Syntax-Guided Synthesis in SMT: A View from Inside the Solver

Andrew Reynolds

March 1, 2021

THE UNIVERSITY OF IOWA

Satisfiability Modulo Theories (SMT) Solvers

- Fully automated reasoners with many applications
  - Verification, *Synthesis*, Symbolic Execution, Theorem Proving, Security Analysis

- SMT solver CVC4
  - Open source, available at : https://cvc4.github.io/

- Acknowledgements:
  - Cesare Tinelli, Clark Barrett, Haniel Barbosa, Andres Noetzli, Aina Niemetz, Mathias Preiner
  - Rest of CVC4 development team (past and present)
  - Viktor Kuncak

Synthesis Conjectures

$\exists f . \forall x . P(f, x)$

There exists a function f for which property P holds for all x

Synthesis Conjectures *Modulo T*

$$\exists f . \forall x . \mathbf{P}(f, x)$$

There exists a function f for which property P holds for all x

Property P is in background theory T, e.g. linear arithmetic

$$\exists f . \forall x . f(x+1) \geq f(x)$$

$\Rightarrow$ Satisfiability Modulo Theories (SMT)

Synthesis Conjectures Modulo T

$$\exists f. \forall x. P(f, x)$$

There exists a function f for which property P holds for all x

Syntax-Guided Synthesis Conjectures Modulo T

$\exists f. \forall x. P(f,x)$

spec(f)

There exists a function f for which property P holds for all x

```
A->A+A|-A|x|y|0|1|ite(B,A,A)
B->B∧B|¬B|A=A|A≥A|⊥
```

syntax(f)

The body of f is generated by the above grammar with start symbol A

⇒ *Syntax-guided synthesis "SyGuS"* **[Alur et al 2013]**

Enumerative Cex-Guided Inductive Synthesis (CEGIS)

```
syntax(f):
A->A+A|-A|x|y|0|1|ite(B,A,A)
B->B∧B|¬B|A=A|A≥A|⊥
```

```
spec(f):
∀xy.f(x,y)=f(y,x)+f(0,1)
```

Solution *Enumerator*

Solution Verifier

[Solar-Lezama 2006, Alur et al 2013]

Enumerative Cex-Guided Inductive Synthesis (CEGIS)

syntax(f):
A->A+A|-A|x|y|0|1|ite(B,A,A)
B->B∧B|¬B|A=A|A≥A|⊥

spec(f):
∀xy.f(x,y)=f(y,x)+f(0,1)

Solution *Enumerator*

Solution Verifier

f:=λxy.x?

Enumerative Cex-Guided Inductive Synthesis (CEGIS)

```
syntax(f):
A->A+A|-A|x|y|0|1|ite(B,A,A)
B->B∧B|¬B|A=A|A≥A|⊥
```

```
spec(f):
∀xy.f(x,y)=f(y,x)+f(0,1)
```

Solution *Enumerator* → `f:=λxy.0?` → Solution Verifier

Solution Verifier → `f(2,1)=f(1,2)+f(0,1)` → Solution *Enumerator*

Solution *Enumerator* → `f:=λxy.y?` → Solution Verifier

Solution Verifier → `f(0,1)=f(1,0)+f(0,1)` → Solution *Enumerator*

Solution *Enumerator* → `f:=λxy.x?` → Solution Verifier

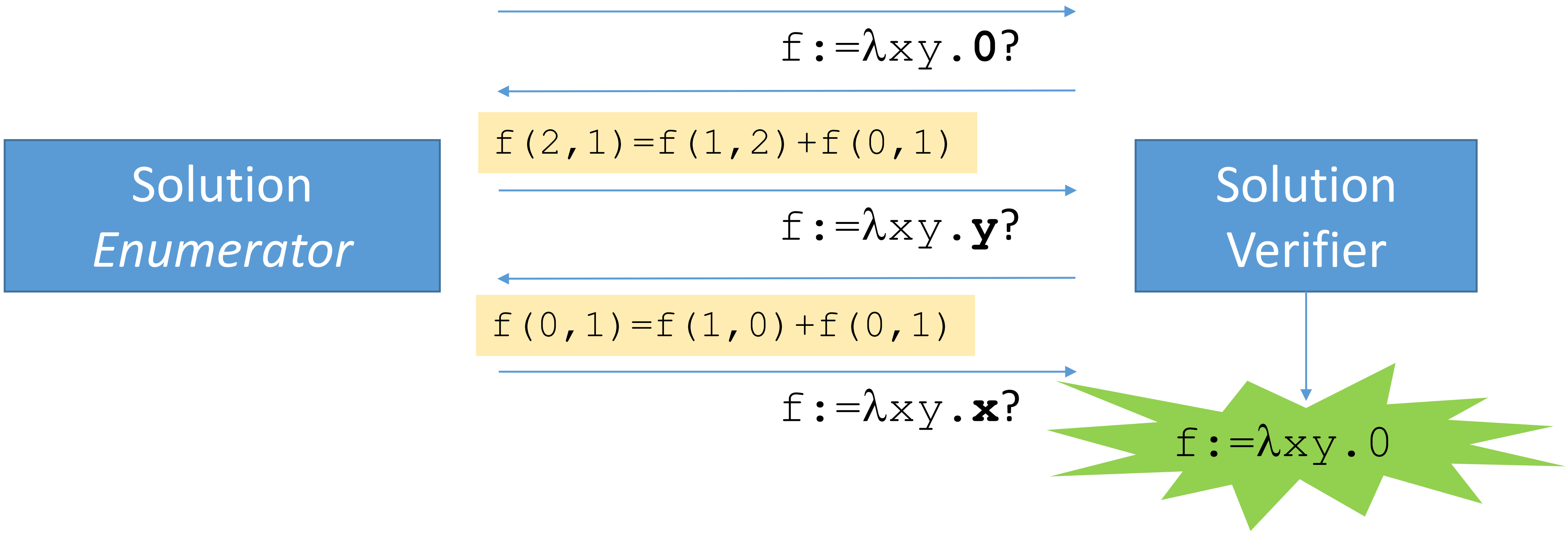
Solution Verifier → `f:=λxy.0`

…new candidate `λxy.0` has no counterexamples wrt **`spec(f)`**

Enumerative Cex-Guided Inductive Synthesis (CEGIS)

syntax(f):
A->A+A|-A|x|y|0|1|ite(B,A,A)
B->B∧B|¬B|A=A|A≥A|⊥

spec(f):
∀xy.f(x,y)=f(y,x)+f(0,1)



⇒ Terms **x**,**y**,0,… are a (fair) enumeration of terms generated by **syntax(f)**

CEGIS using SMT solvers

```
syntax(f):
A->A+A|-A|x|y|0|1|ite(B,A,A)
B->B∧B|¬B|A=A|A≥A|⊥
```

```
spec(f):
∀xy.f(x,y)=f(y,x)+f(0,1)
```

Solution Enumerator

SMT Solver

Solution Verifier

CEGIS inside an SMT solver

```
syntax(f):
A->A+A|-A|x|y|0|1|ite(B,A,A)
B->B∧B|¬B|A=A|A≥A|⊥
```

```
spec(f):
∀xy.f(x,y)=f(y,x)+f(0,1)
```

SMT Solver (CVC4)

[Reynolds et al CAV 2015]

Solution Enumerator

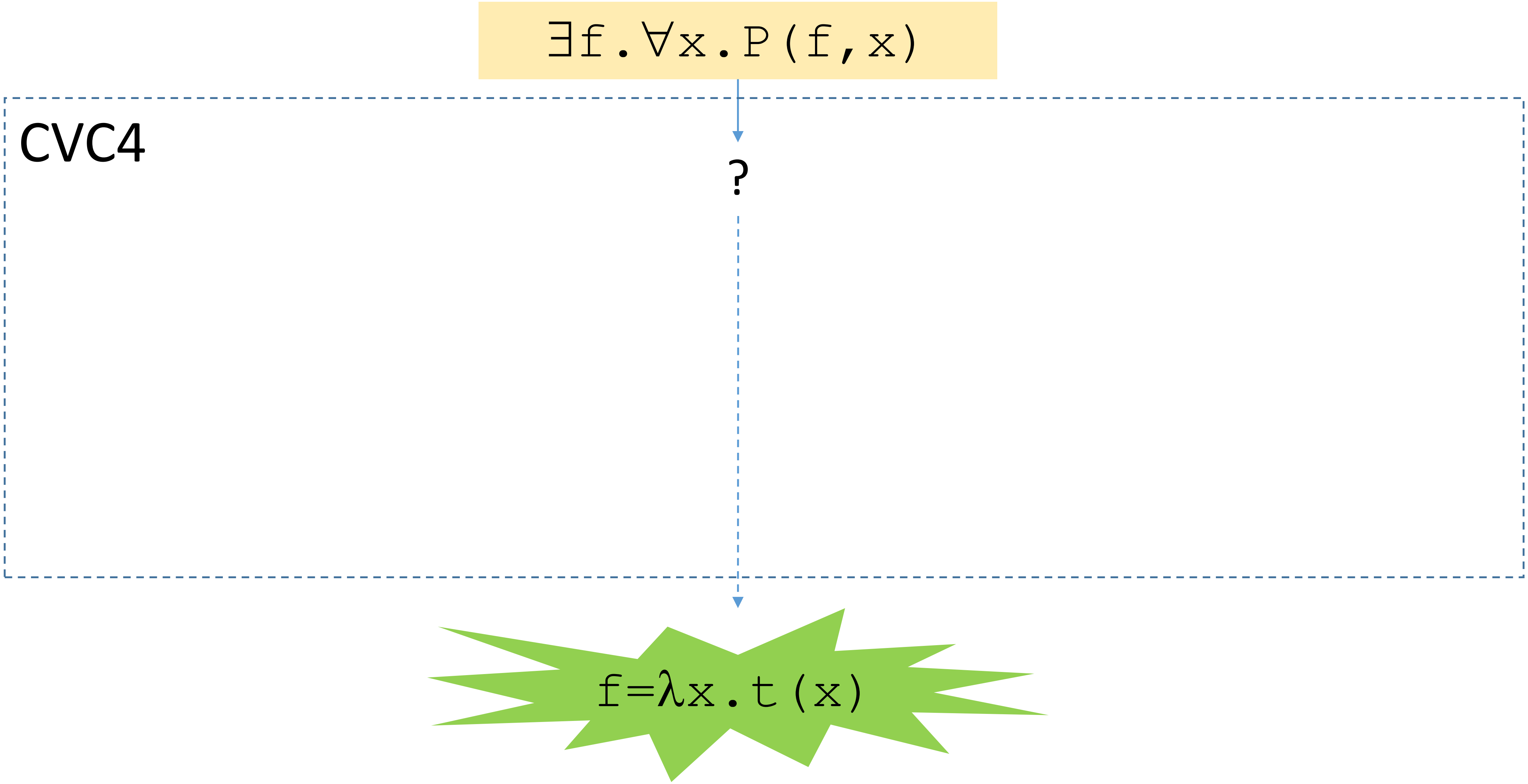
Solution Verifier

- ✓ Synthesis algorithms that use internal state of SMT solver
- ✓ Tight integration between enumerator and verifier

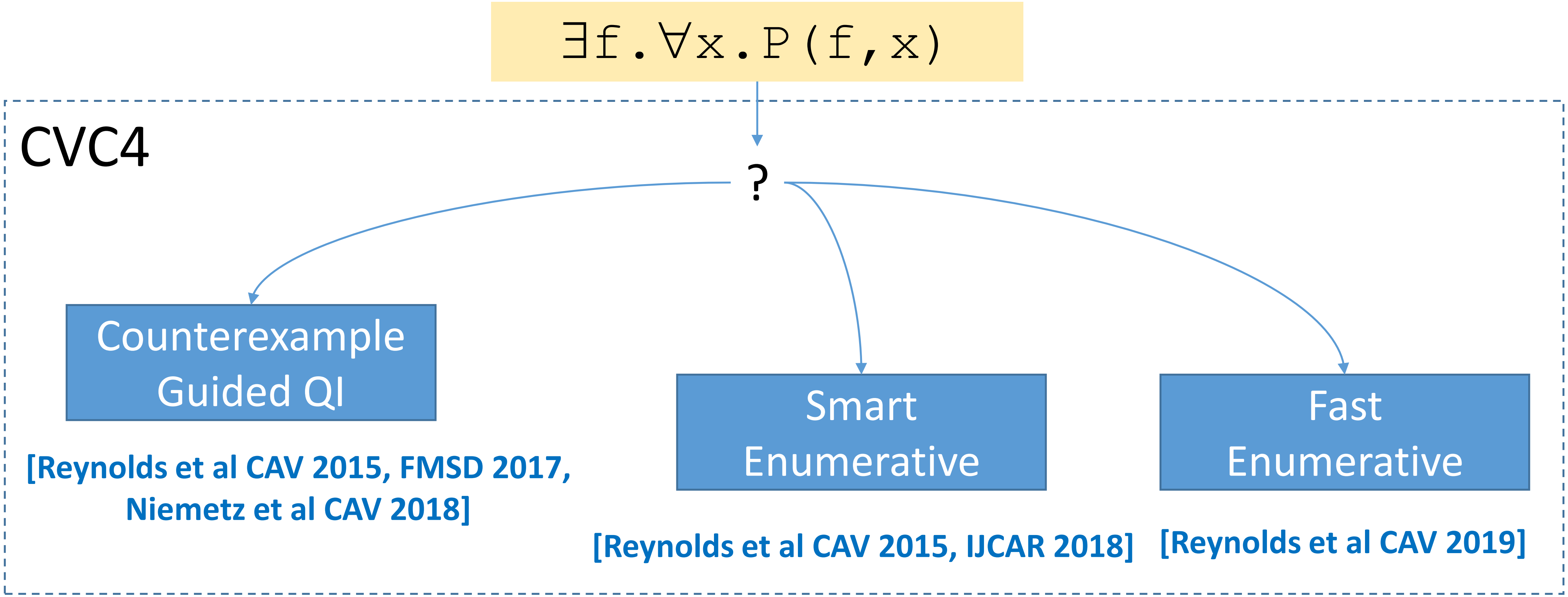
In This Talk

- Synthesis approaches used by SMT+SyGuS solver CVC4:
    1. Counterexample-guided quantifier instantiation
    2. Smart Enumerative SyGuS
    3. Fast Enumerative SyGuS
- Internal applications of SyGuS for SMT solvers
- Future work

Synthesis Solver CVC4



Synthesis Solvers in CVC4



$\Rightarrow$ Best approach to apply depends on the conjecture

Approach #1:
Counterexample-Guided Instantiation

Synthesis via Counterexample-Guided Instantiation

- Some synthesis conjectures are *essentially first-order*:

$$\neg\exists f.\forall xy.\ \mathbf{f(x,y)} \geq x \wedge \mathbf{f(x,y)} \geq y \wedge (\mathbf{f(x,y)} = x \vee \mathbf{f(x,y)} = y)$$

"$\mathbf{f(x,y)}$ is the maximum of x and y"

Synthesis via Counterexample-Guided Instantiation

$$\neg\exists f.\forall xy.\ f(x,y) \geq x \wedge f(x,y) \geq y \wedge (f(x,y) = x \vee f(x,y) = y)$$

$\mathtt{Int} \times \mathtt{Int} \rightarrow \mathtt{Int}$

Anti-skolemize

$$\neg\forall xy.\exists z.\ \mathbf{z} \geq x \wedge \mathbf{z} \geq y \wedge (\mathbf{z} = x \vee \mathbf{z} = y)$$

$\mathtt{Int}$

[Reynolds et al CAV2015]

Synthesis via Counterexample-Guided Instantiation

$$\neg\exists f.\forall xy.\ f(x,y)\geq x \wedge f(x,y)\geq y \wedge (f(x,y)=x \vee f(x,y)=y)$$

$\mathtt{Int} \times \mathtt{Int} \rightarrow \mathtt{Int}$

Anti-skolemize

$$\neg\forall xy.\exists z.\ \mathbf{z} \geq x \wedge \mathbf{z} \geq y \wedge (\mathbf{z} = x \vee \mathbf{z} = y)$$

$\mathtt{Int}$

Solve via first-order ∀ techniques

Counterexample Guided QI

[Reynolds et al CAV2015]

Counterexample-Guided ∀-Instantiation

Quantifier Elimination Procedures

$\Leftarrow(\Rightarrow)^?$

Instantiation-Based procedures for FO $\exists\forall$ formulas

$\Leftarrow\Rightarrow$

Synthesis procedures for single-invocation properties

Counterexample-Guided ∀-Instantiation: Caveats

1. Specification must be *single invocation*
   - e.g. where functions-to-synthesize are applied to the list of universal variables

   ✓ $\exists f. \forall xy. f(x,y) \geq x \land f(x,y) \geq y$

   ✓ $\exists fg. \forall x. f(x) = g(x)$

   🛇 $\exists f. \forall xy. f(x,y) = f(y,x)$

2. If syntax restrictions are present, CEGQI may violate them
   - Heuristic fitting of solution from CEGQI **[Reynolds et al CAV2015]**

3. A term selection strategy must be known for the theory

   ✓ Linear arithmetic, small finite domains, BV, …

   🛇 Strings, non-linear arithmetic, …

Approach #2:

Smart Enumerative SyGuS

Smart Enumerative SyGuS

Conjecture

$\exists f. \forall xy.\ f(x,y) \geq x \wedge f(x,y) = f(y,x)$

Syntactic Restrictions

```
fInt := x | y | 0 | 1 | +(fInt,fInt) |
        ite(fBool,fInt,fInt)
fBool := ≥(fInt,fInt) | =(fInt,fInt)
```

Smart Enumerative SyGuS

Conjecture

$\exists f.\forall xy.\ f(x,y) \geq x \wedge f(x,y) = f(y,x)$

Inductive Datatype

~~Syntactic Restrictions~~

```
fInt := x | y | 0 | 1 | +(fInt,fInt) |
        ite(fBool,fInt,fInt)
fBool := ≥(fInt,fInt) | =(fInt,fInt)
```

View syntactic restrictions as an *inductive datatypes*

Smart Enumerative SyGuS

Conjecture

$\exists \mathbf{f}. \forall xy.\ \mathbf{f}(x,y) \geq x \wedge \mathbf{f}(x,y) = \mathbf{f}(y,x)$

Int × Int → Int

Encode using *deep embedding* involving **fInt**

$\exists \mathbf{d}. \forall xy.\ E(\mathbf{d},x,y) \geq x \wedge E(\mathbf{d},x,y) = E(\mathbf{d},y,x)$

fInt

"Evaluation function" E : **fInt** × Int × Int → Int

Inductive Datatype

fInt := **x** | **y** | **0** | **1** | **+**(fInt,fInt) |
ite(fBool,fInt,fInt)

fBool := **≥**(fInt,fInt) | **=**(fInt,fInt)

Smart Enumerative SyGuS

Conjecture

$\exists f. \forall xy.\ f(x,y) \geq x \wedge f(x,y) = f(y,x)$

$\exists d. \forall xy.\ \mathbf{E}(d,x,y) \geq x \wedge \mathbf{E}(d,x,y) = \mathbf{E}(d,y,x)$

Smart Enumerative SyGuS

Inductive Datatype

```
fInt := x | y | 0 | 1 | +(fInt,fInt) |
        ite(fBool,fInt,fInt)
fBool := ≥(fInt,fInt) | =(fInt,fInt)
```

Solve via datatypes theory solver + CEGIS

Models for d ⇔ candidate solutions

Pruning via Theory Rewriting

Conjecture

$\exists d. \forall xy. E(d,x,y) \geq x \wedge E(d,x,y) = E(d,y,x)$

Inductive Datatype

```
fInt := x | y | 0 | 1 | +(fInt,fInt) |
        ite(fBool,fInt,fInt)
fBool := ≥(fInt,fInt) | =(fInt,fInt)
```

Pruning via Theory Rewriting

Conjecture

$\exists d. \forall xy.\ E(d,x,y) \geq x \wedge E(d,x,y) = E(d,y,x)$

Inductive Datatype

```
fInt  := x | y | 0 | 1 | +(fInt,fInt) |
          ite(fBool,fInt,fInt)
fBool := ≥(fInt,fInt) | =(fInt,fInt)
```

- Solver generates a stream of candidate models:
  - d^M = **x**
  - d^M = **y**
  - d^M = **+(1,y)**
  - d^M = **+(0,x)**
  - d^M = **+(y,1)**
  - …

Pruning via Theory Rewriting

Conjecture

$\exists d. \forall xy. E(d,x,y) \geq x \wedge E(d,x,y) = E(d,y,x)$

Inductive Datatype

```
fInt  := x | y | 0 | 1 | +(fInt,fInt) |
         ite(fBool,fInt,fInt)
fBool := ≥(fInt,fInt) | =(fInt,fInt)
```

- Solver generates a stream of candidate models:
  - d^M = `x`
  - d^M = `y`
  - d^M = `+(1,y)`
  - d^M = `+(0,x)`
  - d^M = `+(y,1)`

Optimization: *Only consider terms* d^M *whose* ***analog*** *is unique up to theory-specific simplification* $\downarrow$

Pruning via Theory Rewriting

Conjecture

$\exists d. \forall xy.\ E(d,x,y) \geq x \wedge E(d,x,y) = E(d,y,x)$

Inductive Datatype

fInt := **x** | **y** | **0** | **1** | **+**(fInt,fInt) |
ite(fBool,fInt,fInt)
fBool := **≥**(fInt,fInt) | **=**(fInt,fInt)

- Solver generates a stream of candidate models, normalizes values $\downarrow$:
  - d^M = **x** … x $=\downarrow$ x
  - d^M = **y** … y $=\downarrow$ y
  - d^M = **+(1,y)** … $1+y$ $=\downarrow$ $y+1$
  - d^M = **+(0,x)** … $0+x$ $=\downarrow$ x
  - d^M = **+(y,1)** … $y+1$ $=\downarrow$ $y+1$

Pruning via Theory Rewriting

Conjecture

$\exists d. \forall xy.\ E(d,x,y) \geq x \wedge E(d,x,y) = E(d,y,x)$

Inductive Datatype

```
fInt := x | y | 0 | 1 | +(fInt,fInt) |
        ite(fBool,fInt,fInt)
fBool := ≥(fInt,fInt) | =(fInt,fInt)
```

- Solver generates a stream of candidate models, normalizes values ↓:
  - d^M = **x** … x =↓ x
  - d^M = **y** … y =↓ y
  - d^M = **+(1,y)** … 1+y =↓ y+1
  - d^M = ~~**+(0,x)**~~ … 0+x =↓ x
  - d^M = ~~**+(y,1)**~~ … y+1 =↓ y+1

Avoid candidate solutions not unique up to theory normalization

Syntactic Constraints in Smart Enumerative SyGuS

$$\neg is_{+}(d) \vee \neg is_{x}(d.1) \vee \neg is_{0}(d.2)$$

"Do not consider solutions where d is +(x,0)"

- Encoding uses *shared selectors* of the form d.1
  - Agnostic to constructor of d **[Reynolds et al IJCAR 2018]**

- Syntactic constraints can be generalized:

$$\neg is_{+}(d) \vee \neg is_{0}(d.2)$$

"Do not consider solutions where d is of the form +(t,0) for any t"

⇒ Leads to stronger search space pruning **[Reynolds et al CAV 2019]**

(Partial) Evaluation Unfolding

$$\texttt{is}_{\texttt{ite}}(\texttt{d}) \Rightarrow \texttt{E(d,x,y)=ite(E(d.1,x,y),E(d.2,x,y),E(d.3,x,y))}$$

"When the top symbol of `d` is **`ite`**, its evaluation behaves like if-then-else"

$$\texttt{is}_{+}(\texttt{d}) \wedge \texttt{is}_{\texttt{x}}(\texttt{d.1}) \wedge \texttt{is}_{\texttt{1}}(\texttt{d.2}) \Rightarrow \texttt{E(d,x,y)=x+1}$$

"When `d` has value **`x+1`**, its evaluation is equal to `x+1`"

- Evaluation unfolding lemmas connect evaluation symbols `E` to theory
- Implementation combines partial and total unfolding
  - Boolean connectives and ITE use partial unfolding, others use total

Approach #3:
Fast Enumerative SyGuS

Fast Enumerative SyGuS

$\exists$f.$\forall$x. Ψ

Fast Enumerative SyGuS

FASTENUM(τ, k):

For all:

- Constructor classes $C \in \mathcal{C}_\tau$, whose elements have type $\tau_1 \times \ldots \times \tau_n \to \tau$,
- Tuple of naturals $(k_1, \ldots k_n)$ such that $k_1 + \ldots + k_n + \text{ite}(n > 0, 1, 0) = k$,

(a) Run FASTENUM(τ_i, k_i) for each $i = 1, \ldots, n$,

(b) Add $\mathsf{C}(t_1, \ldots, t_n)$ to S_τ^k for all tuples $(t_1, \ldots, t_n)$ with $t_i \in S_{\tau_i}^{k_i}$ and all constructors $\mathsf{C} \in C$.

```
(sygus-enum 0)
(sygus-candidate (max 0))
(sygus-enum 0)
(sygus-enum 1)
(sygus-enum x)
(sygus-enum x)
(sygus-candidate (max x))
(sygus-enum x)
(sygus-enum y)
(sygus-enum y)
(sygus-candidate (max y))
(sygus-enum y)
(sygus-enum (+ x x))
(sygus-enum (+ x 1))
(sygus-enum (+ 1 1))
(sygus-enum (+ 1 y))
(sygus-enum (+ y y))
(sygus-enum (+ y x))
(sygus-enum (- 1 x))
(sygus-enum (- 1 y))
(sygus-enum (- y 1))
(sygus-enum (- 0 1))
(sygus-enum (- x 1))
(sygus-enum (- y x))
(sygus-enum (- 0 y))
(sygus-enum (- x y))
(sygus-enum (- 0 x))
(sygus-enum (+ (+ x x) 1))
(sygus-enum (+ (+ x 1) 1))
```

- Directly enumerate terms based on custom iterator data structures

Fast Enumerative SyGuS (vs. Smart)

PROS:

- Can use (basic) theory rewriting to prune redundant terms
- Very fast
  - Roughly 100x term throughput w.r.t smart enumeration

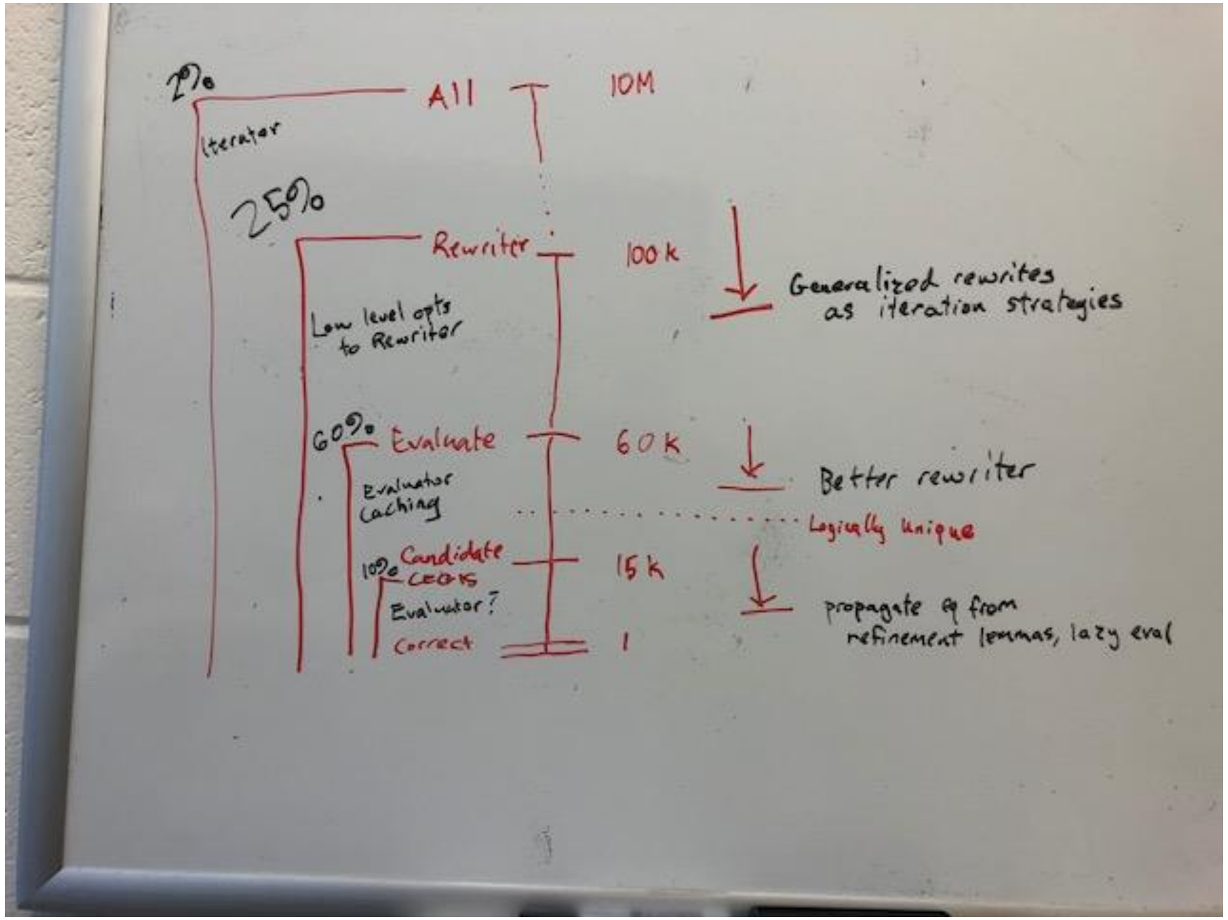
CONS:

- Cannot easily generalize syntactic constraints
  - Thus, more advanced pruning techniques are not (easily) applicable

SUMMARY:

- Fast is usually better, smart is required for harder conjectures **[Reynolds et al CAV 2019]**

Profiling Fast Enumerative SyGuS



- Evaluating terms on concrete examples is the bottleneck (~70% of runtime)

Summary of Solvers

- If Ψ is single invocation, no grammar restrictions, has theory QI
  (#1) Use *counterexample-guided quantifier instantiation*
- Else:
  - If Ψ has multiple function-to-synthesize or grammar with Bool connectives
    (#2) Use *smart enumerative SyGuS*
  - Else:
    (#3) Use *fast enumerative SyGuS*

Solvers are Supplemented with Additional Techniques

- For CEGQI:
  - Partial quantifier elimination as a preprocessing pass
  - Heuristic solution reconstruction
- For enumerative:
  - Divide-and-conquer **[Alur et al 2017]**
  - Piecewise-Independent Unification (UNIF+PI) **[Barbosa et al FMCAD2019]**
  - Theory-specific constant repair **[Abate et al 2019]**
  - Static grammar minimization and symmetry breaking
  - Variable agnostic enumeration

Ongoing work

- Internal use of SyGuS *for improving the SMT solver*
  - For designing QI algorithms **[Niemetz et al CAV 2018, Brain et al CAV2019]**
  - Discovering rewrite rules **[Noetzli et al SAT 2019]**
  - User-guided test case generation
  - Quantifier instantiation via enumerative SyGuS **[Niemetz et al TACAS 2021]**
- Algorithms that utilize enumerative *SyGuS as a black box*
  - Invariant synthesis **[Barbosa et al FMCAD 2019]**
  - Abduction **[Reynolds et al IJCAR 2020]**
  - Interpolation
  - Optimization
- Low-level optimizations

Thanks!

- SyGuS techniques in talk available in SMT solver CVC4(…5)
  - Open-source : https://cvc4.github.io/
  - Includes Python and C++ APIs for SyGuS
  - Java API coming soon

- Questions?