Towards Unbounded Heap Support for Predicate Analysis Using SMT Arrays

Stephan Lukasczyk

2016–09–23

University of Passau, 94032 Passau, Germany

Motivation

```c
extern void __VERIFIER_error();
extern void * malloc(int);

int * getArray(int v, int p) {
  int *arr = (int*) malloc(5 * sizeof(int));
  arr[p] = v;
  return arr;
}

void main(void) {
  int val = 2;
  int pos = 3;
  int *arr = getArray(val, pos);
  int read;

  read = arr[pos];
  if (val == read)
    ERROR: __VERIFIER_error();
}
```

Motivation

```
(assert
  (let (
      (baseAddressArr (+ |__ADDRESS_OF_main::arr| 4))
      (array[p] (+ |getArray::arr@2| (* 4 |getArray::p@2|))))
    (and
      (= |main::val@2| 2)
      (= |main::pos@2| 3)
      (> |__ADDRESS_OF_main::arr| 0)
      (= |getArray::v@2| |main::val@2|)
      (= |getArray::p@2| |main::pos@2|)
      (= |getArray::__CPAchecker_TMP_0@3| |__ADDRESS_OF_malloc#2|)
      (= |getArray::arr@2| |getArray::__CPAchecker_TMP_0@3|)
      (= (*signed_int@2 array[p]) |getArray::v@2|)
      (= |getArray::__retval__@2| |getArray::arr@2|)
      (> baseAddressArr 0)
      (>= |__ADDRESS_OF_malloc#2| baseAddressArr)
      (let ((mallocOffset4 (+ |__ADDRESS_OF_malloc#2| 16)))
        (or (= mallocOffset4 array[p]) (= (*signed_int@2 mallocOffset4) (*signed_int@1 mallocOffset4))))
      (let ((mallocOffset3 (+ |__ADDRESS_OF_malloc#2| 12)))
        (or (= mallocOffset3 array[p]) (= (*signed_int@2 mallocOffset3) (*signed_int@1 mallocOffset3))))
      (let ((mallocOffset2 (+ |__ADDRESS_OF_malloc#2| 8)))
        (or (= mallocOffset2 array[p]) (= (*signed_int@2 mallocOffset2) (*signed_int@1 mallocOffset2))))
      (let ((mallocOffset1 (+ |__ADDRESS_OF_malloc#2| 4)))
        (or (= mallocOffset1 array[p]) (= (*signed_int@2 mallocOffset1) (*signed_int@1 mallocOffset1))))
      (let ((mallocOffset0 (+ |__ADDRESS_OF_malloc#2| 0)))
        (or (= mallocOffset0 array[p]) (= (*signed_int@2 mallocOffset0) (*signed_int@1 mallocOffset0))))
      (= |main::arr@3| |getArray::__retval__@2|)
      (= |main::read@3| (*signed_int@2 (+ |main::arr@3| (* 4 |main::pos@2|))))
      (= |main::val@2| |main::read@3|))))
```

Predicate Analysis

- use predicates from logics to model data states
- implemented as CPA in CPAchecker
- takes C statements
- uses Satisfiability Modulo Theories (SMT) formulae

Quantifier-free SMT Theories

- Equality and Uninterpreted Functions
- Linear Arithmetic over Integers or Reals
- Bit Vectors
- Arrays

Defining the Heap-Array Converter

- previous: two converters (simple, uninterpreted functions)
- new: *heap-array formula converter*
- SMT arrays instead of uninterpreted functions

Defining the Heap-Array Converter

- previous: two converters (simple, uninterpreted functions)
- new: *heap-array formula converter*
- SMT arrays instead of uninterpreted functions
- “simple” statements with basic theories
- SMT arrays only for heap access modelling
- heap model: one SMT array per C data type

Heap-Array Converter—Discussion

- avoid disjunctions
- lower number of formula clauses
- eliminate size bounds for arrays

Heap-Array Converter—Discussion

- avoid disjunctions
- lower number of formula clauses
- eliminate size bounds for arrays

- quantifiers for interpolation on arrays
- higher complexity for solvers

Example Using Heap-Array Converter

```
(assert
  (let ((baseAddressArr (+ |__ADDRESS_OF_main::arr| 4)))
    (and
      (= |main::val@2| 2)
      (= |main::pos@2| 3)
      (> |__ADDRESS_OF_main::arr| 0)
      (= |getArray::v@2| |main::val@2|)
      (= |getArray::p@2| |main::pos@2|)
      (= |getArray::__CPAchecker_TMP_0@3| |__ADDRESS_OF_malloc#2|)
      (= |getArray::arr@2| |getArray::__CPAchecker_TMP_0@3|)
      (= *signed_int@2 (store *signed_int@1
          (+ |getArray::arr@2| (* 4 |getArray::p@2|)) |getArray::v@2|))
      (= |getArray::__retval__@2| |getArray::arr@2|)
      (> baseAddressArr 0)
      (>= |__ADDRESS_OF_malloc#2| baseAddressArr)
      (= |main::arr@3| |getArray::__retval__@2|)
      (= |main::read@3|
        (select *signed_int@2 (+ |main::arr@3| (* 4 |main::pos@2|))))
      (= |main::val@2| |main::read@3|))))
```

Quantifiers for C Initializers

- Initializer statements in C

```
int x[10] = {0};
```

- Use universal quantifier in formula

$$init \in \mathcal{A} \forall i \in \mathbb{N}, 0 \leq i < \mathfrak{S}(init) : init[i] = 0$$

($\mathcal{A}$: set of possible arrays; function $\mathfrak{S}$ returns size of array)

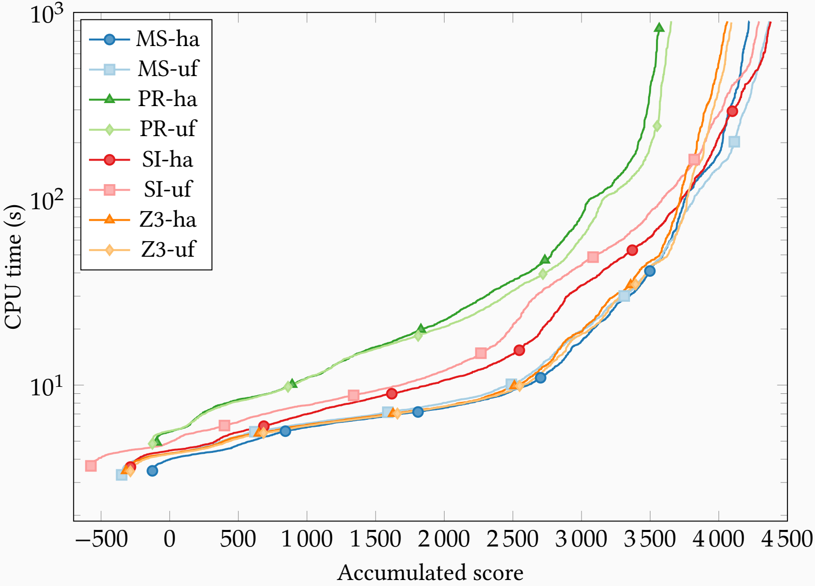
- Problem: Arrays + Quantifiers $\Rightarrow$ Undecidable

Prerequisites for the Evaluation

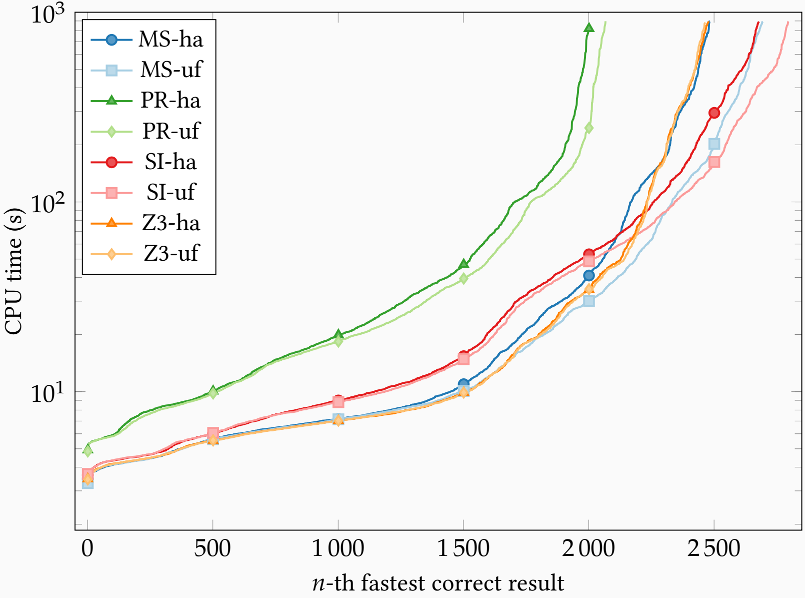
- SV-COMP Categories: ArraysReach, ControlFlow, DeviceDrivers64, ECA, HeapReach, Loops, ProductLines, Sequentialized, Simple (4 552 files)
- Customized ArraysReach (880 files)
- Each run: 900 s
- SMT solvers: MathSAT5, PRINCESS, SMTInterpol, Z3

- Machines: 2× 16 core Intel Xeon (3.4 GHz), 135 GB RAM, Ubuntu 14.04 (64bit), Kernel 4.2, Java 8
- Run limits: 15 GB RAM, two CPU cores, 13 GB Java heap, 10 MB stack size
- cf. https://research.lukasczyk.me/heaparray for supplementary web page with all results

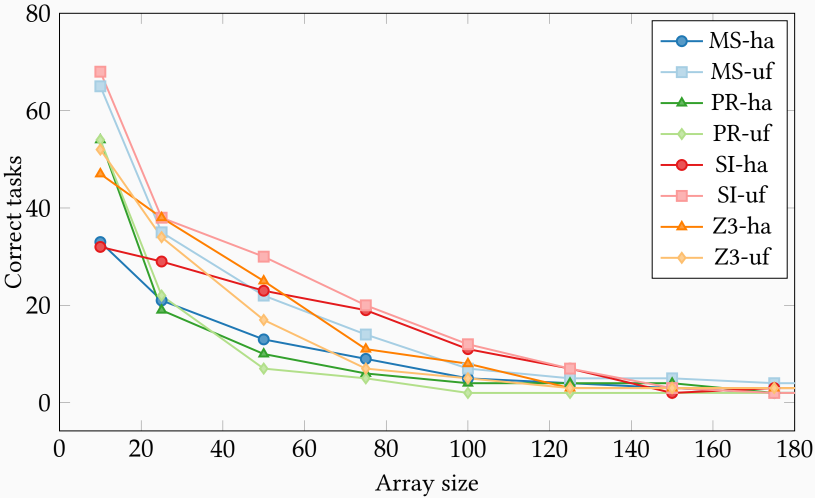
Comparison of HA and UF



Comparison of HA and UF...



Behaviour on Larger Arrays



Influence of Quantifiers on Initializers

Only on sets DeviceDrivers64, HeapReach, and Sequentialized

	PR-hq	PR-ha	Z3-hq	Z3-ha
total	2 462	2 462	2 462	2 462
correct	1 177	1 271	1 454	1 470
true	1 065	1 151	1 278	1 276
false	112	120	176	194
incorrect	0	1	17	12
true	0	0	13	0
false	0	1	4	12
score (4 478)	2 242	2 406	2 252	2 554

Summary

- Implementation and evaluation of heap-array converter
- Arrays harder for solvers than uninterpreted functions
- Quantifier necessary for array interpolation
  $\Rightarrow$ Undecidable
- Better results on arrays with sizes between 25 and 150, but more tasks necessary
- Quantifiers difficult for array initializers
- Unbounding UFs with quantifiers
  (done by Philipp Wendler)

Summary

- Implementation and evaluation of heap-array converter
- Arrays harder for solvers than uninterpreted functions
- Quantifier necessary for array interpolation
  $\Rightarrow$ Undecidable
- Better results on arrays with sizes between 25 and 150, but more tasks necessary
- Quantifiers difficult for array initializers
- Unbounding UFs with quantifiers
  (done by Philipp Wendler)

`cpa.predicate.useArraysForHeap`

`cpa.predicate.useQuantifiersOnArrays`