On Solving Universally Quantified Horn Clauses [SAS2013]

Nicolaj Bjorner, *Microsoft Research*
Ken McMillan, *Microsoft Research*
Andrey Rybalchenko, *Microsoft Research* and *Technische Universitat Munchen*
Presented by
Julien Braine, *ENS Lyon*

31 January 2017

Program Verification

Domain

Bugs in programs

Two complementary techniques

- Finding Bugs
- **Ensuring the absence of bugs**

Main use

High security softwares: avionics, cars, satellites, ...

Motivating Example

Motivation for arrays

Arrays are a first start to deal with memory

```
1  char* some_array_value();
2  #define N 100
3
4  int main(){
5    char a[N];
6    char* b = some_array_value();
7    for (int i=0 ; i < N; i++) {   (*)
8      a[i] = b[i];
9    }
10   int j = rand()%N;
11   assert(a[j]==b[j]);
12 }
```

Figure: Motivating example: array copy

Language Independent Representation

Goal

Verification problem = satisfiability of a formula

Motivation

- Non ambiguous semantics
- Very few elements of language to deal with
- Can reuse logic results

How

Horn clauses = formula expressing derivation trees (Prolog like)

Horn solvers

- SMT solvers such as Z3
- Datalog techniques such as Spacer
- Abstract Interpretation techniques

Horn clauses

$$
\begin{aligned}
&\mathit{Start}(a, b)\\
&\mathit{Loop}(a, b, 0) \leftarrow \mathit{Start}(a, b)\\
&\mathit{Loop}(a', b, i+1) \leftarrow \mathit{Loop}(a, b, i) \wedge a' = a[i \leftarrow b[i]] \wedge i < N\\
&\mathit{EndLoop}(a, b) \leftarrow \mathit{Loop}(a, b, i) \wedge i \geq N\\
&\mathit{Assert}(a, b, j) \leftarrow \mathit{EndLoop}(a, b) \wedge j = \mathit{randval}\%N\\
&\mathit{false} \leftarrow \mathit{Assert}(a, b, j) \wedge a[j] \neq b[j]
\end{aligned}
$$

Figure: Horn clauses for array copy

Main difficulty

Useful invariants on arrays are quantified: $\forall k, k < i \Rightarrow a[k] = b[k]$

Moving quantifiers from invariants to formula

Main Idea

Enforce the invariant to be quantified: $P(a, \ldots)$ replaced by $\forall x, P(x, a, \ldots)$
$\Rightarrow$ solver does not need to search for a quantified P anymore

Transformation of Horn clauses

$P'(a') \leftarrow P(a) \wedge \phi(a, a')$
becomes:
$\forall y, P'(y, a') \leftarrow \forall x, P(x, a) \wedge \phi(a, a')$
and simplified to:
$P'(y, a') \leftarrow \forall x, P(x, a) \wedge \phi(a, a')$

Problem

How to remove quantifiers in the premises ?

Instantiating the Quantifiers

$\forall$ = infinite conjunction

$$P'(y, a') \leftarrow \forall x, P(x, a) \wedge \phi(a, a')$$
$$\equiv P'(y, a') \leftarrow \bigwedge_x P(x, a) \wedge \phi(a, a')$$

Theorem

Using only a subset of a conjunction in the premises is sound

Solution

Select a finite number of relevant x

Technique

The paper suggests "E-matching" $\Rightarrow$ Syntax based

The case of arrays

Instantiate on array reads and the result variable

$$Start(x_a, a, x_b, b)$$

$$Loop(x_a, a, x_b, b, 0) \leftarrow Start(x_a, a, x_b, b)$$

$$Loop(x'_a, a', x_b, b, i+1) \leftarrow \left\{ \begin{array}{l} Loop(x_a, a, x_b, b, i) \\ Loop(x_a, a, i, b, i) \end{array} \right\} \wedge a' = a[i \leftarrow b[i]] \wedge i < N$$

$$EndLoop(x_a, a, x_b, b) \leftarrow Loop(x_a, a, x_b, b, i) \wedge i \geq N$$

$$Assert(x_a, a, x_b, b, j) \leftarrow EndLoop(x_a, a, x_b, b) \wedge j = randval\%N$$

$$False \leftarrow \left\{ \begin{array}{l} Assert(x_a, a, x_b, b, j) \\ Assert(x_a, a, j, b, j) \\ Assert(j, a, x_b, b, j) \\ Assert(j, a, j, b, j) \end{array} \right\} \wedge a[j] \neq b[j]$$

Figure: Quantifier transformation of the Horn clauses of array copy

Theorem

Instantiating on array reads and the result variable is a complete instantiation (not in the article)

Experimental Results

Results

Showing those of SAS2016 for arrays (same resulting clauses)

Timing in seconds, CPU time.

The machine has 32 i3-3110M cores, 64 GiB RAM

Benchmark	Z3/PDR		Z3/Spacer		Eldarica		Distinct number
	Res	Time	Res	Time	Res	Time	
bin search array	sat	0.38	timeout(300s)		Exception		N=1
find mini array	sat	2.99	sat	1.16	sat	58.97	N=1
bubble sort array	sat	4.42	sat	3.17	sat	68.86	N=2
insert sort array	sat	146.42	timeout(300s)		timeout(300s)		N=2
insert sort list	timeout(300s)		sat	1.35	timeout(300s)		N=2

Conclusion

Sorting algorithms in minutes!

Conclusion

Main Contribution of the paper

A general technique to obtain quantified invariants

Consequences

We can prove sorting algorithms on arrays !

Limits

How to find the relevant instantiation ?

Thank you for your attention !

Any Questions ?