Counter-Based WCET Analysis

Mihail Asavoae

Aussois — 1 DEC 2014

Thanks to

Remy Boutonnet

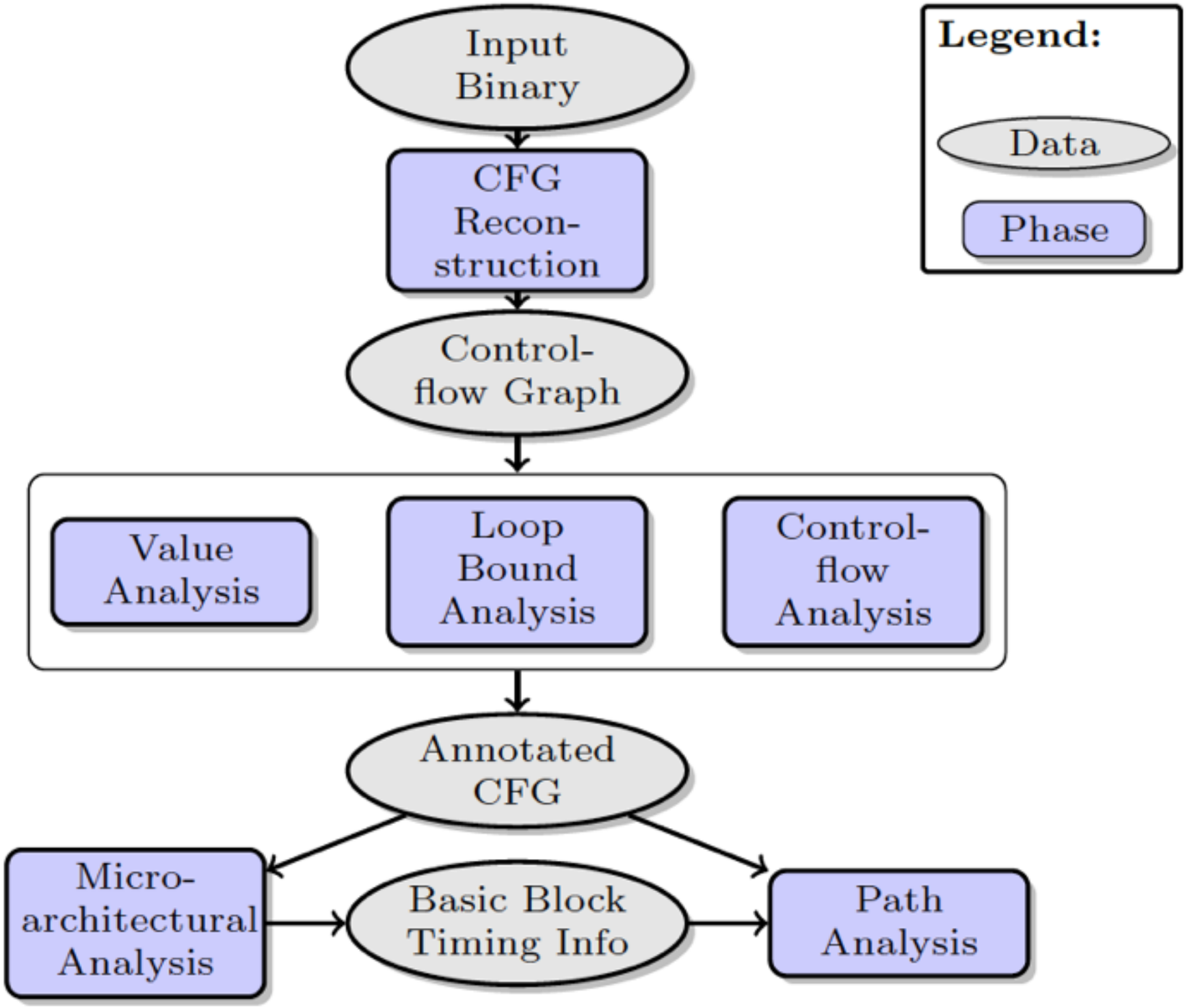
Fabienne Carrier

Nicolas Halbwachs

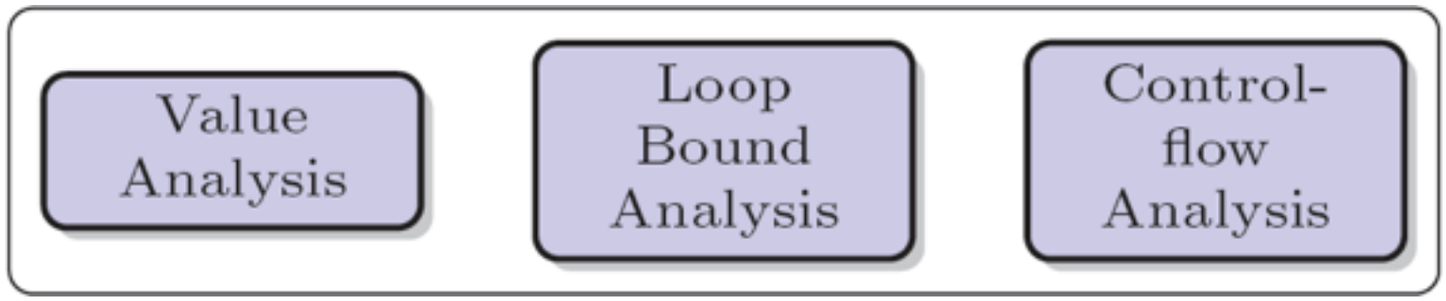
Claire Maiza

Pascal Raymond

Typical Workflow for the WCET Analysis

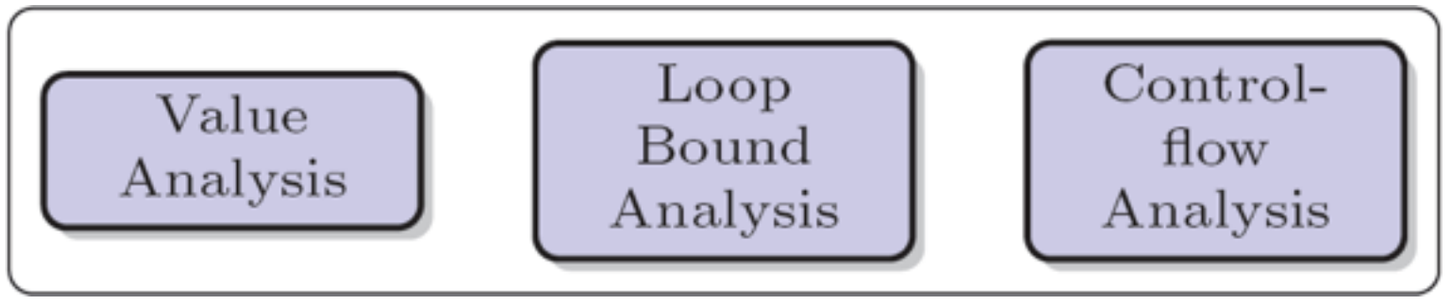


Typical Workflow for the WCET Analysis



W-SEPT Project

Desiderate: to improve the precision and the traceability of semantics information through the compilation flow, from high-level description (such as Lustre and Scade synchronous languages) down to C and binary levels.



Coordinated by Verimag: **Claire Maiza, Pascal Raymond**

Partners: **Verimag**, Grenoble
Inria, Rennes
IRIT, Toulouse
Continental, Toulouse

Counters - Intro (I)

Consider the program

```
x = 0;
while c1 {
      if(x < 10){
            ...
      }
      if(c2){
            ...
            x++;
      }
}
```

Counters - Intro (II)

Consider the program, instrumented with counter variables:

```
x = 0;
while c1 {
    if(x < 10){
        ...
    }
    if(c2){
        ...
        x++;
    }
}
```

```
x = 0; α = 0; β = 0; γ = 0;
while c1 {
    α++;
    if(x < 10){
        ...
        β++;
    }
    if(c2){
        ...
        x = x++; γ++;
    }
}
```

Counters - Intro (III)

Consider the program, instrumented with counter variables:

- an analysis, using the abstract domain of polyhedra, automatically discovers the following invariants, at the end of the program:

$$\gamma = x$$
$$\beta + \gamma \leq \alpha + 10$$
$$\gamma \leq \alpha$$
$$\beta \leq \alpha$$

```
x = 0; α = 0; β = 0; γ = 0;
while c1 {
    α++;
    if(x < 10){
        ...
        β++;
    }
    if(c2){
        ...
        x = x++; γ++;
    }
}
```

Property: at most 10 iterations will execute both « then » branches.

Outline

Consider the program, instrumented with counter variables:

do analyzers use it ?

```
x = 0; α = 0; β = 0; γ = 0;
while c1 {
    α++;
    if(x < 10){
        ...
        β++;
    }
    if(c2){
        ...
        x = x++; γ++;
    }
}
```

Property: at most 10 iterations will execute both « then » branches.

Outline

Consider the program, instrumented with counter variables:

do analyzers use it ?

how many counters ?

```
x = 0; α = 0; β = 0; γ = 0;
while c1 {
    α++;
    if(x < 10){
        ...
        β++;
    }
    if(c2){
        ...
        x = x++; γ++;
    }
}
```

Property: at most 10 iterations will execute both « then » branches.

Outline

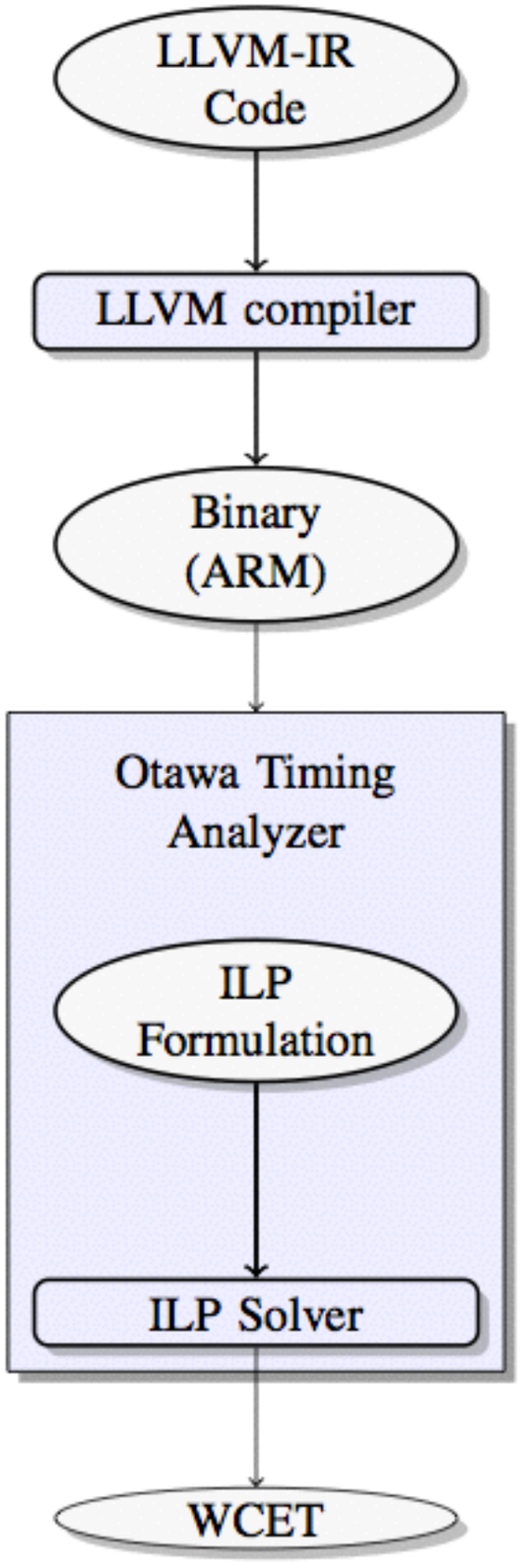
Consider the program, instrumented with counter variables:

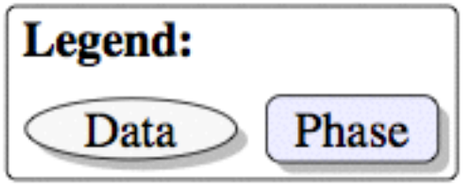
- do analyzers use it ?
- how many counters ?
- are all invariants useful ?
- is the method scalable ?
- what kind of app ?

```
x = 0; α = 0; β = 0; γ = 0;
while c1 {
    α++;
    if(x < 10){
        ...
        β++;
    }
    if(c2){
        ...
        x = x++; γ++;
    }
}
```

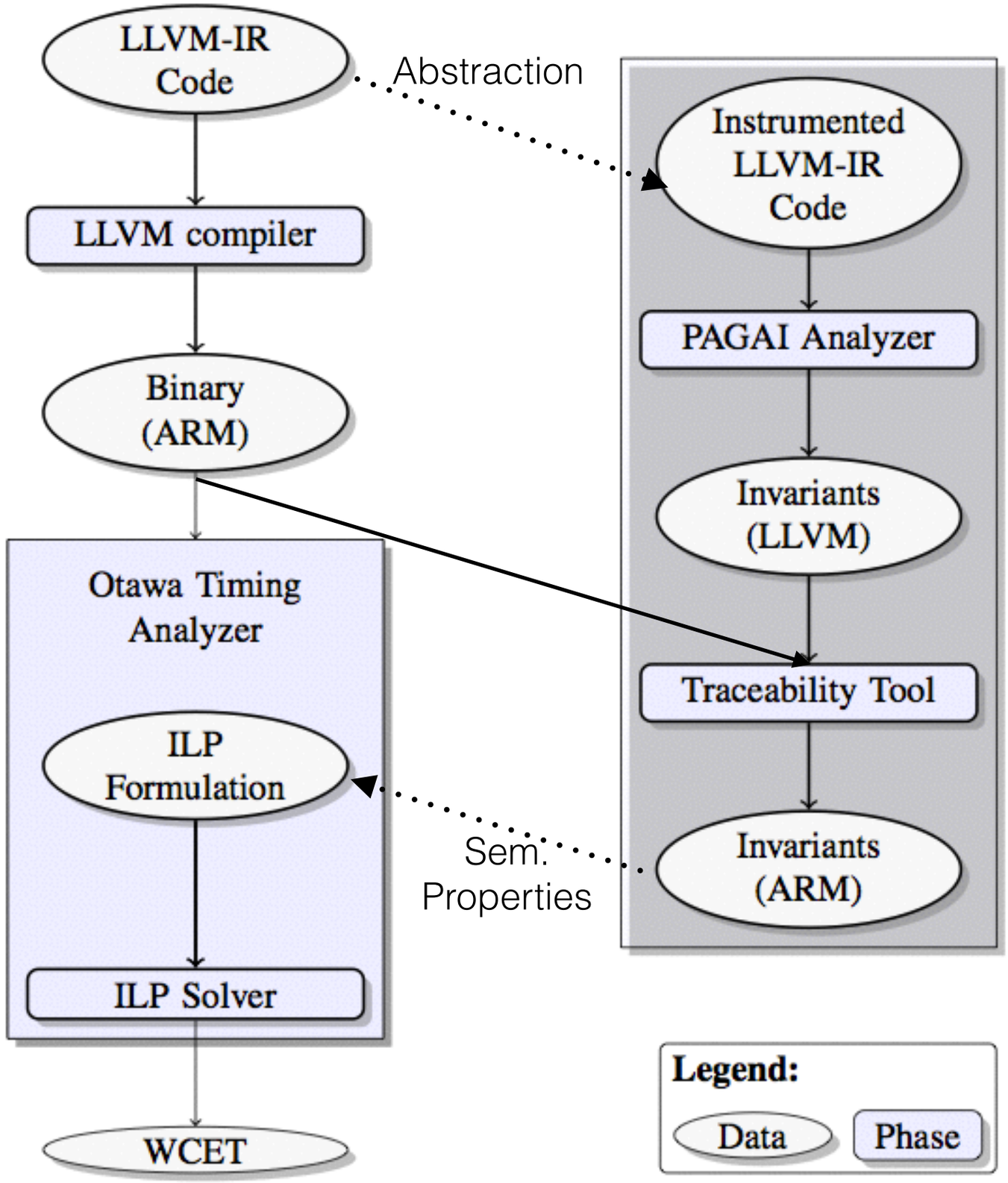
Property: at most 10 iterations will execute both « then » branches.

Workflow on LLVM Infrastructure





Workflow on LLVM Infrastructure



Tools (I)

PAGAI: a path sensitive static analyzer

Julien Henry [1]

Université Joseph Fourier, VERIMAG
Grenoble, France

David Monniaux[1]

CNRS, VERIMAG
Grenoble, France

Matthieu Moy[1]

Grenoble-INP, VERIMAG
Grenoble, France

Abstract

We describe the design and the implementation of PAGAI, a new static analyzer working over the LLVM compiler infrastructure, which computes inductive invariants on the numerical variables of the analyzed program.
PAGAI implements various state-of-the-art algorithms combining abstract interpretation and decision procedures (SMT-solving), focusing on distinction of paths inside the control flow graph while avoiding systematic exponential enumerations. It is parametric in the abstract domain in use, the iteration algorithm, and the decision procedure.
We compared the time and precision of various combinations of analysis algorithms and abstract domains, with extensive experiments both on personal benchmarks and widely available GNU programs.

Keywords: Static Analysis, Program Verification, Abstract Interpretation, Decision Procedure, Satisfiability Modulo Theories.

in TAPAS12

Tools (I)

PAGAI: a path sensitive static analyzer

Julien Henry [1]

Université Joseph Fourier, VERIMAG
Grenoble, France

David Monniaux[1]

CNRS, VERIMAG
Grenoble, France

Matthieu Moy[1]

Grenoble-INP, VERIMAG
Grenoble, France

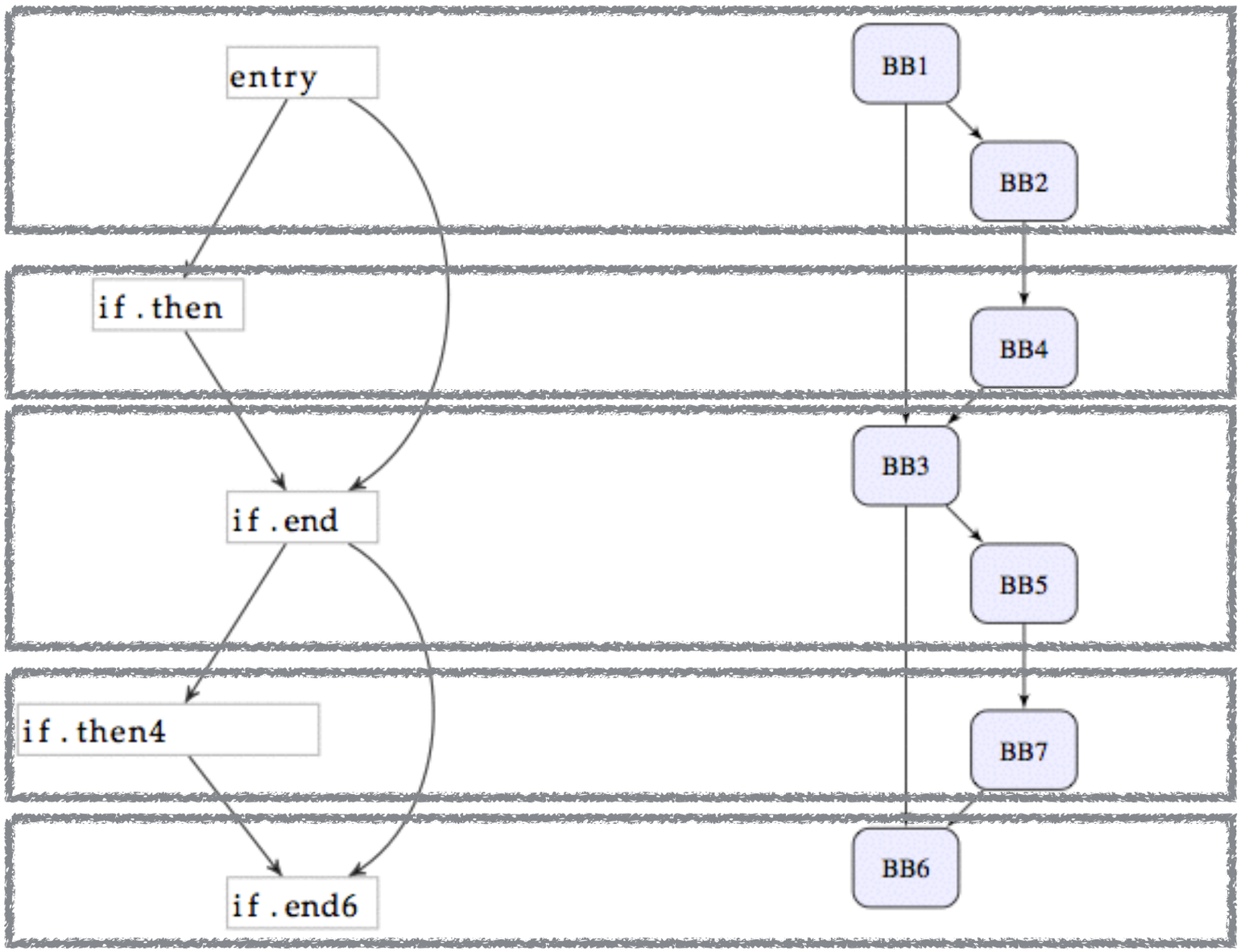
Abstract

We describe the design and the implementation of PAGAI, a new static analyzer working over the LLVM compiler infrastructure, which computes inductive invariants on the numerical variables of the analyzed program.
PAGAI implements various state-of-the-art algorithms combining abstract interpretation and decision procedures (SMT-solving), focusing on abstraction of paths inside the control flow graph while avoiding systematic exponential enumeration. It is parametric in the abstract domain in use, the iteration algorithm, and the decision procedure.
We compared the time and precision of various combinations of analysis algorithms and abstract domains, with extensive experiments both on personal benchmarks and widely available GNU programs.

Keywords: Static Analysis, Program Verification, Abstract Interpretation, Decision Procedure, Satisfiability Modulo Theories.

in TAPAS12

Tools (II)



LLVM-IR CFG **ARM CFG**

Tools (II)

- the **traceability tool** maps LLVM-IR blocks/edges to ARM blocks/edges

- in the most general case, it is a many-to-many mapping, because:
  - *of the traceability* of the LLVM compiler, the code generator produces ARM blocks which are not in single-input single-output form
  - *of the Static Single Assignment (SSA) form* used by the LLVM compiler

- *Some precision could be lost because of complicated matching*

WCET Analyzers

based on IPET

Chronos

SWEET
Swedish Execution Time Analysis Tool.

OTAWA

AbsInt
Angewandte Informatik

WCET Analyzers

based on IPET

Chronos

SWEET
Swedish Execution Time Analysis Tool.

pattern matching

OTAWA

AbsInt
Angewandte Informatik

WCET Analyzers

based on IPET

Chronos

SWEET
Swedish Execution Time Analysis Tool.

pattern matching

interval analysis

OTAWA

AbsInt
Angewandte Informatik

WCET Analyzers

based on IPET

Chronos

X


```
y = 2 ;
x = y ;
if (x > 3) { …
```

WCET Analyzers

based on IPET

Chronos

X

SWEET
Swedish Execution Time Analysis Tool.

```
while (x < 10) {
...
if (c1)... // modified x
...
x++
```

OTAWA

X

AbsInt
Angewandte Informatik

X

WCET Analyzers

based on IPET

Chronos

X

```
x = 0;
while c1 {
    if(x < 10){
        ...
    }
    if(c2){
        ...
        x++;
    }
}
```

SWEET
Swedish Execution Time Analysis Tool.

X

OTAWA

X

AbsInt
Angewandte Informatik

X

On Counter Variables

General observations:

- two kinds of counters: those initialized one time or those resettable

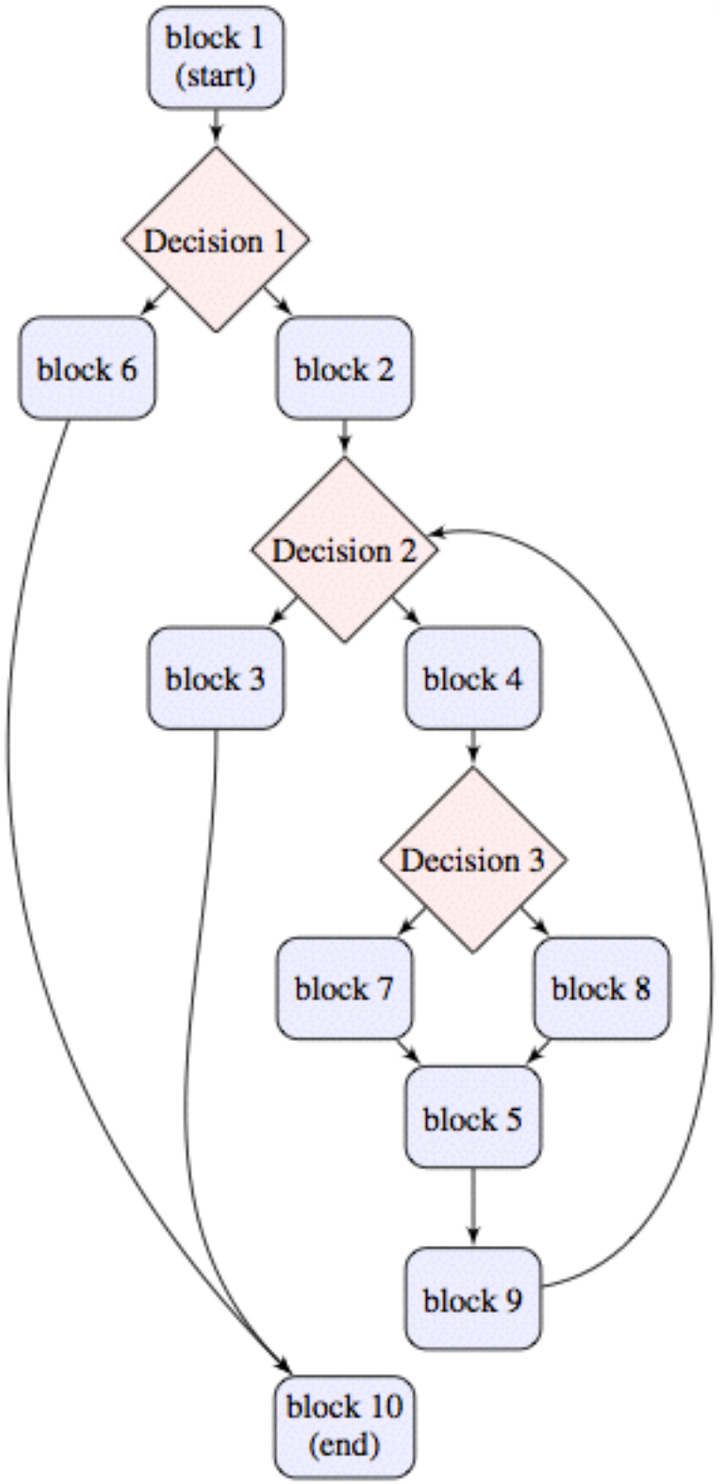
- an exhaustive placement of counters does not scale well, hence it is necessary:

- to design an **optimal counter-placement procedure**, to insert a small number of counters at meaningful locations

- to apply on code fragments (i.e. functions)

Counter-placement (I)

Heuristics - combination of syntactic and semantic criteria

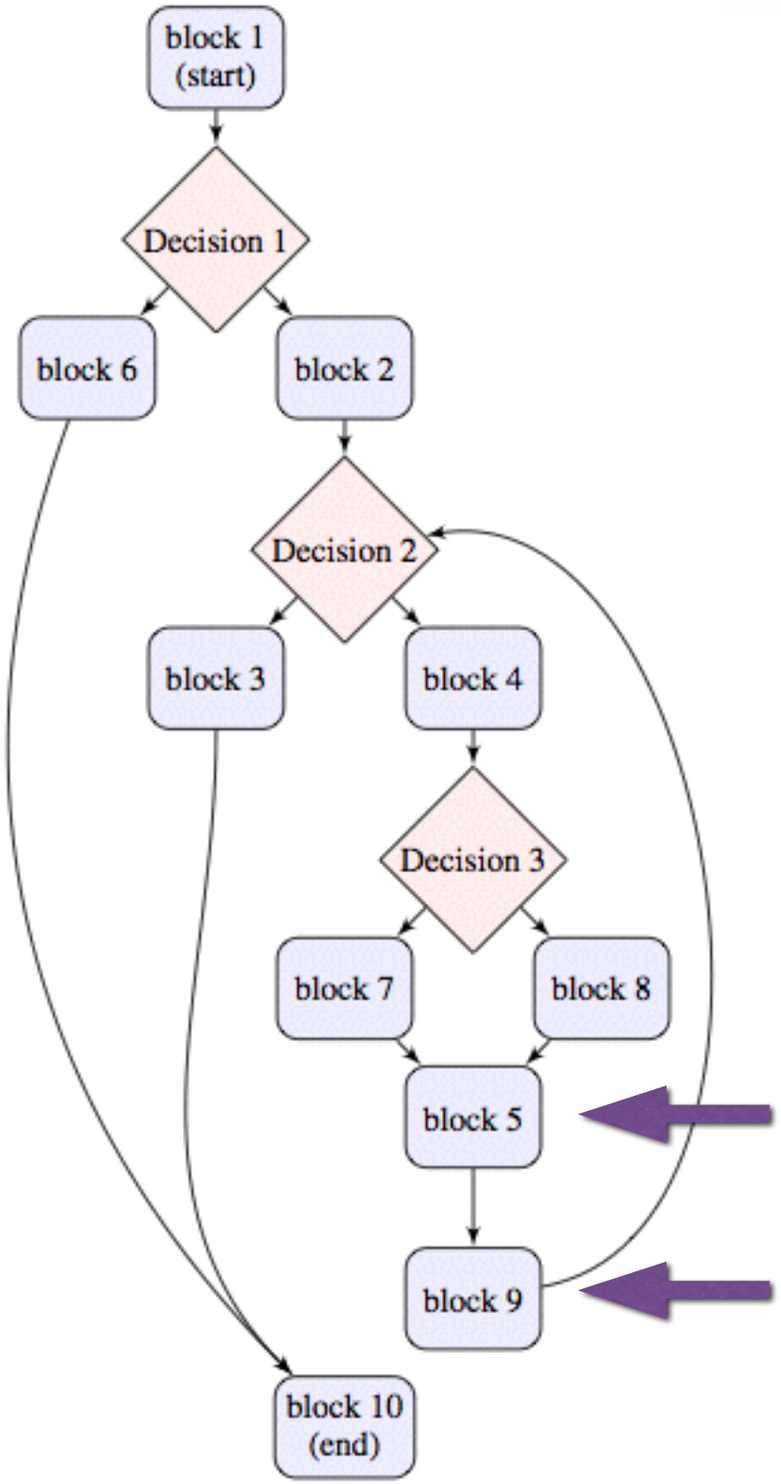


Counter-placement (II)

Heuristics - combination of syntactic and semantic criteria

Syntactic:

- same scope level



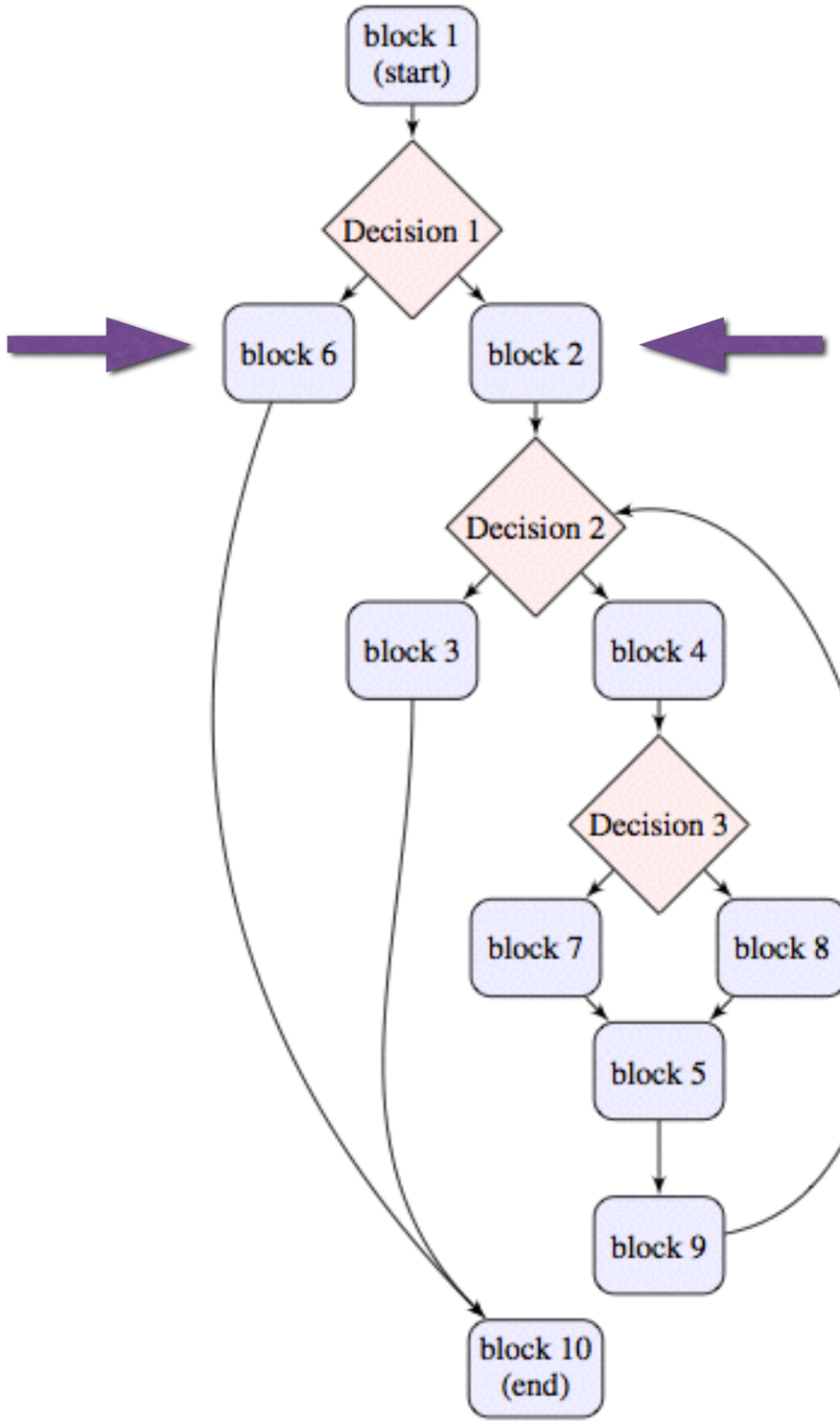
Counter-placement (III)

Heuristics - combination of syntactic and semantic criteria

Syntactic:

- same scope level

- unbalanced branches



Counter-placement (IV)

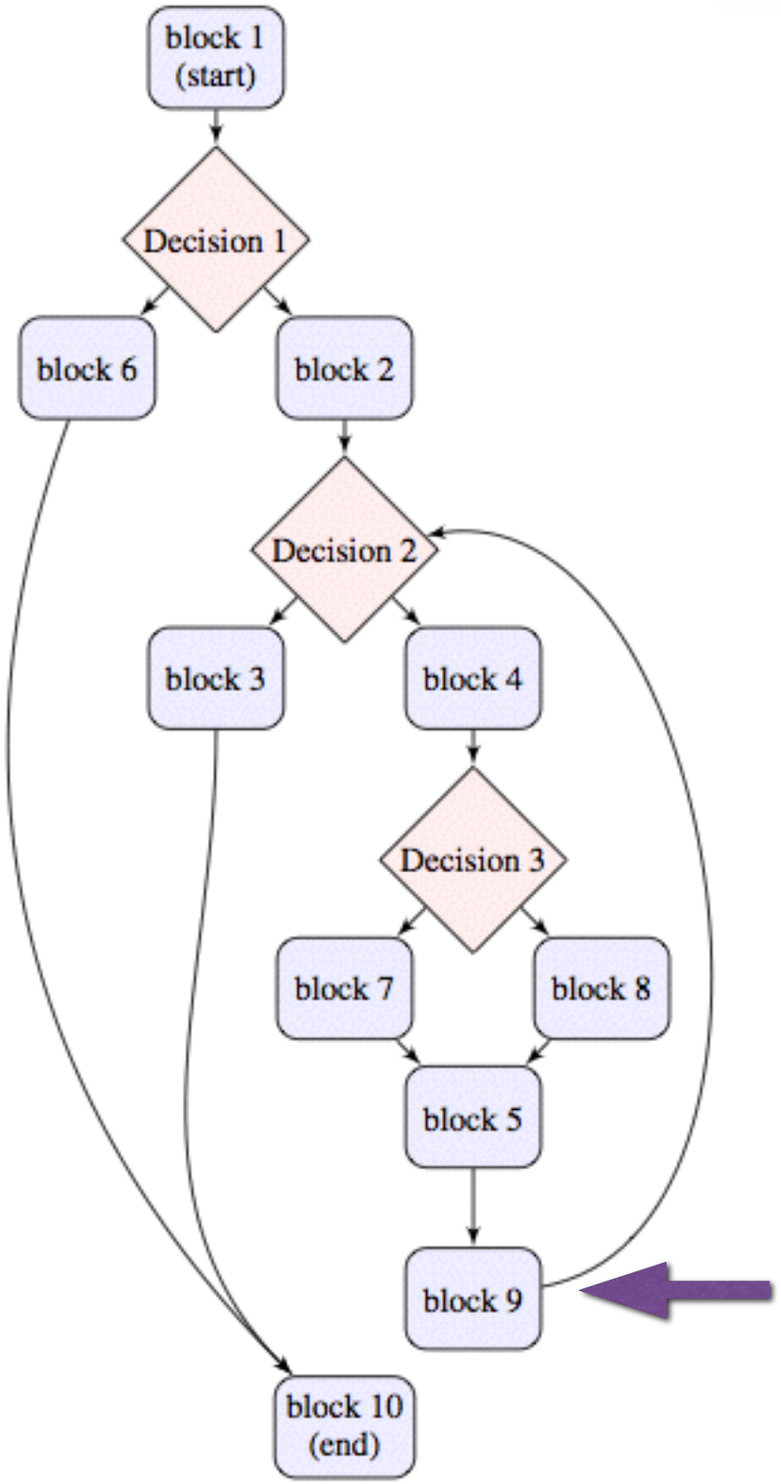
Heuristics - combination of syntactic and semantic criteria

Syntactic:

- same scope level

- unbalanced branches

- back-edges for loops

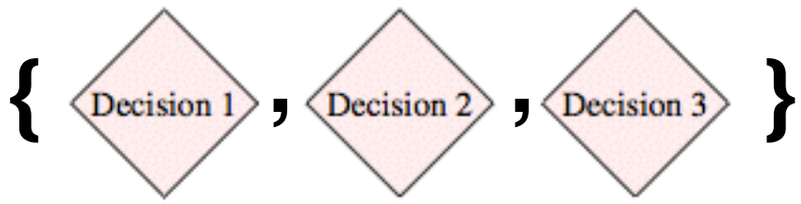


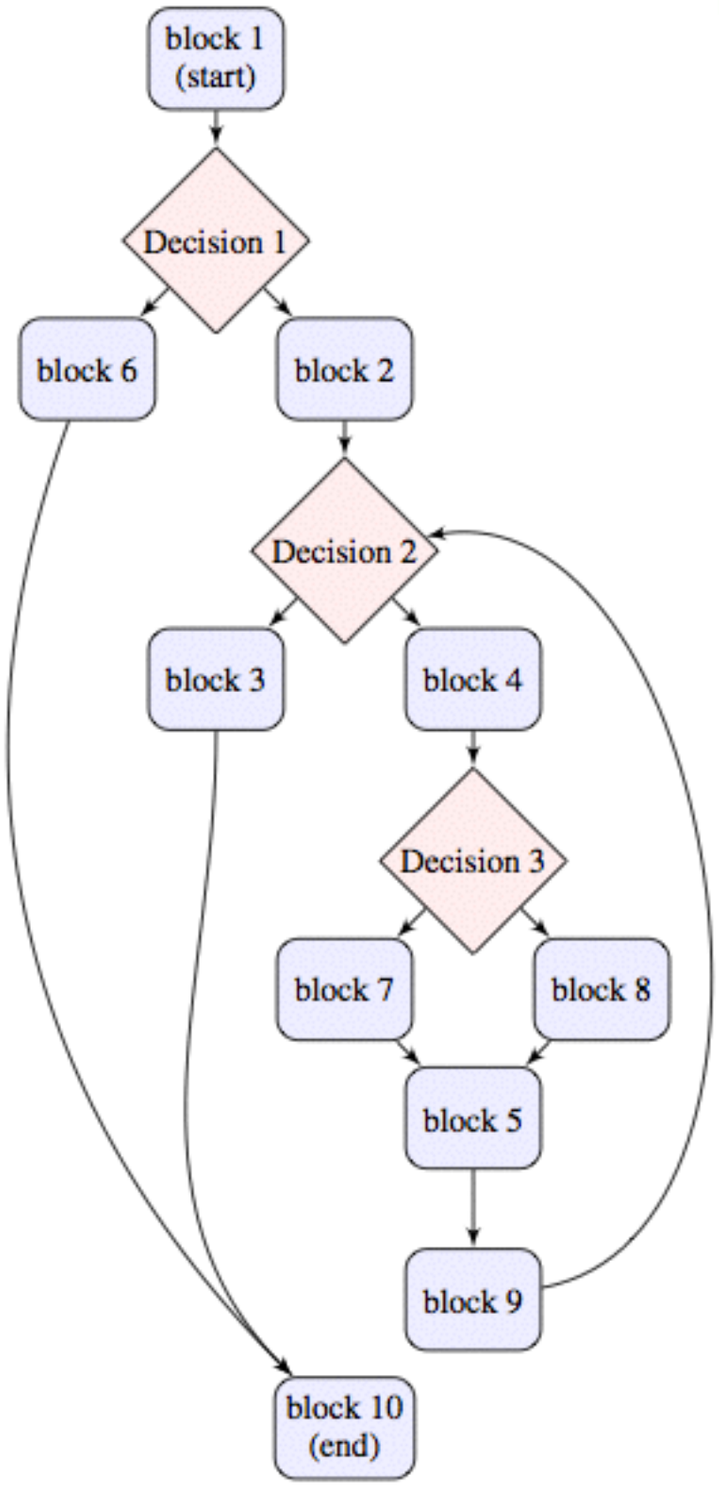
Counter-placement (V)

Heuristics - combination of syntactic and semantic criteria

Semantic:

- program slicing based on a set of variables of interest





Example (I)

```
for (i = 0; i < 10; i++) {
  s1 = s1 + t[i];
  x = i + 1;
  if (!(i < max_i)) break;
}
int s2 = s1;
if (x < 6)
  for (i=0; i < 10; i++) {
    s2 = s2 + 2;
    if (!(i < 5)) break;
  }
int f = 0;
if (2*s1 < s2)
  for (i = 0; i < 10; i++) {
    f = f + 2;
    if (!(i < 4)) break;
  }
```

intended for 4 iterations

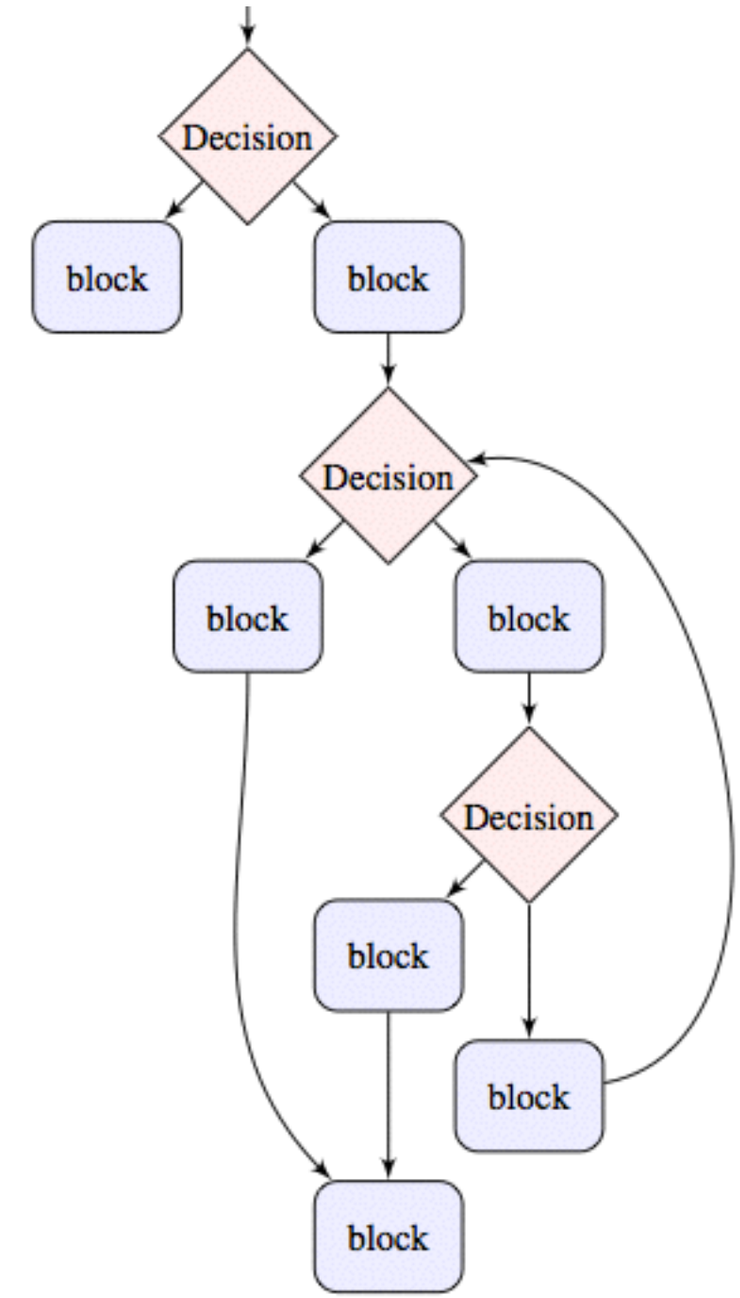
intended for 5 iterations

dead code

array t - elements > 5; max_i is 3

Example (II)

```
for (i = 0; i < 10; i++) {
  s1 = s1 + t[i];
  x = i + 1;
  if (!(i < max_i)) break;
}
int s2 = s1;
if (x < 6)
  for (i=0; i < 10; i++) {
    s2 = s2 + 2;
    if (!(i < 5)) break;
  }
int f = 0;
if (2*s1 < s2)
  for (i = 0; i < 10; i++) {
    f = f + 2;
    if (!(i < 4)) break;
  }
```



array t - elements > 5; max_i is 3

Example (III)

```
for (i = 0; i < 10; i++) {
  s1 = s1 + t[i];
  x = i + 1;
  if (!(i < max_i)) break;
}
int s2 = s1;
if (x < 6)
  for (i=0; i < 10; i++) {
    s2 = s2 + 2;
    if (!(i < 5)) break;
  }
int f = 0;
if (2*s1 < s2)
  for (i = 0; i < 10; i++) {
    f = f + 2;
    if (!(i < 4)) break;
  }
```

```
ILP constraints:

. . .
-x20_main+ x16_main= 0;
-x20_main+ x23_main+ x21_main= 0;
-x23_main+ x10_main= 0;
-5*x15_main+ x24_main= 0;
-5*x15_main+ x19_main= 0;
-4*x15_main+ x25_main= 0;
-x15_main+ x26_main= 0;
1-x3_main+ x9_main= 0;
-x3_main+ x8_main= 0;
-1+x13_main= 0;
1-x23_main>= 0;
1-x15_main>= 0;
-1+x23_main+ x15_main>= 0;
. . .
```

Initial : 26 counters

Example (IV)

```
for (i = 0; i < 10; i++) {
  s1 = s1 + t[i];
  x = i + 1;
  if (!(i < max_i)) break;
}
int s2 = s1;
if (x < 6)
  for (i=0; i < 10; i++) {
    s2 = s2 + 2;
    if (!(i < 5)) break;
  }
int f = 0;
if (2*s1 < s2)
  for (i = 0; i < 10; i++) {
    f = f + 2;
    if (!(i < 4)) break;
  }
```

ILP constraints:

```
. . .
-x20_main+ x16_main= 0;
-x20_main+ x23_main+ x21_main= 0;
-x23_main+ x10_main= 0;
-5*x15_main+ x24_main= 0;
-5*x15_main+ x19_main= 0;
-4*x15_main+ x25_main= 0;
-x15_main+ x26_main= 0;
1-x3_main+ x9_main= 0;
-x3_main+ x8_main= 0;
-1+x13_main= 0;
1-x23_main>= 0;
1-x15_main>= 0;
-1+x23_main+ x15_main>= 0;
. . .
```

Placement algorithm : 10 counters

Example - Tools

```
for (i = 0; i < 10; i++) {
  s1 = s1 + t[i];
  x = i + 1;
  if (!(i < max_i)) break;
}
int s2 = s1;
if (x < 6)
  for (i=0; i < 10; i++) {
    s2 = s2 + 2;
    if (!(i < 5)) break;
  }
int f = 0;
if (2*s1 < s2)
  for (i = 0; i < 10; i++) {
    f = f + 2;
    if (!(i < 4)) break;
  }
```

ILP constraints:

```
. . .
-x20_main+ x16_main= 0;
-x20_main+ x23_main+ x21_main= 0;
-x23_main+ x10_main= 0;
-5*x15_main+ x24_main= 0;
-5*x15_main+ x19_main= 0;
-4*x15_main+ x25_main= 0;
-x15_main+ x26_main= 0;
1-x3_main+ x9_main= 0;
-x3_main+ x8_main= 0;
-1+x13_main= 0;
1-x23_main>= 0;
1-x15_main>= 0;
-1+x23_main+ x15_main>= 0;
. . .
```

Chronos - requires manual annotations of loop bounds

Example - Tools

```
for (i = 0; i < 10; i++) {
  s1 = s1 + t[i];
  x = i + 1;
  if (!(i < max_i)) break;
}
int s2 = s1;
if (x < 6)
  for (i=0; i < 10; i++) {
    s2 = s2 + 2;
    if (!(i < 5)) break;
  }
int f = 0;
if (2*s1 < s2)
  for (i = 0; i < 10; i++) {
    f = f + 2;
    if (!(i < 4)) break;
  }
```

ILP constraints:

. . .
-x20_main+ x16_main= 0;
-x20_main+ x23_main+ x21_main= 0;
-x23_main+ x10_main= 0;
-5*x15_main+ x24_main= 0;
-5*x15_main+ x19_main= 0;
-4*x15_main+ x25_main= 0;
-x15_main+ x26_main= 0;
1-x3_main+ x9_main= 0;
-x3_main+ x8_main= 0;
-1+x13_main= 0;
1-x23_main>= 0;
1-x15_main>= 0;
-1+x23_main+ x15_main>= 0;
. . .

- finds all loop bounds = 10

Example - Tools

```
for (i = 0; i < 10; i++) {
  s1 = s1 + t[i];
  x = i + 1;
  if (!(i < max_i)) break;
}
int s2 = s1;
if (x < 6)
  for (i=0; i < 10; i++) {
    s2 = s2 + 2;
    if (!(i < 5)) break;
  }
int f = 0;
if (2*s1 < s2)
  for (i = 0; i < 10; i++) {
    f = f + 2;
    if (!(i < 4)) break;
  }
```

ILP constraints:

```
. . .
-x20_main+ x16_main= 0;
-x20_main+ x23_main+ x21_main= 0;
-x23_main+ x10_main= 0;
-5*x15_main+ x24_main= 0;
-5*x15_main+ x19_main= 0;
-4*x15_main+ x25_main= 0;
-x15_main+ x26_main= 0;
1-x3_main+ x9_main= 0;
-x3_main+ x8_main= 0;
-1+x13_main= 0;
1-x23_main>= 0;
1-x15_main>= 0;
-1+x23_main+ x15_main>= 0;
. . .
```

SWEET
Swedish Execution Time Analysis Tool.

- finds accurate bounds for first two loops

Example - Tools

```
for (i = 0; i < 10; i++) {
  s1 = s1 + t[i];
  x = i + 1;
  if (!(i < max_i)) break;
}
int s2 = s1;
if (x < 6)
  for (i=0; i < 10; i++) {
    s2 = s2 + 2;
    if (!(i < 5)) break;
  }
int f = 0;
if (2*s1 < s2)
  for (i = 0; i < 10; i++) {
    f = f + 2;
    if (!(i < 4)) break;
  }
```

ILP constraints:

. . .
-x20_main+ x16_main= 0;
-x20_main+ x23_main+ x21_main= 0;
-x23_main+ x10_main= 0;
-5*x15_main+ x24_main= 0;
-5*x15_main+ x19_main= 0;
-4*x15_main+ x25_main= 0;
-x15_main+ x26_main= 0;
1-x3_main+ x9_main= 0;
-x3_main+ x8_main= 0;
-1+x13_main= 0;
1-x23_main>= 0;
1-x15_main>= 0;
-1+x23_main+ x15_main>= 0;
. . .

AbsInt Angewandte Informatik

- finds accurate bounds and detect dead code

Experiments

Benchmark	LOC	#Cntr	#Inv	WCET init	WCET fin.	Improv.
selector	134	14	14	1112	528	52.6%
roll-control	234	25	19	501	501	0 %
cruise-control	234	35	31	881	852	3.3%
even	82	9	8	2807	2210	21.3%

- **loop bounds**
- **path infeasibility:** relations like $x_{b1} + x_{b2} \leq 1$ (for pairwise exclusive branches), $x_b = 0$ (for dead code) and $x_{b1} + x_{b2} \leq 5 + x_{b3}$
- **structural**

Conclusions

- proposed a technique to exploit the program semantics for the WCET analysis

- easy to use (i.e. plugin) for a typical IPET workflows

- experimented with on Otawa timing analyzer

Thank You!