

# Access Permissions Inference via Abstract Interpretation

Pietro Ferrara and Peter Müller

ETH Zurich  
Switzerland


Microsoft Research, Redmond, WA, U.S.A.

# Access permissions

- OO programs
- Modular reasoning
  - > Design by Contracts
- Side effects
  - > Problem for modular reasoning!
- Restrict this scenario:
  - > Access a location iff we have the permission

```
class Coord {  
    int x, y;  
    //requires acc(x)  
    //ensures acc(x)  
    //ensures x==t  
    void updateX(int t) {  
        x=t;  
    }  
}
```

```
void test(Coord c) {  
    c.x=1;  
    c.y=1;  
    c.updateX(-1);  
    assert(c.y==1);  
}
```



# Fine-grained Permissions

- **Full permission:**
  - > Read&write access
- **Partial permission:**
  - > Read access
- **No permission:**
  - > No access
- **Fine-grained:**
  - > Fractional, counting, Chalice, ...

```
class Coord {  
  //invariant acc(x, 50%)  
  int x, y;  
  //requires acc(x, 50%)  
  //ensures acc(x, 50%)  
  //ensures x=t ✓  
  void updateX(int t) {  
    acquire this;  
    x=t;  
    release this;  
  }  
}
```

this.x ↦ 50%  
+50%

Read access on this.x

# Motivations and goals

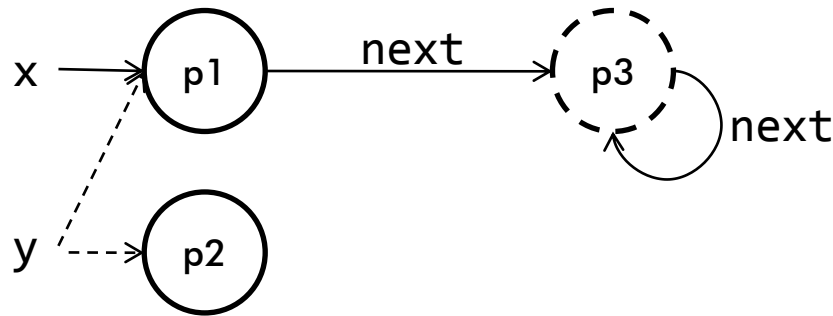
- **Annotation overhead of permissions**
  - > Quite verbose
  - > The code already contains all the accesses
- **Start with a program without annotation**
- **Apply static analysis**
  - > Based on abstract interpretation
  - > To infer the permissions that could be specified
    - Strong enough to perform the heap accesses
    - As weak as possible

# Outline

1. Introduction
2. Symbolic permissions
3. Annotation inference
4. A case study: the Owicki-Gries example
5. Experimental results & Conclusion

# Heap Abstraction

- Not a contribution of this work
- Generic approach
  - > Given a heap access, it returns a heap id



- Implemented a simple standard analysis
  - > Abstract heap nodes with program points
  - > Apply other heap analyses, e.g., TVLA

# Symbolic Permissions

- Many ways to specify permissions
- Symbolic values
  - > Pre-conditions:
    - $Pre(C, m, p.f)$
    - Method  $m$  in class  $C$  over path  $p.f$
  - > Post-conditions
  - > Monitor invariants
- Symbolic values:  $\overline{SV}$

```
class Coord {  
  int x, y;  
  void updateX(int t) {  
    acquire this;  
    x=t;  
    release this;  
  }  
}
```

$this.x \mapsto Pre(\text{Coord}, \text{updateX}, this.x)$   
 $this.y \mapsto Pre(\text{Coord}, \text{updateX}, this.y)$

???

Precondition+Monitor invariant!

# Symbolic Levels

- Inhale and exhale
  - > several times
  - > on the same location
- Sum of symbolic values
  - > At a given pp
  - > For each location
- Values:

```
class Coord {  
  int x, y;  
  void updateX(int t) {  
    acquire this;  
    x=t;  
    release this;  
  }  
}
```

$\text{this.x} \mapsto$

$Pre(\text{Coord}, \text{updateX}, \text{this.x})$   
 $+ MI(\text{Coord}, \text{this.x})$

$\text{this.y} \mapsto$

$Pre(\text{Coord}, \text{updateX}, \text{this.y})$   
 $+ MI(\text{Coord}, \text{this.y})$

$$\overline{AV} = \left\{ \sum_i a_i * s_i + c \text{ where } a_i, c \in \mathbb{Z}, s_i \in \overline{SV} \right\}$$

# Lattice Structure

- Surely owned permissions
- Upper bound  
> Minimum
- Goal  
> Infer enough permissions to perform the heap accesses contained in the code

```
if(...)  
    c = new Coord();  
else ...;  
c.x=5;
```

$c.x \mapsto \text{full}$   
 $c.y \mapsto \text{full}$   $\xrightarrow{\gamma}$   $\{\text{full}\}$

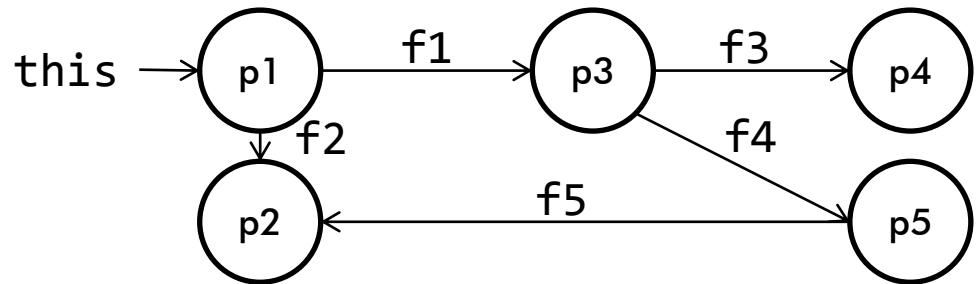
$\sqcup$   
 $c.x \mapsto 0$   
 $c.y \mapsto 0$   $\xrightarrow{\gamma}$   $\{0, \dots, \text{full}\}$

$=$

$c.x \mapsto 0$   
 $c.y \mapsto 0$

# What could be specified?

- **Reachable( $\overline{reach}$ )**
  1. Start from local variables
  2. Fields of added nodes
  3. Until we visit all reachable nodes
    - They are finite
    - We compute *one* of the shortest paths
  4. Rename the paths



1<sup>st</sup> step:

p1 reachable through this

2<sup>nd</sup> step:

p2 reachable through this.f2

p3 reachable through this.f1

3<sup>rd</sup> step:

p4 reachable through this.f1.f3

p5 reachable through this.f1.f4

# Abstract Semantics

- Based on inhale (+) and exhale (-)
  - > Method call
    - Exhale precondition
    - Inhale postcondition
  - > Acquire a monitor
    - Inhale invariant
  - > Release a monitor
    - Exhale invariant
- Rely on  $\overline{reach}$

```
class Coord {  
    int x, y;  
    void updateX(int t) {  
        acquire this;  
        x=t;  
        release this;  
    }  
}
```

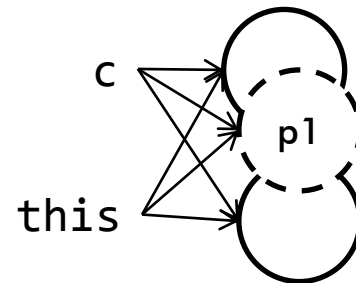
$this.x \mapsto$   
 $Pre(Coord, updateX, this.x)$   
 $+ MI(Coord, this.x)$

$this.y \mapsto$   
 $Pre(Coord, updateX, this.y)$   
 $+ MI(Coord, this.y)$

# Unsoundness

- Summary node
  - > Many concrete nodes
  - > We cannot inhale!
- Arguments/fields of compatible types
  - > Potential aliases
  - > Summary node
- Unsoundness
  - > Improve the amount of inferred contracts

```
class Coord {  
  int x, y;  
  void copy(Coord c) {  
    acquire this;  
    x=c.x;  
    y=c.y;  
    release this;  
  }  
}
```



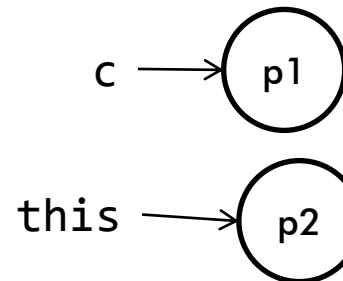
$p1.x \mapsto 0$

$Pre(\text{Coord}, \text{copy}, \text{this}.x)$   
 $+Pre(\text{Coord}, \text{copy}, c.x)$

# Unsoundness

- Arguments/fields of compatible types
  - > We suppose they are not aliases
- Summary node
  - > We inhale
- Unsoundness
  - > Improve the amount of inferred contracts

```
class Coord {  
    int x, y;  
    void copy(Coord c) {  
        acquire this;  
        x=c.x;  
        y=c.y;  
        release this;  
    }  
}
```



$p1.x \mapsto \text{Pre}(\text{Coord}, \text{copy}, c.x)$   
 $p2.x \mapsto \text{Pre}(\text{Coord}, \text{copy}, \text{this}.x)$   
 $+ \text{MI}(\text{Coord}, \text{this}.x)$

# Outline

1. Introduction
2. Symbolic permissions
3. Annotation inference
4. A case study: the Owicki-Gries example
5. Experimental results & Conclusion

# Constraint Inference

- Heap accesses impose constraints:

> Write: full

> Read: non-zero

- Inhale:  $\bar{s} \leq \text{full}$

- Exhale:  $\bar{s} \geq 0$

- $\forall \bar{s} \in \overline{SV} : 0 \leq \bar{s} \leq \text{full}$

- $\text{Post}(C, m, \dots) == \text{exit}$

```
class Coord {  
  int x, y;  
  void updateX(int t) {  
    acquire this;  
    x=t;  
    release this;  
  }  
}
```

$Pre(\text{Coord}, \text{updateX}, \text{this.}_) + MI(\text{Coord}, \text{this.}_) \leq \text{full}$

$Pre(\text{Coord}, \text{updateX}, \text{this.x}) + MI(\text{Coord}, \text{this.x}) == \text{full}$

$Pre(\text{Coord}, \text{updateX}, \text{this.}_) \geq 0$

$Pre(\text{Coord}, \text{updateX}, \text{this.}_) == \text{Post}(\text{Coord}, \text{updateX}, \text{this.}_)$

# Permission Systems

- Various systems

- > Fractional

- > Counting

- > Chalice

- Combination

- Parameters

- > Full perm.

- > Fractional perm.

- > Infinitesimal perm.

- > Read perm.

System	full	fract	infin	ensRd(p)
Fractional	1	✓	✗	$p > 0$
Counting	MAX	✗	✗	$p \geq 1$
Chalice	100	✓	✓	$p \geq \epsilon$

- $Pre(\text{Coord}, \text{updateX}, \text{this.}_) + MI(\text{Coord}, \text{this.}_) \leq 1$
- $Pre(\text{Coord}, \text{updateX}, \text{this.x}) + MI(\text{Coord}, \text{this.x}) == 1$
- $Pre(\text{Coord}, \text{updateX}, \text{this.}_) \geq 0$
- $Pre(\text{Coord}, \text{updateX}, \text{this.}_) == Post(\text{Coord}, \text{updateX}, \text{this.}_)$

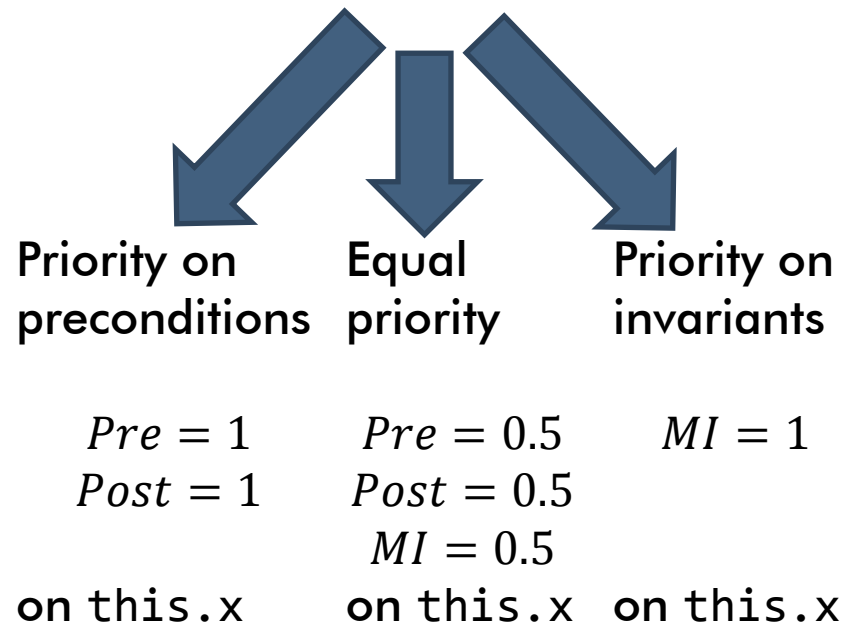
# Linear Programming

- **Linear programming**
  - > Solve our system
- **Objective function:**
  - >  $\sum_i n_i * s_i$  where
    - n: integer coefficient
    - s: symbolic value
  - > Minimize it
    - Goal: permissions as weak as possible
  - > Higher coefficient  $\Rightarrow$  lower priority

$$Pre(\text{Coord}, \text{updateX}, \text{this.}_) + MI(\text{Coord}, \text{this.}_) \leq 1$$

$$Pre(\text{Coord}, \text{updateX}, \text{this.x}) + MI(\text{Coord}, \text{this.x}) == 1$$

$$Pre(\text{Coord}, \text{updateX}, \text{this.}_) \geq 0$$

$$Pre(\text{Coord}, \text{updateX}, \text{this.}_) == Post(\text{Coord}, \text{updateX}, \text{this.}_)$$


# Outline

1. Introduction
2. Symbolic permissions
3. Annotation inference
4. A case study: the Owicki-Gries example
5. Experimental results & Conclusion

# Discussion

```
class Cell {  
  int x, c1, c2;  
  invariant x==c1+c2,  
}
```

rd(x) &&  
rd(c1) &&  
rd(c2)

- Can discharge  $\text{acc}(x)$  in the invariant of Cell
- Cannot discharge  $\text{acc}(c1)$  or  $\text{acc}(c2)$  in the invariant of Cell
- precondition of  $\text{Inc}()$  in W1 plus invariant in Cell == 100% for c1
- precondition of  $\text{Inc}()$  in W2 plus invariant in Cell == 100% for c2

```
class W1 {  
  Cell c;  
  void Inc()  
  ens c.c1==old(c.c1)+1  
  acquire c;  
  c.x=c.x+1;  
  c.c1=c.c1+1;  
  release c;  
}
```

rd(c.c1)

acc(c.c1)  
&&  
acc(c.x)

```
class W2 {  
  Cell c;  
  void Inc()  
  ens c.c2==old(c.c2)+1  
  acquire c;  
  c.x=c.x+1;  
  c.c2=c.c2+1;  
  release c;  
}
```

rd(c.c2)

acc(c.c2)  
&&  
acc(c.x)

```
  acquire c;  
  assert c.x==2;  
}
```

# Abstract Semantics

```
class Cell {
  int x, c1, c2;
  invariant x==c1+c2
}
```

$* \rightarrow \text{MI}(\text{Cell}, *)$

```
void main() {
  Cell c = new Cell();
  share c;
  W1 w1 = new W1();
  w1.c=c;
  W2 w2 = new W2();
  w2.c=c;
  t1 = fork w1.Inc();
  t2 = fork w2.Inc();
  join t1;
  join t2;
  acquire c;
  assert c.x==2;
}
```

```
class W1 {
  Cell c;
  ens c.c1==old(c.c1)+1 {
    acquire c;
    c.x=c.x+1;
    c.c1=c.c1+1;
    release c;
  }
}
```

$c.c1 \rightarrow \text{Post}(\text{W1}, \text{Inc}(), c.c1)$

$c.* \rightarrow \text{Pre}(\text{W1}, \text{Inc}(), c.*) + \text{MI}(\text{Cell}, *)$

```
class W2 {
  Cell c;
  ens c.c2==old(c.c2)+1 {
    acquire c;
    c.x=c.x+1;
    c.c2=c.c2+1;
    release c;
  }
}
```

$c.c2 \rightarrow \text{Post}(\text{W2}, \text{Inc}(), c.c2)$

$c.* \rightarrow 100 - \text{MI}(\text{Cell}, *) - \text{Pre}(\text{W1}, \text{Inc}(), c.*) - \text{Pre}(\text{W2}, \text{Inc}(), c.*)$

$c.* \rightarrow \text{Pre}(\text{W2}, \text{Inc}(), c.*) + \text{MI}(\text{Cell}, *)$

$c.* \rightarrow 100$

# Constraint Inference

```

class Cell {
  int x, c1, c2;
  invariant x==c1+c2
}
  *→MI(Cell,*)

void main() {
  Cell c = new Cell();
  share c;
  W1 w1 = new W1();
  w1.c=c;
  W2 w2 = new W2();
  w2.c=c;
  t1 = fork w1.Inc();
  t2 = fork w2.Inc();
  join t1;
  join t2;
  acquire c;
  assert c.x==2;
}

```

$c.x \rightarrow 100 - MI(\text{Cell}, *) - \text{Pre}(W1, \text{Inc}(), c.*) - \text{Pre}(W2, \text{Inc}(), c.*)$

```

class W1 {
  Cell c;
  c.c1→Post(W1,Inc(),c.c1)
  ens c.c1==old(c.c1)+1 {
    acquire c;
    c.x=c.x+1;
    c.c1=c.c1+1;
    release c;
  }
}

```

$c.* \rightarrow \text{Pre}(W1, \text{Inc}(), c.*) + MI(\text{Cell}, *)$

$Post(W1, \text{Inc}, c.c1) \geq \epsilon$   
 $Post(W2, \text{Inc}, c.c2) \geq \epsilon$   
 $MI(\text{Cell}, \text{this}.*) \geq \epsilon$   
 $Pre(W1, \text{Inc}, c.c1) + MI(\text{Cell}, c1) == 100$   
 $Pre(W1, \text{Inc}, c.x) + MI(\text{Cell}, x) == 100$   
 $Pre(W2, \text{Inc}, c.c2) + MI(\text{Cell}, c2) == 100$   
 $Pre(W2, \text{Inc}, c.x) + MI(\text{Cell}, x) == 100$   
 $100 - MI(\text{Cell}, *) - \text{Pre}(W1, \text{Inc}, c.*) - \text{Pre}(W2, \text{Inc}, c.*) \geq 0$

# Solutions

$$\boxed{Post(W1, Inc, c.c1) \geq \epsilon}$$

$$Post(W1, Inc, c.c2) \geq \epsilon$$

$$MI(Cell, this.*) \geq \epsilon$$

$$\boxed{Pre(W1, Inc, c.c1) + MI(Cell, c1) == 100}$$

$$Pre(W1, Inc, c.x) + MI(Cell, x) == 100$$

$$Pre(W2, Inc, c.c2) + MI(Cell, c2) == 100$$

$$Pre(W2, Inc, c.x) + MI(Cell, x) == 100$$

$$100 - MI(Cell, *) - Pre(W1, Inc, c.*) - Pre(W2, Inc, c._) \geq 0$$

- **Maximize monitor invariants**

Symbolic value	Numerical permission
$Pre(W1, Inc, c.x) = Post(W1, Inc, c.x)$	0
$\boxed{Pre(W1, Inc, c.c1) = Post(W1, Inc, c.c1)}$	$\boxed{\epsilon}$
$MI(Cell, c1)$	$\boxed{100 - \epsilon}$
$MI(Cell, x)$	100

# Solutions

$$Post(W1, Inc, c.c1) \geq \epsilon$$

$$Post(W1, Inc, c.c2) \geq \epsilon$$

$$MI(Cell, this.*) \geq \epsilon$$

$$Pre(W1, Inc, c.c1) + MI(Cell, c1) == 100$$

$$Pre(W1, Inc, c.x) + MI(Cell, x) == 100$$

$$Pre(W2, Inc, c.c2) + MI(Cell, c2) == 100$$

$$Pre(W2, Inc, c.x) + MI(Cell, x) == 100$$

$$100 - MI(Cell, *) - Pre(W1, Inc, c.*) - Pre(W2, Inc, c._) \geq 0$$

- Maximize pre- and post- conditions

Symbolic value	Numerical permission
$Pre(W1, Inc, c.x) = Post(W1, Inc, c.x)$	0
$Pre(W1, Inc, c.c1) = Post(W1, Inc, c.c1)$	$100 - \epsilon$
$MI(Cell, c1)$	$\epsilon$
$MI(Cell, x)$	100

# Solutions

$$Post(W1, Inc, c.c1) \geq \epsilon$$

$$Post(W1, Inc, c.c2) \geq \epsilon$$

$$MI(Cell, this.*) \geq \epsilon$$

$$Pre(W1, Inc, c.c1) + MI(Cell, c1) == 100$$

$$Pre(W1, Inc, c.x) + MI(Cell, x) == 100$$

$$Pre(W2, Inc, c.c2) + MI(Cell, c2) == 100$$

$$Pre(W2, Inc, c.x) + MI(Cell, x) == 100$$

$$100 - MI(Cell, *) - Pre(W1, Inc, c.*) - Pre(W2, Inc, c._) \geq 0$$

- Maximize pre- and post- conditions

Symbolic value

Numerical permission

Pre(W1, Inc, c.c1)

If we discharge  $n > 0$  over  $Pre(W1, Inc, c.x)$

Then  $MI(Cell, x) = 100 - n$

We have to discharge  $n$  over  $Pre(W2, Inc, c.x)$  too

And  $100 - (100 - n) - n - n$  cannot be  $\geq 0$ !

# Solutions

$$Post(W1, Inc, c.c1) \geq \epsilon$$

$$Post(W1, Inc, c.c2) \geq \epsilon$$

$$MI(Cell, this.*) \geq \epsilon$$

$$Pre(W1, Inc, c.c1) + MI(Cell, c1) == 100$$

$$Pre(W1, Inc, c.x) + MI(Cell, x) == 100$$

$$Pre(W2, Inc, c.c2) + MI(Cell, c2) == 100$$

$$Pre(W2, Inc, c.x) + MI(Cell, x) == 100$$

$$100 - MI(Cell, *) - Pre(W1, Inc, c.*) - Pre(W2, Inc, c._) \geq 0$$

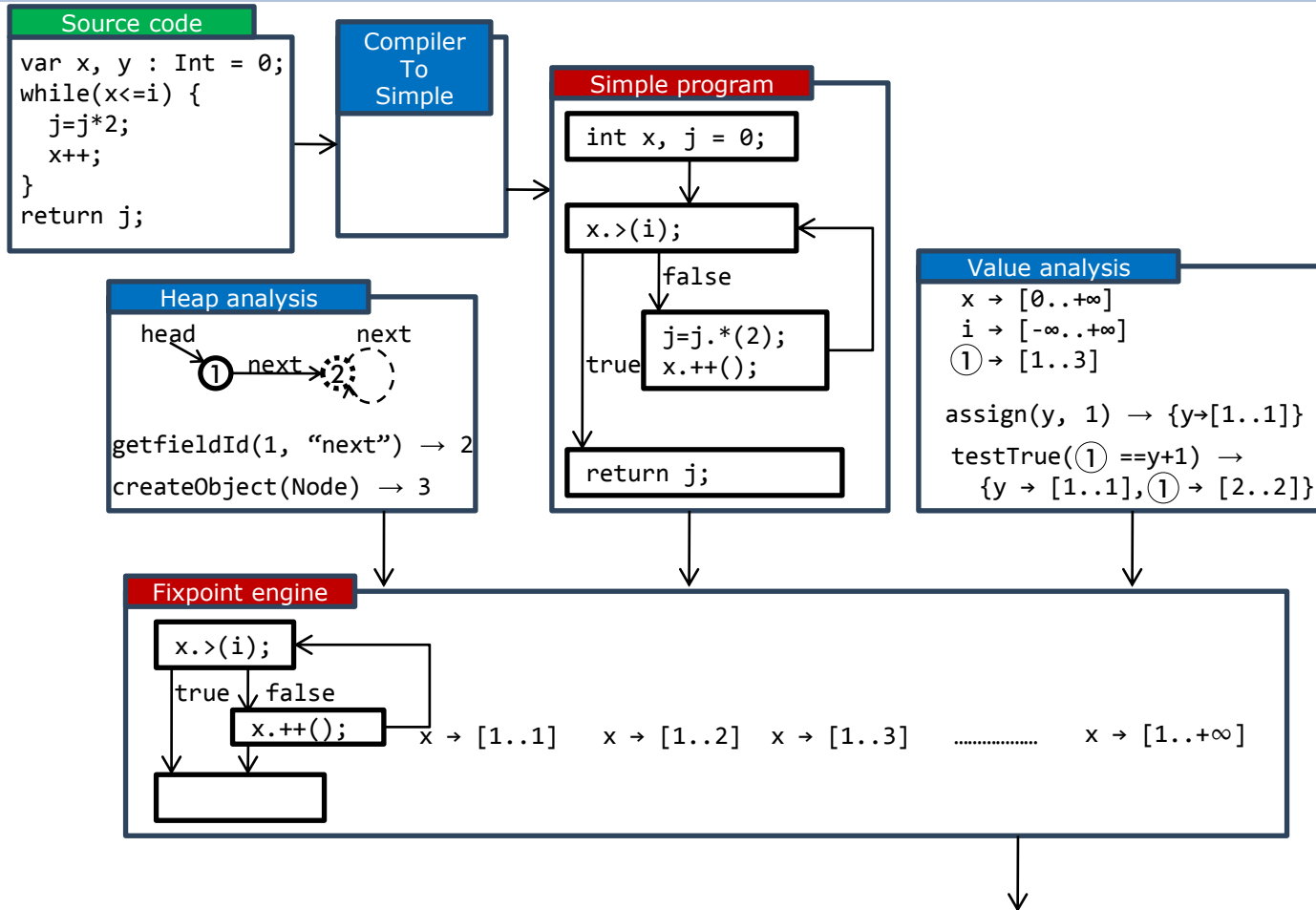
- **Same priority**

Symbolic value	Numerical permission
$Pre(W1, Inc, c.x) = Post(W1, Inc, c.x)$	0
$Pre(W1, Inc, c.c1) = Post(W1, Inc, c.c1)$	50
$MI(Cell, c1)$	50
$MI(Cell, x)$	100

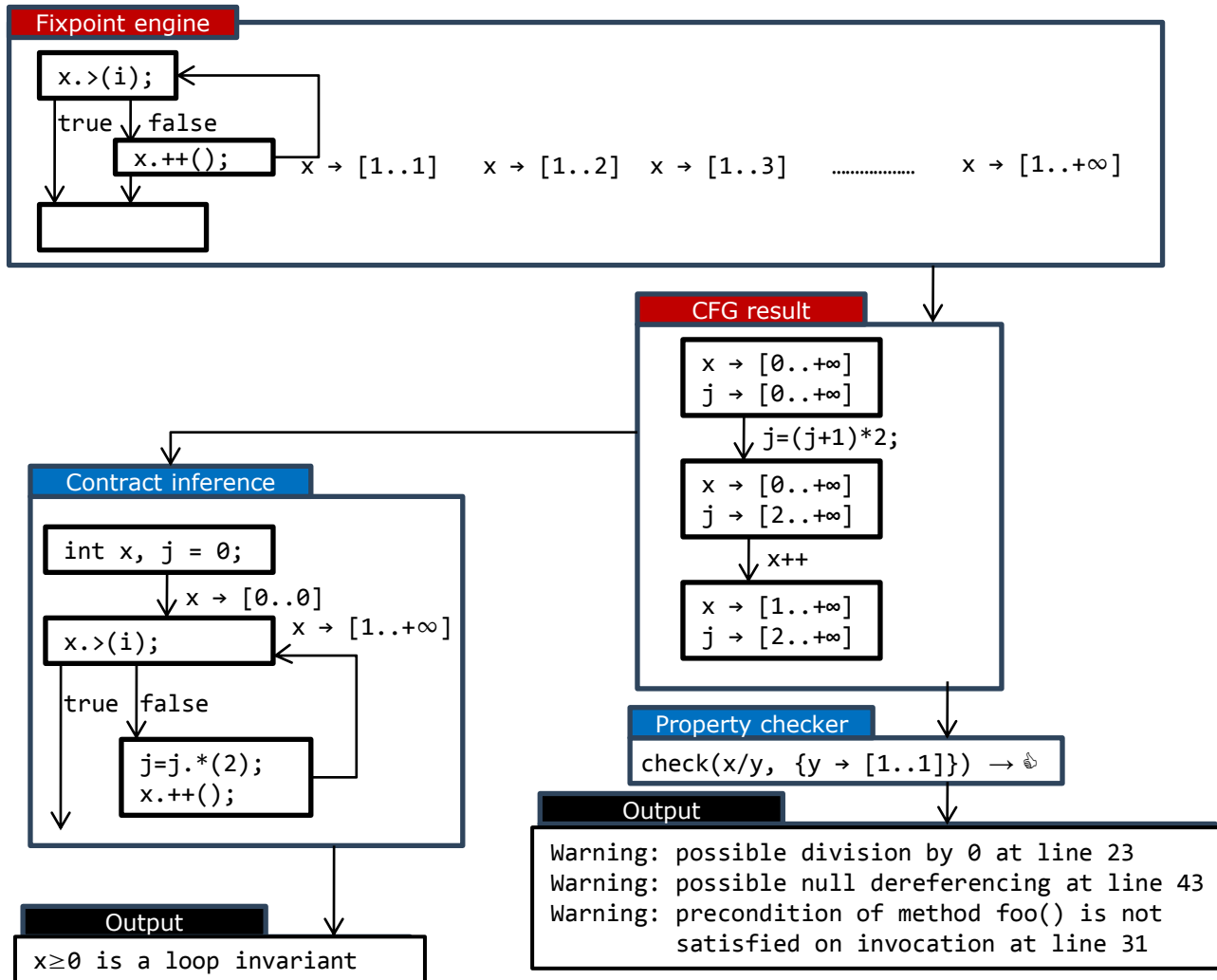
# Outline

1. Introduction
2. Symbolic permissions
3. Annotation inference
4. A case study: the Owicki-Gries example
5. Experimental results & Conclusion

# Sample's Structure



# Sample's Structure



# Experimental Results

- Implemented in Sample
  - > Generic static analyzer
- Intel Core 2 Quad CPU 2.83 Ghz
  - > 4 GB RAM
  - > Windows 7
- Fast
- Precise

Program	Time (msec)	% inf. contr.
Fig1	45	100%
Fig2	12	100%
Fig3	9	100%
Fig4	3	100%
Fig5	143	100%
Fig6	53	100%
Fig11	15	100%
Fig12	15	100%
Fig13	706	100%
OwickiGries	164	100%
Cell	115	100%
Linkedlist	78	100%
Swap	10	100%
AssociationList	668	36%
HandOverHand	564	36%
Master	76	100%
CellLib	148	100%
CompositePattern	1217	71%
Spouse	221	100%
Account	12	100%
Stack	76	67%
Iterator	46	100%

Chalice's tutorial

Chalice's distribution

Vericool

Verifast

# Conclusion

- **Static analysis to infer symbolic permissions**
- **System of linear constraints**
  - > Imposed by the semantics
- **Solved using linear programming**
  - > Many possible solutions
    - Priorities through the objective function
- **Implemented**
  - > Fast and precise