Approximation of the Worst-Case Execution Time Using Structural Analysis

Matteo Corti and Thomas Gross

ETH Zürich

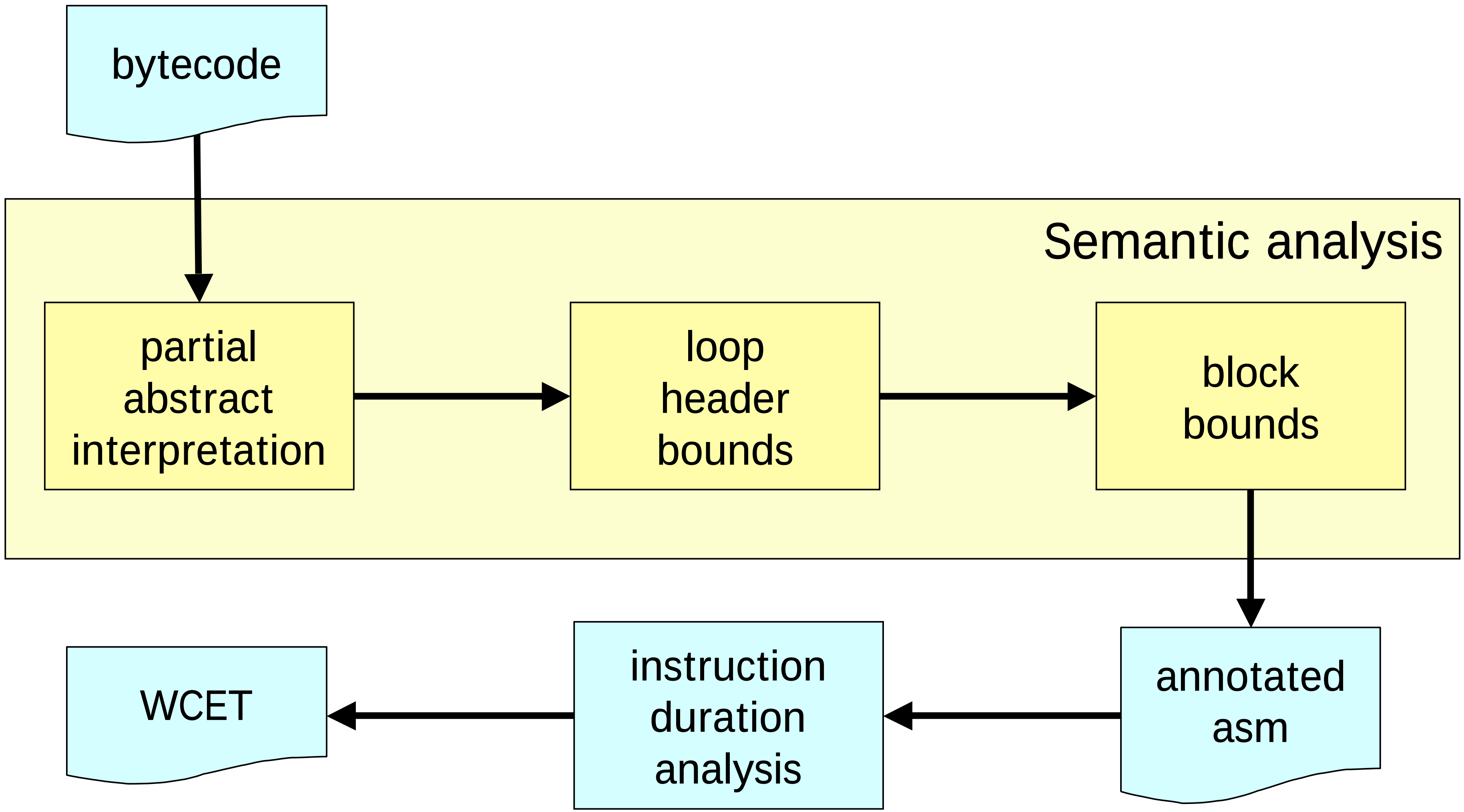
Goal

- Worst-case execution time estimation of soft-real time Java applications.
- We focus on semantic analysis:
  - compute a tight bound on the max and min number of iterations for every block
  - consider different path frequencies inside loops
  - avoid path enumeration

Outline

- Goal
- Loop bounds
- Block bounds
- Complexity and related work
- Testing environment
- Results
- Concluding remarks

System's overview

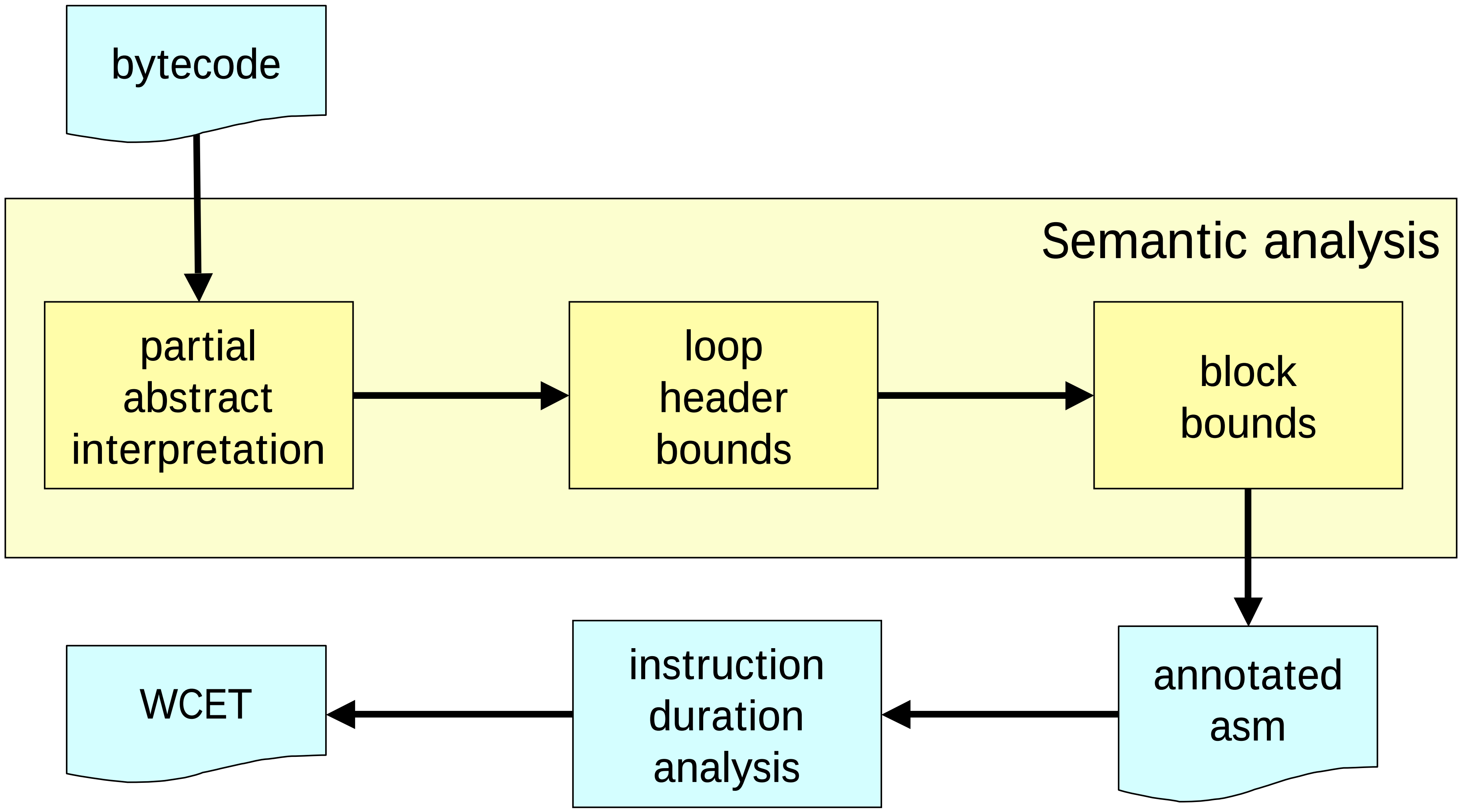


Java

- Whole program analysis.
- Variable type based analysis to resolve polymorphism.
- We consider only local integer variables for the loop analysis.

- **Our block iterations bounding technique is language independent.**

System's overview



Partial abstract interpretation

- We perform a limited abstract interpretation pass over **linear code**.
- We discover some false paths (not containing cycles).
- We gather information on possible variables' values.

```
void foo(int i) {
  if (i > 0) {

    for(;i<10;i++) {
      bar();
    }
  }
}
```

$i \in [1..\infty[$

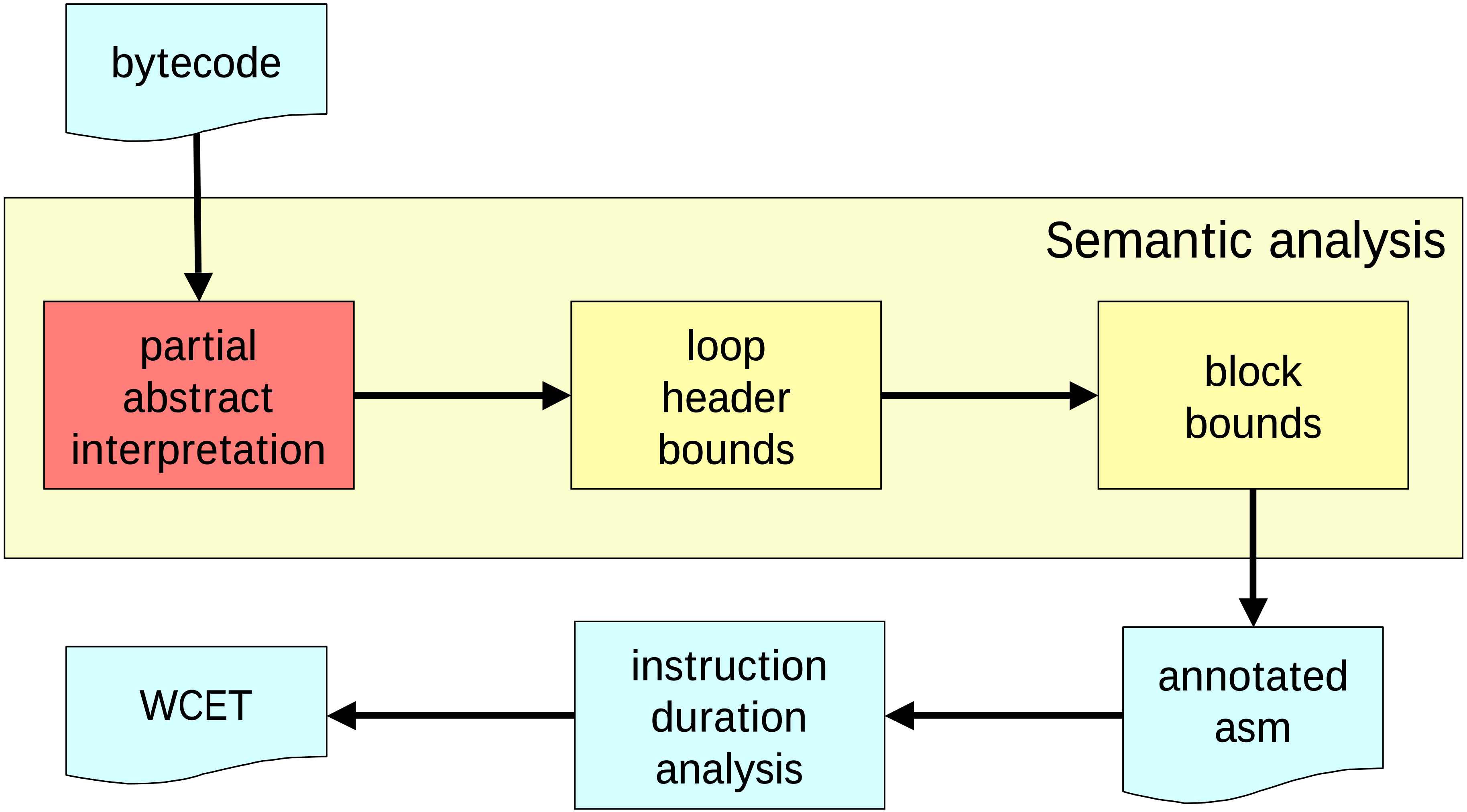
Partial abstract interpretation

Benchmark	Infeasible paths
_201_compress	2
_202_jess	3
_205_raytrace	7
_209_db	2
_213_javac	240
_222_mpegaudio	19
_228_jack	22

Partial abstract interpretation

Benchmark	Infeasible longest path
JavaLayer	2
linpack	2
whetstone	1

System's overview



Loop bounds

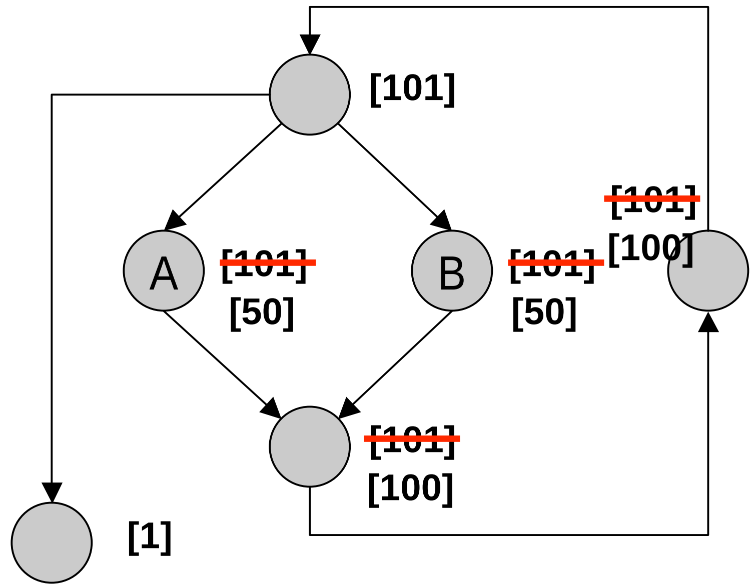
- **Bounds on the loop header computed similarly to C. Healy [RTAS'98].**
- We introduce noncontiguous sets of integers to easily handle equality operators.

- *Iteration branch*: a block where the conditional jump could be responsible for a loop exit.
- For each edge *e* and iteration branch *ib* we compute the possible number of iterations.

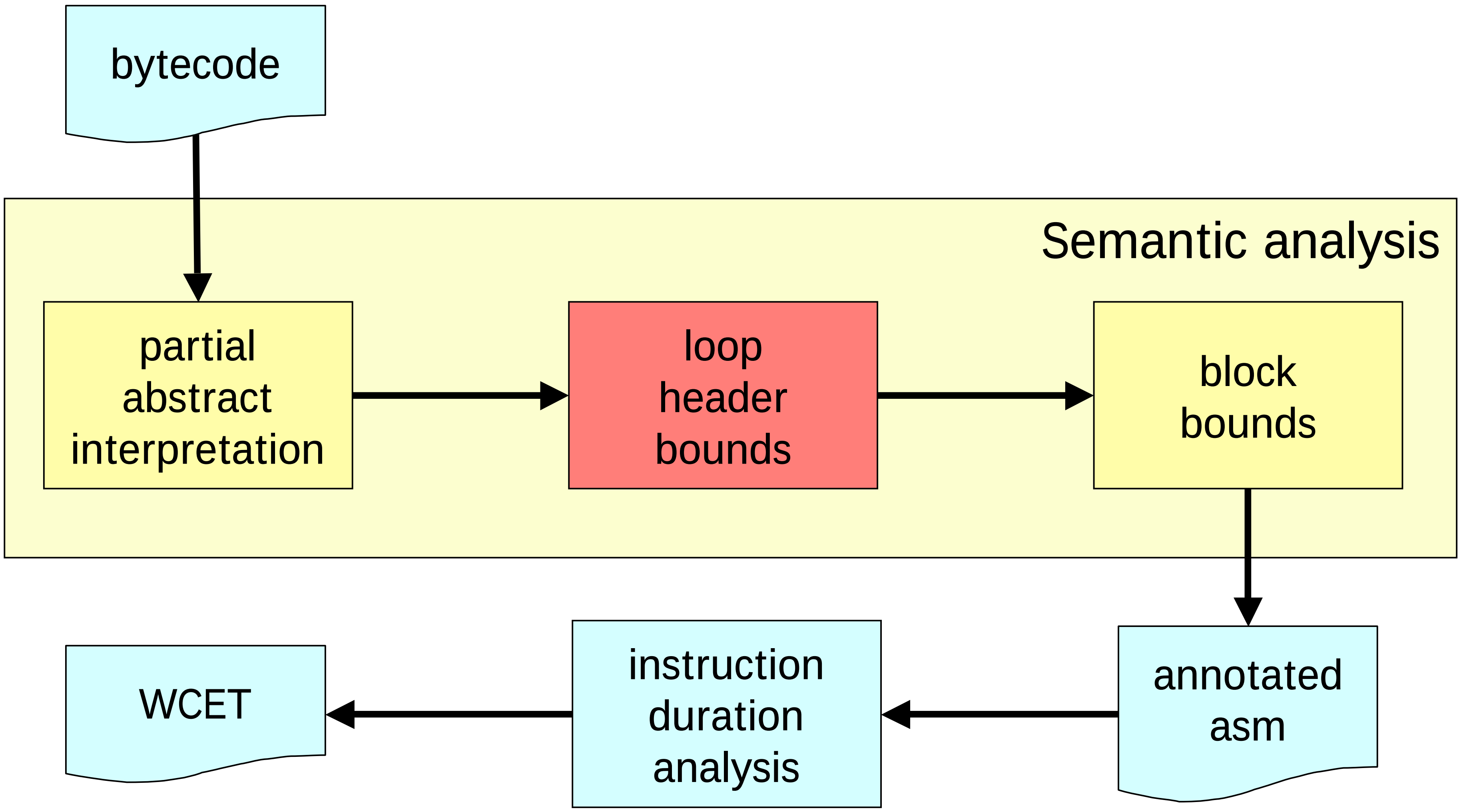
Loop bounds

- The bounds on the iterations of the header are safe for the whole loop.
- But: some parts of the loop could be executed less frequently:

```
for(int i=0; i<100; i++) {
  if (i < 50) {
    A;
  } else {
    B;
  }
}
```

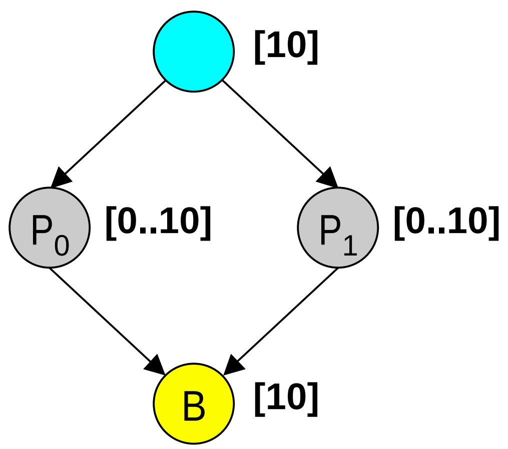


System's overview



Basic block iterations

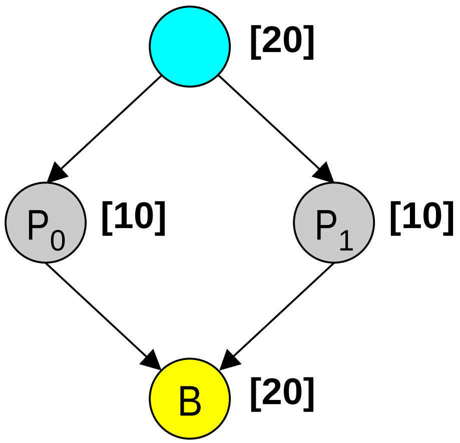
- The number of iterations of a block is **not** a local property (based on immediate predecessors).




```
void foo(boolean b) {
  for(int i=0; i<10; i++) {
    if (b) {
      P0;
    } else {
      P1;
    }
    B;
  }
}
```

Basic block iterations

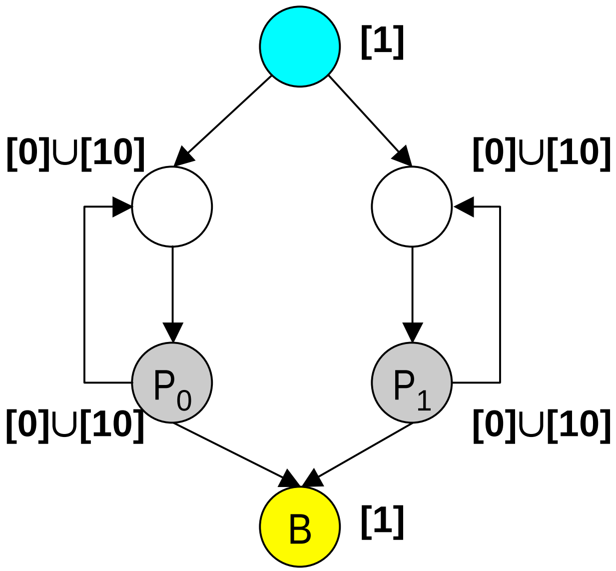
- The number of iterations of a block is **not** a local property (based on immediate predecessors).




```
void foo(boolean b) {
  for(int i=0; i<20; i++) {
    if (i<10) {
      P0;
    } else {
      P1;
    }
    B;
  }
}
```

Basic block iterations

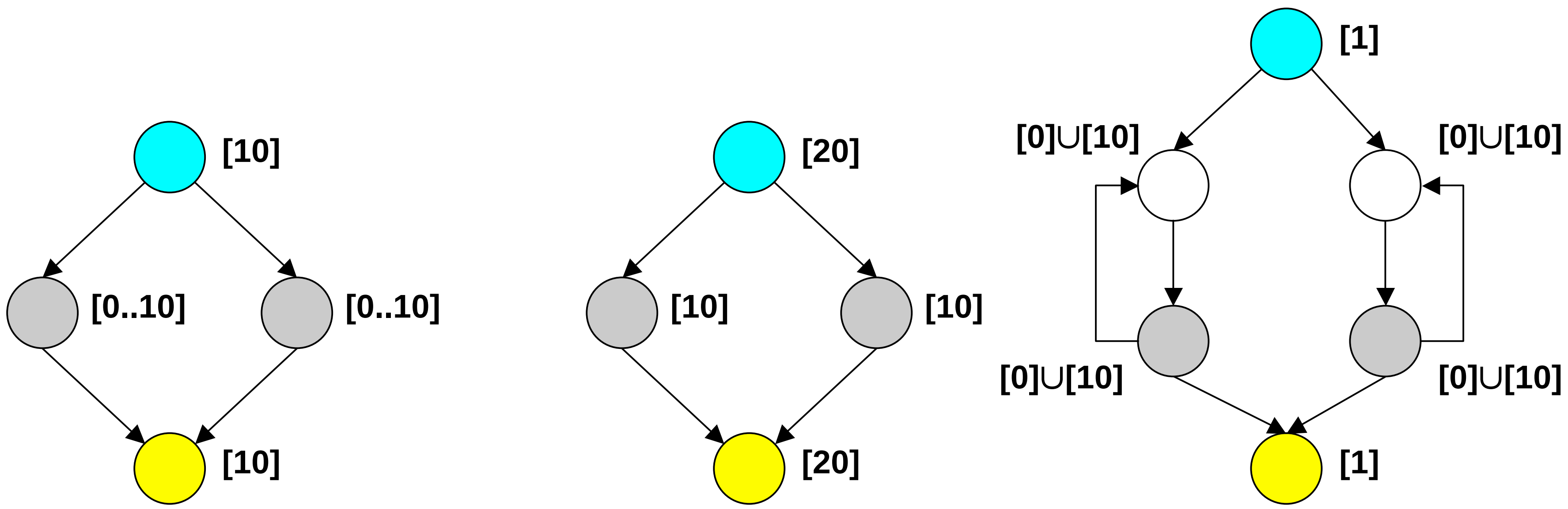
- The number of iterations of a block is **not** a local property (based on immediate predecessors).




```
void foo(boolean b) {
  int i = 0;
  if (b) {
    do {
      i++; P0;
    } while (i<10);
  } else {
    do {
      i++; P1;
    } while (i<10);
  }
  B;
}
```

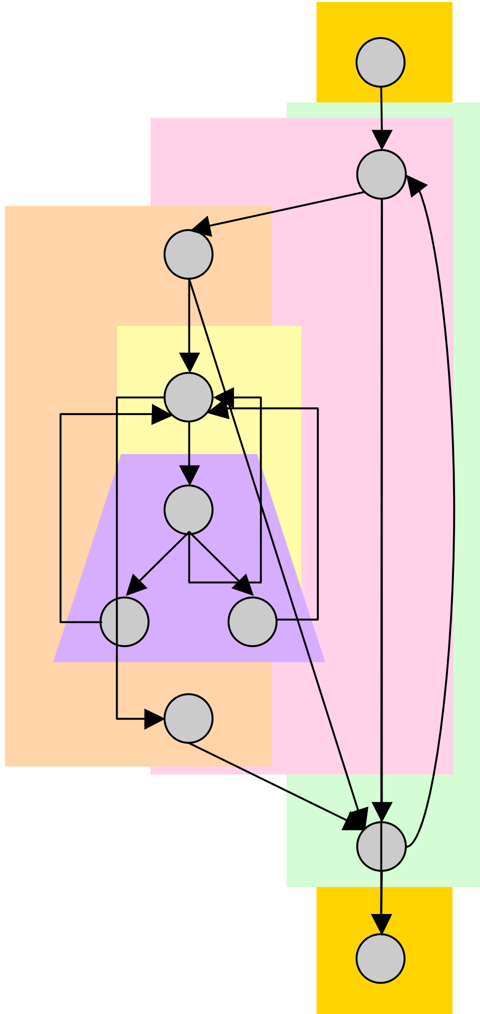
Basic block iterations

- The number of iterations of a block is **not** a local property (based on immediate predecessors).



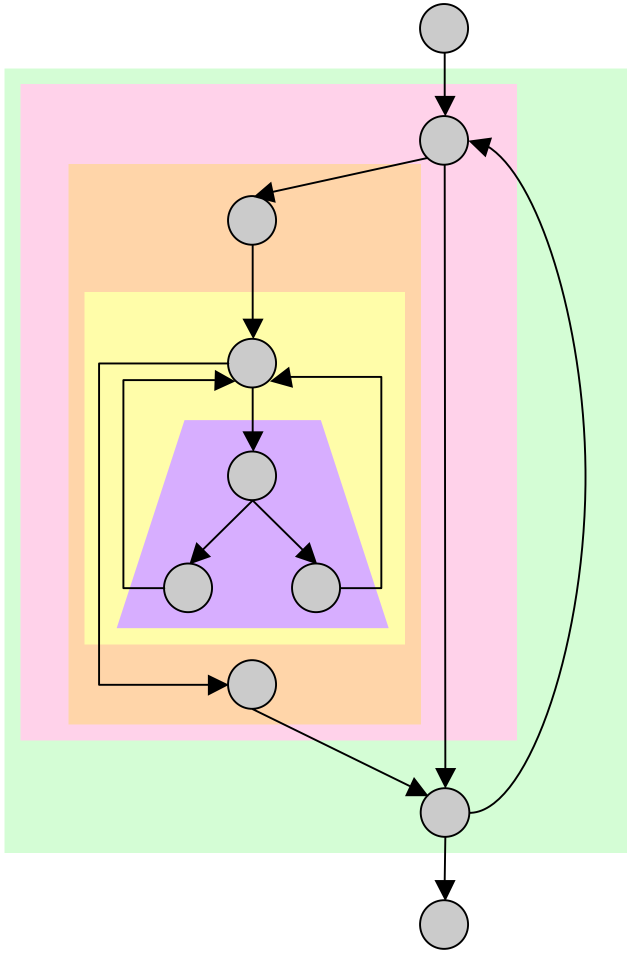
Structural analysis

- Powerful interval analysis.
- Recognizes semantic constructs.
- Useful when the source code is not available.
- **Iteratively matches the blocks with predefined patterns.**



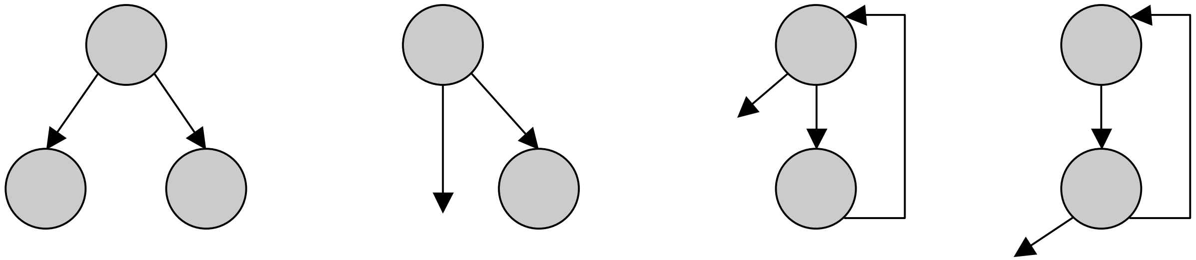
Structural analysis

- Powerful interval analysis.
- Recognizes semantic constructs.
- Useful when the source code is not available.
- **Iteratively matches the blocks with predefined patterns.**



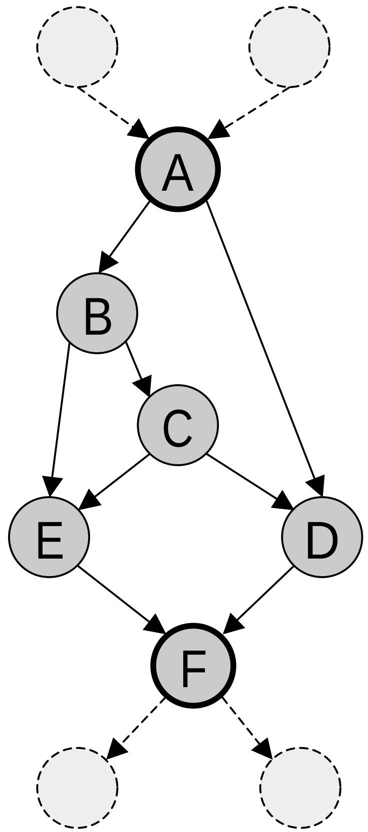
Structural analysis

Static patterns:



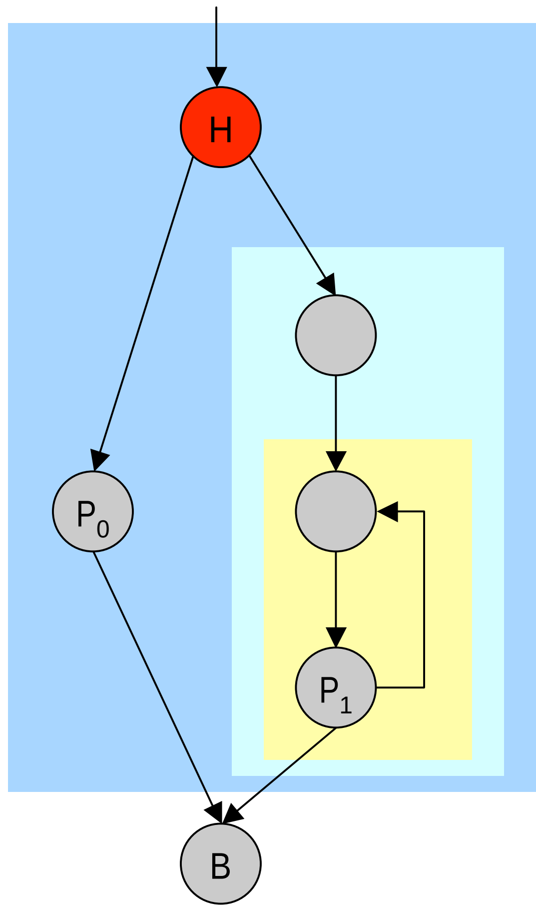
Dynamic patterns:

```
if (A || (B && C)) {
  D;
} else {
  E;
}
F;
```



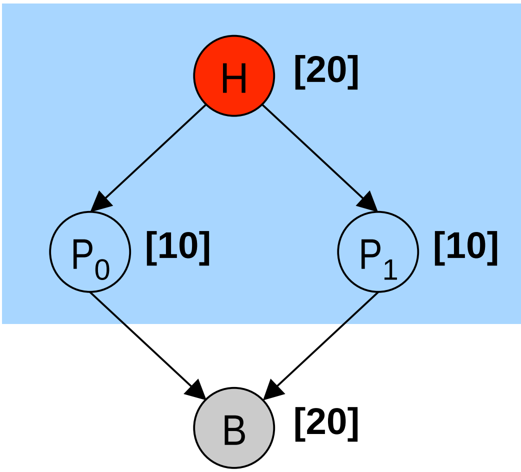
Block iterations

- Block iterations are computed using the CFG root and the iteration branches.
- The header and the type of the biggest semantic region that includes all the predecessors of a node determine its number of iterations.



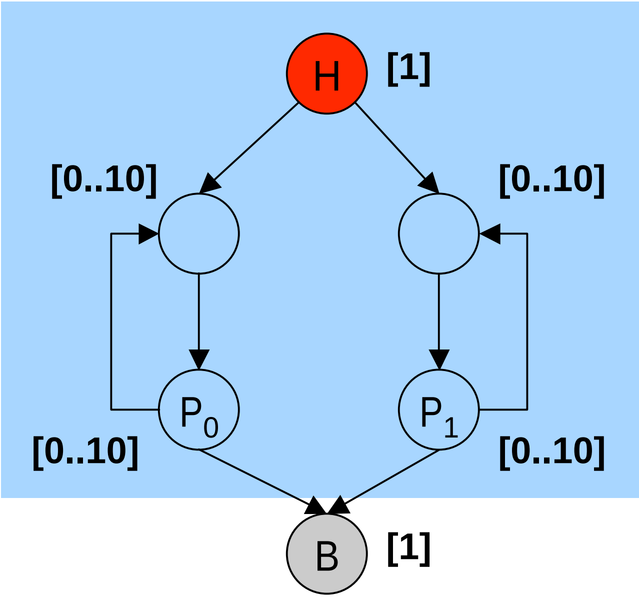
- Complete algorithm in the paper.

Example



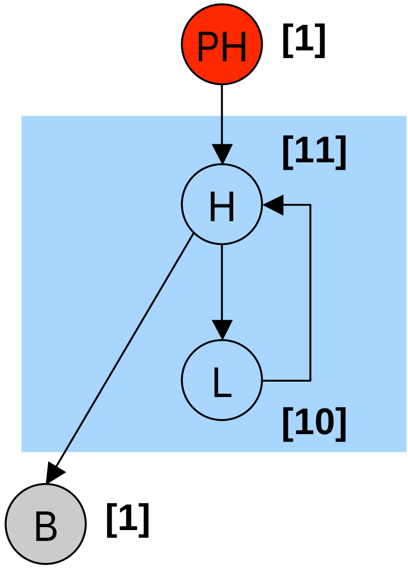

```
void foo(boolean b) {
  for(int i=0; i<20; i++) {
    if (i<10) {
      P0;
    } else {
      P1;
    }
    B;
  }
}
```

Example




```
void foo(boolean b) {
  int i = 0;
  if (b) {
    do {
      i++; P0;
    } while (i<10);
  } else {
    do {
      i++; P1;
    } while (i<10);
  }
  B;
}
```

Example




```
void foo(boolean b) {
  PH;
  for(int i=0; i<10; i++) {
    L;
  }
  B;
}
```

Related work

- Automatically detected value-dependent constraints [Healy, RTAS'99]:
  - per block bounds
  - requires path enumeration (in the loop body)

- We propagate the header bounds to the blocks in quadratic time:
  - Structural analysis: $O(B^2)$
  - Loop bounds: O(B)
  - **Block bounds: O(B)**

Evaluation: hardware-level analysis

- The semantic analysis is platform independent.
- Evaluation: Pentium III on Linux.
- We approximate the effects of caches and pipelines:
  - we assume that the effects of an instruction fade over time.
  - caches and pipelines are analyzed locally.
- Possible sources of inaccuracies:
  - cache misses and pipeline stalls
  - **but not** the number of iterations of an instruction (conservative)

Evaluation

- **Base**: the bounds on the iterations of the loop header are used for the whole loop.
- **Enhanced**: structural analysis is used to consider different path frequencies in loop bodies.

Results: synthetic benchmarks

Example 1

```
for (i=0; i<10000; i++) {
  if (i<5000) {
B1  array[i] = -array[i];
  }
  if (array[i] > max) {
B2  max = array[i];
  }
}
```

Example 2

```
for(i=0; i<10; i++) {
  for (j=0; j<10; j++) {
    if(j<9) {
B3    m[i][j] *= m[i][j];
    } else {
      for(k=0; k<9; k++) {
B4      m[i][j]+=m[i][k];
      }
    } } }
```

	B1	B2	B3	B4
Base	10’000	10’000	100	100
Enhanced	5’000	10’000	90	10

Results

Benchmark	Max observed [cycles]	Enhanced [cycles]		Base [cycles]	
MatMult	$2.68 \cdot 10^{9}$	$2.73 \cdot 10^{9}$	2%	$2.73 \cdot 10^{9}$	2%
Jacobi	$0.88 \cdot 10^{10}$	$1.08 \cdot 10^{10}$	22%	$1.08 \cdot 10^{10}$	22%
JavaLayer	$2.67 \cdot 10^{9}$	$1.30 \cdot 10^{10}$	**487%**	$1.49 \cdot 10^{9}$	558%
SciMark	$2.47 \cdot 10^{10}$	$1.42 \cdot 10^{11}$	**579**%	$2.12 \cdot 10^{11}$	858%
_201_compress	$9.45 \cdot 10^{9}$	$1.11 \cdot 10^{10}$	**117**%	$4.76 \cdot 10^{12}$	50’370%

_201_compress

```
for (data = 0; data < N; data++) {

   // compress data

   if (data == 10'000) {

      // update structures

   }

}
```

Concluding remarks

- We tighten the bounds of basic blocks iteration considering different paths inside loop bodies.
- We do not perform path enumeration
- Tests with real applications validate the semantic analysis.

Questions?