Implementing Branch and Bound Algorithms in SMT

Andrew Reynolds
Two Sigma
July 12, 2016

THE UNIVERSITY OF IOWA
Overview

• Satisfiability Modulo Theories and DPLL(T)
• Finite Model Finding in SMT
  • Branch and bound for finding small models
  • Variants of the approach
  • Relationship to Optimization
• Recent trends, future work
Satisfiability Modulo Theories (SMT)

\((\forall x. P(x) \lor f(b) = b+1) \land \exists y. (\neg P(y) \land f(y) < y)\)

- We are often interested in establishing \textit{T-satisfiability} of formulas with:
Satisfiability Modulo Theories (SMT)

$(\forall x. P(x) \lor f(b) = b+1) \land \exists y. (\neg P(y) \land f(y) < y)$

• We are often interested in establishing $T$-satisfiability of formulas with:
  • Boolean structure
Satisfiability Modulo Theories (SMT)

$(\forall x. P(x) \lor f(b) = b+1) \land \exists y. (\neg P(y) \land f(y) < y)$

• We are often interested in establishing $T$-satisfiability of formulas with:
  • Boolean structure
  • Constraints in a background theory $T$, e.g. UFLIA
Satisfiability Modulo Theories (SMT)

\((\forall x. P(x) \lor f(b) = b+1) \land \exists y. (\neg P(y) \land f(y) < y)\)

- We are often interested in establishing \(T\)-satisfiability of formulas with:
  - **Boolean structure**
  - Constraints in a background theory \(T\), e.g. UFLIA
  - ...even existential and universal quantifiers
DPLL(T): Basics

\((P(a) \lor f(b) > a+1)\)
\((\neg P(b) \lor \forall x. P(x))\)
\((f(b) = a-5 \lor \neg P(a))\)
DPLL(T): Basics

- Consider the propositional abstraction of the formula

\[
(P(a) \lor f(b) > a + 1) \\
(\neg P(b) \lor \forall x. P(x)) \\
(f(b) = a - 5 \lor \neg P(a))
\]

- Consider the propositional abstraction of the formula
DPLL(T): Basics

• Consider the propositional abstraction of the formula
DPLL(T): Basics

- Find propositional satisfying assignment via off-the-shelf SAT solver
DPLL(T): Basics

(P(a) \lor f(b) > a + 1) \\
(\neg P(b) \lor \forall x. P(x)) \\
(f(b) = a - 5 \lor \neg P(a)) \\

\textbf{SAT Solver}

- A \rightarrow true
- B \rightarrow true
- C \rightarrow false
- D \rightarrow true
- E \rightarrow true

\textbullet \ Find propositional satisfying assignment via off-the-shelf SAT solver
DPLL(T): Basics

- Consider the original atoms

SAT Solver

\((P(a) \lor f(b) > a + 1)\)

\((-P(b) \lor \forall x . P(x))\)

\((f(b) = a - 5 \lor \neg P(a))\)

- Consider the original atoms
DPLL(T): Basics

(SAT Solver)

\[ P(a), f(b) > a + 1, \neg P(b), \forall x. P(x), f(b) = a - 5 \]

Propositional assignment can be seen as a set of T-literals \( M \)
- Must check if \( M \) is T-satisfiable
DPLL(T): Basics

\[
(P(a) \lor f(b) > a + 1) \\
(\neg P(b) \lor \forall x. P(x)) \\
(f(b) = a - 5 \lor \neg P(a))
\]

⇒ Distribute ground literals to T-solvers, \(\forall\) literals to quantifiers module
DPLL(T): Basics

\( (P(a) \lor f(b) > a + 1) \)
\( (\neg P(b) \lor \forall x. P(x)) \cup (\neg f(b) > a + 1 \lor \neg f(b) = a - 5) \)
\( (f(b) = a - 5 \lor \neg P(a)) \)

⇒ These solvers may choose to add conflicts/lemmas to clause set
DPLL($T_1+..+T_n$): Overview

SAT Solver

T-Clauses $F$

Satisfying Assignment $M$

$T_n$-solver

$T_1$-solver

Conflicts, lemmas

Nelson-Oppen combination

Quantifiers Module

$M_1$

$M_n$

...when $F$ is propositionally unsatisfiable

[$\Rightarrow$ Each of these components may:
  - Report $M$ is $T$-unsatisfiable by reporting conflict clauses
  - Report lemmas if they are unsure]

[Nieuwenhuis/Oliveras/Tinelli 06]
DPLL($T_1 + \ldots + T_n$): Overview

SAT Solver

T-Clauses $F$

Satisfying Assignment $M$

$M_1$ $\rightarrow$ $T_1$-solver

$\vdots$

$M_n$ $\rightarrow$ $T_n$-solver

Q $\rightarrow$ Quantifiers Module

sat

[unsat]

...when $F$ is propositionally unsatisfiable

$\Rightarrow$ If no component adds a lemma, then it must be the case that $M$ is $T_1 + \ldots + T_n$-satisfiable

[Nieuwenhuis/Oliveras/Tinelli 06]
Common Theories Supported by SMT Solvers

- SMT solvers support:
  - Arbitrary Boolean combinations of ground theory constraints
  - Examples of supported theories:
    - Uninterpreted functions: \( f(a) = g(b, c) \)
    - Linear real/integer arithmetic: \( a \geq b + 2 \cdot c + 3 \)
    - Arrays: \( \text{select}(A, i) = \text{select}(\text{store}(A, i + 1, 3), i) \)
    - BitVectors: \( \text{bvule}(x, \#\text{xFF}) \)
    - Algebraic Datatypes: \( x, y : \text{List}; \text{tail}(x) = \text{cons}(0, y) \)
    - ...
  - \( \forall \) over each of these
Common Theories Supported by SMT Solvers

• SMT solvers support:
  • Arbitrary Boolean combinations of ground theory constraints
  • Examples of supported theories:
    • Uninterpreted functions: ⇒ Congruence Closure [Nieuwenhuis/Oliveras 2005]
    • Linear real/integer arithmetic: ⇒ Simplex [deMoura/Dutertre 2006]
    • Arrays: ⇒ [deMoura/Bjorner 2009]
    • BitVectors: ⇒ Bitblasting, lazy approaches [Bruttomesso et al 2007, Hadarean et al 2014]
    • Algebraic Datatypes: ⇒ [Barrett et al 2007]
    • …
  • ∀ over each of these
SMT Solvers have Partial Support for $\forall$

$\forall x: \text{Int.} \ P(x)$

$P$ is true for all integers $x$

• Satisfiability problem for $\forall$ is generally **undecidable**

• Heuristic Techniques for “unsat”:
  • E-matching [Detlefs et al 2003, Ge et al 2007, de Moura/Bjorner 2007]

• Limited Techniques have completeness guarantees:
  • Local theory extensions [Sofronie-Stokkermans 2005]
  • Array fragments [Bradley et al 2006, Alberti et al 2014]
  • Complete Instantiation [Ge/de Moura 2009]
  • Finite Model Finding [Reynolds et al 2013]
SMT Solvers have Partial Support for $\forall$

$\forall x: \text{Int.} \ P(x)$

$P$ is true for all integers $x$

- Satisfiability problem for $\forall$ is generally **undecidable**
- Heuristic Techniques for “unsat”:
  - E-matching [Detlefs et al 2003, Ge et al 2007, de Moura/Bjorner 2007]
- Limited Techniques have completeness guarantees:
  - Local theory extensions [Sofronie-Stokkermans 2005]
  - Complete Instantiation [Ge/de Moura 2009]
  - **Finite Model Finding** [Reynolds et al 2013]

$\Rightarrow$ Focus of next slides
Finite Model Finding: Motivation

List := cons(head: Int, tail: List) | nil

\( \forall x: \text{List}. \text{length}(x) = \text{ite}(\text{is-cons}(x), 1 + \text{length}(\text{tail}(x)), 0) \)
\( \forall xy: \text{List}. \text{append}(x) = \text{ite}(\text{is-cons}(x), \text{cons}(\text{head}(x), \text{append}(\text{tail}(x), y)), y) \)
\( \forall x: \text{List}. \text{rev}(x) = \text{ite}(\text{is-cons}(x), \text{append}(\text{rev}(\text{tail}(x)), \text{cons}(\text{head}(x), \text{nil})), \text{nil}) \)

\( \exists xy: \text{List}. \text{rev}(\text{append}(x, y)) \neq \text{append}(\text{rev}(y), \text{rev}(x)) \)

CVC4

Conjecture holds
Finite Model Finding: Motivation

List := cons( head : Int, tail : List ) | nil

\[ \forall x \in L. \text{length}(x) = \text{ite}(\text{is-cons}(x), 1 + \text{length}(\text{tail}(x)), 0) \]

\[ \forall xy \in L. \text{append}(x) = \text{ite}(\text{is-cons}(x), \text{cons}(\text{head}(x), \text{append}(\text{tail}(x), y)), y) \]

\[ \forall x \in L. \text{rev}(x) = \text{ite}(\text{is-cons}(x), \text{append}(\text{rev}(\text{tail}(x)), \text{cons}(\text{head}(x), \text{nil})), \text{nil}) \]

...$

\exists xy \in \text{List}. \text{rev}(\text{append}(x, y)) \neq \text{append}(\text{rev}(y), \text{rev}(x))$

CVC4

Conjecture holds

...but what if the conjecture does not hold?
Finite Model Finding: Motivation

Axioms $A$ (with $\forall$), negated conjecture $P$

CVC4

UNSAT

Unknown

Conjecture $P$ holds

Manual Inspection

Candidate $CEX$
Finite Model Finding: Motivation

- Conjecture P holds
- Axioms A (with ∀), negated conjecture P
- CVC4
  - UNSAT
  - SAT
- Candidate CEX
  - Manual Inspection
- Concrete CEX for conjecture P
Finite Model Finding in DPLL(T)

T-Clauses $F$

SAT Solver

unsat

$\forall xy: U.P(x,y)$

$M$}

$M_1$ \rightarrow $T_1$-solver

$\vdots$

$M_n$ \rightarrow $T_n$-solver

Quantifiers Module
Finite Model Finding in DPLL(T)

- Given universally quantified formula $\forall xy: U\cdot P(x, y)$

Diagram:
- SAT Solver
  - T-Clauses $F$
  - $\text{unsat}$
  - $M$
  - $M_1$, $T_1$-solver
  - $\ldots$
  - $M_n$, $T_n$-solver
  - Quantifiers Module
Given universally quantified formula $\forall xy: U. P(x, y)$

- If $U$ can be interpreted as finite, e.g. $\{a, b, c, d, e\}$:
Finite Model Finding in DPLL(T)

Given universally quantified formula $\forall x y : U \cdot P(x, y)$

- If U can be interpreted as finite, e.g. \{a, b, c, d, e\}:
  - Can be reduced to a finite set of instances
Finite Model Finding in DPLL(T)

- Given universally quantified formula $\forall xy: U \cdot P(x, y)$
  - If $U$ can be interpreted as finite, e.g. $\{a, b, c, d, e\}$:
    - Can be reduced to a finite set of instances

- Can be very large

\[ P(a, a) \land \ldots P(e, a) \land \
P(a, b) \land \ldots \
P(a, c) \land \ldots \
P(a, d) \land \ldots \
P(a, e) \land \ldots P(e, e) \]
Finite Model Finding in SMT

• Address large # instantiations by:
  1. Only add instantiations that refine model [Reynolds et al CADE13]
     • Model-based quantifier instantiation [Ge/deMoura CAV 2009]
  2. **Minimizing** model sizes [Reynolds et al CAV13]
     • Find interpretation that minimizes the #elements in $\mathbb{U}$
1. Model-Based Quantifier Instantiation
Model-Based Quantifier Instantiation

• Basic idea:
Model-Based Quantifier Instantiation

• Basic idea:
  1. Build candidate interpretation $M$, compute $P^M(x, y)$
Model-Based Quantifier Instantiation

• Basic idea:
  1. Build candidate interpretation $M$, compute $P^M(x, y)$
  2. Add instances (if any) that evaluate to false
Model-Based Quantifier Instantiation

- Basic idea:
  - ...and repeat
Model-Based Quantifier Instantiation

• Basic idea:
  • ...and repeat
Model-Based Quantifier Instantiation

- Basic idea:
  - ...and repeat
Model-Based Quantifier Instantiation

- Basic idea:
  - ...and repeat
• 1203 satisfiable benchmarks from the TPTP library
  • Graph shows # instances required by exhaustive instantiation
  • E.g. $\forall xyz: U. P(x, y, z)$, if $|U|=4$, requires $4^3=64$ instances
Model-based Instantiation: Impact

- CVC4 Finite Model Finding + Exhaustive instantiation
  - Scales only up to ~150k instances with a 30 sec timeout
Model-based Instantiation: Impact

- CVC4 Finite Model Finding + Model-Based instantiation [Reynolds et al CADE13]
  - Scales to >2 billion instances with a 30 sec timeout, only adds fraction of possible instances
2. Minimizing Model Sizes
Minimizing Model Sizes

• Finding small models is important
  (leads to exponentially fewer possible instances of $\forall$)

To establish T-satisfiability of:

$$G \land \forall x : U. P(x)$$

...where $G$ is a set of ground constraints, and $U$ is an uninterpreted sort

First, find a model $M$ of $G$ such that $|U^M|$ is minimized

• To minimize $|U^M|$:
  • Modifications to the DPLL search procedure in the SAT solver
  • Additional theory solver for cardinality constraints
Minimizing Model Sizes

• Abstractly, organize DPLL search by fixing the cardinality of $U$

1. Search for models where $|U|=1$

2. If none exist, search for models where $|U|=2$

3. etc.
Minimizing Model Sizes

• Abstractly, organize DPLL search by fixing the cardinality of $U$.

1. Search for models where $|U|=1$

2. If none exist, search for models where $|U|=2$

3. etc.

$\Rightarrow$ Extend the SMT solver with a theory solver for cardinality constraints.
Theory of finite cardinality constraints

- Theory solver for $T_{FCC}$
  - $FCC = $ finite cardinality constraints
Theory of finite cardinality constraints

- Theory of **finite cardinality constraints** $T_{FCC}$
  - Signature $\Sigma_{FCC}$:
    - Predicates $|U| \leq k$ for each uninterpreted sort $U$ and positive numeral $k$

Examples:
- $a \neq b \land |U| \leq 1$ … $T_{FCC}$-unsatisfiable
- $a \neq b \land a \neq c \land |U| \leq 2$ … $T_{FCC}$-satisfiable (where $b_M = c_M$)
Theory of finite cardinality constraints

- Theory of finite cardinality constraints $T_{FCC}$
  - Signature $\Sigma_{FCC}$:
    - Predicates $|U| \leq k$ for each uninterpreted sort $U$ and positive numeral $k$

- Examples:
  - $a, b, c : U$
  - $a \neq b \land |U| \leq 1 \ldots T_{FCC}$-unsatisfiable
  - $a \neq b \land a \neq c \land |U| \leq 2 \ldots T_{FCC}$-satisfiable (where $b^M = c^M$)
Theory of finite cardinality constraints

• Decision procedure for $T_{FCC}$:
  • Given input $G$
    ...where $G$ is a set of equalities and disequalities
  • Consider the disequality graph $(V,E)$ induced by $G$:
    • Vertices $V$ are equivalence classes
    • Edges $E$ are disequalities
Theory of finite cardinality constraints

• Decision procedure for $T_{\text{FCC}}$:
  • Given input $G$
    ...where $G$ is a set of equalities and disequalities
  • Consider the disequality graph $(V,E)$ induced by $G$:
    • Vertices $V$ are equivalence classes
    • Edges $E$ are disequalities

\[ a \neq b, b \neq c, c \neq d, d \neq e, e \neq a \]
\[ |U| \leq 3 \]
Theory of finite cardinality constraints

\[ a \neq b, b \neq c, c \neq d, d \neq e, e \neq a \]
\[ |U| \leq 3 \]
Theory of finite cardinality constraints

\[ a \neq b, \ b \neq c, \ c \neq d, \ d \neq e, \ e \neq a \]
\[ |U| \leq 3 \]

- Decision procedure for \( T_{\text{FCC}} \):

Let \( k \) be the smallest \( k \) such that \( |U| \leq k \)
- If there is a \((k+1)\)-clique, answer “unsat”
- If there are \( k \) or fewer vertices, answer “sat”
- Otherwise, split the problem: \( t_1 = t_2 \lor t_1 \neq t_2 \) for some vertices \( t_1, t_2 \)
Theory of finite cardinality constraints

\[ a \neq b, \ b \neq c, \ c \neq d, \ d \neq e, \ e \neq a \]
\[ |U| \leq 3 \]

Split: \( a = d \) \lor \( a \neq d \)
Theory of finite cardinality constraints

\[ a \neq b, \ b \neq c, \ c \neq d, \ d \neq e, \ e \neq a \]
\[ a = d \]
\[ |U| \leq 3 \]

\[ \text{Split: } a = d \lor a \neq d \]
Theory of finite cardinality constraints

\[ a \neq b, \quad b \neq c, \quad c \neq d, \quad d \neq e, \quad e \neq a \]
\[ a = d \]
\[ |U| \leq 3 \]

Split: \( a = d \lor a \neq d \)

Split: \( e = c \lor e \neq c \)
Theory of finite cardinality constraints

$a \neq b, b \neq c, c \neq d, d \neq e, e \neq a$

$a = d, e = c$

$|U| \leq 3$

Split: $a = d \lor a \neq d$

Split: $e = c \lor e \neq c$

Diagram: $a, d \neq c, e$
Theory of finite cardinality constraints

\[ a \neq b, \ b \neq c, \ c \neq d, \ d \neq e, \ e \neq a \]
\[ a = d, \ e = c \]
\[ |U| \leq 3 \]

Split: \( a = d \) \lor a \neq d

Split: \( e = c \) \lor e \neq c

3 equivalence classes ... answer “sat”
Theory of finite cardinality constraints

• Decision procedure for $T_{FCC}$
  • Sound, complete and terminating for $T_{FCC}$-satisfiability
  • Fully integrated into DPLL(T) framework
    • Incremental, generates conflict clauses

• Incorporates optimizations: [Reynolds et al CAV13]
  • Finds k-cliques (an NP-hard problem) via a fast incomplete check
  • Heuristics for which vertices to split
Minimizing Model Sizes with $T_{FCC}$

- Theory solver for $T_{FCC}$ can be used in part for finding minimal models.

Search for models where $|U|=1$.

If none exist, search for models where $|U|=2$.

etc.
Minimizing Model Sizes with $T_{\text{FCC}}$

- Theory solver for $T_{\text{FCC}}$ can be used in part for finding minimal models
  - Introduce incremental bounds on cardinality in DPLL search

\[
|U| \leq 1 \quad \neg|U| \leq 1 \\
|U| \leq 2 \quad \neg|U| \leq 2 \\
|U| \leq 3 \quad \neg|U| \leq 3
\]

Search for models where $|U|=1$

If none exist, search for models where $|U|=2$

\[\Rightarrow \text{DPLL solver chooses tightest bound as first decision}\]
Finding Minimal Counterexamples for ITP

List := cons( head : Int, list : Tail ) | nil
L: “subterm-closed structure” of List

∀x:L.length(x)=ite(is-cons(x),1+length(tail(x)),0)
∀xy:L.append(x,y)=ite(is-cons(x),cons(head(x),append(tail(x),y)),y)
∀x:L.rev(x)=ite(is-cons(x),append(rev(tail(x)),cons(head(x),nil),nil),nil)
...
∃xy:L.rev(append(x,y))≠append(rev(y),rev(x))

CVC4
Finding Minimal Counterexamples for ITP

```
List := cons( head : Int, list : Tail ) | nil
L: “subterm-closed structure” of List

∀x:L.length(x)=ite(is-cons(x),1+length(tail(x)),0)
∀xy:L.append(x,y)=ite(is-cons(x),cons(head(x),append(tail(x),y)),y)
∀x:L.rev(x)=ite(is-cons(x),append(rev(tail(x)),cons(head(x),nil),nil)

∃xy:L.rev(append(x,y))≠append(rev(y),rev(x))
```

CVC4

 unfolds

UNSAT

∀xy:L.rev(append(x,y))=append(rev(y),rev(x)) holds
Finding Minimal Counterexamples for ITP

\[
\text{List} := \text{cons( head : Int, list : Tail ) | nil}
\]

L: “subterm-closed structure” of List

\[
\forall x : \text{L}. \text{length}(x) = \text{ite}(\text{is-cons}(x), 1 + \text{length}(\text{tail}(x)), 0)
\]

\[
\forall xy : \text{L}. \text{append}(x, y) = \text{ite}(\text{is-cons}(x), \text{cons}(\text{head}(x), \text{append}(\text{tail}(x), y)), y)
\]

\[
\forall x : \text{L}. \text{rev}(x) = \text{ite}(\text{is-cons}(x), \text{append}(\text{rev}(\text{tail}(x)), \text{cons}(\text{head}(x), \text{nil}), \text{nil})
\]

\[
\exists xy : \text{L}. \text{rev}(\text{append}(x, y)) \neq \text{append}(\text{rev}(x), \text{rev}(y))
\]

CVC4
Finding Minimal Counterexamples for ITP

List := cons( head : Int, list : Tail ) | nil
L: “subterm-closed structure” of List

∀x:L.length(x)=ite(is-cons(x),1+length(tail(x)),0)
∀xy:L.append(x,y)=ite(is-cons(x),cons(head(x),append(tail(x),y)),y)
∀x:L.rev(x)=ite(is-cons(x),append(rev(tail(x)),cons(head(x),nil)),nil)
...

∃xy:L.rev(append(x,y))≠append(rev(x),rev(y))

CVC4

Counterexample M:
M (x)=cons(0,nil)
M (y)=cons(1,nil)
Finding Minimal Counterexamples: Challenge

Tree := node( left : Tree, data : Int, right : Tree ) | leaf
T: “subterm-closed structure” of Tree

∀x:T. depth(x) = \text{ite}(\text{is-node}(x), 1 + \text{max}(\text{depth}(\text{left}(x)), \text{depth}(\text{right}(x))), 0)

∃k:T. depth(k) ≥ 4

- Find a tree with depth at least 4
Finding Minimal Counterexamples: Challenge

Tree := node( left : Tree, data : Int, right : Tree ) | leaf
T: “subterm-closed structure” of Tree

\[ \forall x:T. \text{depth}(x) = \text{ite}(\text{is-node}(x), 1 + \text{max}(\text{depth}(\text{left}(x)), \text{depth}(\text{right}(x))), 0) \]

\[ \exists k:T. \text{depth}(k) \geq 4 \]

- Find a tree with depth at least 4

  Consider all \( k \) of depth 0
  Consider all \( k \) of depth 1
  Consider all \( k \) of depth 2

  Combinatorial explosion \( \Rightarrow \) solver is slow!

  Consider all \( k \) of depth 3
  ...
Finding Minimal Counterexamples: Challenge

Tree := node( left : Tree, data : Int, right : Tree ) | leaf
T: “subterm-closed structure” of Tree

∀x:T. depth(x) = \text{ite}(\text{is-node}(x), 1 = \text{max}(\text{depth}(\text{left}(x)), \text{depth}(\text{right}(x))), 0)

∃k:T. depth(k) ≥ 4

- Find a tree with depth at least 4
Finding (Non-Minimal) CEX: Challenge

List := cons( head : Int, list : Tail ) | nil
L: “subterm-closed structure” of List

∀x:L.all-pos(x)=ite(is-cons(x),head(x)>0∧all-pos(tail(x)),true)

∃k:L.is-cons(k) ∧ all-pos(k)

• Find a non-empty list of positive integers
Finding (Non-Minimal) CEX: Challenge

List := cons( head : Int, list : Tail ) | nil
L: “subterm-closed structure” of List

∀x:L.all-pos(x)=ite(is-cons(x),head(x)>0∧all-pos(tail(x)),true)

∃k:L.is-cons(k) ∧ all-pos(k)

• Find a non-empty list of positive integers

Search is unfair ⇒ solver is non-terminating!
Branch and Bound: Hybrid Approach?

• Guide search so that eventually it will consider small models
  ⇒ In development
Branch and Bound: Use Cases

• Similar approach can be used for:
  1. ∀ bounded by symbolic numeric (integer) range
  2. ∀ bounded by set membership
  3. Model finding for theory of strings + length
  4. Syntax-Guided Synthesis
Use case #1: Bounded Integer ∀
Variant: Bounded Integer \( \forall \)

- \( \forall x: \text{Int}. \ 0 \leq x < t \Rightarrow P(x) \)

Search for models where \( t < 0 \).
If none exist, search for models where \( t = 0 \).

\( \Rightarrow \) Incrementally bound the value of term \( t \).
Use case #2: Sets + Cardinality
Theory of **Finite Sets + Cardinality**

- Parametric theory of finite sets of elements $E$

- **Signature $\Sigma_{Set}$:**
  - Empty set $\emptyset$, Singleton $\{a\}$
  - Membership $\in : E \times \text{Set} \rightarrow \text{Bool}$
  - Subset $\subseteq : \text{Set} \times \text{Set} \rightarrow \text{Bool}$
  - Set connectives $\cup, \cap, \setminus : \text{Set} \times \text{Set} \rightarrow \text{Set}$

- Example input: $x = y \cap z \land a + 5 \in x \land y \subseteq w$

- Applications in programming languages, e.g. Alloy
Theory of Finite Sets + Cardinality

• Recently:
  • Extended signature of theory to include:
    • **Cardinality** $|.| : \text{Set} \rightarrow \text{Int}$
  • Extended **decision procedure** for cardinality constraints
    • Fully integrated component in DPLL(T) [Bansal et al IJCAR2016]

• Example input: $x = y \cup z \land |x| = 14 \land |y| \geq |z| + 5$
Theory of Finite Sets + Cardinality

• Decision procedure builds **cardinality graph** where
  • Cardinality of leaves are disjoint sum of parents

\[
|y| = |y \setminus z| + |y \cap z| \\
|z| = |z \setminus y| + |y \cap z| \\
|y \cup z| = |y \setminus z| + |y \cap z| + |z \setminus y|
\]

[Bansal/Reynolds/Barrett/Tinelli IJCAR2016]
Theory of Finite Sets + Cardinality

• Decision procedure builds cardinality graph where
  • Cardinality of leaves are disjoint sum of parents
    • Equalities between sets

\[ y \cup z = y \setminus z \setminus y \setminus z \setminus z \]

[Bansal/Reynolds/Barrett/Tinelli IJCAR2016]
Theory of Finite Sets + Cardinality

• Decision procedure builds **cardinality graph** where
  - Cardinality of leaves are disjoint sum of parents
    - Equalities between sets $\rightarrow$ merge leaves

\[
\begin{align*}
\text{x} &= \text{y} \cup \text{z} \\
\text{x} \cap (\text{y} \setminus \text{z}) &= |\text{x} \cap (\text{y} \setminus \text{z})| + |\text{x} \cap \text{y} \cap \text{z}| + |\text{x} \cap (\text{z} \setminus \text{y})|
\end{align*}
\]

[Bansal/Reynolds/Barrett/Tinelli IJCAR2016]
Branch and Bound: Set Membership \( \forall \)

- \( \forall x:\text{Int}. x \in S \Rightarrow P(x) \)

Search for models where \(|S| = 0\)

If none exist, search for models where \(|S| = 1\)

etc.

\( \Rightarrow \) Make use of native set cardinality operator \(|. |: \text{Set} \rightarrow \text{Int} \)
Set Membership \( \forall \)

- Increased power to encode:

\[
\forall x. x \in S \Rightarrow P(x) \land |S| \geq k \quad \text{...} \quad P \text{ holds for at least } k \text{ points}
\]

\[
\forall x. x \in S \Rightarrow x < 10 \quad \text{...} \quad \text{All elements of } S \text{ are } < 10
\]

\[
\forall xy. x \in S \land y \in T \Rightarrow x < y \quad \text{...} \quad \text{All elements of } S \text{ are } < \text{ those in } T
\]
Use case #3: Theory of Strings
Theory of **Strings + Length**

- **Signature \( \Sigma_s \):**
  - Constants from a fixed finite alphabet e.g. “a”, “ab”, ...
  - String concatenation \(_ \cdot _: \text{Str} \times \text{Str} \rightarrow \text{Str}\)
  - Length \( \text{len}(\_): \text{Str} \rightarrow \text{Int}\)
  - Extended functions \(\text{str\.substr}, \text{str\.contains}, \text{str\.to\.int}, \text{int\.to\.str}, \text{str\.replace}, \text{str\.indexOf}\)

- **Example input:**
  
  \[
  \text{len}(x) > \text{len}(y) \land \text{str\.contains}(y, "ab")
  \]
char buff[15];
char pass;
cout << "Enter the password :";
gets(buff);
if (regex_match(buff, std::regex("([A-Z]+)"))) {
    if(strcmp(buff, "PASSWORD")) {
        cout << "Wrong Password";
    } else {
        cout << "Correct Password";
        pass = 'Y';
    }
} else {
    cout << "Wrong Password";
}
if(pass == 'Y') {  /* Grant the root permission*/  

• Models may correspond to security vulnerabilities

(set-logic QF_S)
(declare-const input String)
(declare-const buff String)
(declare-const pass0 String)
(declare-const rest String)
(declare-const pass1 String)
(assert (= (str.len buff) 15))
(assert (= (str.len pass1) 1))
(assert (or (< (str.len input) 15) (= input (str.+ buff pass0 rest))))
(assert (str.in.re buff re.+ (re.range "A" "Z")))
(assert (ite (= buff "PASSWORD")
             (= pass1 "Y")
             (= pass1 pass0))
(assert (not (= buff "PASSWORD"))
(assert (= pass1 "Y"))

encode: Theory of Strings + Length : Models

- Models may correspond to security vulnerabilities
Theory of Strings + Length

• Theoretical complexity of:
  • Word equation problem is in PSPACE
  • ...with length constraints is OPEN
  • ...with extended functions is UNDECIDABLE

• Instead, focus on:
  • Solver that is efficient in practice
    • Often, for applications like symbolic execution, able to find models
Theory of Strings + Length

- Rule-based algebraic calculus [Liang et al 2014]:
  - Handled unbounded strings
  - E.g. HAMPI [Kiezun et al 2009] reduces to fixed-width Bit Vectors
  - Refutation-sound and model-sound, e.g. “unsat” and “sat” can be trusted
  - Refutation-incomplete, not guaranteed to terminate for “unsat”
  - Finite-model complete
    - ...assuming a branch and bound strategy
Branch and Bound: Theory of Strings + Length

• Given input $F[s_1, ..., s_n]$ for strings $s_1...s_n$:

$$\sum_{i=1}^{n}|s_i| \leq 0$$

Search for models where sum of lengths=0

$$\sum_{i=1}^{n}|s_i| \leq 1$$

Search for models where sum of lengths=1

$$\sum_{i=1}^{n}|s_i| \leq 2$$

etc.

$$\neg \sum_{i=1}^{n}|s_i| \leq 0$$

$$\neg \sum_{i=1}^{n}|s_i| \leq 1$$

$$\neg \sum_{i=1}^{n}|s_i| \leq 2$$

⇒ Incrementally bound the sum of lengths of strings
Use case #4: Syntax-Guided Synthesis
Syntax-Guided Synthesis

\[ \exists f: \text{Prog.} \forall i. S(f, i) \]

• Interested in synthesis conjectures of the above form:

There exists a program \( f \),

...such that for all inputs \( i \),

...a (universal) specification \( S(f, i) \) holds
Syntax-Guided Synthesis

\[ \exists f: \text{Prog.} \forall i. S(f, i) \]

- Problem is **UNDECIDABLE**
  - Involves second-order \( \forall \) on \( f \), universal \( \forall \) on \( i \)
Syntax-Guided Synthesis

\[ \exists f : \text{Prog.} \forall i. S(f, i) \]

\[
P = \text{ite}(C, P, P) | +(P, P) | -(P, P) | 0 | 1 | i
\]

\[
C = \geq(P, P) | =(P, P) | \text{not}(C)
\]

• Problem is UNDECIDABLE
  • Involves second-order \( \forall \) on \( f \), universal \( \forall \) on \( i \)

• A way to simplify the problem is to restrict the space of solutions
  • Solutions belong to a grammar \( P \) specifying syntax for \( f \)
Syntax-Guided Synthesis

\[ \exists f: \mathcal{P}. \forall i. S_{\mathcal{P}}(f, i) \]

\[ \mathcal{P} = \text{ite}(C, \mathcal{P}, \mathcal{P}) | +(\mathcal{P}, \mathcal{P}) | -(\mathcal{P}, \mathcal{P}) | 0 | 1 | i \]

\[ C = \geq(\mathcal{P}, \mathcal{P}) | = (\mathcal{P}, \mathcal{P}) | \text{not}(C) \]

- Problem is UNDECIDABLE
  - Involves second-order \( \forall \) on \( f \), universal \( \forall \) on \( i \)
- A way to simplify the problem is to restrict the space of solutions
  - Solutions belong to a grammar \( \mathcal{P} \) specifying syntax for \( f \)
- Grammar \( \mathcal{P} \) can be seen in SMT as an inductive datatype
  - Use deep embedding into specification \( S_{\mathcal{P}} \), solve for \( f \) as \( \mathcal{P} \) [Reynolds et al CAV15]
Syntax-Guided Synthesis

\[ \exists f : P. \forall i. S_p(f, i) \]

\[ P = \text{ite}(C, P, P) \mid + (P, P) \mid - (P, P) \mid 0 \mid 1 \mid i \]

\[ C = \geq (P, P) \mid = (P, P) \mid \text{not}(C) \]

• Consider solutions (naively) by enumeration:

  \[ f^M = 0 \quad \text{check } \forall i. S_p(0, i) \]
  \[ f^M = 1 \quad \text{check } \forall i. S_p(1, i) \]
  \[ f^M = \ldots \]
  \[ f^M = 1+1 \]
  \[ f^M = i+1 \]
  \[ f^M = \ldots \]
  \[ f^M = \text{ite}(\geq(i, 0), i, 0) \]

  \[ \text{check } \forall i. S_p(\text{ite}(\geq(i, 0), i, 0), i) \]

• In practice, guided via CE-guided inductive synthesis loop [Solar-Lezama 2013]
Syntax-Guided Synthesis

\[ \exists f : P. \forall i. S_p(f, i) \]

\[ P = \text{ite}(C, P, P) | + (P, P) | - (P, P) | 0 | 1 | i \]
\[ C = \geq (P, P) | = (P, P) | \text{not}(C) \]

- Consider solutions (naively) by enumeration:

  \[
  \begin{align*}
  f^M &= 0 & & \text{check } \forall i. S_p(0, i) \\
  f^M &= 1 & & \text{check } \forall i. S_p(1, i) \\
  f^M &= \ldots & & \ldots \\
  f^M &= 1+1 & & \text{check } \forall i. S_p(1+1, i) \\
  f^M &= i+1 & & \text{check } \forall i. S_p(i+1, i) \\
  f^M &= \ldots & & \ldots \\
  f^M &= \text{ite}(\geq (i, 0), i, 0) & & \text{check } \forall i. S_p(\text{ite}(\geq (i, 0), i, 0), i) 
  \end{align*}
  \]

- In practice, guided via CE-guided inductive synthesis loop [Solar-Lezama 2013]
  \[ \Rightarrow \text{Finite-model completeness} \] if we consider smaller solutions before larger ones
Syntax-Guided Synthesis

• To enumerate smaller solutions before larger ones:
  • Introduce notion of term size of datatype (# constructor applications), e.g.:
    • size(i) = 1
    • size(i+1) = 3
    • size(ite(i≥0,i,i+1)) = 8

• Extend theory of datatypes with size bound predicates:
  • size(t) ≤ k
    ... where t is a datatype term and numeral k
  • Decision procedure extends to predicates of this form
Branch and Bound: Syntax-Guided Synthesis

\[ \exists f : P. \forall i. S(f, i) \]

Search for programs of size 1

- \( \text{size}(f) \leq 1 \)
- \( \neg \text{size}(f) \leq 1 \)

Search for programs of size 2

- \( \text{size}(f) \leq 2 \)
- \( \neg \text{size}(f) \leq 2 \)

Search for programs of size 3

- \( \text{size}(f) \leq 3 \)
- \( \neg \text{size}(f) \leq 3 \)

\[ \Rightarrow \text{Use size bound predicate on inductive datatypes} \]
Each of these variants:

• Modify DPLL search
  • ...to minimize some (numeric) quantity:
    • Finite model finding: cardinality of sorts
    • Bounded integer $\forall$: value of numeric bounds
    • Bounded set membership: cardinality of sets
    • Strings: sum of lengths
    • Syntax-guided synthesis: term size

• Have similar challenges/tradeoffs for strategies:
  • Minimal $\Rightarrow$ finite-model complete, slow
  • Non-minimal $\Rightarrow$ incomplete, can be fast
Current Trends in SMT

• Incorporation of many **new theories:**
  • Strings and regular expressions
  • Floating point
  • Sets with cardinality constraints
  • Finite Relations
  • ...

• Increased **support for** $\forall$

• **New solving algorithms**
  • Natural domain SMT, mcSat [Jovanovic/deMoura 2013]

• Some work on **Optimization** Modulo Theories
Optimization Modulo Theories

• Some SMT solvers support optimization queries:
  • νZ (extension of Z3) [Bjorner/Phan 2014]
  • OptiMathSAT (extension of MathSat) [Sebastiani/Tomasi 2014]
Optimization Modulo Theories

\[ F[cost] \cup l \leq cost \leq u \]

• Given input \( F[cost] \) where \( l \leq cost \leq u \),
  • Find model that minimizes \( cost \)
Optimization Modulo Theories

\[ F[cost] \cup l \leq cost \leq u \]

return \( cost = l \)

...if \( l = u \)
Optimization Modulo Theories

\[ F[\text{cost}] \cup l \leq \text{cost} \leq u \]

\[ \text{cost} < \text{pivot} \]

\[ \neg \text{cost} < \text{pivot} \]

• Otherwise, split on pivot for some \( l < \text{pivot} < u \)
• If we find model where $\text{cost}^M = v$, update upper bound
Optimization Modulo Theories

If we find model where \( \text{cost}^M = v \), update upper bound

\[ F[\text{cost}] \cup 1 \leq \text{cost} \leq u \]

\( \text{cost} < \text{pivot} \)

\( \neg \text{cost} < \text{pivot} \)

\( 1 \leq \text{cost} \leq v \)
Optimization Modulo Theories

\[ F[cost] \cup \ l \leq cost \leq u \]

\[ cost < pivot \]
\[ \neg cost < pivot \]

• If no model found, update lower bound

\[ \text{UNSAT} \]
Optimization Modulo Theories

\( F[\text{cost}] \cup l \leq \text{cost} \leq u \)

\( \text{cost} < \text{pivot} \)

\( \neg \text{cost} < \text{pivot} \)

\( \text{pivot} \leq \text{cost} \leq u \)

• If no model found, update lower bound
Optimization Modulo Theories

• Similarly, uses branch and bound to minimize cost
  • Modify the behavior of the DPLL search

• Improvements:
  • Use LP solvers to minimize size of cost in models
  • Use conflict analysis to terminate when “unsat” does not depend on cost
Future Work

Expressivity of Constraints

<table>
<thead>
<tr>
<th></th>
<th>LA constraints</th>
<th>SMT(LA)</th>
<th>SMT(LA+T)</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Satisfiability</td>
<td></td>
<td></td>
<td></td>
<td>SMT</td>
</tr>
<tr>
<td>Optimization (LA)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Expressivity of Queries

...
## Future Work

### Expressivity of Constraints

<table>
<thead>
<tr>
<th>Expressivity of Queries</th>
<th>LA constraints</th>
<th>SMT(LA)</th>
<th>SMT(LA+T)</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Satisfiability</td>
<td></td>
<td></td>
<td></td>
<td>SMT</td>
</tr>
<tr>
<td>Optimization (LA)</td>
<td></td>
<td></td>
<td>LP</td>
<td>LGDP</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

⇒ Extensions of **optimization** queries for **rich set of theories** supported by SMT solvers
Summary

• SMT solvers + DPLL(T) used in many applications
• Can be modified to support **model finding** and **optimization**
  • Extensions of theories, e.g. native support for cardinality
  • Modifications to decision heuristics in SAT solver
Thanks for listening!

• SMT Solver CVC4:
    • Supports many theories:
      • UF, Linear arithmetic, Arrays, Strings, Sets, …
    • and techniques mentioned in this talk:
      • Finite model finding, syntax-guided synthesis, etc.