native support for **background theories**
  – Different from **SMT** solvers:
    • Increased ability to answer “satisfiable”
Outline

• **Intro** to Satisfiability Modulo Theories (SMT)
• Finite Model Finding in SMT
  – Details of Approach
  – Theoretical Properties
  – Experimental Results
• Extension to Bounded Integer Quantification
Satisfiability Modulo Theories

$(f(a) = b \lor f(a) = c) \land (c + 1 = b) \land f(c) = g(c)$
Satisfiability Modulo Theories

\[( f(a) = b \lor f(a) = c ) \land c+1 = b \land f(c) = g(c) \]

\[\downarrow\] Abstract to propositional logic

\[( A \lor B ) \land C \land D \]
Satisfiability Modulo Theories

\[(f(a) = b \lor f(a) = c) \land (c+1 = b) \land f(c) = g(c)\]

\[(A \lor B) \land C \land D\]

Find satisfying assignment: A, C, D
Satisfiability Modulo Theories

\[( f(a) = b \lor f(a) = c ) \land c + 1 = b \land f(c) = g(c) \]

\[ ( A \lor B ) \land C \land D \]

Find satisfying assignment: \( A, C, D \)

Check T-consistency: \( f(a) = b, c + 1 = b, f(c) = g(c) \)

\[ \Rightarrow \text{This can be done with ground theory solver} \]
SMT with Quantified Formulas

\[(f(a) = b \lor f(a) = c) \land (c + 1 = b) \land (\forall x. f(x) = g(x))\]

Find satisfying assignment: A, C, D

Check T-consistency: \[f(a) = b, c + 1 = b, (\forall x. f(x) = g(x))\]

• Satisfying assignment contains *quantified formulas*  
  \[\Rightarrow\] Challenge: This is generally *undecidable*
DPLL(T) Architecture

Formula $F$

SAT Solver

F is sat

Satisfying assignment $A$ for $F$

Theory Solvers

F is unsat

$A$ is T-Consistent

Clauses to add to $F$  

$A$ is T-Inconsistent

UNSAT, proof

SAT, model
**DPLL(T) Architecture: Challenge**

- **Challenge:** What if determining the consistency of $\mathcal{A}$ is difficult?
- **For quantified formulas, determining T-consistency is **undecidable**
If sat assignment contains quantified formula $Q$,

- **Heuristically** add instances of $Q$ to $F$ [Detlefs et al 2003]
  - Typically based on pattern matching
  - May discover refutation, if right instances are added
  - **No way to answer SAT**
Why Models are Important

- Verification Condition for $P$
- (with quantifiers)

SMT solver

-UnSAT

- Unknown

- Property $P$ is verified

Manual Inspection

Candidate Model
Why Models are Important

SMT solver

Verification Condition for P

(With quantifiers)

UNSAT

Unknown

Candidate Model

SAT

Concrete counterexample for Property P

Property P is verified

Manual Inspection

Allowed for Property P
Model-Based Approach for Quantifiers

• Given:
  – Set of ground formulas $\mathcal{F}$
  – Set of universally quantified formulas $\mathcal{Q}$

• To determine the satisfiability of $\mathcal{F} \land \mathcal{Q}$,
  – Construct candidate models for $\mathcal{Q}$, based on satisfying assignments for $\mathcal{F}$
    • Model-Based Quantifier Instantiation (MBQI)
      – [Ge/deMoura 2009]
DPLL(T) Architecture (Extended)

Ground Formulas $F$

SAT Solver

Theory Solvers

Model Verifier

Satisfying assignment $A$ for $F$

$A$ is T-Consistent

Candidate model $M$

$M$ is a model for $Q$

UNSAT, proof

Clauses to add to $F$

else

$F$ is sat

else

else
When can we represent/check models for $Q$?

• **Focus of thesis:** Finite Model Finding
  – Limited to quantifiers over:
    • Uninterpreted sorts
      – Can represent memory addresses, values, sets, etc.
    • Other **finite sorts**
      – Fixed width bitvectors, datatypes, ...
• **Useful in applications:**
  – Software verification, automated theorem proving
Running Example

\[
\begin{align*}
\text{distinct}(\text{NewYork, Boston, Seattle}) \\
\text{travels}(\text{person}_1, \text{Boston}) \\
\neg \text{salesman}(\text{person}_2) \\
\text{salesman}(\text{person}_3)
\end{align*}
\]

\[
\forall x : \text{Person}, y : \text{City}. \\
\text{salesman}(x) \implies \text{travels}(x,y)
\]
Running Example

**F**
- SAT Solver
- Theory Solvers
- Model Verifier
- SAT, model M
- Clauses to add to F

**Q**
- Candidate model M

**F**
- distinct(NewYork, Boston, Seattle)
- travels(person₁, Boston)
- ¬salesman(person₂)
- salesman(person₃)
- ∀ x y . salesman(x) ⇒ travels(x,y)

**Q**
- UNSAT, proof
Find Satisfying Assignment $A$ for $F$

- $A$ is Theory-Consistent according to the theory of equality
Construct Candidate Model $M$ from $A$

\[ A := \{ \text{distinct}(\text{NewYork, Boston, Seattle}), \]
\[ \text{travels}(\text{person}_1, \text{Boston}), \]
\[ \neg \text{salesman}(\text{person}_2), \]
\[ \text{salesman}(\text{person}_3) \} \]

\[ M := \]
\[ \text{travels} : \]
\[ \text{person}_1, \text{Boston} \rightarrow \text{true} \]
\[ ... \rightarrow \text{false} \]
\[ \text{salesman} : \]
\[ \text{person}_3 \rightarrow \text{true} \]
\[ ... \rightarrow \text{false} \]
Determine if $M$ satisfies $Q$

$M :=$

travels :

person$_1$, Boston $\rightarrow$ true
...
$\rightarrow$ false

salesman :

person$_3$ $\rightarrow$ true
...
$\rightarrow$ false

$Q : \forall xy. \text{salesman}(x) \implies \text{travels}(x,y)$
Add Clauses back to $\bar{F}$

$M :=$

\[
\begin{align*}
\text{travels :} & \quad \text{person}_1, \text{Boston} \rightarrow \text{true} \\
& \quad \ldots \rightarrow \text{false} \\
\text{salesman :} & \quad \text{person}_3 \rightarrow \text{true} \\
& \quad \ldots \rightarrow \text{false}
\end{align*}
\]

$Q : \forall xy. \text{salesman}(x) \Rightarrow \text{travels}(x, y)$

$\Psi[x, y]$

- $\Psi$ is false for person$_3$, NewYork
- Add $\Psi[ \text{person}_3, \text{NewYork} ]$ to $\bar{F}$
- Will rule out $M$ on next iteration
  - Model “refinement” process
Finding Small Models: Motivation

$M :=$

Person : \{ person_1, person_2, person_3 \}  
City : \{ NewYork, Boston, Seattle \}  
...

$Q : \forall x : \text{Person}, y : \text{City}.  
\text{salesman}(x) \Rightarrow \text{travels}(x, y)$

- Naïvely, to determine whether $M$ is model for $Q$:
  - Check if $M$ satisfies all instances $S$ of $Q$

- Challenge: $S$ can be very large
  - For $Q$ with $n$ vars, domain size $d$, $|S|$ can be $O(d^n)$
    - In example, $3 \times 3 = 9$

- Solution:
  - Search for candidate models with small domain sizes
    - Use finite cardinality constraints [CAV 2013]
Outline of Approach

1. Satisfying assignment $A$ for $F$
   - $F$ is satisfiable
2. Candidate model $M$
   - $A$ is T-consistent
3. SAT model $M$
   - Clauses to add to $F$

• Require methods for:
  1. Finding satisfying assignments
     • Esp. ones that induce models with small domain sizes
  2. Building candidate models
  3. Checking candidate models
1. Finding satisfying assignments
2. Building candidate models
3. Checking candidate models
Finding Minimal Models in DPLL(T)

- **Idea**: fix domain sizes incrementally $1, 2, 3, \ldots$

  \[ \text{distinct}(\text{NewYork}, \text{Boston}, \text{Seattle}) \]
  
  \[ \neg \text{salesman}(\text{person}_1) \]
  
  \[ \neg \text{salesman}(\text{person}_2) \]

- **Facts**:
  - $|\text{Person}| \leq 1$
  - $\neg |\text{Person}| \leq 1$
  - $|\text{Person}| \leq 2$
  - $|\text{Person}| \leq 3$
  - $\neg |\text{Person}| \leq 2$
  - $\neg |\text{Person}| \leq 3$

  \[ \text{travels}(\text{person}_1, \text{Boston}) \]

- **Search Strategy**:
  - Search for models of size $1$
  - If none exist, search for models of size $2$
  - etc.

**Fixed-Cardinality DPLL(T)**
Finding Minimal Models in DPLL(T)

- **Requires**: method to find cardinality conflicts
  - E.g. determine when > 1, 2, 3, ... “Person” must exist

\[
\begin{align*}
\text{distinct(NewYork, Boston, Seattle)} \\
\text{travels(person}_1, \text{ Boston)} \\
\text{salesman(person}_2) \\
\text{salesman(person}_3)
\end{align*}
\]
Finding Minimal Models in DPLL(T)

- Extend SMT solver with theory solver for:
  \[ \Rightarrow \text{Theory of } \text{EUF} + \text{finite cardinality constraints} \ (\text{EFCC}) \]
Theory Solver for EFCC

• Interested in models $\mathbb{M}$ where:
  – Domain elements of $\mathbb{M}$ are equivalence classes of terms
• Thus, signature of EFCC has predicates of form $|S| \leq k$
  – Satisfied iff $\leq k$ equivalence classes of terms of sort $S$ exist
• To check if cardinality constraints are satisfied:
  – Based on disequality graph $(V, E)$
    • Vertices $V$ are equivalence classes of sort $S$
    • Edges $E$ are disequalities between terms of sort $S$
  – So, $f(a) \neq a, f(a) \neq c, f(c) = c$ becomes:
Theory of EFCC and k-Colorability

• Assume a single sort $S$ with cardinality $k$
  – Check if corresponding $(V, E)$ is $k$-colorable
    • If no, then report a cardinality conflict ($C \Rightarrow \neg |S| \leq k$)
      – where $C$ is an explanation of subgraph that is not $k$-colorable
    • If yes, we cannot be sure that a model of size $k$ exists
      – Due to theory reasoning:

\[ |S| \leq 2 \]

$\Rightarrow$ Must explicitly merge equivalence classes
Theory of EFCC: Challenges

- Why finite cardinality constraints are challenging:
  - Interaction with **theory reasoning**
  - k-colorability is **NP-complete**
  - Analysis must be **incremental**

- Solution:
  - Explicitly merge equivalence classes
  - Use heuristic **region-based** approach which:
    - May quickly detect when disequality graph is not k-colorable
    - Suggests pairs of equivalence classes to merge
Region-Based Approach

- Partition the graph \((V, E)\) into \textit{regions} with high edge density

For \(|S| \leq k\) we maintain the invariant:
- No \textit{clique of size} \(k+1\) exists having nodes from multiple regions

Thus, we only need to search for cliques \textit{local to regions}
- Region can be ignored if it has \(\leq k\) nodes

\(|S| \leq 2\)
Extension to Multiple Sorts

- **Challenge:**
  - Fair Strategy for enumerating cardinalities

- **Example:**
  
  \[
  \text{person}_1 \neq \text{person}_2 \lor \big( \text{city}_1 \neq \text{city}_2 \ldots \text{city}_1 \neq \text{city}_{1000} \ldots \text{city}_{999} \neq \text{city}_{1000} \big)
  \]

  - Formula has model with 2 persons, 1 city
  - But we may search for models where
    - # persons, cities : (1, 1), (1, 2), ..., (1, 1000)
  - With quantified formulas, this leads to *incompleteness*
    - May imply no finite models exist in a branch
Fixed-Cardinality DPLL(T) for Multiple Sorts

- Uses extended signature containing:
  - Boolean predicates of form $|\Sigma| \leq k$
  - Satisfied if and only if $\leq k$ equivalence classes for all sorts exist

$\begin{array}{c}
|\Sigma| \leq 2 \\
\neg |\Sigma| \leq 2 \\
|\Sigma| \leq 3 \\
\neg |\Sigma| \leq 3 \\
|\Sigma| \leq 4 \\
\neg |\Sigma| \leq 4
\end{array}$

Search for models
- # persons, cities: (1,1)
- # persons, cities: (1,2) or (2,1)
- # persons, cities: (1,3), (2,2), or (3,1)

$\Rightarrow$ Gives a fair strategy
Properties : Ground Solver

- For ground inputs $F$,
  - Fixed-cardinality DPLL(T), using Theory EFCC:
    - **Sound, terminating, and complete**
      - Eventually either:
        » Determines $F$ is unsatisfiable
        » Constructs candidate model $M$ of finite (minimal) size
1. Finding satisfying assignments
2. Building candidate models
3. Checking candidate models

SAT Solver

Theory Solvers

Model Verifier

Ground Formulas $F$

Quantified Formulas $Q$

Satisfying assignment $A$ for $F$

Candidate model $M$

$F$ is sat

A is T-Consistent

$M$ is a model for $Q$

Clauses to add to $F$

UNSAT, proof

SAT, model $M$
Model Representation

• Represent a function/predicate as a list of entries:

\[ C_1 \rightarrow v_1, \ldots, C_n \rightarrow v_n \]

  – Where

    • \( C_1, \ldots, C_n \) are “conditions”
    • \( v_1, \ldots, v_n \) are “values”

• E.g. unary predicate “P” true only for \( v \) represented as:

\[ (v) \rightarrow T, (\ast) \rightarrow \bot \]

  – Interpreted as an if-then-else:

\[ \lambda \ x. \ \text{ite}( \ x = v, \ T, \ \bot ) \]
Model Construction

• Candidate models $\mathbb{M}$:
  – Domain elements are equivalence classes $[t_1], [t_2], \ldots$
  – Are constructed from sat assignment $A$ for $F$
  – Consist of definitions $D_f$ for each $f \in \Sigma$, where each $D_f$:
    • Is partially determined by ground equalities from $A$
      – For each equality $f(t_1, ..., t_n) = t$ in $A$, 
        » Entry ( $[t_1], \ldots, [t_n]$ ) $\mapsto [t] \in D_f$
    • Has default value
      – Determined by distinguished $f$-application $e$
        » Entry ( $*, \ldots, *$ ) $\mapsto [e] \in D_f$
Constructing Models : Example

\[ F \]

\[ \begin{align*}
\text{distinct}(\text{NewYork, Boston, Seattle}) \\
\neg \text{travels}(\text{person}_1, \text{Boston}) \\
\neg \text{salesman}(\text{person}_2) \\
\text{salesman}(\text{person}_3) \\
\text{salesman}(\text{person}_1) \Rightarrow \text{travels}(\text{person}_1, \text{NewYork})
\end{align*} \]

\[ Q \]

\[ \forall \ x \ y . \text{salesman}(x) \Rightarrow \text{travels}(x, y) \]

- Guide choice of default values based on:
  - \text{person}_1 for Person
  - \text{NewYork} for City

- Assume \( Q \) has been instantiated with these terms
Constructing Models : Example

- Choose default based on value of \( \text{travels}( \text{person}_1, \text{NewYork}) \)

\[
\begin{align*}
F &= \{ \text{distinct}(\text{NewYork, Boston, Seattle}), \text{travels}(\text{person}_1, \text{Boston}), \lnot \text{salesman}(\text{person}_2), \text{salesman}(\text{person}_3) \} \\
Q &= \forall x y . \text{salesman}(x) \Rightarrow \text{travels}(x, y)
\end{align*}
\]

\[
\begin{align*}
\mathcal{A} &:= \\
&\{ \ldots, \text{travels}(\text{person}_1, \text{Boston}) = T, \text{travels}(\text{person}_1, \text{NewYork}) = T \}
\end{align*}
\]

\[
\begin{align*}
\mathcal{D}_{\text{travels}} : \\
&(\text{person}_1, \text{NewYork}) \rightarrow T, \\
&(\text{person}_1, \text{Boston}) \rightarrow \bot, \\
&(\ast, \ast) \rightarrow T
\end{align*}
\]
SAT Solver

1. Finding satisfying assignments
   - Satisfying assignment $A$ for $F$
   - $F$ is SAT
   - $A$ is $T$-Consistent

2. Building candidate models
   - Candidate model $M$

3. Checking candidate models
   - SAT model $M$

Ground Formulas $F$

UNSAT, proof

Clauses to add to $F$

Theory Solvers

Model Verifier

Quantified Formulas $Q$
• To check if $\mathcal{M}$ is a model for $\mathcal{Q}$:
  – Naïvely, test if every instance of $\mathcal{Q}$ is true in $\mathcal{M}$
  – Or, choose a *representative* set of instances of $\mathcal{Q}$
    • Only add instances that are *false* in $\mathcal{M}$
    • Identify *sets of instances* of $\mathcal{Q}$ that are equisatisfiable
Checking Candidate Models

\[ F \]

distinct(NewYork, Boston, Seattle)
\neg travels(person\_1, Boston)
\neg salesman(person\_2)
salesman(person\_3)
salesman(person\_1) \Rightarrow travels(person\_1, NewYork)

\[ Q \]

\forall \ x \ y. \ salesman(x) \Rightarrow travels(x,y)

\[ D_{salesman}: \]
\hspace{1cm} (person\_2) \rightarrow \bot,
\hspace{1cm} (person\_3) \rightarrow \top,
\hspace{1cm} (*) \rightarrow \top

\[ D_{travels}: \]
\hspace{1cm} (person\_1, NewYork) \rightarrow \top
\hspace{1cm} (person\_1, Boston) \rightarrow \bot,
\hspace{1cm} (*) , (*) \rightarrow \top \}
Checking Candidate Models

\[ \text{distinct}(\text{NewYork}, \text{Boston}, \text{Seattle}) \]
\[ \neg \text{travels}(\text{person}_1, \text{Boston}) \]
\[ \neg \text{salesman}(\text{person}_2) \]
\[ \text{salesman}(\text{person}_3) \]
\[ \text{salesman}(\text{person}_1) \Rightarrow \text{travels}(\text{person}_1, \text{NewYork}) \]

\[ \begin{align*}
\mathcal{D}_{\text{salesman}} : & \\
(\text{person}_2) & \rightarrow \bot, \\
(\text{person}_3) & \rightarrow \top, \\
(* *) & \rightarrow \top
\end{align*} \]

\[ \begin{align*}
\mathcal{D}_{\text{travels}} : & \\
(\text{person}_1, \text{NewYork}) & \rightarrow \top \\
(\text{person}_1, \text{Boston}) & \rightarrow \bot, \\
(*, *) & \rightarrow \top
\end{align*} \]

\[ \forall \ x \ y . \text{salesman}(x) \Rightarrow \text{travels}(x, y) \]
Checking Candidate Model $\mathcal{M}$

- To check if $\mathcal{M}$ satisfies quantified formula $\mathcal{Q}$:
  - Choose representative set of instances $S$ of $\mathcal{Q}$
    $\Rightarrow$ This is somewhat heuristic
  - For each $\Psi$ in $S$,
    - If $\mathcal{M}(\Psi) = \text{false}$, add $\Psi$ to $F$
  - If no instances added, then $\mathcal{M}$ satisfies $\mathcal{Q}$

- Alternate, improved approach:
  - Directly compute the interpretation of $\mathcal{Q}$ in $\mathcal{M}$
    - Using same data structure that represents functions in $\mathcal{M}$
Computing Interpretations of Terms

\[ \forall x \, y . \, \text{salesman}(x) \Rightarrow \text{travels}(x, y) \]

\[ \text{D}_{\text{salesman}(x)}: \]
\[ (\text{person}_2, *) \rightarrow \bot, \]
\[ (\text{person}_3, *) \rightarrow \top, \]
\[ (\ast, \ast) \rightarrow \top \]

\[ \text{D}_{\text{travels}(x, y)}: \]
\[ (\text{person}_1, \text{NewYork}) \rightarrow \top \]
\[ (\text{person}_1, \text{Boston}) \rightarrow \bot, \]
\[ (\ast, \ast) \rightarrow \top \]
Computing Interpretations of Terms

\[ \forall x \ y . \ salesman(x) \Rightarrow travels(x, y) \]

**\( D_{salesman(x)} \):**

\[
\begin{align*}
(person_2, \ast) & \rightarrow \bot, \\
(person_3, \ast) & \rightarrow T, \\
(\ast, \ast) & \rightarrow T
\end{align*}
\]

**\( D_{travels(x,y)} \):**

\[
\begin{align*}
(person_1, \text{NewYork}) & \rightarrow T \\
(person_1, \text{Boston}) & \rightarrow \bot, \\
(\ast, \ast) & \rightarrow T
\end{align*}
\]

\[ D_{salesman(x)} \times D_{travels(x,y)}: \]

\[
\begin{align*}
(person_2, \ast) & \rightarrow (\bot, T), \\
(person_3, \ast) & \rightarrow (T, T), \\
(person_1, \text{NewYork}) & \rightarrow (T, T) \\
(person_1, \text{Boston}) & \rightarrow (T, \bot), \\
(\ast, \ast) & \rightarrow (T, T)
\end{align*}
\]
Computing Interpretations of Terms

\[ Q : \forall x \ y . \ salesman(x) \Rightarrow travels(x,y) \]

\[
\begin{align*}
D_{salesman(x)} : & \quad (person_2, \ast) \rightarrow \bot, \\
& \quad (person_3, \ast) \rightarrow T, \\
& \quad (\ast, \ast) \rightarrow T \\
D_{travels(x,y)} : & \quad (person_1, \text{NewYork}) \rightarrow T \\
& \quad (person_1, \text{Boston}) \rightarrow \bot, \\
& \quad (\ast, \ast) \rightarrow T \\
\end{align*}
\]

\[
\begin{align*}
D_{salesman(x) \times D_{travels(x,y)}} : & \quad (person_2, \ast) \rightarrow (\bot, T), \\
& \quad (person_3, \ast) \rightarrow (T, T), \\
& \quad (person_1, \text{NewYork}) \rightarrow (T, T) \\
& \quad (person_1, \text{Boston}) \rightarrow (T, \bot), \\
& \quad (\ast, \ast) \rightarrow (T, T) \\
\end{align*}
\]

Apply interpreted predicate
Computing Interpretations of Terms

\[ \text{Q} : \forall \; x \; y \; . \; \text{salesman}(x) \Rightarrow \text{travels}(x, y) \]

\[ \text{D}_{\text{salesman}(x)} : \]
- \((\text{person}_2, *) \rightarrow \bot, \)
- \((\text{person}_3, *) \rightarrow T, \)
- \((*, *) \rightarrow T\)

\[ \text{D}_{\text{travels}(x,y)} : \]
- \((\text{person}_1, \text{NewYork}) \rightarrow T\)
- \((\text{person}_1, \text{Boston}) \rightarrow \bot, \)
- \((*, *) \rightarrow T\)  

\[ \text{D}_{\text{salesman}(x) \Rightarrow \text{travels}(x,y)} : \]
- \((\text{person}_2, *) \rightarrow (\bot, T), \)
- \((\text{person}_3, *) \rightarrow (T, T), \)
- \((\text{person}_1, \text{NewYork}) \rightarrow (T, T)\)
- \((\text{person}_1, \text{Boston}) \rightarrow (T, \bot)\)
- \((*, *) \rightarrow (T, T)\)

\[ \Rightarrow \]

\[ = \]

\[ \text{Add } \text{Q}[ \text{person}_1/x, \text{Boston}/y ] \text{ to } \mathcal{F} \]
1. Find Satisfying Assignment
   – Use **EFCC Solver** to find Small Candidate Models
2. Construct Candidate Models
3. Model-Based Quantifier Instantiation
   – Two methods: **Generalizing Evaluations, Constructing Interpretations**
Properties: Finite Model Finding

For inputs \((F, Q)\), quantifiers in \(Q\) over free sorts

- Fixed-cardinality DPLL(T) + quantifier instantiation:
  - Sound
  - Finite Model Complete
    - If \((F, Q)\) has a finite model, we will eventually answer “SAT”
  - Refutationally Complete (when containing no theory symbols)
    - If \((F, Q)\) is unsatisfiable, we will eventually answer “UNSAT”

* - under certain restrictions
Finite Model Finding: Properties

• For unsatisfiable \((F, Q)\), quant. of \(Q\) over free sorts
  – When \((F, Q)\) contain theory symbols
    • Approach has weaker completeness property:
      – If there exists a set \(I\) of instances of \(Q\) where:
        » \(I\) is finite
        » \(F \land I\) is UNSAT
      – Then,
        » Fixed-cardinality DPLL(T)+QI terminates, answering UNSAT

• Thus, approach is only non-terminating when:
  – \((F, Q)\) is SAT, but only has infinite models
  – \((F, Q)\) is UNSAT, but all finite subsets are SAT
Enhancements

• **Heuristic Instantiation**
  – First see if instantiations based on heuristics exist
    • If not, resort to model-based instantiation
  – May lead to:
    • Discovering easy conflicts, if they exist
    • Arriving at model faster
      – Instantiations rule out spurious models

• **Sort Inference**
  – Reduce symmetries in problem

• **Relevancy**
  – Reduce the size of satisfying assignments
Experiments

• Implemented state of the art SMT solver CVC4
• Experiments on:
  – DVF Benchmarks
    • Taken from verification tool DVF used by Intel
    • Both SAT/UNSAT benchmarks
      – SAT benchmarks generated by removing necessary pf assumptions
    • Many theories: UF, arithmetic, arrays, datatypes
    • Quantifiers only over free sorts
      – Memory addresses, Values, Sets, ...
  – TPTP Benchmarks
    • Automated theorem proving community
    • No theory reasoning
  – Isabelle Benchmarks
    • Provable and unprovable goals, contains some arithmetic
Results: DVF

<table>
<thead>
<tr>
<th>SAT</th>
<th>german</th>
<th>refcount</th>
<th>agree</th>
<th>apg</th>
<th>bmk</th>
<th>Total</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>45</td>
<td>6</td>
<td>42</td>
<td>19</td>
<td>37</td>
<td>149</td>
<td></td>
</tr>
<tr>
<td>z3</td>
<td>45</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>46</td>
<td>8.1</td>
</tr>
<tr>
<td>cvc4+i</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0.0</td>
</tr>
<tr>
<td>cvc4+f</td>
<td>45</td>
<td>6</td>
<td>42</td>
<td>18</td>
<td>36</td>
<td>147</td>
<td>1413.1</td>
</tr>
<tr>
<td>cvc4+fi</td>
<td>45</td>
<td>6</td>
<td>42</td>
<td>19</td>
<td>36</td>
<td>148</td>
<td>1333.9</td>
</tr>
<tr>
<td>cvc4+fm</td>
<td>45</td>
<td>6</td>
<td>42</td>
<td>19</td>
<td>37</td>
<td>149</td>
<td>605.4</td>
</tr>
<tr>
<td>cvc4+fmi</td>
<td>45</td>
<td>6</td>
<td>42</td>
<td>19</td>
<td>37</td>
<td>149</td>
<td>409.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UNSAT</th>
<th>german</th>
<th>refcount</th>
<th>agree</th>
<th>apg</th>
<th>bmk</th>
<th>Total</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>145</td>
<td>40</td>
<td>488</td>
<td>304</td>
<td>244</td>
<td>1221</td>
<td></td>
</tr>
<tr>
<td>z3</td>
<td>145</td>
<td>40</td>
<td>488</td>
<td>304</td>
<td>244</td>
<td>1221</td>
<td>31.0</td>
</tr>
<tr>
<td>cvc4+i</td>
<td>145</td>
<td>40</td>
<td>484</td>
<td>304</td>
<td>244</td>
<td>1217</td>
<td>21.3</td>
</tr>
<tr>
<td>cvc4+f</td>
<td>145</td>
<td>40</td>
<td>476</td>
<td>298</td>
<td>242</td>
<td>1201</td>
<td>7512.2</td>
</tr>
<tr>
<td>cvc4+fi</td>
<td>145</td>
<td>40</td>
<td>488</td>
<td>302</td>
<td>244</td>
<td>1219</td>
<td>1181.4</td>
</tr>
<tr>
<td>cvc4+fm</td>
<td>145</td>
<td>40</td>
<td>471</td>
<td>300</td>
<td>242</td>
<td>1198</td>
<td>6949.7</td>
</tr>
<tr>
<td>cvc4+fmi</td>
<td>145</td>
<td>40</td>
<td>488</td>
<td>302</td>
<td>244</td>
<td>1219</td>
<td>1185.0</td>
</tr>
</tbody>
</table>

- CVC4 with finite model finding (cvc4+f)
- Effective for answering SAT
- Using heuristic instantiation, solves 4 UNSAT that cvc4 cannot
## Results: TPTP

<table>
<thead>
<tr>
<th></th>
<th>SAT</th>
<th>UNSAT</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EPR</td>
<td>NEQ</td>
<td>SEQ</td>
<td>PEQ</td>
<td>Total</td>
<td>EPR</td>
</tr>
<tr>
<td><strong>NEQ</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>NEQ</strong></td>
</tr>
<tr>
<td><strong>SEQ</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>SEQ</strong></td>
</tr>
<tr>
<td><strong>PEQ</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>PEQ</strong></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>Total</strong></td>
</tr>
<tr>
<td><strong>EPR</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>EPR</strong></td>
</tr>
<tr>
<td><strong>NEQ</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>NEQ</strong></td>
</tr>
<tr>
<td><strong>SEQ</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>SEQ</strong></td>
</tr>
<tr>
<td><strong>PEQ</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>PEQ</strong></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>Total</strong></td>
</tr>
</tbody>
</table>

|      |    |    |    |    |    |    |    |    |    |    |
|      |    |    |    |    |    |    |    |    |    |    |

| z3   | 320 | 155 | 164 | 249 | 888 | **989** | **412** | **3310** | 1320 | **6031** |
| cvc3 | 27  | 0   | 0   | 0   | 27  | 787     | 381     | 3019     | 883  | 5070     |
| iprover | **363** | 128 | 107 | 396 | 994 | 835     | 105     | 2690     | **1523** | 5153     |
| iprover+f | 362 | 226 | 178 | 468 | 1234 | 213     | 1       | 121      | 48   | 383      |
| paradox | 340 | **304** | 185 | **526** | **1355** | 723     | 17      | 339      | 186  | 1265     |
| cvc4+i | 32  | 0   | 0   | 0   | 32  | 821     | 383     | 3152     | 1045 | 5401     |
| cvc4+f | 295 | 178 | 143 | 375 | 991  | 759     | 247     | 887      | 651  | 2544     |
| cvc4+fm | 298 | 221 | 178 | 391 | 1088 | 759     | 169     | 1010     | 703  | 2641     |
| cvc4+fM | 301 | 235 | **200** | 395 | 1131 | 759     | 198     | 1073     | 733  | 2763     |
| cvc4+fMi | 292 | 207 | 153 | 385 | 1037 | 762     | 236     | 1281     | 746  | 3025     |

- **CVC4 Placed 3\textsuperscript{rd} in FNT (non-theorem) division of CASC 24**

**cvc4 :**
- f : finite model
- i : heuristic
- m : model-based
- M : model-based (version 2)

10 second timeout
Results: TPTP

- Model-Based Instantiation is often essential
  - Solves when naïve approach requires ~775 billion instances

---

cvc4:
- f: finite model
- m: model-based
- M: model-based (version 2)
## Results: Isabelle

### SAT

<table>
<thead>
<tr>
<th>SAT</th>
<th>ArrowOrder</th>
<th>FFT</th>
<th>FTA</th>
<th>Hoare</th>
<th>NS_Shared</th>
<th>QEpres</th>
<th>StrongNorm</th>
<th>TwoSquares</th>
<th>TypeSafe</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>cvc3</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>17</td>
</tr>
<tr>
<td>z3</td>
<td>1</td>
<td>19</td>
<td>24</td>
<td>46</td>
<td>10</td>
<td>47</td>
<td>1</td>
<td>17</td>
<td>12</td>
<td>177</td>
</tr>
<tr>
<td>cvc4+i</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>17</td>
</tr>
<tr>
<td>cvc4+f</td>
<td>26</td>
<td>123</td>
<td>149</td>
<td>56</td>
<td>75</td>
<td>12</td>
<td>50</td>
<td>84</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cvc4+fi</td>
<td>26</td>
<td>133</td>
<td>158</td>
<td>56</td>
<td>75</td>
<td>12</td>
<td>44</td>
<td>87</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cvc4+fm</td>
<td>22</td>
<td>120</td>
<td>147</td>
<td>36</td>
<td>75</td>
<td>12</td>
<td>36</td>
<td>87</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cvc4+fM</td>
<td>28</td>
<td>126</td>
<td>163</td>
<td>44</td>
<td>94</td>
<td>12</td>
<td>43</td>
<td>87</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cvc4+fMi</td>
<td>31</td>
<td>136</td>
<td>161</td>
<td>61</td>
<td>101</td>
<td>12</td>
<td>44</td>
<td>85</td>
<td></td>
<td>785</td>
</tr>
</tbody>
</table>

### UNSAT

<table>
<thead>
<tr>
<th>UNSAT</th>
<th>ArrowOrder</th>
<th>FFT</th>
<th>FTA</th>
<th>Hoare</th>
<th>NS_Shared</th>
<th>QEpres</th>
<th>StrongNorm</th>
<th>TwoSquares</th>
<th>TypeSafe</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>cvc3</td>
<td>287</td>
<td>250</td>
<td>877</td>
<td>577</td>
<td>102</td>
<td>291</td>
<td>206</td>
<td>552</td>
<td>216</td>
<td>3358</td>
</tr>
<tr>
<td>z3</td>
<td>254</td>
<td>230</td>
<td>797</td>
<td>507</td>
<td>135</td>
<td>242</td>
<td>240</td>
<td>491</td>
<td>329</td>
<td>3225</td>
</tr>
<tr>
<td>cvc4+i</td>
<td>253</td>
<td>233</td>
<td>749</td>
<td>476</td>
<td>99</td>
<td>265</td>
<td>234</td>
<td>523</td>
<td>267</td>
<td>3099</td>
</tr>
<tr>
<td>cvc4+f</td>
<td>123</td>
<td>94</td>
<td>350</td>
<td>209</td>
<td>41</td>
<td>99</td>
<td>83</td>
<td>361</td>
<td>127</td>
<td>1487</td>
</tr>
<tr>
<td>cvc4+fi</td>
<td>155</td>
<td>164</td>
<td>509</td>
<td>374</td>
<td>37</td>
<td>168</td>
<td>100</td>
<td>452</td>
<td>195</td>
<td>2154</td>
</tr>
<tr>
<td>cvc4+fm</td>
<td>112</td>
<td>86</td>
<td>357</td>
<td>212</td>
<td>26</td>
<td>119</td>
<td>82</td>
<td>349</td>
<td>120</td>
<td>1463</td>
</tr>
<tr>
<td>cvc4+fM</td>
<td>88</td>
<td>92</td>
<td>381</td>
<td>202</td>
<td>29</td>
<td>109</td>
<td>93</td>
<td>365</td>
<td>149</td>
<td>1508</td>
</tr>
<tr>
<td>cvc4+fMi</td>
<td>154</td>
<td>164</td>
<td>515</td>
<td>371</td>
<td>37</td>
<td>167</td>
<td>100</td>
<td>452</td>
<td>195</td>
<td>2155</td>
</tr>
</tbody>
</table>

- For UNSAT, cvc4 with finite model finding is **orthogonal**:  
  - Solves 170 unsat that cvc3 cannot, 365 z3 cannot, 229 that cvc4+i cannot
Extension to (Bounded) Integers

• A formula of the form

\[ \forall x_1 \ldots x_n : \text{Int. } L_1 \leq x_1 \leq U_1 \land \ldots \land L_n \leq x_n \leq U_n \Rightarrow \Psi \]

• Where \( x_i \not\in \text{FV}(L_j, U_j) \), for \( i < j \)

This has *Bounded Integer Quantification*

• Example: \( \forall xy. 0 \leq x \leq 20 \land 0 \leq y \leq f(x) \Rightarrow P(x, y) \)

• Can be handled similar as before
  – Minimize bounds, (naïvely) instantiate exhaustively
Bounded Integer Quantification

- Idea: Fix values of bound $c$

\[ Q : \forall x : \text{Int. } 0 \leq x \leq c \implies P(x) \]

- Approach is sound, and model complete
  - When input has model, it eventually terminates with “SAT”
Results

- Set of verification benchmarks from Intel
  - Arrays, datatypes, integer arithmetic
  - Symbolic bounds for integer quantification, e.g.
    \[ \forall x : \text{Int. } 0 \leq x \leq c \Rightarrow P(x), \text{ where } c \text{ is free constant} \]
- CVC4 (with fmf) finds small models \( \mathbb{M} \), i.e.
  - Value of \( \mathbb{M}[c] \) is 2 to 3, at most 10

<table>
<thead>
<tr>
<th></th>
<th>SAT (263) solved</th>
<th>UNSAT (843) solved</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>solved</td>
<td>time</td>
</tr>
<tr>
<td>z3</td>
<td>257</td>
<td>957.9</td>
</tr>
<tr>
<td>cvc4+i</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>cvc4+fi</td>
<td>263</td>
<td>90.8</td>
</tr>
</tbody>
</table>

CVC4:
- f: bounded integer techniques
- i: heuristic

600 second timeout
Summary

• CVC4 with finite model finding:
  – Incorporates various instantiation strategies:
    • Model-based quantifier instantiation
    • Heuristic instantiation (E-matching)
  – Has important properties:
    • Finite-Model Completeness
    • Refutational Completeness (under certain conditions)
  – Approach can be extended to integers, theory of strings
  – Improves the state-of-the-art, over:
    • SMT solvers
      – Increased ability to answer “satisfiable”
    • Automated Theorem Provers
      – Efficient reasoning about background theories at QF level
Thank you

• Acknowledgements:
  – Collaborators: Cesare Tinelli, Amit Goel, Sava Krstíc, Clark Barrett, Morgan Deters, Leonardo de Moura
  – Dissertation Committee: Aaron Stump, Hantao Zhang, Sriram Pemmaraju

• Questions?