Model Finding for Recursive Functions in SMT

Andrew Reynolds

Jasmin Christian Blanchette

Cesare Tinelli

QUANTIFY August 3, 2015

Recursive Functions

- Recursive function definitions:

```
f( x:Int ) := if x≤0 then 0 else f(x-1)+x
```

- Are useful in applications:
  - Software verification
  - Theorem Proving
- Often, interested in finding models for
  - Conjectures `(∃x.)P(f,x)` in the presence of recursive functions `f`
    - This poses a challenge to current Satisfiability Modulo Theories (SMT) solvers

Recursive Functions

- Recursive function definitions:

$$f(\ x{:}Int\ )\ :=\ \text{if}\ x\leq 0\ \text{then}\ 0\ \text{else}\ f(x-1)+x$$

- Can be expressed in SMT as quantified formulas (with theories):

$$\forall x{:}Int.\ f(x)=ite(x\leq 0,0,f(x-1)+x)$$

- SMT solver must handle inputs of the form:

$$\left(\begin{array}{c}\forall \mathbf{x}.f_1(\mathbf{x})=t_1\\ \ldots\\ \forall \mathbf{x}.f_n(\mathbf{x})=t_n\end{array}\right) \wedge G$$

Set of function definitions

Conjecture

Recursive Functions

- In this talk:
  - Existing techniques for quantified formulas in SMT
    - Limited in their ability to find models when recursive functions are present
  - A satisfiability-preserving translation $\mathbb{A}$ for function definitions
    - Allows us to use existing techniques for model finding
  - Evaluation of translation $\mathbb{A}$ on benchmarks from theorem proving/verification

Existing Techniques for Quantified Formulas in SMT

- Heuristic Techniques for UNSAT:
  - E-matching [Detlefs et al 2003, Ge et al 2007, de Moura/Bjorner 2007]
- Limited Techniques for SAT:
  - Local theory extensions [Sofronie-Stokkermans 2005]
  - Array fragments [Bradley et al 2006, Alberti et al 2014]
  - Complete Instantiation [Ge/de Moura 2009]
    - Implemented in Z3
  - Finite Model Finding [Reynolds et al 2013]
    - Implemented in CVC4

Existing Techniques for Quantified Formulas in SMT

- Heuristic Techniques for UNSAT:
  - E-matching [Detlefs et al 2003, Ge et al 2007, de Moura/Bjorner 2007]
- Limited Techniques for SAT:
  - Local theory extensions [Sofronie-Stokkermans 2005]
  - Array fragments [Bradley et al 2006, Alberti et al 2014]
  - Complete Instantiation [Ge/de Moura 2009]
    - Implemented in Z3
  - Finite Model Finding [Reynolds et al 2013]
    - Implemented in CVC4

Focus of next slides

Complete Instantiation in **Z3**

- Complete method for ∀ in essentially uninterpreted fragment

$\forall$`x:Int.(f(`**`x`**`)=g(`**`x`**`)+5)` $\wedge$ `f(a)=g(b)`

All occurrences of **x** are children of UF

Complete Instantiation in **Z3**

$\forall x{:}Int.\,(f(x)=g(x)+5) \;\wedge\; f(a)=g(b)$

$R(f_1)=R(g_1)=R(x),\ a\in R(f_1),\ b\in R(g_1)$

$\therefore\ R(x)=\{a,b\}$

Relevant domain $R(x)$ of variable x is $\{a,b\}$

Complete Instantiation in **Z3**

$$\forall x{:}Int.\ (f(x)=g(x)+5)\ \wedge\ f(a)=g(b)$$

equisatisfiable to

$$R(f_1)=R(g_1)=R(x),\ a\in R(f_1),\ b\in R(g_1)$$
$$\therefore\ R(x)=\{a,b\}$$

$$f(a)=g(a)+5\ \wedge\ f(b)=g(b)+5\ \wedge\ f(a)=g(b)$$

SAT

Finite Model Finding in **CVC4**

- Finite Model-complete method for finite/uninterpreted ∀

$$\forall xy{:}\mathbf{U}.\,(x \neq y \Rightarrow f(x) \neq f(y)) \;\wedge\; a \neq b$$

All variables have finite/uninterpreted sort **U**

Finite Model Finding in **CVC4**

$$\forall xy{:}U.\,(x \neq y \Rightarrow f(x) \neq f(y)) \;\wedge\; a \neq b$$

M(U) := {a,b}

Model interprets U as the set M(U)={a,b}

Finite Model Finding in **CVC4**

$\forall xy{:}U.(x \neq y \Rightarrow f(x) \neq f(y)) \wedge a \neq b$

equisatisfiable to

$M(U) := \{a,b\}$

$a \neq a \Rightarrow f(a) \neq f(a)$
$a \neq b \Rightarrow f(a) \neq f(b)$
$b \neq a \Rightarrow f(b) \neq f(a)$
$b \neq b \Rightarrow f(b) \neq f(b)$
$\wedge\ a \neq b$

SAT

…Both fail on most Recursive Function Definitions!

- Example:

$$\forall x{:}Int.(f(x)=ite(x \leq 0,0,f(x-1)+x)) \wedge f(k)>100$$

…Both fail on most Recursive Function Definitions!

- Example:

$$\forall x{:}Int.\,(f(x) = ite(x \leq 0, 0, f(\mathbf{x}-1)+\mathbf{x})) \wedge f(k) > 100$$

- Complete instantiation:
  - Fails, since body has subterm $f(\mathbf{x}-1)+\mathbf{x}$ with unshielded variable **x**
    - $R(x)=\{k, k-1, k-2, k-3, \ldots\}$

…Both fail on most Recursive Function Definitions!

- Example:

$$\forall x{:}\mathbf{Int}.\,(f(x)=ite(x\leq 0,0,f(\mathbf{x}-1)+\mathbf{x}))\ \wedge\ f(k)>100$$

- Complete instantiation:
  - Fails, since body has subterm `f(x-1)+x` with unshielded variable **x**
    - `R(x)={k,k-1,k-2,k-3,…}`
- Finite Model Finding:
  - Fails, since quantification is over infinite type **Int**
    - `M(Int)={…, -3, -2, -1, 0, 1, 2, 3, …}`

Running example

$$\forall x{:}\mathrm{Int}.\,(f(x) = \mathrm{ite}(x \leq 0, 0, f(x-1)+x)) \;\wedge\; f(k) > 100$$

- Function `f`
  - Returns the sum of all positive integers up to `x`, when `x` is non-negative
- Formula is satisfiable
  - By models interpreting k as an integer ≥ 14

Can we make the problem easier?

$$\underbrace{\forall x{:}\mathtt{Int}.\,(f(x)=\mathrm{ite}(x\leq 0,0,f(x-1)+x))\ \wedge\ f(k)>100}_{\Phi}$$

- What if we assume function definitions in Φ are *well-behaved*?
  - E.g. we know that `f` is terminating
- Introduce translation $\mathbb{A}$, which:
  - Restricts quantification to subset of the domain of function definitions
  - Under right assumptions, preserves satisfiability
- Use existing techniques for model finding in Z3, CVC4 on $\mathbb{A}(\Phi)$

Translation A

```
∀x:Int.ite(x≤0,
          f(x)=0,
          f(x)=f(x-1)+x))∧
f(k)>100
```

Translation $\mathtt{A}$: Part 1

$$
\begin{aligned}
&\forall \mathtt{x}{:}\boldsymbol{\alpha}.\mathtt{ite}(\boldsymbol{\gamma}(\mathtt{x}) \leq 0,\\
&\qquad\qquad \mathtt{f}(\boldsymbol{\gamma}(\mathtt{x})) = 0,\\
&\qquad\qquad \mathtt{f}(\boldsymbol{\gamma}(\mathtt{x})) = \mathtt{f}(\boldsymbol{\gamma}(\mathtt{x}) - 1) + \boldsymbol{\gamma}(\mathtt{x})) \wedge\\
&\mathtt{f}(\mathtt{k}) > 100
\end{aligned}
$$

- Introduce uninterpreted sort $\boldsymbol{\alpha}$
  - Conceptually, α represents the set of relevant arguments of $\mathtt{f}$
    - Restrict the domain of function definition quantification to $\boldsymbol{\alpha}$
- Introduce uninterpreted function $\boldsymbol{\gamma} : \alpha \rightarrow \mathtt{Int}$
  - Maps between abstract and concrete domains

Translation $\mathbb{A}$: Part 2

$$
\begin{aligned}
&\forall x{:}\alpha.\,\mathtt{ite}(\gamma(x) \le 0,\\
&\qquad f(\gamma(x)) = 0,\\
&\qquad f(\gamma(x)) = f(\gamma(x)-1) + \gamma(x) \land \mathbf{(\exists z{:}\alpha.\,\gamma(z)=\gamma(x)-1)}) \land\\
&f(k) > 100 \;\land\; \mathbf{(\exists z{:}\alpha.\,\gamma(z)=k)}
\end{aligned}
$$

- Add appropriate constraints regarding α, γ
  - Each relevant concrete value must be mapped to by some abstract value

Translation A

$$\forall x{:}\alpha.\,\mathtt{ite}(\gamma(\mathbf{x})\le 0,\ \ \mathtt{f}(\gamma(\mathbf{x}))=0,\ \ \mathtt{f}(\gamma(\mathbf{x}))=\mathtt{f}(\gamma(\mathbf{x})-1)+\gamma(\mathbf{x})\wedge(\exists z{:}\alpha.\gamma(z)=\gamma(\mathbf{x})-1))\wedge$$
$$\mathtt{f}(k)>100\ \wedge\ (\exists z{:}\alpha.\gamma(z)=k)$$

- $\forall$ is essentially uninterpreted

Translation A

```
∀x:α.ite(γ(x)≤0,
         f(γ(x))=0,
         f(γ(x))=f(γ(x)-1)+γ(x)∧(∃z:α.γ(z)=γ(x)-1))∧
f(k)>100 ∧ (∃z:α.γ(z)=k)
```

- ∀ is essentially uninterpreted, and over finite/uninterpreted sorts

Translation A

```
∀x:α.ite(γ(x)≤0,
         f(γ(x))=0,
         f(γ(x))=f(γ(x)-1)+γ(x)∧(∃z:α.γ(z)=γ(x)-1))∧
f(k)>100 ∧ (∃z:α.γ(z)=k)
```

- ∀ is essentially uninterpreted, and over finite/uninterpreted sorts
  - ⇒Both **Z3** (complete instantiation) and **CVC4** (finite model finding) find model for this benchmark in <.1 second

Translation $\mathtt{A}$

$$\forall x{:}\alpha.\,\mathtt{ite}(\gamma(x)\leq 0,\ f(\gamma(x))=0,\ f(\gamma(x))=f(\gamma(x)-1)+\gamma(x)\wedge(\exists z{:}\alpha.\gamma(z)=\gamma(x)-1))\wedge$$
$$f(k)>100\ \wedge\ (\exists z{:}\alpha.\gamma(z)=k)$$

- Formula is satisfied by a model **M** where:
  - $\mathtt{M}(k) := 14$, $\mathtt{M}(f) := \lambda\, x.\mathtt{ite}(x=14, 105, \mathtt{ite}(x=13, 91, \ldots\ \mathtt{ite}(x=1, 1, 0)\ldots))$

$\Rightarrow$ *M is correct only for relevant inputs of original formula, and not e.g.* $f(15)=0$

- Nevertheless, $\mathtt{A}$ is satisfiability-preserving under right assumptions

Translation $\mathbb{A}$: Properties

- Translation $\mathbb{A}$ is:
  - Refutation sound
    - When $\mathbb{A}(\Phi)$ is unsatisfiable, Φ is unsatisfiable
  - Model sound, when function definitions are admissible
    - When $\mathbb{A}(\Phi)$ is satisfiable, Φ is satisfiable

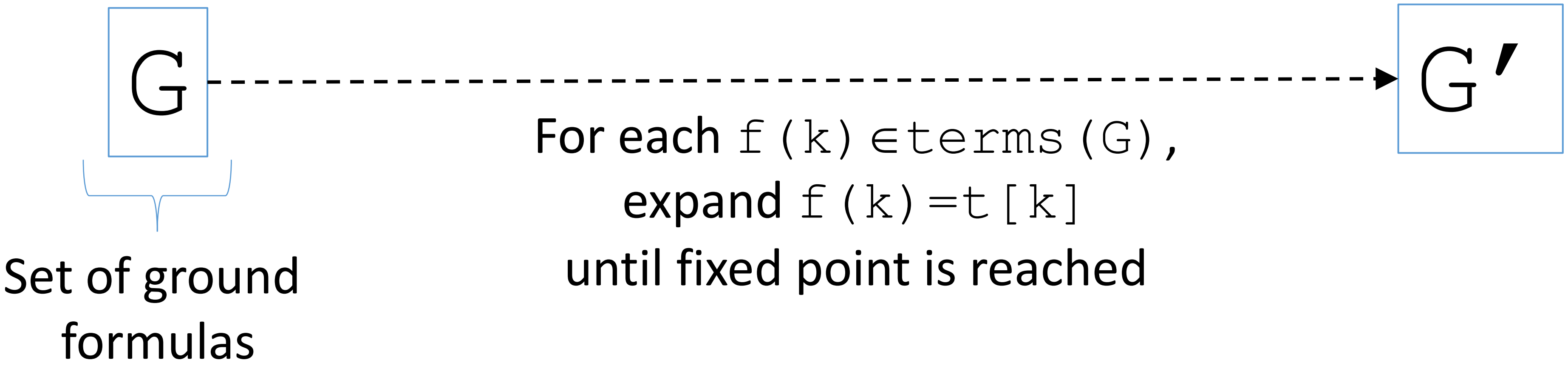
Translation $\mathbb{A}$: Properties

- Translation $\mathbb{A}$ is:
  - Refutation sound
    - When $\mathbb{A}(\Phi)$ is unsatisfiable, Φ is unsatisfiable
  - Model sound, when function definitions are ***admissible***
    - When $\mathbb{A}(\Phi)$ is satisfiable, Φ is satisfiable

Focus of next slides

Admissible Function Definitions

- Given a function definition $\forall x.f(x)=t[x]$



- The definition $\forall x.f(x)=t$ is *admissible* if:

G' has model $\Rightarrow$ $G' \wedge \forall x.f(x)=t[x]$ is also has model

Admissible Function Definitions

- Examples of admissible definitions:
  - Terminating functions: $\forall x.f(x)=ite(x\leq 0,0,f(x-1)+x)$
    - …f is well-founded (terminating)
  - Even non-terminating, tail recursive: $\forall x.f(x)=f(x-1)+1$

Inadmissible Function Definitions

- Examples of inadmissible definitions:
  - Inconsistent definitions: $\forall$`x.f(x)=f(x)+1`
    - …no model for $\forall$`x.f(x)=f(x)+1`
  - Others: `{`$\forall$`x.f(x)=f(x)+g(x),` $\forall$`x.g(x)=g(x)}`
    - …some ground formulas are inconsistent wrt these definitions
    - Such cases are subtle, but rarely occur in practice

Evaluation

- Considered two sets of benchmarks:
  - **Isa**
    - Challenge problems for inductive theorem provers
    - Purely datatypes + recursive functions
  - **Leon**
    - Taken from Leon verification tool (EPFL)
    - Many theories: datatypes + recursive functions + bitvectors + arrays + sets + arithmetic
- Consider mutated forms of these benchmarks (**Isa-mut, Leon-mut**)
  - Obtained by swapping subterms in conjectures
  - High likelihood to have models
- All benchmarks considered with/without translation $\mathbb{A}$

Evaluation : solved SAT benchmarks

	Z3		CVC4f		
	φ	$\mathcal{A}(\varphi)$	φ	$\mathcal{A}(\varphi)$	Total
Isa	0	0	0	0	79
Leon	0	2	0	**9**	166
Isa-Mut	0	35	0	**153**	213
Leon-Mut	11	75	6	**169**	427
Total	11	112	6	**331**	885

- Translation increases ability of SMT solvers for finding models:
  - Z3: 11 -> 112
  - CVC4: 6 -> 331
- Finds counterexamples to verification conditions of interest in **Leon**

Evaluation : solved UNSAT benchmarks

	Z3		CVC4f		Total
	φ	$\mathcal{A}(\varphi)$	φ	$\mathcal{A}(\varphi)$	
Isa	14	15	**15**	15	79
Leon	73	78	**80**	76	166
Isa-Mut	17	18	**18**	18	213
Leon-Mut	83	98	**104**	95	427
Total	187	209	**217**	204	885

- Translation has mixed impact on UNSAT benchmarks:
  - Z3 : 187 -> 209
  - CVC4 : 217 -> 204

Translation as Preprocessor in CVC4

- CVC4 supports SMT LIB version 2.5 command:

```
…
(define-fun-rec f ((x Int)) Int
     (ite (<= x 0) 0 (+ (f (- x 1)) x)))
(assert (> (f k) 100))
(check-sat)
```

Translation as Preprocessor in CVC4

- Input (without `A`) is equivalent to:

```
…
(assert (forall ((x Int))
     (= (f x) (ite (<= x 0) 0 (+ (f (- x 1)) x)))))
(assert (> (f k) 100))
(check-sat)
```

Translation as Preprocessor in CVC4

- Input (with `A`) is equivalent to:

```
…
(declare-sort a 0)
(declare-fun g (a) Int)
(assert (forall ((x a))
      (ite  (<= (g x) 0)
            (= (f (g x)) 0)
            (and (= (f (g x)) (+ (f (- (g x) 1)) (g x))
                  (exists ((z a)) (= (g z) (- (g x) 1))))))))
(assert (and (> (f k) 100) (exists ((z a)) (= (g z) k)))
(check-sat)
```

⇒ Enabled as preprocessor by command line parameter "`--fmf-fun`"

Translation as Preprocessor in CVC4

- Model (with `A`) outputted is:

```
(model
(define-fun f (($x1 Int)) Int
      (ite (= $x1 14) 105 (ite (= $x1 13) 91 (ite (= $x1 12) 78
      (ite (= $x1 11) 66 (ite (= $x1 10) 55 (ite (= $x1 4) 10
      (ite (= $x1 9) 45 (ite (= $x1 8) 36 (ite (= $x1 7) 28
      (ite (= $x1 6) 21 (ite (= $x1 3) 6 (ite (= $x1 5) 15
      (ite (= $x1 2) 3 (ite (= $x1 1) 1 0)))))))))))))))
(define-fun k () Int 14))
```

- Gives model that is correct for relevant inputs of function `f`

Summary

- Translation `A`:
  - Increases ability of SMT solvers for model finding recursive functions
    - Complete instantiation in Z3
    - Finite Model Finding in CVC4
  - Is model-sound for admissible function definitions
  - Implemented as a preprocessor in CVC4 "`--fmf-fun`"
    - Responsibility on user to show function definitions are admissible

Future Work

- Increase scope of evaluation
  - Comparison against existing counterexample generators (Leon, Nitpick, …)
- Use of CVC4 as backend
  - To Leon verification system
  - To Isabelle proof assistant
- Identify additional sufficient conditions for admissibility
  - E.g. productive corecursive functions

Thanks!

- CVC4:
  - Available at http://cvc4.cs.nyu.edu/downloads/

- To use translation `A` as a preprocessor:
  - Use command line option “`--fmf-fun`”