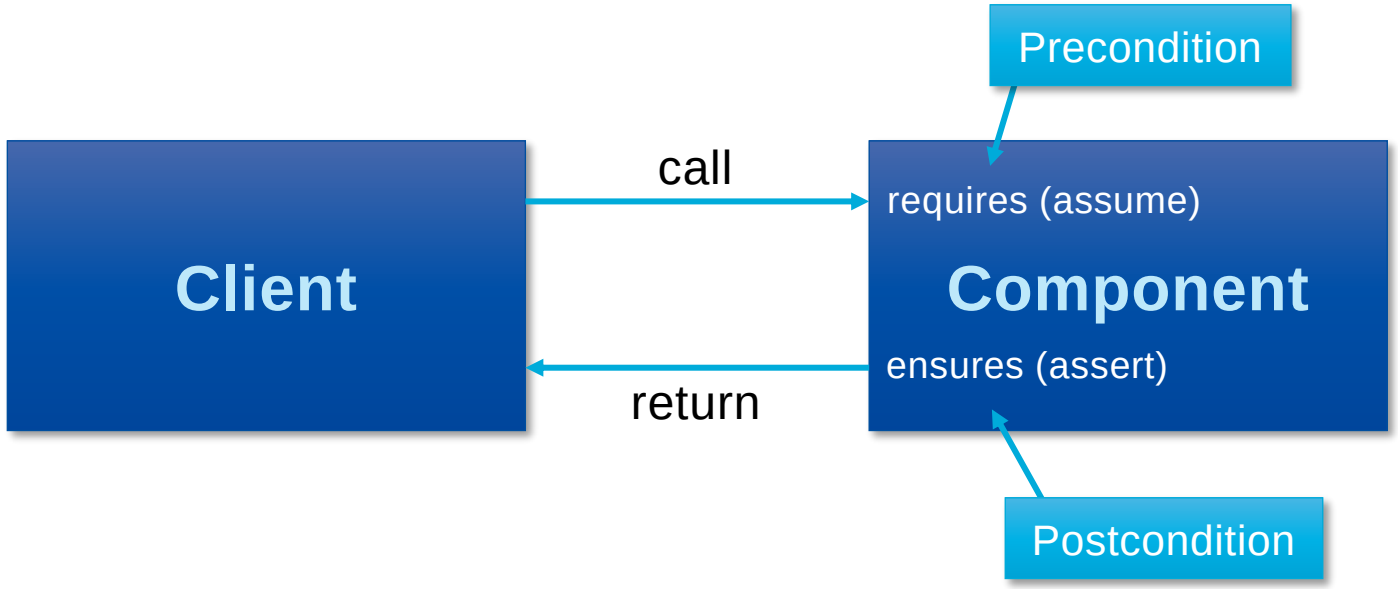
Carl von Ossietzky
Universität
Oldenburg

Detecting Redundant Preconditions

Nicola Thoben, Heike Wehrheim
09.09.2024

Design by Contract



Example

```
x = 0;
y = N;
z = M;
while (x < N) {
    x++;
    y--;
    z--;
}
```

Example

```
assume(N >= 0);
assume(N <= 1000);
assume(N >= -100);
assume(M >= 0);
x = 0;
y = N;
z = M;
while (x < N) {
    x++;
    y--;
    z--;
}
assert(y == 0);
```

Preconditions

Postcondition

Redundant Preconditions

```
assume(N >= 0);
assume(N <= 1000);
assume(N >= -100);
assume(M >= 0);
x = 0;
y = N;
z = M;
while (x < N) {
    x++;
    y--;
    z--;
}
assert(y == 0);
```

Detecting Redundant Preconditions

Implication Checking

Dependency Calculation

Predicate Checking

Implication Checking

pre_1 — `assume(N >= 0);`
pre_2 — `assume(N <= 1000);`
pre_3 — `assume(N >= -100);`
pre_4 — `assume(M >= 0);`

```
x = 0;
y = N;
z = M;
while (x < N) {
 x++;
 y--;
 z--;
}
assert(y == 0);
```

$$\bigwedge_{j \in (\{1,\ldots,n\} \setminus \{i\})} pre_j \Rightarrow pre_i$$

Implication Checking

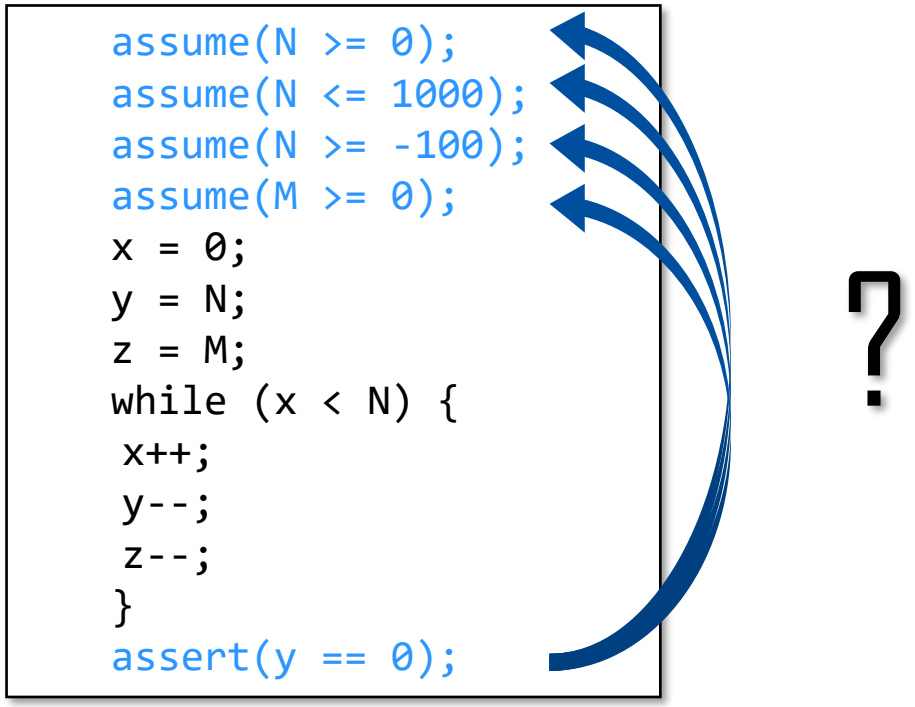
pre_1 `assume(N >= 0);`
pre_2 `assume(N <= 1000);`
pre_3 ~~`assume(N >= -100);`~~
pre_4 `assume(M >= 0);`

```
x = 0;
y = N;
z = M;
while (x < N) {
 x++;
 y--;
 z--;
}
assert(y == 0);
```

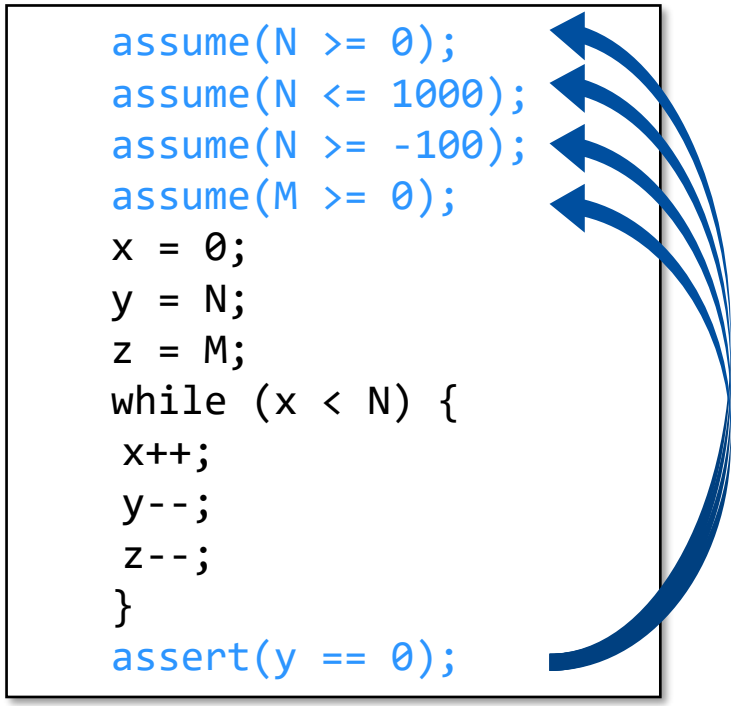
$$\bigwedge_{j \in (\{1,\ldots,n\} \setminus \{i\})} pre_j \Rightarrow pre_i$$

$$\boldsymbol{N \geq 0} \wedge N \leq 1000 \wedge M \geq 0$$
$$\Rightarrow$$
$$\boldsymbol{N \geq -100}$$

Dependency Calculation



Dependency Calculation

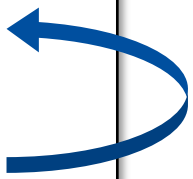


?

Program Dependence Graph

Dependency Calculation

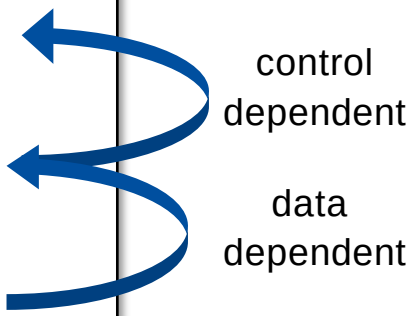
```
assume(N >= 0);
assume(N <= 1000);
assume(N >= -100);
assume(M >= 0);
x = 0;
y = N;
z = M;
while (x < N) {
 x++;
 y--;
 z--;
}
assert(y == 0);
```



data
dependent

Dependency Calculation

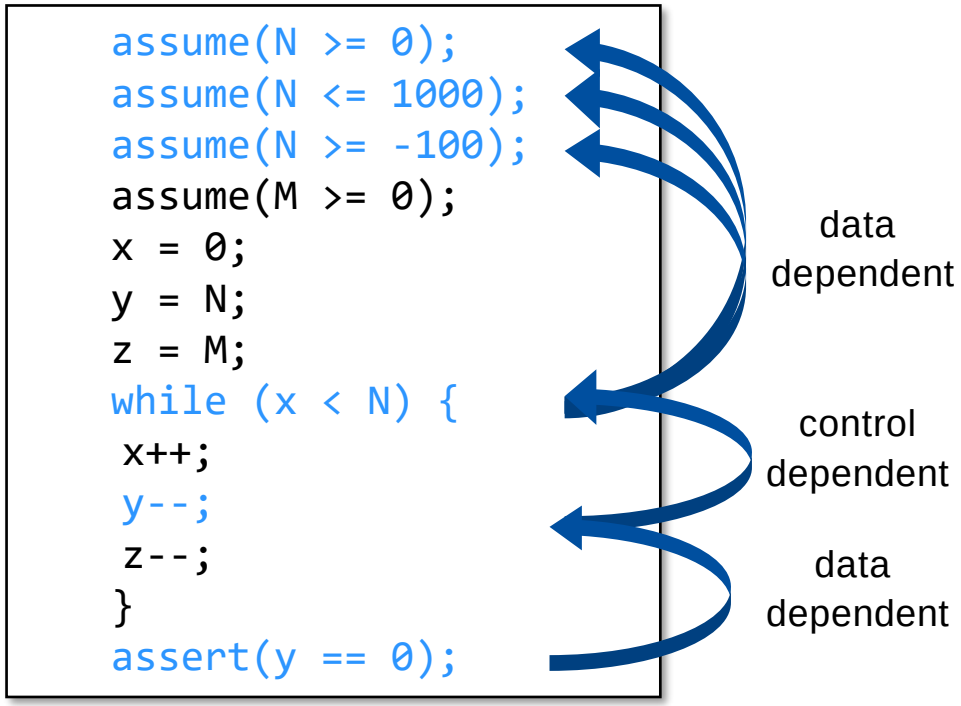
```
assume(N >= 0);
assume(N <= 1000);
assume(N >= -100);
assume(M >= 0);
x = 0;
y = N;
z = M;
while (x < N) {
 x++;
 y--;
 z--;
}
assert(y == 0);
```



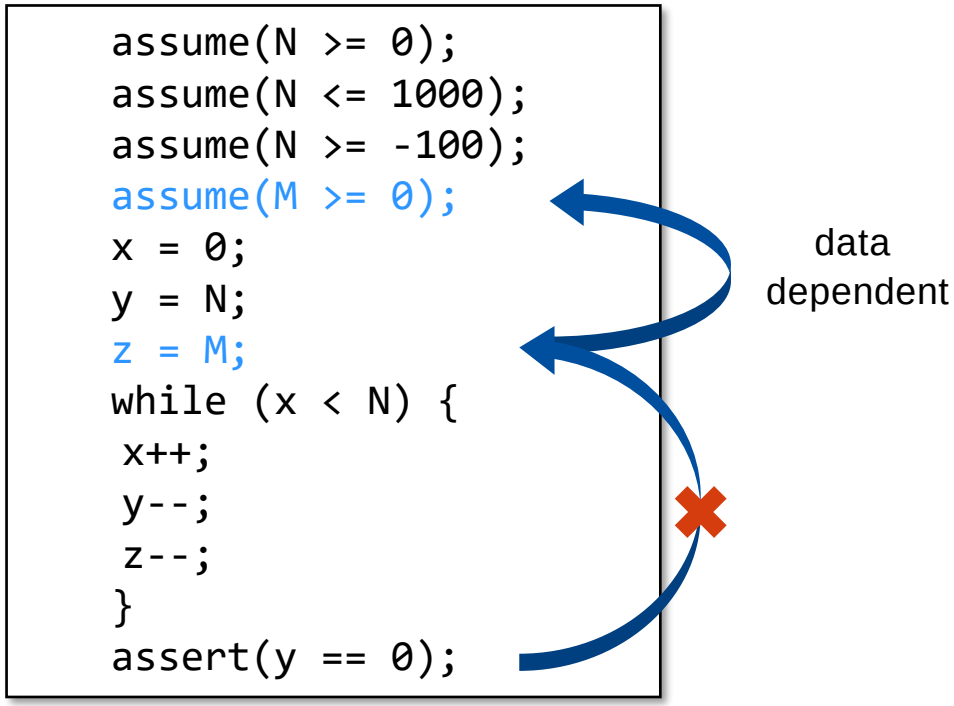
control dependent

data dependent

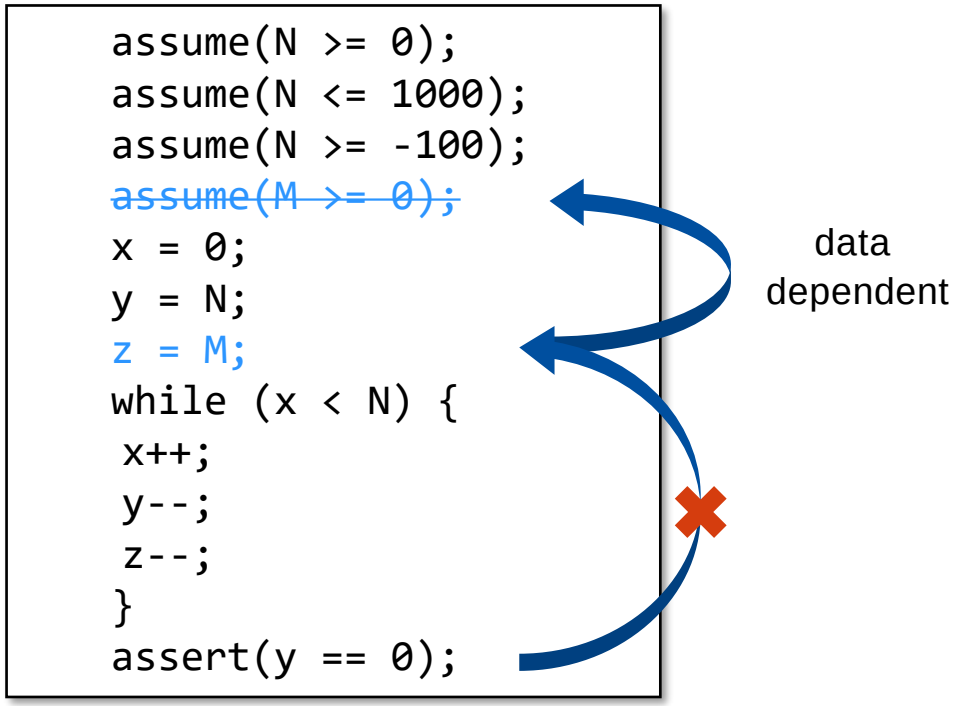
Dependency Calculation



Dependency Calculation



Dependency Calculation

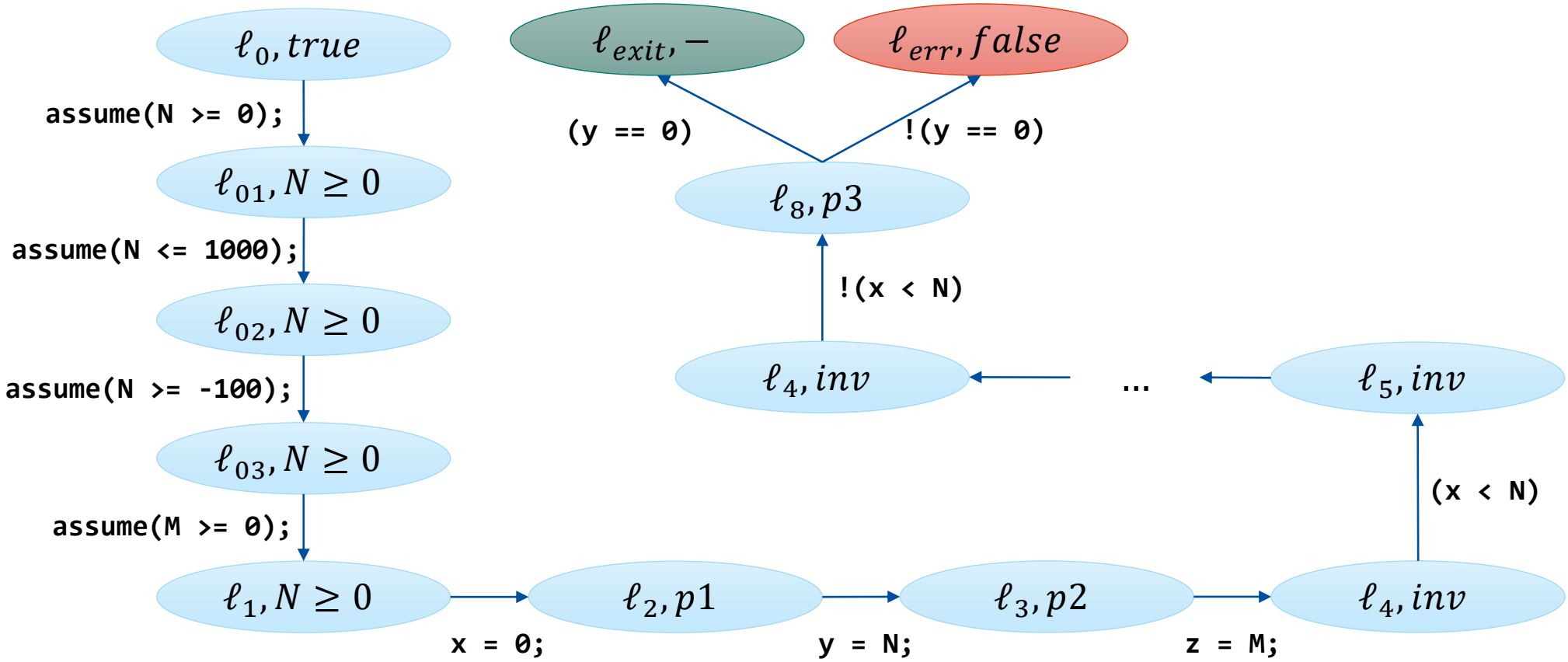


Predicate Checking

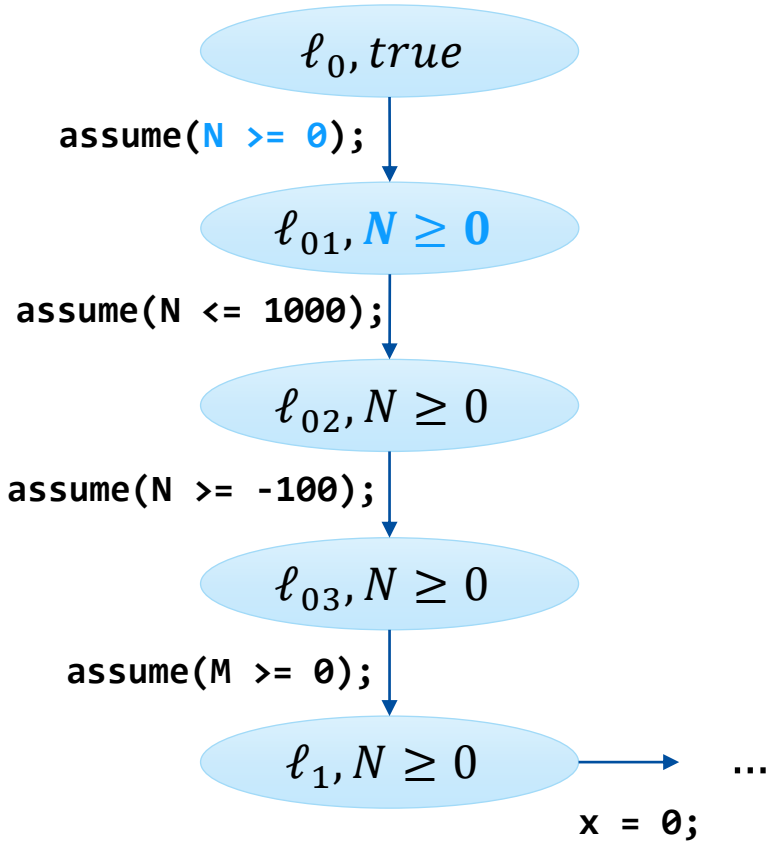
1. Build ARG

2. Inspect ARG

Predicate Checking: Building ARG

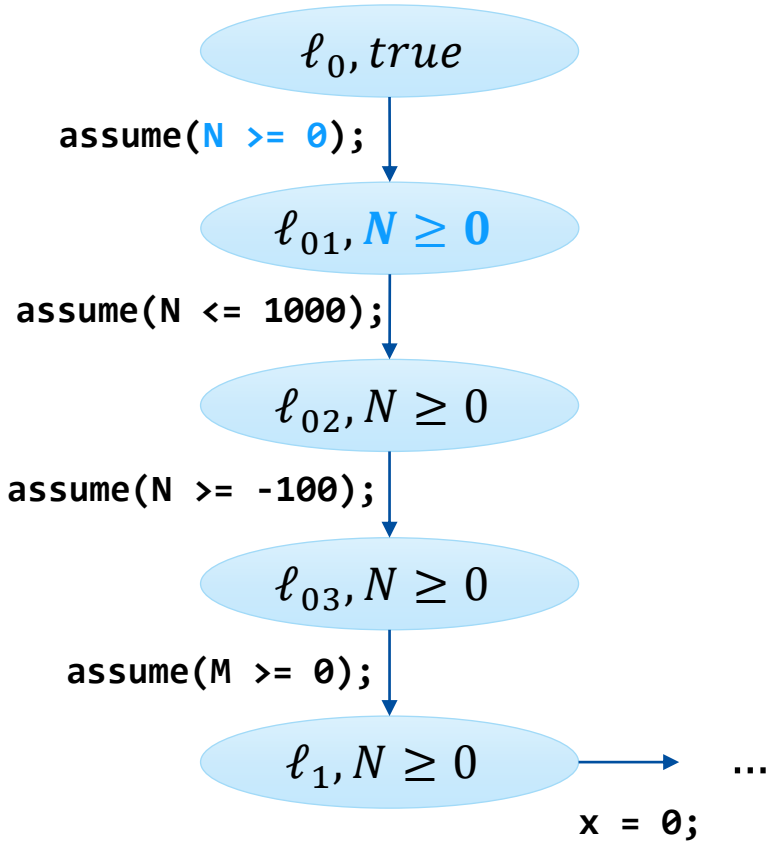


Predicate Checking: Inspect ARG




```
assume(N >= 0);
assume(N <= 1000);
assume(N >= -100);
assume(M >= 0);
x = 0;
y = N;
z = M;
while (x < N) {
 x++;
 y--;
 z--;
}
assert(y == 0);
```

Predicate Checking: Inspect ARG




```
assume(N >= 0);
assume(N <= 1000);
assume(N >= -100);
assume(M >= 0);
x = 0;
y = N;
z = M;
while (x < N) {
 x++;
 y--;
 z--;
}
assert(y == 0);
```

Evaluation - Benchmark

39 C Programs

Redundant Preconditions

3 Types of Preconditions

- Independency Preconditions
- Implication Preconditions
- Range Preconditions

Evaluation

detected redundant precon.

all redundant precon. of this type

	IC	DC	PC
Independency Precon.	0 (39)	39 (39)	39 (39)
Implication Precon.	39 (39)	0 (39)	39 (39)
Range Precon.	2 (17)	0 (17)	11 (17)
total	**41 (95)**	**39 (95)**	**89 (95)**

all redundant precon.

Carl von Ossietzky Universität Oldenburg

Evaluation: H1 - Effectiveness

… we expect
IC and DC to be incomparable…

	IC	DC	PC
Independency Precon.	0 (39)	39 (39)	39 (39)
Implication Precon.	39 (39)	0 (39)	39 (39)
Range Precon.	2 (17)	0 (17)	11 (17)
total	**41 (95)**	**39 (95)**	**89 (95)**

Evaluation: H1 - Effectiveness

… and
PC to be superior to both IC and DC…

	IC	DC	PC
Independency Precon.	0 (39)	39 (39)	39 (39)
Implication Precon.	39 (39)	0 (39)	39 (39)
Range Precon.	2 (17)	0 (17)	11 (17)
total	**41 (95)**	**39 (95)**	**89 (95)**

Carl von Ossietzky Universität Oldenburg

Evaluation: H2 - Completeness

… we expect
IC and DC to be incomplete…

	IC	DC	PC
Independency Precon.	0 (39)	39 (39)	39 (39)
Implication Precon.	39 (39)	0 (39)	39 (39)
Range Precon.	2 (17)	0 (17)	11 (17)
total	**41 (95)**	**39 (95)**	**89 (95)**

Carl von Ossietzky Universität Oldenburg

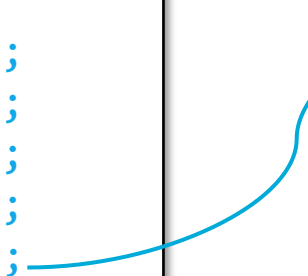
Evaluation: H2 - Completeness

… and
PC to be complete…

	IC	DC	PC
Independency Precon.	0 (39)	39 (39)	39 (39)
Implication Precon.	39 (39)	0 (39)	39 (39)
Range Precon.	2 (17)	0 (17)	11 (17)
total	**41 (95)**	**39 (95)**	**89 (95)**

Example: Incompleteness of PC

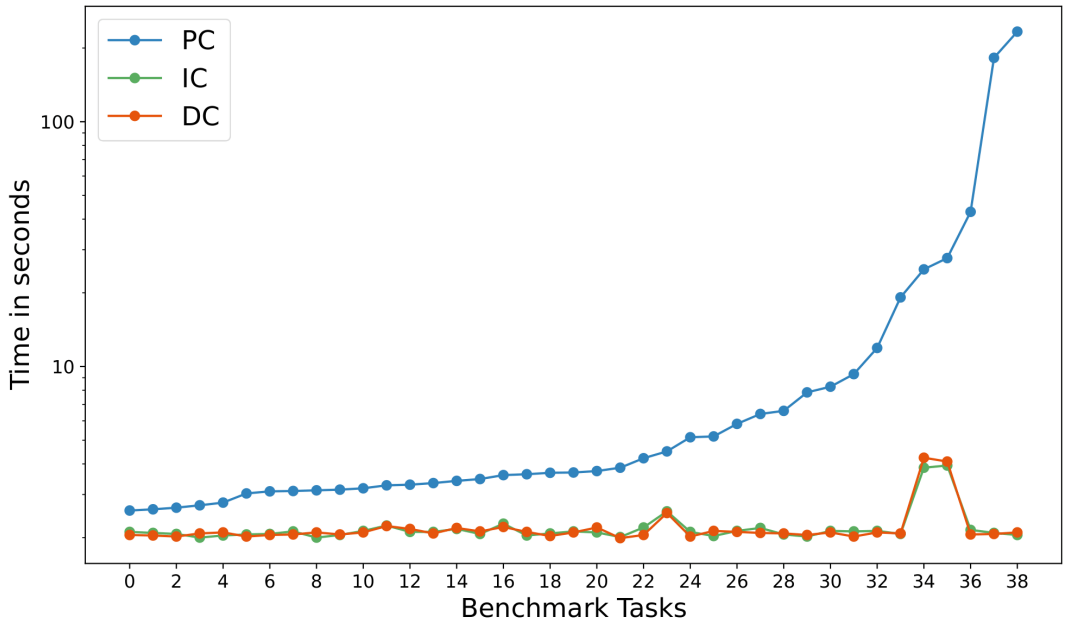
```
// necessary
assume(num1 >= 0);
assume(num2 >= 0);
assume(num3 >= 0);
assume(num4 >= 0);

// redundant
assume(dep == 0);
assume(num1 < 100);
assume(num2 < 100);
assume(num3 < 100);
assume(num4 < 100);
assume(num2 >= -8);

...

assert(...);
```

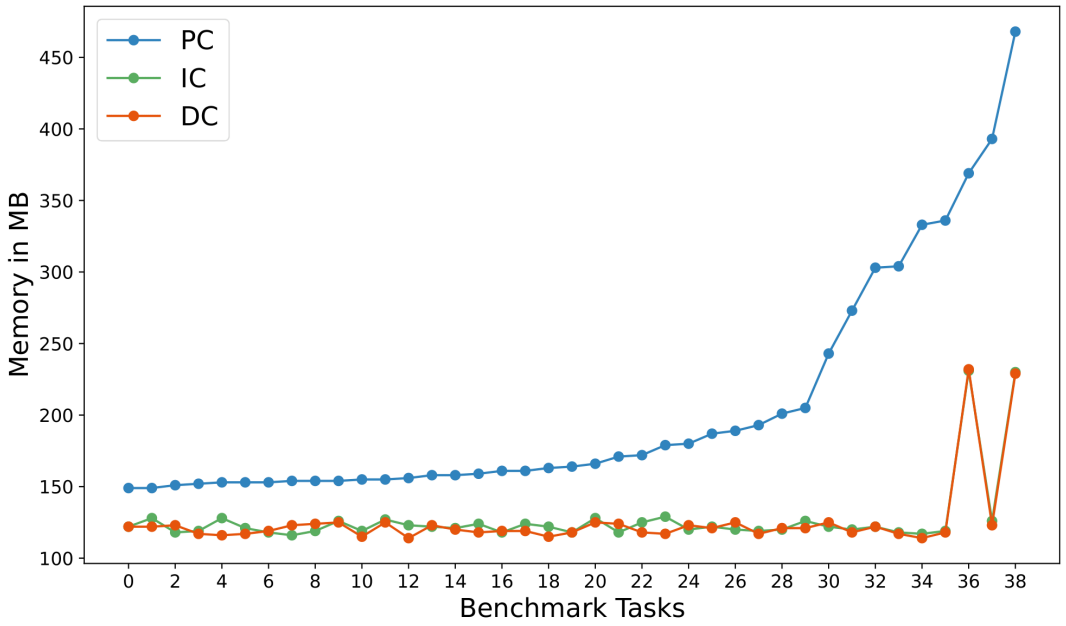


```
// necessary
assume(num1 >= 0);
assume(num2 >= 0);
assume(num3 >= 0);
assume(num4 >= 0);

// redundant
assume(num2 >= -8);
assume(dep == 0);
assume(num1 < 100);
assume(num2 < 100);
assume(num3 < 100);
assume(num4 < 100);

...

assert(...);
```

Evaluation: H3 - Efficiency

…we expect
IC and DC to require an equal, low runtime and memory consumption and **PC to need more resources.**





Carl von Ossietzky Universität Oldenburg

Summary

Hypotheses 1
Effectiveness

Hypotheses 2
Completeness

Hypotheses 3
Efficiency

Future Work

- IC and DC as preprocessing for PC
- PC based on witnesses
- Other tools for benchmark generation

introclass_smallest.c

```
void main(){
  int num1, num2, num3, num4,
  smaller1, smaller2, smallest;

//assumes
  dep=10;

  if (num1 < num2)
    smaller1 = num1;
  else
    smaller1 = num2;
  if (num4 < num3)
    smaller2 = num4;
  else
    smaller2 = num3;

  if (smaller1 < smaller2)
    smallest = smaller1;
  else smallest = smaller2;

  __VERIFIER_assert(smallest <= smaller1 && smallest <=
smaller2 && smallest >= 0);
}
```

Redundant Preconditions

Program	P
Preconditions	$Pre = \{pre_1, \dots, pre_n\}$
	$pre = \bigwedge_{i=0}^{n} pre_i$
Postconditions	$Post = \{post_1, \dots, post_n\}$
	$post = \bigwedge_{i=0}^{n} post_i$
Contract	$(Pre, Post)$
Execution path	$\pi = (\ell_0, c_0) \xrightarrow{g_1} (\ell_1, c_1) \xrightarrow{g_2} \cdots \xrightarrow{g_n} (\ell_n, c_n)$

P **satisfies a contract**
$P \vDash (Pre, Post)$
if for all π such that
$c_0 \vDash pre$ and $\ell_n \in L_{exit}$
we have $c_n \vDash post$

pre_i is **redundant** in Pre if
$P \vDash (Pre \backslash \{pre_i\}, Post)$

Π is **group-redundant** in Pre if
$P \vDash (Pre \backslash \{\Pi\}, Post)$