Syntax-Guided Synthesis (SyGuS)

Andres Noetzli

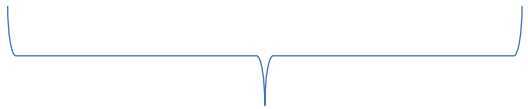
Slides: Andrew Reynolds

Function Synthesis

```
int f(int x, int y)
{
  ???
}
@ensures: @ret ≥ x ∧ @ret ≥ y ∧ (@ret = x ∧ @ret = y)
```

Synthesis Conjectures

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



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

Synthesis Conjectures

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

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

$$f(x,y) \geq x \wedge f(x,y) \geq y \wedge (f(x,y) = x \wedge f(x,y) = y)$$

Synthesis Conjectures *Modulo T*

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

There exists a function f for which property $\mathbb{P}$ holds for all x

Property $\mathbb{P}$ is in some background theory T, e.g.

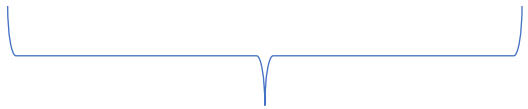
- Linear (or non-linear) arithmetic
- Fixed-width bit-vectors
- Strings
- …

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

$\Rightarrow$ Satisfiability Modulo Theories (SMT)

Synthesis Conjectures Modulo T

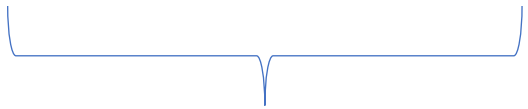
$$\exists f.\forall x.P(f,x)$$



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

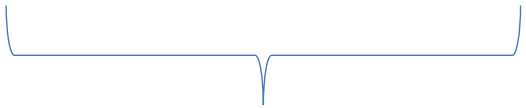
Synthesis Conjectures Modulo T

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



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

Many Approaches to Synthesis

- Naively: treat function f as free uninterpreted function, ask SMT solver to find a model for f

  $\Rightarrow$ This is hard for SMT solvers. Need to use specialized techniques

- Counterexample-guided Inductive Synthesis (CEGIS)
  - Enumerative
  - Symbolic
  - Stochastic
- Specialized solutions for fragments
  - Counterexample-guided quantifier instantiation [Reynolds et al 2015]

Counterexample-Guided Inductive Synthesis (CEGIS)

```
spec(f):
∀xy.f(x,y)=f(y,x)
```

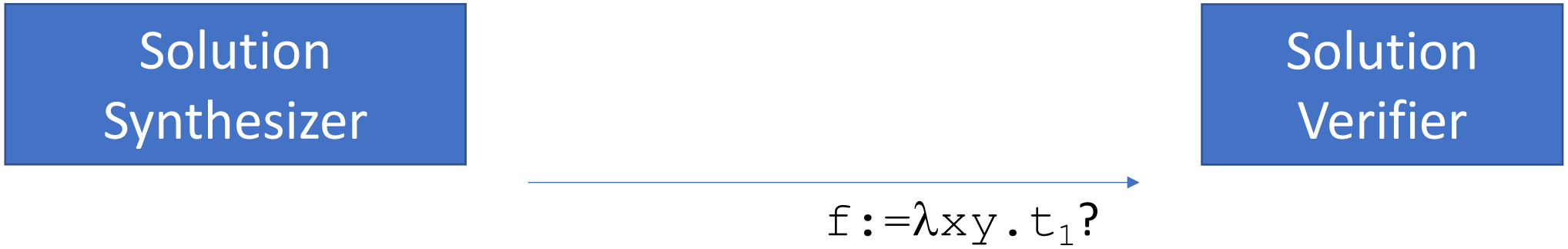
Solution Synthesizer

Solution Verifier

[Solar-Lezama 2006, Alur et al 2013]

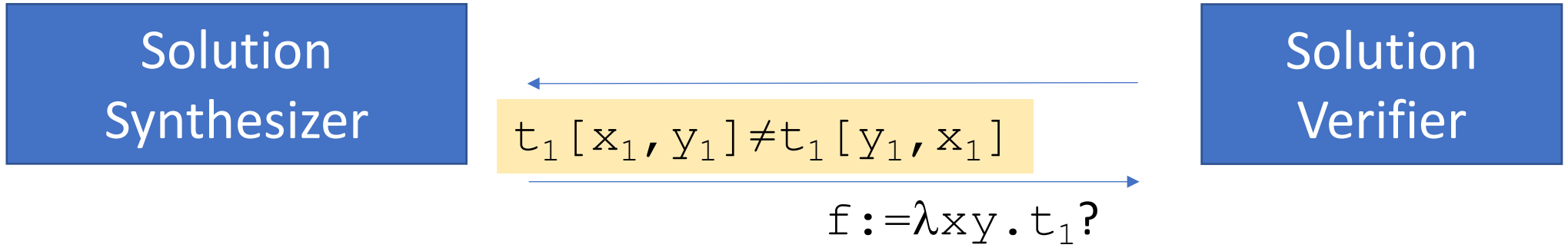
Counterexample-Guided Inductive Synthesis (CEGIS)

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

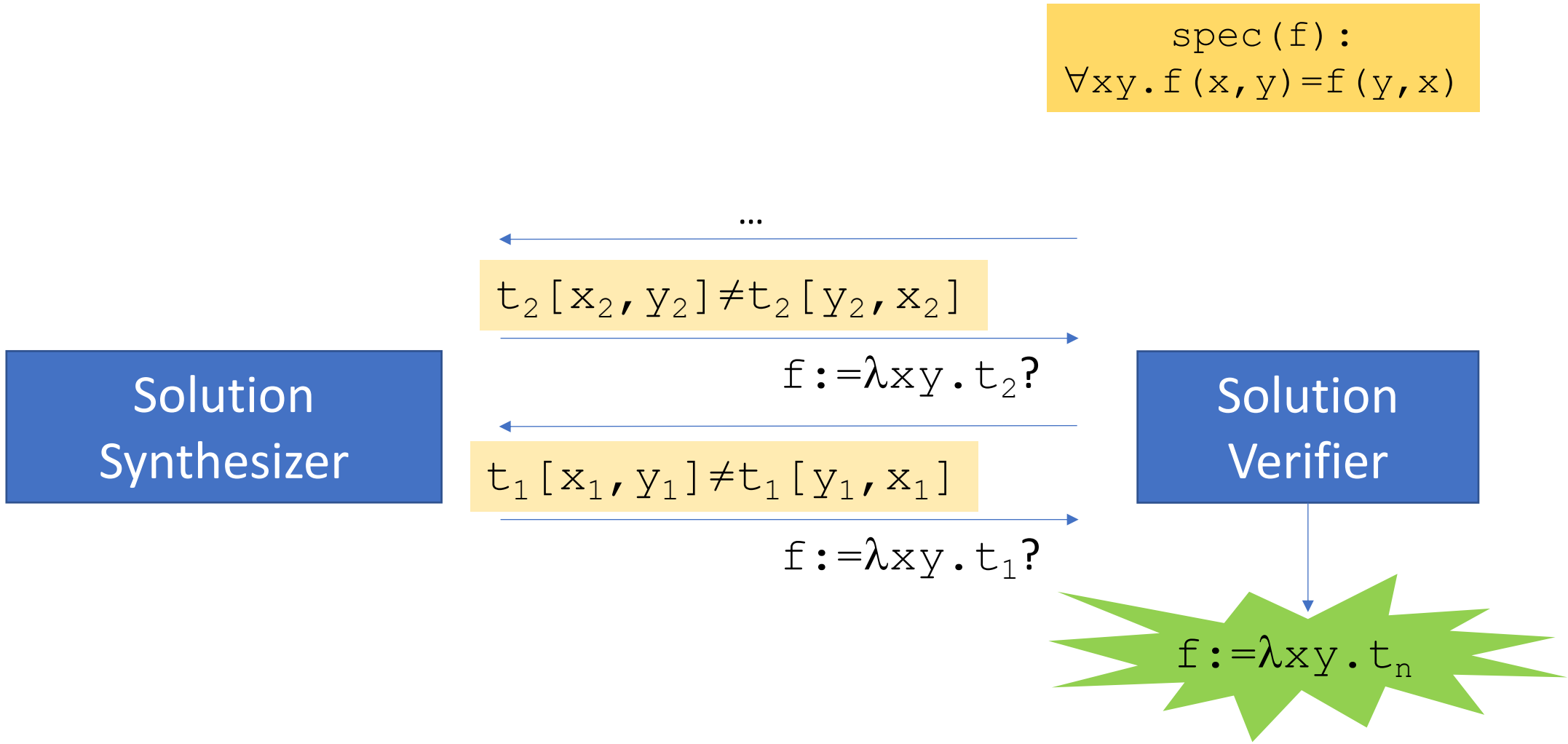


Counterexample-Guided Inductive Synthesis (CEGIS)

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



Counterexample-Guided Inductive Synthesis (CEGIS)



Enumerative 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|⊥
```

Production Rules

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

Solution *Enumerator*

Solution Verifier

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*

Solution Verifier

`f(0,1)=f(1,0)+f(0,1)`

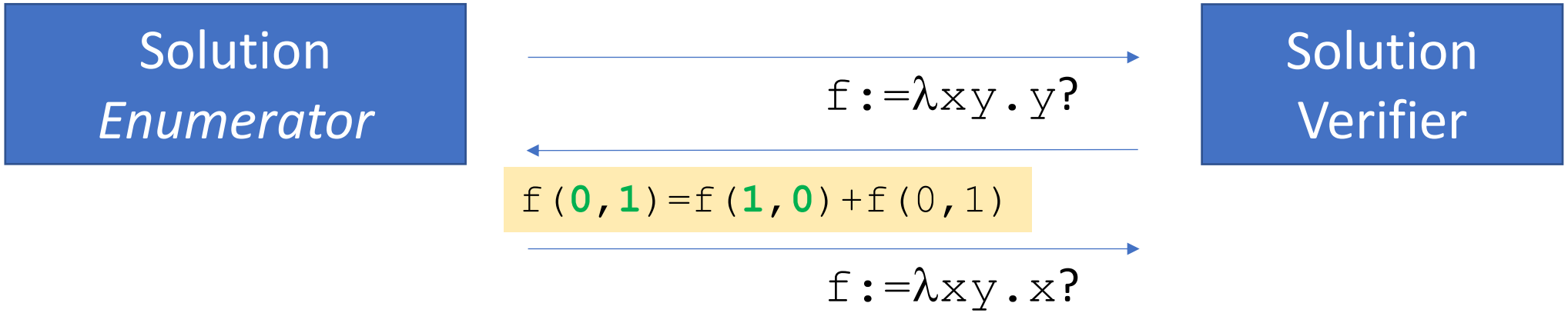
`f:=λxy.x?`

..counterexample: spec does not hold for **`(x,y)=(0,1)`**

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)
```

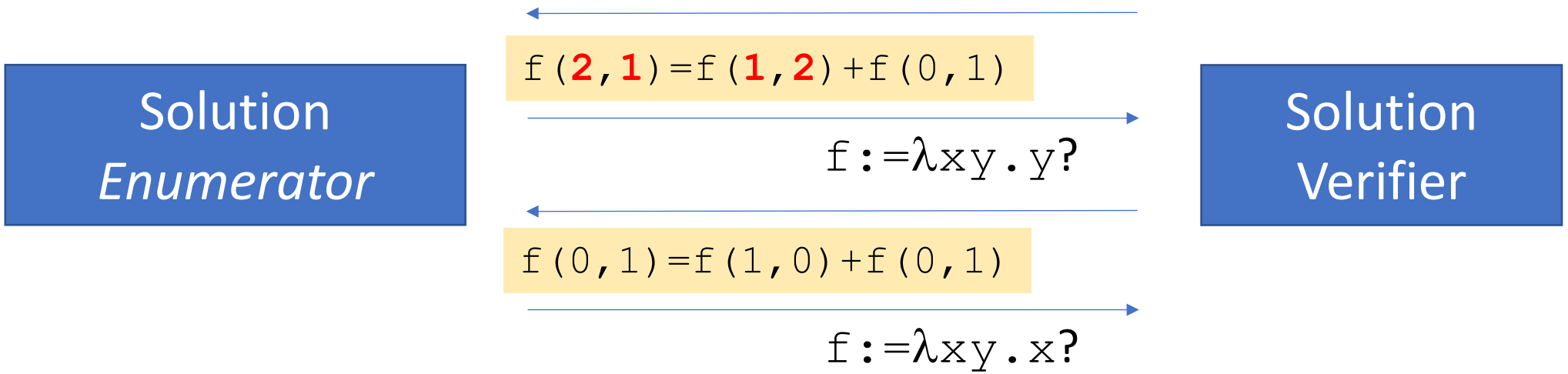


…new candidate λxy.y satisfies **(x,y)=(0,1)**

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)
```

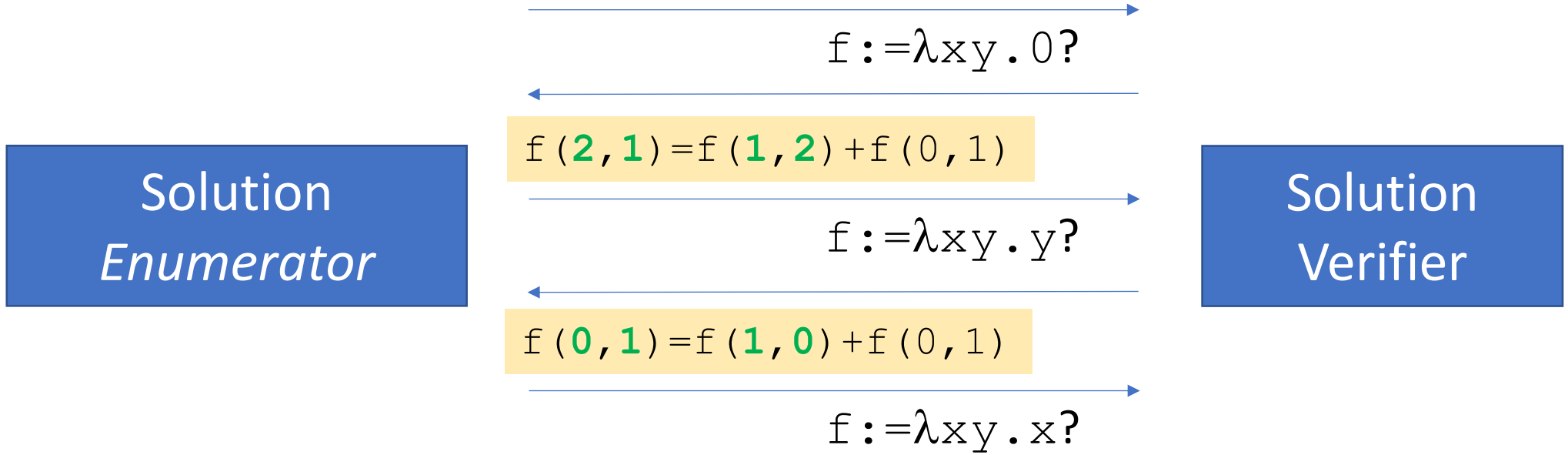


…counterexample: spec does not hold for **(x,y)=(2,1)**

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)
```

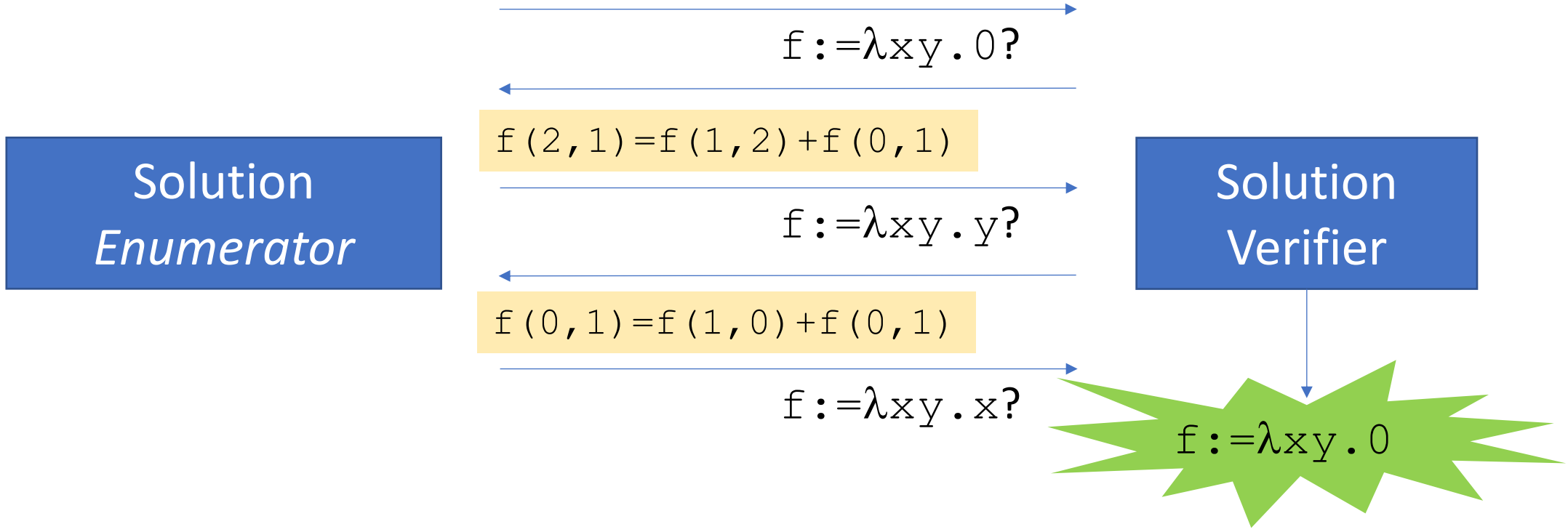


…new candidate λxy.0 satisfies **(x,y)=(0,1),(2,1)**

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)`

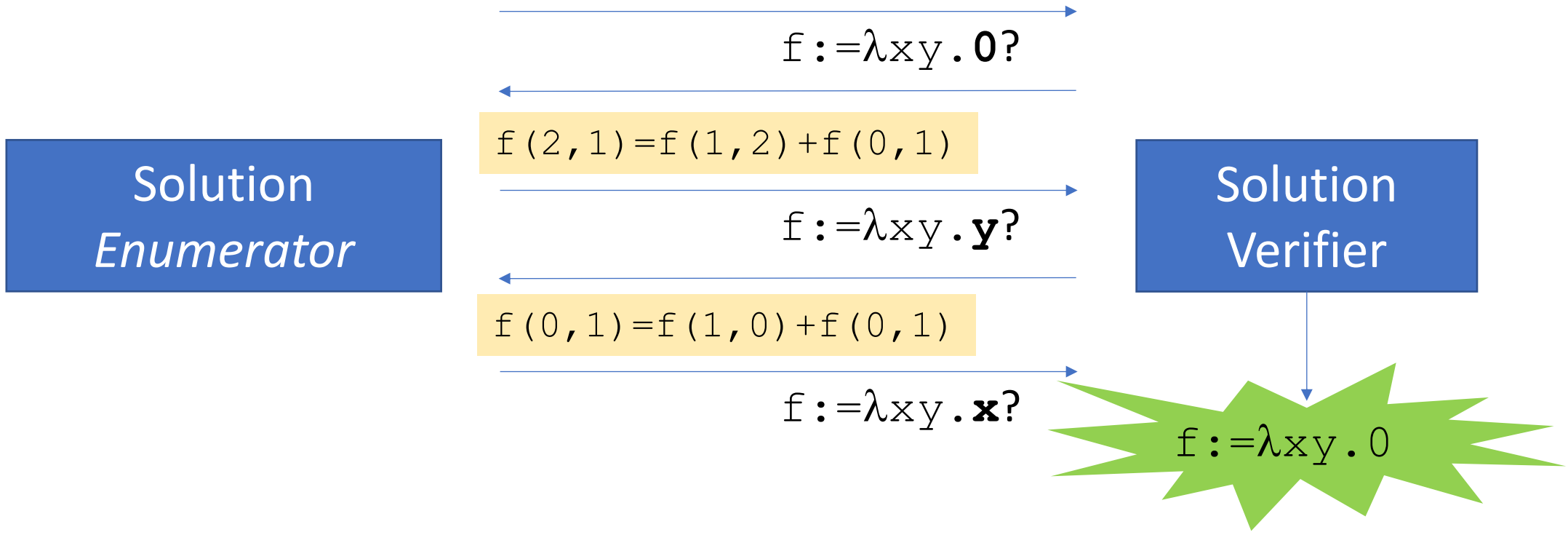


…new candidate `λxy.0` has no counterexamples wrt **`spec(f)`**

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)**

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

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

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

Enumerative Synthesis in SMT Solvers

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

- 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,…`

- 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]**

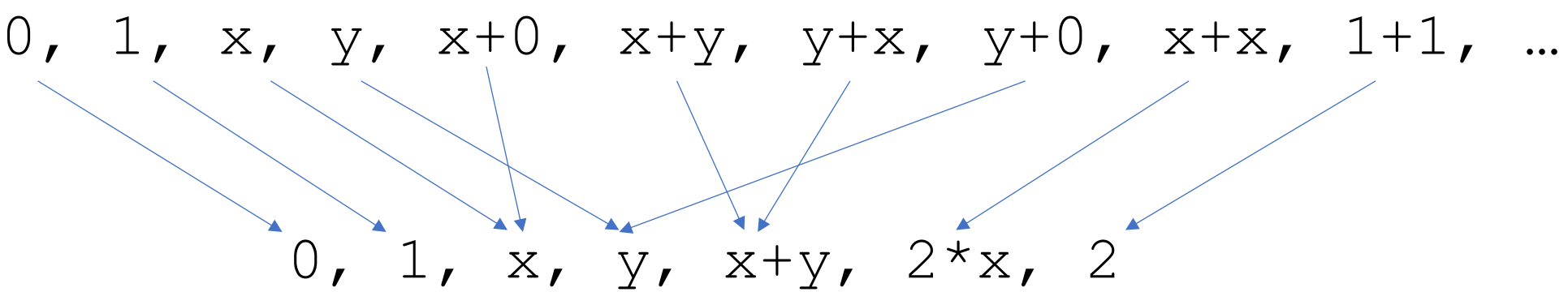
Syntax-Guided Enumeration for T

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

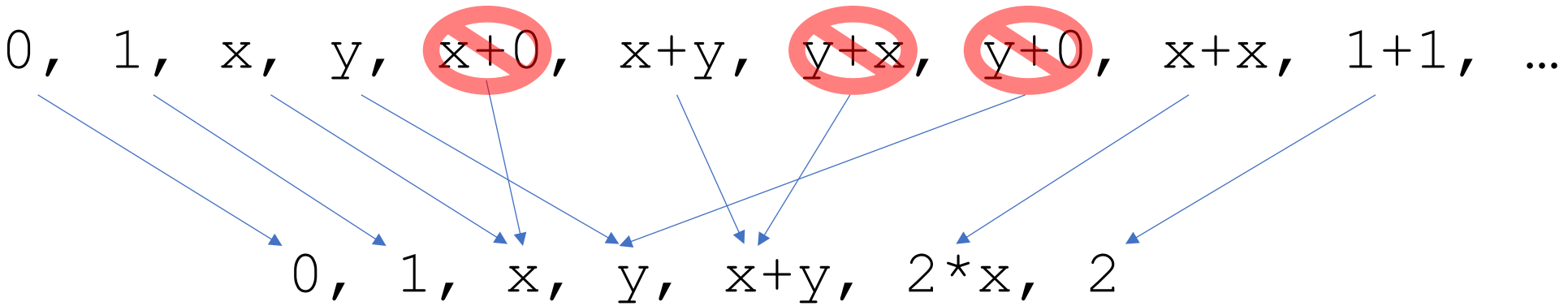
```
0, 1, x, y, x+0, x+y, y+x, y+0, x+x, 1+1, …
```

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

- String, bit-vector, floating-point terms

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)`

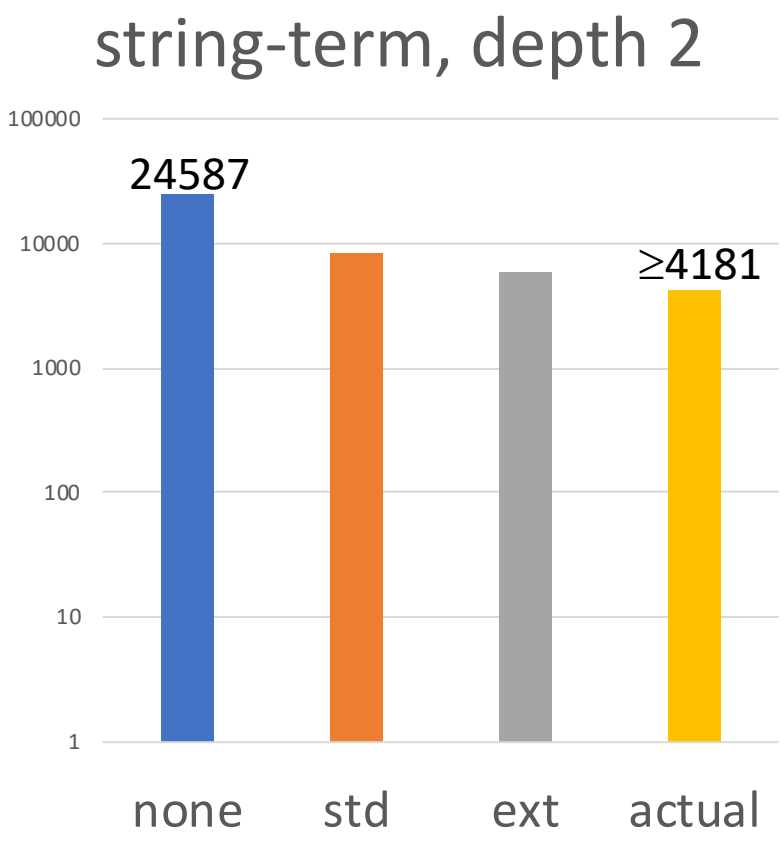
  `contains(x,x++"A")` → ⊥ `replace(x,x++y,y)` → `replace(x,x++y,"")`

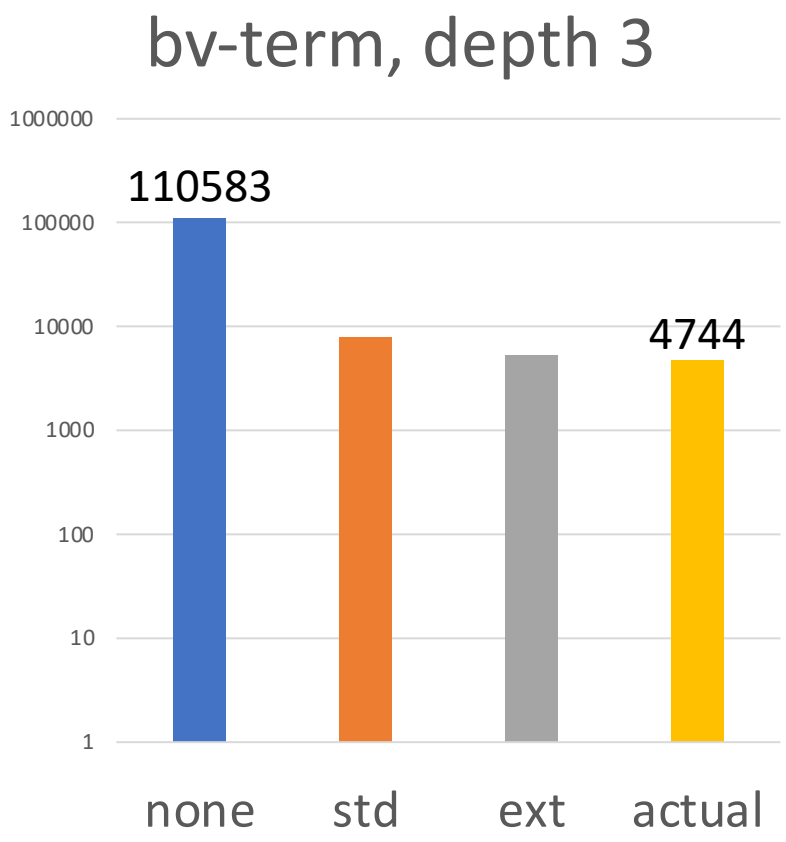
- Booleans

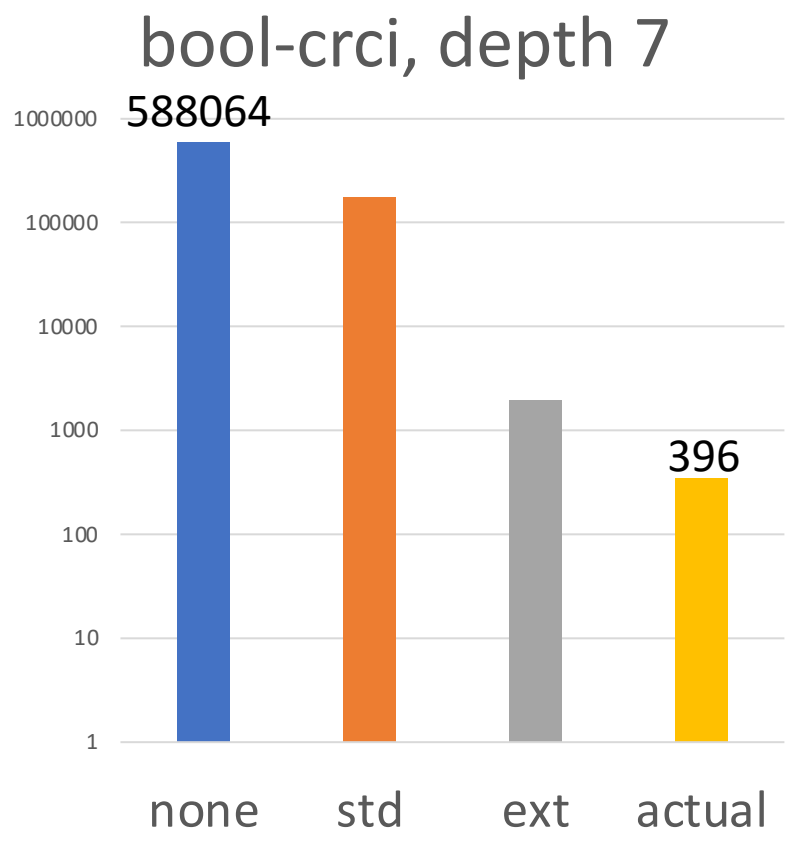
  A∧(A∨B) → A∧B (A∨C)∧(A∨B) → A∧(C∨B)

  A=A&B → ¬A∨B (A∨B)=(A∨B∨C) → A∨B∨¬C

Statistics: CVC4's Current Rewriter(s)







none: # terms from the grammar at given depth

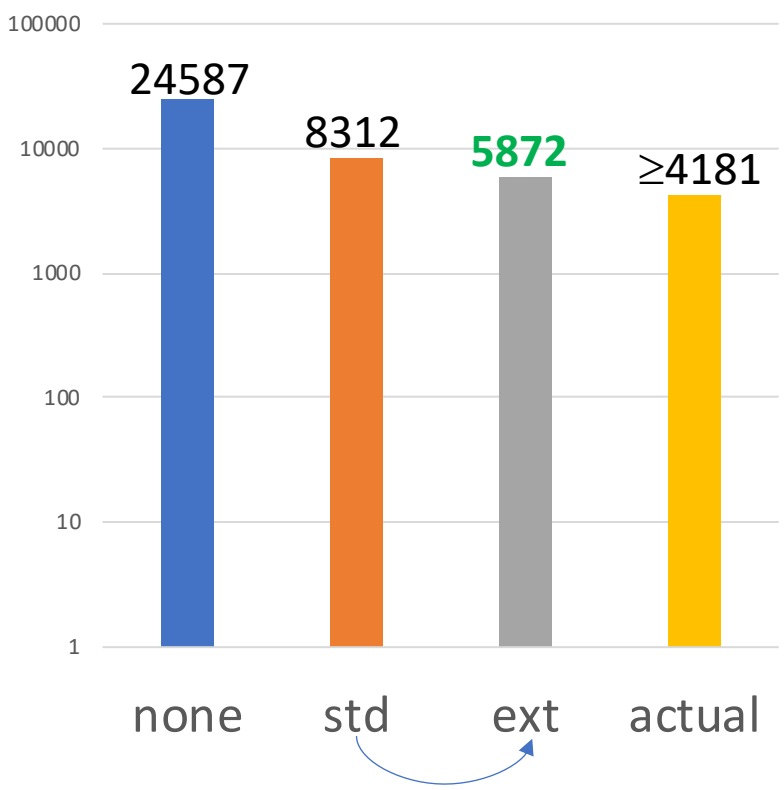
actual: # T-unique terms from grammar at given depth

std: CVC4 version 1.5's rewriter (2 years ago)

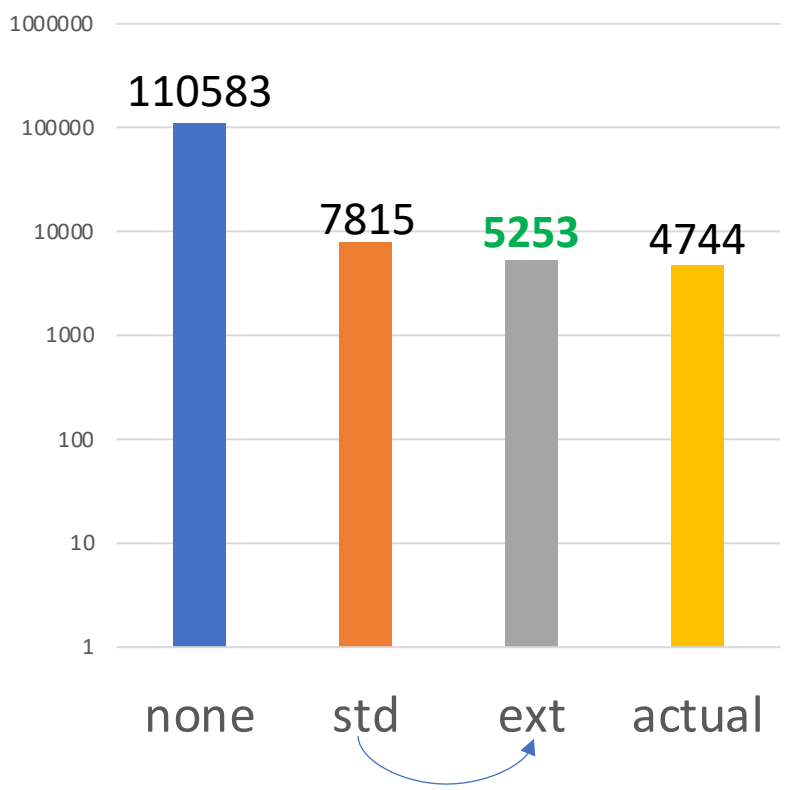
ext: CVC4's aggressive rewriter

Statistics: CVC4's Current Rewriter(s)

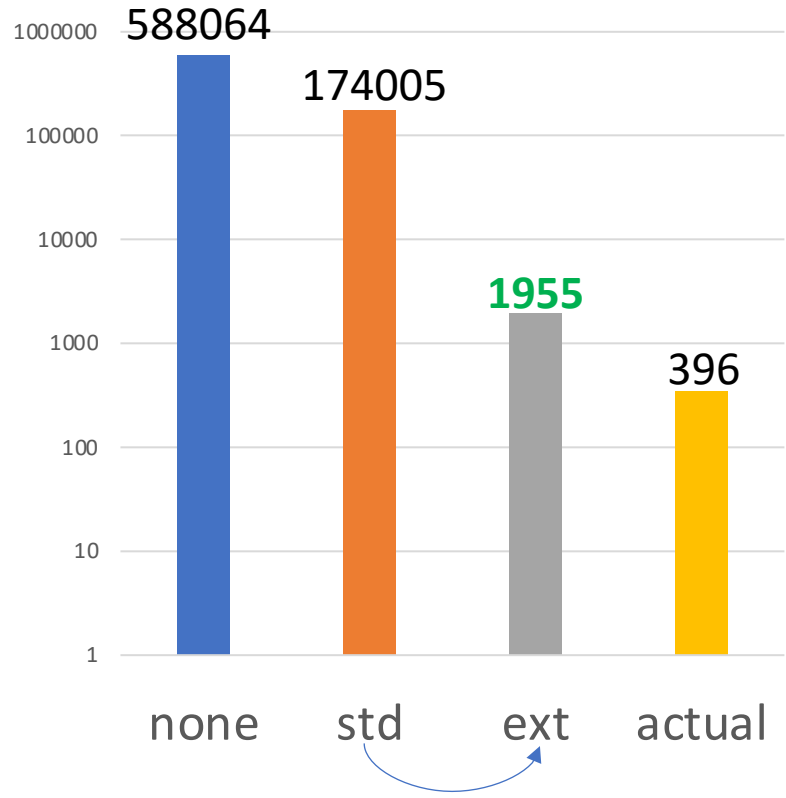
string-term, depth 2



bv-term, depth 3



bool-crci, depth 7



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

Generalizing Rewrites

- How can we generalize rewrites?

$$ite(x = 0 \wedge y \geq 0, 0, x) \rightsquigarrow x$$

Generalizing Rewrites

- How can we generalize rewrites?

$$ite(x = 0 \wedge y \geq 0, 0, x) \overset{w}{\rightsquigarrow} x$$

Generalizing Rewrites

- How can we generalize rewrites?

$$ite(x = 0 \wedge w, 0, x) \rightsquigarrow x$$

Result stays the same regardless of term w

[Reynolds 2019]

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**

CEGIS with Generalization

- How can we generalize failed candidate solutions?

Spec: $f(x, y) \leq x - 1$

$$ite(x \geq 0, x, y + 1)$$

fails on (3,3)

CEGIS with Generalization

- How can we generalize failed candidate solutions?

Spec: $f(x, y) \leq x - 1$

$$ite(x \geq 0, x, \overset{w}{y + 1})$$

CEGIS with Generalization

- How can we generalize failed candidate solutions?

Spec: $f(x, y) \leq x - 1$

$$ite(x \geq 0, x, w)$$

still fails on (3,3)

[Reynolds 2019]

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
```

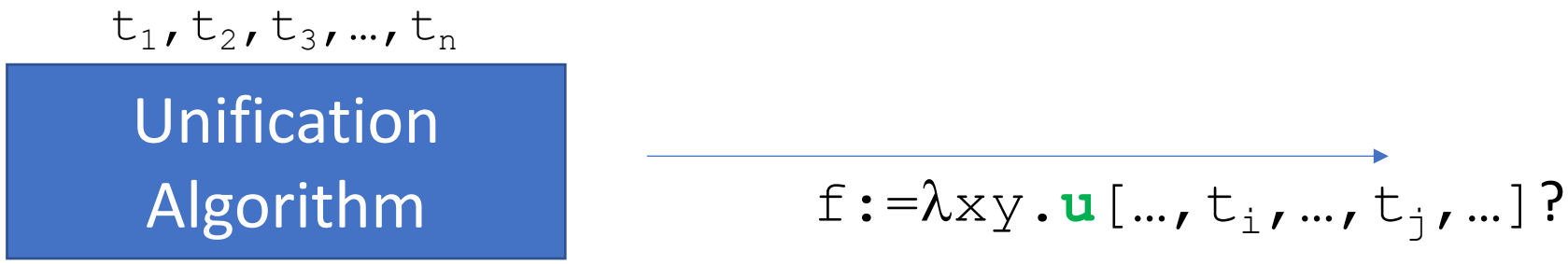
```
f := λxy.
(if0 (bvand (bvnot x) #x0000000000000001)
(if0 (bvand (bvnot (shr4 x)) #x0000000000000001)
(if0 (bvand (shr1 (shr16 x)) #x0000000000000001)
(if0 (bvand (bvnot (shr1 x)) #x0000000000000001) (bvor #x0000000000000001 x) x)
(if0 (bvand (bvnot (shr1 x)) #x0000000000000001) x
(if0 (bvand (shr16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



- Example: ITE/decision tree learning (divide and conquer)

Unification for ITE

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

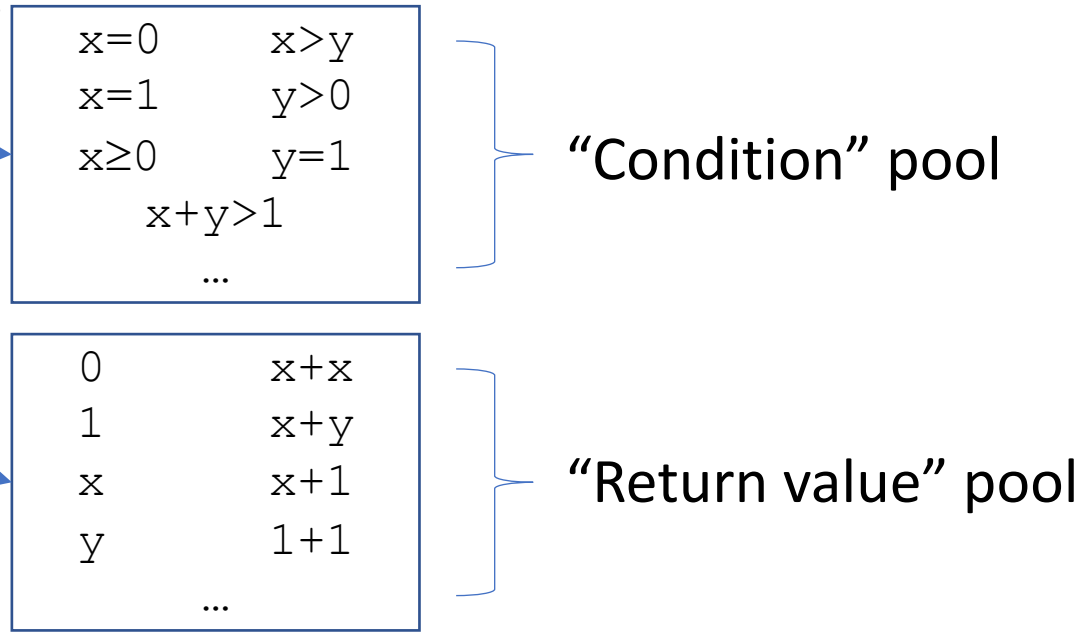
[Alur/Radhakrishna/Udupa et al 2017]

Unification for ITE

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

…

Enumerate



[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

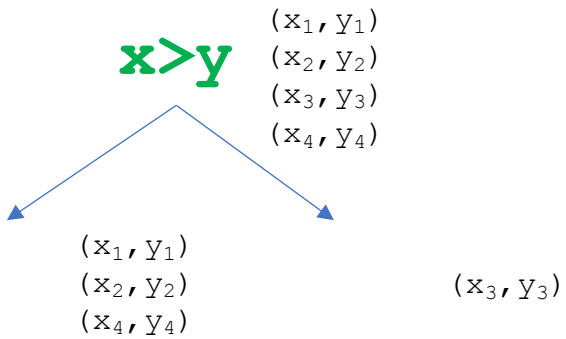
A->A+A|-A|x|y|0|1|**ite(B,A,A)**

…

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

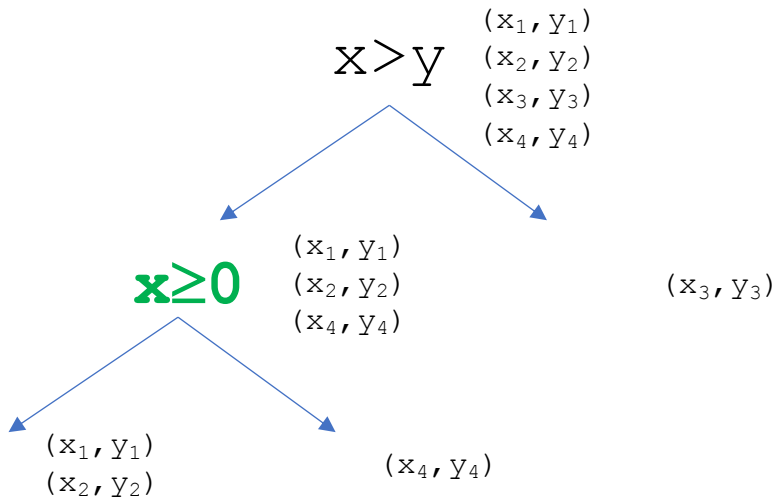
A->A+A|-A|x|y|0|1|**ite(B,A,A)**

…

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

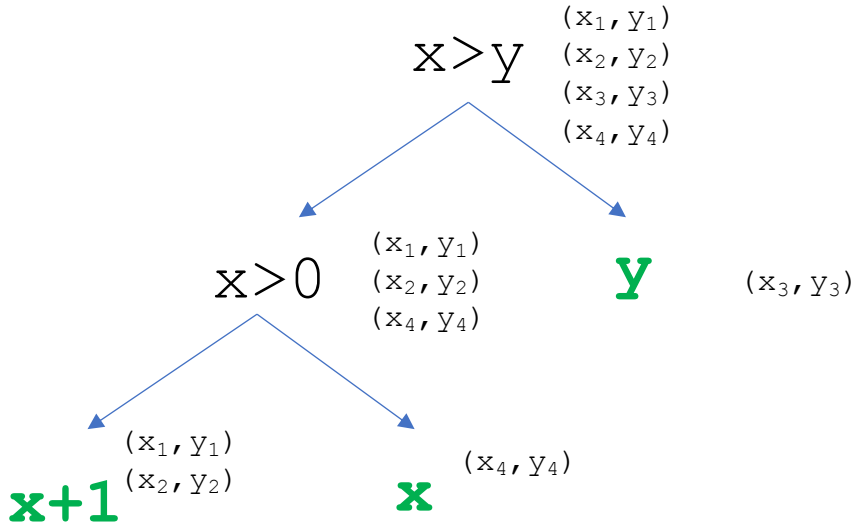
A->A+A|-A|x|y|0|1|**ite(B,A,A)**

…

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)**

…

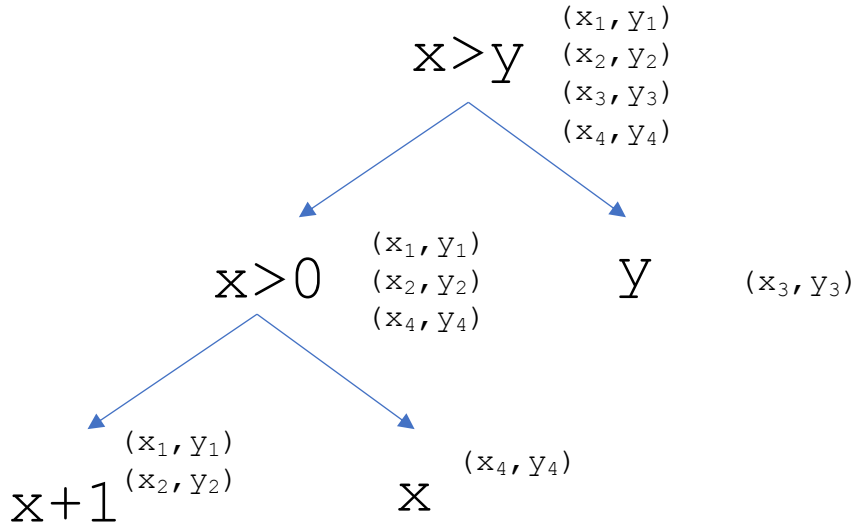
x=0	x>y
x=1	y>0
x≥0	y=1
x+y>1	

…

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

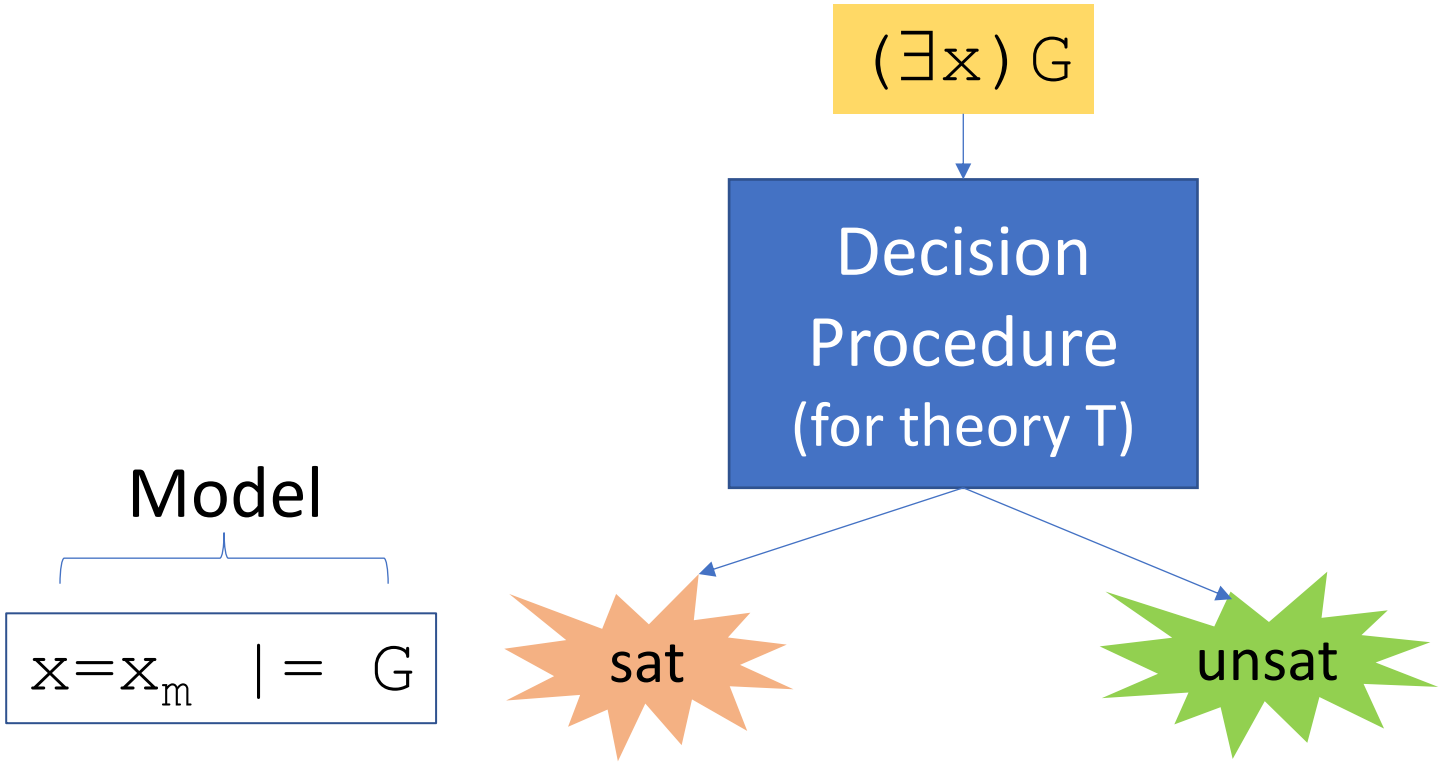
  [Pahdi et al PLDI 2016]
- ICE-Based Refinement Type Discovery for Higher-Order Functional Programs

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

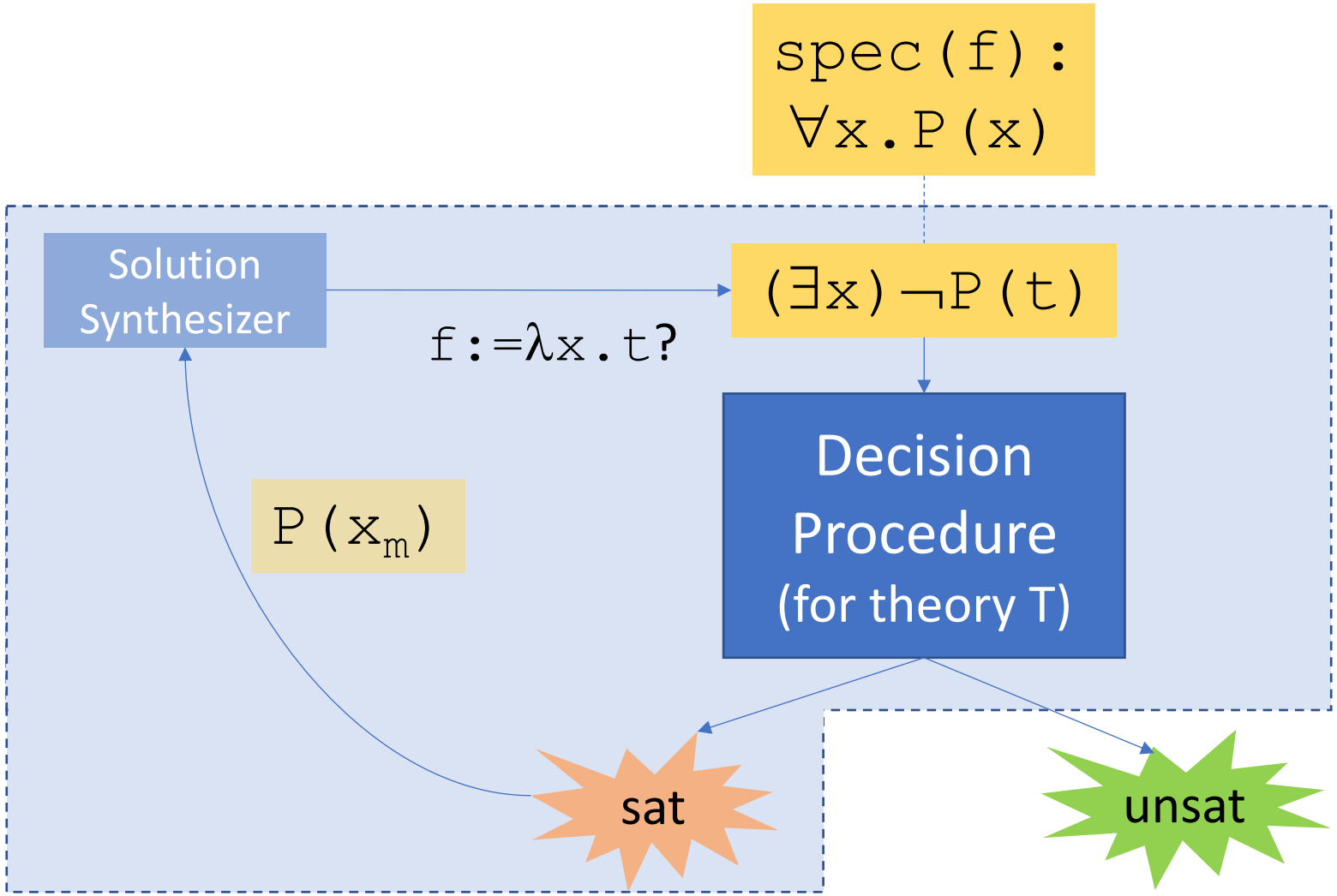
  [Neider et al TACAS 2018]

(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

Progress in Syntax-Guided Synthesis

- SyGuS comp initiative
  - Started 2014, synthesis solvers are improving each year
- This years competition:
  - **GENERAL:** CVC4 (696/886)
    - ⇒ **Program snippets, circuit synthesis, theorem proving, planning**
  - **CLIA:** DryadSynth (87/88)
  - **Invariant Synthesis:** CVC4 (592/829)
    - ⇒ **Model checking**
  - **PBE strings:** CVC4 (204/210)
    - ⇒ **Flash fill, data wrangling**
  - **PBE bit-vectors:** CVC4 (751/753)

Application #2: Solving Quantified Inputs via Invertibility Conditions

Solving Quantified Bit-Vectors: Example

- Consider quantifier elimination problem for bit-vectors:

$$\exists x{:}BV_8.\, s+x=t \Leftrightarrow ???$$

- Common solving technique is model-based instantiation **[Wintersteiger et al 2013]**
  - Based on *values*: above is equivalent $s+\mathbf{\#x00}=t \vee s+\mathbf{\#x01}=t \vee s+\mathbf{\#x02}=t \vee \ldots$
  - Scalability issues for larger bit-widths

- **Idea:** Use SyGuS to find terms that are solutions to bit-vector equations
  - Based on *symbolic solved form*: above is equivalent $s+\mathbf{(t-s)}=t$ (which is $\Leftrightarrow \mathtt{true}$)

Solving Quantified Bit-Vectors

- Some equations are “invertible”, e.g.:
  - `s+x=t` … always has solution **`x=t-s`**
- Some equations are not, e.g.:
  - `s*x=t` … `x` has *no solution* when, e.g. `s=2`, `t=3`

[Niemetz/Preiner/Reynolds/Barrett/Tinelli CAV 2018]

Solving Quantified Bit-Vectors

- `s+x=t` … always has solution **`x=t-s`**
- `s*x=t` … `x` has *no solution* when, e.g. `s=2, t=3`

- **Challenge:** it is possible to characterize exactly when `x` has a solution?
  - Find a predicate `IC(s,t)` such that x has a solution iff `IC(s,t)` is satisfiable
  - We call IC an “invertibility condition”
  - Invertibility Conditions $\Rightarrow$ QE procedure for “linear” BV formulas

[Niemetz/Preiner/Reynolds/Barrett/Tinelli CAV 2018]

Finding Invertibility Conditions

- Goal: for each BV operator $\oplus$ and BV relation ~, find *invertibility condition* `IC(s,t)` such that:

$$\exists IC.\ \exists x.(x \oplus s \sim t) \Leftrightarrow IC(s,t)$$

  - Also need to consider multiple arguments for non-commutative operators
    - E.g. `x >> s = t` and `s >> x = t`

Finding Invertibility Conditions

$$\exists IC.\ \exists x.(x \oplus s \sim t) \Leftrightarrow IC(s,t)$$

- Signature of BV has >15 operators, 4 relations (=/≠, signed/unsigned inequality)
  - $\Rightarrow$ Total of 162 invertibility conditions to find
  - $\Rightarrow$ Some took several hours to find by hand $\therefore$ *use SyGuS*

Use SyGuS to Find Invertibility Conditions

- Invertibility conditions as a synthesis problem:

$$\exists IC.\ \forall st.\ (\exists x.\ x \oplus s \sim t) \Leftrightarrow IC(s,t)$$

- Find predicate `IC` that is the invertibility condition for $x \oplus s \sim t$

Use SyGuS to Find Invertibility Conditions

- Invertibility conditions as a synthesis problem:

$$\exists IC.\ \forall st.\ (\exists x.\ x \oplus s \sim t) \Leftrightarrow IC(s,t)$$

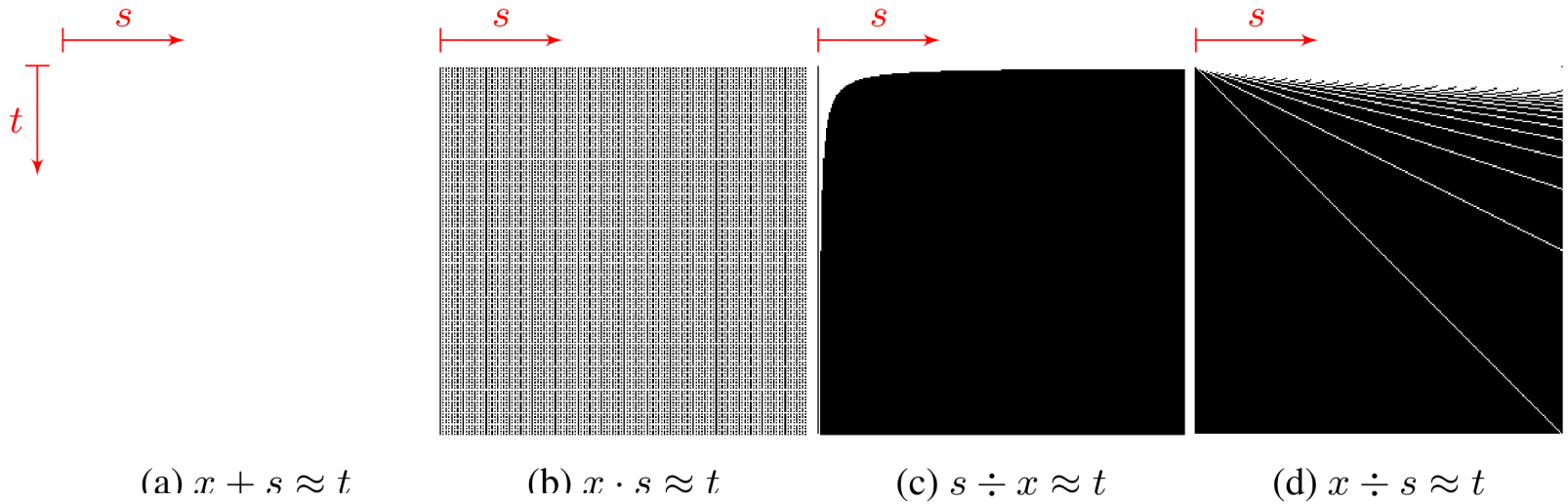
Challenge: 3 levels of quantification (SyGuS only deals with 2)

Use SyGuS to Find Invertibility Conditions

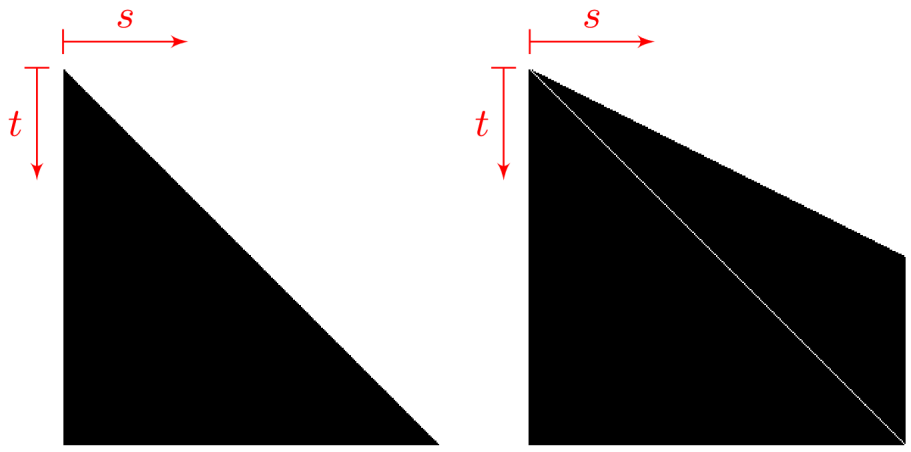
$$\exists IC.\ \forall st.\left(\bigvee_{c=0,\ldots,15} c \oplus s \sim t\right) \Leftrightarrow IC(s,t)$$

- Solution: find invertibility conditions for a small fixed bitwidth (**4-bits**)
  - In practice, these solutions generalize to all bit-widths
- Using this technique, found **118** of **162** conditions
  - Many simpler than hand-crafted ones
- When combined with hand-crafted ICs, found all **162** conditions

Visualizing BV Invertibility Conditions



(a) $x + s \approx t$ (b) $x \cdot s \approx t$ (c) $s \div x \approx t$ (d) $x \div s \approx t$



(a) x rem $s \approx t$ (b) s rem $x \approx t$

White = IC is true

Black = IC is false

(shown for 8-bit)

Invertibility Conditions for BV

$\ell[x]$	$\approx$	$\not\approx$
$x \cdot s \bowtie t$	$(-s \mid s)\ \&\ t \approx t$	$s \not\approx 0 \vee t \not\approx 0$
$x \bmod s \bowtie t$	$\sim(-s) \geq_u t$	$s \not\approx 1 \vee t \not\approx 0$
$s \bmod x \bowtie t$	$(t + t - s)\ \&\ s \geq_u t$	$s \not\approx 0 \vee t \not\approx 0$
$x \div s \bowtie t$	$(s \cdot t) \div s \approx t$	$s \not\approx 0 \vee t \not\approx \sim 0$
$s \div x \bowtie t$	$s \div (s \div t) \approx t$	$\begin{cases} s\ \&\ t \approx 0 & \text{for } \kappa(s) = 1 \\ \top & \text{otherwise} \end{cases}$
$x\ \&\ s \bowtie t$	$t\ \&\ s \approx t$	$s \not\approx 0 \vee t \not\approx 0$
$x \mid s \bowtie t$	$t \mid s \approx t$	$s \not\approx \sim 0 \vee t \not\approx \sim 0$
$x \gg s \bowtie t$	$(t \ll s) \gg s \approx t$	$t \not\approx 0 \vee s <_u \kappa(s)$
$s \gg x \bowtie t$	$\bigvee_{i=0}^{\kappa(s)} s \gg i \approx t$	$s \not\approx 0 \vee t \not\approx 0$
$x \gg_a s \bowtie t$	$(s <_u \kappa(s) \Rightarrow (t \ll s) \gg_a s \approx t) \wedge$ $(s \geq_u \kappa(s) \Rightarrow (t \approx \sim 0 \vee t \approx 0))$	$\top$
$s \gg_a x \bowtie t$	$\bigvee_{i=0}^{\kappa(s)} s \gg_a i \approx t$	$(t \not\approx 0 \vee s \not\approx 0) \wedge$ $(t \not\approx \sim 0 \vee s \not\approx \sim 0)$
$x \ll s \bowtie t$	$(t \gg s) \ll s \approx t$	$t \not\approx 0 \vee s <_u \kappa(s)$
$s \ll x \bowtie t$	$\bigvee_{i=0}^{\kappa(s)} s \ll i \approx t$	$s \not\approx 0 \vee t \not\approx 0$
$x \circ s \bowtie t$	$s \approx t[\kappa(s) - 1 : 0]$	$\top$
$s \circ x \bowtie t$	$s \approx t[\kappa(t) - 1 : \kappa(t) - \kappa(s)]$	$\top$

Table 2. Conditions for the invertibility of bit-vector operators over (dis)equality. Those for $\cdot$, $\&$ and $\mid$ are given modulo commutativity of those operators.

ICs for =,≠

…not pictured:
ICs for `bvuge`, `bvugt`, `bvsge`, `bvsgt`

[Niemetz/Preiner/Reynolds/Barrett/Tinelli CAV 2018]

Invertibility Conditions for BV

$\ell[x]$	$\approx$	$\not\approx$
$x \cdot s \bowtie t$	$(-s \mid s)\ \&\ t \approx t$	$s \not\approx 0 \vee t \not\approx 0$
$x \bmod s \bowtie t$	$\sim(-s) \geq_u t$	$s \not\approx 1 \vee t \not\approx 0$
$s \bmod x \bowtie t$	$(t + t - s)\ \&\ s \geq_u t$	$s \not\approx 0 \vee t \not\approx 0$
$x \div s \bowtie t$	$(s \cdot t) \div s \approx t$	$s \not\approx 0 \vee t \not\approx \sim 0$
$s \div x \bowtie t$	$s \div (s \div t) \approx t$	$\begin{cases} s\ \&\ t \approx 0 & \text{for } \kappa(s) = 1 \\ \top & \text{otherwise} \end{cases}$
$x\ \&\ s \bowtie t$	$t\ \&\ s \approx t$	$s \not\approx 0 \vee t \not\approx 0$
$x \mid s \bowtie t$	$t \mid s \approx t$	$s \not\approx \sim 0 \vee t \not\approx \sim 0$
$x \gg s \bowtie t$	$(t \ll s) \gg s \approx t$	$t \not\approx 0 \vee s <_u \kappa(s)$
$s \gg x \bowtie t$	$\bigvee_{i=0}^{\kappa(s)} s \gg i \approx t$	$s \not\approx 0 \vee t \not\approx 0$
$x \gg_a s \bowtie t$	$(s <_u \kappa(s) \Rightarrow (t \ll s) \gg_a s \approx t) \wedge$ $(s \geq_u \kappa(s) \Rightarrow (t \approx \sim 0 \vee t \approx 0))$	$\top$
$s \gg_a x \bowtie t$	$\bigvee_{i=0}^{\kappa(s)} s \gg_a i \approx t$	$(t \not\approx 0 \vee s \not\approx 0) \wedge$ $(t \not\approx \sim 0 \vee s \not\approx \sim 0)$
$x \ll s \bowtie t$	$(t \gg s) \ll s \approx t$	$t \not\approx 0 \vee s <_u \kappa(s)$
$s \ll x \bowtie t$	$\bigvee_{i=0}^{\kappa(s)} s \ll i \approx t$	$s \not\approx 0 \vee t \not\approx 0$
$x \circ s \bowtie t$	$s \approx t[\kappa(s) - 1 : 0]$	$\top$
$s \circ x \bowtie t$	$s \approx t[\kappa(t) - 1 : \kappa(t) - \kappa(s)]$	$\top$

Table 2. Conditions for the invertibility of bit-vector operators over (dis)equality. Those for $\cdot$, & and | are given modulo commutativity of those operators.

$$\exists x.\ s * x = t$$

$$\Leftrightarrow$$

$$(-s \mid s)\ \&\ t = t$$

- Condition above encodes
  "s has fewer trailing zeroes than t"
- Conditions like this one are concise, subtle
  $\Rightarrow$ Excellent target for SyGuS

[Niemetz/Preiner/Reynolds/Barrett/Tinelli CAV 2018]

(Compact) Quantifier Instantiation for BV

- Theorem:

Theorem 9. *Let $\psi[\boldsymbol{x}]$ be a quantifier-free formula in the signature of T_{BV}.*

1. $\mathcal{S}_{c}^{BV}$ *is a finite selection function for* $\boldsymbol{x}$ *and* ψ *for all* $c \in \{\mathbf{m}, \mathbf{k}, \mathbf{s}, \mathbf{b}\}$.
2. $\mathcal{S}_{\mathbf{m}}^{BV}$ *is monotonic.*
3. $\mathcal{S}_{\mathbf{k}}^{BV}$ *is 1-finite if* ψ *is unit linear invertible.*
4. $\mathcal{S}_{\mathbf{k}}^{BV}$ *is monotonic if* ψ *is unit linear invertible.*

"Linear invertible" quantified BV formulas $\forall \mathtt{x} \,.\, \mathtt{P}[\mathtt{x}]$ containing at-most one occurrence of $\mathtt{x}$ can be solved via a single quantifier-free satisfiability query

$\Rightarrow$ Compact quantifier elimination for this fragment (size is independent of bit-width)

[Niemetz/Preiner/Reynolds/Barrett/Tinelli CAV 2018]

Experimental Results

- Quantifier Instantiation based on Invertibility Conditions in CVC4
- Won quantified bit-vector (BV) category of SMT-COMP 2018

unsat	Boolector	CVC4	Q3B	Z3	cegqi$_m$	cegqi$_k$	cegqi$_s$	cegqi$_b$
h-uauto	14	12	93	24	10	103	105	**106**
keymaera	3917	3790	3781	**3923**	3803	3798	3888	3918
psyco	**62**	**62**	49	**62**	**62**	39	**62**	61
scholl	57	36	13	**67**	36	27	36	35
tptp	55	52	**56**	**56**	**56**	**56**	**56**	**56**
uauto	**137**	72	131	**137**	72	72	135	**137**
ws-fixpoint	74	71	**75**	74	**75**	74	**75**	**75**
ws-ranking	16	8	18	**19**	15	11	12	11
Total unsat	4332	4103	4216	4362	4129	4180	4369	**4399**

sat	Boolector	CVC4	Q3B	Z3	cegqi$_m$	cegqi$_k$	cegqi$_s$	cegqi$_b$
h-uauto	15	10	**17**	13	16	**17**	16	**17**
keymaera	**108**	21	24	**108**	20	13	36	75
psyco	131	**132**	50	131	**132**	60	**132**	129
scholl	**232**	160	201	204	203	188	208	211
tptp	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**
uauto	14	14	15	**16**	14	14	14	14
ws-fixpoint	45	49	**54**	36	45	51	49	50
ws-ranking	19	15	**37**	33	33	31	31	32
Total sat	**581**	418	415	558	480	391	503	545

Total (5151)	4913	4521	4631	4920	4609	4571	4872	**4944**

Table 4. Results on satisfiable and unsatisfiable benchmarks with a 300 second timeout.