Accelerating Invariant Generation

Kumar Madhukar, Björn Wachter, Daniel Kroening
Matt Lewis and Mandayam Srivas

Tata Research Development and Design Center
University of Oxford
Chennai Mathematical Institute

Formal Methods in Computer-Aided Design
September 27-30, 2015

Background

- program analyzers often rely on invariant generation to reason about loops
- *unrolling* is ineffective for non-trivial programs
- *acceleration* summarizes loops by computing a closed-form representation
- derive loop "accelerators" from the closed-form

This paper

- two conjectures:
  1. accelerators support the invariant synthesis performed by program analyzers, irrespective of the underlying approach
  2. analyzers supported by acceleration outperform other state-of-the-art tools performing similar analysis
- is an experimental evaluation of our conjectures

An example

```
#define a 2

int main()
  unsigned int i, j, n, sn = 0;
  j = i;

  while(i < n)
    sn = sn + a;
    i++;

  assert((sn == (n-j)*a) || sn == 0);
```

Acceleration

- general case is as difficult as the original verification problem
- transitive closure is rarely effectively computable
- frequently not possible to obtain a *precise* accelerator
- can be over-approximative or under-approximative
- often tuned to the analysis technique to be applied subsequently

  e.g., abstract interpretation or predicate abstraction

Our acceleration method

- based on templates; uses polynomials of degree 2
- relies on constraint solvers to compute accelerators
- added to the programs as additional paths, with a non-deterministic choice
- the transformation preserves safety - the acceleration neither over- nor under-approximates

Accelerated example

```
int nondet_int(); unsigned nondet_uint();
#define a 2

int main()
  unsigned int i, j, n, k, sn = 0;
  j = i;
  while(i < n)
    if(nondet_int()) // accelerate
      k = nondet_uint(); sn = sn + k*a; i = i + k;
      assume(i <= n); // no overflow

    else // original body
      sn = sn + a; i++;

  assert((sn == (n-j)*a) || sn == 0);
```

Experimental setup: benchmarks

- 201 benchmarks: 138 safe, 63 unsafe
  - InvGen and Dagger benchmark suites
  - benchmark suite listed in "Beautiful Interpolants" paper at CAV 2013
  - the *loops* category in SV-COMP 2015
  - acceleration benchmarks in the regression suite of CBMC
- removed some examples: those not supported by the acceleration (arrays in general), those with syntax errors

Experimental setup: tools

- compared CBMC and IMPARA (with and without acceleration)
- very different techniques: CBMC is a bounded model checker; IMPARA uses LAWI
- compared accelerated results with UFO and CPACHECKER
- UFO: abstract interpretation with numerical domains + ability to generalize using interpolants, in an abstraction refinement loop
- CPACHECKER: broad portfolio of techniques: interpolation, abstract interpretation, predicate abstraction, etc.

Experimental setup: overall

- dual-core machine running at 2.73 GHz with 2 GB RAM
- timeout after 60 seconds
- benchmarks, tool-specific options and results available at

  `http://www.cmi.ac.in/~madhukar/fmcad15`

Results

Tools	Number of instances					Score
	correct proofs	wrong proofs	correct alarms	wrong alarms	no results	
CPACHECKER 1.3.4	83	16	35	14	53	**−75**
UFO SV-COMP 2014	52	2	18	2	127	**86**
CBMC r4503	32	0	35	0	134	**99**
+ Acceleration	53	0	45	12	91	**79**
IMPARA 0.2	78	1	36	15	71	**90**
+ Acceleration	86	0	47	12	56	**147**

Score = (2·*correct proofs*) − (12·*wrong proofs*)+*correct alarms*−(6·*wrong alarms*) - as per SV-COMP 2015.

Results

Tools	Number of instances					Score
	correct proofs	wrong proofs	correct alarms	wrong alarms	no results	
CPACHECKER 1.3.4	**83**	**16**	**35**	**14**	**53**	−75
UFO SV-COMP 2014	**52**	**2**	**18**	**2**	**127**	86
CBMC r4503	32	0	35	0	134	99
+ Acceleration	53	0	45	12	91	79
IMPARA 0.2	**78**	**1**	**36**	**15**	**71**	90
+ Acceleration	**86**	**0**	**47**	**12**	**56**	147

- IMPARA + Acceleration clearly outperforms IMPARA, UFO and CPACHECKER
- increase in correct proofs as well as correct alarms

Results

Tools	Number of instances					Score
	correct proofs	wrong proofs	correct alarms	wrong alarms	no results	
CPAchecker 1.3.4	**83**	16	35	14	53	−75
Ufo SV-COMP 2014	52	2	18	2	127	86
Cbmc r4503	32	0	35	0	134	99
+ Acceleration	53	0	45	12	91	79
Impara 0.2	78	1	36	15	71	90
+ Acceleration	**86**	0	47	12	56	147

- CPAchecker comes close in the number of correct proofs
- uses a broad portfolio of techniques

Results

Tools	Number of instances					Score
	correct proofs	wrong proofs	correct alarms	wrong alarms	no results	
CPAchecker 1.3.4	83	16	35	14	53	−75
Ufo SV-COMP 2014	52	2	18	2	127	86
Cbmc r4503	**32**	**0**	**35**	**0**	**134**	99
+ Acceleration	**53**	**0**	**45**	**12**	**91**	79
Impara 0.2	**78**	**1**	**36**	**15**	**71**	90
+ Acceleration	**86**	**0**	**47**	**12**	**56**	147

- both Impara and Cbmc are characterized by very weak invariant inference
- expected to benefit substantially from acceleration

Results

Tools	Number of instances					Score
	correct proofs	wrong proofs	correct alarms	wrong alarms	no results	
CPAchecker 1.3.4	83	16	35	14	53	−75
Ufo SV-COMP 2014	52	2	18	2	127	86
Cbmc r4503	**32**	0	**35**	0	134	99
+ Acceleration	**53**	0	**45**	12	91	79
Impara 0.2	78	1	36	15	71	90
+ Acceleration	86	0	47	12	56	147

- benefit for tools making a monolithic SAT query (e.g., Cbmc) is evident
- many more proofs and counterexamples with a far lesser unwinding

Results

Tools	Number of instances					Score
	correct proofs	wrong proofs	correct alarms	wrong alarms	no results	
CPACHECKER 1.3.4	**83**	**16**	**35**	**14**	**53**	−75
UFO SV-COMP 2014	**52**	**2**	**18**	**2**	**127**	86
CBMC r4503	32	0	35	0	134	99
+ Acceleration	53	0	45	12	91	79
IMPARA 0.2	78	1	36	15	71	90
+ Acceleration	86	0	47	12	56	147

- acceleration would help UFO and CPACHECKER as well
- an interpolation procedure on a loop unwinding gets overly specific interpolants (Beyer et al., PLDI 2007)
- presenting transitive closure of loop to the interpolating procedure helps

Results

Tools	Number of instances					Score
	correct proofs	wrong proofs	correct alarms	wrong alarms	no results	
CPACHECKER 1.3.4	83	**16**	35	14	53	−75
UFO SV-COMP 2014	52	2	18	2	127	86
CBMC r4503	32	0	35	0	134	99
+ Acceleration	53	0	45	12	91	79
IMPARA 0.2	78	1	36	15	71	90
+ Acceleration	86	0	47	12	56	147

- wrong proofs for CPACHECKER mainly arise from deriving mathematical-integer invariants
- these invariants do not hold in presence of overflows

Results

Tools	Number of instances					Score
	correct proofs	wrong proofs	correct alarms	wrong alarms	no results	
CPAchecker 1.3.4	83	16	35	14	53	−75
Ufo SV-COMP 2014	52	2	18	2	127	86
Cbmc r4503	32	0	35	**0**	134	**99**
+ Acceleration	53	0	45	**12**	91	**79**
Impara 0.2	78	1	36	15	71	90
+ Acceleration	86	0	47	12	56	147

- the score dips for Cbmc + Acceleration, as compared to Cbmc, due to the wrong alarms (that are heavily penalized at SV-COMP)
- miscategorized as *safe*; actually *unsafe* due to overflow

Acceleration helps generalization in LAWI

```
int main()
  unsigned int n = nondet_uint();
  int x = n;
  int y = 0;

  // loop invariant: x + y == n
  while(x > 0)
    x = x - 1;
    y = y + 1;

  assert(y == n);
```

- Without acceleration, IMPARA falls back to loop unwinding
- gets the loops invariant for the accelerated program

Caveats

- only an experimental evaluation
- over "academic" benchmarks
- couldn't actually try accelerated benchmarks on other tools; CBMC's acceleration works on goto-binaries
- there is a `--dump-c` option (experimental)

Conclusion

- quantified the benefits of acceleration for checking safety properties
- source-level transformation enables integration with other invariant generation techniques
- better quantifier handling should boost it further
- invariants over the interval domain may help in ruling out overflows

References

- D. Kroening, M. Lewis, and G. Weissenbacher, "Under-approximating loops in C programs for fast counterexample detection," in Computer Aided Verification (CAV), ser. LNCS, vol. 8044. Springer, 2013.
- D. Kroening, M. Lewis, and G. Weissenbacher, "Proving safety with trace automata and bounded model checking," in Formal Methods (FM), ser. LNCS, vol. 9109. Springer, 2015.

Thank you!

Thank you!

Questions?