

# COUNTEREXAMPLE-GUIDED MODEL SYNTHESIS

Mathias Preiner<sup>\*†</sup>, Aina Niemetz<sup>\*†</sup> and Armin Biere<sup>\*</sup>

\* Johannes Kepler University

† Stanford University

SMT Workshop

July 22-23, 2017

Heidelberg, Germany



# Introduction

- Counterexample-Guided  
Combine counterexample-guided quantifier instantiation with . . .
- Synthesis  
. . . syntax-guided synthesis to synthesize . . .
- Model  
. . . interpretations for Skolem functions.
- Quantified Bit-Vectors

# Fixed-Size Bit-Vectors

**Bit-Vector:** vector of bits of a fixed size

- Constant values: 0011, 00000011,  $\mathfrak{3}_{[8]}$ , ...
- Variables:  $x_{[16]}$ ,  $y_{[9]}$ , ...
- Operators:
  - bitwise:  $\sim$ ,  $\&$ ,  $|$ ,  $\oplus$ ,  $\ll$ ,  $\gg$ , ...
  - arithmetic:  $+$ ,  $-$ ,  $*$ ,  $/$ , ...
  - predicates:  $=$ ,  $<$ ,  $\leq$ , ...
  - string operations: concat, extract, extension, ...

**Example with Quantifiers**

$$\forall x_{[4]} \exists y_{[4]} . (x \& 1100) + y = 0000$$

# Quantified Bit-Vectors

## State-of-the-Art

- **Z3**: Model-based quantifier instantiation (MBQI) [de Moura'09]
  - combined with E-matching
- **CVC4**: Counterexample-guided quantifier instantiation (CEGQI) [Reynolds'15]
- **Q3B**: BDD-based approach [Strejcek'16]
  - relies on simplifications, approximation techniques, variable ordering

## Our approach

### **Counterexample-Guided Model Synthesis (CEGMS)**

- ▷ Combines synthesis with variant of CEGQI

# Counterexample-Guided Model Synthesis

**Example**  $\varphi := \forall x_{[32]} \exists y_{[32]} . x + y = 0$

**Skolem**  $\varphi_S := \forall x_{[32]} . x + f(x) = 0$

Ground Instances of  $\varphi_S$

$x$	$x + f(x) = 0$
0	$0 + f(0) = 0$
1	$1 + f(1) = 0$
2	$2 + f(2) = 0$
$\vdots$	$\vdots$
$2^{32}-1$	...

Function Table  $f$

$x$	$f(x)$
0	0
1	-1
2	-2
$\vdots$	$\vdots$
$2^{32}-1$	$-(2^{32}-1)$

Goal

$$f := \lambda x. -x$$

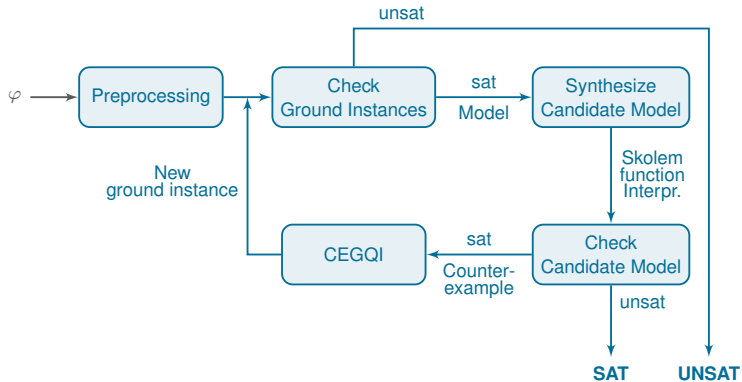
↓

$$\forall x_{[32]} . x + -x = 0 \checkmark$$

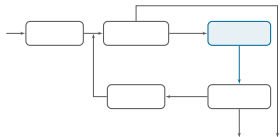
How?

**Synthesize + Refine**

# Workflow



# Synthesis of Candidate Models



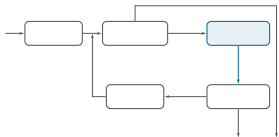
## Enumerative Learning [Alur'13]

- enumerate expressions based on a **syntax/grammar**
  - ▷ start with smallest expressions (inputs)
  - ▷ enumerate expressions in increasing size
  - ▷ prune enumeration space
- check if expression conforms to some set of **test cases**
- return candidate expression if set of test cases is covered

## Synthesis of Skolem Functions

- apply enumerative learning to each Skolem symbol
- use set of ground instances  $G$  as “test cases”
- substitute Skolem symbol with candidate expression in  $G$
- evaluate  $g_i \in G$ 
  - ▷ return candidate interpretation if all  $g_i \in G$  are satisfied

## Synthesis of Candidate Models (cont.)



### Example

Inputs:  $x, y, 0, 1$

Operators:  $=, +, \&, \text{ite}$

### Size Enumerated Expressions

1	$x$	$y$	$0$	$1$				
2	$x = y$	$x = 0$	$x = 1$	$y = x$	$y = 0$	$y = 1$	$x + y$	...
3	$(x + y) + x$	$(x + y) + y$	$(x + y) + 1$	...				
4	$(x = y) \& (x = 0)$	$(x = y) \& (x = 1)$	...	$\text{ite}(x = y, x, y)$	...			
:								

Expression Size:  $\text{size}(x = y) := \text{size}(x) + \text{size}(y)$

▷ Large enumeration space!



# Pruning Enumeration Space

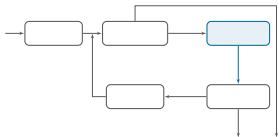
Idea: discard “similar” expressions

While enumerating expressions ...

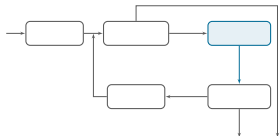
1. generate **signature** for each expression
2. if signature already cached **discard** expression
3. else **cache** signature

Signature Computation of Expression  $e$

- set of ground instances  $G := \{g_1, \dots, g_n\}$
- substitute Skolem symbol  $f$  in  $G$  with  $e$
- evaluate resulting  $g_1', \dots, g_n'$
- store evaluations (Boolean values) as vector of size  $n$  (= signature)
  - ▷ if every value is true,  $e$  is a candidate interpretation for  $f$



## Example: Synthesis



Example:  $z = \min(x, y)$

$$\varphi := \forall x y \exists z . (x < y \rightarrow z = x) \wedge (x \geq y \rightarrow z = y)$$

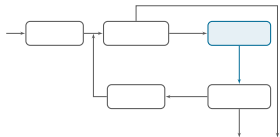
$$\varphi_S := \forall x y . (x < y \rightarrow f_z(x, y) = x) \wedge (x \geq y \rightarrow f_z(x, y) = y)$$

Inputs for  $f_z$        $\{ x, y \}$

Operators             $\{ =, <, \geq, \wedge, \rightarrow, ite \}$

Ground Inst.  $G$      $\{ f_z(0, 0) = 0,$   
                           $f_z(0, 1) = 0,$   
                           $f_z(2, 1) = 1 \}$

## Example: Synthesis cont.



Size	Enumerated Expressions
------	------------------------

1	$x, y$
---	--------

2	$x = y, x < y, y < x, x \geq y, y \geq x$
---	-------------------------------------------

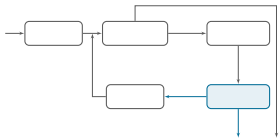
3	–
---	---

4	$(x = y \wedge x < y), \dots, (x = y \rightarrow x < y), \dots, ite(x < y, x, y)$
---	-----------------------------------------------------------------------------------

### Signature of Candidate $ite(x < y, x, y)$

$$\underbrace{ite(0 < 0, 0, 0)}_{\top} = 0, \quad \underbrace{ite(0 < 1, 0, 1)}_{\top} = 0, \quad \underbrace{ite(2 < 1, 2, 1)}_{\top} = 1$$

## Example: Check Candidate Model



Candidate Model  $\{f_z := \lambda x y . ite(x < y, x, y)\}$

Check

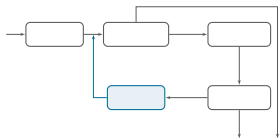
$$\neg \varphi_S[\lambda x y . ite(x < y, x, y) / f_z]$$
$$\equiv \exists x y . (x < y \wedge ite(x < y, x, y) \neq x) \vee (x \geq y \wedge ite(x < y, x, y) \neq y)$$

SMT Solver Check

$$\underbrace{(a < b \wedge ite(a < b, a, b) \neq a)}_{\perp} \vee \underbrace{(a \geq b \wedge ite(a < b, a, b) \neq b)}_{\perp}$$

- **unsat**: candidate model is **valid**
- **sat**: found counterexample, **refine**

## Example: Refinement



Assume Candidate Model  $\{f_z := \lambda x y . x\}$

SMT Solver Check

$$\underbrace{(a < b \wedge a \neq a)}_{\perp} \vee \underbrace{(a \geq b \wedge a \neq b)}_{\top}$$

- ▷ Solver returns **sat**, candidate model is **invalid**
- ▷ Solver produces counterexample  $\{a = 1, b = 0\}$

Add New Instance of  $\varphi_S$  to  $G$

$$G := G \cup \{\varphi_S[1/x, 0/y]\}$$

# Dual Counterexample-Guided Model Synthesis

**Idea** Find instantiation for  $\forall$ -variables s.t. formula is **unsatisfiable**.

**How** Apply CEGMS to the **dual** formula  $\neg\varphi$

**Duality**  $\text{CEGMS}(\neg\varphi) \text{ sat} \rightsquigarrow \text{CEGMS}(\varphi) \text{ unsat}$

$\text{CEGMS}(\neg\varphi) \text{ unsat} \rightsquigarrow \text{CEGMS}(\varphi) \text{ sat}$

**Original**  $\varphi := \exists a b c \forall x . \underbrace{(a * c) + (b * c) \neq (x * c)}_{\text{unsat with } \varphi[a+b/x]}$

**Dual**  $\neg\varphi := \forall a b c \exists x . \underbrace{(a * c) + (b * c) = (x * c)}_{\text{sat with } \neg\varphi[a+b/x]}$

▷  $\text{CEGMS}(\varphi)$  and  $\text{CEGMS}(\neg\varphi)$  can be executed in **parallel**

# Experiments

	SMT-LIB (191)				New <sup>1</sup> (4838)			
	Solved	Sat	Unsat	Time [s]	Solved	Sat	Unsat	Time [s]
<b>Boolector</b>	142	51	91	59529	4527	465	4062	389020
<b>Boolector+s</b>	164	72	92	32996	4526	467	4059	390613
<b>Boolector+d</b>	162	67	95	35877	4572	<b>518</b>	4054	342412
<b>Boolector+ds</b>	<b>172</b>	<b>77</b>	<b>95</b>	<b>24163</b>	<b>4704</b>	517	<b>4187</b>	<b>187411</b>

**Boolector** ... CEGQI only    **+s** ... synthesis    **+d** ... dual (parallel)

**Limits** 1200 seconds CPU time, 7GB memory

---

<sup>1</sup>LIA, LRA, NIA, NRA SMT-LIB benchmarks translated to BV

## Experiments

	SMT-LIB (191)				New (4838)			
	Solved	Sat	Unsat	Time [s]	Solved	Sat	Unsat	Time [s]
<b>Boolector+ds</b>	172	77	<b>95</b>	24163	4704	<b>517</b>	4187	187411
<b>CVC4</b>	145	64	81	57652	4362	339	4023	580402
<b>Q3B</b>	<b>187</b>	<b>93</b>	94	<b>9086</b>	4367	327	4040	581252
<b>Z3</b>	161	69	92	37534	4732	476	<b>4256</b>	<b>133241</b>

Limits 1200 seconds CPU time, 7GB memory



# Experiments

## Synthesis Overhead (Runtime)

- up to 75% on solved benchmarks
- up to 98% on unsolved benchmarks

## Refinement Iterations

- up to 300 iterations on solved benchmarks
- up to 9400 iterations on unsolved benchmarks

## Synthesized Terms

- |         |                           |                            |
|---------|---------------------------|----------------------------|
| ■ $c$   | ■ $(x_i \text{ op } x_j)$ | ■ $\sim(c * x_i)$          |
| ■ $x_i$ | ■ $(c \text{ op } x_i)$   | ■ $(x_i + (c + \sim x_j))$ |

$x_i$  ... universal variables,     $c$  ... constant value,     $op$  ... bit-vector operator

# Conclusion

- **simple** approach for solving quantified bit-vectors
  - only requires two instances of ground theory solvers
  - enumerative learning algorithm straightforward to implement
- **competitive** with the state-of-the-art in solving BV
  - no simplification techniques yet
  - no E-matching or other quantifier instantiation heuristics
- **future directions**
  - improve synthesis approach
    - ▷ employ divide and conquer approach from [Alur'17]
    - ▷ employ other synthesis approaches?
  - generalize counterexamples via synthesis
  - model reconstruction from unsatisfiable dual formulas
  - useful for other theories?

## References I

- [Alur'17] Rajeev Alur and Arjun Radhakrishna and Abhishek Udupa. Scaling Enumerative Program Synthesis via Divide and Conquer. TACAS, Pages 319-336. 2017
- [Alur'13] Abhishek Udupa and Arun Raghavan and Jyotirmoy V. Deshmukh and Sela Mador-Haim and Milo M. K. Martin and Rajeev Alur. TRANSIT: specifying protocols with concolic snippets. SIGPLAN, Pages 287-296. 2013
- [Strejcek'16] Martin Jonás and Jan Strejcek. Solving Quantified Bit-Vector Formulas Using Binary Decision Diagrams. SAT, Pages 267-283. 2016
- [de Moura'09] Yeting Ge and Leonardo Mendonça de Moura. Complete Instantiation for Quantified Formulas in Satisfiability Modulo Theories. CAV, Pages 306-320. 2009
- [Reynolds'15] Andrew Reynolds and Morgan Deters and Viktor Kuncak and Cesare Tinelli and Clark W. Barrett. Counterexample-Guided Quantifier Instantiation for Synthesis in SMT. CAV, Pages 198-216. 2015

## References II