Synthesizing invariants by solving solvable loops

Steven de Oliveira[1], Saddek Bensalem[2], Virgile Prevosto[1]

1 : CEA, List 2 : Université Grenoble Alpes

1 Results

	PILAT Input		PILAT results			Abs. Int. [2]
Program	Var	Degree	# invariants	Candidate generation (in ms)	Optimization (in s)	Proof (in s)
Deterministic						
Example 1	2	2	1	3	–	1.6
Dampened oscillator	2	2	1	7	–	0.036
Harmonic oscillator	2	2	1	4	–	0.035
Sympletic oscillator	2	2	1	2	–	0.008
[1] filter	2	1	1	3.5	–	0.0017
Non deterministic						
Simple linear filter	2	2	1	1.5	1.3	6.5
Example 3	2	2	1	3	1.7	4.3
Linear filter	4	2	1	1.9	1	
Lead-lag controller	2	1	2	2	2.5	6
Gaussian regulator	3	2	1	7	2.5	–
Controller	4	2	5	66	14	–
Low-pass filter	5	2	2	60	7	–

Performance results with our implementation. Tests have been performed on a Dell Precision M4800 with 16GB RAM and 8 cores. The first part of the benchmark are non deterministic loops. The second part represents deterministic loops (no optimization necesary). Tests with abstract interpretation have been performed with the fixpoint solver described in [2] by attempting to prove goals implied by the invariants our tool synthesizes when possible.

References

1. A. Adjé, S. Gaubert, and E. Goubault. Coupling policy iteration with semi-definite relaxation to compute accurate numerical invariants in static analysis. *Logical Methods in Computer Science*, 8(1), 2012.
2. A. Miné, J. Breck, and T. Reps. An algorithm inspired by constraint solvers to infer inductive invariants in numeric programs. *European Symposium on Programming Languages and Systems*, 2015.

2 Detailed results

2.1 Example 1

```
int main(){
  float x,y;
  while(x < 4){
    x = 0.68 * (x-y);
    y = 2*0.68*y + x;
  }

  return 1;

}
```

Invariant generated : $-cst \leqslant 1. * (x * x) + 1. * (y * y) \leqslant cst$

2.2 Dampened oscillator

```
 int main(){
  float x0,x1,tx0,tx1;
  while(1){
    tx0 = x0 + 0.01 * x1;
    tx1 = -0.1 * x0 + 0.99*x1;

    x0 = tx0;
    x1 = tx1;
  }

  return 1;

}
```

Invariant generated :
$-cst \leqslant (1. * (x1 * x0) + 10. * (x0 * x0)) + 1. * (x1 * x1) \leqslant cst$

2.3 Harmonic oscillator

```
int main(){
  float x0,x1,tx0,tx1;
  while(1){
    tx0 = 0.95 * x0 + 0.09975 * x1;
    tx1 = -0.1 * x0 + 0.95*x1;

    x0 = tx0;
    x1 = tx1;
  }
```

Invariant generated :
$-cst \leqslant 1.00250626566 * (x0 * x0) + 1. * (x1 * x1) \leqslant cst$

2.4 Symplectic SEU Oscillator

```
int main(){
  float v,x;
  while (v >= 1/2) {
    x = (1 - 0.05) * x + (0.1 - 0.00025 ) * v;
    v = -0.1 *x+(1-0.05 )* v ;
  }
}
```

Invariant generated : $-cst \leqslant 0.105 * (x * v) + 1.05 * (v * v) + 1. * (x * x) \leqslant cst$

2.5 [1] filter

```
float float_interval(float, float);

int main(){
  float x,y;
  while(1){
    x = (0.75) * x - (0.125)* y;
    y = x;
  }
  return 0;

}
```

Invariants generated : $-cst \leqslant -6. * x + 1. * y \leqslant cst$

2.6 Simple filter

```
int main(){
  float x,y;
```

```
  float k;
  while(x < 4){
    k=float_interval(-0.1,0.1); /* 0 */
    x = 0.68 * (x-y) + k;
    y = 2*0.68*y + x;

  }

  return 1;

}
```

Invariant generated : $|1. * (x * x) + 1. * (y * y)| \leqslant 14.892578125$

2.7 Example 3

```
int main(){
  float s0 = 0,s1 = 0,r;
  while(1){
    r = 1.5*s0 - 0.7*s1 + float_interval(-0.1,0.1);
    s1 = s0;
    s0 = r;
  }
  return 0;
}
```

Invariant generated : $|(-2.14285714286 * (s1 * s0) + 1.42857142857 * (s0 * s0)) + 1. * (s1 * s1)| \leqslant 0.830078125;$

2.8 Linear filter

```
int main(){
  float s0 = 0,s1 = 0,r;
  int N = 50;
  while(N > 0){
    r = 1.5*s0 - 0.7*s1 + float_interval(-1.6,1.6);
    s1 = s0;
    s0 = r;
    N--;
  }

  return 0;

}
```

Invariant generated : $|(-2.14285714286 * (s0 * s1) + 1.42857142857 * (s1 * s1)) + 1. * (s0 * s0)| \leqslant 137.451171875$

2.9 Lead lag controller

```
 int main(){
  float x0p,x1p,x0,x1;
  while(1){/*
    x1 = 0.01*x0 + x1;
    x0 = 0.499*x0 - 0.05*x1 + 0.0005*x0 + float_interval(-1,1);
          */
    x0p = x0; x1p = x1;

    x0 = 0.499*x0p - 0.05*x1p + float_interval(-1,1);
    x1 = 0.010*x0p + x1p;


  }

  return 1;

}
```

Invariants generated :

$0.02 * x0 + 1. * x1 <= 70.1172$
$10. * x0 + 1. * x1 <= 20.1172$

2.10 Gaussian regulator

```
int main(){
  float x0,x1,x2,tx0,tx1,tx2,in;
  while(1){
    in = float_interval(-1,1);
    tx0 = 0.9379 * x0 - 0.0381 * x1 - 0.0414 * x2 + 0.0237 * in;
    tx1 = -0.0404 * x0 + 0.968 * x1 - 0.0179 * x2 + 0.0143 * in;
    tx2 = 0.0142 * x0 - 0.0197 * x1 + 0.9823 * x2 + 0.0077 * in;
    x0 = tx0;
    x1 = tx1;
    x2 = tx2;

  }

  return 0;

}
```

Invariants generated :
$|(1.2187798948 * x0 + 1.16137161588 * x1) + 1. * x2| \leqslant 1.171875$
$| - 2.45498840354 * x1 * x0 + -0.788574527791 * x2 * x0 + 0.868152061813 * x0 * x0 + 1.12295787049 * x2 * x1 + 1.73559323995 * x1 * x1 + 1. * x2 * x2| \leqslant 10.7422$

2.11 Controller

```
int main(){
  float x0,x1,x2,x3,tx0,tx1,tx2,tx3,in0,in1;
  while(1){
    in0 = float_interval(-1,1);
    in1 = float_interval(-1,1);
    tx0 = 0.6227 * x0 + 0.3871 * x1 - 0.113 * x2 + 0.0102 * x3 + 0.3064 * in0 + 0.1826 * in1;
    tx1 =  -0.3407 * x0 + 0.9103 * x1 - 0.3388 * x2 + 0.0649 * x3 - 0.0054 * in0 + 0.6731 * in1;
    tx2 =  0.0918 * x0 - 0.0265 * x1 - 0.7319 * x2 + 0.2669 * x3 + 0.0494 * in0 + 1.6138 * in1;
    tx3 = 0.2643 * x0 - 0.1298 * x1 - 0.9903 * x2 + 0.3331 * x3 - 0.0531 * in0 + 0.4012 * in1;

    x0 = tx0;
    x1 = tx1;
    x2 = tx2;
    x3 = tx3;

  }

  return 1;

}
```

$| - 0.00203592622443 * x1 * x0 + 0.0881698609486 * x2 * x0 + -0.0355121282196 * x3 * x0 + 0.000315277812672 * x0 * x0 + -0.284681200032 * x2 * x1 + 0.114660896235 * x3 * x1 + 0.00328678028134 * x1 * x1 + -4.96561965553 * x3 * x2 + 6.16434464085 * x2 * x2 + 1. * x3 * x3| \leqslant 20.166015625;$
$| - 0.00861093083991 * x1 * x0 + 0.298520354653 * x2 * x0 + -0.126853461757 * x3 * x0 + 0.00193714038684 * x0 * x0 + -0.418765740617 * x2 * x1 + 0.190035991969 * x3 * x1 + 0.00760806829668 * x1 * x1 + -4.0401517826 * x3 * x2 + 3.86658391065 * x2 * x2 + 1. * x3 * x3| \leqslant 18.212890625;$
$| - 0.0289556589751 * x1 * x0 + 0.339803909038 * x2 * x0 + -0.218194795294 * x3 * x0 + 0.0119022421734 * x0 * x0 + -0.413335822158 * x2 * x1 + 0.265411087703 * x3 * x1 + 0.017610761369 * x1 * x1 + -3.11468390967 * x3 * x2 + 2.42531396428 * x2 * x2 + 1. * x3 * x3| \leqslant 5.17578125;$
$| - 0.0177560641098 * x0 + 0.0573304481174 * x1 + -2.48280982777 * x2 + 1. * x3| \leqslant 5005.46875;$
$| - 0.109097397647 * x0 + 0.132705543852 * x1 + -1.55734195483 * x2 + 1. * x3| \leqslant 582.03125;$

2.12 Low pass filter

```
int main(){
```

```
  float x0,x1,x2,x3,x4,tx0,tx1,tx2,tx3,tx4,in0;

  while(1){
    in0 = float_interval(-1,1);
    x0 = 0.4250 * tx0 + 0.8131 * in0;
    x1 = 0.3167 * tx0 + 0.1016 * tx1 - 0.4444* tx2 + 0.1807 * in0;
    x2 = 0.1278 * tx0 + 0.4444 * tx1 +0.8207 * tx2 + 0.0729 * in0;
    x3 = 0.0365 * tx0 + 0.1270 * tx1 + 0.5202 * tx2 + 0.4163 * tx3 - 0.5714 * tx4 + 0.0208 * in0;
    x4 = 0.0147 * tx0 + 0.0512 * tx1 + 0.2099 * tx2 + 0.57104 * tx3 + 0.7694 * tx4 + 0.0084 * in0;

    tx0 = x0;
    tx1 = x1;
    tx2 = x2;
    tx3 = x3;
    tx4 = x4;
  }

}
```

Invariants generated : $|1.00164325052 * tx1 * tx0 + 0.000140144389337 * tx2 * tx0 + -1.6191550573 * tx3 * tx0 + -1.00126210564 * tx4 * tx0 + 0.72511353991 * tx0 * tx0 + 2.62046247703 * tx2 * tx1 + -0.618060865742 * tx3 * tx1 + -2.00110233268 * tx4 * tx1 + 1.00110265182 * tx1 * tx1 + 1.00020053093 * tx3 * tx2 + -2.61878326237 * tx4 * tx2 + 2.61995131748 * tx2 * tx2 + 0.617955897795 * tx4 * tx3 + 0.999369968498 * tx3 * tx3 + 1. * tx4 * tx4| <= 17.724609375;$
$|1.00123175519 * tx1 * tx0 + -0.999123490687 * tx2 * tx0 + 2.61966963703 * tx0 * tx0 + 1.61813681368 * tx2 * tx1 + 1. * tx1 * tx1 + 1. * tx2 * tx2| <= 8.642578125;$