Model Finding for Recursive Functions in SMT

Andrew Reynolds

Jasmin Christian Blanchette

Simon Cruanes

Cesare Tinelli

IJCAR June 30, 2016

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 `(∃k.)P[f,k]` 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 ) := if x≤0 then 0 else f(x-1)+x`

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

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

- SMT solver must handle inputs of the form:

$$\left.\begin{array}{c}\forall \mathbf{x}.\, f_1(\mathbf{x}) = t_1 \\ \dots \\ \forall \mathbf{x}.\, f_n(\mathbf{x}) = t_n\end{array}\right. \wedge\ (\exists \mathbf{k}.)\, P[f_1 \dots f_n, \mathbf{k}]$$

Set of function definitions

Conjecture

Outline

- 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
  - Implementation of translation $\mathbb{A}$
    - As a preprocessor in SMT solver **CVC4**
    - In model finder for HOL **Nunchaku**
  - Evaluation 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

$$\forall x{:}Int.\ (\underbrace{f(\mathbf{x})=g(\mathbf{x})+5}_{\text{All occurrences of } \mathbf{x} \text{ are children of UF}}) \wedge f(a)=g(b)$$

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

Complete Instantiation in **Z3**

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

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

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

Complete Instantiation in **Z3**

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

equisatisfiable to

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

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

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

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}`

$$\begin{array}{l} a \neq a \Rightarrow f(a) \neq f(a) \\ a \neq b \Rightarrow f(a) \neq f(b) \\ b \neq a \Rightarrow f(b) \neq f(a) \\ b \neq b \Rightarrow f(b) \neq f(b) \end{array} \ \wedge\ a \neq b$$

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:

  `∀x:Int.(f(x)=ite(x≤0,0,f(x-1)+x)) ∧ 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,…}`

…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

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

- Example:
  - **f** returns the sum of all positive integers up to `x`, when `x` is non-negative
  - `f(k)` is greater than 100 for some **k**
- Formula is satisfiable: interpret $k\geq 14$

⇒Neither **Z3** or **CVC4** can establish "sat" for this benchmark

Can we make the problem easier?

$$\Phi \left\{ \begin{array}{l} \texttt{∀x:Int.(f(x)=ite(x≤0,0,f(x-1)+x))} \wedge \\ \texttt{f(k)>100} \end{array} \right.$$

- What if we assume function definitions in Φ are *well-behaved*?
  - E.g. we know that `f` is terminating

Can we make the problem easier?

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

- What if we assume function definitions in Φ are *well-behaved*?
  - E.g. we know that `f` is terminating

$\Rightarrow$ Then, we may restrict $\forall$ to subset of the domain of function definitions

Can we make the problem easier?

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

$\}\ \Phi$

- What if we assume function definitions in Φ are *well-behaved*?
  - E.g. we know that `f` is terminating

$\Rightarrow$ Then, we may restrict $\forall$ to subset of the domain of function definitions and....

$\}$ Translation "A"

Use existing techniques for model finding in **Z3**, **CVC4** on $\mathtt{A}(\Phi)$

Translation A

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

Translation $\mathtt{A}$: Part 1

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

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

Translation $\mathbb{A}$: Part 2

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

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

Translation A

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

- $\forall$ is essentially uninterpreted

Translation A

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

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

Translation $\mathbb{A}$

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

- $\forall$ 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 A

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

- Formula is satisfied by a model M where:
  - M (**k**) := **14**
  - M (f) := λ x.ite(x=14,105,ite(x=13,91,… ite(x=1,1,0)…))

Translation $\mathtt{A}$

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

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

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

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

- **Intuition**:
  - If we evaluate f-applications in conjecture until a fixed point is reached:

$\forall x. f(x) = t[x]$
$f(k) \geq 100$

Ψ

Admissible Function Definitions

- **Intuition**:
  - If we evaluate f-applications in conjecture until a fixed point is reached:

$\forall x. f(x) = t[x]$
$f(k) \geq 100$

→

$\forall x. f(x) = t[x]$
$f(k) \geq 100$

Admissible Function Definitions

- **Intuition**:
  - If we evaluate f-applications in conjecture until a fixed point is reached:

∀x.f(x)=t[x]
f(k)≥100

→

∀x.f(x)=t[x]
f(k)≥100 f(k)=ite(k<0,0,f(k-1)+k)

Admissible Function Definitions

- **Intuition**:
  - If we evaluate f-applications in conjecture until a fixed point is reached:

$$\forall x.f(x)=t[x]$$
$$f(k)\geq 100$$

$$\forall x.f(x)=t[x]$$
$$f(k)\geq 100$$
$$f(k)=ite(k<0,0,\mathbf{f(k-1)}+k)$$
$$f(k-1)=ite(k<0,0,f(k-2)+k)$$

Admissible Function Definitions

- **Intuition**:
  - If we evaluate f-applications in conjecture until a fixed point is reached:

$$\forall x. f(x) = t[x]$$

$$f(k) \geq 100$$

$$\forall x. f(x) = t[x]$$

$$f(k) \geq 100$$
$$f(k) = ite(k<0, 0, f(k-1)+k)$$
$$f(k-1) = ite(k<0, 0, \mathbf{f(k-2)}+k)$$
$$\ldots$$
$$f(k-14) = ite(k<0, 0, f(k-15)+k)$$

Admissible Function Definitions

- **Intuition**:
  - If we evaluate f-applications in conjecture until a fixed point is reached:

$$\forall x. f(x) = t[x]$$
$$f(k) \geq 100$$

Ψ^*

$$\forall x. f(x) = t[x]$$
$$f(k) \geq 100$$
$$f(k) = ite(k<0, 0, f(k-1)+k)$$
$$f(k-1) = ite(k<0, 0, f(k-2)+k)$$
$$\ldots$$
$$f(k-14) = ite(k<0, 0, f(k-15)+k)$$

Admissible Function Definitions

- **Intuition**:
  - If we evaluate f-applications in conjecture until a fixed point is reached:

$$\forall x. f(x) = t[x]$$
$$f(k) \geq 100$$

$$\forall x. f(x) = t[x]$$

Ψ^*:

$$f(k) \geq 100$$
$$f(k) = ite(k<0, 0, f(k-1)+k)$$
$$f(k-1) = ite(k<0, 0, f(k-2)+k)$$
$$\ldots$$
$$f(k-14) = ite(k<0, 0, f(k-15)+k)$$

- Definition of f is admissible if:
  - *Ψ^* has a model $\Leftrightarrow$ $\Psi^* \wedge \forall x. f(x) = t[x]$ has a model*

Admissible Function Definitions

- Given a function definition $\Delta \Leftrightarrow \forall x . f(x) = t[x]$
  - A (ground) formula Ψ^* is closed under function expansion w.r.t Δ if:

    $\Psi^* \models f(k) = t[k]$

    for all f-terms $f(k)$ occurring in Ψ^*
  - Δ is admissible if:

    Ψ^* has a model $\Leftrightarrow$ $\Psi^* \wedge \Delta$ has a model

    for every Ψ^* that is closed under function expansion

Admissible Function Definitions

- Given a function definition $\Delta \Leftrightarrow \forall x.f(x)=t[x]$
  - A (ground) formula Ψ^* is closed under function expansion w.r.t Δ if:

    $\Psi^* \models f(k)=t[k]$
    for all f-terms $f(k)$ occurring in Ψ^*

  - Δ is admissible if:

    Ψ^* has a model $\Leftrightarrow$ $\Psi^* \wedge \Delta$ has a model
    for every Ψ^* that is closed under function expansion

- Thus, to establish $\Delta \wedge \Psi$ has a model, suffices to:

  Find Ψ^* s.t:

  1. $\Psi^* \models \Psi$
  2. Ψ^* is closed under function expansion
  3. Ψ^* has a model

Admissible Function Definitions

- Given a function definition $\Delta \Leftrightarrow \forall \texttt{x.f(x)=t[x]}$
  - A (ground) formula Ψ^* is closed under function expansion w.r.t Δ if:

    $\Psi^* \models \texttt{f(k)=t[k]}$
    for all $\texttt{f}$-terms $\texttt{f(k)}$ occurring in Ψ^*

  - Δ is admissible if:

    Ψ^* has a model $\Leftrightarrow$ $\Psi^* \wedge \Delta$ has a model
    for every Ψ^* that is closed under function expansion

- Thus, to establish $\Delta \wedge \Psi$ has a model, suffices to:

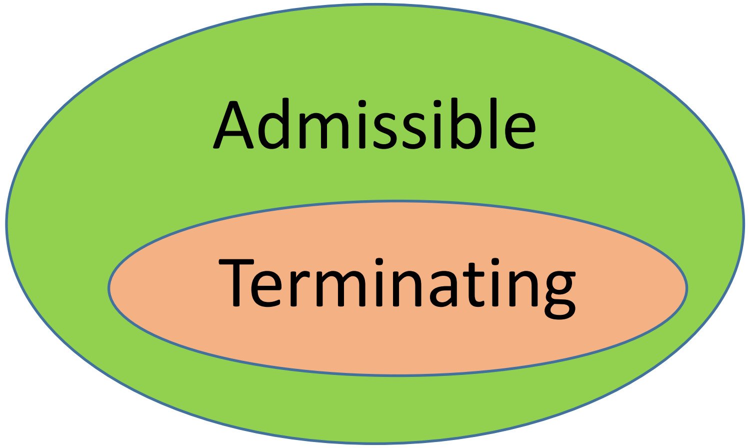
  Find Ψ^* s.t:

  1. $\Psi^* \models \Psi$
  2. Ψ^* is closed under function expansion
  3. Ψ^* has a model

  The SMT solver can do this

Admissible Function Definitions

- Examples of admissible definitions:
  - Terminating functions: `∀x.f(x)=ite(x≤0,0,f(x-1)+x)`
    - …`f` is well-founded (terminating)
  - Some non-terminating, tail recursive: `∀x.f(x)=f(x-1)+1`
  - …and productive corecursive functions



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

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`) found is:

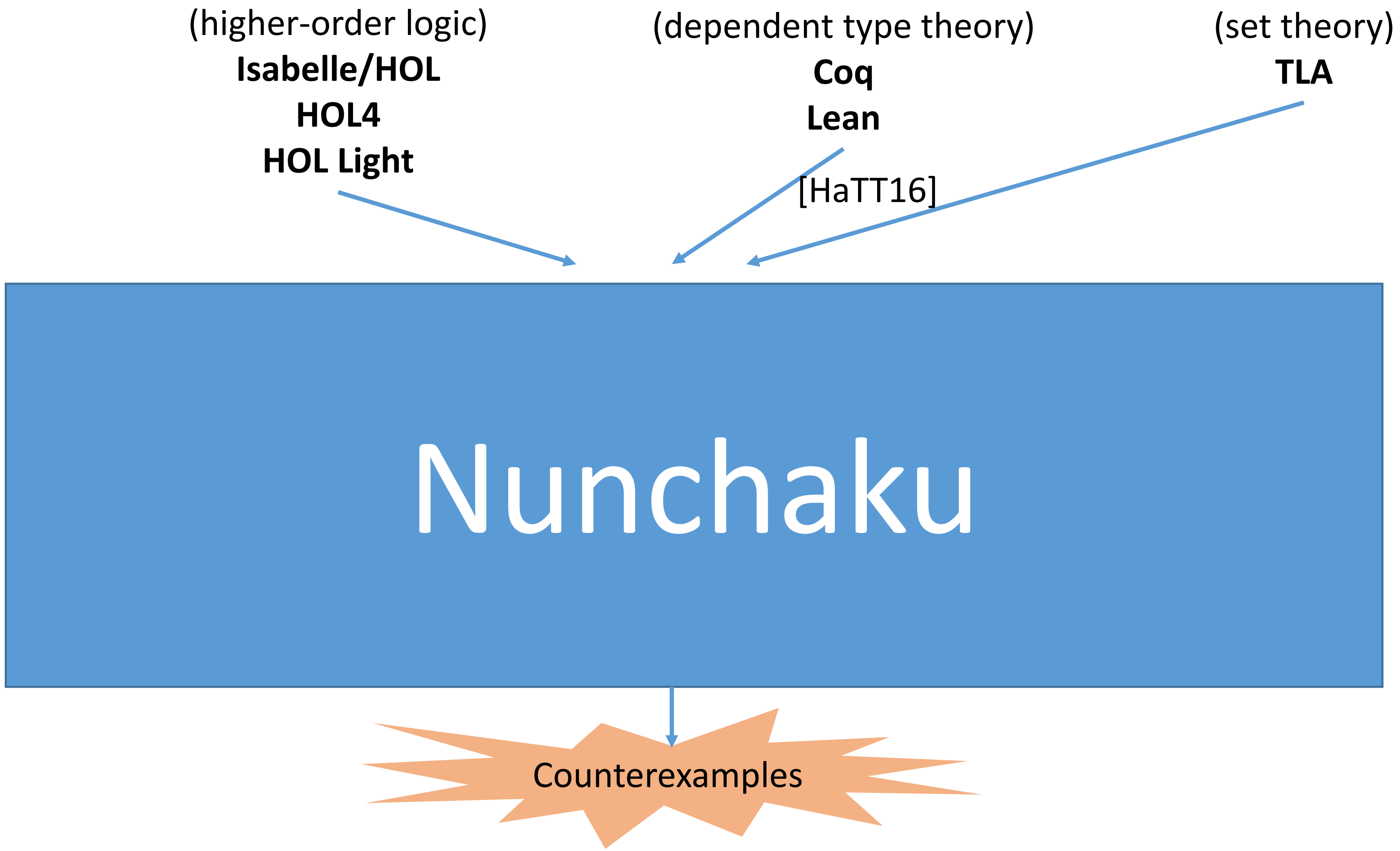
```
(model
(define-fun f (($x1 Int)) Int
      (ite (= $x1 14) 105 (ite (= $x1 13) 91 (ite (= $x1 12) 78
      (ite (= $x1 11) 66 (ite (= $x1 10) 55 (ite (= $x1 4) 10
      (ite (= $x1 9) 45 (ite (= $x1 8) 36 (ite (= $x1 7) 28
      (ite (= $x1 6) 21 (ite (= $x1 3) 6 (ite (= $x1 5) 15
      (ite (= $x1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

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

CVC4: Optimizations for Finite Model Finding

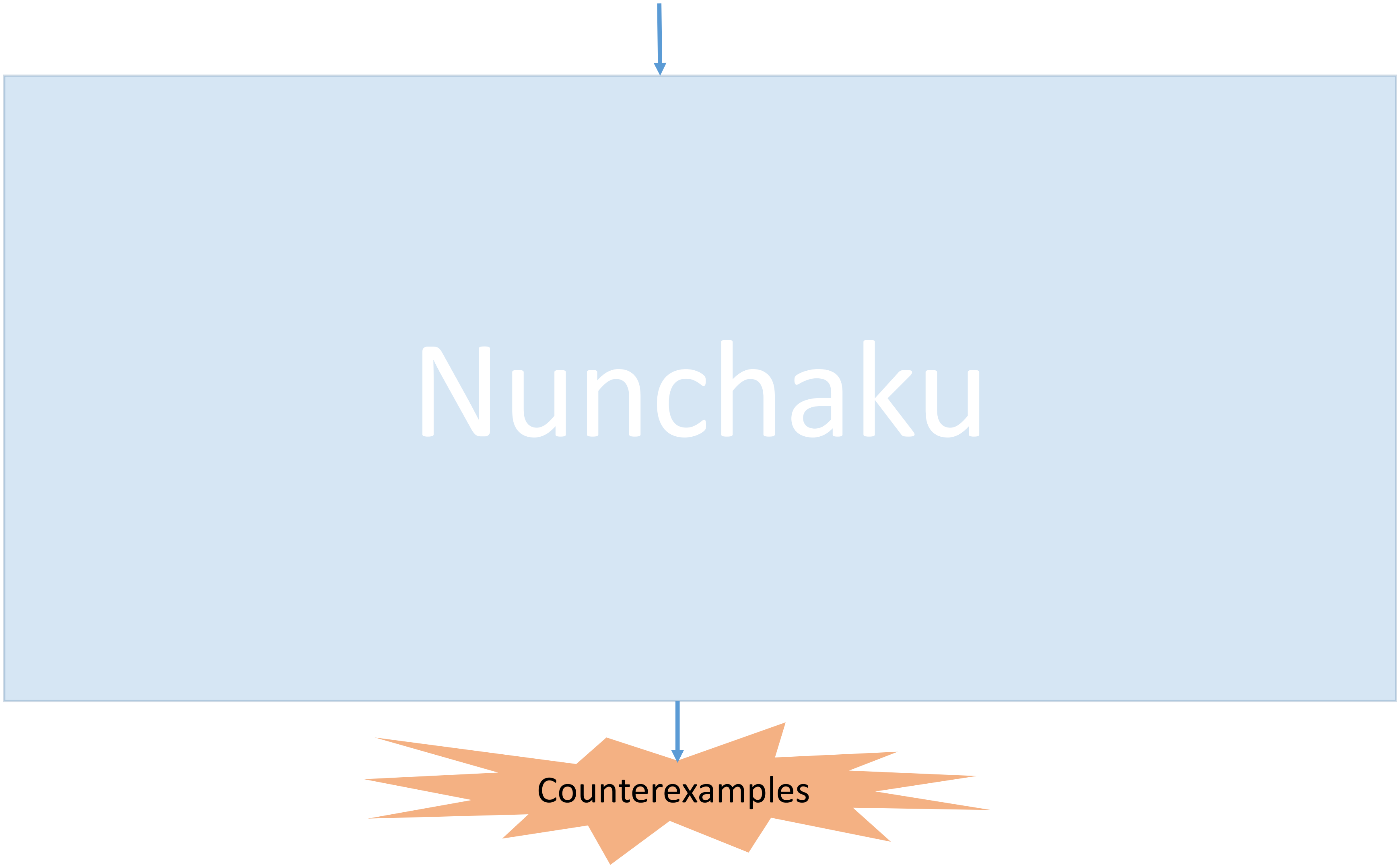
- Considered optimizations specialized to recursive functions:
  - Allow sorts of cardinality 0
  - Infer the monotonicity of sorts
  - Compute minimal satisfying assignments based on relevancy

Nunchaku: counterexamples for HOL

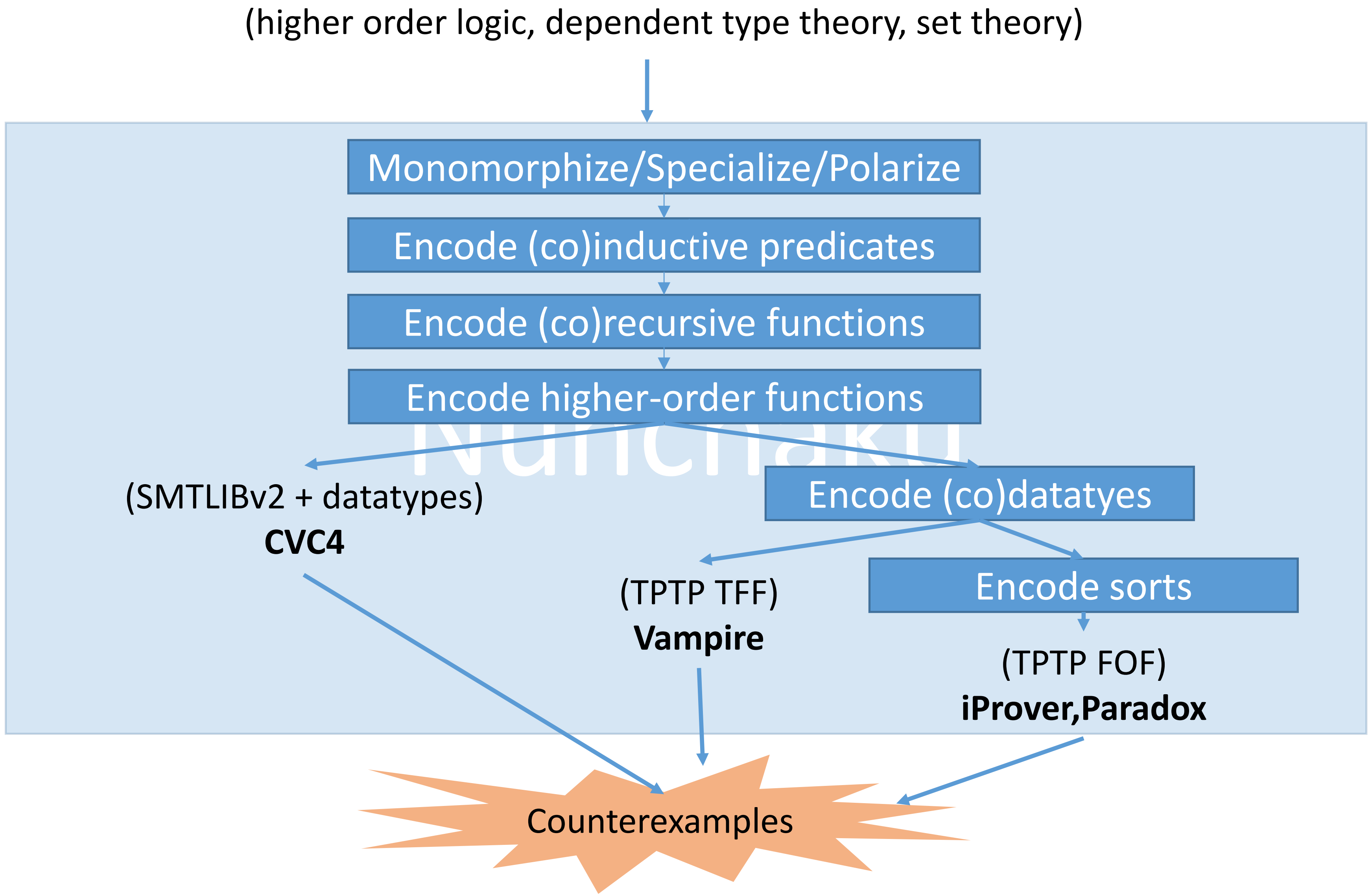


Nunchaku: counterexamples for HOL

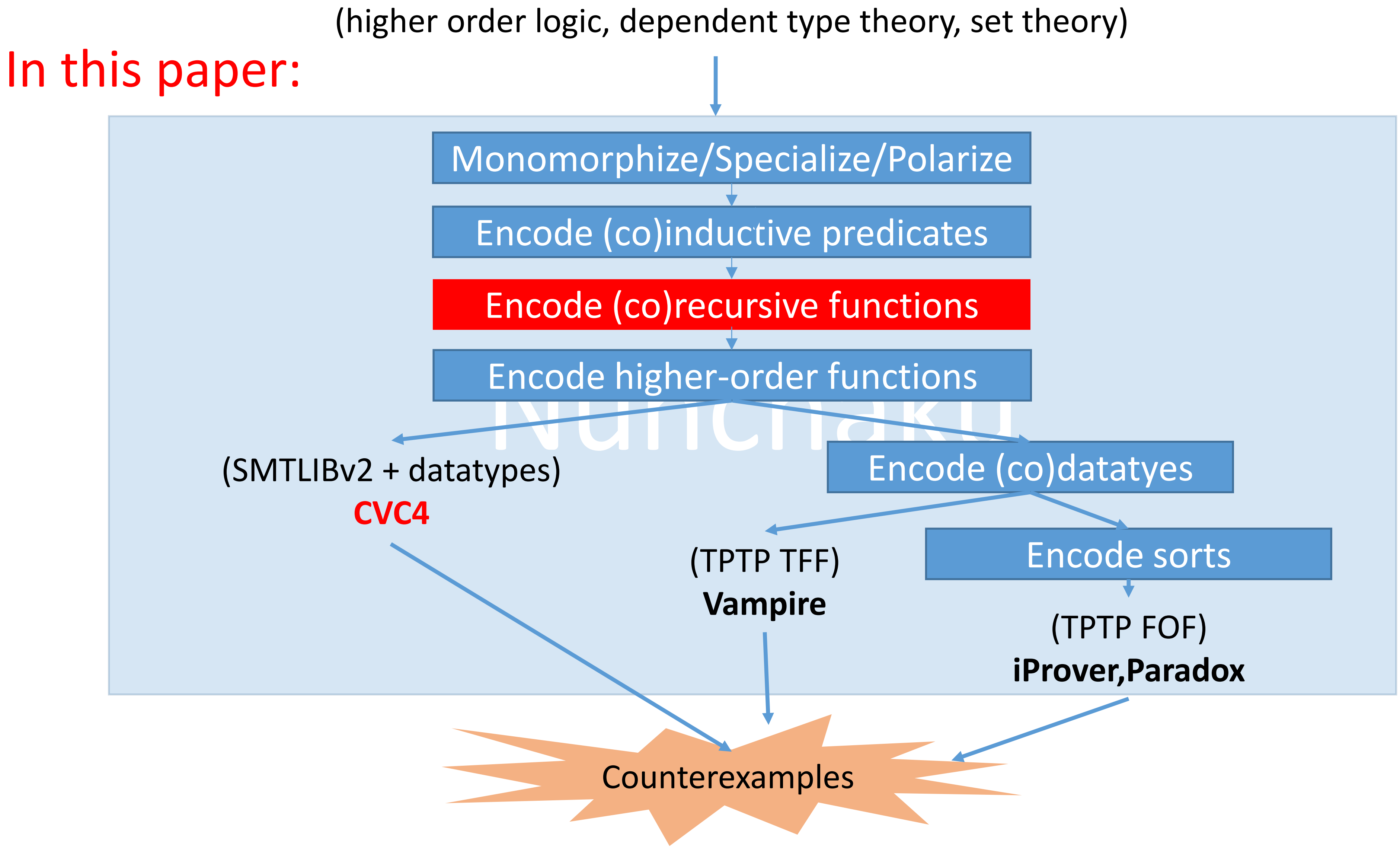
(higher order logic, dependent type theory, set theory)



Nunchaku: counterexamples for HOL



Nunchaku: counterexamples for HOL



Evaluation

- Considered three sets of benchmarks:
  - **Ip**
    - Challenge problems for inductive theorem provers
    - Datatypes + recursive functions
  - **Leon**
    - Verification conditions from Leon verification tool (EPFL)
    - Many theories: datatypes + recursive functions + bitvectors + arrays + sets + arithmetic
  - **Nun-Mut**
    - Mutated form of Isabelle conjectures of interest to Nunchaku project
    - (Co)datatypes + (co)recursive functions
- Consider mutated forms of the first two sets (**Ip-mut, Leon-mut**)
  - Obtained by swapping subterms in conjectures
- All benchmarks considered with/without translation $\mathbb{A}$

Evaluation : solved SAT benchmarks

	Z3		CVC4h		CVC4f		
	φ	$\mathcal{A}(\varphi)$	φ	$\mathcal{A}(\varphi)$	φ	$\mathcal{A}(\varphi)$	#
IsaPlanner	0	0	0	0	0	0	79
IsaPlanner-Mut	0	41	0	0	0	**153**	166
Leon	0	2	0	0	0	**9**	213
Leon-Mut	11	78	6	6	6	**189**	427
Nunchaku-Mut	3	27	0	0	3	**199**	357
Total	14	148	6	6	8	**550**	885

- Translation increases ability of SMT solvers for finding models:
  - Z3: 14 $\rightarrow$ 148
  - CVC4f: 8 $\rightarrow$ 550
- Finds counterexamples to verification conditions of interest in **Leon**

Evaluation : solved UNSAT benchmarks

	Z3		CVC4h		CVC4f		
	φ	$\mathcal{A}(\varphi)$	φ	$\mathcal{A}(\varphi)$	φ	$\mathcal{A}(\varphi)$	#
IsaPlanner	14	**15**	**15**	**15**	1	**15**	79
IsaPlanner-Mut	**18**	**18**	**18**	**18**	4	**18**	166
Leon	74	79	**80**	**80**	17	78	213
Leon-Mut	84	98	**104**	98	24	100	427
Nunchaku-Mut	**61**	59	46	53	45	59	357
Total	251	269	263	264	91	**270**	885

- Translation also improves performance on UNSAT benchmarks:
  - Z3 : 251 $\rightarrow$ 269
  - CVC4 : 263 $\rightarrow$ 264
  - CVC4f : 91 $\rightarrow$ 270

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`"
    - In Nunchaku, a counterexample generator for higher-order logic

Future Work

- Use translation in Nunchaku
  - Support of multiple backends: CVC4, Paradox, Vampire?
- Improved support for finite model finding in SMT
  - Currently the bottlebeck
- 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`"

- Nunchaku
  - Available at https://github.com/nunchaku-inria/