

Better Lemmas with Lambda Extraction

Mathias Preiner, Aina Niemetz and Armin Biere

Institute for Formal Models and Verification (FMV)
Johannes Kepler University, Linz, Austria
<http://fmv.jku.at/>

FMCAD 2015
September 27-30, 2015
Austin, Texas, USA

Introduction

Better Lemmas?

... in the context of **lemmas on demand** for the theory of arrays

- more succinct
- stronger
- reduce number of lemmas → speeds up solving

How?

- ① **identify** array patterns in sequences of array operations
- ② **generalize** them as lambda terms
- ③ to create **better lemmas** on demand

→ considerably improves solver performance

→ particularly on instances from **symbolic execution**

Introduction

Theory of Arrays [McCarthy'62]

- introduces two function symbols to access/modify arrays
 - *read*(a, i) read value from array a on index i
 - *write*(a, i, e) write value e to array a at index i
- reason about memory in SW and HW verification

Limitations

- operate on **single indices** only
- no succinct operations over **multiple indices**
e.g. *memset* or *memcpy* operations
- not possible to reason about **variable** number of indices
(without quantifiers)

Introduction

Theory of Arrays [McCarthy'62]

- introduces two function symbols to access/modify arrays
 - $read(a, i)$ read value from array a on index i
 - $write(a, i, e)$ write value e to array a at index i
- reason about memory in SW and HW verification

Limitations

- operate on **single indices** only
- no succinct operations over **multiple indices**
e.g. *memset* or *memcpy* operations
- not possible to reason about **variable** number of indices
(without quantifiers)

Introduction

Arrays as Lambdas

UCLID [CAV'02]

- restricted lambda terms to tackle limitations
- eager elimination of lambda terms
- might result in exponential blow-up in formula size

Boolector [DIFTS'13]

- decision procedure for lambda terms
- lazy handling of lambda terms
- avoid worst-case exponential blow-up
- array engine in Boolector
→ treats arrays and array operations as functions

Introduction

Arrays as Lambdas

UCLID [CAV'02]

- restricted lambda terms to tackle limitations
- **eager elimination** of lambda terms
- might result in exponential blow-up in formula size

Boolector [DIFTS'13]

- decision procedure for lambda terms
- **lazy** handling of lambda terms
- avoid worst-case exponential blow-up
- **array engine** in Boolector
→ treats arrays and array operations as functions

Arrays as Lambdas

Representation

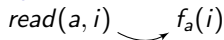
Array Variable Uninterpreted Function

a f_a



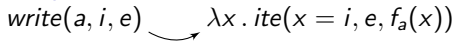
Read Operation Function Application

$read(a, i)$ $f_a(i)$



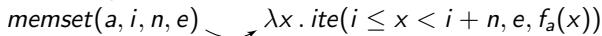
Write Operation Lambda Term

$write(a, i, e)$ $\lambda x . ite(x = i, e, f_a(x))$



Memset Operation Lambda Term

$memset(a, i, n, e)$ $\lambda x . ite(i \leq x < i + n, e, f_a(x))$



Motivation

Example

Set 4 consecutive indices of array a to value e starting from index i .

Array representation

$a_1 := \text{write}(a, i, e)$

$a_2 := \text{write}(a_1, i + 1, e)$

$a_3 := \text{write}(a_2, i + 2, e)$

$a_4 := \text{write}(a_3, i + 3, e)$

→ requires $n = 4$ writes

→ n arbitrarily big

Lambda term representation

$\lambda_4 := \lambda x . \text{ite}(i \leq x \wedge x < i + 4, e, f_a(x))$

→ more compact representation

→ symbolic size n

→ better lemmas

Our goal: Identify array patterns and represent them as lambda terms

Motivation

Example

Set 4 consecutive indices of array a to value e starting from index i .

Array representation

$a_1 := \text{write}(a, i, e)$

$a_2 := \text{write}(a_1, i + 1, e)$

$a_3 := \text{write}(a_2, i + 2, e)$

$a_4 := \text{write}(a_3, i + 3, e)$

→ requires $n = 4$ writes

→ n arbitrarily big

Lambda term representation

$\lambda_4 := \lambda x . \text{ite}(i \leq x \wedge x < i + 4, e, f_a(x))$

→ more compact representation

→ symbolic size n

→ better lemmas

Our goal: Identify array patterns and represent them as lambda terms

Motivation

Example

Set 4 consecutive indices of array a to value e starting from index i .

Array representation

$a_1 := \text{write}(a, i, e)$

$a_2 := \text{write}(a_1, i + 1, e)$

$a_3 := \text{write}(a_2, i + 2, e)$

$a_4 := \text{write}(a_3, i + 3, e)$

→ requires $n = 4$ writes

→ n arbitrarily big

Lambda term representation

$\lambda_4 := \lambda x. \text{ite}(i \leq x \wedge x < i + 4, e, f_a(x))$

→ more compact representation

→ symbolic size n

→ better lemmas

Our goal: Identify array patterns and represent them as lambda terms

Lambda Extraction

memset Pattern

```
(set-logic QF_ABV)
(declare-fun a () (Array (_ BitVec 8) (_ BitVec 32)))
(declare-fun e () (_ BitVec 32))
...
(assert
  (= a_init
    (store
      (store
        (store
          (store a (_ bv0 8) e)
            (_ bv1 8) e)
          (_ bv2 8) e)
        (_ bv3 8) e)))
...
(exit)
```

Lambda Extraction

memset Pattern

```
(set-logic QF_ABV)
(declare-fun a () (Array (_ BitVec 8) (_ BitVec 32)))
(declare-fun e () (_ BitVec 32))
...
(assert
  (= a_init
    (store
      (store
        (store
          (store a (_ bv0 8) e)
            (_ bv1 8) e)
          (_ bv2 8) e)
        (_ bv3 8) e)))
...
(exit)
```

Lambda Extraction

memset Pattern

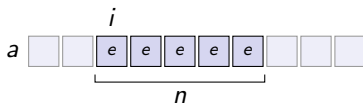
$memset(a, i, n, e)$

a ... base array

i ... start address

n ... size (constant)

e ... value



Lambda Term

$\lambda_{memset} := \lambda x . ite(i \leq x < i + n, e, f_a(x))$

Lambda Extraction

Loop Initialization Pattern: $i \rightarrow e$

```
(set-logic QF_ABV)
(declare-fun a () (Array (_ BitVec 8) (_ BitVec 32)))
(declare-fun e () (_ BitVec 32))
...
(assert
  (= a_init
    (store
      (store
        (store
          (store a (_ bv0 8) e)
            (_ bv2 8) e)
          (_ bv4 8) e)
        (_ bv6 8) e)))
...
(exit)
```

Lambda Extraction

Loop Initialization Pattern: $i \rightarrow e$

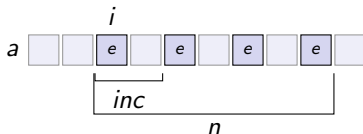
```
(set-logic QF_ABV)
(declare-fun a () (Array (_ BitVec 8) (_ BitVec 32)))
(declare-fun e () (_ BitVec 32))
...
(assert
  (= a_init
    (store
      (store
        (store
          (store a (_ bv0 8) e)
            (_ bv2 8) e)
          (_ bv4 8) e)
        (_ bv6 8) e)))
...
(exit)
```

Lambda Extraction

Loop Initialization Pattern: $i \rightarrow e$

$for (j = i; j < i + n; j = j + inc) \{ a[j] = e; \}$

a ... base array
 i ... start address
 n ... size (constant)
 inc ... increment (constant)
 e ... value



Lambda Term

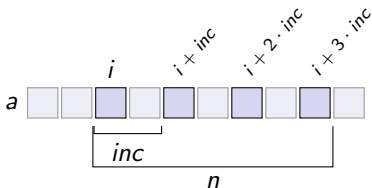
$\lambda_{i \rightarrow e} := \lambda x . ite(i \leq x \wedge x < i + n \wedge (inc \mid (x - i)), e, f_a(x))$

Lambda Extraction

Loop Initialization Pattern: $i \rightarrow i$

$for (j = i; j < i + n; j = j + inc) \{a[j] = j; \}$

a ... base array
 i ... start address
 n ... size (constant)
 inc ... increment (constant)



Lambda Term

$\lambda_{i \rightarrow i} := \lambda x . ite(i \leq x \wedge x < i + n \wedge (inc \mid (x - i)), x, f_a(x))$

Variation: $i \rightarrow i + 1$

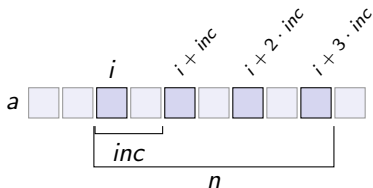
$for (j = i; j < i + n; j = j + inc) \{a[j] = j + 1; \}$

Lambda Extraction

Loop Initialization Pattern: $i \rightarrow i$

for ($j = i; j < i + n; j = j + inc$) { $a[j] = j;$ }

a ... base array
i ... start address
n ... size (constant)
inc ... increment (constant)



Lambda Term

$\lambda_{i \rightarrow i} := \lambda x . ite(i \leq x \wedge x < i + n \wedge (inc \mid (x - i)), x, f_a(x))$

Variation: $i \rightarrow i + 1$

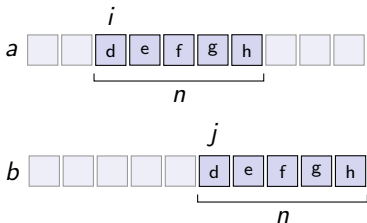
for ($j = i; j < i + n; j = j + inc$) { $a[j] = j + 1;$ }

Lambda Extraction

memcpy Pattern

$memcpy(a, b, i, j, n)$

- a ... source array
- b ... destination array
- i ... source address
- j ... destination address
- n ... size (constant)



Lambda Term

$\lambda_{memcpy} := \lambda x. ite(j \leq x < j + n, f_a(i + x - j), f_b(x))$

Lambda Extraction

Better Lemma Generation

Write sequence

$a_1 := \text{write}(a, 5, e)$

$a_2 := \text{write}(a_1, 6, e)$

$a_3 := \text{write}(a_2, 7, e)$

Conflict

$j = 7 \wedge \text{read}(a_3, j) \neq e$

$j = 6 \wedge \text{read}(a_3, j) \neq e$

$j = 5 \wedge \text{read}(a_3, j) \neq e$

Lemmas

$j = 7 \rightarrow \text{read}(a_3, j) = e$

$j = 6 \rightarrow \text{read}(a_3, j) = e$

$j = 5 \rightarrow \text{read}(a_3, j) = e$

→ $n=3$ lemmas in

worst-case

→ covers single indices

Lambda term

$\lambda_3 := \lambda x . \text{ite}(5 \leq x \wedge x < 8, e, f_a(x))$

Conflict

$j = 7 \wedge \lambda_3(j) \neq e$

Lemma

$5 \leq j \wedge j < 8 \rightarrow \lambda_3(j) = e$

→ only **one lemma** generated

→ covers **index range**

Lambda Extraction

Better Lemma Generation

Write sequence

$a_1 := \text{write}(a, 5, e)$

$a_2 := \text{write}(a_1, 6, e)$

$a_3 := \text{write}(a_2, 7, e)$

Conflict

$j = 7 \wedge \text{read}(a_3, j) \neq e$

$j = 6 \wedge \text{read}(a_3, j) \neq e$

$j = 5 \wedge \text{read}(a_3, j) \neq e$

Lemmas

$j = 7 \rightarrow \text{read}(a_3, j) = e$

$j = 6 \rightarrow \text{read}(a_3, j) = e$

$j = 5 \rightarrow \text{read}(a_3, j) = e$

→ $n=3$ lemmas in

worst-case

→ covers single indices

Lambda term

$\lambda_3 := \lambda x . \text{ite}(5 \leq x \wedge x < 8, e, f_a(x))$

Conflict

$j = 7 \wedge \lambda_3(j) \neq e$

Lemma

$5 \leq j \wedge j < 8 \rightarrow \lambda_3(j) = e$

→ only **one lemma** generated

→ covers **index range**

Lambda Extraction

Better Lemma Generation

Write sequence

$a_1 := \text{write}(a, 5, e)$

$a_2 := \text{write}(a_1, 6, e)$

$a_3 := \text{write}(a_2, 7, e)$

Conflict

$j = 7 \wedge \text{read}(a_3, j) \neq e$

$j = 6 \wedge \text{read}(a_3, j) \neq e$

$j = 5 \wedge \text{read}(a_3, j) \neq e$

Lemmas

$j = 7 \rightarrow \text{read}(a_3, j) = e$

$j = 6 \rightarrow \text{read}(a_3, j) = e$

$j = 5 \rightarrow \text{read}(a_3, j) = e$

→ $n=3$ lemmas in

worst-case

→ covers single indices

Lambda term

$\lambda_3 := \lambda x . \text{ite}(5 \leq x \wedge x < 8, e, f_a(x))$

Conflict

$j = 7 \wedge \lambda_3(j) \neq e$

Lemma

$5 \leq j \wedge j < 8 \rightarrow \lambda_3(j) = e$

→ only **one lemma** generated

→ covers **index range**

Lambda Extraction

Better Lemma Generation

Write sequence

$a_1 := \text{write}(a, 5, e)$

$a_2 := \text{write}(a_1, 6, e)$

$a_3 := \text{write}(a_2, 7, e)$

Conflict

$j = 7 \wedge \text{read}(a_3, j) \neq e$

$j = 6 \wedge \text{read}(a_3, j) \neq e$

$j = 5 \wedge \text{read}(a_3, j) \neq e$

Lemmas

$j = 7 \rightarrow \text{read}(a_3, j) = e$

$j = 6 \rightarrow \text{read}(a_3, j) = e$

$j = 5 \rightarrow \text{read}(a_3, j) = e$

→ **n=3** lemmas in

worst-case

→ covers **single indices**

Lambda term

$\lambda_3 := \lambda x . \text{ite}(5 \leq x \wedge x < 8, e, f_a(x))$

Conflict

$j = 7 \wedge \lambda_3(j) \neq e$

Lemma

$5 \leq j \wedge j < 8 \rightarrow \lambda_3(j) = e$

→ only **one lemma** generated

→ covers **index range**

Lambda Extraction

Better Lemma Generation

Write sequence

$a_1 := \text{write}(a, 5, e)$

$a_2 := \text{write}(a_1, 6, e)$

$a_3 := \text{write}(a_2, 7, e)$

Conflict

$j = 7 \wedge \text{read}(a_3, j) \neq e$

$j = 6 \wedge \text{read}(a_3, j) \neq e$

$j = 5 \wedge \text{read}(a_3, j) \neq e$

Lemmas

$j = 7 \rightarrow \text{read}(a_3, j) = e$

$j = 6 \rightarrow \text{read}(a_3, j) = e$

$j = 5 \rightarrow \text{read}(a_3, j) = e$

→ **n=3 lemmas** in

worst-case

→ covers **single indices**

Lambda term

$\lambda_3 := \lambda x . \text{ite}(5 \leq x \wedge x < 8, e, f_a(x))$

Conflict

$j = 7 \wedge \lambda_3(j) \neq e$

Lemma

$5 \leq j \wedge j < 8 \rightarrow \lambda_3(j) = e$

→ only **one lemma** generated

→ covers **index range**

Lambda Extraction

Better Lemma Generation

Write sequence

$a_1 := \text{write}(a, 5, e)$

$a_2 := \text{write}(a_1, 6, e)$

$a_3 := \text{write}(a_2, 7, e)$

Conflict

$j = 7 \wedge \text{read}(a_3, j) \neq e$

$j = 6 \wedge \text{read}(a_3, j) \neq e$

$j = 5 \wedge \text{read}(a_3, j) \neq e$

Lemmas

$j = 7 \rightarrow \text{read}(a_3, j) = e$

$j = 6 \rightarrow \text{read}(a_3, j) = e$

$j = 5 \rightarrow \text{read}(a_3, j) = e$

→ **n=3 lemmas** in

worst-case

→ covers **single indices**

Lambda term

$\lambda_3 := \lambda x . \text{ite}(5 \leq x \wedge x < 8, e, f_a(x))$

Conflict

$j = 7 \wedge \lambda_3(j) \neq e$

Lemma

$5 \leq j \wedge j < 8 \rightarrow \lambda_3(j) = e$

→ only **one lemma** generated

→ covers **index range**

Lambda Extraction

Better Lemma Generation

Write sequence

$a_1 := \text{write}(a, 5, e)$

$a_2 := \text{write}(a_1, 6, e)$

$a_3 := \text{write}(a_2, 7, e)$

Conflict

$j = 7 \wedge \text{read}(a_3, j) \neq e$

$j = 6 \wedge \text{read}(a_3, j) \neq e$

$j = 5 \wedge \text{read}(a_3, j) \neq e$

Lemmas

$j = 7 \rightarrow \text{read}(a_3, j) = e$

$j = 6 \rightarrow \text{read}(a_3, j) = e$

$j = 5 \rightarrow \text{read}(a_3, j) = e$

→ **n=3 lemmas** in

worst-case

→ covers **single indices**

Lambda term

$\lambda_3 := \lambda x . \text{ite}(5 \leq x \wedge x < 8, e, f_a(x))$

Conflict

$j = 7 \wedge \lambda_3(j) \neq e$

Lemma

$5 \leq j \wedge j < 8 \rightarrow \lambda_3(j) = e$

→ only **one lemma** generated

→ covers **index range**

Lambda Merging

Workflow

Lambda sequence

$$\lambda_1 := \lambda z . \text{ite}(z = i_1, e, f_a(z))$$

$$\lambda_2 := \lambda y . \text{ite}(y = i_2, e, \lambda_1(y))$$

$$\lambda_3 := \lambda x . \text{ite}(x = i_3, e, \lambda_2(x))$$

→ i_1, i_2, i_3 arbitrary

Merge Lambdas $\lambda_1, \lambda_2, \lambda_3$

Simplification

$$\lambda_4 := \lambda x . \text{ite}(x = i_3 \vee x = i_2 \vee x = i_1, e, f_a(x))$$

Lambda Merging

Workflow

Lambda sequence

$$\lambda_1 := \lambda z . \text{ite}(z = i_1, e, f_a(z))$$

$$\lambda_2 := \lambda y . \text{ite}(y = i_2, e, \lambda_1(y))$$

$$\lambda_3 := \lambda x . \text{ite}(x = i_3, e, \lambda_2(x))$$

→ i_1, i_2, i_3 arbitrary

Merge Lambdas $\lambda_1, \lambda_2, \lambda_3$

$$\lambda_3 := \lambda x . \text{ite}(x = i_3, e, \lambda_2(x))$$

$$\lambda_2[y/x]$$

Simplification

$$\lambda_4 := \lambda x . \text{ite}(x = i_3 \vee x = i_2 \vee x = i_1, e, f_a(x))$$

Lambda Merging

Workflow

Lambda sequence

$$\lambda_1 := \lambda z . \text{ite}(z = i_1, e, f_a(z))$$

$$\lambda_2 := \lambda y . \text{ite}(y = i_2, e, \lambda_1(y))$$

$$\lambda_3 := \lambda x . \text{ite}(x = i_3, e, \lambda_2(x))$$

→ i_1, i_2, i_3 arbitrary

Merge Lambdas $\lambda_1, \lambda_2, \lambda_3$

$$\lambda_3 := \lambda x . \text{ite}(x = i_3, e, \text{ite}(x = i_2, e, \lambda_1(x)))$$

$$\lambda_1[z/x]$$

Simplification

$$\lambda_4 := \lambda x . \text{ite}(x = i_3 \vee x = i_2 \vee x = i_1, e, f_a(x))$$

Lambda sequence

$$\lambda_1 := \lambda z . \text{ite}(z = i_1, e, f_a(z))$$

$$\lambda_2 := \lambda y . \text{ite}(y = i_2, e, \lambda_1(y))$$

$$\lambda_3 := \lambda x . \text{ite}(x = i_3, e, \lambda_2(x))$$

→ i_1, i_2, i_3 arbitrary

Merge Lambdas $\lambda_1, \lambda_2, \lambda_3$

$$\lambda_3 := \lambda x . \text{ite}(x = i_3, e, \text{ite}(x = i_2, e, \text{ite}(x = i_1, e, f_a(x))))$$

Simplification

$$\lambda_4 := \lambda x . \text{ite}(x = i_3 \vee x = i_2 \vee x = i_1, e, f_a(x))$$

Lambda sequence

$$\lambda_1 := \lambda z . ite(z = i_1, e, f_a(z))$$

$$\lambda_2 := \lambda y . ite(y = i_2, e, \lambda_1(y))$$

$$\lambda_3 := \lambda x . ite(x = i_3, e, \lambda_2(x))$$

→ i_1, i_2, i_3 arbitrary

Merge Lambdas $\lambda_1, \lambda_2, \lambda_3$

$$\lambda_3 := \lambda x . ite(x = i_3, e, ite(x = i_2, e, ite(x = i_1, e, f_a(x))))$$

Simplification

$$\lambda_4 := \lambda x . ite(x = i_3 \vee x = i_2 \vee x = i_1, e, f_a(x))$$

Lambda Merging

Better Lemma Generation

Lambda term

$$\lambda_4 := \lambda x . ite(x = i_3 \vee x = i_2 \vee x = i_1, e, f_a(x))$$

Conflict

$$i_1 = j \wedge \lambda_4(j) \neq e$$

Lemma

$$j = i_3 \vee j = i_2 \vee j = i_1 \rightarrow \lambda_4(j) = e$$

→ covers all indices in one disjunction (one lemma generated)

- orthogonal
- not as compact as lambda extraction
- still generates better lemmas

Lambda Merging

Better Lemma Generation

Lambda term

$$\lambda_4 := \lambda x . ite(x = i_3 \vee x = i_2 \vee x = i_1, e, f_a(x))$$

Conflict

$$i_1 = j \wedge \lambda_4(j) \neq e$$

Lemma

$$j = i_3 \vee j = i_2 \vee j = i_1 \rightarrow \lambda_4(j) = e$$

→ covers all indices in one disjunction (one lemma generated)

- orthogonal
- not as compact as lambda extraction
- still generates better lemmas

Lambda Merging

Better Lemma Generation

Lambda term

$$\lambda_4 := \lambda x . ite(x = i_3 \vee x = i_2 \vee x = i_1, e, f_a(x))$$

Conflict

$$i_1 = j \wedge \lambda_4(j) \neq e$$

Lemma

$$j = i_3 \vee j = i_2 \vee j = i_1 \rightarrow \lambda_4(j) = e$$

→ covers all **indices** in one disjunction (**one lemma** generated)

- orthogonal
- not as compact as lambda extraction
- still generates **better lemmas**

Lambda Merging

Better Lemma Generation

Lambda term

$$\lambda_4 := \lambda x . ite(x = i_3 \vee x = i_2 \vee x = i_1, e, f_a(x))$$

Conflict

$$i_1 = j \wedge \lambda_4(j) \neq e$$

Lemma

$$j = i_3 \vee j = i_2 \vee j = i_1 \rightarrow \lambda_4(j) = e$$

→ covers all **indices** in one disjunction (**one lemma** generated)

- orthogonal
- not as compact as lambda extraction
- still generates **better lemmas**

Experiments

Setup

Configurations

- Boolector_{Base}
- Boolector_E
- Boolector_M
- Boolector_X
- Boolector_{XM}
- Boolector_{XME}

E ... lambda elimination enabled

M ... lambda merging enabled

X ... lambda extraction enabled

Benchmarks

- all non-extensional benchmarks from QF_ABV of SMT-LIB (13317 in total)

Limits

- 1200s **time limit**
- 7GB **memory limit**
- 1200s **penalty** if limit reached

Experiments performed on

- 2.83GHz Intel Core 2 Quad machines with 8GB RAM

Experiments

Overview

Solver	Solved	TO	MO	Time [s]
Boolector _{Base}	13242	68	7	122645
Boolector _E	13242	49	26	120659
Boolector _{XME}	13246	47	24	111114
Boolector _X	13256	54	7	99834
Boolector _M	13259	50	8	105647
Boolector _{XM}	13263	46	8	84760

TO ... time out

MO ... memory out

Time ... CPU time

Experiments

Benchmark Family Overview

Family	Boolector _{Base}		Boolector _{XM}		Extracted Patterns					Merged
	Slvd	[s]	Slvd	[s]	λ_{mset}	$\lambda_{i \rightarrow e}$	$\lambda_{i \rightarrow i+1}$			
					λ_{mcpy}	$\lambda_{i \rightarrow i}$				
bench (119)	119	2	119	0.3	208	0	34	0	0	1118
bmc (39)	38	1361	39	182	256	3	56	0	0	6010
brubiere (98)	75	29455	75	28854	0	10	0	0	0	75821
brubiere2 (22)	17	7299	20	3241	1392	0	8	0	0	4194
brubiere3 (8)	0	9600	1	8435	0	0	0	0	0	19966
btfmt (1)	1	134	1	134	0	0	0	0	0	0
calc2 (36)	36	862	36	863	0	0	0	0	0	0
dwp (4188)	4187	2668	4187	2089	42	0	0	0	0	26068
ecc (55)	54	1792	54	1845	125	0	0	0	0	0
egt (7719)	7719	222	7719	212	3893	0	0	0	0	7257
jager (2)	0	2400	0	2400	14028	0	239	0	0	153721
klee (622)	622	12942	622	154	9373	0	10049	0	0	33406
pipe (1)	1	10	1	10	0	0	0	0	0	0
platania (275)	247	42690	258	31189	0	0	0	58	120	9039
sharing (40)	40	2460	40	2458	0	0	0	0	0	0
stp (40)	34	8749	39	2695	60	0	297	0	0	498472
stp_sa (52)	52	0.7	52	0.7	0	0	0	0	0	0
totals (13317)	13242	122645	13263	84760	29377	13	10683	58	120	835072

Total extraction time: 41s

Total merge time: 24s

Experiments

Benchmark Family Overview

Family	Boolector _{Base}		Boolector _{XM}		Extracted Patterns					Merged
	Slvd	[s]	Slvd	[s]	λ_{mset}	$\lambda_{i \rightarrow e}$	$\lambda_{i \rightarrow i+1}$			
					λ_{mcpy}	$\lambda_{i \rightarrow i}$				
bench (119)	119	2	119	0.3	208	0	34	0	0	1118
bmc (39)	38	1361	39	182	256	3	56	0	0	6010
brubiere (98)	75	29455	75	28854	0	10	0	0	0	75821
brubiere2 (22)	17	7299	20	3241	1392	0	8	0	0	4194
brubiere3 (8)	0	9600	1	8435	0	0	0	0	0	19966
btfont (1)	1	134	1	134	0	0	0	0	0	0
calc2 (36)	36	862	36	863	0	0	0	0	0	0
dwp (4188)	4187	2668	4187	2089	42	0	0	0	0	26068
ecc (55)	54	1792	54	1845	125	0	0	0	0	0
egt (7719)	7719	222	7719	212	3893	0	0	0	0	7257
jager (2)	0	2400	0	2400	14028	0	239	0	0	153721
klee (622)	622	12942	622	154	9373	0	10049	0	0	33406
pipe (1)	1	10	1	10	0	0	0	0	0	0
platania (275)	247	42690	258	31189	0	0	0	58	120	9039
sharing (40)	40	2460	40	2458	0	0	0	0	0	0
stp (40)	34	8749	39	2695	60	0	297	0	0	498472
stp_sa (52)	52	0.7	52	0.7	0	0	0	0	0	0
totals (13317)	13242	122645	13263	84760	29377	13	10683	58	120	835072

Total extraction time: 41s

Total merge time: 24s

Experiments

Benchmark Family Overview

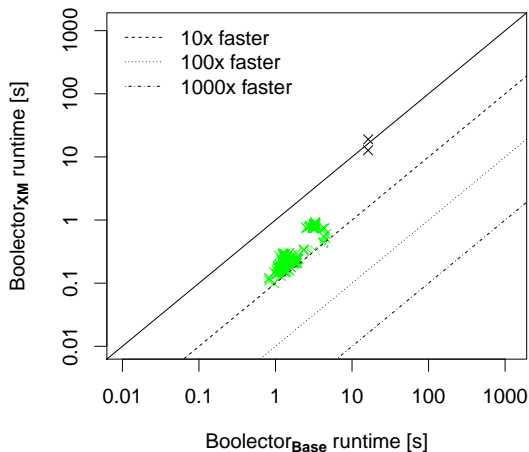
Family	Boolector _{Base}		Boolector _{XM}		Extracted Patterns					Merged
	Slvd	[s]	Slvd	[s]	λ_{mset}	$\lambda_{i \rightarrow e}$	$\lambda_{i \rightarrow i+1}$			
					λ_{mcpy}	$\lambda_{i \rightarrow i}$				
bench (119)	119	2	119	0.3	208	0	34	0	0	1118
bmc (39)	38	1361	39	182	256	3	56	0	0	6010
brubiere (98)	75	29455	75	28854	0	10	0	0	0	75821
brubiere2 (22)	17	7299	20	3241	1392	0	8	0	0	4194
brubiere3 (8)	0	9600	1	8435	0	0	0	0	0	19966
btfmt (1)	1	134	1	134	0	0	0	0	0	0
calc2 (36)	36	862	36	863	0	0	0	0	0	0
dwp (4188)	4187	2668	4187	2089	42	0	0	0	0	26068
ecc (55)	54	1792	54	1845	125	0	0	0	0	0
egt (7719)	7719	222	7719	212	3893	0	0	0	0	7257
jager (2)	0	2400	0	2400	14028	0	239	0	0	153721
klee (622)	622	12942	622	154	9373	0	10049	0	0	33406
pipe (1)	1	10	1	10	0	0	0	0	0	0
platania (275)	247	42690	258	31189	0	0	0	58	120	9039
sharing (40)	40	2460	40	2458	0	0	0	0	0	0
stp (40)	34	8749	39	2695	60	0	297	0	0	498472
stp_sa (52)	52	0.7	52	0.7	0	0	0	0	0	0
totals (13317)	13242	122645	13263	84760	29377	13	10683	58	120	835072

Total extraction time: 41s

Total merge time: 24s

Experiments

Scatter Plot klee Benchmarks (symbolic execution)



622 benchmarks

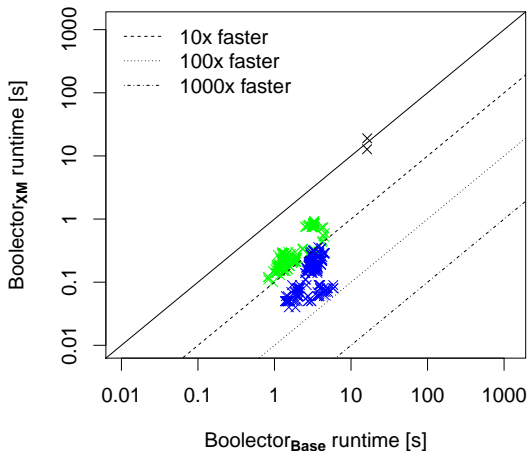
× 155 instances
2-10x faster

× 201 instances
10-100x faster

× 264 instances
100-580x faster

Experiments

Scatter Plot klee Benchmarks (symbolic execution)



622 benchmarks

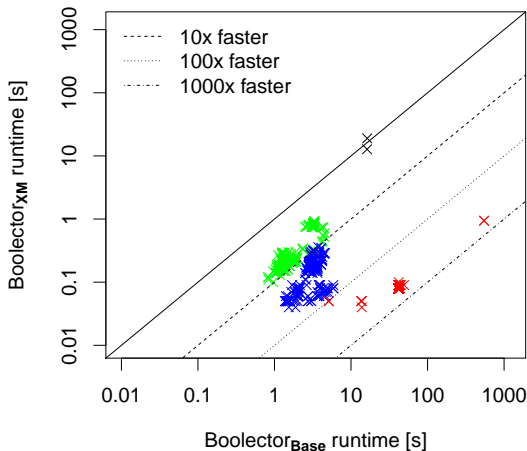
× 155 instances
2-10x faster

× 201 instances
10-100x faster

× 264 instances
100-580x faster

Experiments

Scatter Plot klee Benchmarks (symbolic execution)



622 benchmarks

× 155 instances
2-10x faster

× 201 instances
10-100x faster

× 264 instances
100-580x faster

Experiments

Lemma Generation Boolector_{Base} vs. Boolector_{XM}

Commonly solved: 13242 instances

Impact on Lemma Generation

- Boolector_{Base}: 699027 lemmas
- Boolector_{XM}: 88762 lemmas
→ Reduction by factor 7.9

Bit-blasted CNF: Reduction by 25% on average

SAT solver time

- Boolector_{Base}: 18175s
- Boolector_{XM}: 13653s
→ Reduction by 25%

Conclusion

Summary




- lambda merging **orthogonal** to lambda extraction
- both techniques improve lemma generation
- **negligible** overhead
- reduces **number of lemmas** and consequently **bit-blasted CNF**
- considerable performance improvements, particularly on **symbolic execution** benchmarks

Future Work

- more array patterns
- more expressive array theory?

Boolector is available at <http://fmv.jku.at/boolector>

References I

-  J. McCarthy. Towards a Mathematical Science of Computation. In IFIP Congress, Pages 21-28. 1962
-  R. E. Bryant and S. K. Lahiri and S. A. Seshia. Modeling and Verifying Systems Using a Logic of Counter Arithmetic with Lambda Expressions and Uninterpreted Functions. In CAV'02, volume 2404 of LNCS. Springer, 2002.
-  M. Preiner and A. Niemetz and A. Biere. Lemmas on Demand for Lambdas. In DIFTS'13, CEUR Workshop Proceedings, volume 1130. CEUR-WS.org, 2013.