Static Analysis by Abstract Interpretation and Decision Procedures

Julien Henry

Université Joseph Fourier

Advisors:
David Monniaux, Matthieu Moy

January 24^{th} 2013

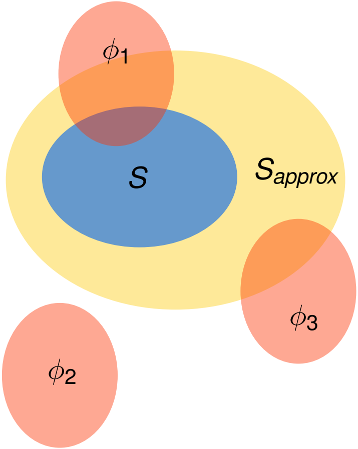
Summary

1. Introduction
2. Weakness of the Standard Approach
3. Contributions

Static Analysis

- Discover properties on programs
- Find program invariants, bugs...
- Allow compile-time optimizations
- blablabla

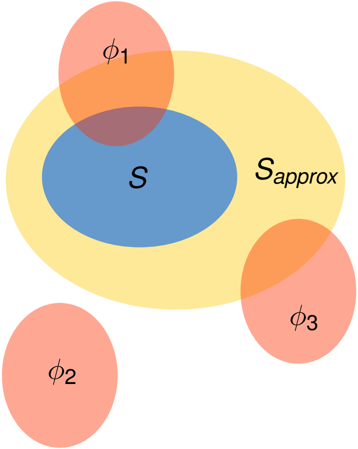
Over-approximation



- $\phi_1 \cap S \neq \emptyset$ and $\phi_1 \cap S_{approx} \neq \emptyset$
  S is unsafe w.r.t ϕ_1 and the analyser emits an alarm.

S	:	set of reachable states	(not computable)
ϕ_1, ϕ_2, ϕ_3	:	error states	(computable)
S_{approx}	:	over-approximation of S	(computable)

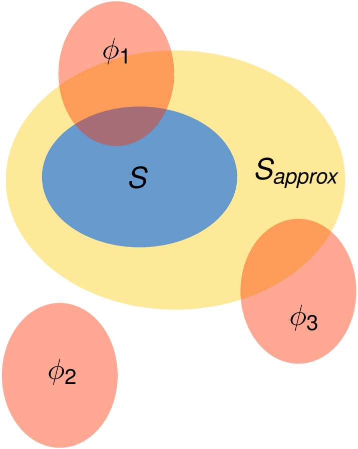
Over-approximation



- $\phi_1 \cap S \neq \emptyset$ and $\phi_1 \cap S_{approx} \neq \emptyset$
  S is unsafe w.r.t ϕ_1 and the analyser emits an alarm.
- $\phi_2 \cap S = \emptyset$ and $\phi_2 \cap S_{approx} = \emptyset$
  S is safe w.r.t ϕ_2 and the analyser proves it.

S	:	set of reachable states	(not computable)
ϕ_1, ϕ_2, ϕ_3	:	error states	(computable)
S_{approx}	:	over-approximation of S	(computable)

Over-approximation



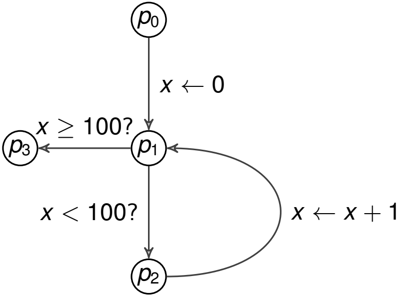
- $\phi_1 \cap S \neq \emptyset$ and $\phi_1 \cap S_{approx} \neq \emptyset$
  S is unsafe w.r.t ϕ_1 and the analyser emits an alarm.
- $\phi_2 \cap S = \emptyset$ and $\phi_2 \cap S_{approx} = \emptyset$
  S is safe w.r.t ϕ_2 and the analyser proves it.
- $\phi_3 \cap S = \emptyset$ and $\phi_3 \cap S_{approx} \neq \emptyset$
  S is safe w.r.t ϕ_3 but the analyser emits a false alarm.

S	:	set of reachable states	(not computable)
ϕ_1, ϕ_2, ϕ_3	:	error states	(computable)
S_{approx}	:	over-approximation of S	(computable)

Abstract Interpretation

- Compute an increasing sequence of the set X of reachable states of the program.
- Fixpoint computation : X grows until $F(X) \subseteq X$.

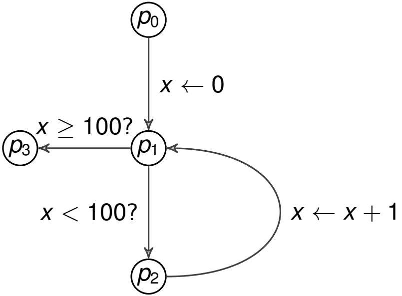
```
x = 0;
while (x < 100) {
    x++;
}
```



Abstract Interpretation

- Compute an increasing sequence of the set X of reachable states of the program.
- Fixpoint computation : X grows until $F(X) \subseteq X$.

```
x = 0;
while (x < 100) {
    x++;
}
```

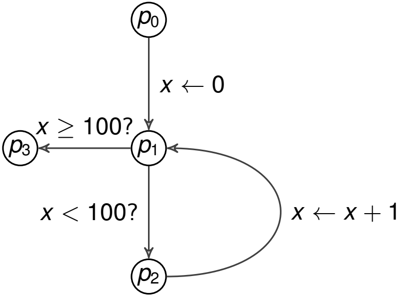


$$X_1 = \{x \mid x = 0 \vee \exists x' \in X_2, x = x' + 1\}$$
$$X_2 = \{x \mid x \in X_1 \wedge x < 100\}$$

Abstract Interpretation

- Compute an increasing sequence of the set X of reachable states of the program.
- Fixpoint computation : X grows until $F(X) \subseteq X$.

```
x = 0;
while (x < 100) {
    x++;
}
```

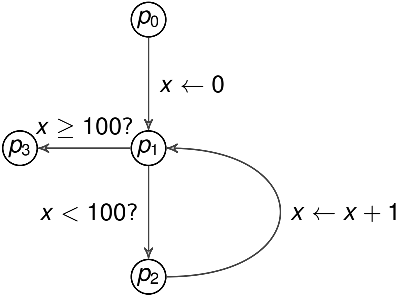


$$
\begin{aligned}
X_1 &= \{x \mid x = 0 \vee \exists x' \in X_2, x = x' + 1\} && \supseteq \{0\} \\
X_2 &= \{x \mid x \in X_1 \wedge x < 100\} && \supseteq \emptyset
\end{aligned}
$$

Abstract Interpretation

- Compute an increasing sequence of the set X of reachable states of the program.
- Fixpoint computation : X grows until $F(X) \subseteq X$.

```
x = 0;
while (x < 100) {
    x++;
}
```

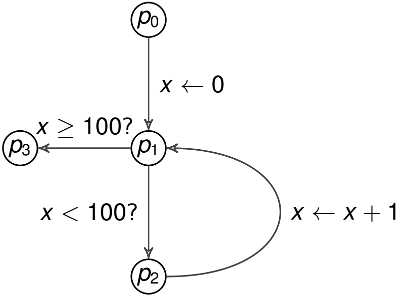


$$
\begin{aligned}
X_1 &= \{x \mid x = 0 \vee \exists x' \in X_2, x = x' + 1\} && \supseteq \{0\} \\
X_2 &= \{x \mid x \in X_1 \wedge x < 100\} && \supseteq \{0\}
\end{aligned}
$$

Abstract Interpretation

- Compute an increasing sequence of the set X of reachable states of the program.
- Fixpoint computation : X grows until $F(X) \subseteq X$.

```
x = 0;
while (x < 100) {
    x++;
}
```

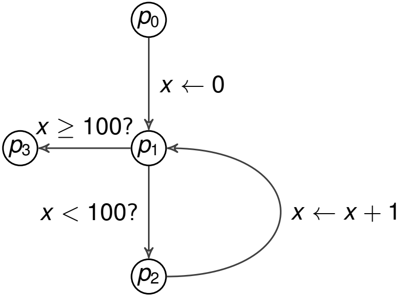


$$
\begin{aligned}
X_1 &= \{x \mid x = 0 \vee \exists x' \in X_2, x = x' + 1\} &\supseteq \{0, 1\} \\
X_2 &= \{x \mid x \in X_1 \wedge x < 100\} &\supseteq \{0\}
\end{aligned}
$$

Abstract Interpretation

- Compute an increasing sequence of the set X of reachable states of the program.
- Fixpoint computation : X grows until $F(X) \subseteq X$.

```
x = 0;
while (x < 100) {
    x++;
}
```

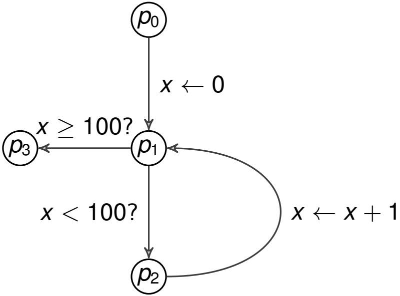


$$X_1 = \{x \mid x = 0 \vee \exists x' \in X_2, x = x' + 1\} \supseteq \{0, 1\}$$
$$X_2 = \{x \mid x \in X_1 \wedge x < 100\} \supseteq \{0, 1\}$$

Abstract Interpretation

- Compute an increasing sequence of the set X of reachable states of the program.
- Fixpoint computation : X grows until $F(X) \subseteq X$.

```
x = 0;
while (x < 100) {
    x++;
}
```

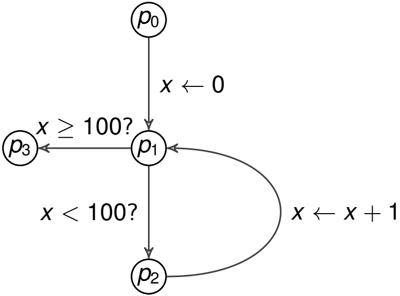


$$
\begin{aligned}
X_1 &= \{x \mid x = 0 \vee \exists x' \in X_2, x = x' + 1\} &\supseteq \{0, 1, 2, \dots\} \\
X_2 &= \{x \mid x \in X_1 \wedge x < 100\} &\supseteq \{0, 1\}
\end{aligned}
$$

Abstract Interpretation

- Compute an increasing sequence of the set X of reachable states of the program.
- Fixpoint computation : X grows until $F(X) \subseteq X$.

```
x = 0;
while (x < 100) {
    x++;
}
```

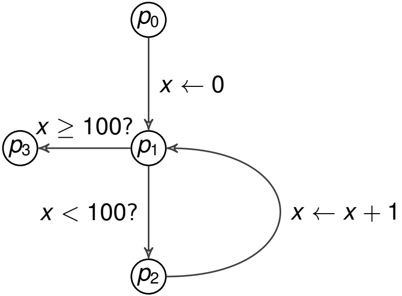


$$X_1 = \{x \mid x = 0 \vee \exists x' \in X_2, x = x' + 1\} \supseteq \{0, 1, 2, \dots\}$$
$$X_2 = \{x \mid x \in X_1 \wedge x < 100\} \supseteq \{0, 1, 2, \dots, 99\}$$

Abstract Interpretation

- Compute an increasing sequence of the set X of reachable states of the program.
- Fixpoint computation : X grows until $F(X) \subseteq X$.

```
x = 0;
while (x < 100) {
    x++;
}
```



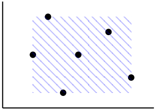
$$X_1 = \{x \mid x = 0 \vee \exists x' \in X_2, x = x' + 1\} \supseteq \{0, 1, 2, \ldots 100\}$$
$$X_2 = \{x \mid x \in X_1 \wedge x < 100\} \supseteq \{0, 1, 2, \ldots, 99\}$$

Abstract Interpretation

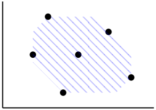
Cousot & Cousot 1977

Abstract domain to represent the set of possible states:

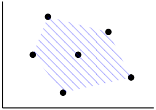
- Intervals



- Octagons



- Convex Polyhedra

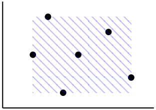


Abstract Interpretation

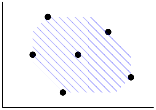
Cousot & Cousot 1977

Abstract domain to represent the set of possible states:

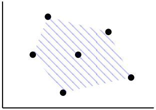
- Intervals



- Octagons



- Convex Polyhedra



$\Rightarrow$ Over-approximation of the set of states

Sources of Imprecision

- Exact set that can’t be expressed in the abstract domain

Sources of Imprecision

- Exact set that can't be expressed in the abstract domain
- Widening operator
  - Ensure termination
  - BUT: may induce huge imprecisions
  - Narrowing tends to recover some precision...

Sources of Imprecision

- Exact set that can't be expressed in the abstract domain
- Widening operator
  - Ensure termination
  - BUT: may induce huge imprecisions
  - Narrowing tends to recover some precision...
- Consider paths that are actually unfeasible
  - because of control flow merges
  - $X_1, X_2 \in A$ but $X_1 \cup X_2 \notin A \Rightarrow$ Least Upper Bound computation

Sources of Imprecision

- Exact set that can't be expressed in the abstract domain
- Widening operator
  - Ensure termination
  - BUT: may induce huge imprecisions
  - Narrowing tends to recover some precision...
- Consider paths that are actually unfeasible
  - because of control flow merges
  - $X_1, X_2 \in A$ but $X_1 \cup X_2 \notin A \Rightarrow$ Least Upper Bound computation

Ph.D topic

Improve precision of Abstract Interpretation, by combining it with Decision Procedures (SMT-solving).

Summary

Summary

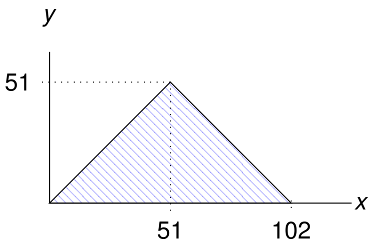
1 Introduction

2 Weakness of the Standard Approach

3 Contributions

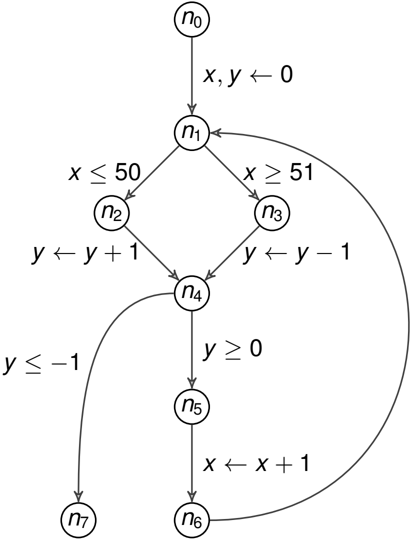
Example of Standard Abstract Interpretation

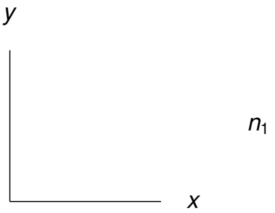
```
x = 0;
y = 0;
while (true) {
        if (x <= 50)
                y++;
        else
                y--;

        if (y < 0) break;
        x++;
}
```

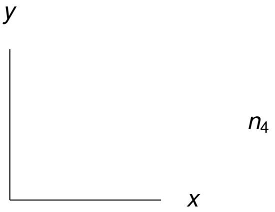


- x and y incremented during 51 iterations
- x incremented and y decremented during 51 iterations

Example of Standard Abstract Interpretation

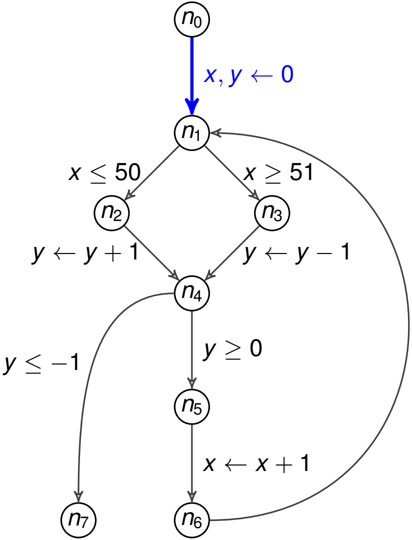


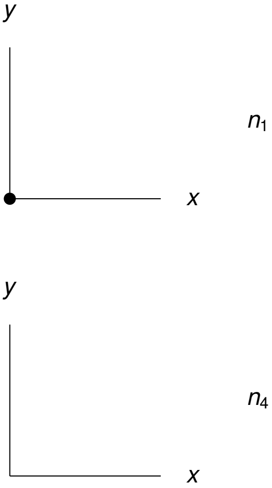




Ascending iterations

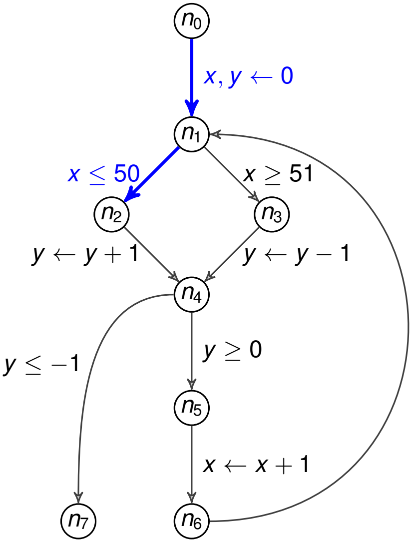
Example of Standard Abstract Interpretation

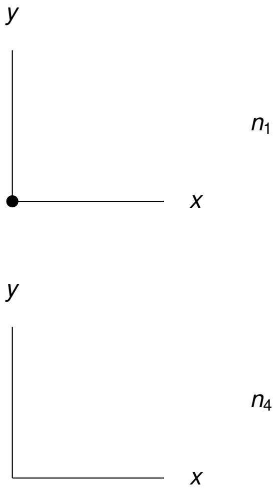




Ascending iterations

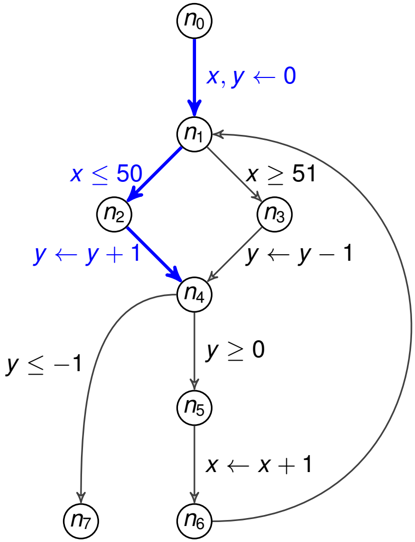
Example of Standard Abstract Interpretation

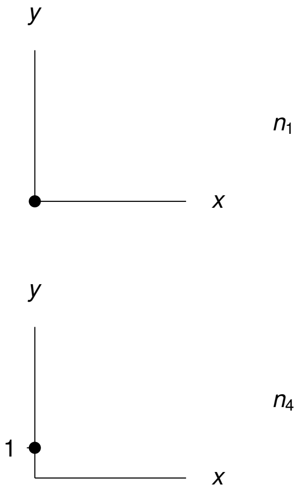




Ascending iterations

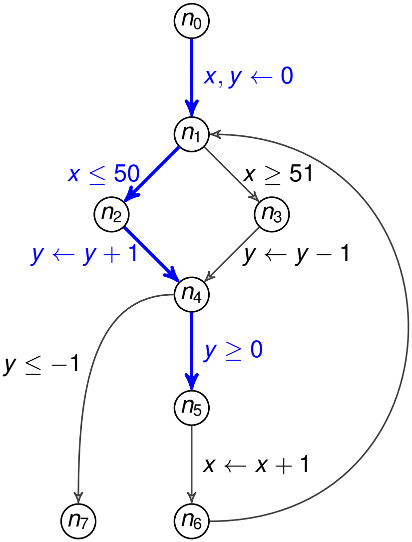
Example of Standard Abstract Interpretation

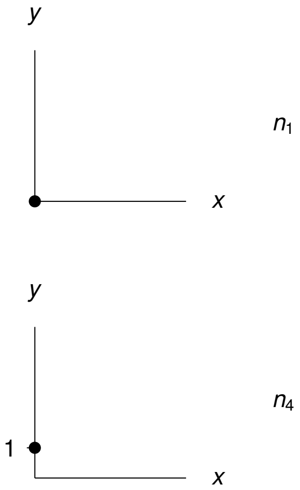




Ascending iterations

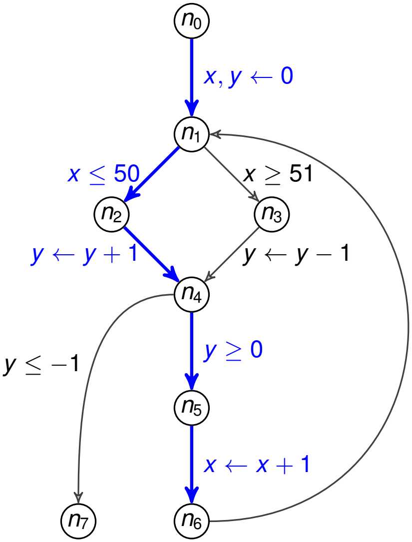
Example of Standard Abstract Interpretation

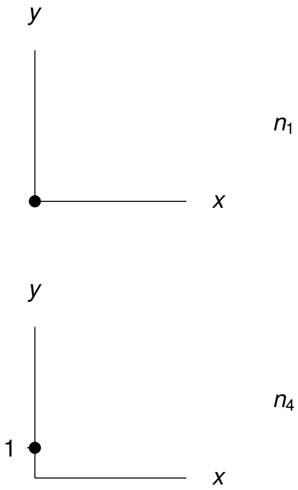




Ascending iterations

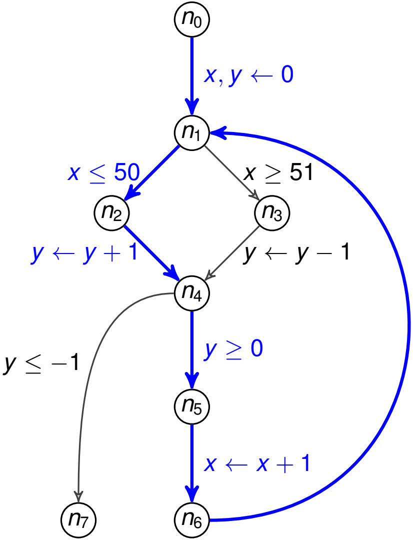
Example of Standard Abstract Interpretation

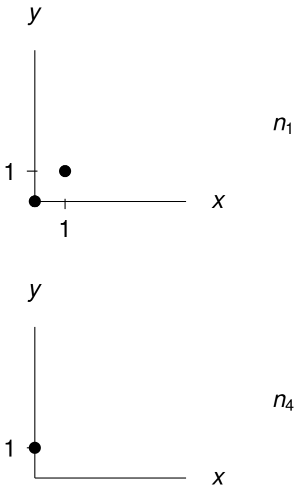




Ascending iterations

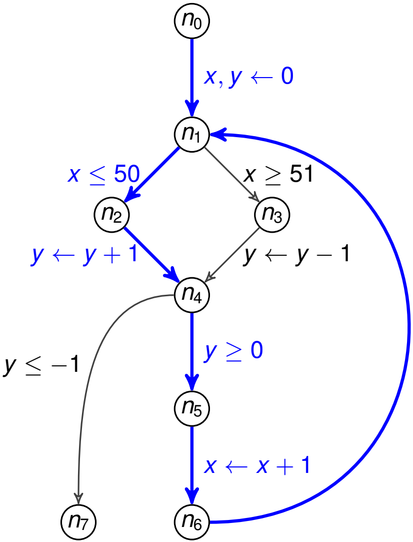
Example of Standard Abstract Interpretation

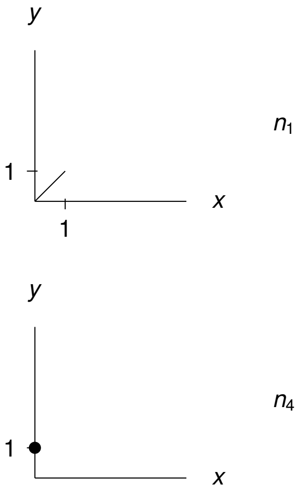




Ascending iterations

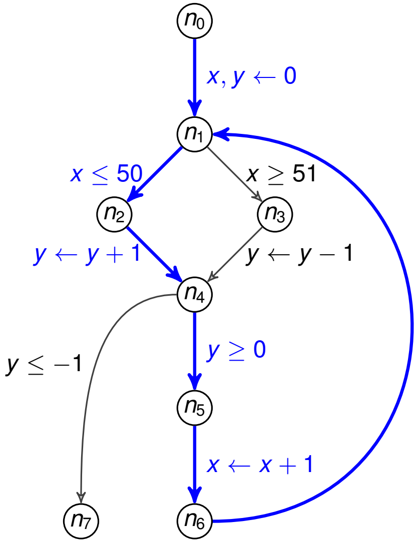
Example of Standard Abstract Interpretation

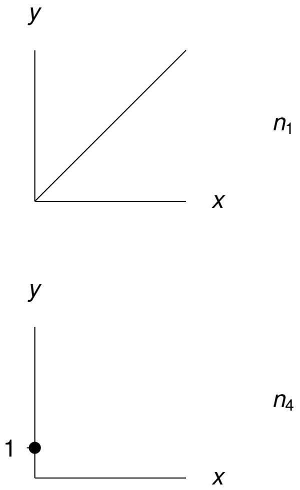




Ascending iterations

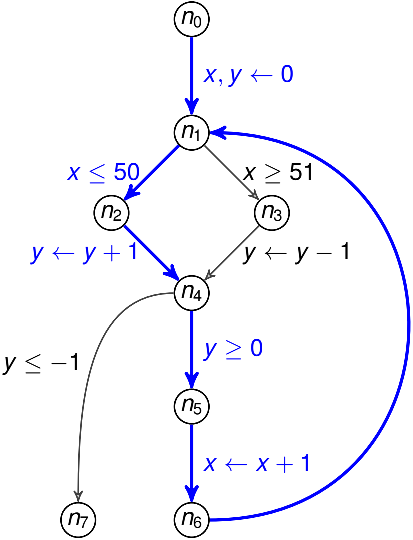
Example of Standard Abstract Interpretation

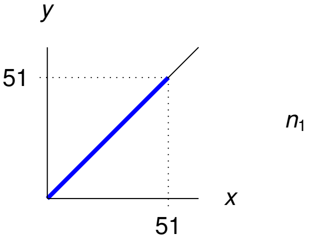


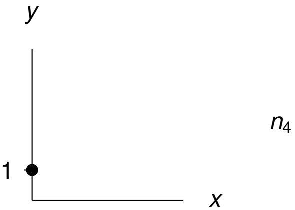


Ascending iterations

Example of Standard Abstract Interpretation

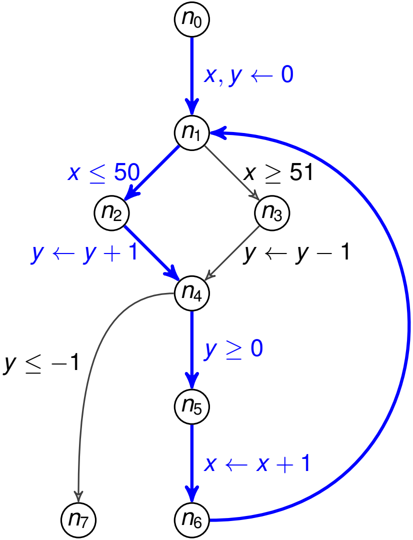


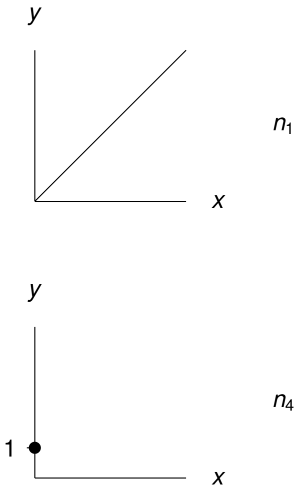




Ascending iterations

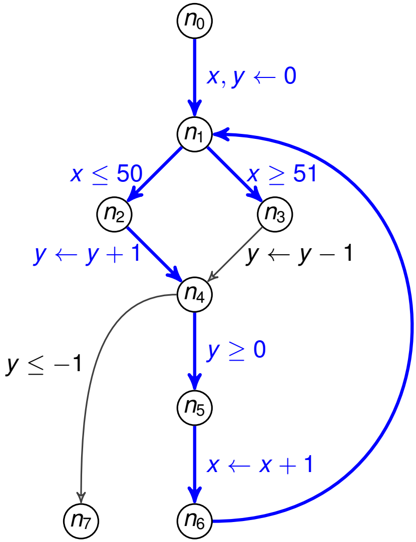
Example of Standard Abstract Interpretation

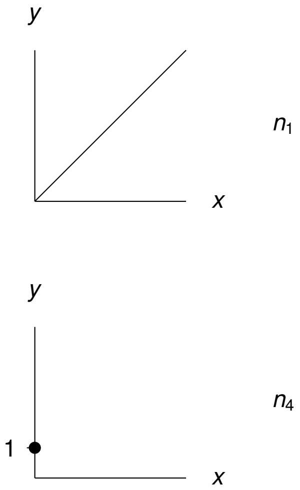




Ascending iterations

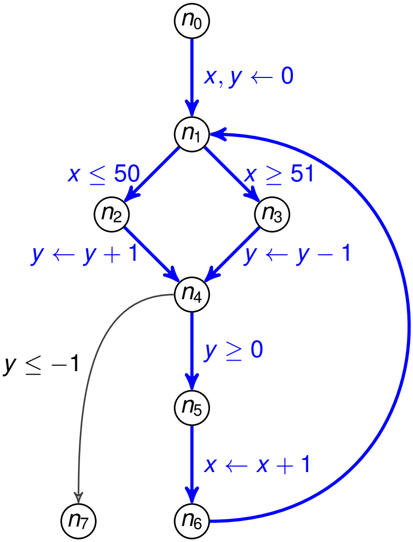
Example of Standard Abstract Interpretation

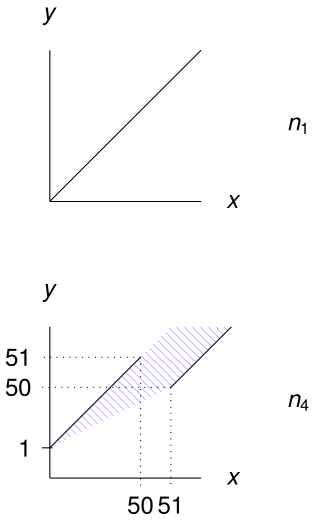




Ascending iterations

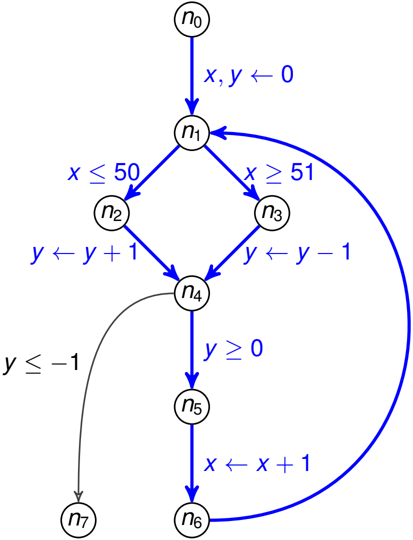
Example of Standard Abstract Interpretation

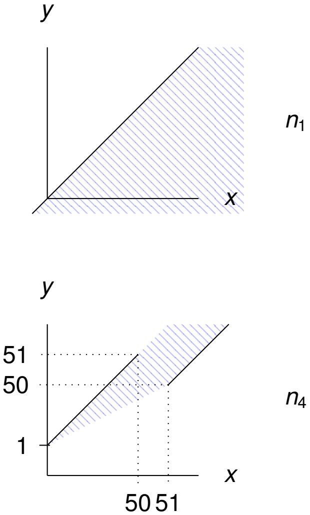




Ascending iterations

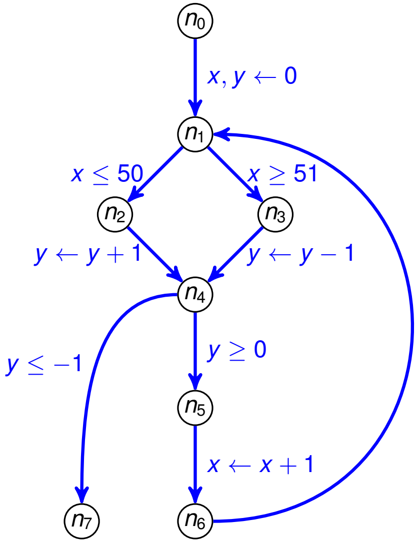
Example of Standard Abstract Interpretation

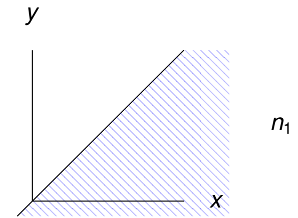


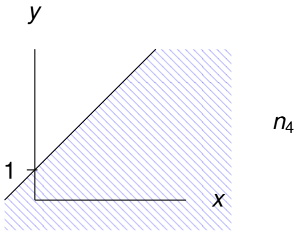


Ascending iterations

Example of Standard Abstract Interpretation

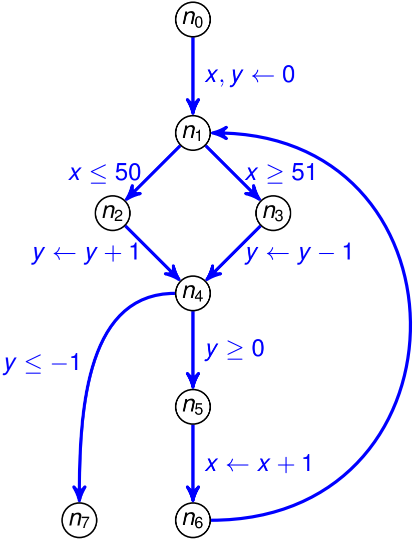


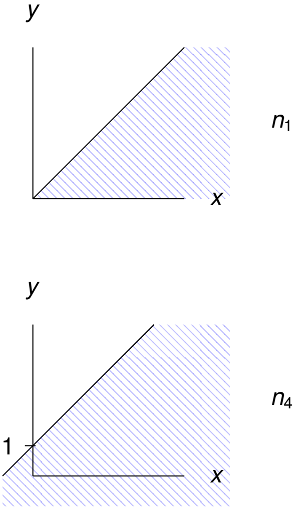




Ascending iterations

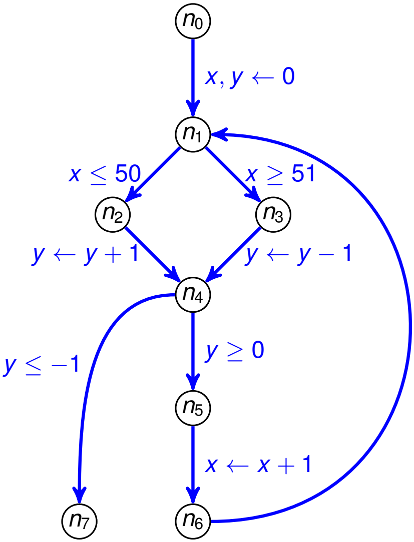
Example of Standard Abstract Interpretation

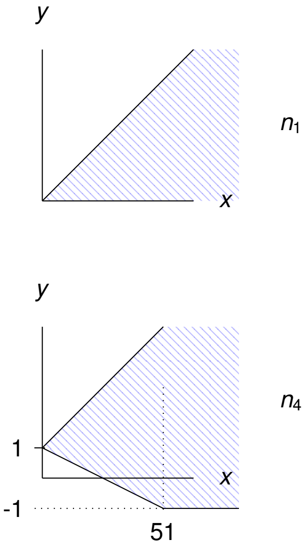




Descending iterations

Example of Standard Abstract Interpretation





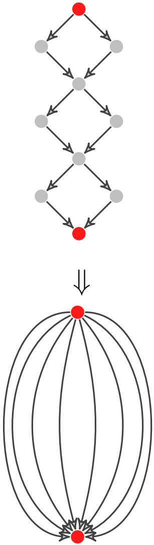
Descending iterations

Summary

Improving precision of Abstract Interpretation

Principle :

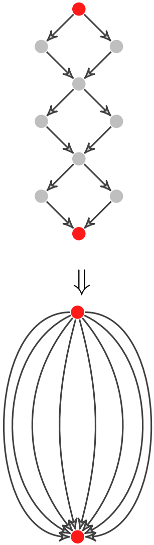
- Avoid as much as possible *Least Upper Bound* operations
- In practice, it consists in distinguishing every paths inside loops



Path Focusing

D. Monniaux & L. Gonnord - SAS 2011

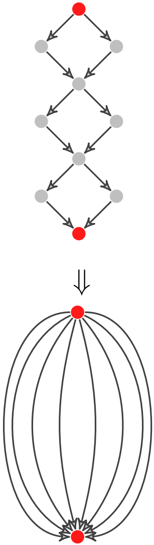
- Take a set P_R of nodes (loop headers)
- Distinguish all the paths between 2 nodes of P_R
- Compute the fixpoint iterations on the resulting “expansed” graph



Path Focusing

D. Monniaux & L. Gonnord - SAS 2011

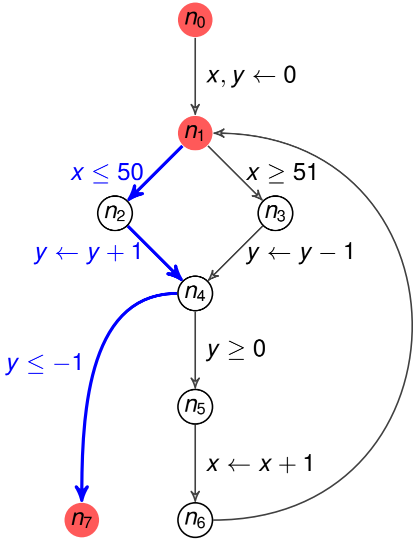
- Take a set P_R of nodes (loop headers)
- Distinguish all the paths between 2 nodes of P_R
- Compute the fixpoint iterations on the resulting "expansed" graph

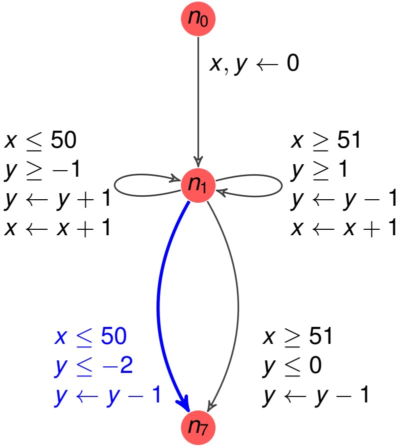


Exponential number of paths $\Rightarrow$

- We don't construct this graph explicitly
- We use SMT-solving to find interesting paths

Example : Expanded Graph





“Interesting” paths

- Abstract Interpretation : we update an invariant candidate X until it becomes an inductive invariant.
- The only “interesting” paths are those that make this invariant computation progress.

Finding Paths Using SMT-solving

We encode the semantics of the control flow graph into an SMT formula :

- 1 boolean predicate per control point and transition
- semantics of the transitions are coded w.r.t a certain theory (e.g Linear Integer Arithmetic, . . .)

We then find interesting paths using SMT queries :

"Does there exist a path starting in the invariant candidate X, that arrives in a state outside the candidate X ?"

PAGAI Static Analyser

PAGAI is a prototype of static analyser implementing state-of-the-art techniques, including our techniques using SMT.

- LLVM IR as input
- Apron Library for the abstract domains
- SMT-lib 2 interface, Microsoft Z3

In TAPAS'12:
PAGAI: a Path Sensitive Static Analyser; Henry, Monniaux, Moy

Experiments on GNU programs and WCET benchmarks

Example

For each loop header, Pagai returns an invariant:

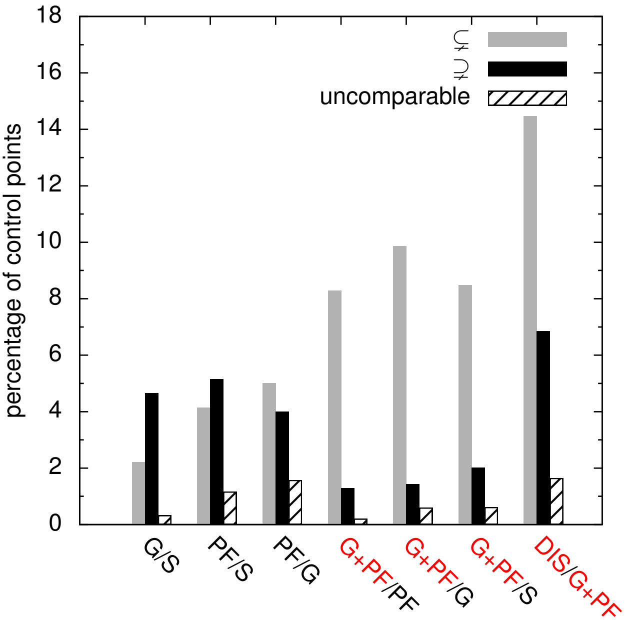
```
int main() {
        int x = 0;
        int y = 0;
        while (1) {
                /* invariant:
                102 + -1 * x + -1 * y >= 0
                y >= 0
                x + -1 * y >= 0
                */
                if (x <= 50) y++;
                else y--;
                if (y < 0) break;
                x++;
        }
}
```

Assert

```
int main() {
        int x = 0;
        int y = 0;
        while (1) {
                /* invariant:
                ...
                */
                if (x <= 50) y++;
                else y--;
                if (y < 0) break;
                x++;
        }
        /* assert OK */
        assert(x == 102);
}
```

Assume

```
void rate_limiter() {
  int x_old;
  int x;
  x_old = 0;
  while (1) {
    /* invariant:
      100000 + -1 * x >= 0
      100000 + x >= 0
    */
    x = input();
    assume (x >= -100000 && x <= 100000);
    if (x > x_old+10) x = x_old+10;
    if (x < x_old-10) x = x_old-10;
    x_old = x;
  }
/* UNREACHABLE */}
```


percentage of control points
18
16
14
12
10
8
6
4
2
0
⊊
⊋
uncomparable
G/S
PF/S
PF/G
G+PF/PF
G+PF/G
G+PF/S
DIS/G+PF

Time

	Size		Execution time (seconds)				
Name	kLOC	$\|P_R\|$	**S**	**G**	**PF**	**G+PF**	**DIS**
a2ps-4.14	55	2012	23	74	34	115	162
gawk-4.0.0	59	902	15	46	12	40	50
gnuchess-6.0.0	38	1222	50	220	81	312	351
gnugo-3.8	83	2801	77	159	92	766	1493
grep-2.9	35	820	41	85	22	65	122
gzip-1.4	27	494	22	268	91	303	230
lapack-3.3.1	954	16422	294	3740	3773	8159	10351
make-3.82	34	993	67	108	53	109	257
tar-1.26	73	1712	37	218	115	253	396

Table: Execution times