Automated Discovery of Invertibility Conditions via Syntax-Guided Synthesis

Andrew Reynolds

Dec 2, 2021

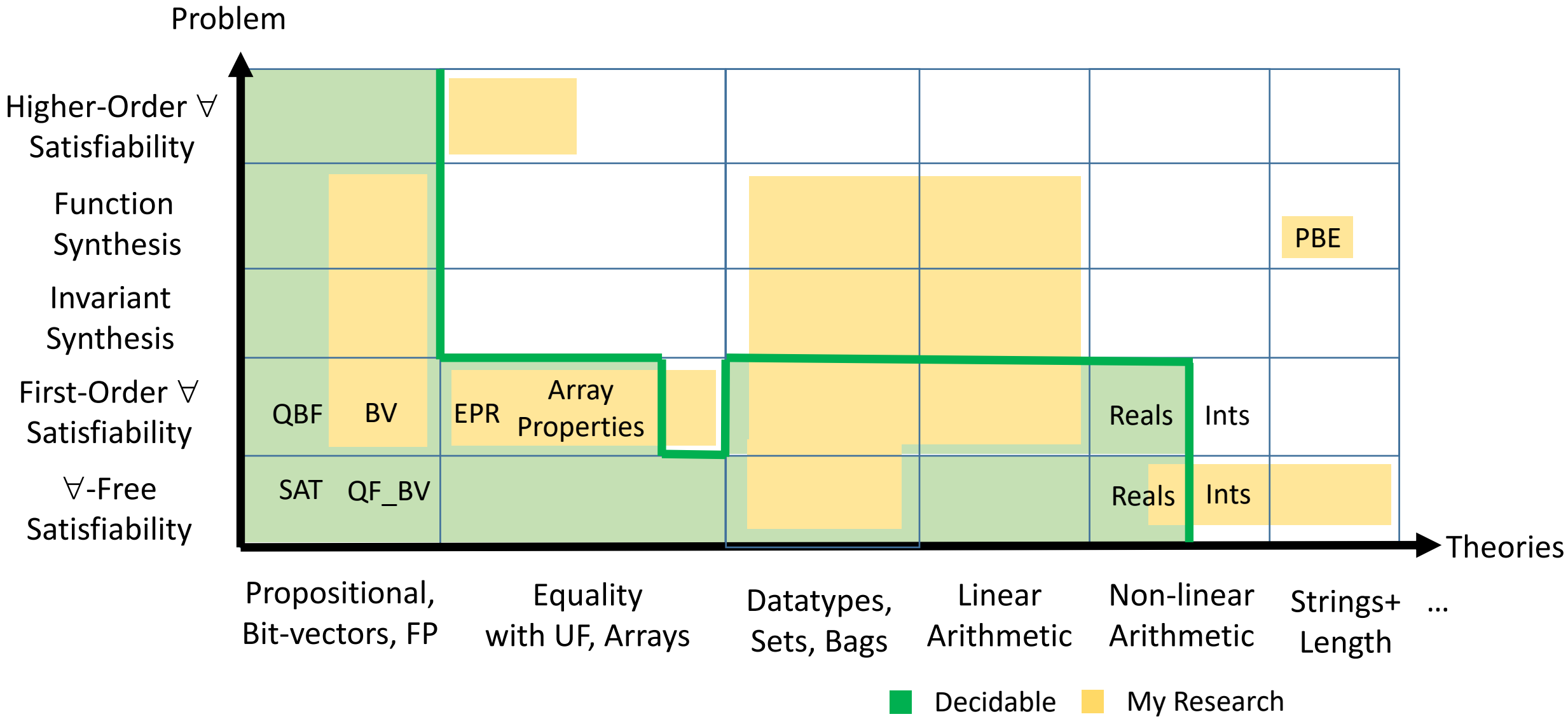
My Research

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

- Development of the SMT solver cvc5
  - Open source, available at : https://cvc5.github.io/

- Acknowledgements:
  - Cesare Tinelli, Clark Barrett, Haniel Barbosa, Andres Noetzli, Aina Niemetz, Mathias Preiner
  - Rest of cvc5 development team (past and present)
  - Viktor Kuncak

SMT Solvers: Standalone Tools for Many Problems



In this Talk:

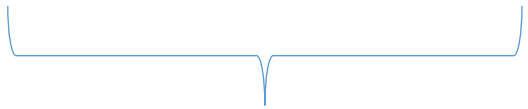
- Using *Syntax-Guided Synthesis* (SyGuS) to aid SMT solver development

- How we use SyGuS for developing cvc5:
  - Discovering rewrite rules **[Noetzli et al SAT 2019]**
  - Automatic test case generation
  - Discovering Invertibility Conditions for Bit-Vectors **[Niemetz et al CAV 2018]**
  - Discovering Invertibility Conditions for *Floating Points* **[Brain et al CAV 2019]**

How does Syntax-Guided Synthesis work?

Synthesis Conjectures

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



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`

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

syntax(f)

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

$\Rightarrow$ *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*

Solution Verifier

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

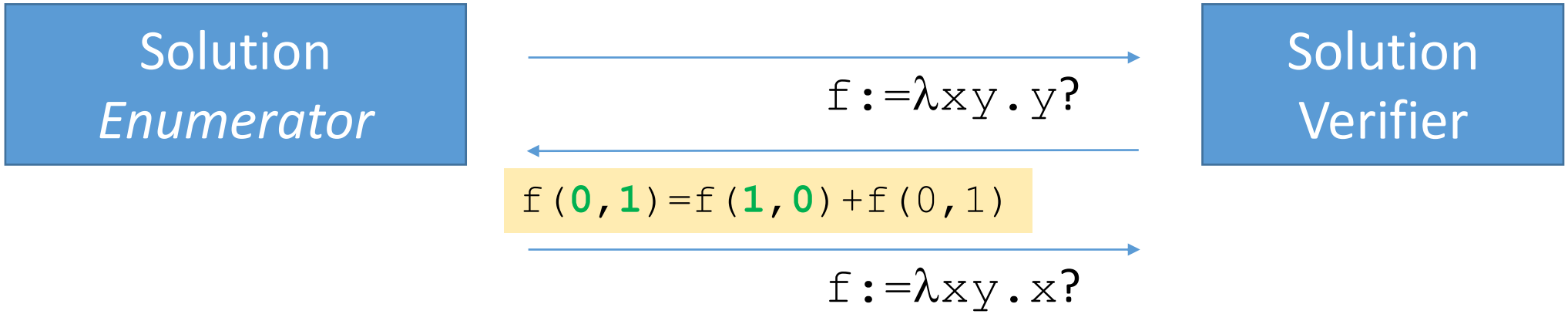
`f:=λxy.x?`

..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)
```

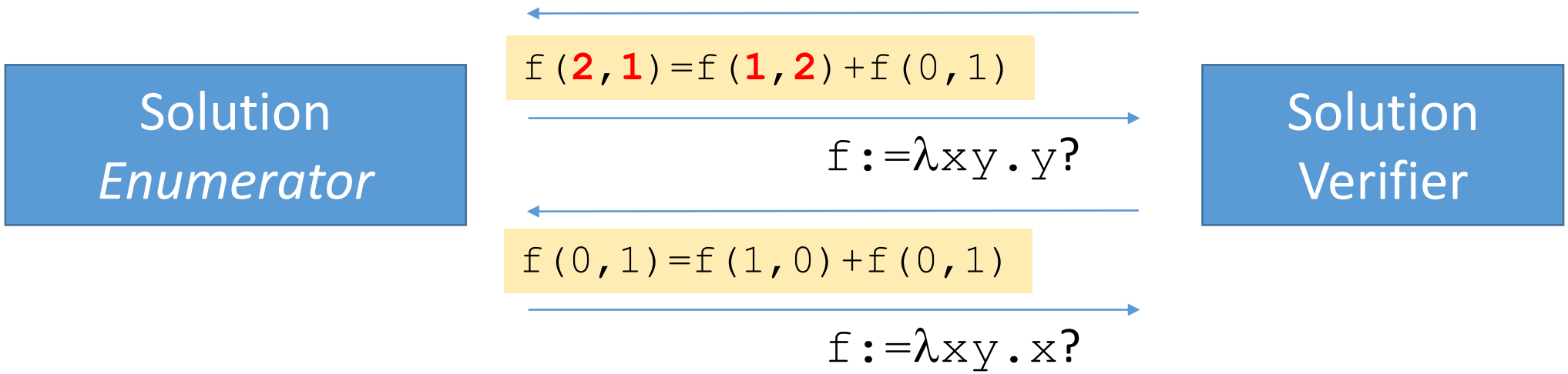


…new candidate λxy.y satisfies **(x,y)=(0,1)**

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)
```

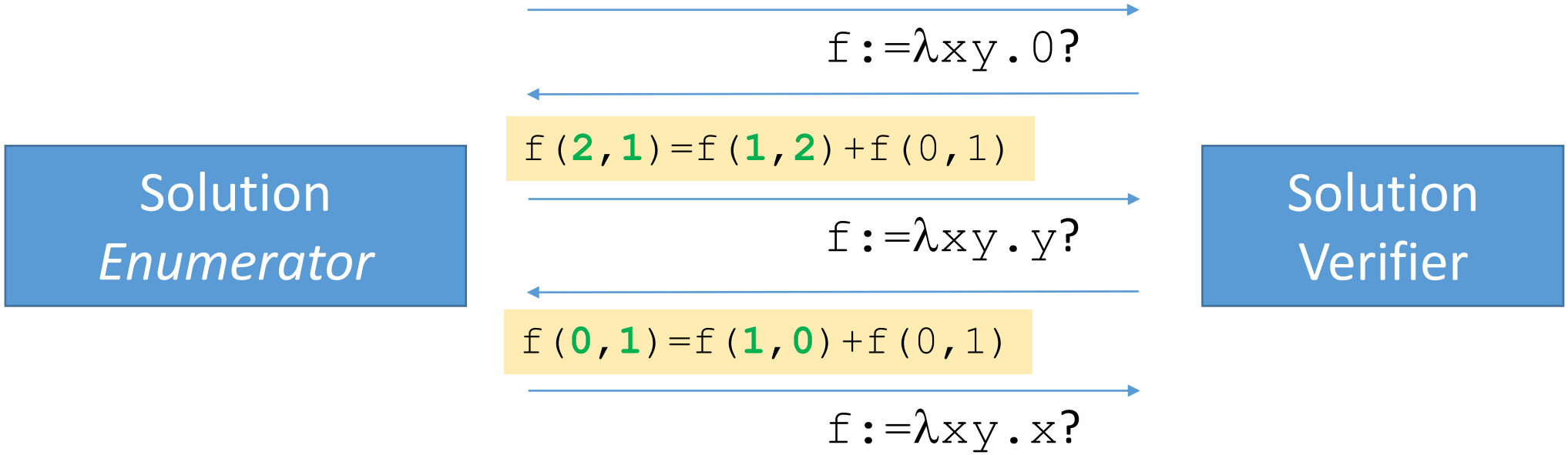


…counterexample: spec does not hold for **(x,y)=(2,1)**

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)
```

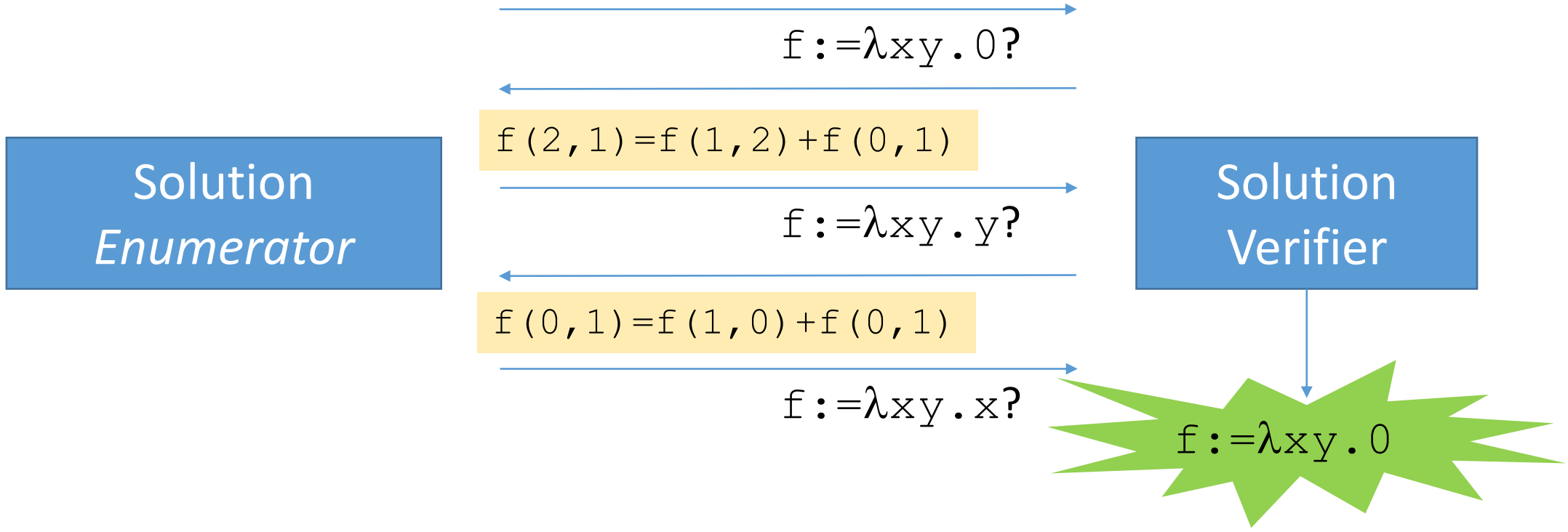


…new candidate λxy.0 satisfies **(x,y)=(0,1),(2,1)**

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)

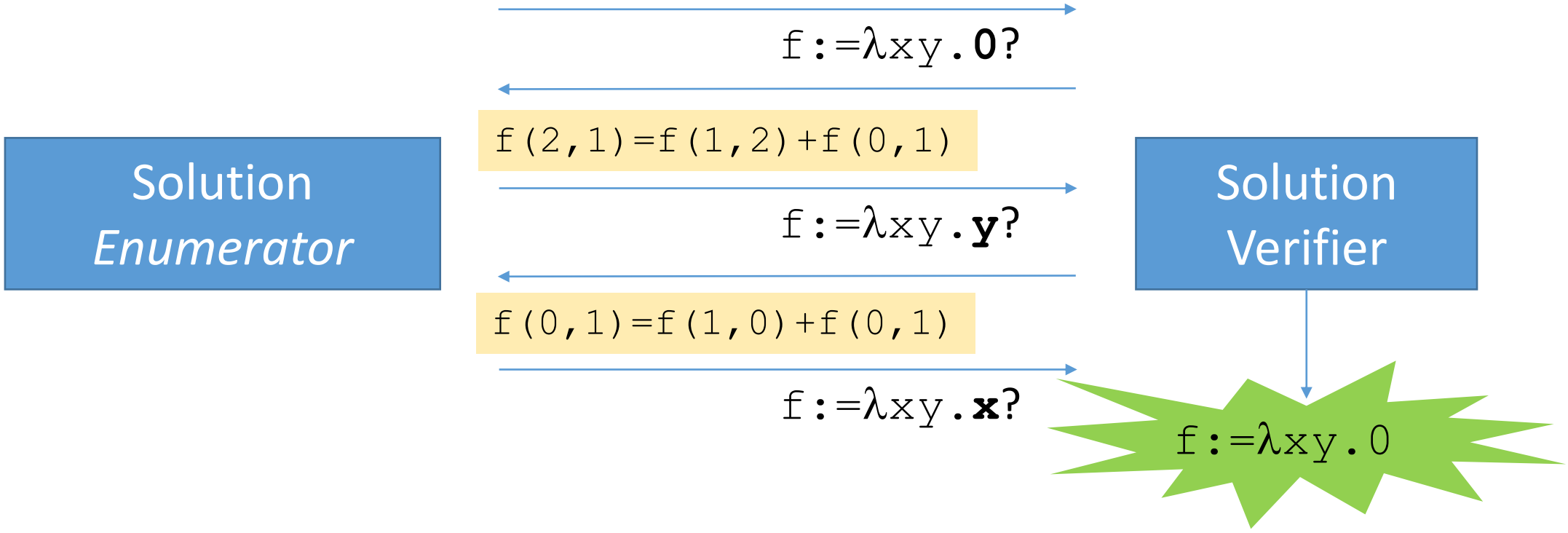


…new candidate λxy.0 has no counterexamples wrt **spec(f)**

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)**

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
(cvc5)

[Reynolds et al CAV 2015]

Solution Enumerator

Solution Verifier

Using an SMT Solver (cvc5) to *Improve Itself*

1. Rewrite Rule Synthesis

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

[Noetzli et al SAT 2019]

2. Semantics-guided test case generation

$$\exists Q . \exists_{\leq n} x \in S .\ Q(x)$$

3. Solving Quantified Bit-Vectors (and Floating Points) Using Invertibility Conditions

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

[Niemetz et al CAV 2018,
Brain et al CAV 2019]

…can tackle these problems using syntax-guided synthesis

Application #1:
Rewrite Rule Synthesis

Rewriting is Important for SMT Solving

- SMT solvers use a "rewriter" to put constraints in some normal form
  - E.g. `x+0`$\rightarrow$`x`, `x-y`$\rightarrow$`x+(-1*y)`, `x=x-2`$\rightarrow\bot$

- Having a good rewriter is highly critical to performance
  - In particular, theory of bit-vectors, strings, floating points
    - Single rewrite may make problem go from hard $\rightarrow$ trivial

Rewrite Rules are *Difficult to Implement*

- Hard to find commonly applicable rewrites
  - Analyze problem instances, solver runs
- What rewrites have I not already implemented?
- Time consuming, many lines of code
  - cvc5's BV rewriter ~3500 LOC
  - cvc5's string rewriter >5000 LOC

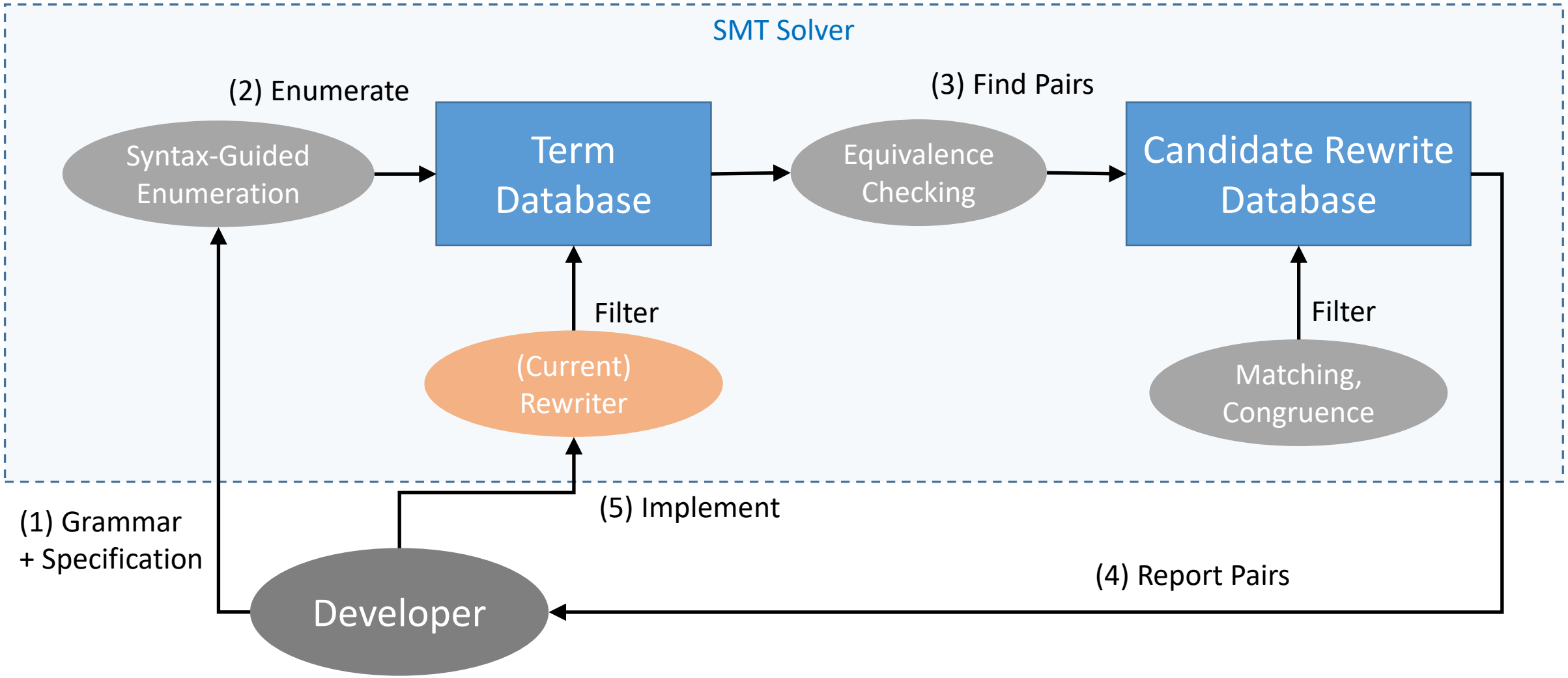
Rewrite Rule Synthesis

- Use the syntax-guided enumeration to assist the developer to implement the solver's rewriter

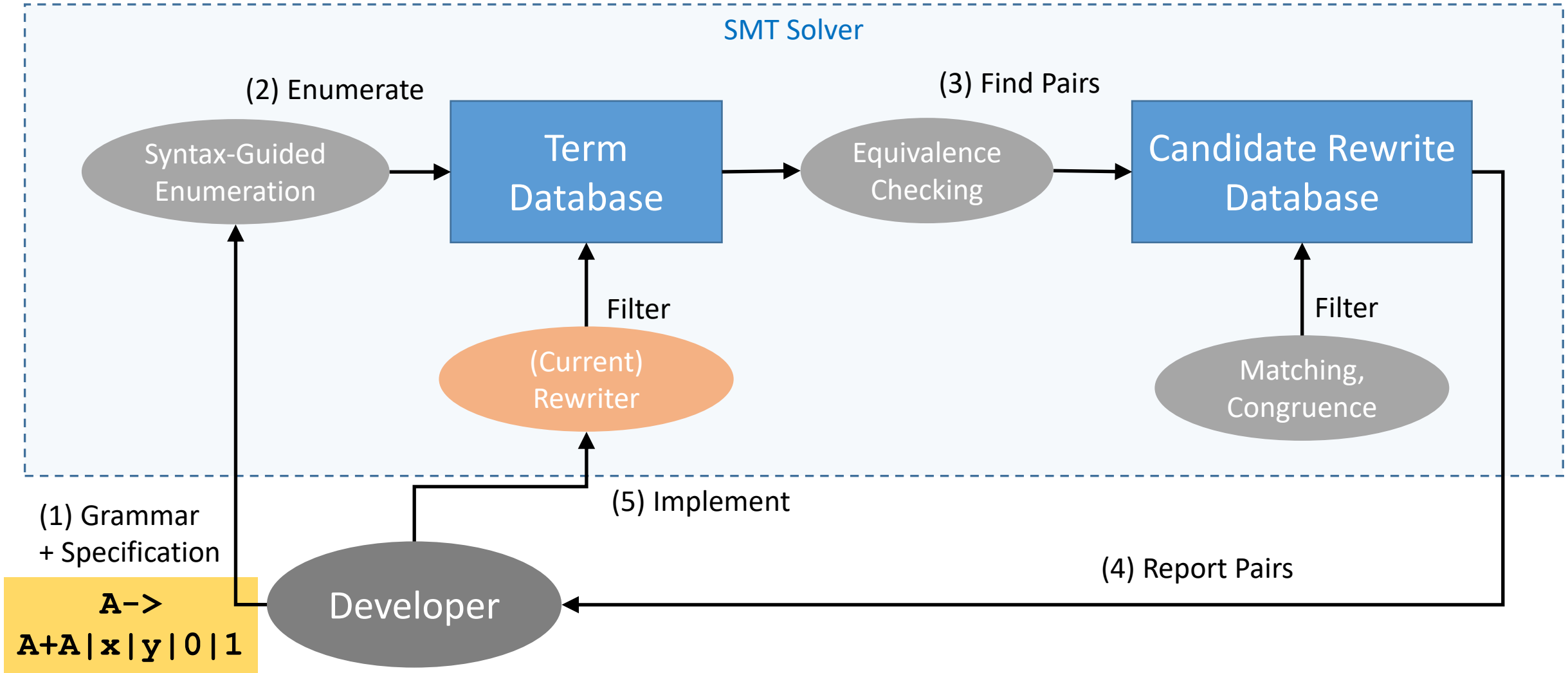
$\Rightarrow$Increase productivity of the developer

$\Rightarrow$Increase confidence in the correctness of the rewriter

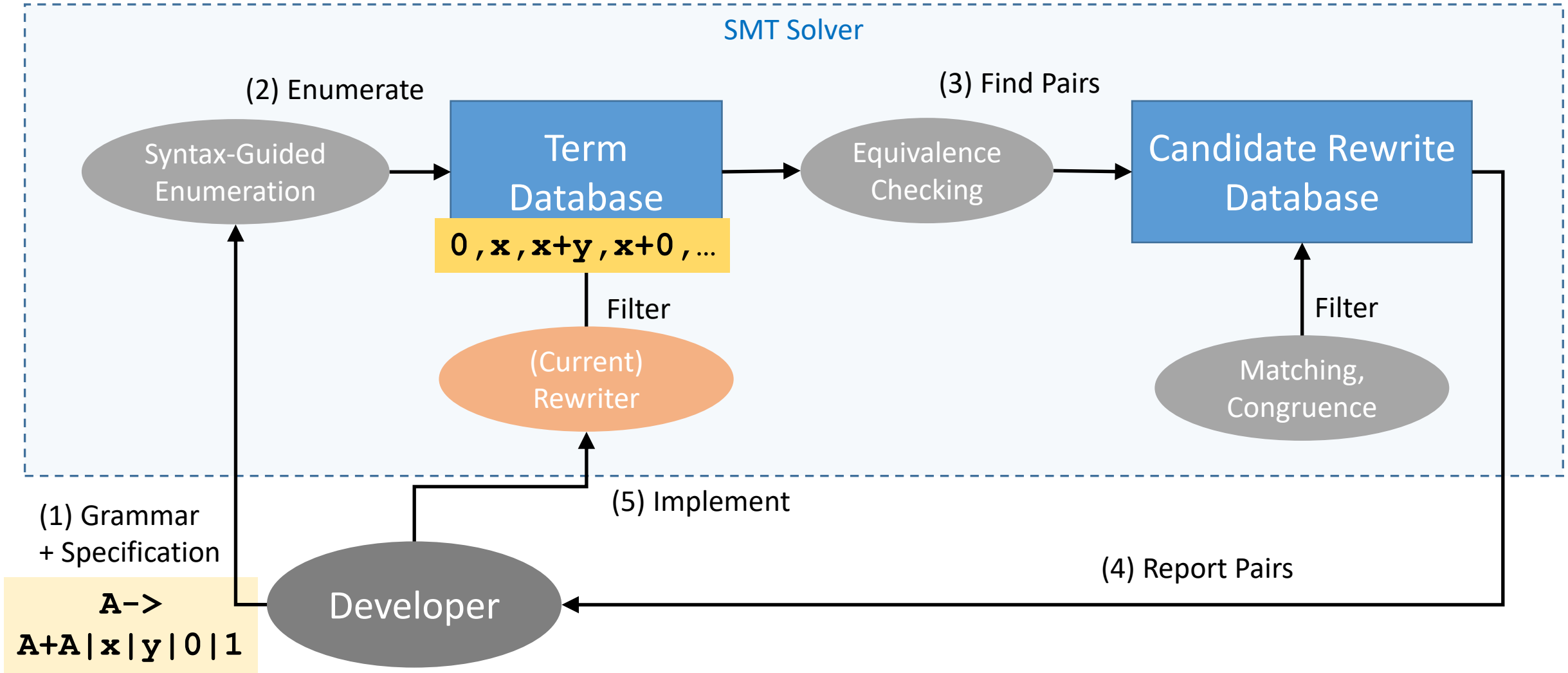
(Partially) Automated Rewrite Rule Generation



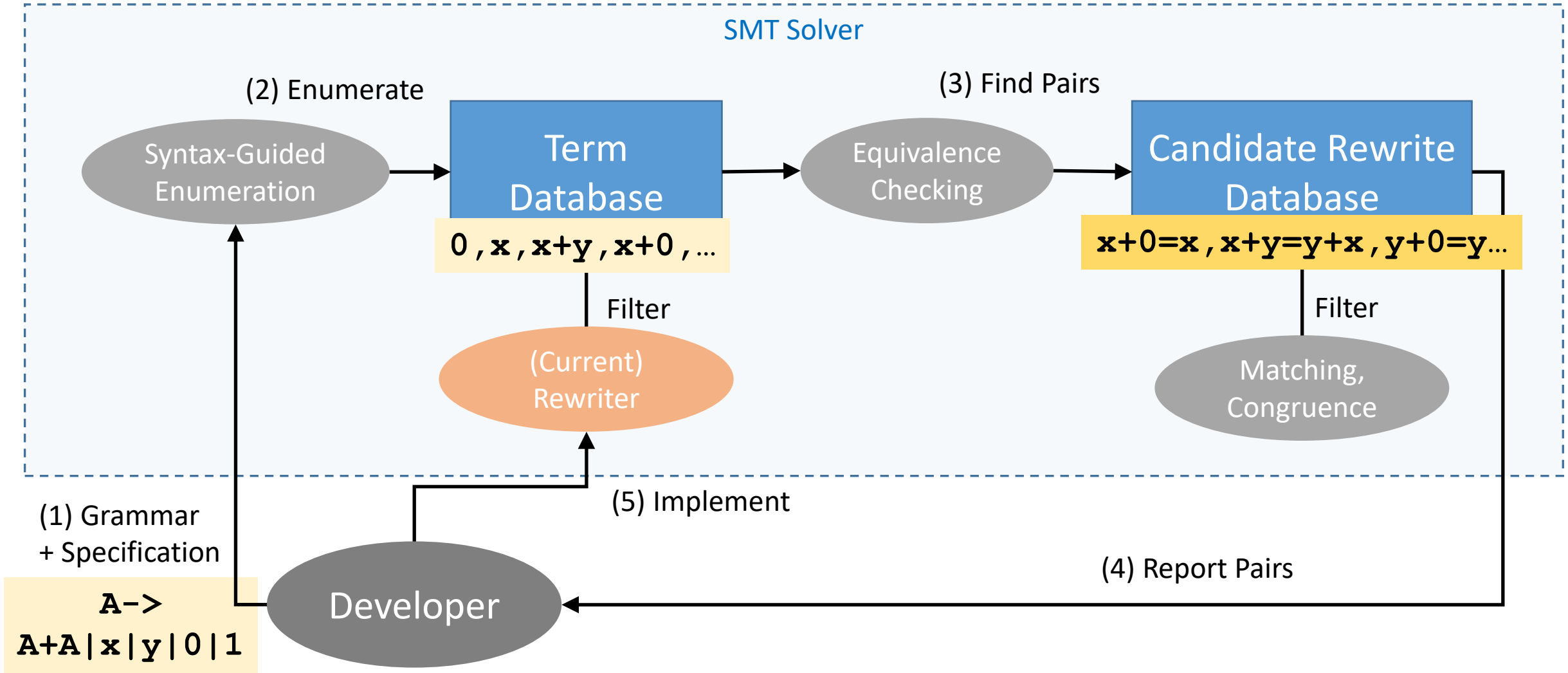
(Partially) Automated Rewrite Rule Generation



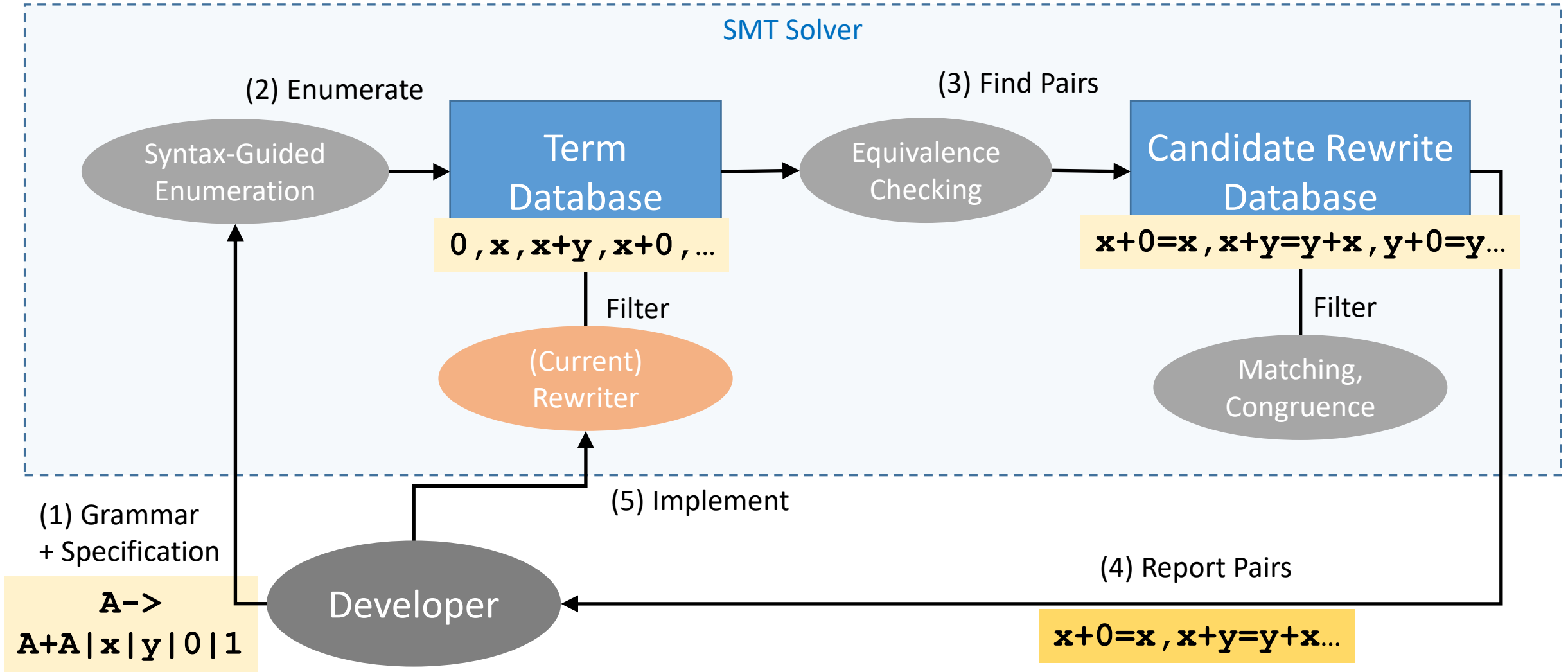
(Partially) Automated Rewrite Rule Generation



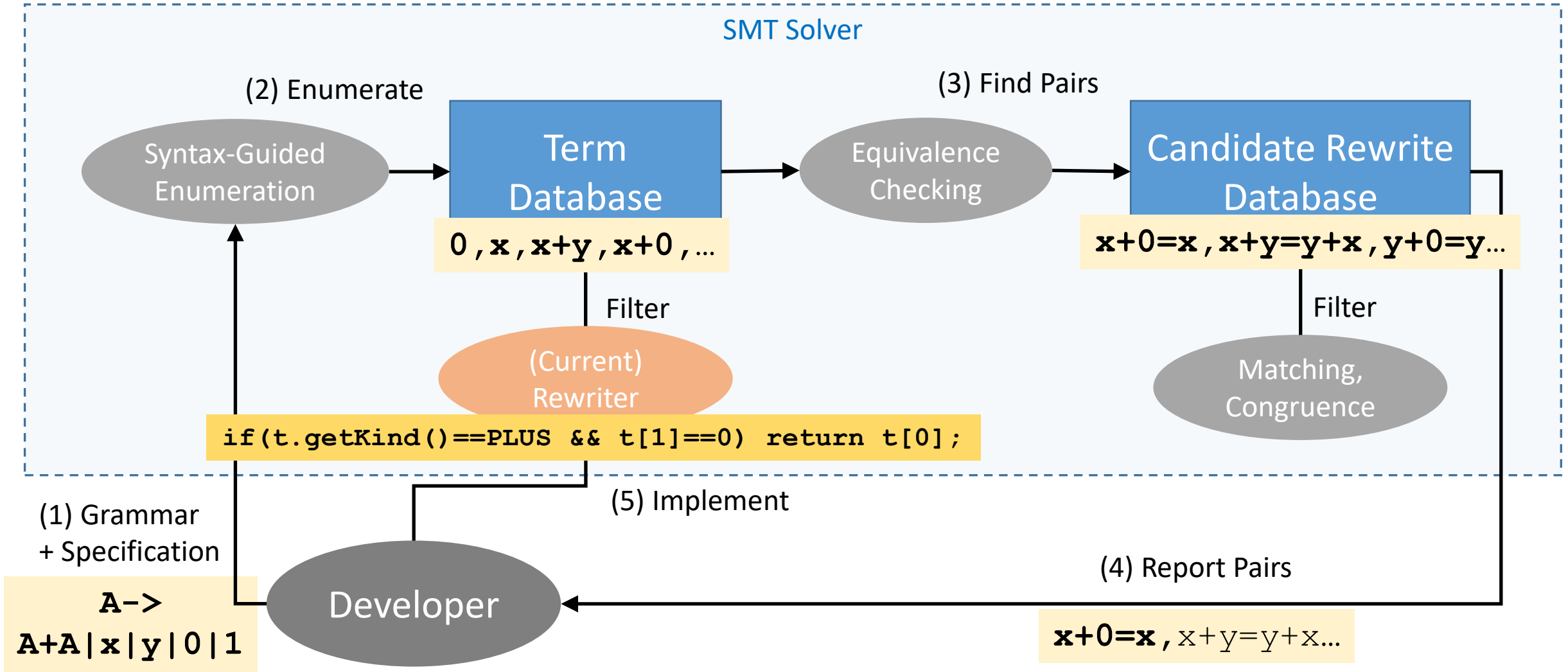
(Partially) Automated Rewrite Rule Generation



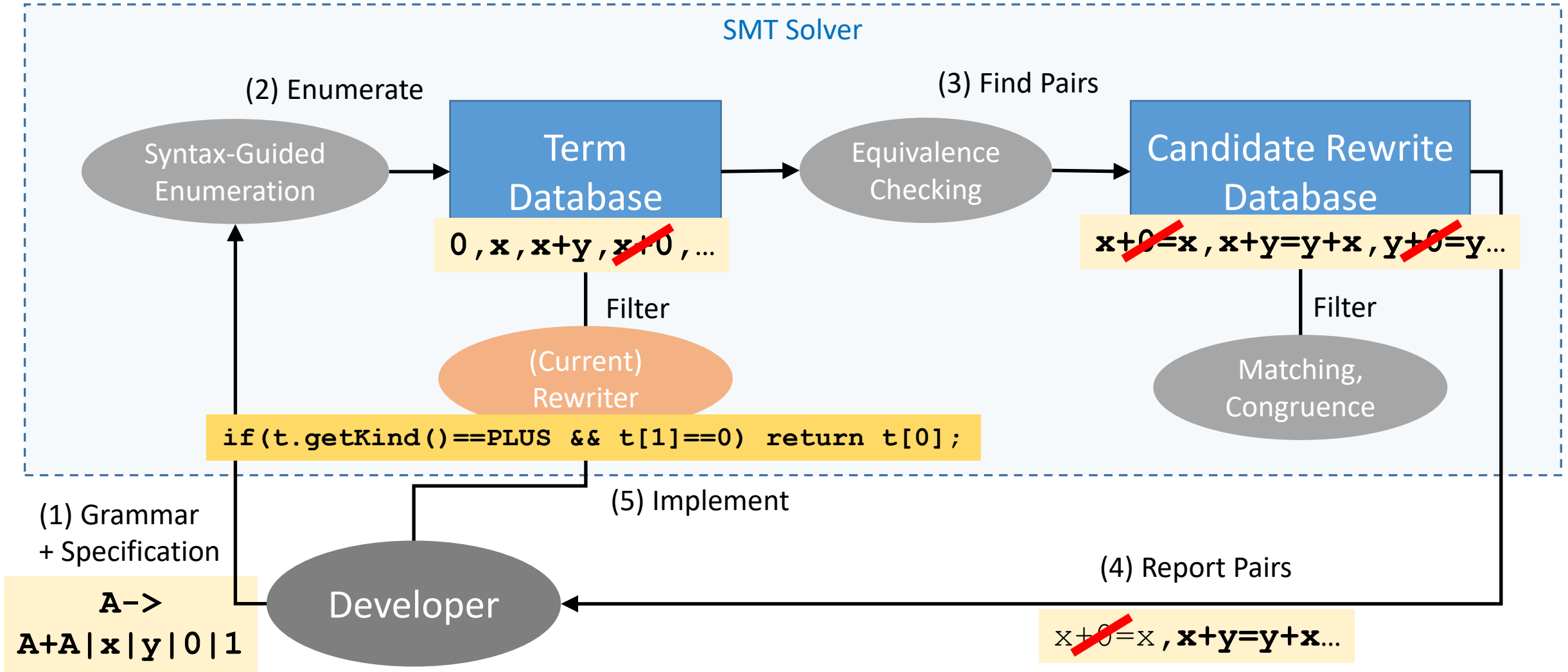
(Partially) Automated Rewrite Rule Generation



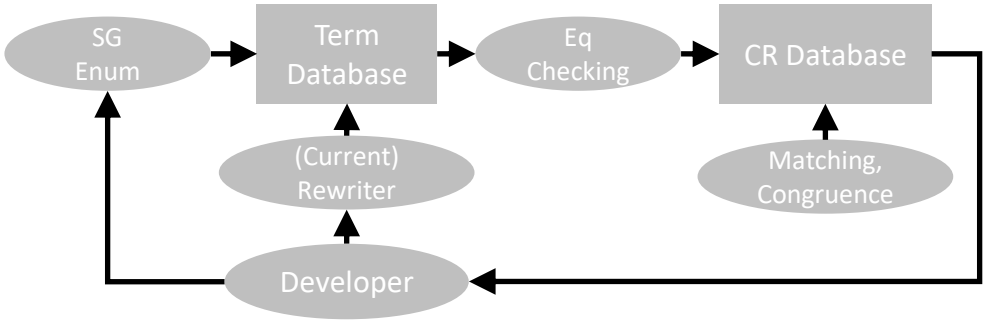
(Partially) Automated Rewrite Rule Generation



(Partially) Automated Rewrite Rule Generation



Experience

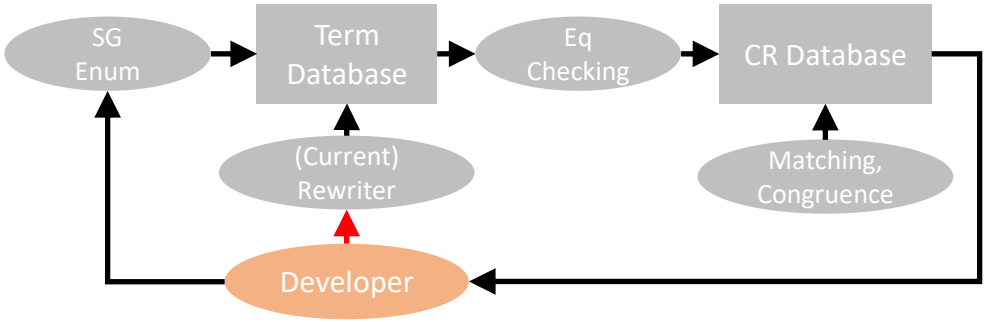



```
(synth-fun f
 ((x String) (y String) (z Int))
 String (
 (Start String (
  x y "A" "B" ""
  (str.++ Start Start)
  (str.replace Start Start Start)
  (str.at Start ie)
  (int.to.str ie)
  (str.substr Start ie ie)))
 (ie Int (
  0 1 z
  (+ ie ie)
  (- ie ie)
  (str.len Start)
  (str.to.int Start)
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

Examples of Rewrites



- 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,"")`

  `"A"++x="A"++y` → `x=y` `contains("ABCD","C"++x++"B")` → ⊥

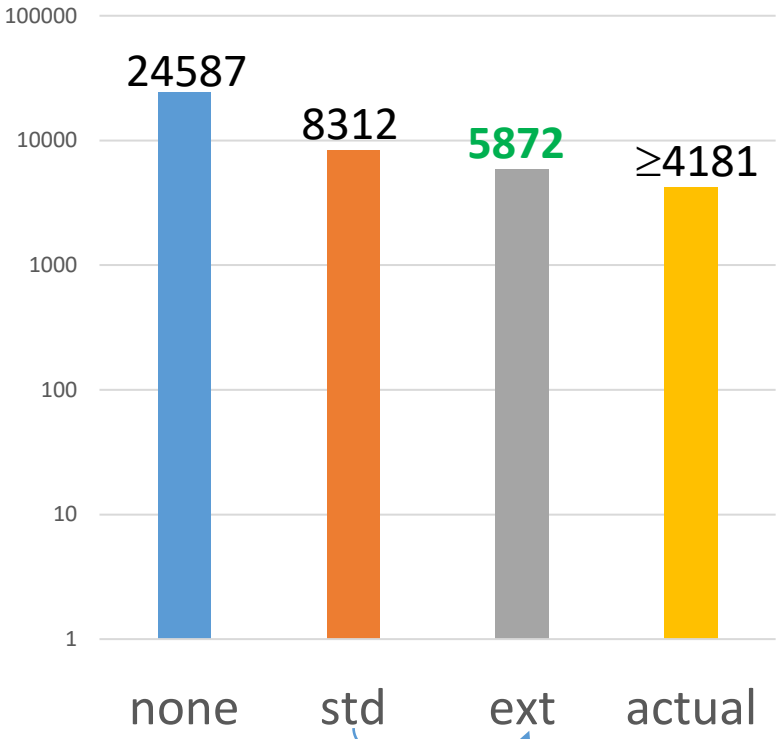
- Booleans

  A∧(A∨B) → A∧B (A∨C)∧(A∨B) → A∧(C∨B)

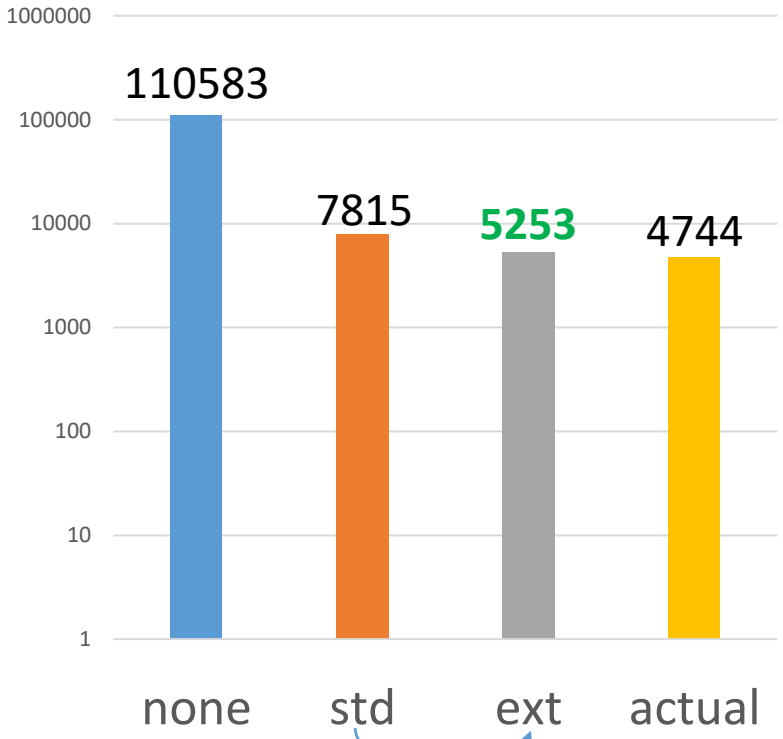
  A=A&B → ¬A∨B (A∨B)=(A∨B∨C) → A∨B∨¬C

Statistics: cvc5's Rewriter (circa 2019):

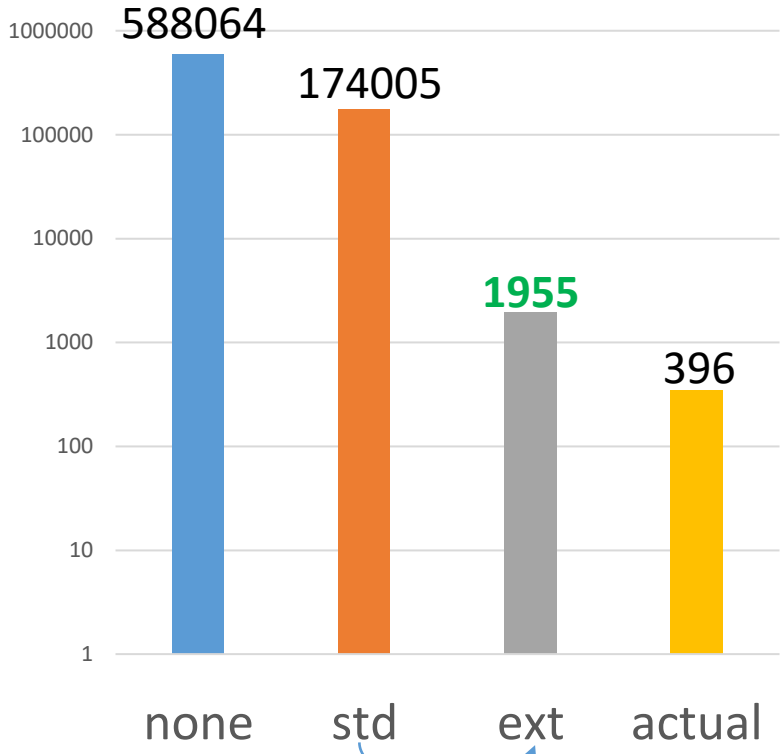
string-term, depth 2



bv-term, depth 3



bool-crci, depth 7



	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**

[Noetzli/Reynolds/Barbosa/Niemetz/Preiner/Barrett/Tinelli SAT 19]

Impact of Aggressive Simplification *for Strings*

Set		all	-arith	-contain	-msets	Z3	OSTRICH
CMU	sat	7947	7746	**7948**	7946	4585	
	unsat	**66**	31	**66**	**66**	52	
	×	173	409	172	174	3549	
TERMEQ	sat	**10**	**10**	**10**	**10**	1	
	unsat	**49**	36	27	**49**	36	
	×	22	35	44	22	44	
SLOG	sat	**1302**	**1302**	**1302**	**1302**	1100	1289
	unsat	**2082**	**2082**	**2082**	**2082**	2075	**2082**
	×	7	7	7	7	216	20
APLAS	sat	**132**	**132**	**132**	**132**	10	
	unsat	**292**	291	171	171	94	
	×	159	160	280	280	479	
Total	sat	9391	9190	**9392**	9390	5696	1289
	unsat	**2489**	2440	2346	2368	2257	2082
	×	361	611	503	483	4288	8870

[Reynolds/Noetzli/Tinelli/Barrett CAV 19]

-arith: w/o arithmetic simplifications
-contain: w/o contain-based simplifications
-mset: w/o multiset-based simplifications

- cvc5 uses >5000 lines of C++ for simplification rules (and growing)
  - Important aspect of modern string solving

Application #2: Improving Confidence in the SMT Solver

Improving Confidence: Rewriting

- *Does my SMT solver implement any unsound rewrites?*

Improving Confidence: Rewriting

Grammar of interest

```
(synth-fun f ((s BV8) (t BV8)) BV8
   ((Start BV8
      (s t #x00 #x01
       ~Start
       -Start
       bvlshr(Start Start)
       Start & Start
       Start + Start
   )))
```

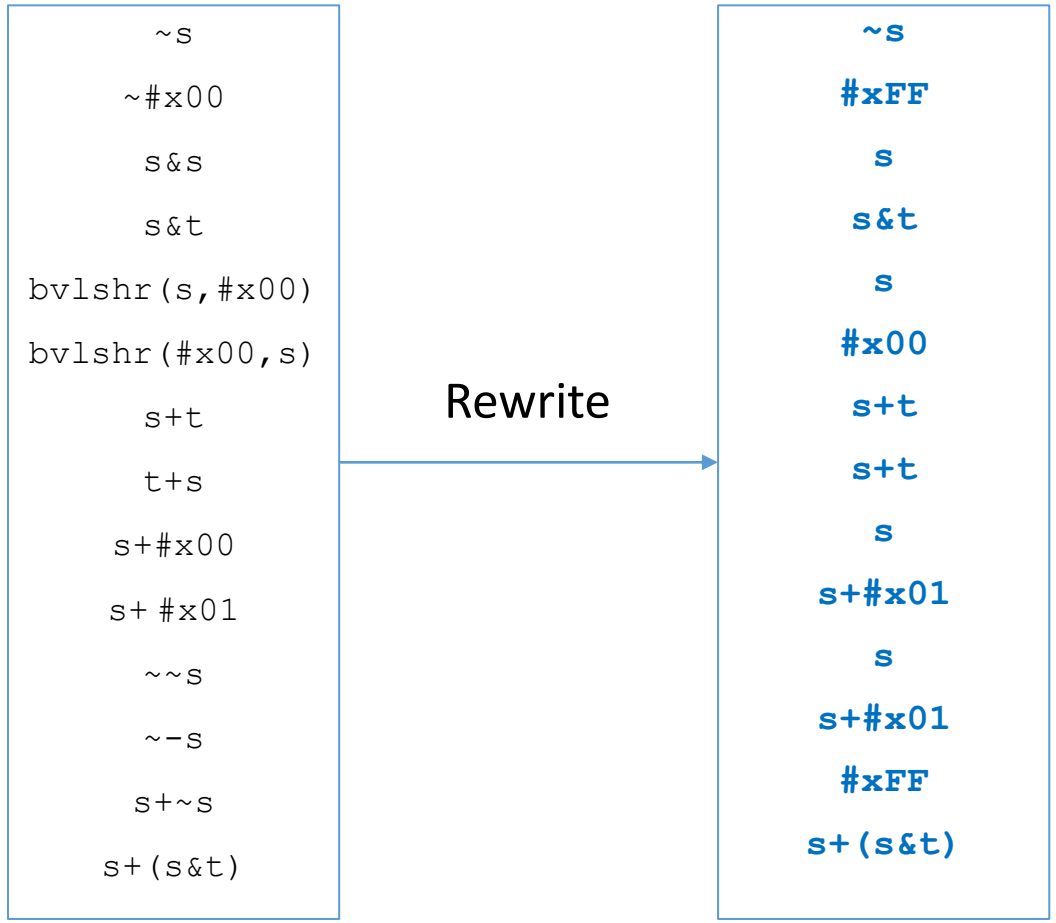
Improving Confidence: Rewriting

```
~s
~#x00
s&s
s&t
bvlshr(s,#x00)
bvlshr(#x00,s)
s+t
t+s
s+#x00
s+#x01
~~s
~-s
s+~s
s+(s&t)
```

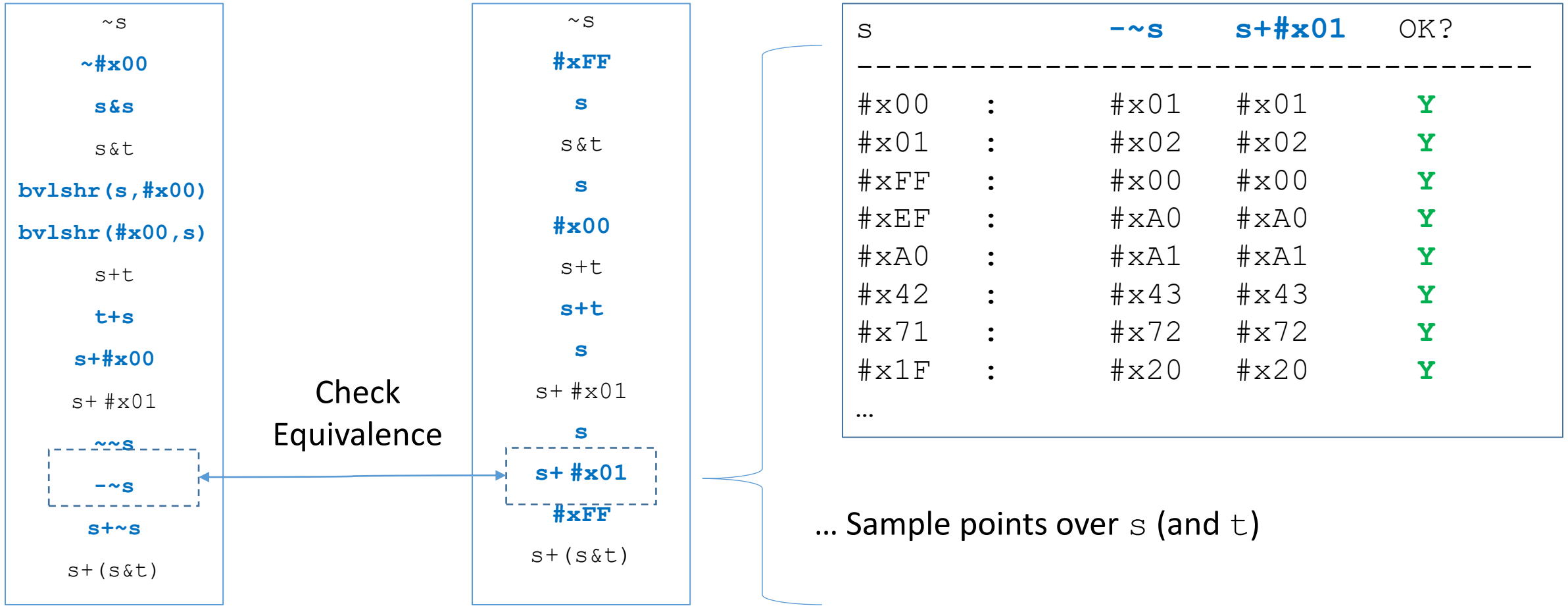
Syntax-Guided Term Enumeration

```
(synth-fun f ((s BV8) (t BV8)) BV8
   ((Start BV8
      (s t #x00 #x01
       ~Start
       -Start
       bvlshr(Start Start)
       Start & Start
       Start + Start
   )))
```

Improving Confidence: Rewriting



Improving Confidence: Rewriting

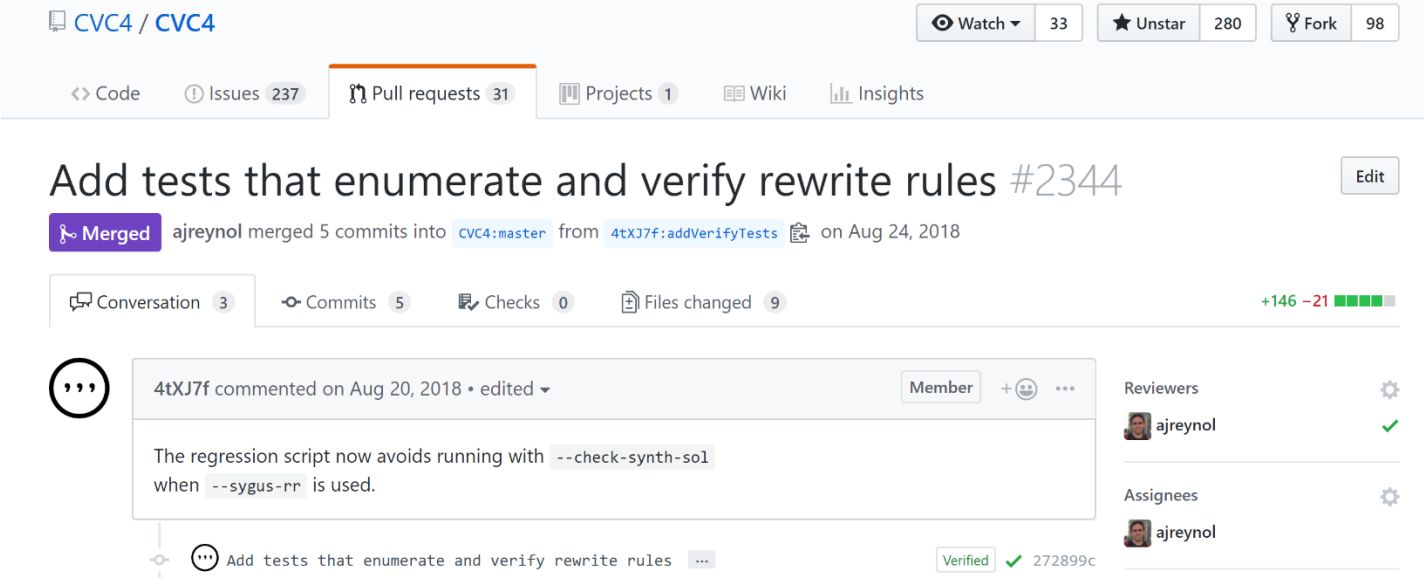


Improving Confidence: Rewriting

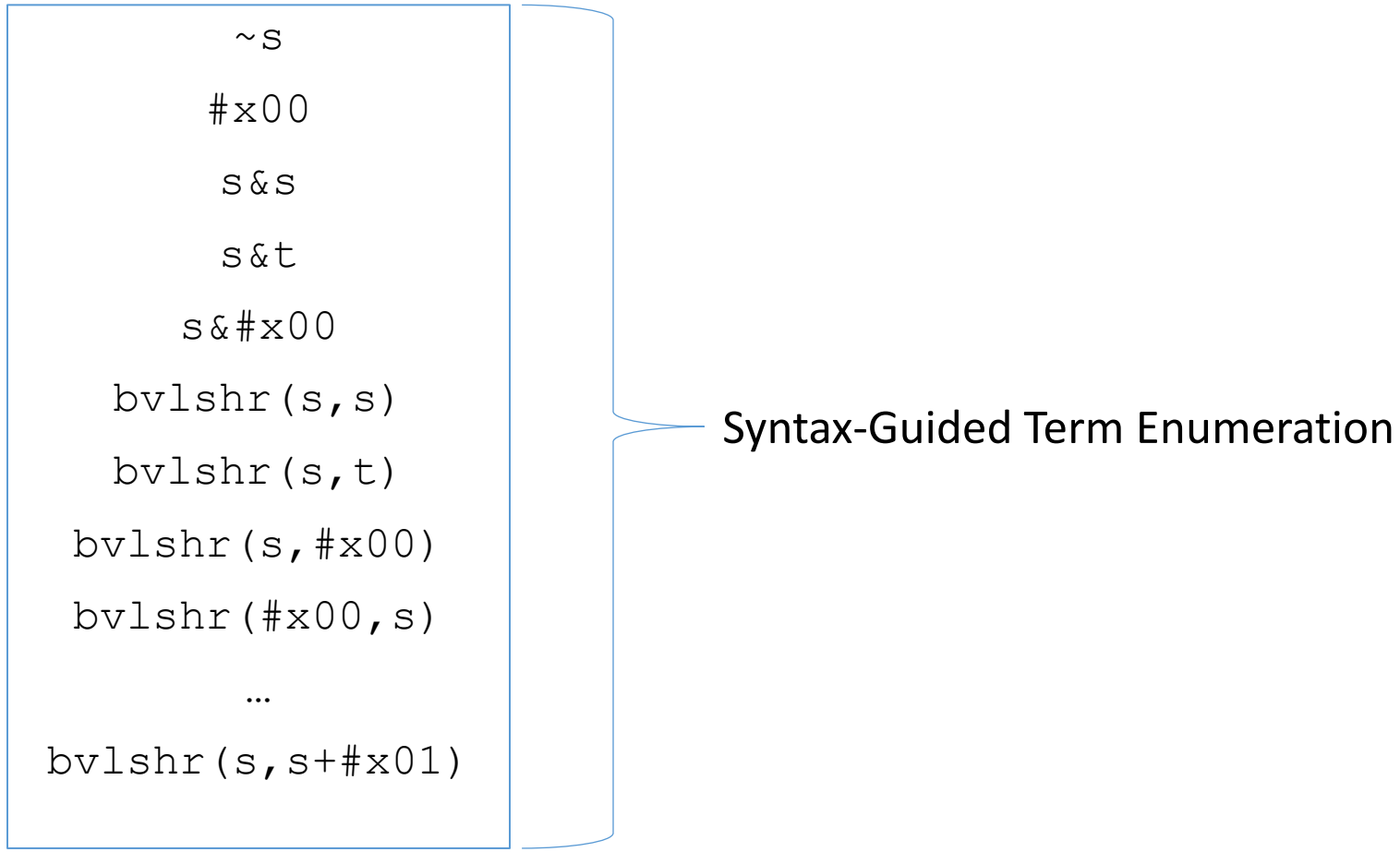
⇒Has been critical for *finding bugs* in newly written rewriter code

```
(unsound-rewrite (bvuge (bvadd x #x0001) x) true)
; --sygus-rr-verify detected unsoundness in the rewriter!
; Terms have the same rewritten form but are not equivalent
; for x=#xFFFF, where they evaluate to:
; (bvuge (bvadd x #x0001) x) = false
; true = true
```

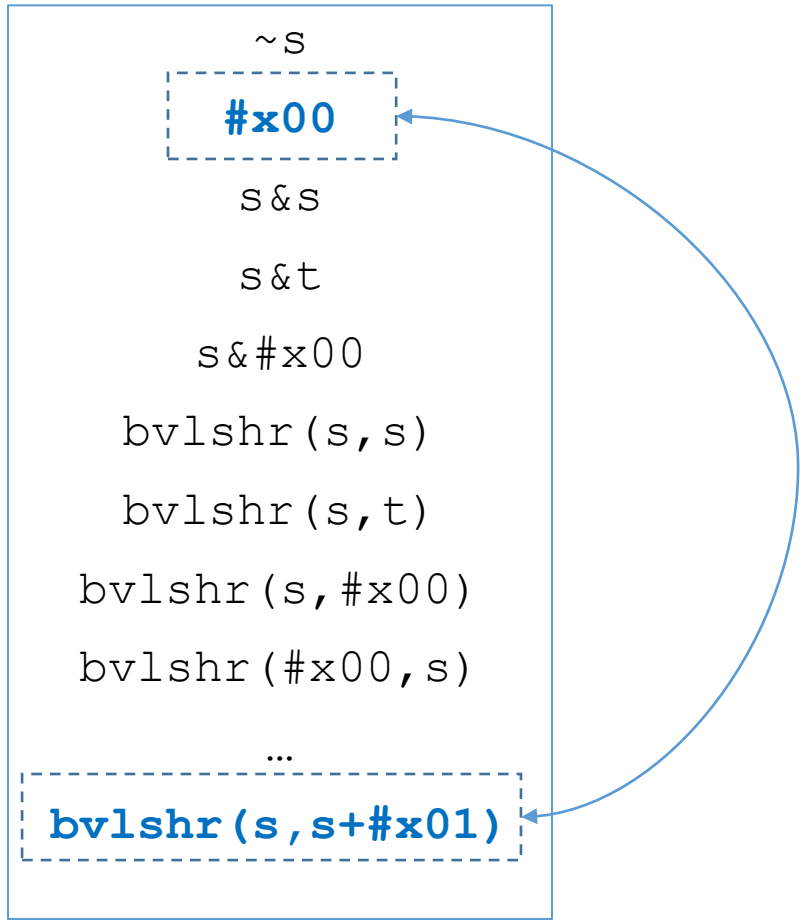
⇒Run as part of cvc5's regression tests:



Improving Confidence: Query Generation



Improving Confidence: Query Generation

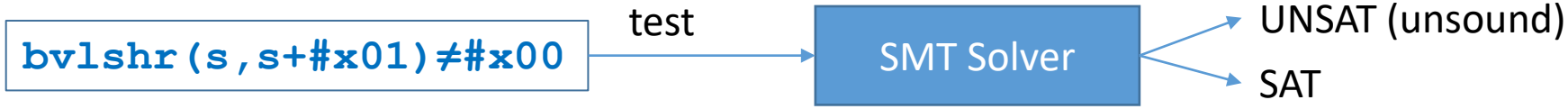


Check Equivalence

s		#x00	bvlshr(s,s+#x01)	=?
#x00	:	#x00	#x01	Y
#x01	:	#x00	#x00	Y
#xFF	:	#x00	#xFF	N
#xEF	:	#x00	#x00	Y
#xA0	:	#x00	#x00	Y
#x42	:	#x00	#x00	Y
#x71	:	#x00	#x00	Y
#x1F	:	#x00	#x00	Y

…

- **Intuition:** terms that are “almost equivalent” => inputs that are “almost unsatisfiable”



Application #3: Solving Quantified Inputs via Invertibility Conditions

Solving Quantified Bit-Vectors: Example

- Consider quantifier elimination problem for bit-vectors:

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

$\Rightarrow$ Applications in proving safety properties, compiler optimzations

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

- **Idea:** Use SyGuS to find *invertibility conditions* for bit-vector equations

  $\Rightarrow$ Compact form for quantifier elimination

What is an Invertibility Condition?

- 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]

What is an Invertibility Condition?

- `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" formulas

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

What is an Invertibility Condition?

- For terms `s`, `t`, operator $\oplus$ and relation ~
  - An *invertibility condition* `IC(s,t)` is a quantifier-free formula such that:

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

    is T-valid.

Finding Invertibility Conditions for BV

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

- Applied to theory of Bit-Vectors **[Niemetz et al CAV 2018]**
- Signature of BV has 15 operators, 4 relations (=/≠, signed/unsigned inequality)
  $\Rightarrow$ Total of 162 invertibility conditions to find
- Some took several hours to find by hand $\therefore$ *use SyGuS*
  - Using SyGuS, found **118** of **162** conditions
    - Many simpler than hand-crafted ones
  - When combined with hand-crafted ICs, found all **162** conditions

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 $\mid$ 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]

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.

Invertibility Conditions for Floating Points

- Say I want to find invertibility condition for *floating point* equation:

$$\exists x : FP_{e,s} . x \oplus s \sim t$$

- Can we apply the same approach to Floating Points?
  - …will it scale?

Invertibility Conditions for Floating Points

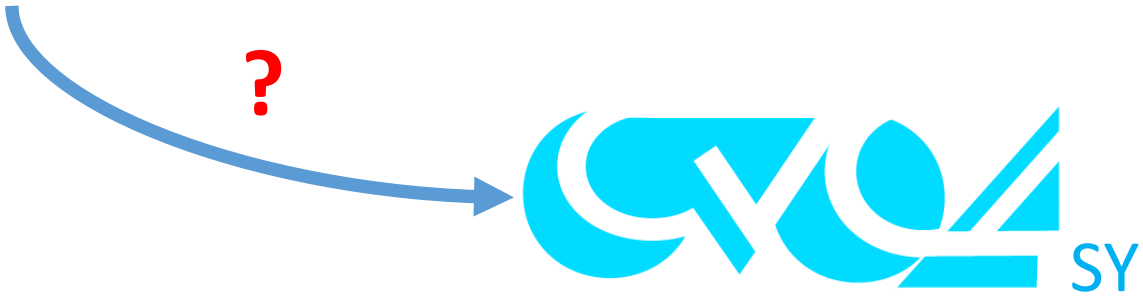
$$\exists \mathbf{IC}.\ \forall st{:}FP_{e,s}.\ (\exists x{:}FP_{e,s}.x \oplus s \sim t) \Leftrightarrow \mathbf{IC}(s,t)$$

- Can express as a synthesis problem:

  …there exists a predicate **IC** that is equivalent to $\exists x{:}FP_{e,s}.x \oplus s \sim t$

Invertibility Conditions for Floating Points

$$\exists IC.\ \forall st{:}FP_{e,s}.\ (\exists x{:}FP_{e,s}.x \oplus s \sim t) \Leftrightarrow IC(s,t)$$



… solve via syntax-guided synthesis (SyGuS)?

How do we find Invertibility Conditions?

$$\exists IC.\ \forall st{:}FP_{\mathbf{e},\mathbf{s}}.\ (\exists x{:}FP_{\mathbf{e},\mathbf{s}}.x \oplus s \sim t) \Leftrightarrow IC(s,t)$$

- **Challenge #1:** problem is parameteric
  - Parameterized by # exponent bits **e**, # significant bits **s**

How do we find Invertibility Conditions?

$$\exists IC.\ \forall st{:}FP_{3,5}.\ (\exists x{:}FP_{3,5}.x \oplus s \sim t) \Leftrightarrow IC(s,t)$$

- **Challenge #1:** problem is parameteric

$\Rightarrow$ Solve for a *fixed format*, check if solution generalizes **[Niemetz et al CAV 2018]**

- Choose e=**3**, s=**5** … large enough to exhibit FP behaviors, small enough for synthesis to scale

How do we find Invertibility Conditions?

$$\exists IC.\ \forall st{:}FP_{3,5}.\ (\exists x{:}FP_{3,5}.x \oplus s \sim t) \Leftrightarrow IC(s,t)$$

How do we find Invertibility Conditions?

$$\exists IC.\ \forall st{:}FP_{3,5}.\ (\exists x{:}FP_{3,5}.x \oplus s \sim t) \Leftrightarrow IC(s,t)$$

- **Challenge #2:** problem has *three levels* of quantifier alternation
  - …SyGuS solvers only typically handle two

How do we find Invertibility Conditions?

$$\exists IC.\ \forall st{:}FP_{3,5}.\ \Big(\bigvee_{c=1}^{227} c \oplus s \sim t \Big) \Leftrightarrow IC(s,t)$$

- **Challenge #2:** problem has three levels of quantifier alternation

$\Rightarrow$ Expand the innermost quantifier **[Niemetz et al CAV 2018]**

- Since domain is (relatively) small and finite
  - Only 227 values for $FP_{3,5}$

How do we find Invertibility Conditions?

$$\exists IC.\ \forall st{:}FP_{3,5}.\ \Big(\bigvee_{c=1}^{227} c \oplus s \sim t \Big) \Leftrightarrow IC(s,t)$$

…can now give as input to SyGuS solver

How do we find Invertibility Conditions?

$$\exists IC.\ \forall st{:}FP_{3,5}.\ \left(\bigvee_{c=1}^{227} c \oplus s \sim t \right) \Leftrightarrow IC(s,t)$$

- **Challenge #3:** Scalability!
  - With respect to BV synthesis @ 4 bits **[Niemetz et al CAV 2018]**:
    - Roughly 14 times more constraints, q.f. FP is much harder than BV

How do we find Invertibility Conditions?

$$\exists IC.\ \forall st{:}FP_{3,5}.\ \Big(\bigvee_{c=1}^{227} c \oplus s \sim t \Big) \Leftrightarrow IC(s,t)$$

- **Challenge #3:** Scalability

How do we find Invertibility Conditions?

$$\exists IC.\ \bigwedge_{s=1}^{227} \bigwedge_{t=1}^{227} c_{s,t} \Leftrightarrow IC(s,t)$$

$\exists x.x*s=t?$ where s, t are constants

- **Challenge #3:** Scalability

  $\Rightarrow$Statically compute the I/O behavior of *entire domain* of $IC(s,t)$

  - Each $c_{s,t}$ is determined by a quantifier-free satisfiability query
    - 227*227 = 51529, takes ~10 minutes

How do we find Invertibility Conditions?

$$\exists IC. \quad \bigwedge_{s=1}^{227} \bigwedge_{t=1}^{227} c_{s,t} \Leftrightarrow IC(s,t)$$

How do we find Invertibility Conditions?

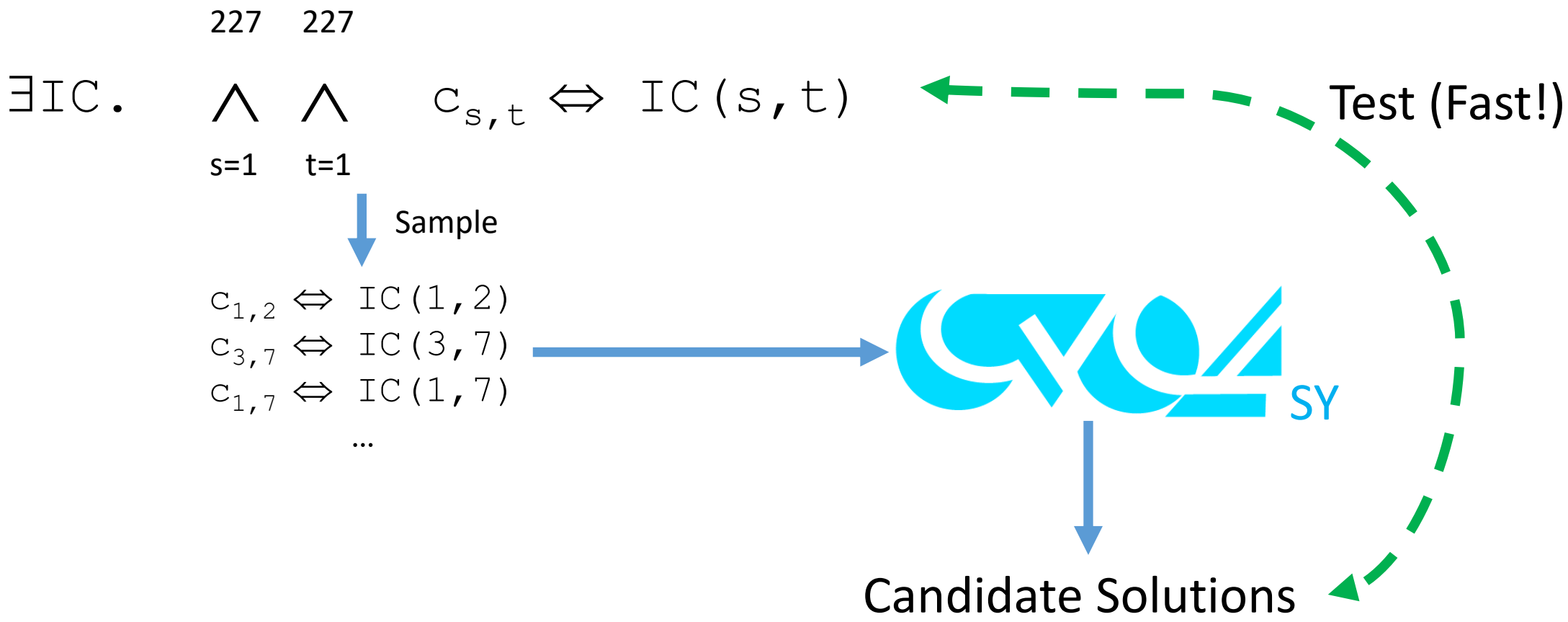
$$\exists IC. \quad \bigwedge_{s=1}^{227} \bigwedge_{t=1}^{227} c_{s,t} \Leftrightarrow IC(s,t)$$

Sample

$c_{1,2} \Leftrightarrow IC(1,2)$
$c_{3,7} \Leftrightarrow IC(3,7)$
$c_{1,7} \Leftrightarrow IC(1,7)$
…

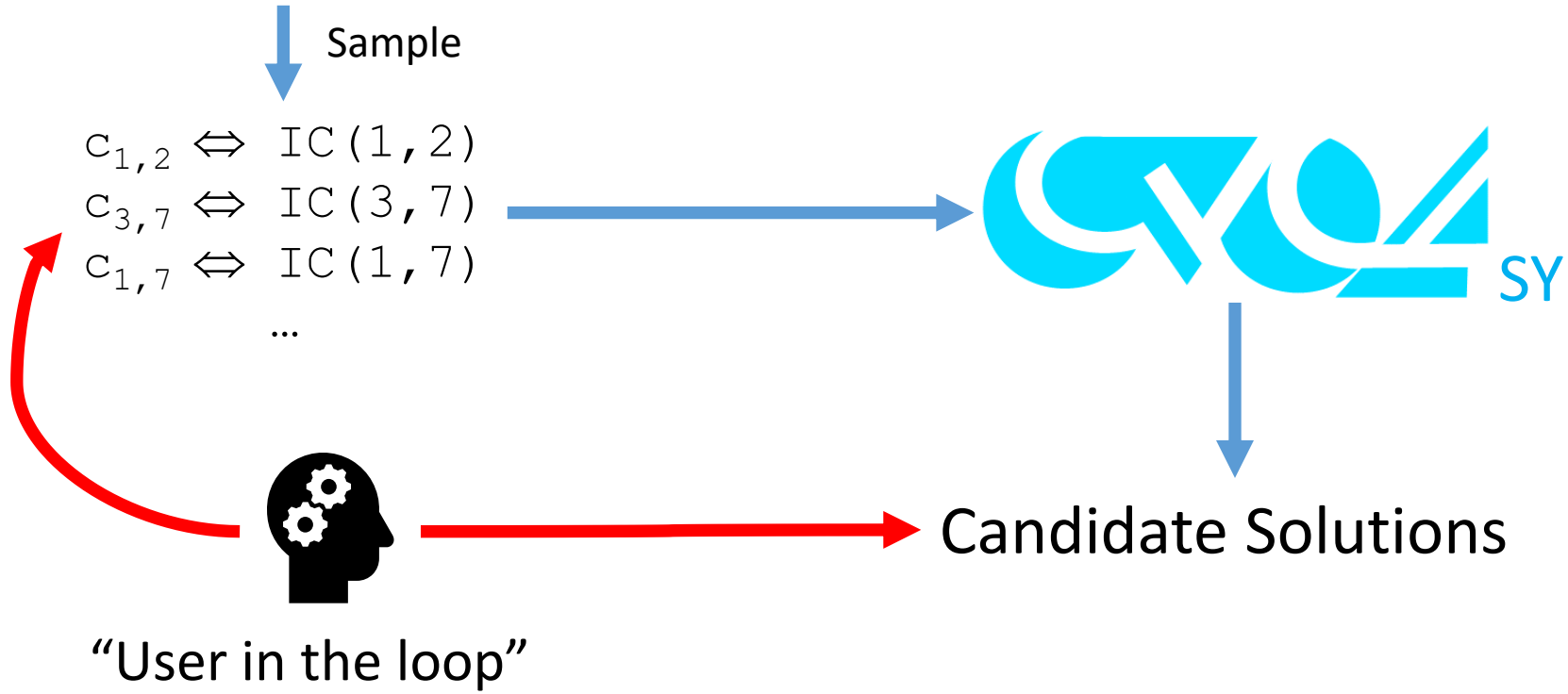
Candidate Solutions

How do we find Invertibility Conditions?



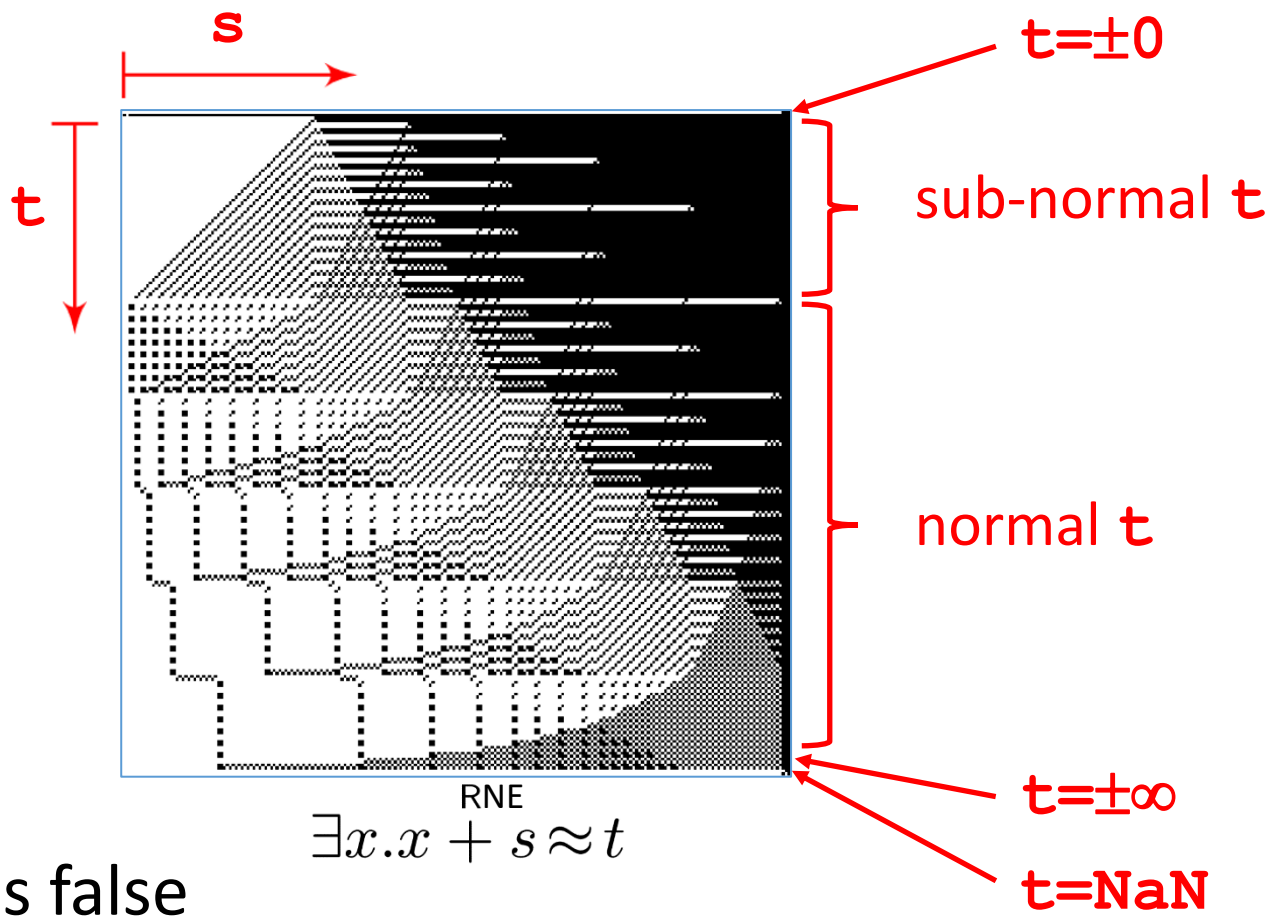
How do we find Invertibility Conditions?

$$\exists IC.\ \bigwedge_{s=1}^{227} \bigwedge_{t=1}^{227} c_{s,t} \Leftrightarrow IC(s,t)$$



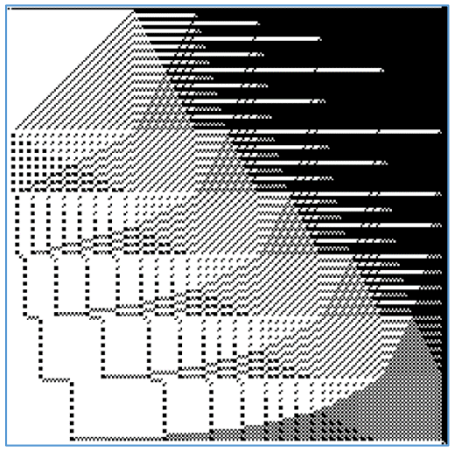
Visualizing I/O Specs of Invertibility Conditions

$$\exists \mathtt{IC}.\ \bigwedge_{s=1}^{227} \bigwedge_{t=1}^{227} c_{s,t} \Leftrightarrow \mathtt{IC}(s,t)$$



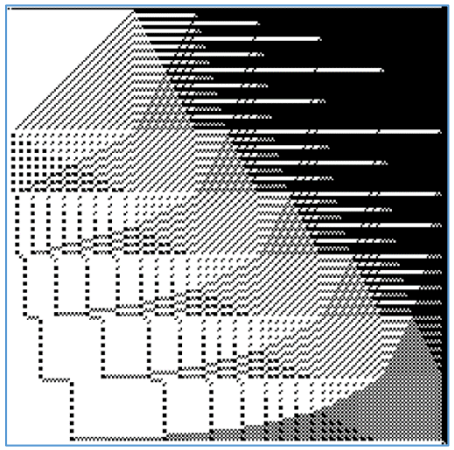
- White = IC is true, Black = IC is false
  - ⇒ Behavior of IC is highly complex!

Synthesizing Invertibility Conditions



SY

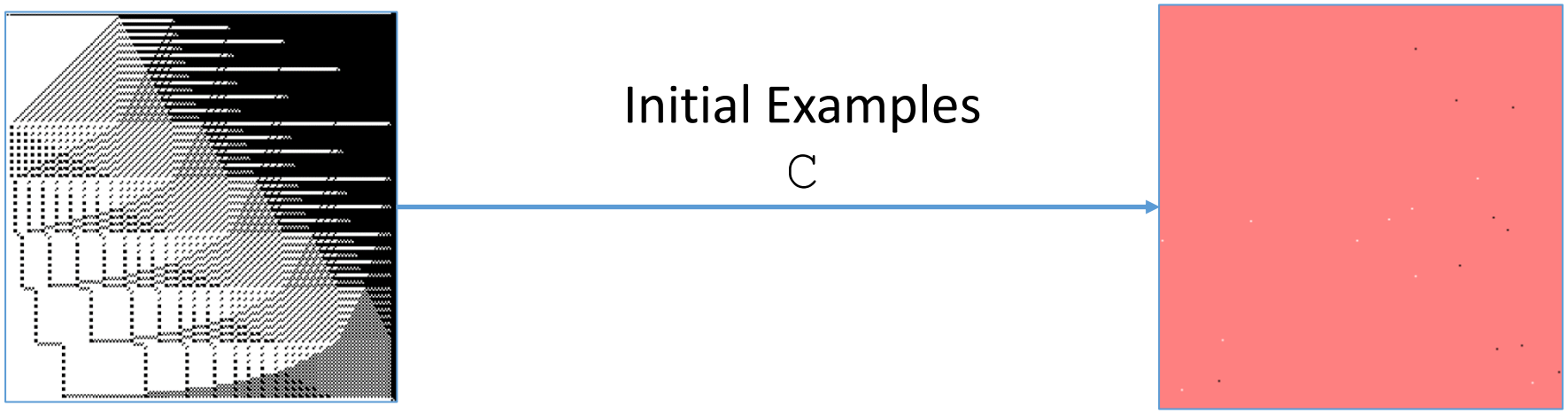
Synthesizing Invertibility Conditions



"Full I/O Specification for IC"
(Verifier)

(Synthesizer)

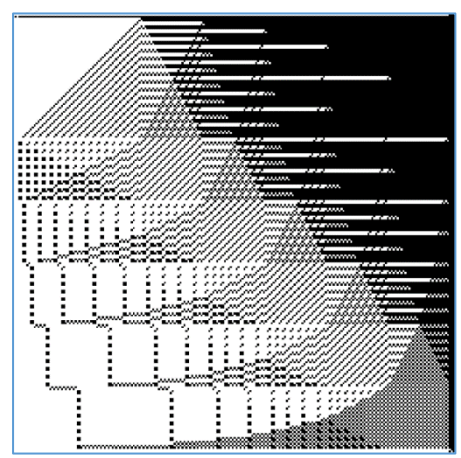
Synthesizing Invertibility Conditions

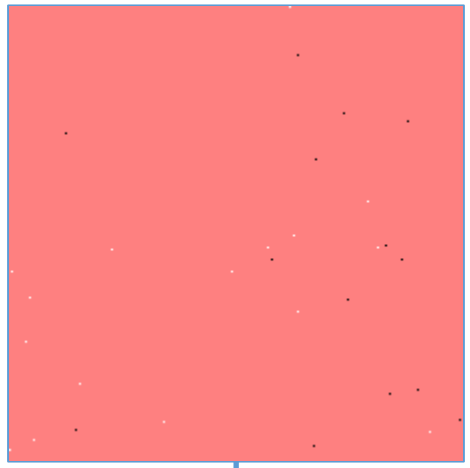


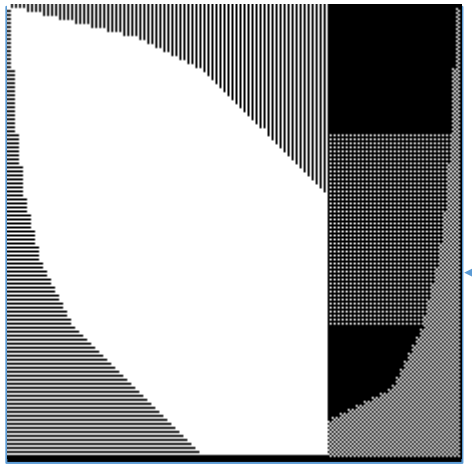
Red = Counterexample not specified

SY

Synthesizing Invertibility Conditions

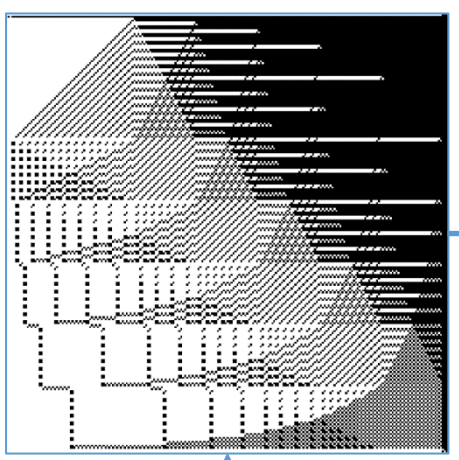






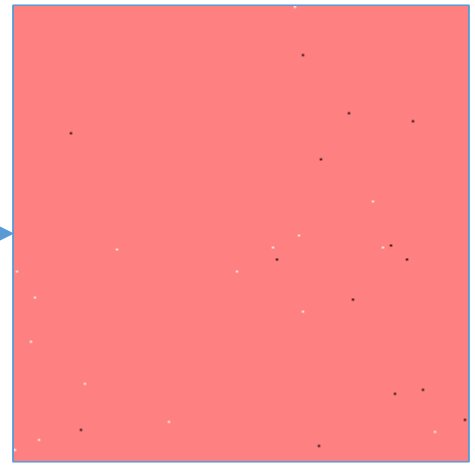
$IC_1(s,t)$

Synthesizing Invertibility Conditions

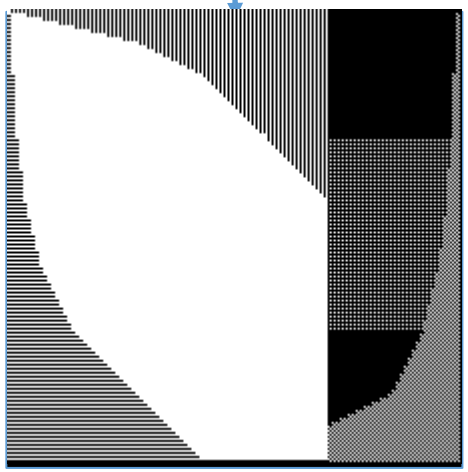


Counterexamples

C, C_1

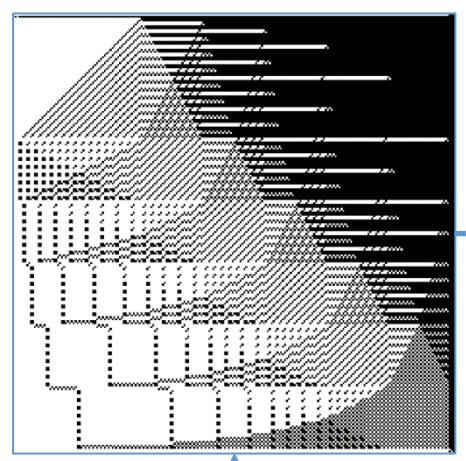


Test



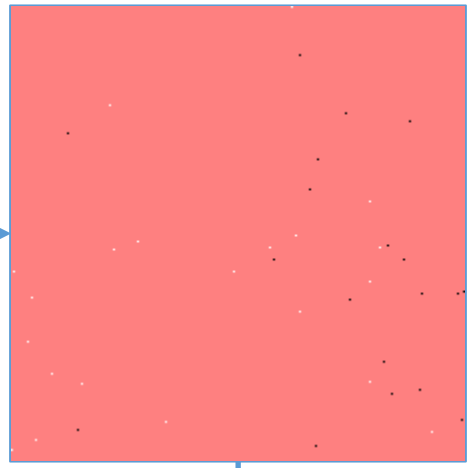
SY

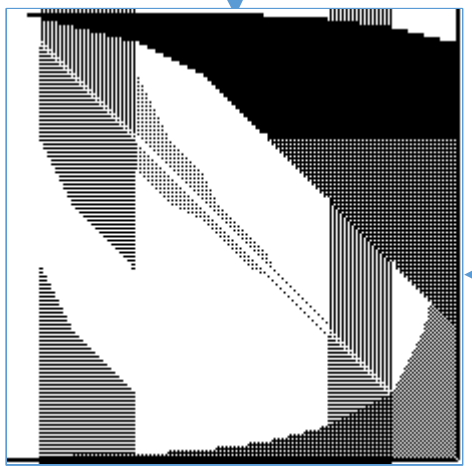
Synthesizing Invertibility Conditions



Counterexamples

C, C_1, C_2

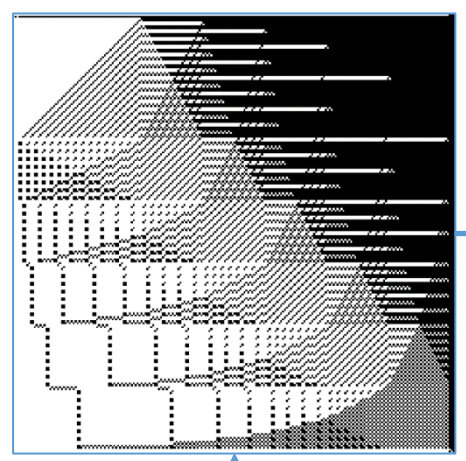




$IC_2(s,t)$

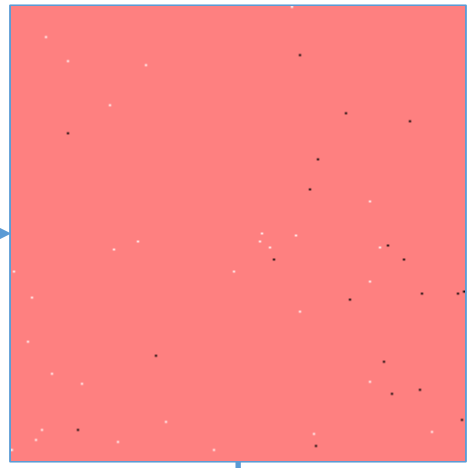
SY

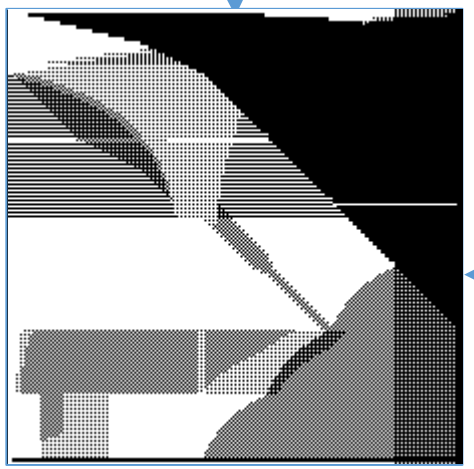
Synthesizing Invertibility Conditions



Counterexamples

C, C_1, C_2, C_3

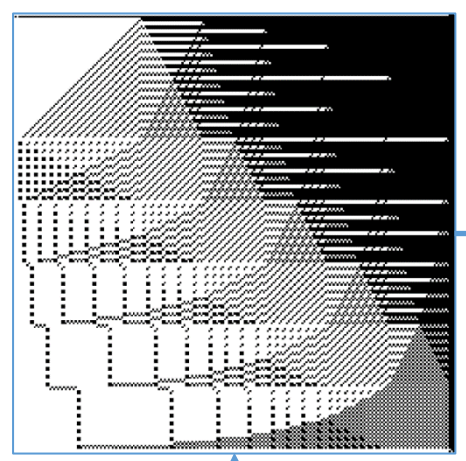




$IC_3(s,t)$

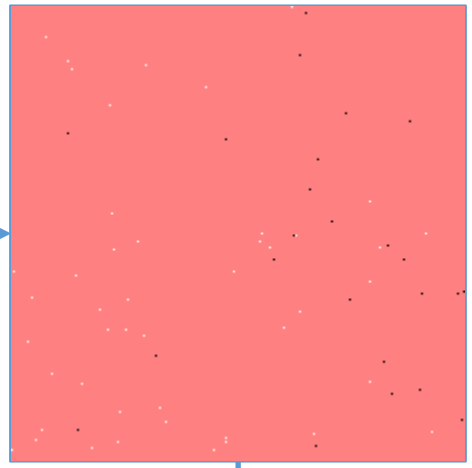
SY

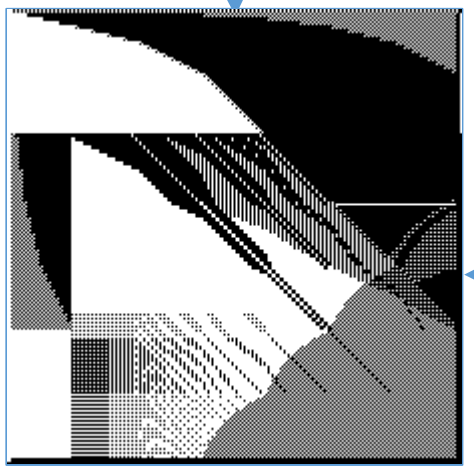
Synthesizing Invertibility Conditions



Counterexamples

C, C_1, C_2, C_3, C_4

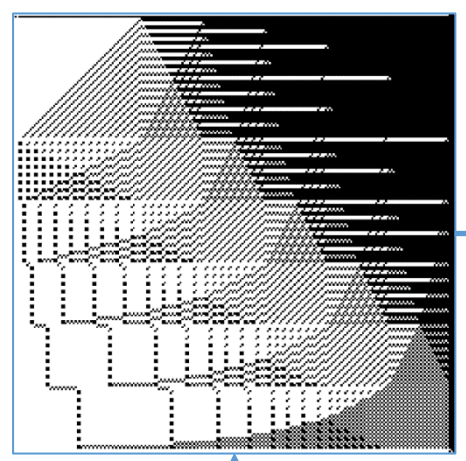




$IC_4(s,t)$

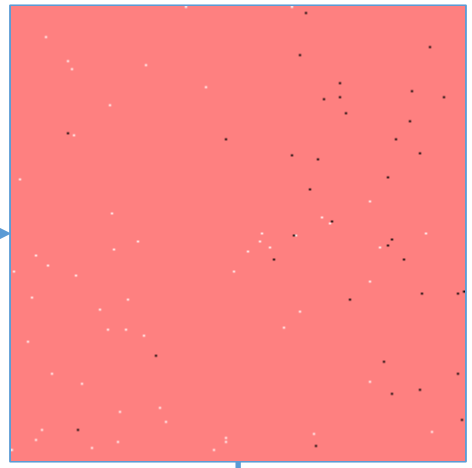
SY

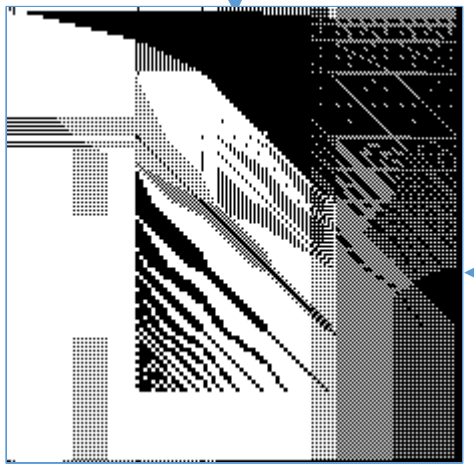
Synthesizing Invertibility Conditions



Counterexamples

$C, C_1, C_2, C_3, C_4, C_5$

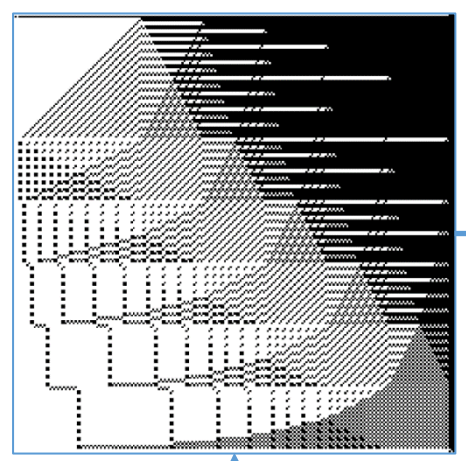




$IC_5(s,t)$

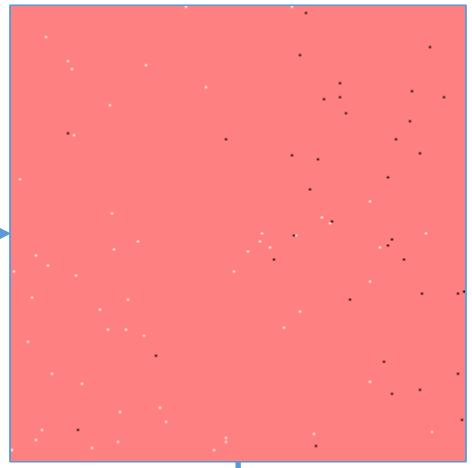
SY

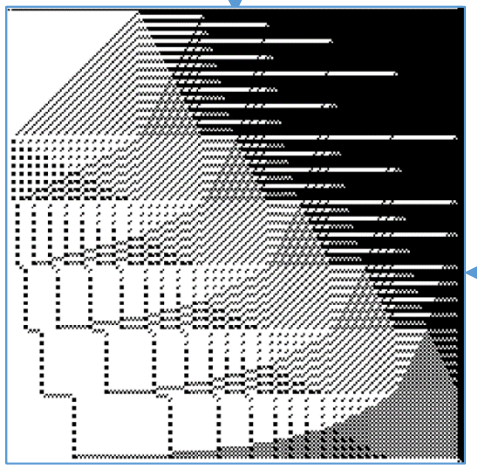
Synthesizing Invertibility Conditions



Counterexamples

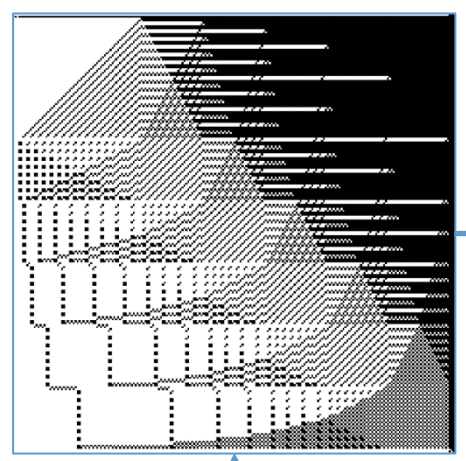
$C, C_1, C_2, C_3, C_4, C_5, C_6$





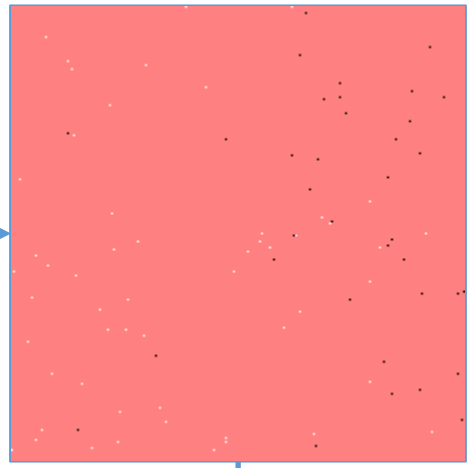
$$t \approx (t \overset{\text{RTP}}{-} s) \overset{R}{+} s \vee t \approx (t \overset{\text{RTN}}{-} s) \overset{R}{+} s \vee s \approx t$$

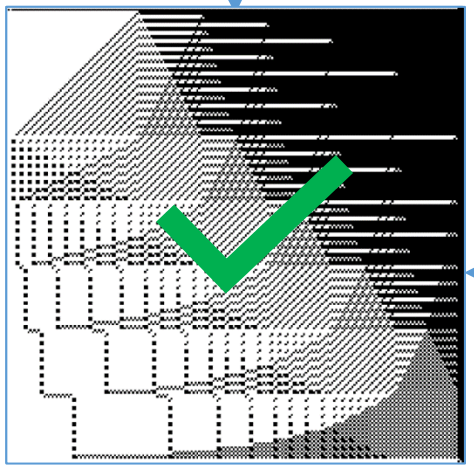
Synthesizing Invertibility Conditions



Counterexamples

$C, C_1, C_2, C_3, C_4, C_5, C_6$





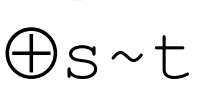
$$t \approx \overset{\text{RTP}}{(t-s)} \overset{R}{+} s \vee t \approx \overset{\text{RTN}}{(t-s)} \overset{R}{+} s \vee s \approx t$$

Finding Invertibility Conditions: Results

- In total, found 167 of 188 conditions

Finding Invertibility Conditions: Results

- In total, found 167 of 188 conditions



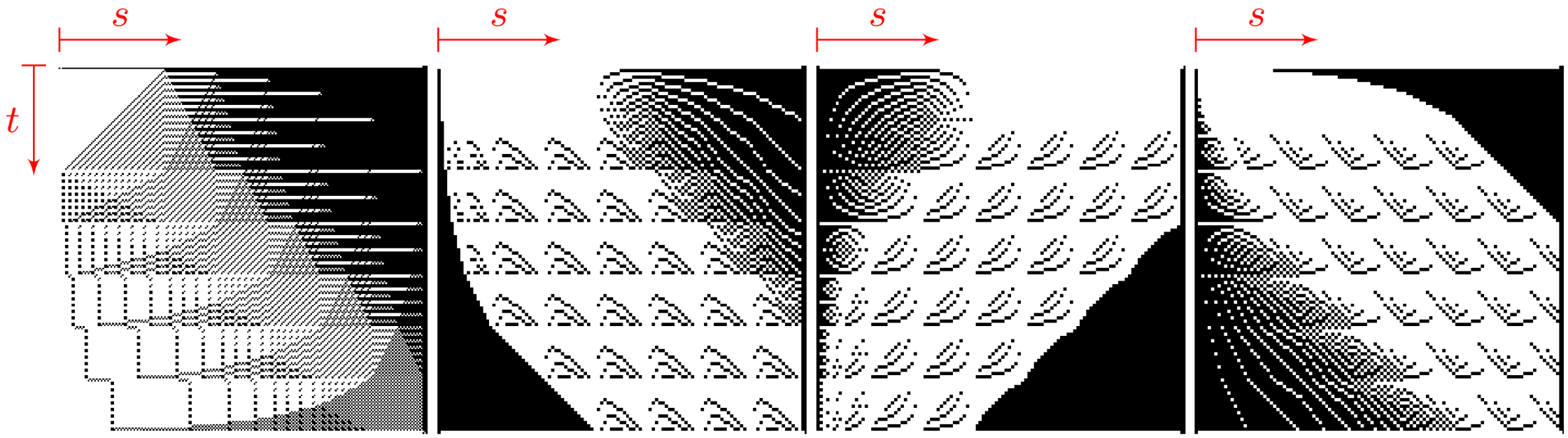
  - **14** choices for argument positions of $\oplus$
    - add_1, $sub_{1,2}$, $mult_1$, $div_{1,2}$, $rem_{1,2}$, $fma_{1,3}$, $sqrt_1$, abs_1, neg_1, $roundToIntegral_1$
  - **13** choices for ~
    - =, $\neq$, $\geq$, >, <, $\leq$, isNormal, isSubnormal, isZero, isInfinite, isNaN, isPositive, isNegative
  - **7** additional ICs for equations with only a relation, e.g. x>t

$\Rightarrow$ **14*13 + 7 = 188**

  - When applicable, ICs hold for each 5 rounding modes {RNE, RNA, RTP, RTN, RTZ} of operator

Finding Invertibility Conditions: Results

- In total, found 167 of 188 conditions



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

$$t \approx (t \overset{\text{RTP}}{-} s) \overset{R}{+} s \vee t \approx (t \overset{\text{RTN}}{-} s) \overset{R}{+} s \vee s \approx t$$

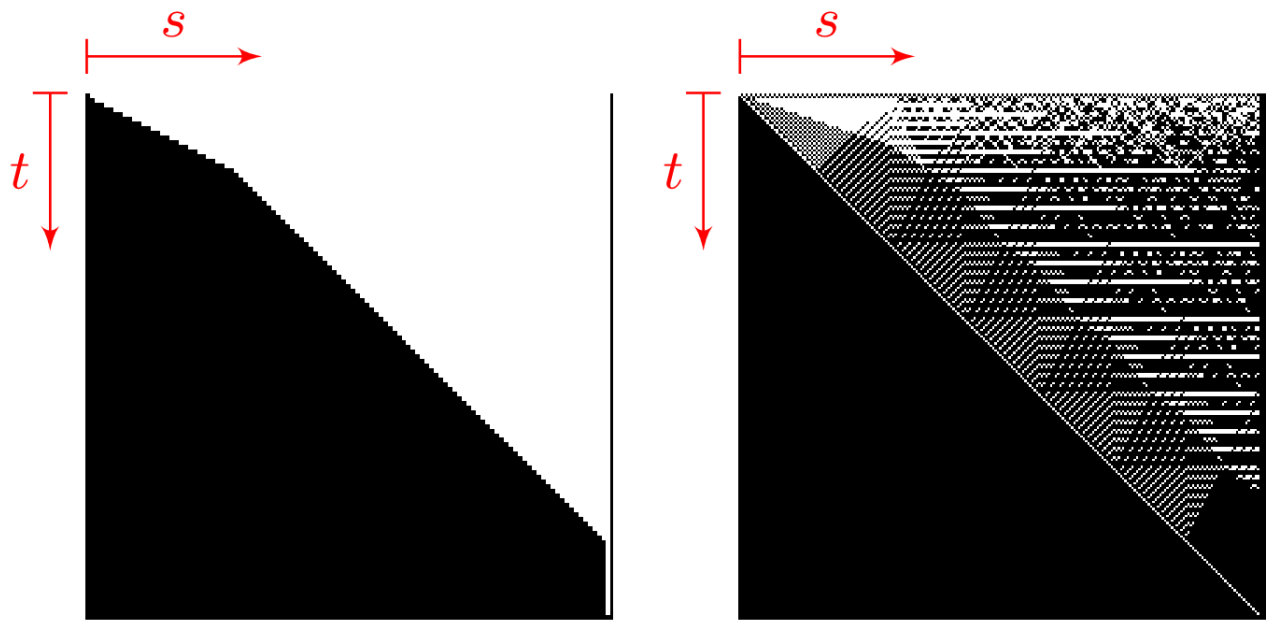
$$t \approx (t \overset{\text{RTP}}{\div} s) \overset{R}{\cdot} s \vee t \approx (t \overset{\text{RTN}}{\div} s) \overset{R}{\cdot} s \vee (s \approx \pm\infty \wedge t \approx \pm\infty) \vee (s \approx \pm 0 \wedge t \approx \pm 0)$$

$$t \approx (s \overset{\text{RTP}}{\cdot} t) \overset{R}{\div} s \vee t \approx (s \overset{\text{RTN}}{\cdot} t) \overset{R}{\div} s \vee (s \approx \pm\infty \wedge t \approx \pm 0) \vee (t \approx \pm\infty \wedge s \approx \pm 0)$$

$$t \approx s \overset{R}{\div} (s \overset{\text{RTP}}{\div} t) \vee t \approx s \overset{R}{\div} (s \overset{\text{RTN}}{\div} t) \vee (s \approx \pm\infty \wedge t \approx \pm\infty) \vee (s \approx \pm 0 \wedge t \approx \pm 0)$$

Finding Invertibility Conditions: Results

- Others were hard to find



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

$$|t \overset{\text{RTP}}{+} t| \leq |s| \vee |t \overset{\text{RTN}}{+} t| \leq |s| \vee \text{ite}(t \approx \pm 0, s \not\approx \pm 0, t \not\approx \pm\infty)$$

???

…behavior is pseudo-random!

Do Invertibility Conditions hold for all FP Formats?

- We **checked** all 167 invertibility conditions on other FP Formats
  - Targeted formats $\mathtt{FP}_{3,5}$, $\mathtt{FP}_{4,5}$, $\mathtt{FP}_{4,6}$, $\mathtt{FP}_{5,11}$, $\mathtt{FP}_{8,24}$, $\mathtt{FP}_{11,53}$
  - All invertibility conditions (appear to) generalize to all formats
    - **Successful in 94.5%** of the above cases using CVC4, Z3 (failures due to timeouts)
    - Approximately 32 days of compute time

- Future work: Format-Independent Verification!

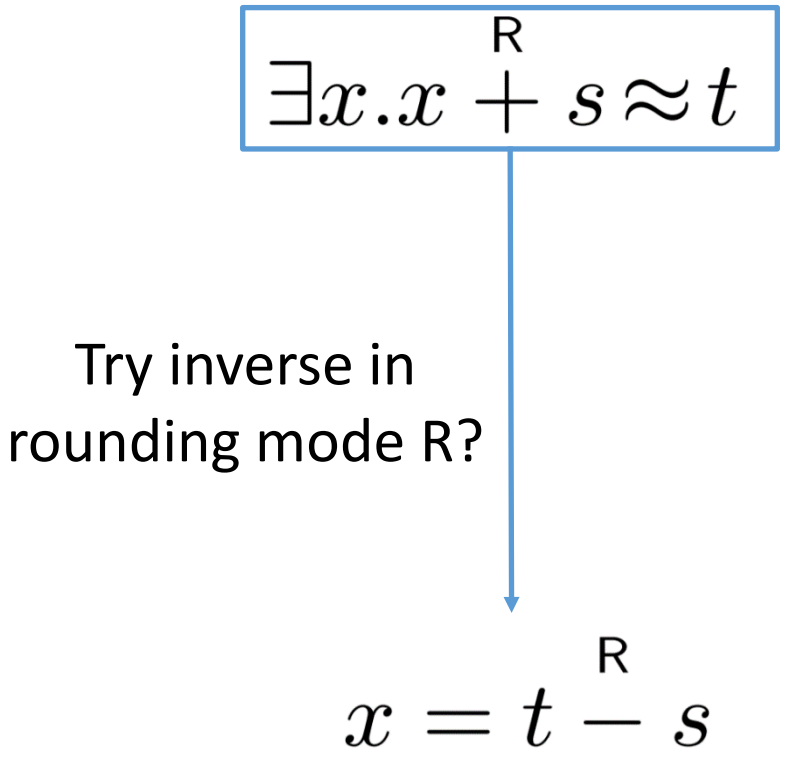
What can we do with Invertibility Conditions?

- Summary of intuition:

$$\exists x.x \overset{\mathsf{R}}{+} s \approx t$$

What can we do with Invertibility Conditions?

- Summary of intuition:



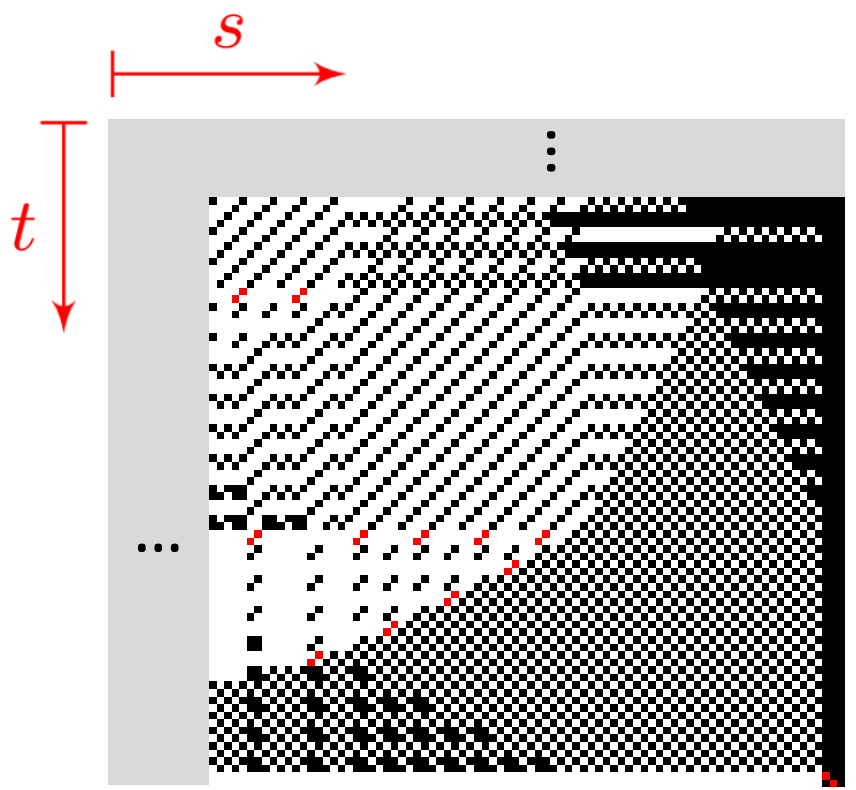
What can we do with Invertibility Conditions?

- Summary of intuition:

$$\exists x.x \overset{R}{+} s \approx t$$

Try inverse in rounding mode R?

$$x = t \overset{R}{-} s$$



red = wrong (showing R=RNE)

What can we do with Invertibility Conditions?

- Summary of intuition:

$$\exists x.x \overset{\mathsf{R}}{+} s \approx t$$

What can we do with Invertibility Conditions?

- Summary of intuition:

$$\boxed{\exists x.x \overset{\text{R}}{+} s \approx t}$$

$$\Updownarrow$$

$$(t - s) \overset{\text{RTP}}{\overset{R}{+}} s \approx t \vee (t - s) \overset{\text{RTN}}{\overset{R}{+}} s \approx t \vee s \approx t$$

What can we do with Invertibility Conditions?

- Summary of intuition:

$$\overset{\text{R}}{\exists x.x + s \approx t}$$

$$\Updownarrow$$

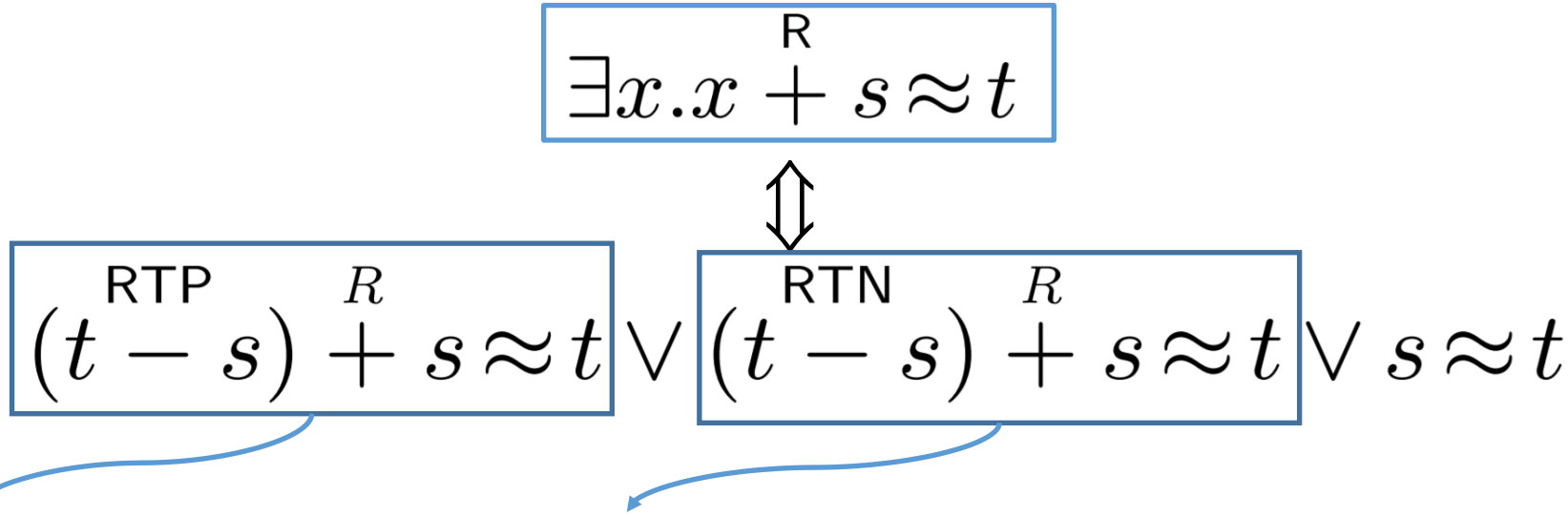
$$\overset{\text{RTP}}{(t - s)} \overset{R}{+} s \approx t \vee \overset{\text{RTN}}{(t - s)} \overset{R}{+} s \approx t \vee s \approx t$$

Try rounding towards positive

$$x \overset{\text{RTP}}{=} t - s$$

What can we do with Invertibility Conditions?

- Summary of intuition:



$$x = \overset{RTP}{t - s} \qquad x = \overset{RTN}{t - s}$$

What can we do with Invertibility Conditions?

- Summary of intuition:

$$\exists x.x \overset{\mathrm{R}}{+} s \approx t$$

$$\Updownarrow$$

$$(t - s) \overset{\mathrm{RTP}}{\overset{R}{+}} s \approx t \vee (t - s) \overset{\mathrm{RTN}}{\overset{R}{+}} s \approx t \vee s \approx t$$

Try rounding towards positive

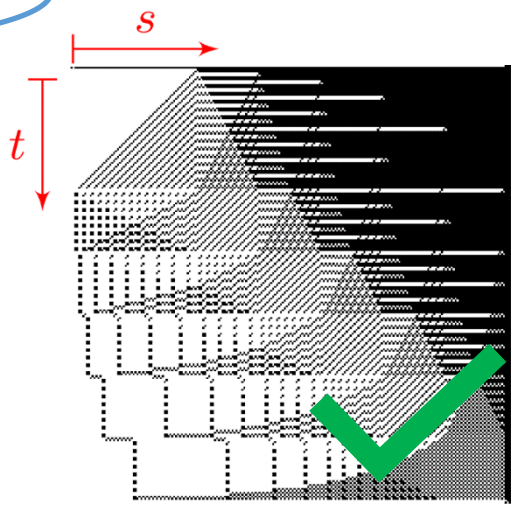
Try rounding towards negative

Corner case, try zero

$$x = t \overset{\mathrm{RTP}}{-} s$$

$$x = t \overset{\mathrm{RTN}}{-} s$$

$$x = \pm 0$$



What can we do with Invertibility Conditions?

- Similar intuition for many other invertibility conditions:

Equation	Set of Possible Solutions
$x \overset{\mathsf{R}}{\cdot} s \approx t$	$x \in \{t \overset{\mathsf{RTP}}{\div} s, t \overset{\mathsf{RTN}}{\div} s\}$
$x \overset{\mathsf{R}}{\div} s \approx t$	$x \in \{t \overset{\mathsf{RTP}}{\cdot} s, t \overset{\mathsf{RTN}}{\cdot} s\}$
$s \overset{\mathsf{R}}{\div} x \approx t$	$x \in \{s \overset{\mathsf{RTP}}{\div} t, s \overset{\mathsf{RTN}}{\div} t\}$
$\overset{\mathsf{R}}{\sqrt{x}} \approx t$	$x \in \{t \overset{\mathsf{RTP}}{\cdot} t, t \overset{\mathsf{RTN}}{\cdot} t\}$
$\mathsf{fma}(x, s, t) \approx \pm 0$	$x \in \{-(t \overset{\mathsf{RTP}}{\div} s), -(t \overset{\mathsf{RTN}}{\div} s)\}$

+ corner cases for ± 0, $\pm\infty$, NaN

Ongoing Work

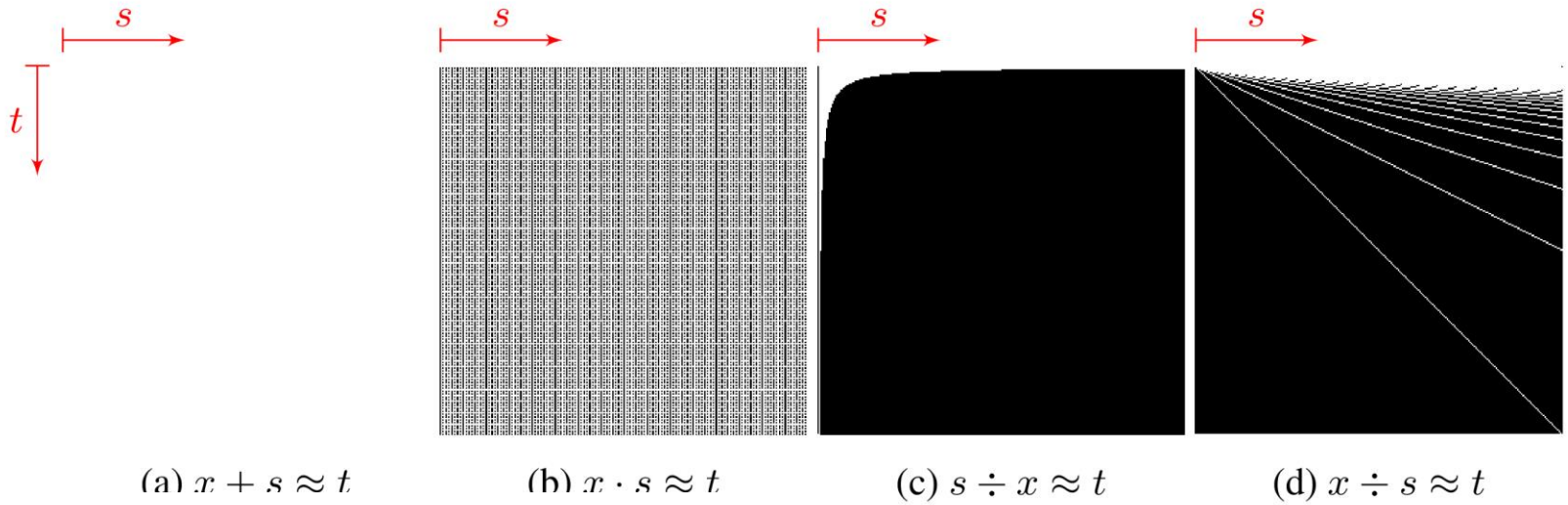
- Further rewrite rules, test case generation
- Solving algorithms based on invertibility conditions
  - Quantifier Elimination
  - Local search techniques **[Niemetz et al CAV2016]**
  - Quantifier Instantiation for Synthesis (for BV, FP, …)
    - Quantifier instantiation procedures ⇔ synthesis procedures
- Bit-width independent verification of invertibility conditions
  - Initial work “Toward Bit-Width Independent Proofs in SMT Solvers” **[CADE 2019]**
- Other applications: Syntax-guided optimization of FP code?
- …Your application here

Thanks for Listening!

- Techniques in talk available in SMT solver cvc5
  - Open-source : https://cvc5.github.io/

- Questions?

Visualizing Invertibility Conditions for Bit-Vectors



(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)