Computing Invariants with Transformers: Experimental Scalability and Accuracy

Vivien Maisonneuve

Olivier Hermant François Irigoin

MINES ParisTech

The Fifth International Workshop on Numerical and Symbolic Abstract Domains
(NSAD 2014)

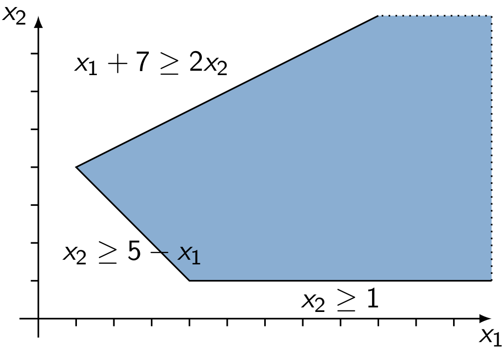
Munich, September 10, 2014

Introduction

Program analysis ⇒ computation of invariants (e.g. model checking).

Abstract domains needed to approximate complex program behaviors.

Here: affine invariants = systems of linear (in)equalities.



Classic Linear Relation Analysis (LRA)

Example by Halbwachs & Henry [SAS'12]

```
void foo() {
    int n = 0;
    while (true)
        if (rand())
            if (n < 60) n++;
            else n = 0;


}
```

Classic Linear Relation Analysis (LRA)

Example by Halbwachs & Henry [SAS'12]

```
void foo() {
    // $P_0 : \Omega$
    int n = 0;

    while (true)

        if (rand())

            if (n < 60) n++;

            else n = 0;



}
```

- Propagation

Classic Linear Relation Analysis (LRA)

Example by Halbwachs & Henry [SAS'12]

```
void foo() {
    // $P_0 : \Omega$
    int n = 0;
    // $P_1 : n = 0$
    while (true)

        if (rand())

            if (n < 60) n++;

            else n = 0;



}
```

- Propagation

Classic Linear Relation Analysis (LRA)

Example by Halbwachs & Henry [SAS'12]

```
void foo() {
    // $P_0 : \Omega$
    int n = 0;
    // $P_1 : n = 0$
    while (true)
        // $P_2 : n = 0$
        if (rand())

            if (n < 60) n++;

            else n = 0;



}
```

- Propagation

Classic Linear Relation Analysis (LRA)

Example by Halbwachs & Henry [SAS'12]

```
void foo() {
    // $P_0 : \Omega$
    int n = 0;
    // $P_1 : n = 0$
    while (true)
        // $P_2 : n = 0$
        if (rand())
            // $P_3 : n = 0$
            if (n < 60) n++;

            else n = 0;



}
```

- Propagation

Classic Linear Relation Analysis (LRA)

Example by Halbwachs & Henry [SAS'12]

```
void foo() {
    // $P_0 : \Omega$
    int n = 0;
    // $P_1 : n = 0$
    while (true)
        // $P_2 : n = 0$
        if (rand())
            // $P_3 : n = 0$
            if (n < 60) n++;
                // $P_4 : n = 1$
            else n = 0;



}
```

- Propagation in each branch

Classic Linear Relation Analysis (LRA)

Example by Halbwachs & Henry [SAS'12]

```
void foo() {
    // $P_0 : \Omega$
    int n = 0;
    // $P_1 : n = 0$
    while (true)
        // $P_2 : n = 0$
        if (rand())
            // $P_3 : n = 0$
            if (n < 60) n++;
                // $P_4 : n = 1$
            else n = 0;
                // $P_5 : \emptyset$


}
```

- Propagation in each branch

Classic Linear Relation Analysis (LRA)

Example by Halbwachs & Henry [SAS'12]

```
void foo() {
    // $P_0 : \Omega$
    int n = 0;
    // $P_1 : n = 0$
    while (true)
        // $P_2 : n = 0$
        if (rand())
            // $P_3 : n = 0$
            if (n < 60) n++;
                // $P_4 : n = 1$
            else n = 0;
                // $P_5 : \emptyset$
            // $P_6 : ?$

}
```

- Propagation in each branch
- Branch output P_6:
  either P_4 or P_5

Classic Linear Relation Analysis (LRA)

Example by Halbwachs & Henry [SAS'12]

```
void foo() {
    // $P_0 : \Omega$
    int n = 0;
    // $P_1 : n = 0$
    while (true)
        // $P_2 : n = 0$
        if (rand())
            // $P_3 : n = 0$
            if (n < 60) n++;
                // $P_4 : n = 1$
            else n = 0;
                // $P_5 : \emptyset$
            // $P_6 : n = 1$

}
```

- Propagation in each branch
- Branch output P_6:
  either P_4 or P_5
  $P_6 = P_4 \sqcup P_5 : n = 1$

Classic Linear Relation Analysis (LRA)

Example by Halbwachs & Henry [SAS'12]

```
void foo() {
    // $P_0 : \Omega$
    int n = 0;
    // $P_1 : n = 0$
    while (true)
        // $P_2 : n = 0$
        if (rand())
            // $P_3 : n = 0$
            if (n < 60) n++;
                // $P_4 : n = 1$
            else n = 0;
                // $P_5 : \emptyset$
            // $P_6 : n = 1$
        // $P_7 : 0 \leq n \leq 1$
}
```

- Propagation in each branch
- Branch output P_6:
  either P_4 or P_5
  $P_6 = P_4 \sqcup P_5 : n = 1$
- Branch output P_7:
  $P_7 = P_2 \sqcup P_6 : 0 \leq n \leq 1$

Classic Linear Relation Analysis (LRA)

Example by Halbwachs & Henry [SAS'12]

```
void foo() {
    // $P_0 : \Omega$
    int n = 0;
    // $P_1 : n = 0$
    while (true)
        // $P_2 : n = 0$
        if (rand())
            // $P_3 : n = 0$
            if (n < 60) n++;
                // $P_4 : n = 1$
            else n = 0;
                // $P_5 : \emptyset$
            // $P_6 : n = 1$
        // $P_7 : 0 \leq n \leq 1$
}
```

- Propagation in each branch
- Branch output P_6:
  either P_4 or P_5
  $P_6 = P_4 \sqcup P_5 : n = 1$
- Branch output P_7:
  $P_7 = P_2 \sqcup P_6 : 0 \leq n \leq 1$
- Loop invariant:
  P_2 entering the loop
  P_7 after one iteration

Classic Linear Relation Analysis (LRA)

Example by Halbwachs & Henry [SAS'12]

```
void foo() {
    // $P_0 : \Omega$
    int n = 0;
    // $P_1 : n = 0$
    while (true)
        // $P_2 : n = 0$
        if (rand())
            // $P_3 : n = 0$
            if (n < 60) n++;
                // $P_4 : n = 1$
            else n = 0;
                // $P_5 : \emptyset$
            // $P_6 : n = 1$
        // $P_7 : 0 \leq n \leq 1$
}
```

- Propagation in each branch
- Branch output P_6:
  either P_4 or P_5
  $P_6 = P_4 \sqcup P_5 : n = 1$
- Branch output P_7:
  $P_7 = P_2 \sqcup P_6 : 0 \leq n \leq 1$
- Loop invariant:
  P_2 entering the loop
  P_7 after one iteration
  Widening:

  $P^* = P_2 \nabla P_7 : 0 \leq n$

PIPS Approach

PIPS: “A source-to-source compilation framework for analyzing and transforming C and Fortran programs”

1. Abstraction of each program instruction, block, function by a transformer = polyhedral approximation of the transfer function
2. Invariant propagation using transformers

PIPS Approach

PIPS: "A source-to-source compilation framework for analyzing and transforming C and Fortran programs"

1. Abstraction of each program instruction, block, function by a transformer = polyhedral approximation of the transfer function
2. Invariant propagation using transformers

Pros:

- Interprocedural analysis
- Nested loops

⇒ Supports large applications

Cons:

- Double abstraction ⇒ less accurate
- Worst case complexity: $2^{2^{|V|}}$ vs. $2^{|V|}$

PIPS: Transformers

```
void foo() {

    int n = 0;

    while (true)

        if (rand())

            if (n < 60) n++;
            else n = 0;
}
```

PIPS: Transformers

```
void foo() {

    int n = 0; // T_0 : n' = 0

    while (true)

        if (rand())

            if (n < 60) n++;
            else n = 0;
}
```

- Elementary instructions

PIPS: Transformers

```
void foo() {

    int n = 0; // $T_0 : n' = 0$

    while (true)

        if (rand())

            if (n < 60) n++; // $T_4 : n' \leq 60, n' = n + 1$
            else n = 0;
}
```

- Elementary instructions

PIPS: Transformers

```
void foo() {

    int n = 0; // $T_0 : n' = 0$

    while (true)

        if (rand())

            if (n < 60) n++; // $T_4 : n' \leq 60, n' = n + 1$
            else n = 0; // $T_5 : n > 60, n' = 0$
}
```

- Elementary instructions

PIPS: Transformers

```
void foo() {

  int n = 0; // $T_0 : n' = 0$

  while (true)

    if (rand())
      // $T_3 = T_4 \sqcup T_5 : n' \leq 60, n' \leq n + 1$
      if (n < 60) n++; // $T_4 : n' \leq 60, n' = n + 1$
      else n = 0; // $T_5 : n > 60, n' = 0$
}
```

- Elementary instructions
- Compound statements

PIPS: Transformers

```
void foo() {

   int n = 0; // $T_0 : n' = 0$

   while (true)

      if (rand()) // $T_2 = T_3 \sqcup \mathrm{Id} : n' \leq n + 1$
         // $T_3 = T_4 \sqcup T_5 : n' \leq 60, n' \leq n + 1$
         if (n < 60) n++; // $T_4 : n' \leq 60, n' = n + 1$
         else n = 0; // $T_5 : n > 60, n' = 0$
}
```

- Elementary instructions
- Compound statements

PIPS: Transformers

```
void foo() {

    int n = 0; // $T_0 : n' = 0$

    while (true) // $T_1 = T_2^* : n' \leq n + 1$

        if (rand()) // $T_2 = T_3 \sqcup \mathrm{Id} : n' \leq n + 1$
            // $T_3 = T_4 \sqcup T_5 : n' \leq 60, n' \leq n + 1$
            if (n < 60) n++; // $T_4 : n' \leq 60, n' = n + 1$
            else n = 0; // $T_5 : n > 60, n' = 0$
}
```

- Elementary instructions
- Compound statements
- Transitive closure
  [Ancourt *et al.*, NSAD'10]

PIPS: Transformers & Invariants

```
void foo() {
    // $P_0 : \Omega$
    int n = 0; // $T_0 : n' = 0$

    while (true) // $T_1 = T_2^* : n' \leq n + 1$

        if (rand()) // $T_2 = T_3 \sqcup \mathrm{Id} : n' \leq n + 1$
            // $T_3 = T_4 \sqcup T_5 : n' \leq 60, n' \leq n + 1$
            if (n < 60) n++; // $T_4 : n' \leq 60, n' = n + 1$
            else n = 0; // $T_5 : n > 60, n' = 0$
}
```

- Elementary instructions
- Compound statements
- Transitive closure
  [Ancourt *et al.*, NSAD'10]

- Invariant propagation using transformers

PIPS: Transformers & Invariants

```
void foo() {
    // $P_0 : \Omega$
    int n = 0; // $T_0 : n' = 0$
    // $P_1 : n = 0$
    while (true) // $T_1 = T_2^* : n' \leq n + 1$

        if (rand()) // $T_2 = T_3 \sqcup \mathrm{Id} : n' \leq n + 1$
            // $T_3 = T_4 \sqcup T_5 : n' \leq 60, n' \leq n + 1$
            if (n < 60) n++; // $T_4 : n' \leq 60, n' = n + 1$
            else n = 0; // $T_5 : n > 60, n' = 0$
}
```

- Elementary instructions
- Compound statements
- Transitive closure [Ancourt *et al.*, NSAD'10]

- Invariant propagation using transformers

PIPS: Transformers & Invariants

```
void foo() {
    // $P_0 : \Omega$
    int n = 0; // $T_0 : n' = 0$
    // $P_1 : n = 0$
    while (true) // $T_1 = T_2^* : n' \leq n + 1$
        // $P_6 : 0 \leq n$
        if (rand()) // $T_2 = T_3 \sqcup \mathrm{Id} : n' \leq n + 1$
            // $T_3 = T_4 \sqcup T_5 : n' \leq 60, n' \leq n + 1$
            if (n < 60) n++; // $T_4 : n' \leq 60, n' = n + 1$
            else n = 0; // $T_5 : n > 60, n' = 0$
}
```

- Elementary instructions
- Compound statements
- Transitive closure [Ancourt *et al.*, NSAD'10]
- Invariant propagation using transformers

Sources of Approximations

For both classic LRA / transformers

- Loops (widening / transitive closure)
- Branches (convex union)

Cumulative impact (multiple control paths nested within loops)

Contents

Tools Used

Comparison of

- PIPS:
  Transformer-based

with

- ASPIC:
  Classic LRA + accelerations
- ISL:
  Presburger-equivalent library with powerful transitive closure heuristics
- PAGAI:
  Classic LRA + decision procedures (SMT-solving)

Tools Used

Comparison of

- PIPS:
  Transformer-based
  C code

with

- ASPIC:
  Classic LRA + accelerations
  Finite state machine
- ISL:
  Presburger-equivalent library with powerful transitive closure heuristics
  Transition relation
- PAGAI:
  Classic LRA + decision procedures (SMT-solving)
  C code (through LLVM IR)

Impact of Cycle Nesting on Convergence Time

Analysis of loop nests:

```
for (i1 = 0; i1 < b1; i1++)
    for (i2 = 0; i2 < b2; i2++)
        ...
```

Depth	1	2	3	4	5	6	7	8	9
ASPIC	0.037	0.043	0.040	0.053	0.047	0.063	0.067	0.087	0.100
ISL	0.000	0.010	0.037	0.083	0.370	0.853	1.197	7.927	5.713
PAGAI	0.067	0.187	0.420	0.797	1.373	2.260	3.620	5.780	9.643
PIPS	0.004	0.009	0.015	0.021	0.030	0.039	0.053	0.071	0.090

Interprocedural Analysis vs. Inlining

```
void mm(int l, int n, int m,
        float A[l][m], float B[l][n], float C[n][m]) {
    // naive matrix multiplication
    // A = B * C
    ...
}

void mp(int n, int p,
        float A[n][n], float B[n][n]) {
    // matrix exponentiation
    // A = B^p
    ...
    mn(...);
    ...
}
```

Interprocedural Analysis vs. Inlining

```
int main(void) {
    ...
    mp(...);
    mp(...);
    ...
}
```

	Inlining	2	3	4	5	6
ASPIC	Yes	0.043	0.061	0.087	0.108	0.149
ISL	Yes	261.810	274.580	370.960	413.300	456.360
PAGAI	Yes	1.417	5.680	14.677	30.007	53.247
	No	0.980	1.383	2.030	2.990	4.467
PIPS	Yes	0.043	0.063	0.084	0.108	0.127
	No	0.048	0.049	0.048	0.050	0.051

Accuracy Results with ALICe

ALICe benchmark: assess the robustness and accuracy of invariant generating tools

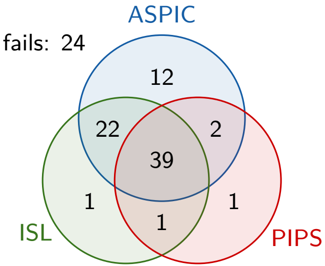
Supports ASPIC, ISL and PIPS; provided with 102 small test cases

- ASPIC: 75 test cases correctly analyzed
- ISL: 63
- PIPS: 43

[Maisonneuve *et al.*, WING'14]

Accuracy Results with ALICe

No tool is strictly better



No trend for invariant accuracy

↯	ASPIC	ISL	PIPS
ASPIC	–	21	23
ISL	49	–	54
PIPS	33	23	–

- ISL good with concurrent loops, unlike PIPS
- ISL slow on large control structures
- ASPIC in difficulty with complex formulæ (no acceleration)

Evaluation & Shortcomings

Evaluation of PIPS approach

- Effective for large programs with function calls, nested loops
- Lacks accuracy for small transition systems challenging invariant generation

Sources of inaccuracy

- Multiple control paths nested within loops:
  convex hulls + transitive closures
- Arithmetic overflows

⇒ Improvements in transformer computation

Control-Path Transformers

```
while (true)
{

    if ... // T₁
    else ... // T₂
}
```

Control-Path Transformers

```
while (true)
{ // $T = T_1 \sqcup T_2$

    if ... // $T_1$
    else ... // $T_2$
}
```

Control-Path Transformers

```
while (true) // $T^* = (T_1 \sqcup T_2)^*$
{ // $T = T_1 \sqcup T_2$

    if ... // $T_1$
    else ... // $T_2$
}
```

Control-Path Transformers

```
// $P$
while (true) // $T^* = (T_1 \sqcup T_2)^*$
{ // $T = T_1 \sqcup T_2$
    // $P' = T^*(P)$
    if ... // $T_1$
    else ... // $T_2$
}
```

Control-Path Transformers

```
// $P$
while (true) // $T^* = (T_1 \sqcup T_2)^*$
{ // $T = T_1 \sqcup T_2$
    // $P' = T^*(P)$
    if ... // $T_1$
    else ... // $T_2$
}
```

Use alternate formula:

$$P' = P \sqcup {T_1}^+(P) \sqcup {T_2}^+(P) \sqcup (T_1 \circ T_2)(P) \sqcup (T_2 \circ T_1)(P) \sqcup \\ ({T_1}^+ \circ T_2 \circ T^*)(P) \sqcup ({T_2}^+ \circ T_1 \circ T^*)(P)$$

Convex hulls are postponed, performed at the invariants instead of transformers ⇒ more information is preserved

Control-Path Transformers

```
void foo() {
    // $P_0 : \Omega$
    int n = 0; // $T_0 : n' = 0$
    // $P_1 : n = 0$
    while (true) // $T_1 = T_2^* : n' \leq n + 1$
        // $P_6 : 0 \leq n$
        if (rand()) // $T_2 = T_3 \sqcup \mathrm{Id} : n' \leq n + 1$
            // $T_3 = T_4 \sqcup T_5 : n' \leq 60, n' \leq n + 1$
            if (n < 60) n++; // $T_4 : n' \leq 60, n' = n + 1$
            else n = 0; // $T_5 : n > 60, n' = 0$
}
```

With convex hulls in the precondition space: $P_6' : 0 \leq n \leq 60$

Iterative Analysis

- At iteration 1, compute transformers and invariants as usual

Example by Dillig *et al.* [OOPSLA'13]

```
void bar(float x) {
    int i, j = 1, a = 0, b = 0;
    i = 0;
    while (rand()) {




        a++; b += j-i; i += 2;
        if (i % 2 == 0) j += 2;
        else j++;
    }
}
```

Iterative Analysis

- At iteration 1, compute transformers and invariants as usual

Example by Dillig *et al.* [OOPSLA'13]

```
void bar(float x) {
    int i, j = 1, a = 0, b = 0;
    i = 0;
    while (rand()) {
        // P1 : 2a = i, j <= 2a + 1, a + 1 <= j


        a++; b += j-i; i += 2;
        if (i % 2 == 0) j += 2;
        else j++;
    }
}
```

Iterative Analysis

- At iteration 1, compute transformers and invariants as usual
- At iteration $n+1$, sharpen transformers with invariants found at iteration n, then recompute invariants

Example by Dillig *et al.* [OOPSLA'13]

```
void bar(float x) {
    int i, j = 1, a = 0, b = 0;
    i = 0;
    while (rand()) {
        // $P_1 : 2a = i, j \leq 2a + 1, a + 1 \leq j$



        a++; b += j-i; i += 2;
        if (i % 2 == 0) j += 2;
        else j++;
    }
}
```

Iterative Analysis

- At iteration 1, compute transformers and invariants as usual
- At iteration $n+1$, sharpen transformers with invariants found at iteration n, then recompute invariants

Example by Dillig *et al.* [OOPSLA'13]

```
void bar(float x) {
    int i, j = 1, a = 0, b = 0;
    i = 0;
    while (rand()) {
        // $P_1 : 2a = i, j \leq 2a + 1, a + 1 \leq j$
        // $P_2 : 2a = i, 2a = j - 1, 0 \leq a, b \leq a$

        a++; b += j-i; i += 2;
        if (i % 2 == 0) j += 2;
        else j++;
    }
}
```

Iterative Analysis

- At iteration 1, compute transformers and invariants as usual
- At iteration $n+1$, sharpen transformers with invariants found at iteration n, then recompute invariants

Example by Dillig *et al.* [OOPSLA'13]

```
void bar(float x) {
    int i, j = 1, a = 0, b = 0;
    i = 0;
    while (rand()) {
        // $P_1 : 2a = i, j \leq 2a + 1, a + 1 \leq j$
        // $P_2 : 2a = i, 2a = j - 1, 0 \leq a, b \leq a$
        // $P_3 : a = b, 2a = i, 2a = j - 1, 0 \leq a$
        a++; b += j-i; i += 2;
        if (i % 2 == 0) j += 2;
        else j++;
    }
}
```

Arbitrary-Precision Numbers

- Polyhedra with huge coefficients in intermediate computations
- Arithmetic overflow $\Rightarrow$ constraint dropped
- Less accurate invariant

GMP support added to PIPS

Experimental Results

Out of 102 test cases in ALICe:

Options	None	CP	IA	CP-IA	CP-IA-MP	ASPIC	ISL
Successes	43	69	45	72	73	75	63
Time (s.)	6.1	7.8	18.5	19.6	151.4	10.9	35.5

(More measurements in the paper)

Failures

- C code generated by ALICe from CFG may blow up exponentially
  Some cases work with native C encoding
- Cases require non-convex invariant or transformer
- Information about behavior of inner loops may be lost to keep a small number of control paths
- Too many control paths, cannot unroll
- ...

Conclusion

- Transformer approach is time-efficient for large pieces of code, with many functions & nested loops
- But lacks of accuracy for small cases
- Improvements in loop invariant generation:
  - control-path transformers
  - iterative analysis
  - arbitrary-precision numbers
- Comparable accuracy in PIPS with ASPIC and ISL

Future Work

- Better support for C code in ALICe
- More test cases
- Improve invariant generation while avoiding exponential blowup

Computing Invariants with Transformers: Experimental Scalability and Accuracy

Vivien Maisonneuve

Olivier Hermant François Irigoin

MINES ParisTech

The Fifth International Workshop on Numerical and Symbolic Abstract Domains
(NSAD 2014)

Munich, September 10, 2014