TATA CONSULTANCY SERVICES
Experience certainty.

Proving Properties of non-array Programs

Thanks to Priyanka Darke

Tata Research Development and Design Centre, Pune, India

December 13, 2017

Background and Motivation

- Bounded Model Checking (BMC) can analyze programs up to known bounds
  - By unrolling loops a finite number of times
- Suitable to find errors

- Limitation : BMC is incomplete
  - Can find errors
  - Cannot prove properties of programs

- Problem :
  - Real world programs can have infinite bounds
  - Loops of large, unknown or infinite bounds
    - Lead to a large number of loop unrollings
  - BMC cannot scale up for such programs

```
if(cond){
  Body;
  if(cond){
    Body;
    if(cond){
      Body;
      assert(!cond);
    }
  }
}
```

Ensures soundness

Aim

- Aim
  - To *prove program properties* using CBMC
    - in the presence of *loops* with large, unknown, or infinite bounds

- Proposed Solution : Loop Abstraction
  - Over-approximate outputs of the loop
    - Outputs – numerical variables, arrays
  - Replace loops in the program by abstract loops of a known small bound
  - Loops abstracted program allows more runs than the original program
  - If the property hold in the abstraction, then it holds in the original program

Output Abstraction - A Simple Abstraction Technique

Original code	Arbitrary Iteration

`unsigned i`
 	
`for(i=0; i<100001; i++)`
 	
`assert(i>=0 && i<=100001)`
	`i = *`
	`assume(i<100001)`
	`i++`
	
	`assume(!(i<100001))`
	
	`assert(i>=0 && i<=100001)`

A More Complex Example

```
nPO = nP-1
while(nP!=nPO)
    state = L
    nPO=nP
    r=glb
    if(r && r -> status )
      glb = r -> next
      state = U
      r -> status = 0
      nP++

assert (state != U)
state = U
```

```
nPO = nP-1;
nP, nPO, r = *
    state = L
    nPO=nP
    r=glb
    if(r && r -> status )
      glb = r -> next
      state = U
      r -> status = 0
      nP++
assume( nP == nPO)
assert (state != U)
state = U
```

Terminology

- Outputs of the loop
  - numerical variables
- Types of loop variables
  - Inputs (I) : Only read (`last`)
  - Outputs (O) : Modified (`c, st`)
  - Input-outputs (IO) : Read and modified (`c`)
  - Pure outputs (PO) : Only modified (`st`)
- $O = IO \cup PO$
- Values of variables in O depend on $I \cup IO$

```
for(c =0; c <=200; c++)
   if(c == last)
   {
       st = 0;
       break;
   }
```

Output Abstraction

Original loop	Abstract loop

`while(c)`
`{`
`  L;`
`}`
`assert(P);`
	`for(t=0; t < |O| && c; t++)`
	`{`
	`  nondet(IO);`
	`  assume(c);`
	`  L;`
	`}`
	`assume(!c);`
	`assert(P);`

Output Abstraction - A Simple Abstraction Technique

Original code

```
unsigned i

for(i=0; i<100001; i++)

assert(i>=0 && i<=100001)
```

Abstracted Code

```
for (t=0; t < 1 && i < 100001; i++)
   i = *
   assume(i<100001)
   i++

assume(!(i<100001))

assert(i>=0 && i<=100001)
```

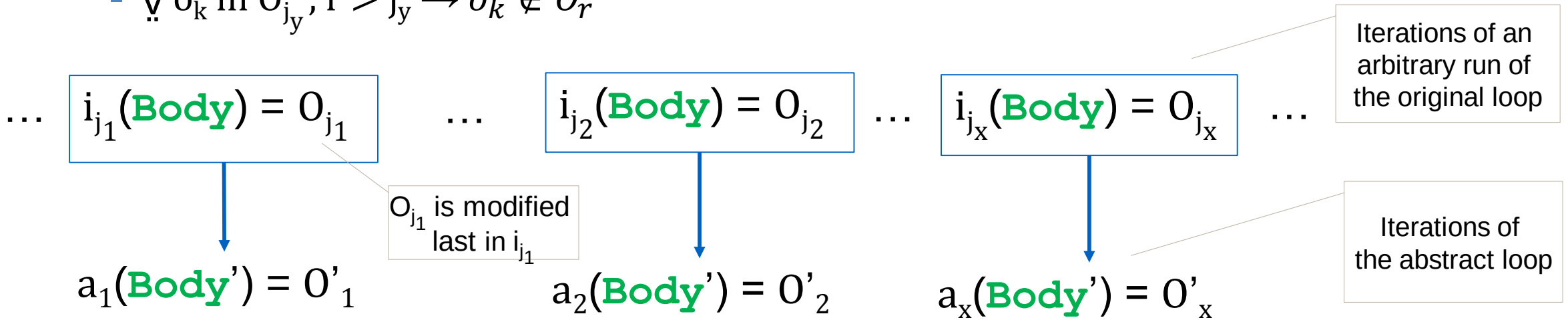
Abstract loop

```
for(t=0; t < |O| && c; t++)
{
        nondet(IO);
        assume(c);
        L;
}
assume(!c);
assert(P);
```

inty.

Output Abstraction – Proof Outline

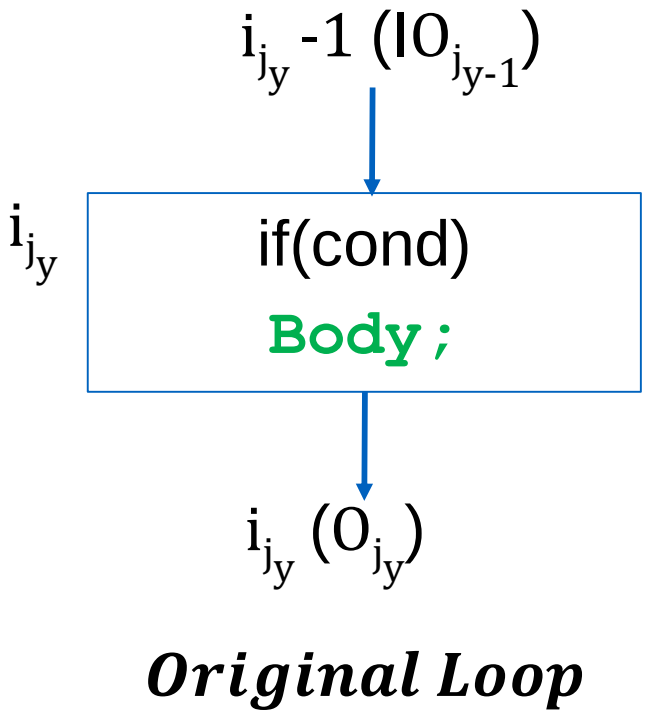
- Abstracts the set of output variables (O) to reduce loop bounds
- Consider an arbitrary run of the loop.
  - Let n iterations of the original loop $[\ i_1, i_2, i_3, ..., i_n\ ]$
  - An abstract loop with iterations $[a_1, a_2, a_3, ..., a_x]$, $x \leq n$
  - a_1 (maps to) i_{j_1}, $a_2 \equiv i_{j_2}$, ..., $a_x \equiv i_{j_x}$
  - i_{j_y} modifies the set of outputs O_{j_y} , $0 < y \leq x$, $0 \leq |O_{j_y}|$ such that
    - $\forall\ o_k$ in O_{j_y} , $r > j_y \rightarrow o_k \notin O_r$



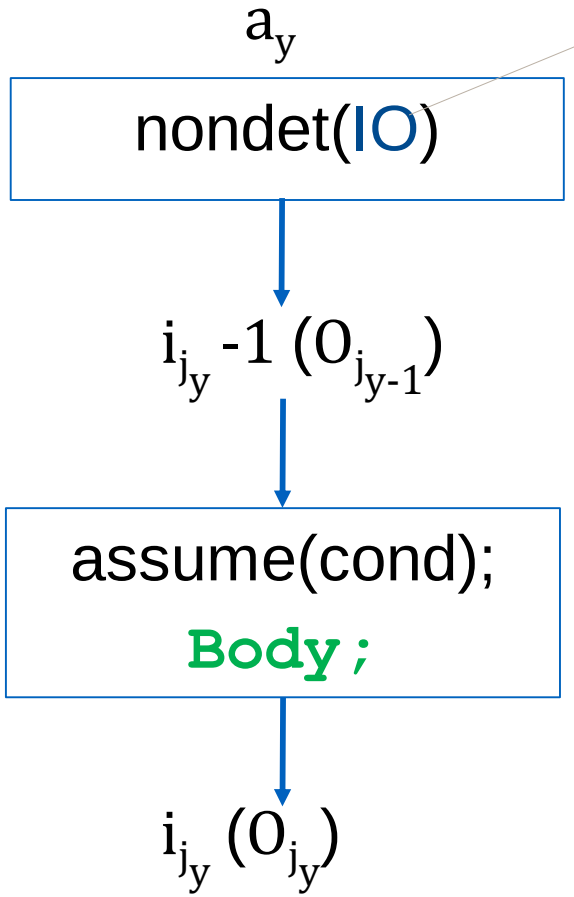
P. Darke, M. Khanzode, A. Nair, U. Shrotri, and R. Venkatesh. Precise analysis of large industry code. In APSEC, pages 306–309, 2012.

Output Abstraction

- Abstraction of iteration i_{j_y}, $0 < y \leq |O|$
- $i_0(O)$ = initial state



Original Loop



IO variables modified in every iteration. Decide the path in each iteration.

Abstract Loop

P. Darke, M. Khanzode, A. Nair, U. Shrotri, and R. Venkatesh. Precise analysis of large industry code. In APSEC, pages 306–309, 2012.

Output Abstraction Algorithm with an Example

Code before Loop Abstraction

```
1.  void xmit (int nP)
2.  {
3.       nPO = nP-1;
4.       while(nP!=nPO) {
5.          state = L;
6.          nPO=nP;
7.          r=glb;
8.          if(r && r -> status ) {
9.            glb = r -> next;
10.           state = U;
11.           r -> status = 0;
12.           nP++;
13.         }
14.    }
15.    assert (state != U) ;
16.    state = U;
17. }
```

Example from : Thomas Ball and Sriram K. Rajamani. Automatically validating temporal safety properties of interfaces. pages 103–122. Springer-Verlag, 2001.

Output Abstraction Algorithm

Code before Loop Abstraction

```
1.  void xmit (int nP)
2.  {
3.       nPO = nP-1;
4.       while(nP!=nPO) {
5.          state = L;
6.          nPO=nP;
7.          r=glb;
8.          if(r && r -> status ) {
9.            glb = r -> next;
10.           state = U;
11.           r -> status = 0;
12.           nP++;
13.         }
14.    }
15.    assert (state != U) ;
16.    state = U;
17. }
```

Input Variables :

- Only read in the loop body.
- No need to abstract.
- Example
  1. **L** (line No. 5)
  2. **U** (line No. 10)

Output Abstraction Algorithm

Code before Loop Abstraction

```
1.  void xmit (int nP)
2.  {
3.       nPO = nP-1;
4.       while(nP!=nPO) {
5.          state = L;
6.          nPO = nP;
7.          r=glb;
8.          if(r && r -> status ) {
9.            glb = r -> next;
10.           state = U;
11.           r -> status = 0;
12.           nP++;
13.         }
14.    }
15.    assert (state != U) ;
16.    state = U;
17. }
```

Input Variables :

- Only read in the loop body.
- No need to abstract.
- Example
  1. **L** (line No. 5)
  2. **U** (line No. 10)

Input-Output Variables :

- Read first and then modified in the loop.
- For ex.1. nPO (line No. 4,6)
  2. nP (line No. 6,12)
  3. glb (line No. 7, 9)

Output Abstraction Algorithm

Code before Loop Abstraction

```
1.  void xmit (int nP)
2.  {
3.       nPO = nP-1;
4.       while(nP!=nPO) {
5.          state = L;
6.          nPO=nP;
7.          r=glb;
8.          if(r && r -> status ) {
9.            glb = r -> next;
10.           state = U;
11.           r -> status = 0;
12.           nP++;
13.         }
14.    }
15.    assert (state != U) ;
16.    state = U;
17. }
```

Input Variables :

- Only read in the loop body.
- No need to abstract.
- Example
  1. **L** (line No. 5)
  2. **U** (line No. 10)

Input-Output Variables :

- Read first and then modified in the loop.
- For ex.1. nPO (line No. 4,6)
  2. nP (line No. 6,12)
  3. glb (line No. 7, 9)

Pure Output Variables :

- Only modified in the loop.
- Generated.
- For ex. 1. state (line No. 5, 10)

Output Abstraction Algorithm

Code before Loop Abstraction

```
1.  void xmit (int nP)
2.  {
3.       nPO = nP-1;
4.       while(nP!=nPO) {
5.          state = L;
6.          nPO=nP;
7.          r=glb;
8.          if(r && r -> status ) {
9.            glb = r -> next;
10.           state = U;
11.           r -> status = 0;
12.           nP++;
13.         }
14.    }
15.    assert (state != U) ;
16.    state = U;
17. }
```

Input Variables :

- Only read in the loop body.
- No need to abstract.
- Example
  1. **L** (line No. 5)
  2. **U** (line No. 10)

Input-Output Variables :

- Read first and then modified in the loop.
- For ex.1. nPO (line No. 4,6)
  2. nP (line No. 6,12)
  3. glb (line No. 7, 9)

Pure Output Variables :

- Only modified in the loop.
- Generated.
- For ex. 1. state (line No. 5, 10)

Number of output variables = |O| = 4

Output Abstraction Algorithm

Code before Loop Abstraction

```
1.  void xmit (int nP)
2.  {
3.       nPO = nP-1;
4.       while(nP!=nPO) {
5.          state = L;
6.          nPO=nP;
7.          r=glb;
8.          if(r && r -> status ) {
9.            glb = r -> next;
10.           state = U;
11.           r -> status = 0;
12.           nP++;
13.         }
14.    }
15.    assert (state != U) ;
16.    state = U;
17. }
```

Code after Loop Abstraction

```
1.  void xmit (int nP)
2.  {
3.       nPO = nP - 1 ;
4.       for( i =0; i<4 && ( nP != nPO ); i++){
5.          nondet( nPO, nP, glb );
6.          assume( nP != nPO );
7.          state = L;
8.          nPO=nP;
9.          r=glb;
10.         if(r && r -> status ){
11.             glb = r -> next;
12.             state = U;  r -> status = 0;
13.             nP++;
14.         }
15.      }
16.      assume ( !(nP != nPO) );
17.      assert (state != U) ;
18.      state = U;
19. }
```

Output Abstraction – Loop Bound Computation

1. Number of output variables = |O| = 4
2. Number of unique blocks modifying pure outputs, b = 1

Number of abs. loop iterations = min(|O|, b)

Code after Loop Abstraction

```
1.  void xmit (int nP)
2.  {
3.        nPO = nP - 1 ;
4.        for( i =0; i<1 && ( nP != nPO ); i++){
5.            nondet( nPO, nP, glb );
6.            assume( nP != nPO );
7.            state = L;
8.            nPO=nP;
9.            r=glb;
10.           if(r && r -> status ){
11.               glb = r -> next;
12.               state = U;   r -> status = 0;
13.               nP++;
14.           }
15.       }
16.       assume ( !(nP != nPO) );
17.       assert (state != U) ;
18.       state = U;
19. }
```

Output Abstraction - Limitation

Original code	Output abstraction

```
int j, i
j=0

for(i=0; i<100001; i++)
    j++


 assert(j>=0 && j<=100001)
```

```
int j, i
j=0 i=0

if(i<100001)
  i = *
  j = *
  assume(i<100001)
  j++ i++

assume(!(i<100001))

assert(j>=0 && j<=100001)
```

Abstract Loop

Assertion fails

Output Acceleration

Original code	Output acceleration

```
void main()
{
 int j, i;
 j=0;

 for(i=0; i<100001; i++) {
     j++;
 }

  assert(j>=0 && j<=100001);
}
```

```
void main()
{
 int j, i;
 j=0; i=0;

 if(i<100001){
   k = 100000
   i = 0 + k
   j = 0 + k
 }
 assert(j>=0 && j<=100001);
}
```

Acceleration

Assertion validated

Loop Abstraction followed by BMC (LABMC)

- A refined form of output abstraction
- Reduces bounds of the program to be analyzed
  - By reducing the bounds of the abstract loops
- Abstract acceleration of the IO variables
  - A **more precise** abstraction than nondeterministic assignments
  - Captures the effect of a sequence of assignments using closed forms
  - Over-approximation
    - A non-deterministically chosen length of the sequence of assignments
- Induction
  - When the property is inside the loop

Priyanka Darke, Bharti Chimdyalwar, R. Venkatesh, Ulka Shrotri, and Ravindra Metta. Overapproximating loops to prove properties using bounded model checking. In DATE, 2015.

Abstract Acceleration and its Forms

- To captures the effect of a sequence of assignments using closed forms
  - Can handle path predicates
- Let io_{k} be the value of variable io after k iterations, $io_{(\mathrm{k},\,\mathrm{a})}$ an abstraction of io_{k}.

Type of I/O	Example	Closed Form	Abstract acceleration

Non-recurrent	`for(i=0;i<10; i++)`
	`  if(x) x = 10;`
	`  else x = 20;`
		$io_{(\mathrm{k},\,\mathrm{a})} = \mathrm{io}_0 \,\|\|\, \Upsilon_1 \,\|\|\, \Upsilon_2 \,\|\|\, \ldots \,\|\|\, \Upsilon_{\mathrm{r}}$	`x = x0||10||20;`

Self-recurrent	`for(x=0;x<10;)`
	`  if(*) x+=1;`
		$io_{(\mathrm{k},\,\mathrm{a})} = io_0 + \sum_{i=1}^{e} k_i\beta_i, \quad k_i \le k$	`k = *`
			`k1 = *;`
			`k1 <= k`
			`x = x0 + 1*k1;`

Self-recurrent – modification along all paths	`for(x=0;x<10;x++);`	$io_{(\mathrm{k},\,\mathrm{a})} = io_0 + \sum_{i=1}^{e} k_i\beta_i, \quad k_i = k$	`k = *;` `x = x0 + 1*k;`

Abstract Acceleration and its Forms

Type of I/O	Example	Closed Form	Abstract acceleration

Self-recurrent – structural refinement	`for(i=0;i<10; i++)`
	`  if(*) x = x++;`
	`  else x = x+2;`
		$io_{(k,a)} = io_0 + \sum_{i=1}^{e} k_i\beta_i \;, \Sigma k_i = k$	**`k = *;`**
			k_1 `= *;`
			k_2 `= *;`
			`assume(` $k == k_1 + k_2$ `);`
			`x =` x_0 `+ 1*`k_1 `+ 2*`k_2`;

Self-recurrent with reset by a constant	`for(x=0; x<10; x++)`
	`  if(*) x = 5;`
		$io_{(k,a)} = io' + \sum_{i=1}^{e} k_i\beta_i \;,$
		$io' = io_0 \mid io' = \Upsilon_j,$
		$1 \le j \le r, \quad \Sigma k_i \le k$
			`k1 = *;`
			`assume(k1 <= k )`
			`x =` **(x_0`||5)`** `+` **`k1*1;`**

Other recurrences or non-linear expressions	`e.g. while(*)x=x<<1;`	$io_{(k,a)} = *$	`x = *;`

Abstract Acceleration

Original code	Loop accelerated code

```
void main()
{
 int j, i;
 j=0;

 for(i=0; i<100000-1; i++) {
     j++;
 }

 assert(j>=0 && j<=100000-1);
}
```

```
void main()
{
 int j, i, t;
 j=0; i=0;
 for(t=0; t<1 && i<1000000-1; t++){
  k = *;
  j = 0 + k*1;
  i = 0 + k*1;
  assume(i<100000-1);
  j++;i++;
 }
 assume(!(i<100000-1));

 assert(j>=0 && j<=100000-1);
}
```

Abstract Loop

Assertion validated

Induction

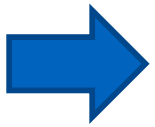
- Induction over the iterations - assertion should hold for all iterations
- Applied when the assertion lies the inside loop body

Original loop	Base case, P(1)	Induction Hypothesis, P(k)

`while(c){`
`  L;`
`  assert(P);`
`}`
	`//First iteration`
	`  if(c){`
	`  L;`
	`  assert(P);`
	`  }`
		`//induction hypothesis`
		`//k iterations`
		`for(t=0; t<|O| && c; t++) {`
		`  accelerate(IO);`
		`  assume(c);`
		`  L;`
		`  assume(P); }`
		`//(k+1)th iteration`
		`if(c) {  L;`
		`  assert(P); }`
		`assume(!c);`

Induction Example

```
while(c<2000)
{
   assert(!(c>2000));
   c++;
}
```

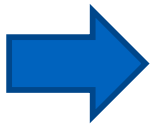


```
//Base case
if(c<2000) {
  assert(!c>2000);
  c++;
}
//Hypothesis
for(t=0; t<1 && c<2000; t++){
  k = *;
  c = c0 + k*1;
  assume(c<2000);
  assume(!c>2000);
  c++;
}
if(c<2000) {
  assert(!c>2000);
  c++;
}
assume(!(c<2000));
```

Benchmark Example

Code before Loop Abstraction

```
j = 0
i = *
assume(0<= i && i <= LARGE_INT)
n = i
while (1)
  assert(i >= 0)
  i = i - 1
  j = j + 1
  if (j >= n)
    break
```



Code after Loop Abstraction

```
  …
  assume(0<= i && i <= LARGE_INT)
  … //Base case
  …
//Hypothesis
   for(t=0; t<1 && 1; t++)
     k = *
     i = i0[n] - k*1
     j = j0[0] + k*1
     assume(1)
     assume(i >= 0)
     i = i - 1
     j = j + 1
     if (j >= n)
       break_ = 1

   if( 1 && !break_ )
     assert(i >= 0)
     … //Body;
   assume(!1)
```

Sliced SV-COMP'18 benchmark - https://github.com/sosy-lab/sv-benchmarks/blob/master/c/loop-invgen/fragtest_simple_true-unreach-call_true-termination.c

Using Loop Abstraction to Detect Static Analysis False Alarms

Automotive battery controller module details

Metrics	Values
Code size	60 KLOC
Total number of loops	272
Number of assertions	186
Average number of relevant loops per assertion	50
Average number of loops with unknown bound per assertion	25
Average number of loops with bound > 100 per assertion	5
Average number of loops abstracted per assertion	29

Model checker verification results

#assertions	CBMC	LLBMC	ESBMC	SATABS	IMPARA	BLAST	LA+CBMC
186	0	1	0	40	1	5	131

- Alarms generated by TCS Embedded Code Analyzer (TCS ECA)
- Assertions for array index out of bounds checks
- Model checker time out value = 10 minutes
- Model checkers using MiniSat in the backend

Using LABMC to Detect Static Analysis False Alarms

- Alarms = array bounds + overflow/underflow + division by zero checks
- Machine configuration: 8GB RAM, 64 bit for all applications; A7 : 4 GB RAM, 64 bit, 4 cores

Embedded Application	KLOC	Static analysis alarms	Alarms after LABMC	% precision improvement	Avg. elimination time per alarm (mins.)	Total execution time
A1	8	94	29	69.15	0.15	13 min.
A2	4.6	196	92	53.06	0.30	59 min.
A3	34	346	251	27.46	0.29	1 hour 40 min.
A4	60	189	62	67.20	0.37	1 hour 9 min.
A5	18.3	226	66	70.80	0.21	47 min.
A6	184	422	145	65.64	1.41	9 hours 55 min
A7	171.4	309	144	53.40	1.87	9 hours 37 min.

% Alarms eliminated = ~58%
Avg. elimination time = 0.8 minutes / alarm

VeriAbs : Verification by Abstraction

- VeriAbs implements LABMC
- SV-COMP 2018
  - International Competition on Software Verification
  - Participation by state-of-the-art verification tools
    - Like CBMC, CPA*, Ultimate*, ESBMC*

https://sv-comp.sosy-lab.org/2018/

- Scoring scheme
  - Correct true : 2 points, correct false : 1 point.
  - Incorrect true : -32, incorrect false : -16.

https://sv-comp.sosy-lab.org/2018/results/results-verified/

VeriAbs participation category	# programs in category	Max. score	Score	VeriAbs competition position
Reach-safety	2941	4775	2760	**SECOND**
Loops (sub-category)	163	274	216	**FIRST**

Numerical Loop Abstraction – Limitation

Original code	Loop accelerated code

```
void main()
{
 int j, i, arr[100000];
 j=0;

 for(i=0; i<100000-1; i++) {
     j++;
     arr[j]=0;
 }

 for(i=0; i<100000-1; i++) {
     j++;
     assert(arr[j]==0);
 }
}
```

```
void main()
{
 int j, i, arr[100000];
 j=0; i=0;
 …
     k = *;
     j = 0 + k*1;
     …
     arr[j]= 0;
 …
     k = *;
     j = 0 + k*1;
     …
     assert(*==0);
 }
}
```

Abstract Loop 1

Abstract Loop 2

Assertion fails

Problem

- Example on which output abstraction works but naïve invariant fails