Towards Enumerative Invariant Synthesis in SMT Solvers

Andrew Reynolds

NSV workshop

July 18, 2018

THE UNIVERSITY OF IOWA

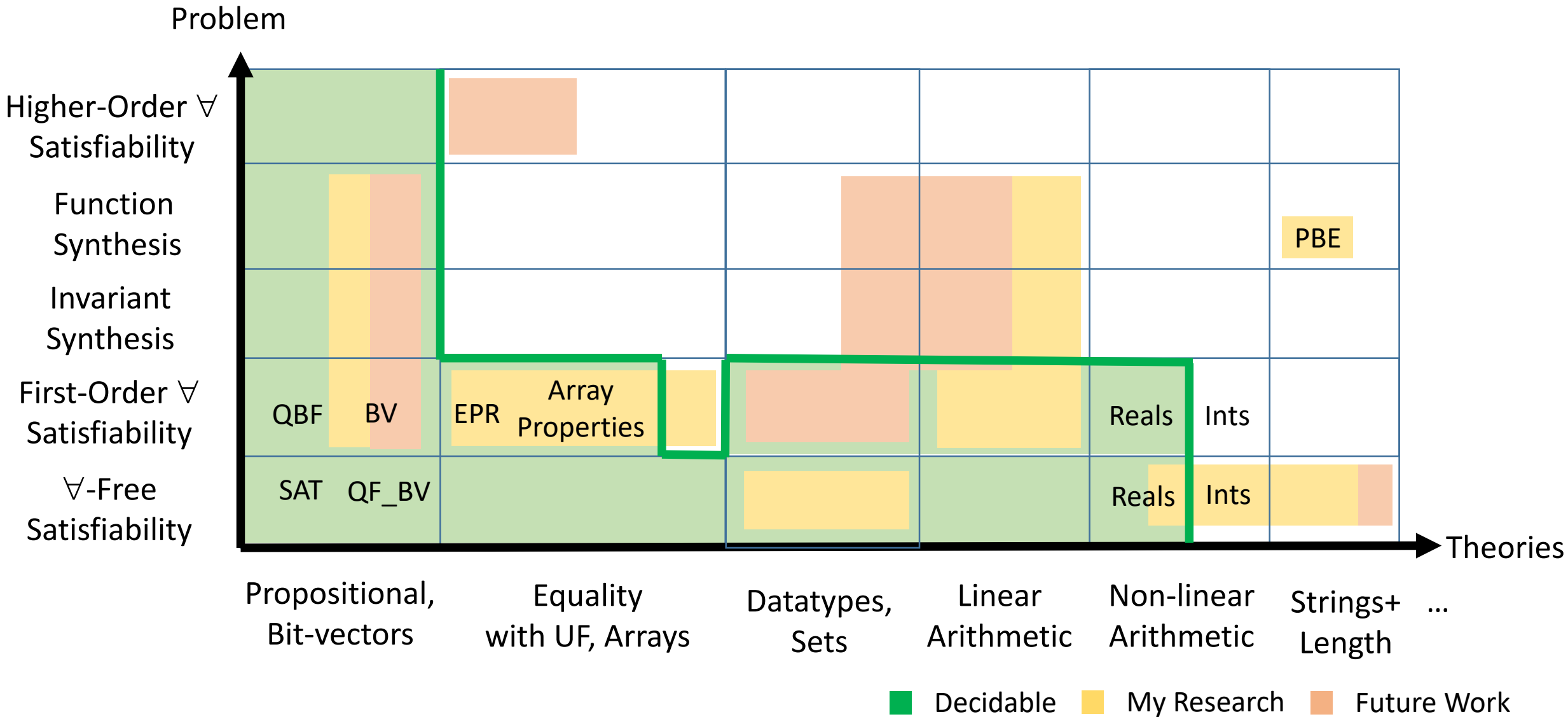
My Research

- Satisfiability Modulo Theories (SMT) Solvers
  - Fully automated reasoners with many applications
    - Verification, Synthesis, Symbolic Execution, Theorem Proving

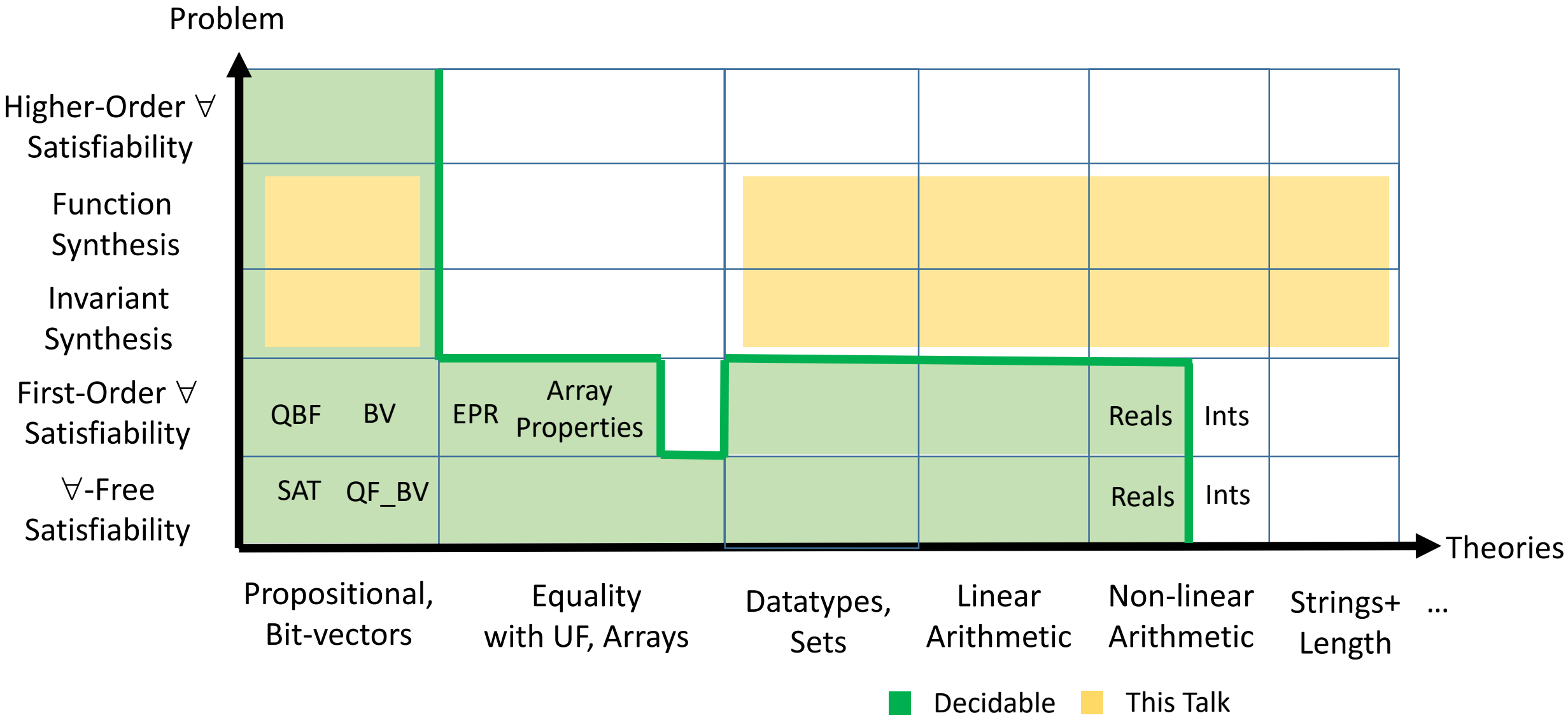
- Development of the SMT solver CVC4
  - Open source
  - Available at : http://cvc4.cs.stanford.edu/web/

- Acknowledgements:
  - Cesare Tinelli, Clark Barrett, CVC4 development team (past and present)

My Research in SMT



This Talk: Synthesis for (Interpreted) Theories



Synthesis Modulo Theories, inside an SMT solver

- Using a combination of:

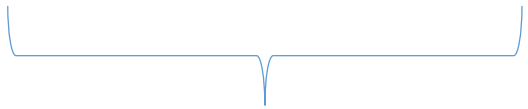
  Syntax-guided enumeration …

  Unification Algorithms …

  Decision Procedures

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 some background theory T, e.g.

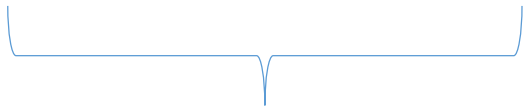
- Linear (or non-linear) arithmetic
- Fixed-Width BitVectors
- Strings
- …

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

$\Rightarrow$ 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

Synthesis Conjectures Modulo T

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

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

$$A \to A+A \mid -A \mid x \mid y \mid 0 \mid 1 \mid ite(B,A,A)$$
$$B \to B \wedge B \mid \neg B \mid A=A \mid A \geq A \mid \bot$$

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

Invariant Synthesis Conjectures Modulo T

$$\exists I.\forall xx'.(pre(x)\Rightarrow I(x))\wedge((I(x)\wedge Tr(x,x'))\Rightarrow I(x'))\wedge(I(x)\Rightarrow post(x))$$

There exists a predicate `I` that is implied by `pre`, implies `post`,
and is inductive wrt relation `Tr`

$$B \to B\wedge B \mid \neg B \mid A=A \mid A\geq A \mid \bot$$
$$A \to A+A \mid -A \mid x \mid y \mid 0 \mid 1$$

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

Many approaches to Synthesis

`synth(` `∃f.∀x.P(f,x)` `)` returns `f := λx.t`

Targeted towards specific theories ⇔ Generic

Incomplete ⇔ “Relatively” complete

Many approaches to Synthesis

`synth(` `∃f.∀x.P(f,x)` `)` returns `f := λx.t`

Targeted towards specific theories ⇔ **Generic**

If theory T has a decision procedure, then there exists a synthesis procedure `synth` for second-order T-conjectures

Incomplete ⇔ **“Relatively” complete**

If the above conjecture has a (finite) solution, procedure `synth` will eventually find it.

Counterexample-Guided Inductive Synthesis (CEGIS)

```
spec(f):
∀xy.f(x,y)=f(y,x)
```

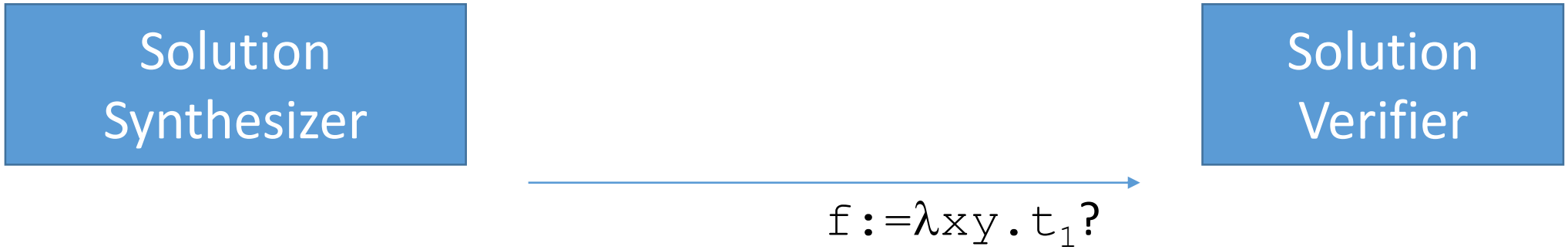
Solution Synthesizer

Solution Verifier

[Solar-Lezama 2006, Alur et al 2013]

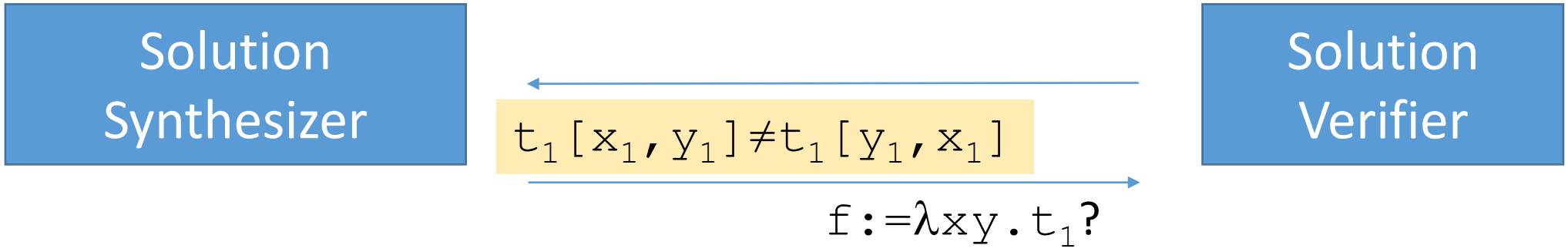
Counterexample-Guided Inductive Synthesis (CEGIS)

spec(f):
$\forall xy.f(x,y)=f(y,x)$

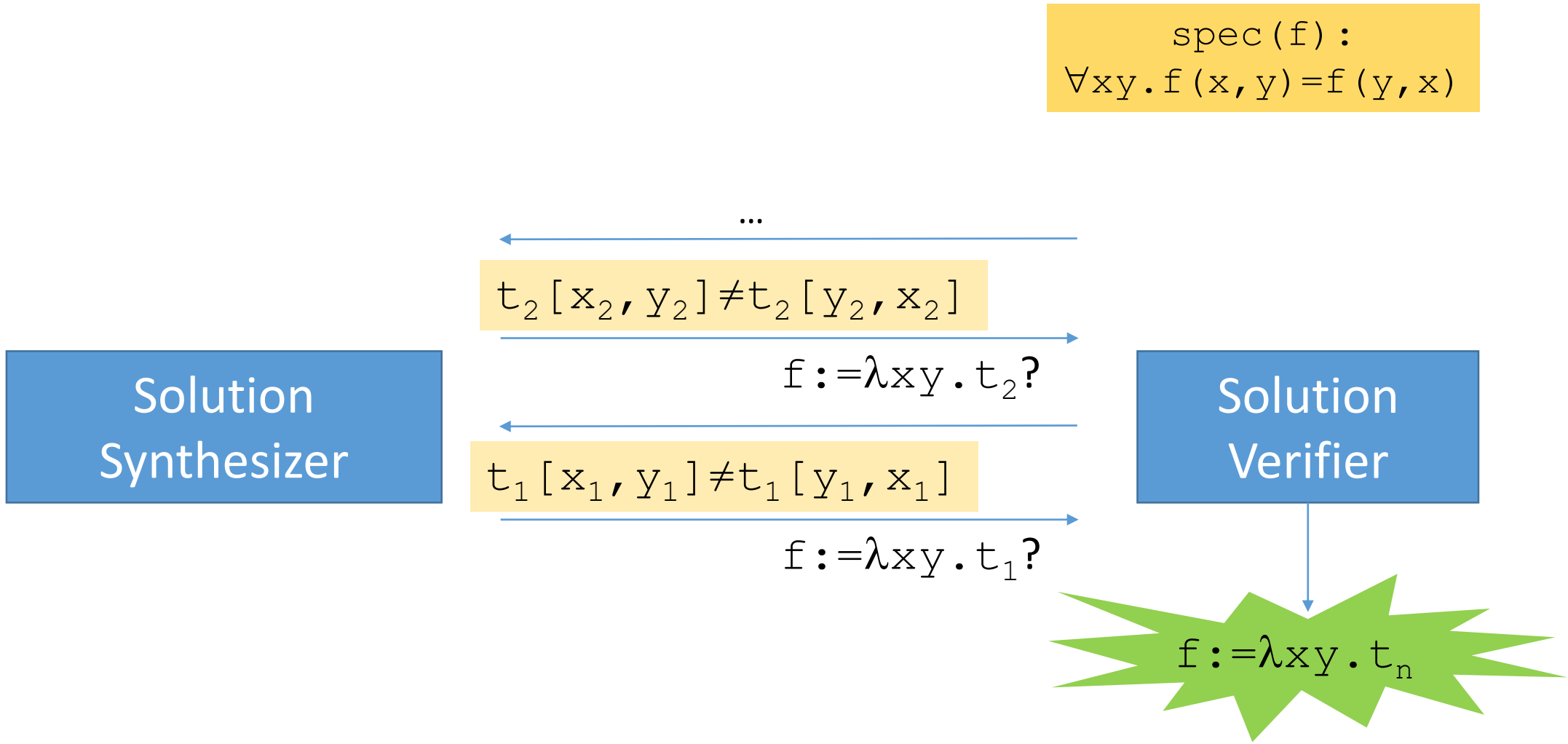


Counterexample-Guided Inductive Synthesis (CEGIS)

spec(f):
$\forall xy.f(x,y)=f(y,x)$



Counterexample-Guided Inductive Synthesis (CEGIS)



Enumerative 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)
```

Solution
Enumerator

Solution
Verifier

Enumerative 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)
```

…

Solution Enumerator

$f := \lambda xy.t_2$?

$f := \lambda xy.t_1$?

Solution Verifier

⇒ Terms t1,t2,t3,… are a (fair) enumeration of terms generated by `syntax(f)`

Enumerative CEGIS + Unification

```
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)
```

Syntax-Guided Enumeration

Solution Verifier

Unification Algorithm

[Alur/Radhakrishna/Udupa et al 2017]

Enumerative CEGIS + Unification

```
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)
```

Syntax-Guided Enumeration

$t_1, t_2, t_3, \ldots, t_n$

Unification Algorithm

Solution Verifier

$f := \lambda xy.u[\ldots, t_i, \ldots, t_j, \ldots]$?

[Alur/Radhakrishna/Udupa et al 2017]

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|⊥
```

Syntax-Guided Enumeration

Unification Algorithm

```
spec(f):
∀xy.f(x,y)=f(y,x)
```

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)
```

SMT Solver
(CVC4)

[Reynolds et al CAV 2015]

Syntax-Guided Enumeration

Unification Algorithm

Solution Verifier

Enumerate (Invariant) Synthesis in SMT Solvers

`synth(`$\exists$`f.`$\forall$`x.P(f,x))` **using:**

1. **Syntax-guided enumeration**
   - Construct a set of "interesting" terms $\{t_1 \ldots t_n\}$
2. **Unification algorithms**
   - From $\{t_1 \ldots t_n\}$, construct candidate u for f
3. **(Decision) Procedures**
   - Check whether c_f satisfies the specification
     - Is $\neg\forall x.P(u,x)$ T-unsatisfiable? $\Rightarrow \neg P(u,k)$, quantifier-free SMT query

Syntax-Guided Enumeration

Syntax-Guided Enumeration for T

- Given grammar:

$$A \to A+A \mid -A \mid x \mid y \mid 0 \mid 1 \mid ite(B,A,A)$$
$$B \to B \wedge B \mid \neg B \mid A=A \mid A \geq A \mid \bot$$

- Goal:
  - Efficiently enumerate the set of useful/interesting terms
    - $0, 1, x, y, 1+1, x+x, y+y, 1+x, 1+y, x+y, \ldots$

- Use SMT solver to explore the search space of terms in a grammar
  - Deep embedding to inductive datatypes **[Reynolds et al CAV2015]**
  - Efficient encodings using shared selectors **[Reynolds et al IJCAR2018]**

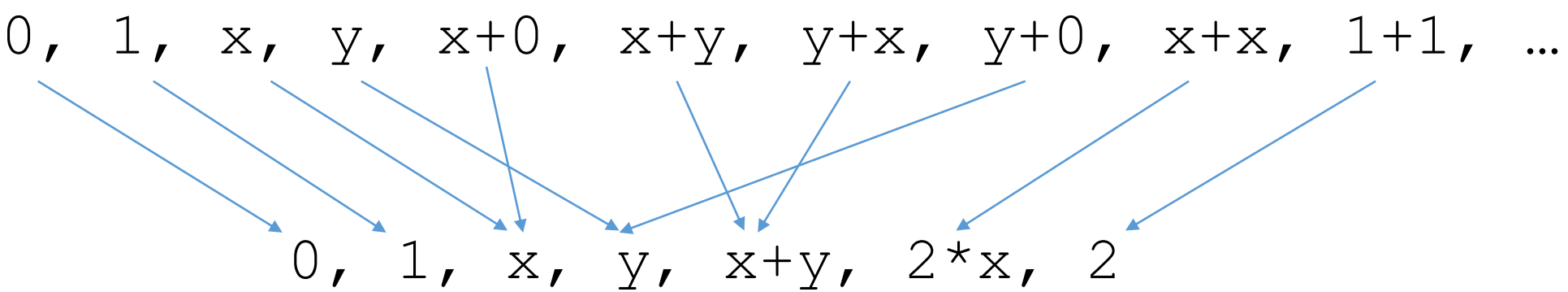
Syntax-Guided Enumeration for T

- How do we determine which terms are (not) interesting?

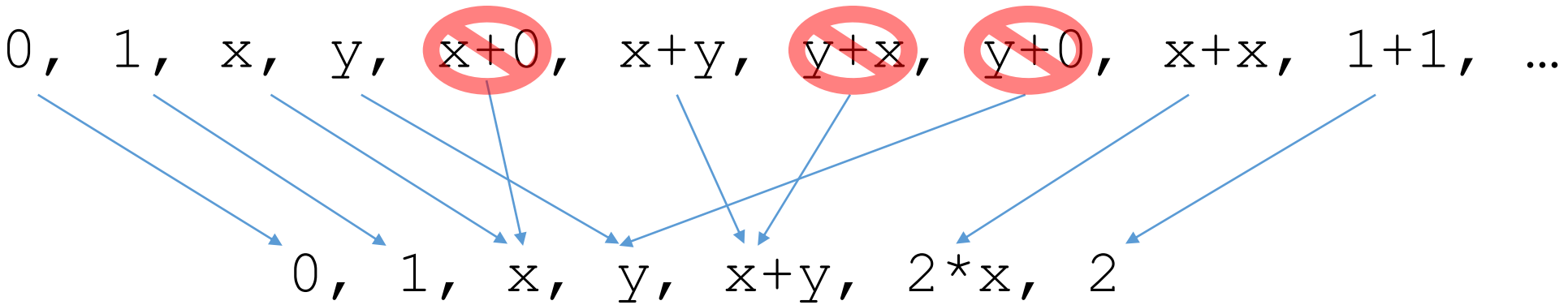
```
0, 1, x, y, x+0, x+y, y+x, y+0, x+x, 1+1, …
```

Syntax-Guided Enumeration for T

- How do we determine which terms are (not) interesting?
  - When enumerating, map terms to their *rewritten form* :



Syntax-Guided Enumeration for T



- Discard all but one term for each equivalence class of terms

Syntax-Guided Enumeration for T

- Gives us a stream of terms that are unique up theory rewriting:

```
0, 1, x, y,        x+y,                    x+x, 1+1, …
```

⇒Requires: Highly aggressive *rewriting techniques* for theories T

- Strings, BitVectors, Floating Points

Examples: Strings, Bit Vectors, Booleans

```
(synth-fun f
 ((x String) (y String) (z Int))
 String (
 (Start String (
  x y "A" "B" ""
  (str.++ Start Start)
  (str.replace Start Start Start)
  (str.at Start ie)
  (int.to.str ie)
  (str.substr Start ie ie)))
 (ie Int (
  0 1 z
  (+ ie ie)
  (- ie ie)
  (str.len Start)
  (str.to.int Start)
  (str.indexof Start Start ie)))))
```

```
(synth-fun f ((s (BitVec 4))
              (t (BitVec 4)))
 (BitVec 4) (
 (Start (BitVec 4) (
  s t #x0
  (bvneg  Start)
  (bvnot  Start)
  (bvadd  Start Start)
  (bvmul  Start Start)
  (bvand  Start Start)
  (bvlshr Start Start)
  (bvor   Start Start)
  (bvshl  Start Start)))))
```

```
(synth-fun f
 ((x Bool) (y Bool)
  (z Bool) (w Bool))
 Bool (
 (Start Bool (
  (and d1 d1) (not d1)
  (or d1 d1) (xor d1 d1)))
 (d1 Bool (
  x (and d2 d2) (not d2)
  (or d2 d2) (xor d2 d2)))
 (d2 Bool (
  w (and d3 d3) (not d3)
  (or d3 d3) (xor d3 d3)))
 (d3 Bool (
  y (and d4 d4) (not d4)
  (or d4 d4) (xor d4 d4)))
 (d4 Bool (z))))
```

Aggressive Rewriting for Theories

- Bit-Vectors

  `bvlshr(x,x)` → `#x0000` `x-(x&y)` → `x&~y` `concat(#x1,x)=concat(#x0,y)` → ⊥

  `x+1` → `~(-x)` `(x&y)+(x|y)` → `x+y` `bvxor(x,x&y)` → `~y&x`

- Strings

  `x++"A"="B"++x` → ⊥ `indexof("ABCDE",x,3)` → `indexof("AAADE",x,3)`

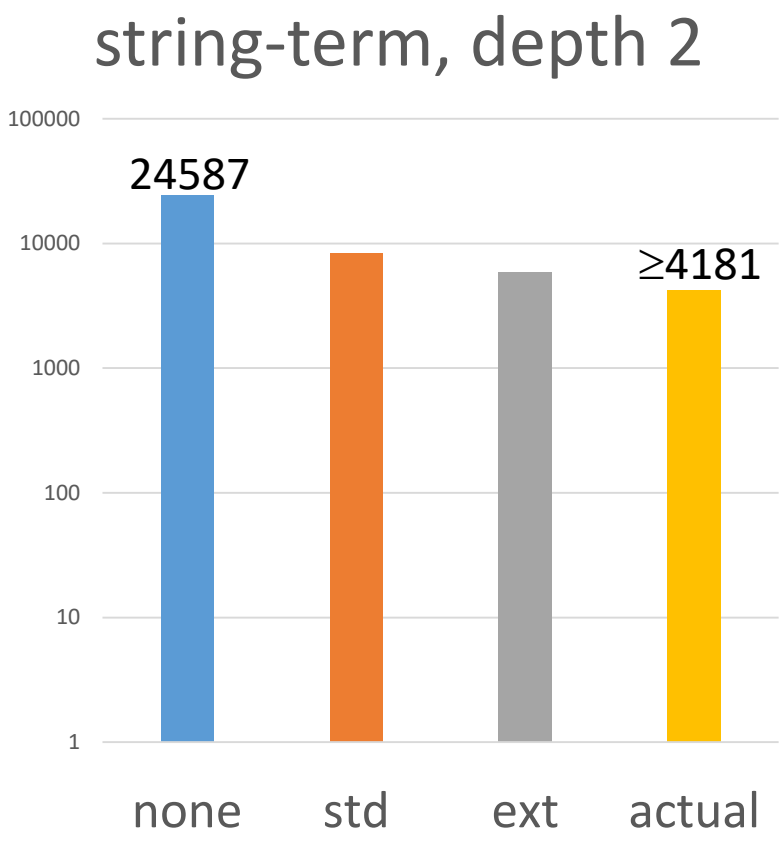
  `contains(x,x++"A")` → ⊥ `replace(x,x++y,y)` → `replace(x,x++y,"")`

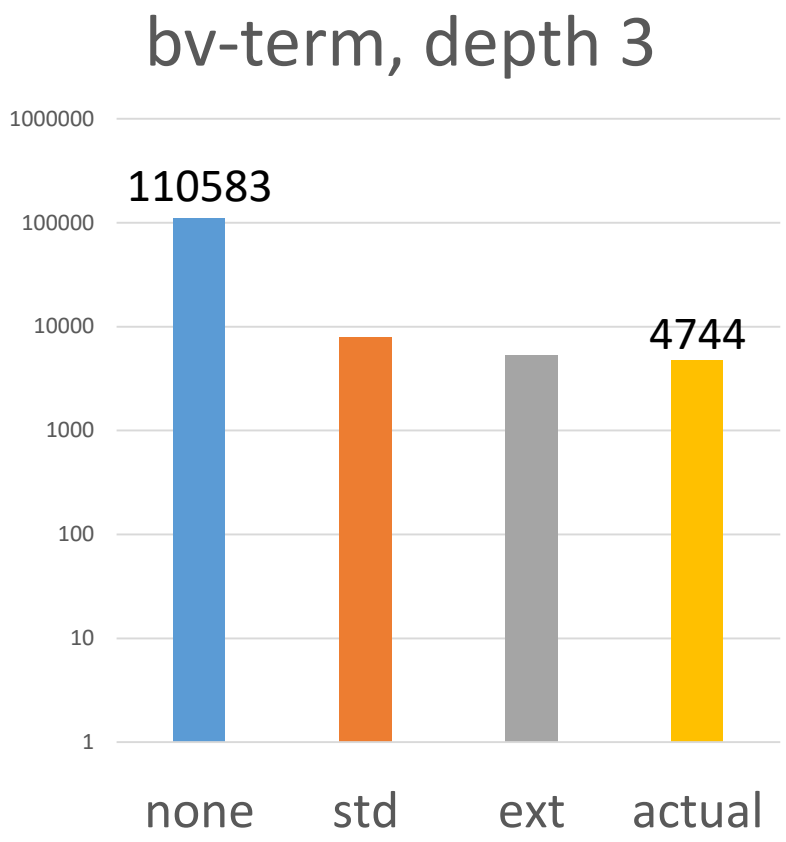
- Booleans

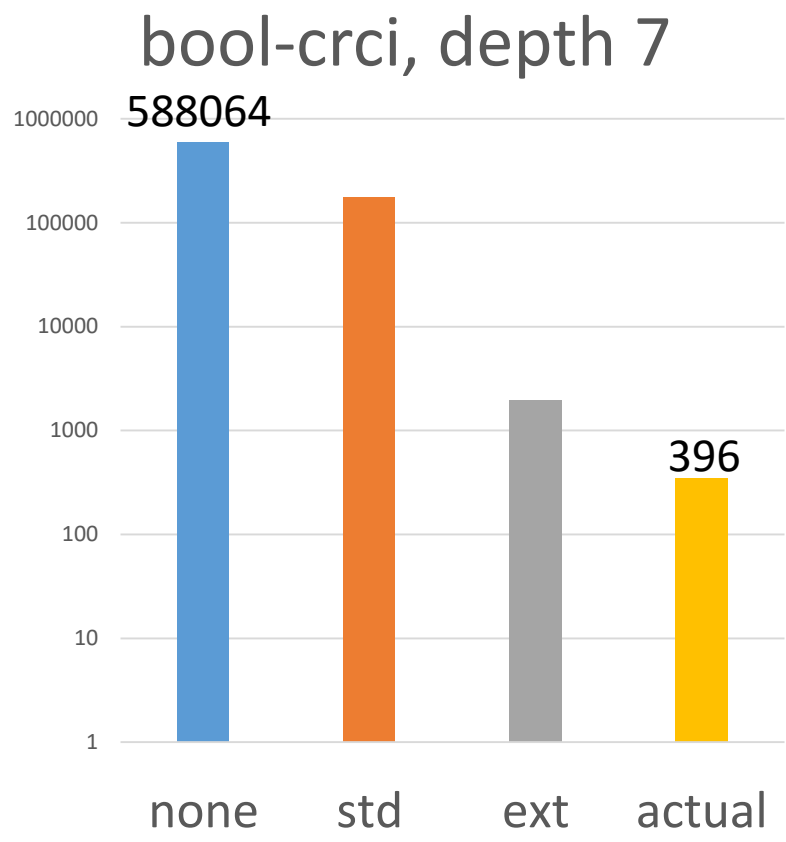
  $A \wedge (A \vee B) \rightarrow A \wedge B$ $(A \vee C) \wedge (A \vee B) \rightarrow A \wedge (C \vee B)$

  $A = A \,\&\, B \rightarrow \neg A \vee B$ $(A \vee B) = (A \vee B \vee C) \rightarrow A \vee B \vee \neg C$

Statistics: CVC4's Current Rewriter(s)







none: # terms from the grammar at given depth

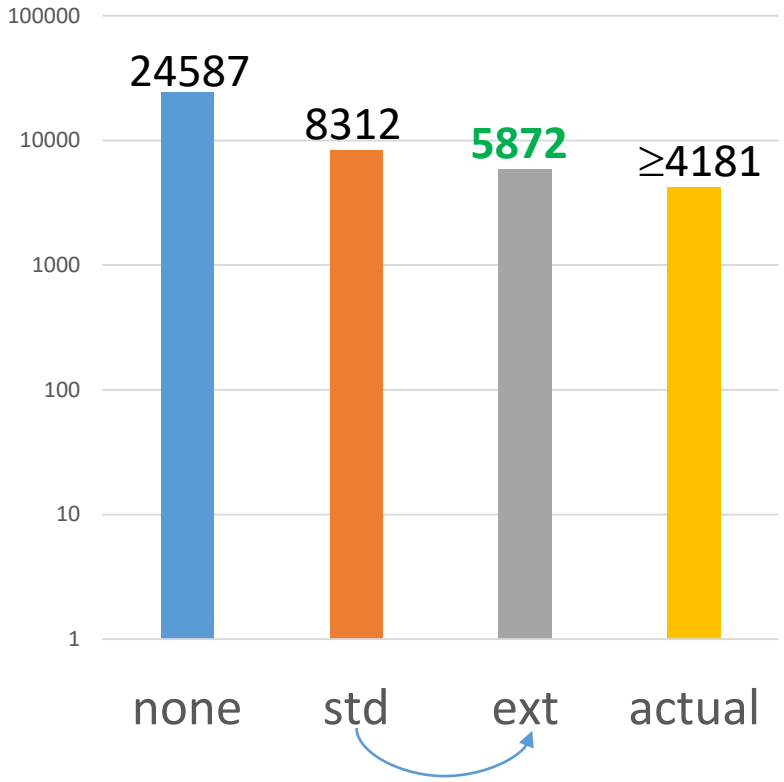
actual: # T-unique terms from grammar at given depth

std: CVC4 version 1.5's rewriter (1 year ago)

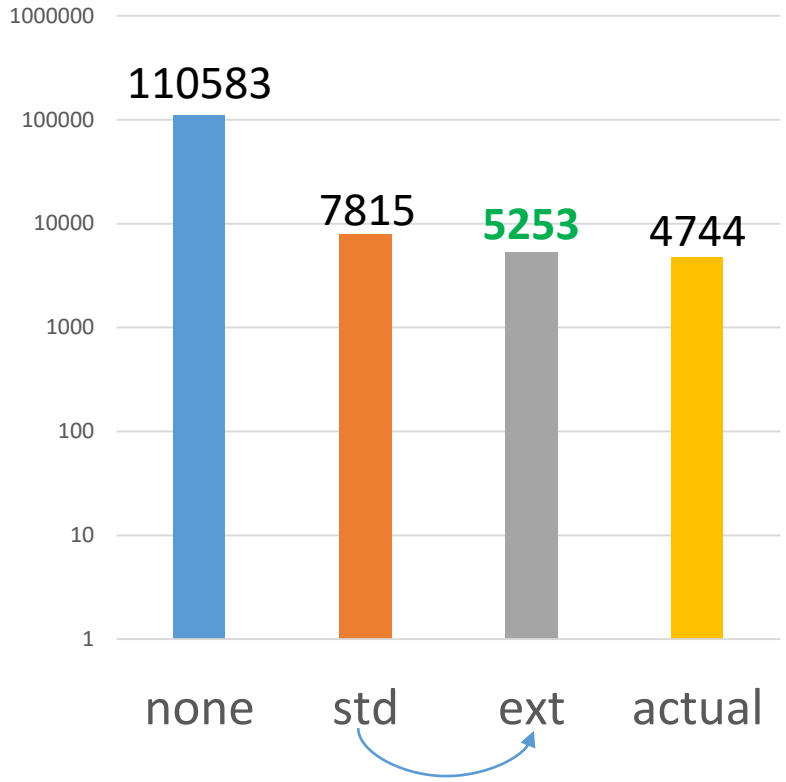
ext: CVC4's aggressive rewriter

Statistics: CVC4's Current Rewriter(s)

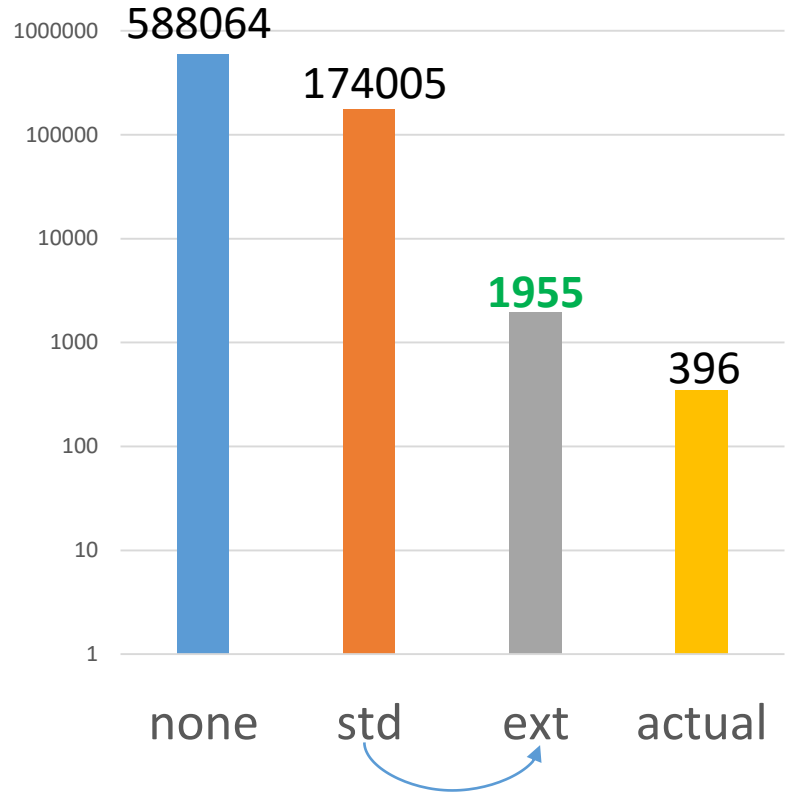
string-term, depth 2



bv-term, depth 3



bool-crci, depth 7



%redundant:	49.7%	**28.8%**	39.3%	**9.7%**	99.8%	**82.2%**
time to enumerate:	90.0	**60.8**	96.4	**55.4**	19128.8	**60.6**

Conjecture-Specific Enumeration

- Given grammar:

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

- Efficiently enumerate the set of useful/interesting terms
  - `0,1,x,y,1+1,x+x,y+y,1+x,1+y,x+y,…`

Conjecture-Specific Enumeration

- Given grammar *and specification*:

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

+ `∃f.∀xy.P(f,xy)`

- Efficiently enumerate the set of useful/interesting terms
  - `0,1,x,~~y~~,1+1,x+x,~~y+y~~,1+x,1+~~y~~,x+~~y~~,…`

E.g. “if there exists a solution for f, for this conjecture, then there exists one that does not involve y or constants greater than one”

⇒ **Even more aggressive pruning**

Unification Algorithms

Does this scale?

```
(synth-fun f ((s (BitVec 4))
              (t (BitVec 4)))
 (BitVec 4) (
 (Start (BitVec 4) (
  s t #x0
  (bvneg  Start)
  (bvnot  Start)
  (bvadd  Start Start)
  (bvmul  Start Start)
  (bvand  Start Start)
  (bvlshr Start Start)
  (bvor   Start Start)
  (bvshl  Start Start)))))
```

- For this Bit-vector grammar, enumerating:
  - Terms of size=1 → **.05 seconds**
  - Terms of size=2 → **.6 seconds**
  - Terms of size=3 → **48 seconds**
  - Terms of size=4 → **5.8 hours** (5.8 million functions, 84050 unique)
  - Terms of size=5 → ??? 100+ days

Example: Programming by Examples (PBE)

```
A->0|1|x|(bvnot A)|
(shl1 A)|(shr1 A)|(shr4 A)|(shr16 A)|
(bvand A A)|(bvor A A)|(bvxor A A)|
(bvadd A A)|(if0 A A A)
```

```
∃f.
f(#x28085a970e13e12c)=#x28085a970e13e12d
f(#xbe5341bebd2a0749)=#xbe5341bebd2a0749
f(#xe239460eed2cc34e)=#xe239460eed2cc34f
….
f(#x4c5ee4be98c5ee7d)=#x4c5ee4be98c5ee7d
f(#xcd67bd5beaac575e)=#xcd67bd5beaac575e
```

Specification is concrete input/output points

Example: Programming by Examples (PBE)

```
A->0|1|x|(bvnot A)|
(shl1 A)|(shr1 A)|(shr4 A)|(shr16 A)|
(bvand A A)|(bvor A A)|(bvxor A A)|
(bvadd A A)|(if0 A A A)
```

```
∃f.
f(#x28085a970e13e12c)=#x28085a970e13e12d
f(#xbe5341bebd2a0749)=#xbe5341bebd2a0749
f(#xe239460eed2cc34e)=#xe239460eed2cc34f
….
f(#x4c5ee4be98c5ee7d)=#x4c5ee4be98c5ee7d
f(#xcd67bd5beaac575e)=#xcd67bd5beaac575e
```

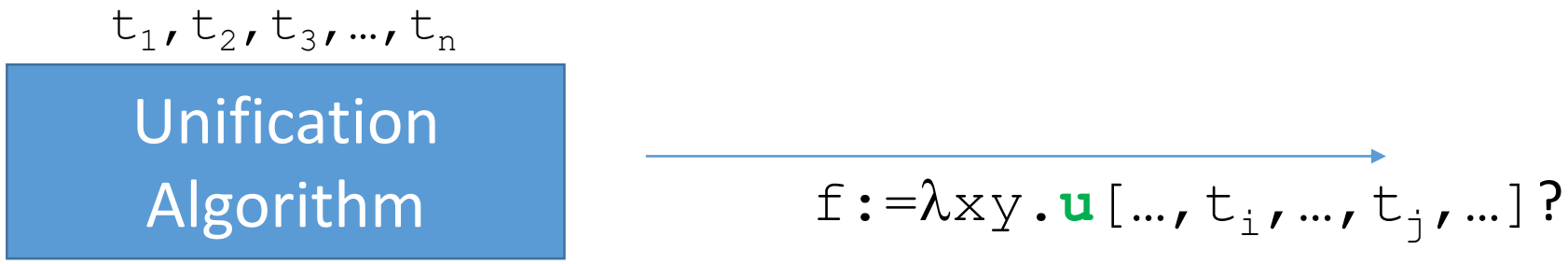
```
f := λxy.
(if0 (bvand (bvnot x) #x0000000000000001)
(if0 (bvand (bvnot (shr4 x)) #x0000000000000001)
(if0 (bvand (shr1 (shr16 x)) #x0000000000000001)
(if0 (bvand (bvnot (shr1 x)) #x0000000000000001) (bvor #x0000000000000001 x) x)
(if0 (bvand (bvnot (shr1 x)) #x0000000000000001) x
(if0 (bvand (shr16 x) #x0000000000000001) x (bvor #x0000000000000001 x))))
(if0 (bvand (shr1 (shr16 x)) #x0000000000000001) (bvor #x0000000000000001 x)
x)) (bvor #x0000000000000001 x)))
```

size 29

⇒ No chance to enumerate this solution by fair enumeration

Unification Algorithms

- Idea: use a high level strategy to build solutions using a pool of enumerated terms



- Two examples:
  - ITE/decision tree learning
  - String concatenation

Unification for ITE

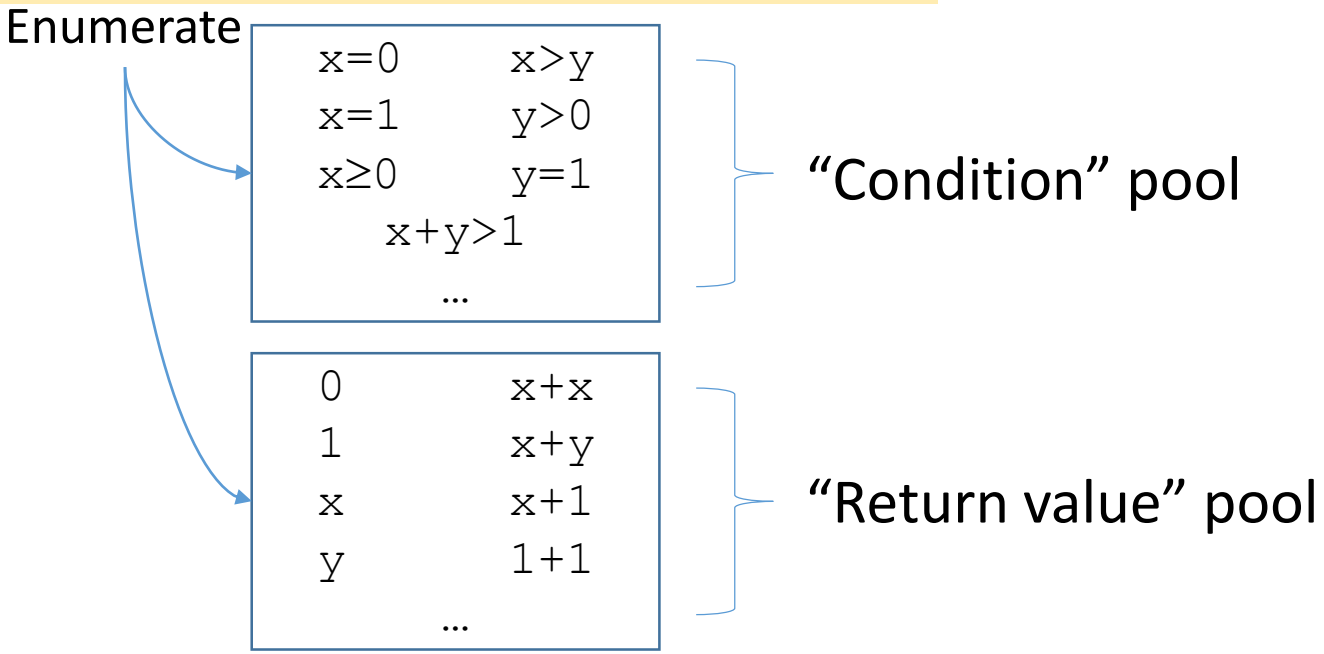
```
A->A+A|-A|x|y|0|1|ite(B,A,A)
                …
```

[Alur/Radhakrishna/Udupa et al 2017]

Unification for ITE

A->A+A|-A|x|y|0|1|~~ite(B,A,A)~~

…



[Alur/Radhakrishna/Udupa et al 2017]

Unification for ITE

```
A->A+A|-A|x|y|0|1|ite(B,A,A)
                …
```

x=0	x>y
x=1	y>0
x≥0	y=1
x+y>1	
…	

0	x+x
1	x+y
x	x+1
y	1+1
…	

$P[f(x_1, y_1)] \wedge$
$P[f(x_2, y_2)] \wedge$
$P[f(x_3, y_3)] \wedge$
$P[f(x_4, y_4)]$

Partial specification

Unification for ITE

A->A+A|-A|x|y|0|1|**ite(B,A,A)**

…

x=0	x>y
x=1	y>0
x≥0	y=1
x+y>1	
…	

0	x+x
1	x+y
x	x+1
y	1+1
…	

$P[f(x_1,y_1)] \wedge P[f(x_2,y_2)] \wedge P[f(x_3,y_3)] \wedge P[f(x_4,y_4)]$

(x_1,y_1)
(x_2,y_2)
(x_3,y_3)
(x_4,y_4)

Unification for ITE

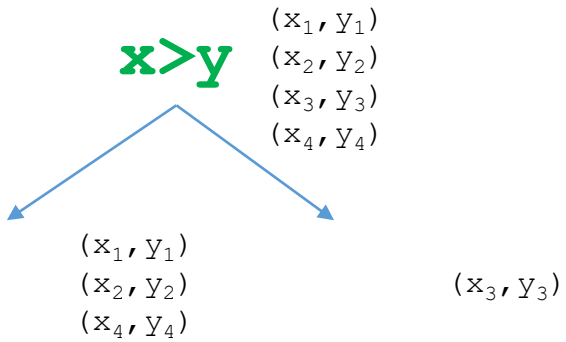
A->A+A|-A|x|y|0|1|**ite(B,A,A)**

…

x=0	**x>y**
x=1	y>0
x≥0	y=1
x+y>1	
…	

0	x+x
1	x+y
x	x+1
y	1+1
…	

$$P[f(x_1,y_1)] \wedge P[f(x_2,y_2)] \wedge P[f(x_3,y_3)] \wedge P[f(x_4,y_4)]$$



Unification for ITE

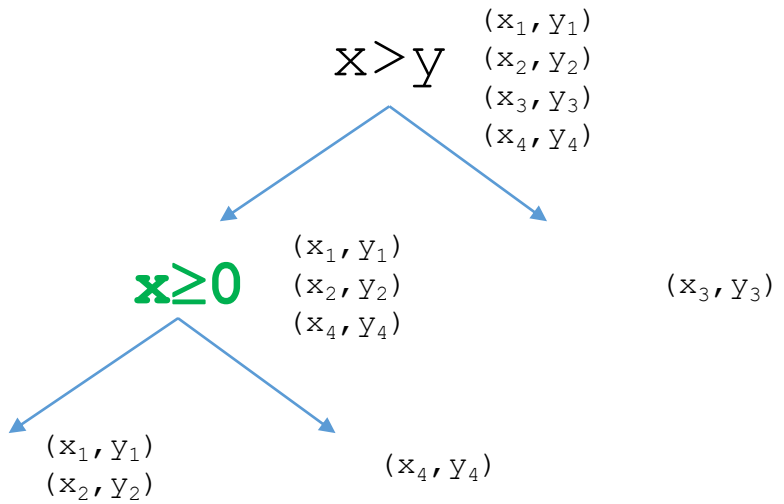
A->A+A|-A|x|y|0|1|**ite(B,A,A)**

…

x=0	x>y
x=1	y>0
x≥0	y=1
x+y>1	
…	

0	x+x
1	x+y
x	x+1
y	1+1
…	

$P[f(x_1,y_1)] \wedge$
$P[f(x_2,y_2)] \wedge$
$P[f(x_3,y_3)] \wedge$
$P[f(x_4,y_4)]$



Unification for ITE

A->A+A|-A|x|y|0|1|**ite(B,A,A)**

…

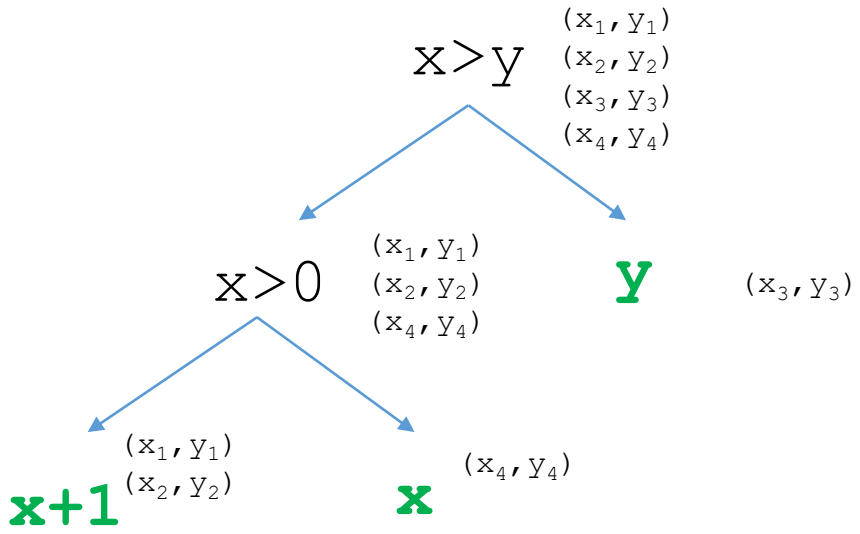
x=0	x>y
x=1	y>0
x≥0	y=1
x+y>1	

…

0	x+x
1	x+y
x	**x+1**
y	1+1

…

$P[f(x_1,y_1)] \wedge$
$P[f(x_2,y_2)] \wedge$
$P[f(x_3,y_3)] \wedge$
$P[f(x_4,y_4)]$



Unification for ITE

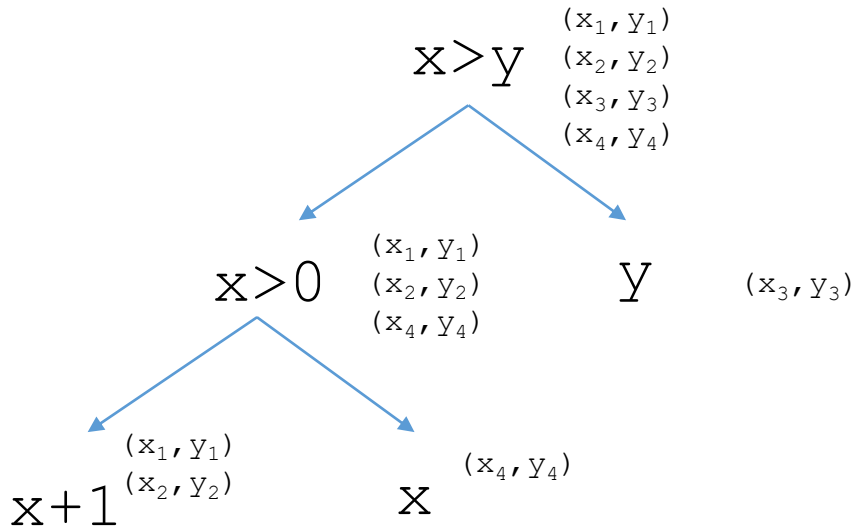
A->A+A|-A|x|y|0|1|**ite(B,A,A)**

…

x=0	x>y
x=1	y>0
x≥0	y=1
x+y>1	
…	

0	x+x
1	x+y
x	x+1
y	1+1
…	

$P[f(x_1,y_1)] \wedge$
$P[f(x_2,y_2)] \wedge$
$P[f(x_3,y_3)] \wedge$
$P[f(x_4,y_4)]$



$f=\lambda x.ite(x>y,ite(x>0,x+1,x),y)$

Related Work

- Learning invariants using decision trees and implication counterexamples

  [Garg et al POPL 2016]
- Data-driven precondition inference with learned features

  [Pahdi et al PLDI 2016]
- ICE-Based Refinement Type Discovery for Higher-Order Functional Programs

  [Champion et al TACAS 2018]
- Invariant Synthesis for Incomplete Verification Engines

  [Neider et al TACAS 2018]

Unification for Strings

```
fStr := x | "_" | "Dr" | ++(fStr,fStr)
        substr(fStr,fInt,fInt)
fInt := 0 | 1 | +(fInt,fInt)
```

$$
\begin{aligned}
&\exists f.\forall x.\\
&\quad f(\text{“Alice”}) = \text{“Dr\_Alice\_A”} \wedge\\
&\quad f(\text{“Bob”}) = \text{“Dr\_Bob\_B”} \wedge\\
&\quad f(\text{“Carl”}) = \text{“Dr\_Carl\_C”} \wedge\\
&\quad f(\text{“David”}) = \text{“Dr\_David\_D”}
\end{aligned}
$$

Unification for Strings

```
fStr := x | "_" | "Dr" | ++(fStr,fStr)
        substr(fStr,fInt,fInt)
fInt := 0 | 1 | +(fInt,fInt)
```

Enumerate

```
          x
        " "
         _
       "Dr"
  substr(x,0,1)
  substr(x,1,1)
  substr(x,0,2)
substr("Dr",0,1)
          …
```

$$
\begin{aligned}
&\exists f.\forall x.\\
&\quad f(\text{“Alice”}) = \text{“Dr\_Alice\_A”} \wedge\\
&\quad f(\text{“Bob”}) = \text{“Dr\_Bob\_B”} \wedge\\
&\quad f(\text{“Carl”}) = \text{“Dr\_Carl\_C”} \wedge\\
&\quad f(\text{“David”}) = \text{“Dr\_David\_D”}
\end{aligned}
$$

Unification for Strings

```
fStr := x | "_" | "Dr" | ++(fStr,fStr)
        substr(fStr,fInt,fInt)
fInt := 0 | 1 | +(fInt,fInt)
```

```
x
" "
_
"Dr"
substr(x,0,1)
substr(x,1,1)
substr(x,0,2)
substr("Dr",0,1)
…
```

Match →

$$
\begin{aligned}
&\exists f.\forall x. \\
&\quad f(\text{“Alice”}) = \text{“Dr\_Alice\_A”} \wedge \\
&\quad f(\text{“Bob”}) = \text{“Dr\_Bob\_B”} \wedge \\
&\quad f(\text{“Carl”}) = \text{“Dr\_Carl\_C”} \wedge \\
&\quad f(\text{“David”}) = \text{“Dr\_David\_D”}
\end{aligned}
$$

Unification for Strings

```
fStr := x | "_" | "Dr" | ++(fStr,fStr)
        substr(fStr,fInt,fInt)
fInt := 0 | 1 | +(fInt,fInt)
```

```
x
" "
"_"
"Dr"
substr(x,0,1)
substr(x,1,1)
substr(x,0,2)
substr("Dr",0,1)
…
```

Match →

$$
\begin{aligned}
&\exists f.\forall x.\\
&\quad f(\text{“Alice”}) = \text{“}\mathbf{Dr}\_\text{Alice\_A”} \wedge\\
&\quad f(\text{“Bob”}) = \text{“}\mathbf{Dr}\_\text{Bob\_B”} \wedge\\
&\quad f(\text{“Carl”}) = \text{“}\mathbf{Dr}\_\text{Carl\_C”} \wedge\\
&\quad f(\text{“David”}) = \text{“}\mathbf{Dr}\_\text{David\_D”}
\end{aligned}
$$

$f=\lambda x.{+}{+}(\text{“}\mathbf{Dr}\text{”},\ldots)$

Unification for Strings

```
fStr := x | "_" | "Dr" | ++(fStr,fStr)
        substr(fStr,fInt,fInt)
fInt := 0 | 1 | +(fInt,fInt)
```

x
" "
"Dr"
substr(x,0,1)
substr(x,1,1)
substr(x,0,2)
substr("Dr",0,1)
…

Match →

$$\exists f.\forall x.$$

$$f(\text{“Alice”}) = \text{“Dr\_Alice\_A”} \wedge$$

$$f(\text{“Bob”}) = \text{“Dr\_Bob\_B”} \wedge$$

$$f(\text{“Carl”}) = \text{“Dr\_Carl\_C”} \wedge$$

$$f(\text{“David”}) = \text{“Dr\_David\_D”}$$

$$f=\lambda x.{+}{+}(\text{“Dr”},\text{“\_”},\ldots)$$

Unification for Strings

```
fStr := x | "_" | "Dr" | ++(fStr,fStr)
        substr(fStr,fInt,fInt)
fInt := 0 | 1 | +(fInt,fInt)
```

x
" "
_
"Dr"
substr(x,0,1)
substr(x,1,1)
substr(x,0,2)
substr("Dr",0,1)
…

Match →

∃f.∀x.

f("Alice") = "Dr_**Alice**_A" ∧

f("Bob") = "Dr_**Bob**_B" ∧

f("Carl") = "Dr_**Carl**_C" ∧

f("David") = "Dr_**David**_D"

f=λx.++("Dr","_",**x**,…)

Unification for Strings

```
fStr := x | "_" | "Dr" | ++(fStr,fStr)
        substr(fStr,fInt,fInt)
fInt := 0 | 1 | +(fInt,fInt)
```

x
"_"
"Dr"
substr(x,0,1)
substr(x,1,1)
substr(x,0,2)
substr("Dr",0,1)
…

Match →

∃f.∀x.

f("Alice") = "Dr_Alice_A" ∧

f("Bob") = "Dr_Bob_B" ∧

f("Carl") = "Dr_Carl_C" ∧

f("David") = "Dr_David_D"

f=λx.++("Dr","_",x,"_",…)

Unification for Strings

```
fStr := x | "_" | "Dr" | ++(fStr,fStr)
        substr(fStr,fInt,fInt)
fInt := 0 | 1 | +(fInt,fInt)
```

x
" "
_
"Dr"
substr(x,0,1)
substr(x,1,1)
substr(x,0,2)
substr("Dr",0,1)
…

Match →

$$\exists f.\forall x.$$

$$f(\text{“Alice”}) = \text{“Dr\_Alice\_}\mathbf{A}\text{”} \wedge$$

$$f(\text{“Bob”}) = \text{“Dr\_Bob\_}\mathbf{B}\text{”} \wedge$$

$$f(\text{“Carl”}) = \text{“Dr\_Carl\_}\mathbf{C}\text{”} \wedge$$

$$f(\text{“David”}) = \text{“Dr\_David\_}\mathbf{D}\text{”}$$

$$f=\lambda x.{+}{+}(\text{“Dr”},\text{“\_”},x,\text{“\_”},\mathbf{substr(x,0,1)})$$

Unification for Strings

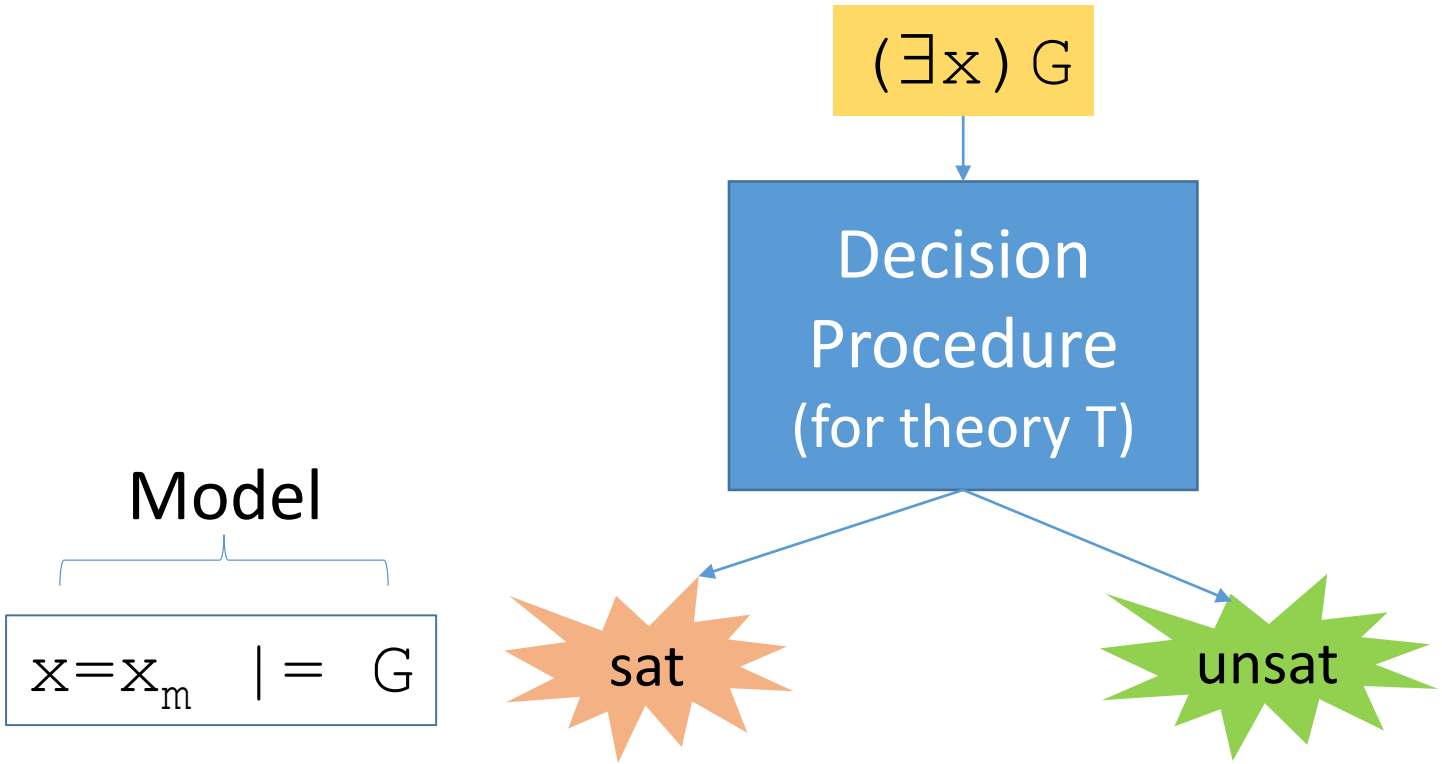
```
fStr := x | "_" | "Dr" | ++(fStr,fStr)
        substr(fStr,fInt,fInt)
fInt := 0 | 1 | +(fInt,fInt)
```

$$
\begin{aligned}
&\exists f.\forall x.\\
&\quad f(\text{“Alice”}) = \text{“Dr\_Alice\_A”} \wedge\\
&\quad f(\text{“Bob”}) = \text{“Dr\_Bob\_B”} \wedge\\
&\quad f(\text{“Carl”}) = \text{“Dr\_Carl\_C”} \wedge\\
&\quad f(\text{“David”}) = \text{“Dr\_David\_D”}
\end{aligned}
$$

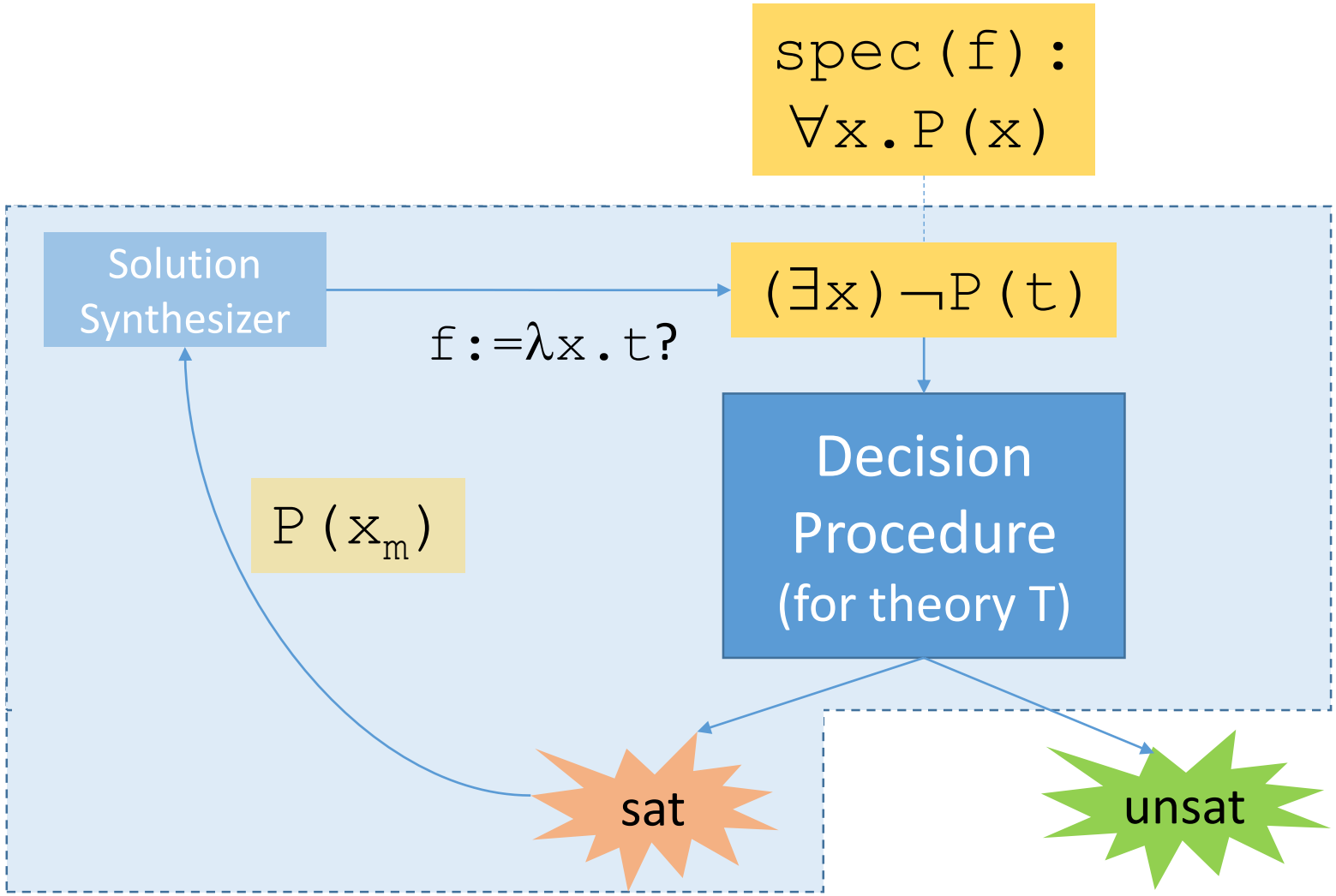
$\Rightarrow$ Return $f=\lambda x.{+}{+}(\text{“Dr”},\text{“\_”},x,\text{“\_”},substr(x,0,1))$

(Decision) Procedures for Verifying Solutions

From Decision Procedures to Synthesis Procedures



From Decision Procedures to Synthesis Procedures



Decision Procedure for T

⇒

Synthesis Procedure for T

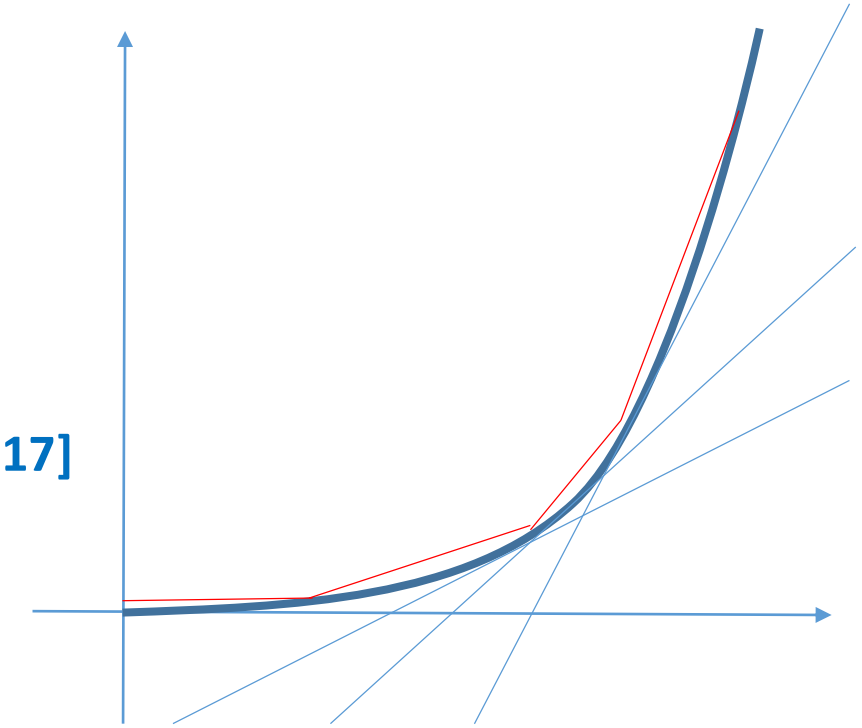
f:=λx.t

CVC4 Supports Many Theories

- Including:
  - Linear arithmetic
  - Bit-Vectors
  - Strings
  - Sets
  - Inductive datatypes and codatatypes
  - Floating points
  - Non-linear arithmetic
- Combinations of all of these (+quantifiers) (+synthesis conjectures)

$\Rightarrow$Entered every division of SMT comp this year

Non-linear Arithmetic



- Incremental linearization

  [Cimatti/Griggio/Irfan/Roveri/Sebastiani TACAS 2017, CADE 2017]

- Model-based reduction:
  - Non-linear arithmetic $\rightarrow$ Uninterpreted functions + Linear Arithmetic

$$(x<c_x \wedge y<c_y) \vee (x>c_x \wedge y>c_y) \Rightarrow x*y>c_y*x+c_x*y+c_x*c_y$$

"Tangent plane"

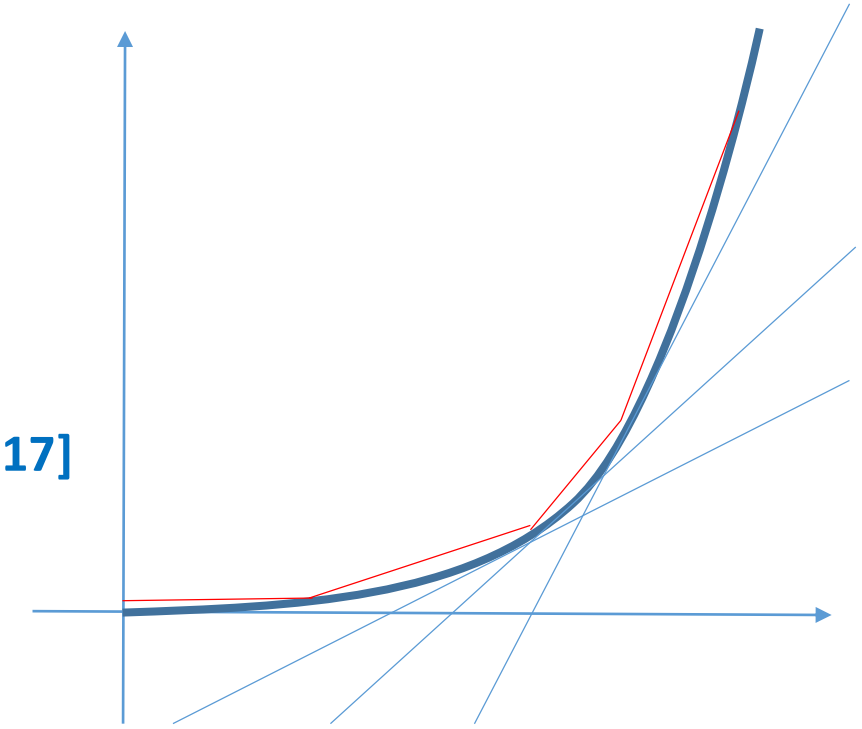
```
x*x<(x-d)*(c*c-d*d)/(c-d)
```

"Secant plane"

Non-linear Arithmetic

- Incremental linearization

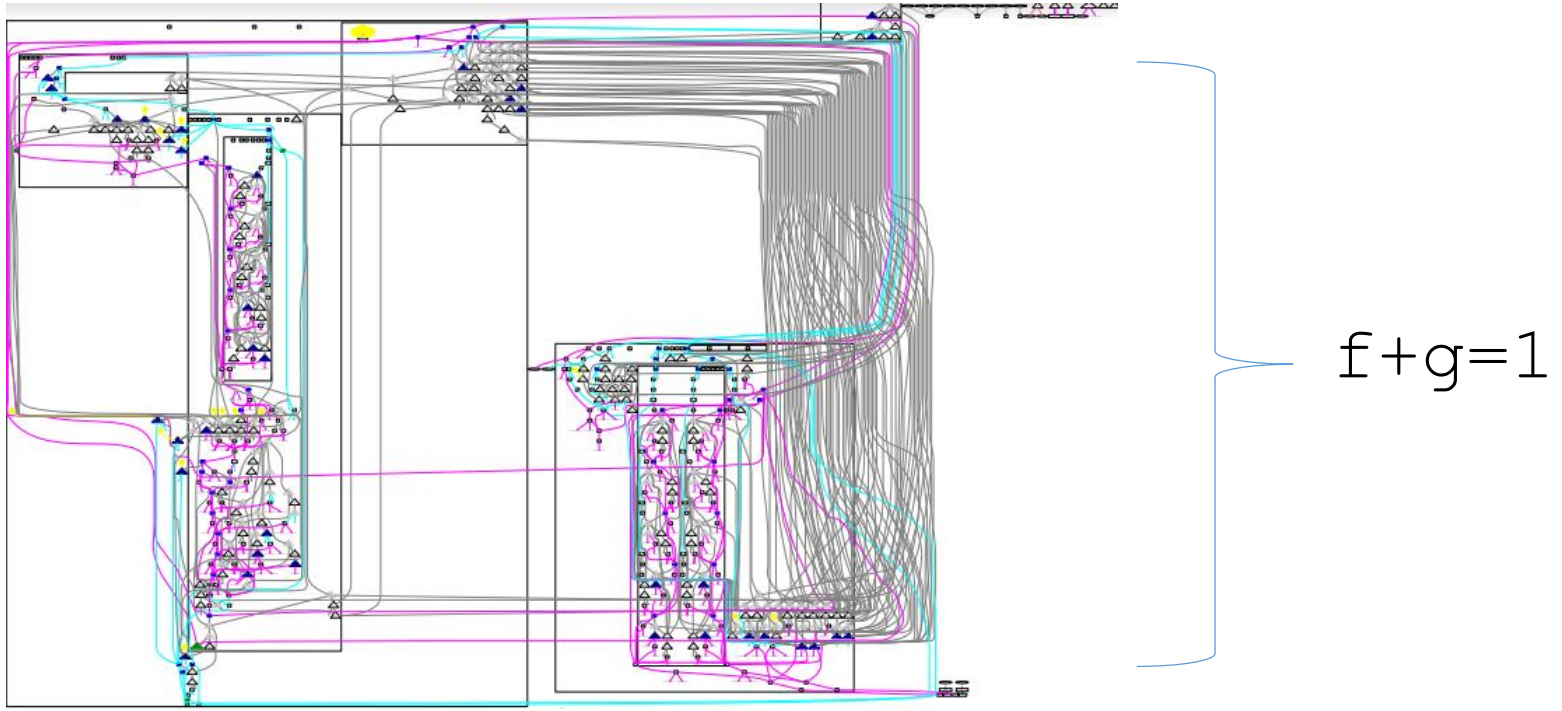
  [Cimatti/Griggio/Irfan/Roveri/Sebastiani TACAS 2017, CADE 2017]



- Can be extended transcendental functions
  - `sin`, `exp`
- Works well for nonlinear arithmetic without real equalities
  - Good for verification applications, bad for theorem proving applications

Floating Points

- State of the art floating point procedure in CVC4 **[Brain et al 2018]**
- Word level conversion to theory of Bit-Vectors using SymFPU
- Many optimizations involving packing/unpacking, encodings:



Putting it Together

CVC4: Open Source Tool for Synthesis Modulo T

- In addition solving Satisfiability Modulo Theories,

  CVC4 has (improving) support for Synthesis Modulo Theories

Progress in Syntax-Guided Synthesis

- SyGuS comp initiative
  - Started 2014, synthesis solvers are improving each year
- This years competition:
  - **GENERAL:** CVC4 (447/598)
    - ⇒ **Program snippets, circuit synthesis, theorem proving, planning**
  - **CLIA:** DryadSynth, CVC4 (85/88)
  - **Invariant Synthesis:** LoopInvGen (115/127)
    - ⇒ **Model checking**
  - **PBE strings:** CVC4 (102/118)
    - ⇒ **Flash fill, data wrangling**
  - **PBE bit-vectors:** CVC4 (724/750)

Using CVC4's Synthesis Solver to Improve Itself

- Solving quantified bit-vectors using Invertibility Conditions

$$\exists C.\forall ab.C(a,b) \Leftrightarrow \exists x.x \oplus a = b$$

[Niemetz et al CAV 2018]

- Rewrites for SMT solvers using Syntax-Guided Synthesis

$$\exists t_1 t_2.\forall x.t_1[x] = t_2[x],\ \ t_1 \not\rightarrow t_2$$

[Reynolds et al SMT 2018]

- **In progress:** Floating point rewrites, abstractions, bound propagation

$$\exists L.\forall x_1 a_1 b_1 x_2 a_2 b_2.$$
$$a_1 \leq x_1 \leq b_1 \wedge a_2 \leq x_2 \leq b_2 \Rightarrow L(a_1,b_1,a_2,b_2) \leq x_1 \oplus x_2$$

Conclusions

- Generic approaches for (Invariant) Synthesis
  - Modulo any theory that the SMT solver supports
- Ongoing work:
  - Faster enumeration
  - Better unification techniques
  - More decision procedures
- Future directions:
  - Invariant synthesis + (pragmatic approaches) for non-linear arithmetic
  - Synthesizing properties of floating point operations

Thanks for Listening!

- CVC4 available at : http://cvc4.cs.stanford.edu/web/