Static Analysis by Abstract Interpretation and Decision Procedures

Julien Henry

University of Grenoble

October 13, 2014

Jury

David Monniaux	Director	CNRS
Matthieu Moy	Co-Advisor	Grenoble-INP
Antoine Miné	Reviewer	ENS Paris
Cesare Tinelli	Reviewer	University of Iowa
Hugues Cassé	Examiner	IRIT
Roland Groz	Examiner	Grenoble-INP
Andreas Podelski	Examiner	University of Freiburg

Static Analysis

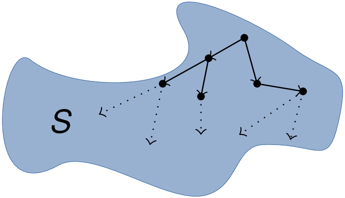
Objective:

- Discover properties on programs (invariants)
- Find possible bugs, or prove their absence.

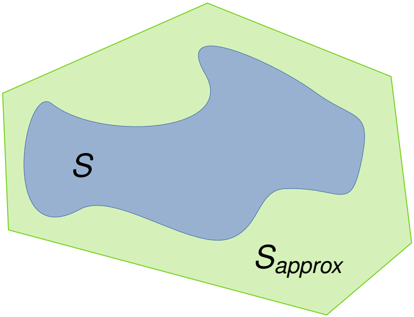
Principle:

Statically compute a set containing the reachable states of the program.

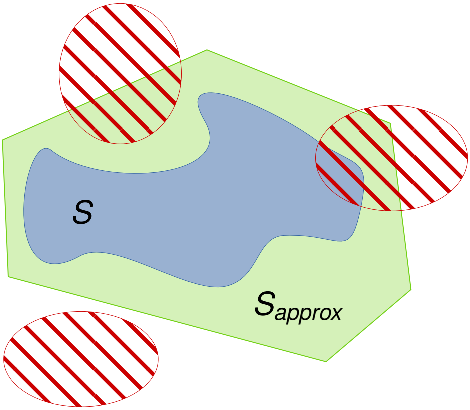
Static Analysis Uses Over-Approximations



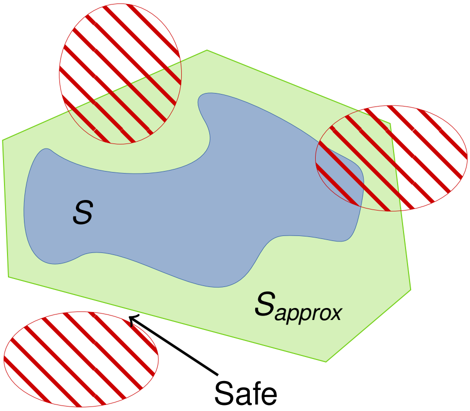
Static Analysis Uses Over-Approximations



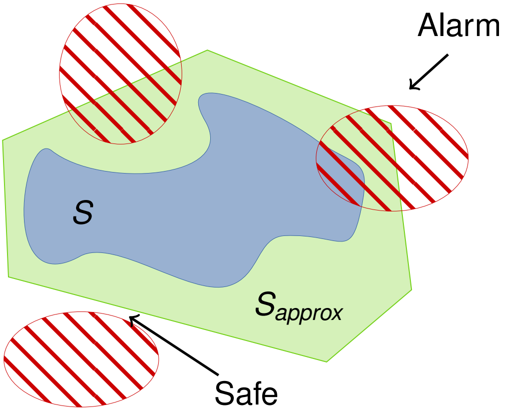
Static Analysis Uses Over-Approximations



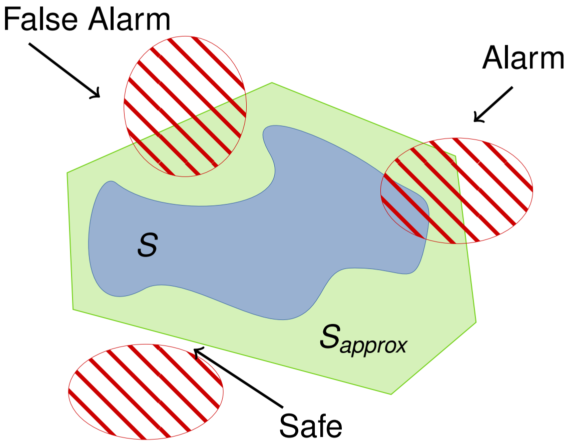
Static Analysis Uses Over-Approximations



Static Analysis Uses Over-Approximations



Static Analysis Uses Over-Approximations



Example - PAGAI Screenshot

File Edit View Search Terminal Help

```
± cat example.c
#include "../../pagai_assert.h"

int input();

int main()
{
 int x=1; int y=1;
 while(input()) {
   int t1 = x;
   int t2 = y;
   x = t1 + t2;
   y = t1 + t2;
 }
 assert(y >= 1);
 return 0;
}
```

File Edit View Search Terminal Help

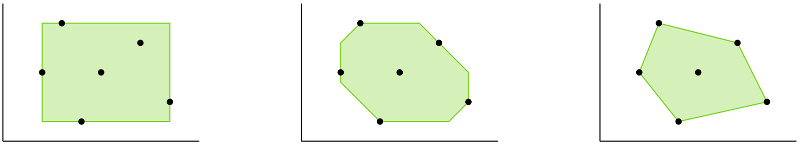
```
± pagai -i example.c
// analysis: AIopt
/* processing Function main */
#include "../../pagai_assert.h"

int input();

int main()
{
 int x=1; int y=1;
 /* reachable */
 while(/* invariant:
       -x+y = 0
       2147483647-x >= 0
       -1+x >= 0
       */
       input()) {
   int t1 = x;
   int t2 = y;
   // unsafe: possible undefined behavior
   x = t1 + t2;
   // safe
   y = t1 + t2;
 }
 /* assert OK */
 assert(y >= 1);
 /* reachable */
 return 0;
}
```

Summary

Abstract Interpretation

Cousot & Cousot 1977

Abstract domain to represent sets of states:



Intervals: $\pm x \leq C$

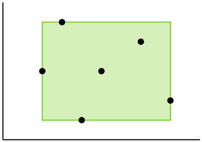
Octagons: $\pm x \pm y \leq C$

Convex Polyhedra: $\sum \alpha_i x_i \leq C$

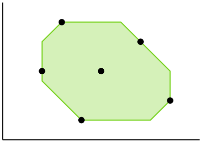
Abstract Interpretation

Cousot & Cousot 1977

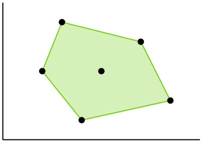
Abstract domain to represent sets of states:



Intervals:
$\pm x \leq C$



Octagons:
$\pm x \pm y \leq C$



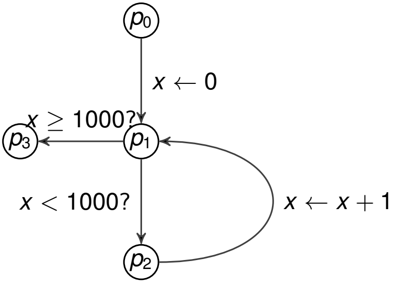
Convex Polyhedra:
$\sum \alpha_i x_i \leq C$

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

Summary

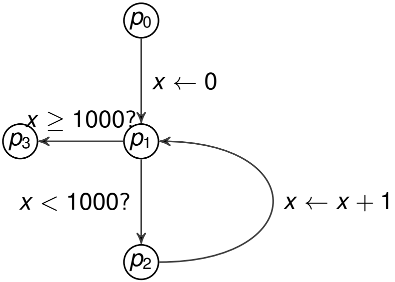
Abstract Interpretation

```
x = 0;
while (x < 1000)
{
    x++;
}
```



Abstract Interpretation

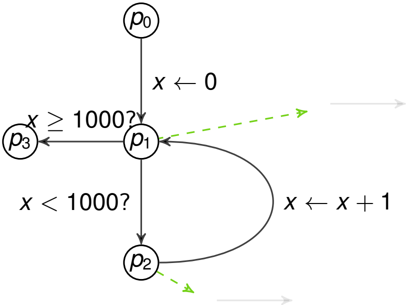
```
x = 0;
while (x < 1000)
{
    x++;
}
```



$$
\begin{aligned}
X_1 &= \{x \mid x = 0 \vee \exists x' \in X_2, x = x' + 1\} \\
X_2 &= \{x \mid x \in X_1 \wedge x < 1000\}
\end{aligned}
$$

Abstract Interpretation

```
x = 0;
while (x < 1000)
{
    x++;
}
```



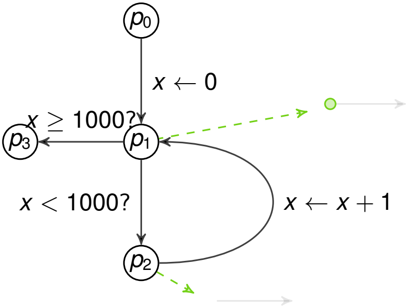
Fixpoint computation:

- All abstract values initialized to $\emptyset$
- Update until the $\subseteq$ is correct

$$
\begin{aligned}
X_1 &= \{x \mid x = 0 \vee \exists x' \in X_2, x = x' + 1\} &\subseteq \emptyset \\
X_2 &= \{x \mid x \in X_1 \wedge x < 1000\} &\subseteq \emptyset
\end{aligned}
$$

Abstract Interpretation

```
x = 0;
while (x < 1000)
{
    x++;
}
```



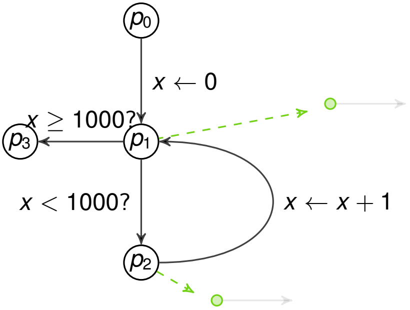
Fixpoint computation:

- All abstract values initialized to $\emptyset$
- Update until the $\subseteq$ is correct

$$
\begin{aligned}
X_1 &= \{x \mid x = 0 \vee \exists x' \in X_2, x = x' + 1\} && \subseteq [0, 0] \\
X_2 &= \{x \mid x \in X_1 \wedge x < 1000\} && \subseteq \emptyset
\end{aligned}
$$

Abstract Interpretation

```
x = 0;
while (x < 1000)
{
    x++;
}
```



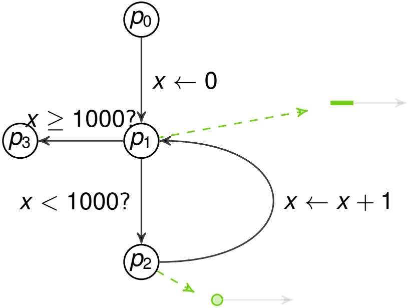
Fixpoint computation:

- All abstract values initialized to $\emptyset$
- Update until the $\subseteq$ is correct

$$
\begin{aligned}
X_1 &= \{x \mid x = 0 \lor \exists x' \in X_2, x = x' + 1\} && \subseteq [0, 0] \\
X_2 &= \{x \mid x \in X_1 \land x < 1000\} && \subseteq [0, 0]
\end{aligned}
$$

Abstract Interpretation

```
x = 0;
while (x < 1000)
{
    x++;
}
```



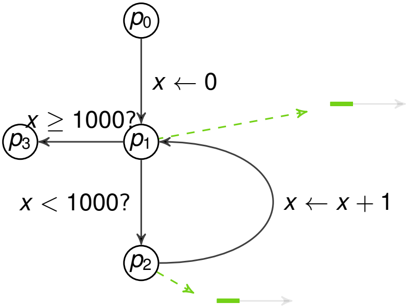
Fixpoint computation:

- All abstract values initialized to $\emptyset$
- Update until the $\subseteq$ is correct

$$
\begin{aligned}
X_1 &= \{x \mid x = 0 \vee \exists x' \in X_2, x = x' + 1\} && \subseteq [0, 1] \\
X_2 &= \{x \mid x \in X_1 \wedge x < 1000\} && \subseteq [0, 0]
\end{aligned}
$$

Abstract Interpretation

```
x = 0;
while (x < 1000)
{
    x++;
}
```



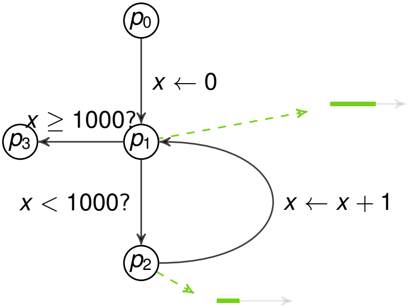
Fixpoint computation:

- All abstract values initialized to $\emptyset$
- Update until the $\subseteq$ is correct

$$
\begin{aligned}
X_1 &= \{x \mid x = 0 \vee \exists x' \in X_2, x = x' + 1\} &\subseteq [0, 1] \\
X_2 &= \{x \mid x \in X_1 \wedge x < 1000\} &\subseteq [0, 1]
\end{aligned}
$$

Abstract Interpretation

```
x = 0;
while (x < 1000)
{
    x++;
}
```



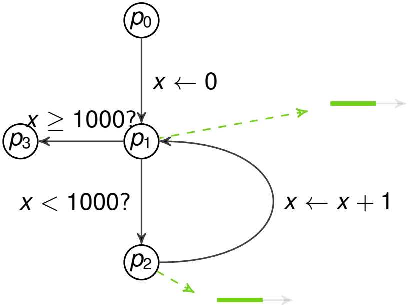
Fixpoint computation:

- All abstract values initialized to $\emptyset$
- Update until the $\subseteq$ is correct

$$
\begin{aligned}
X_1 &= \{x \mid x = 0 \lor \exists x' \in X_2, x = x' + 1\} &\subseteq [0, 2] \\
X_2 &= \{x \mid x \in X_1 \land x < 1000\} &\subseteq [0, 1]
\end{aligned}
$$

Abstract Interpretation

```
x = 0;
while (x < 1000)
{
    x++;
}
```



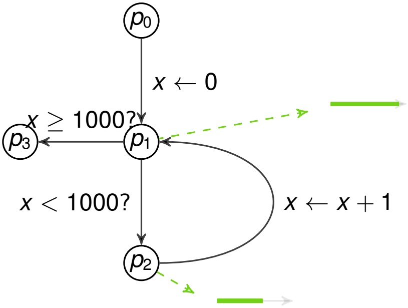
Fixpoint computation:

- All abstract values initialized to $\emptyset$
- Update until the $\subseteq$ is correct

$$
\begin{aligned}
X_1 &= \{x \mid x = 0 \vee \exists x' \in X_2, x = x' + 1\} && \subseteq [0, 2] \\
X_2 &= \{x \mid x \in X_1 \wedge x < 1000\} && \subseteq [0, 2]
\end{aligned}
$$

Abstract Interpretation

```
x = 0;
while (x < 1000)
{
    x++;
}
```



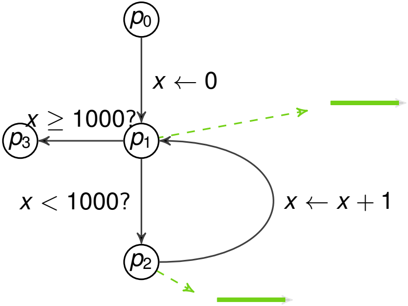
Fixpoint computation:

- All abstract values initialized to $\emptyset$
- Update until the $\subseteq$ is correct

$$
\begin{aligned}
X_1 &= \{x \mid x = 0 \vee \exists x' \in X_2, x = x' + 1\} &\subseteq [0, 3] \\
X_2 &= \{x \mid x \in X_1 \wedge x < 1000\} &\subseteq [0, 2]
\end{aligned}
$$

Abstract Interpretation

```
x = 0;
while (x < 1000)
{
    x++;
}
```



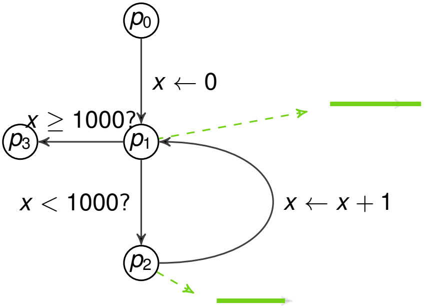
Fixpoint computation:

- All abstract values initialized to $\emptyset$
- Update until the $\subseteq$ is correct

$$
\begin{aligned}
X_1 &= \{x \mid x = 0 \vee \exists x' \in X_2, x = x' + 1\} && \subseteq [0,3] \\
X_2 &= \{x \mid x \in X_1 \wedge x < 1000\} && \subseteq [0,3]
\end{aligned}
$$

Abstract Interpretation

```
x = 0;
while (x < 1000)
{
    x++;
}
```



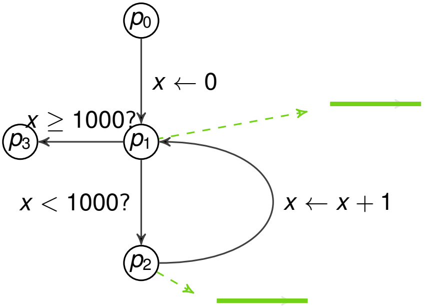
Fixpoint computation:

- All abstract values initialized to $\emptyset$
- Update until the $\subseteq$ is correct

$$X_1 = \{x \mid x = 0 \vee \exists x' \in X_2, x = x' + 1\} \subseteq [0, 4]$$
$$X_2 = \{x \mid x \in X_1 \wedge x < 1000\} \subseteq [0, 3]$$

Abstract Interpretation

```
x = 0;
while (x < 1000)
{
    x++;
}
```



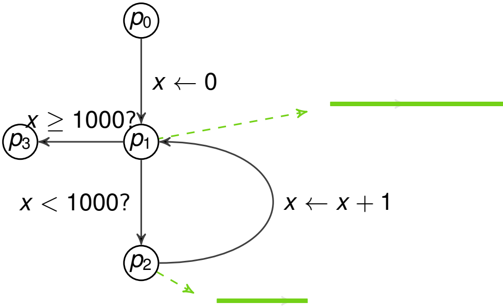
Fixpoint computation:

- All abstract values initialized to $\emptyset$
- Update until the $\subseteq$ is correct

$$
\begin{aligned}
X_1 &= \{x \mid x = 0 \vee \exists x' \in X_2, x = x' + 1\} && \subseteq [0, 4] \\
X_2 &= \{x \mid x \in X_1 \wedge x < 1000\} && \subseteq [0, 4]
\end{aligned}
$$

Abstract Interpretation

```
x = 0;
while (x < 1000)
{
    x++;
}
```



Fixpoint computation:

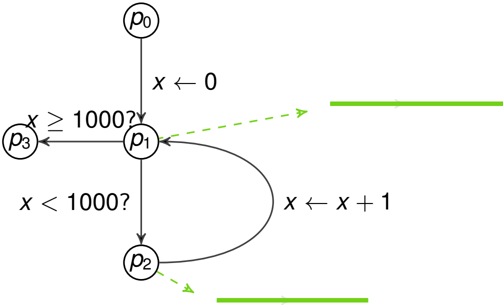
- All abstract values initialized to $\emptyset$
- Update until the $\subseteq$ is correct

WIDENING

$$
\begin{aligned}
X_1 &= \{x \mid x = 0 \lor \exists x' \in X_2, x = x' + 1\} &\subseteq [0, +\infty[\\
X_2 &= \{x \mid x \in X_1 \land x < 1000\} &\subseteq [0, 4]
\end{aligned}
$$

Abstract Interpretation

```
x = 0;
while (x < 1000)
{
    x++;
}
```



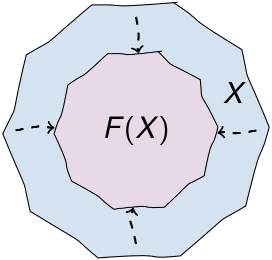
Fixpoint computation:

- All abstract values initialized to $\emptyset$
- Update until the $\subseteq$ is correct

$$
\begin{aligned}
X_1 &= \{x \mid x = 0 \vee \exists x' \in X_2, x = x' + 1\} && \subseteq [0, +\infty[\\
X_2 &= \{x \mid x \in X_1 \wedge x < 1000\} && \subseteq [0, 999]
\end{aligned}
$$

Narrowing

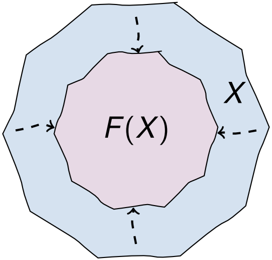
An inductive invariant has been found: $F(X) \subseteq X$, we can recover precision by iterating once more:



$$
\begin{aligned}
X_1 &= \{x \mid x = 0 \vee \exists x' \in X_2, x = x' + 1\} && \subseteq [0, +\infty[\\
X_2 &= \{x \mid x \in X_1 \wedge x < 1000\} && \subseteq [0, 999]
\end{aligned}
$$

Narrowing

An inductive invariant has been found: $F(X) \subseteq X$, we can recover precision by iterating once more:



$$
\begin{aligned}
X_1 &= \{x \mid x = 0 \vee \exists x' \in X_2, x = x' + 1\} & \subseteq [0, 1000] \\
X_2 &= \{x \mid x \in X_1 \wedge x < 1000\} & \subseteq [0, 999]
\end{aligned}
$$

Sources of Imprecision

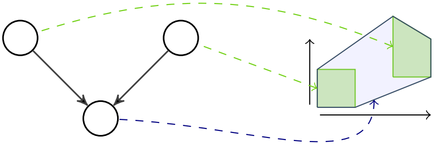
- Limited expressivity of the abstract domain (e.g. linear inequalities)

Sources of Imprecision

- Limited expressivity of the abstract domain (e.g. linear inequalities)
- Widening operator
  - Ensures termination, but may induce huge imprecisions
  - Narrowing tends to recover some precision…

Sources of Imprecision

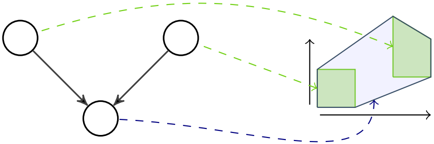
- Limited expressivity of the abstract domain (e.g. linear inequalities)
- Widening operator
  - Ensures termination, but may induce huge imprecisions
  - Narrowing tends to recover some precision...
- Control flow merges
  - Analysis catches paths that are unfeasible concretely



$\bigsqcup^{\#}$ = Least Upper Bound

Sources of Imprecision

- Limited expressivity of the abstract domain (e.g. linear inequalities)
- Widening operator
  - Ensures termination, but may induce huge imprecisions
  - Narrowing tends to recover some precision...
- Control flow merges
  - Analysis catches paths that are unfeasible concretely



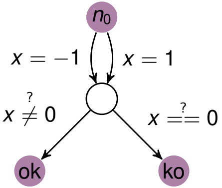
$\bigsqcup^{\#}$ = Least Upper Bound

In this thesis
Improve precision of the analysis by limiting the bad effects of **widenings** and **least upper bounds**

Summary

Example

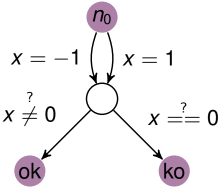
```
if (input())
    x = 1;
else
    x = -1;
    // (here)
if (x == 0)
    abort();
else
    OK();
```

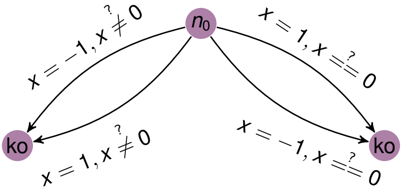


- x can be 1 or -1
- Least upper bound yields $x \in [-1, 1]$ at point `(here)`
- `if (x == 0)` seems feasible with traditional AI

Example

```
if (input())
    x = 1;
else
    x = -1;
    // (here)
if (x == 0)
    abort();
else
    OK();
```

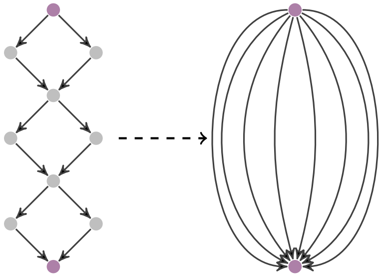




- x can be 1 or -1
- Least upper bound yields $x \in [-1, 1]$ at point `(here)`
- `if (x == 0)` seems feasible with traditional AI

Path Focusing (Monniaux & Gonnord SAS11)

Idea: delay control-flow merges

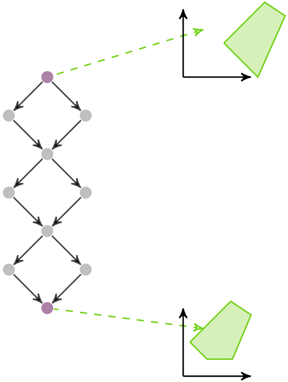


- Expand and distinguish every paths inside loops
- Abstraction only at the loop headers
- Succinctly represent the set of paths using a logical formula (SMT)

"Interesting" paths

Algorithm: update an abstract value X until it becomes an inductive invariant: $F(X) \subseteq X$.

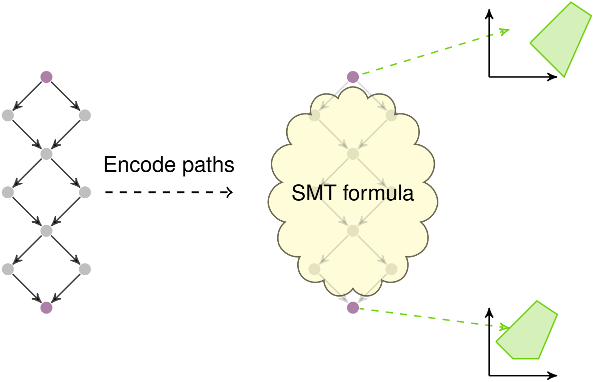
The only "interesting" paths are those that make this invariant computation progress.



"Interesting" paths

Algorithm: update an abstract value X until it becomes an inductive invariant: $F(X) \subseteq X$.

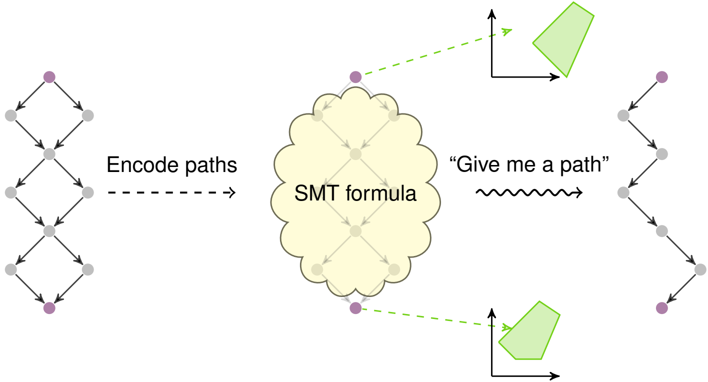
The only "interesting" paths are those that make this invariant computation progress.



"Interesting" paths

Algorithm: update an abstract value X until it becomes an inductive invariant: $F(X) \subseteq X$.

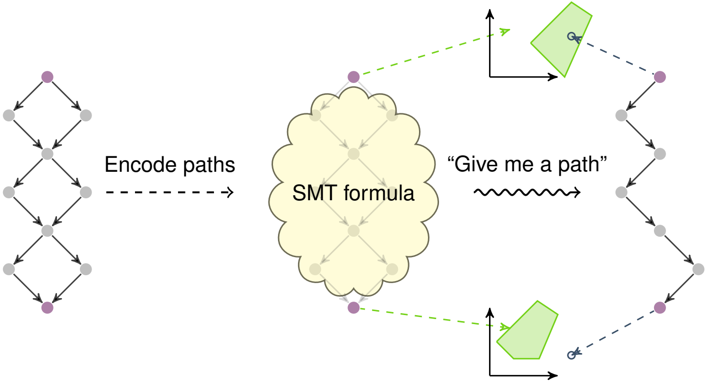
The only "interesting" paths are those that make this invariant computation progress.



"Interesting" paths

Algorithm: update an abstract value X until it becomes an inductive invariant: $F(X) \subseteq X$.

The only "interesting" paths are those that make this invariant computation progress.



Using SMT-solving for Choosing Paths

SMT formula ρ expressing the semantics of the program paths:

- Control-Flow encoded using Booleans
- Over-approximation of the instructions semantics in LIRA

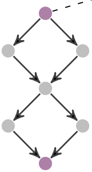
Using SMT-solving for Choosing Paths

SMT formula ρ expressing the semantics of the program paths:

- Control-Flow encoded using Booleans
- Over-approximation of the instructions semantics in LIRA

"Does there exist a path starting inside the candidate invariant, that goes to a state outside the candidate invariant?"

check-sat: $\rho \wedge b_i^s \wedge X_i \wedge \bigvee_{j \in Succ(i)} (b_j^d \wedge \neg X_j)$



starting value

destination value

starting point

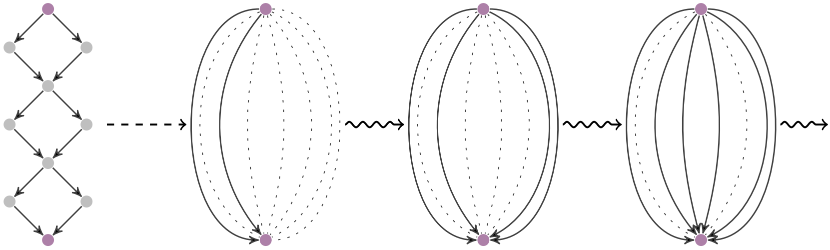
destination

Guided Path Analysis (Henry & al)

SAS'12: "Succinct Representations for Abstract Interpretation"

Imprecision due to widening spreads

Apply narrowing **before** it is too late
→ before an invariant for the entire program is found

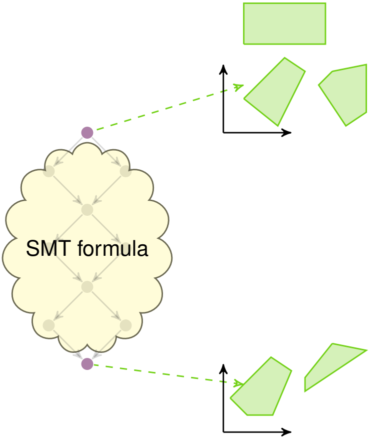


Compute **precise** invariants for a sequence of subprograms

Guided Path Analysis (Henry & al)

Extensions:

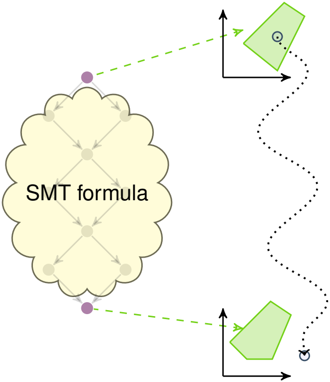
1. Disjunctive Invariants



Guided Path Analysis (Henry & al)

Extensions:

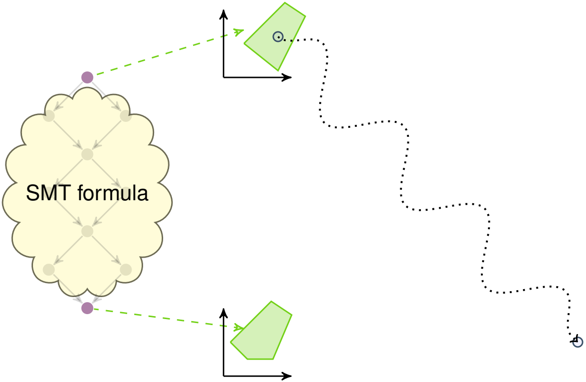
1. Disjunctive Invariants
2. “Interesting traces” **far** from the current abstract value



Guided Path Analysis (Henry & al)

Extensions:

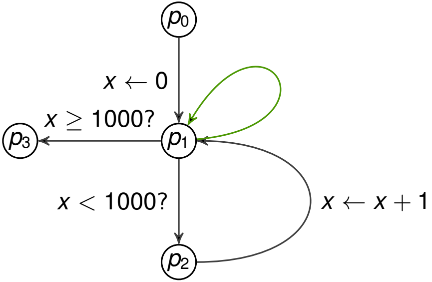
1. Disjunctive Invariants
2. “Interesting traces” **far** from the current abstract value



Improve the Decreasing Sequence (Halbwachs & Henry)

SAS'12: "When the Decreasing Sequence Fails"

Decreasing sequence does not always work



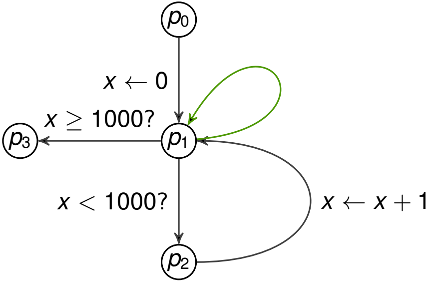
$$
\begin{aligned}
X_1 &= \{x \mid x = 0 \vee \exists x' \in X_2, x = x' + 1 \vee x \in X_1\} &\subseteq [0, +\infty) \\
X_2 &= \{x \mid x \in X_1 \wedge x < 1000\} &\subseteq [0, 999]
\end{aligned}
$$

Improve the Decreasing Sequence (Halbwachs & Henry)

SAS'12: "When the Decreasing Sequence Fails"

Decreasing sequence does not always work

Restart an analysis from a different, **well chosen**, initial value



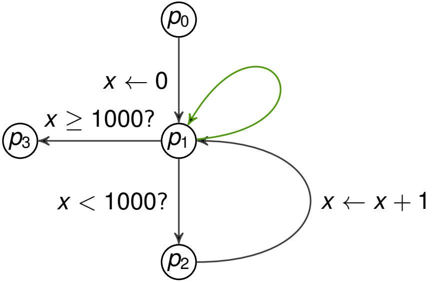
$$
\begin{aligned}
X_1 &= \{x \mid x = 0 \lor \exists x' \in X_2, x = x' + 1 \lor x \in X_1\} && \subseteq \bot \\
X_2 &= \{x \mid x \in X_1 \land x < 1000\} && \subseteq [0, 999]
\end{aligned}
$$

Improve the Decreasing Sequence (Halbwachs & Henry)

SAS'12: "When the Decreasing Sequence Fails"

Decreasing sequence does not always work

Restart an analysis from a different, **well chosen**, initial value



$$
\begin{aligned}
X_1 &= \{x \mid x = 0 \vee \exists x' \in X_2, x = x' + 1 \vee x \in X_1\} &\subseteq [0, 1000] \\
X_2 &= \{x \mid x \in X_1 \wedge x < 1000\} &\subseteq [0, 999]
\end{aligned}
$$

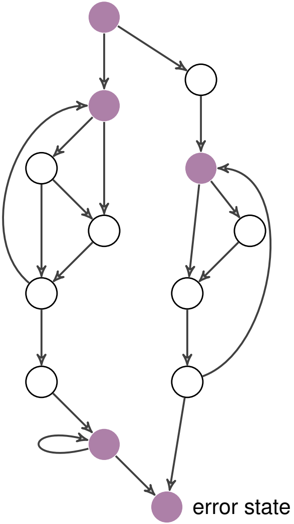
Summary

Incremental Analysis

- Abstract Interpretation can be parametrized in many ways
- From very cheap to very expensive techniques/abstract domains

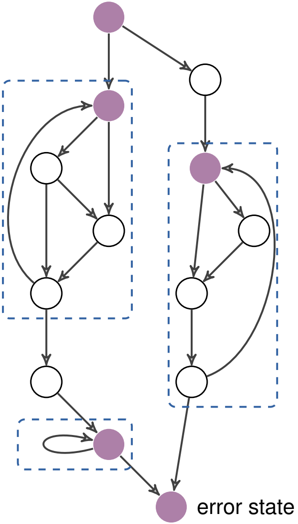
Run cheap techniques first, and refine program portions if needed

Principle



Complicated CFG

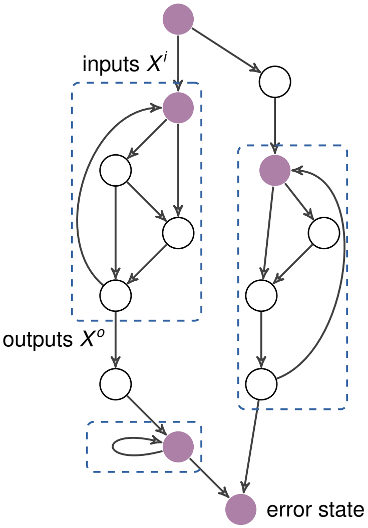
Principle



Select blocks/portions
to be abstracted:

- Loops,
- Function calls,
- Complicated program portions

Principle

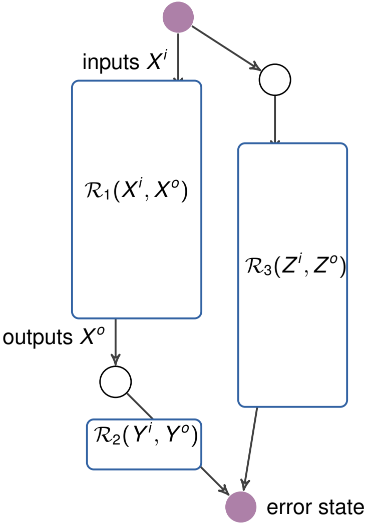


Select blocks/portions to be abstracted:

- Loops,
- Function calls,
- Complicated program portions

Each block has **input** and **output** variables

Principle

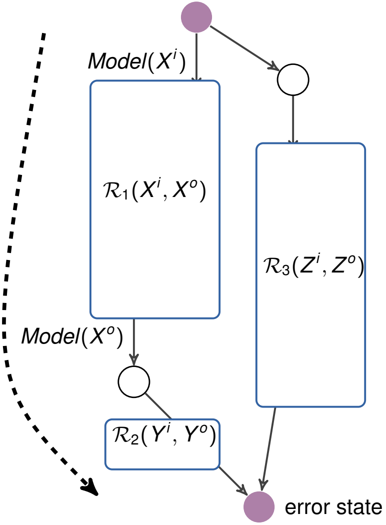


Abstract each block by a logical formula.

$\mathcal{R}_1(X^i, X^o)$ is a formula involving **inputs** and **outputs**
Example: $x^i > 0 \Rightarrow x^o = x^i + 1$

Initialized to **true**
(= safe over-approximation)

Principle



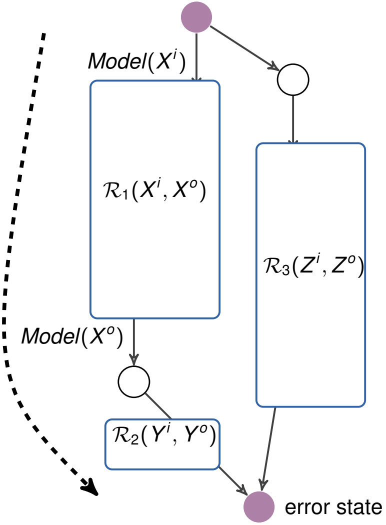
SMT query:

"Is there a path to the error state ?"

$$Model(X^i) \wedge Model(X^o) \wedge \mathcal{R}_1(X^i, X^o)$$

is SAT

Principle



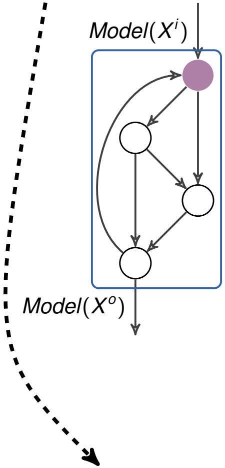
SMT query:

"Is there a path to the error state ?"

$$Model(X^i) \wedge Model(X^o) \wedge \mathcal{R}_1(X^i, X^o)$$
is SAT

$\rightarrow$ Improve precision of $\mathcal{R}_1(X^i, X^o)$ s.t. above formula becomes UNSAT

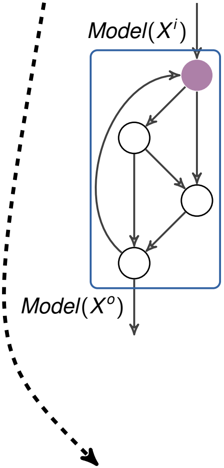
Principle



Compute a new relation for the block, with the knowledge of the input context $Model(X^i)$

Of the form:
$Model(X^i) \Rightarrow \mathcal{F}(X^i, X^o)$

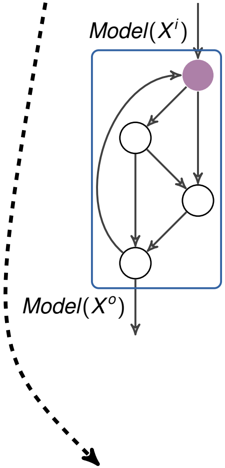
Principle



Compute a new relation for the block, with the knowledge of the input context $Model(X^i)$

Of the form:
$Model(X^i) \Rightarrow \mathcal{F}(X^i, X^o)$
Not sufficiently general...

Principle

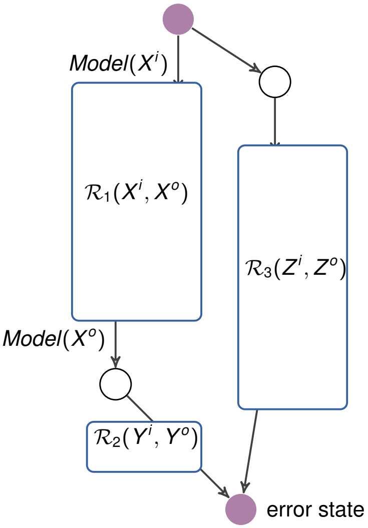


Compute a new relation for the block, with the knowledge of the input context $Model(X^i)$

Of the form:
$Model(X^i) \Rightarrow \mathcal{F}(X^i, X^o)$
Not sufficiently general...

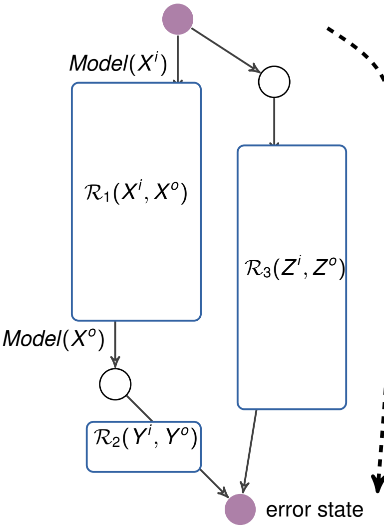
Generalize the valid context during analysis:
$C(X^i) \Rightarrow \mathcal{F}(X^i, X^o)$
$Model(X^i) \Rightarrow C(X^i)$

Principle



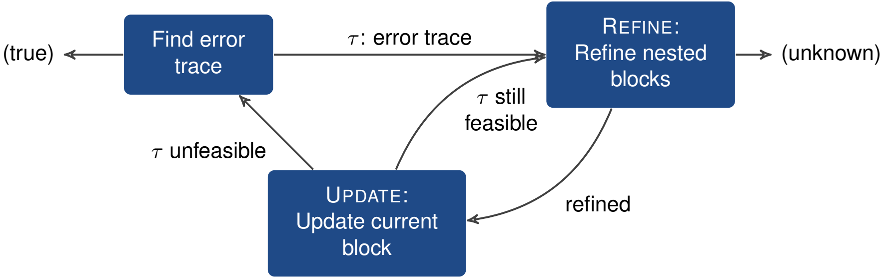
Continue and search for new error trace

Principle



Continue and search for new error trace

Modular Static Analysis: Overview



Summary

PAGAI, in one slide

TAPAS'12: "PAGAI: a path sensitive static analyser"

Static analyzer for LLVM IR, written in C++, > 20.000 LOC

- Most of the techniques described here are implemented
- PAGAI checks:
  - array out-of-bounds
  - integer overflows
  - `assert` statements
- Handles **real** C programs & SV-Comp benchmarks
- Already used outside Verimag

Comparisons of Various Techniques

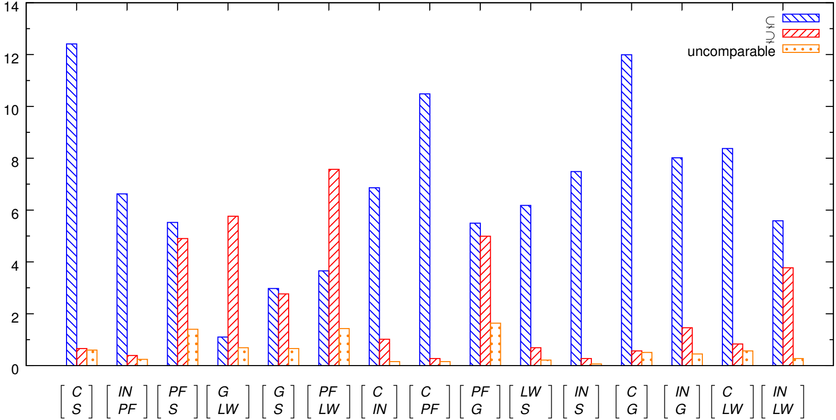
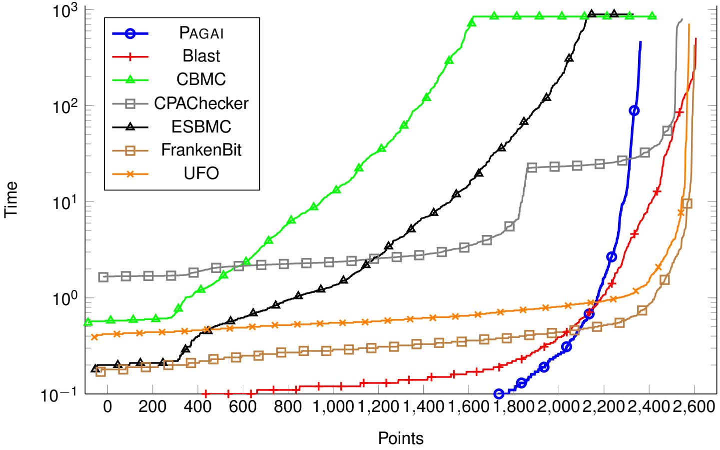


Figure : Malärdalen benchmarks

`http://www.mrtc.mdh.se/projects/wcet/benchmarks.html`

Software-Verification Competition



Summary

Target: Reactive Control Systems

```
void main() {
  while (1) {
        READ_INPUTS();
        COMPUTE();
        WRITE_OUTPUTS();
  }
}
```

1 “big” infinite loop

$\sim$ Loop-free body

Goal: WCET for 1 loop iteration $<$ some bound

Our Method

LCTES'14: "How to Compute Worst-Case Execution Time by Optimization Modulo Theory and a Clever Encoding of Program Semantics"

Input:

- **Loop-free** control-flow graph of the loop body
- timings for basic blocks (# clock cycles)
  - given by an external tool, e.g. OTAWA
  - runs a panel of static analysis, considering micro-architecture

Principle: Encode the problem into SMT and optimize a cost function

Output:

- WCET for the entire CFG + Worst Case path

Computing the WCET

Optimization modulo Theory:
We search for the trace maximizing the variable cost.

Using any off-the-shelf SMT solver

Computing the WCET

Optimization modulo Theory:
We search for the trace maximizing the variable cost.

Using any off-the-shelf SMT solver

Dichotomy strategy:
Maintain an interval containing the WCET

- Initial interval $[0, 100]$

Computing the WCET

Optimization modulo Theory:
We search for the trace maximizing the variable cost.

Using any off-the-shelf SMT solver

Dichotomy strategy:
Maintain an interval containing the WCET

- Initial interval $[0, 100]$
- Is there a trace where *cost* > 50? Yes, 70

Computing the WCET

Optimization modulo Theory:
We search for the trace maximizing the variable cost.

Using any off-the-shelf SMT solver

Dichotomy strategy:
Maintain an interval containing the WCET

- Initial interval $[0, 100]$
- Is there a trace where *cost* > 50? Yes, 70
- new interval $[70, 100]$

Computing the WCET

Optimization modulo Theory:
We search for the trace maximizing the variable cost.

Using any off-the-shelf SMT solver

Dichotomy strategy:
Maintain an interval containing the WCET

- Initial interval $[0, 100]$
- Is there a trace where *cost* > 50? Yes, 70
- new interval $[70, 100]$
- Is there a trace where *cost* > 85? No

Computing the WCET

Optimization modulo Theory:
We search for the trace maximizing the variable <u>cost</u>.

Using any off-the-shelf SMT solver

Dichotomy strategy:
Maintain an interval containing the WCET

- Initial interval $[0, 100]$
- Is there a trace where *cost* > 50? Yes, 70
- new interval $[70, 100]$
- Is there a trace where *cost* > 85? No
- new interval $[70, 85]$

Computing the WCET

Optimization modulo Theory:
We search for the trace maximizing the variable <u>cost</u>.

Using any off-the-shelf SMT solver

Dichotomy strategy:
Maintain an interval containing the WCET

- Initial interval $[0, 100]$
- Is there a trace where *cost* > 50? Yes, 70
- new interval $[70, 100]$
- Is there a trace where *cost* > 85? No
- new interval $[70, 85]$
- ...

A Really Simple Example

$b_1, \dots, b_n$ unconstrained Booleans, **xi**'s and **yi**'s are the timing costs

```
if (b1) { /*c1=2*/ } else { /*c1=3*/ } //cost c1
if (b1) { /*c1'=3*/ } else { /*c1'=2*/ } //cost c1'
...
if (bn) { /*cn=2*/ } else { /*cn=3*/ } //cost cn
if (bn) { /*cn'=3*/ } else { /*cn'=2*/ } //cost cn'
```

"Obviously" all traces take time $(3+2)n = 5n$.

A Really Simple Example

$b_1, \ldots, b_n$ unconstrained Booleans, **xi**'s and **yi**'s are the timing costs

```
if (b1) { /*c1=2*/ } else { /*c1=3*/ } //cost c1
if (b1) { /*c1'=3*/ } else { /*c1'=2*/ } //cost c1'
...
if (bn) { /*cn=2*/ } else { /*cn=3*/ } //cost cn
if (bn) { /*cn'=3*/ } else { /*cn'=2*/ } //cost cn'
```

"Obviously" all traces take time $(3 + 2)n = 5n$.

SMT approach (using DPLL(T)) will find 5n, but in exponential time. . .

Why such high cost?

$$(b_1 \Rightarrow c_1 = 2) \wedge (\neg b_1 \Rightarrow c_1 = 3) \wedge (b_1 \Rightarrow c'_1 = 3) \wedge (\neg b_1 \Rightarrow c'_1 = 2) \wedge$$
$$\cdots \wedge$$
$$(b_n \Rightarrow c_n = 2) \wedge (\neg b_n \Rightarrow c_n = 3) \wedge (b_n \Rightarrow c'_n = 3) \wedge (\neg b_n \Rightarrow c'_n = 2) \wedge$$
$$c_1 + c'_1 + \cdots + c_n + c'_n > 5n$$

A SMT-solver based on "DPLL($\mathcal{T}$)":

- enumerates a Boolean choice tree over $b_1, \ldots, b_n$
- cuts branches when encountering **inconsistent numerical constraints.**

What are the inconsistent numerical constraints here (**blocking clauses**)?

Take the satisfying assignment where all the b_i's are set to true (the c_i's $= 2$ and c'_i's $= 3$)

THEORY ATOMS

$c_1 \leq 2$ $\quad$ $c_n \leq 2$

$c_1 \leq 3$ $\quad \cdots \quad$ $c_n \leq 3$

$\neg(c'_1 \leq 2)$ $\quad$ $\neg(c'_n \leq 2)$

$c'_1 \leq 3$ $\quad$ $c'_n \leq 3$

$c_1 + c'_1 + \cdots + c_n + c'_n > 5n$

BLOCKING CLAUSE

Take the satisfying assignment where all the b_i's are set to true (the c_i's $= 2$ and c_i''s $= 3$)

THEORY ATOMS		
$c_1 \leq 2$		$c_n \leq 2$
$c_1 \leq 3$	$\cdots$	$c_n \leq 3$
$\neg(c_1' \leq 2)$		$\neg(c_n' \leq 2)$
$c_1' \leq 3$		$c_n' \leq 3$
$c_1 + c_1' + \cdots + c_n + c_n' > 5n$		

BLOCKING CLAUSE		
$c_1 \leq 2$		$c_n \leq 2$
$\cancel{c_1 \leq 3}$	$\cdots$	$\cancel{c_n \leq 3}$
$\cancel{\neg(c_1' \leq 2)}$		$\cancel{\neg(c_n' \leq 2)}$
$c_1' \leq 3$		$c_n' \leq 3$
$c_1 + c_1' + \cdots + c_n + c_n' > 5n$		

Only cuts one single program trace…

Take the satisfying assignment where all the b_i's are set to true (the c_i's $= 2$ and c_i''s $= 3$)

THEORY ATOMS			BLOCKING CLAUSE		
$c_1 \leq 2$		$c_n \leq 2$	$c_1 \leq 2$		$c_n \leq 2$
$c_1 \leq 3$	$\cdots$	$c_n \leq 3$	$\cancel{c_1 \leq 3}$	$\cdots$	$\cancel{c_n \leq 3}$
$\neg(c_1' \leq 2)$		$\neg(c_n' \leq 2)$	$\cancel{\neg(c_1' \leq 2)}$		$\cancel{\neg(c_n' \leq 2)}$
$c_1' \leq 3$		$c_n' \leq 3$	$c_1' \leq 3$		$c_n' \leq 3$
$c_1 + c_1' + \cdots + c_n + c_n' > 5n$			$c_1 + c_1' + \cdots + c_n + c_n' > 5n$		

Only cuts one single program trace...
2^n of them. The solver has to prove them inconsistent one by one.

Untractability Issue

SMT solvers miss "obvious" properties

```
...
if (b_i) { /* ci=2 */ } else { /* ci=3*/ }
if (b_i) { /* ci'=3 */ } else { /* ci'=2*/ }
...
```

Human remark: "**obviously**, $c_i + c_i' \leq 5$"

"Normal" DPLL(T)-based SMT solvers **do not invent new atomic predicates**: they can't learn it...

Untractability Issue

SMT solvers miss “obvious” properties

```
...
if (b_i) { /* ci=2 */ } else { /* ci=3*/ }
if (b_i) { /* ci'=3 */ } else { /* ci'=2*/ }
...
```

Human remark: “**obviously**, $c_i + c_i' \leq 5$”

“Normal” DPLL(T)-based SMT solvers **do not invent new atomic predicates**: they can’t learn it...

What if we simply conjoin these predicates to the SMT formula ?

Again, take the satisfying assignment where all the b_i's are set to true (the c_i's $= 2$ and c_i''s $= 3$)

THEORY ATOMS

$c_1 \leq 2$		$c_n \leq 2$
$c_1 \leq 3$	$\cdots$	$c_n \leq 3$
$\neg(c_1' \leq 2)$		$\neg(c_n' \leq 2)$
$c_1' \leq 3$		$c_n' \leq 3$
$c_1 + c_1' \leq 5$		$c_n + c_n' \leq 5$

$c_1 + c_1' + \cdots + c_n + c_n' > 5n$

BLOCKING CLAUSE

Again, take the satisfying assignment where all the b_i's are set to true (the c_i's $= 2$ and c'_i's $= 3$)

THEORY ATOMS			BLOCKING CLAUSE		
$c_1 \leq 2$		$c_n \leq 2$	$\cancel{c_1 \leq 2}$		$\cancel{c_n \leq 2}$
$c_1 \leq 3$	$\cdots$	$c_n \leq 3$	$\cancel{c_1 \leq 3}$	$\cdots$	$\cancel{c_n \leq 3}$
$\neg(c'_1 \leq 2)$		$\neg(c'_n \leq 2)$	$\cancel{\neg(c'_1 \leq 2)}$		$\cancel{\neg(c'_n \leq 2)}$
$c'_1 \leq 3$		$c'_n \leq 3$	$\cancel{c'_1 \leq 3}$		$\cancel{c'_n \leq 3}$
$c_1 + c'_1 \leq 5$		$c_n + c'_n \leq 5$	$c_1 + c'_1 \leq 5$		$c_n + c'_n \leq 5$
$c_1 + c'_1 + \cdots + c_n + c'_n > 5n$			$c_1 + c'_1 + \cdots + c_n + c'_n > 5n$		

Prunes all the 2^n traces at once.

Solution

- Distinguish “portions” in the program.
- Compute upper bound B_i on WCET for each portion i (recursive call or rougher bound)
- Conjoin these constraints to the previous SMT formula $c_1 + \cdots + c_5 \leq B_1$, $c_6 + \cdots + c_{10} \leq B_2$, etc.
- The obtained formula is **equivalent**
- Do the binary search as before

Solving time from “nonterminating after one night” to “a few seconds”.

Experiments with ARMv7

OTAWA for Basic Block timings
Z3 SMT solver

Cuts : only syntactic criterion

Benchmark name	WCET bounds (#cycles) ILP IPET	SMT	diff	Analysis time (seconds) with cuts	no cuts	#cuts
statemate	3297	3211	2.6%	943.5	$+\infty$	143
nsichneu (1 iteration)	17242	13298	22.7%	6hours	$+\infty$	378
cruise-control	881	873	0.9%	0.1	0.2	13
digital-stopwatch	1012	954	5.7%	0.6	2104.2	53
autopilot	12663	5734	54.7%	1808.8	$+\infty$	498
fly-by-wire	6361	5848	8.0%	10.8	$+\infty$	163
miniflight	17980	14752	18.0%	40.9	$+\infty$	251
tdf	5789	5727	1.0%	13.0	$+\infty$	254

- Malardalen WCET Benchmarks
- Scade designs
- Industrial Code

Conclusion

SMT can be used for static analysis in many ways:

- Improve precision of abstract interpreters (least upper bounds)
- Find program traces that violate some property
- Counter-Example Guided approaches
- Worst-Case Execution Time estimation using optimization

PAGAI static analyzer for LLVM IR, open-source

- Checks array out-of-bounds & integer overflows
- Proves assert statements over numerical variables
- Binaries for Linux/Mac and source code:

`http://pagai.forge.imag.fr`