

Automatic Inference of Access Permissions

Pietro Ferrara and Peter Mueller

ETH Zurich
Switzerland

Sémantique et Interprétation Abstraite
École Normale Supérieure, Paris, France

Outline

1. Access permissions
2. Chalice
3. Abstract domain
4. Inference of permissions
5. Sample and experimental results
6. Conclusion

Full Permissions

- OO programs

test(new BadCoord())
breaks the assertion

> Side effect

- Problem for modular reasoning!

```
class Coord {  
  int x, y;  
  void flipH() {  
    x=-x;  
  }  
}
```

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

```
class BadCoord  
  extends Coord {  
  void flipH() {  
    super.flipH();  
    y=-y;  
  }  
}
```

- Restrict this scenario:
 - > Access a location iff we have the permission

Pietro Ferrara: "Automatic Inference of Access Permissions"

Sémantique et Interprétation Abstraite, École Normale Supérieure, Paris, France

Full Permissions

```
class Coord {
  int x, y;
  void flipH()
  req acc(x) {
    x=-x;
  }
}

void test(Coord c)
req acc(c.x) && acc(c.y) {
  c.x=1;
  c.y=1;
  c.flipH();
  assert(c.y==1);
}

class BadCoord
  extends Coord {
  void flipH()
  req acc(x) && acc(y) {
    super.flipH();
    y=-y;
  }
}
```

- Weaker pre-conditions:
 - > The same or fewer accesses
- Stronger post-conditions:
 - > The same or more accesses

Fractional Permissions

- Fine-grained permissions:
 - > Full permission → read & write access
 - > Partial permission → only read access
 - > No permission → no access

```
class Time {
  int h, s;
  int diff(Time t)
  requires rd(t.h) && rd(t.s)
    && rd(h) && rd(s) {
    return (t.h-h)*24+t.s-s;
  }
}
```

```
void client() {
  Time t1=new Time();
  Time t2=new Time();
  t1.h=1; t2.h=1;
  t1.s=10; t2.s=20;
  t1.diff(t2);
  assert(t1.h==1 && t2.h==1 &&
         t1.s==10 && t2.s==20);
}
```

Pietro Ferrara: “Automatic Inference of Access Permissions”

Sémantique et Interprétation Abstraite, École Normale Supérieure, Paris, France

Permission Transfer

- Permissions are transferred by methods

```
class Coord {  
  int x, y;  
  void flipH()  
  req acc(x)  
  ens acc(x) && x != -old(x) {  
    x = -x;  
  }  
}
```

```
void test(Coord c)  
req acc(x) && acc(y) {  
  c.x=1;  
  c.y=1;  
  c.flipH();  
  assert(c.x == -1);  
}
```

> Specification is self-framing

- Usually, a method
 - > requires what it needs
 - > gives back what it owns

Outline

1. Access permissions
2. Chalice
3. Abstract domain
4. Inference of permissions
5. Sample and experimental results
6. Conclusion

Chalice

- Experimental language
 - > Verification purposes
 - Pre- and post-conditions
 - Class (monitor) invariants
 - Loop invariants
 - Abstract predicates
- Focused on multithreading features:
 - > Join/Fork of threads
 - > Sharing of objects
 - > Monitors

Permissions

- In Chalice, permissions are percentages
 - > 100% → read & write access
 - > $\geq \epsilon$ → read access
 - ϵ infinitesimal (smallest) permission
 - > 0% → no access
 - > Inhale: obtain permissions
 - > Exhale: give away permissions
- Permissions specified in:
 - > Method, monitor invariants, abstract predicates
 - > Loop invariants

Pietro Ferrara: “Automatic Inference of Access Permissions”

Sémantique et Interprétation Abstraite, École Normale Supérieure, Paris, France

Permissions

- $\text{acc}(x)$: 100% permission on x
- $\text{acc}(x, n)$: $n\%$ permission on x
- $\text{rd}(x)$: ϵ permission on x
- $\text{o=new Obj}()$
 - > we own $\text{acc}(\text{o}.*)$
 - > o is thread-local
- share o
 - > The monitor invariant of o is exhaled

Permission Transfer

- Permissions can be transferred through:
 - > Method calls
 - Exhale pre-condition and inhale post-condition
 - > Locks
 - Acquire: inhale the monitor invariant
 - Release: exhale the monitor invariant
 - > Fork and join of threads
 - Fork: exhale the pre-condition of the run method
 - Join: inhale its post-condition
 - > Share of objects
 - Exhale the monitor invariant

Pietro Ferrara: "Automatic Inference of Access Permissions"

Sémantique et Interprétation Abstraite, École Normale Supérieure, Paris, France


Permission Transfer

- Permissions can be transferred through:
 - > Fork and join of threads
 - Fork: exhale the pre-condition of the run method
 - Join: inhale its post-condition

```
class Node {  
  int value;  
  void update(int x)  
  requires acc(value)  
  ensures acc(value) {  
    this.value=x;  
  }  
}
```

```
void main() {  
  Node n = new Node();  
  t1=fork n.update(10);  
  t2=fork n.update(5);  
}
```

We own 0%!
Method call
not allowed!



Permission Transfer

- Permissions can be transferred through:
 - > Fork and join of threads
 - Fork: exhale the pre-condition of the run method
 - Join: inhale its post-condition

```
class Node {
  int value;
  void update(int x)
  requires acc(value)
  ensures acc(value) {
    this.value=x;
  }
}
```

```
void main() {
  Node n = new Node();
  t1=fork n.update(10);
  join t1;
  t2=fork n.update(5);
  join t2;
}
```

Example

- Permissions can be transferred through:
 - > Locks
 - Acquire: inhale the monitor invariant
 - Release: exhale the monitor invariant

```
class Node {
    int value;
    invariant acc(value);
    void update(int x) {
        acquire this;
        this.value=x;
        release this;
    }
}

void main() {
    Node n = new Node();
    share n;
    t1=fork n.update(10);
    t2=fork n.update(5);
}
```

Owicki-Gries Example

```
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

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

```
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)

Outline

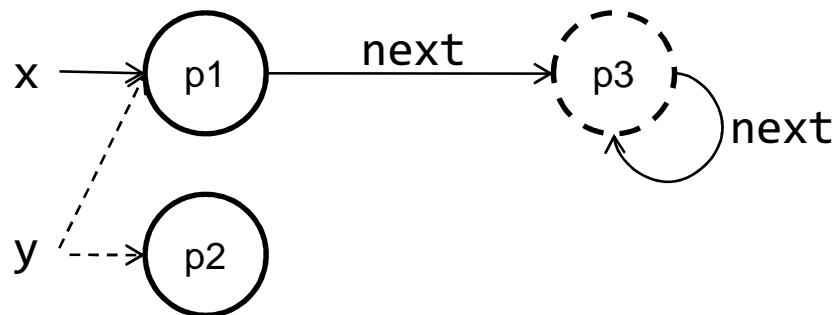
1. Access permissions
2. Chalice
3. Abstract domain
4. Inference of permissions
5. Sample and experimental results
6. Conclusion

Goal of the Analysis

- Program without annotation on permissions
- Static analysis
 - > Infer permissions that could be specified
- Impose some constraints
 - > E.g., write requires level == 100%
- Solve the system
- Final goal:
 - > Numerical permissions
 - Enough to access heap locations

Heap Abstraction

- Not a contribution of our work
- Simple standard analysis
 - > Abstract heap nodes with program points
- Generic approach
 - > Apply other heap analyses, e.g., TVLA
 - > Given an heap access, it returns an heap id



Pietro Ferrara: “Automatic Inference of Access Permissions”

Sémantique et Interprétation Abstraite, École Normale Supérieure, Paris, France

Symbolic Permissions

- Different ways of specifying permissions
- Adopt symbolic values to represent them
 - > Pre-conditions: $Pre(C, m, p.f)$
 - Method m in class C over path $p.f$
 - > Post-conditions: $Post(C, m, p.f)$
 - Method m in class C over path $p.f$
 - > Monitor invariants: $MI(C, p.f)$
 - Class C over path $p.f$
- Set of all symbolic values: \overline{SV}
- ≤ 100 and ≥ 0

Pietro Ferrara: "Automatic Inference of Access Permissions"

Sémantique et Interprétation Abstraite, École Normale Supérieure, Paris, France

Symbolic Levels

- Inhale (add) and exhale (subtract)
 - > several times
 - > on the same location
 - > during the execution of a program
- Summation of symbolic permissions
 - > At a given point of the program
 - > For each abstract heap location
$$\overline{AV} = \left\{ \sum_i a_i * s_i + c \text{ where } a_i, c \in \mathbb{Z}, s_i \in \overline{SV} \right\}$$
- Permissions we **surely** own

Lattice Structure

$$(\sum_j a_j^1 * \bar{s}_j + c_1) \leq_{\overline{AV}} (\sum_j a_j^2 * \bar{s}_j + c_2) = \text{true}$$

$$\Updownarrow$$

$$c_1 \geq c_2 \wedge \forall j : a_j^1 \geq a_j^2$$

$$\begin{aligned} (\sum_j a_j^1 * \bar{s}_j + c_1) \sqcup_{\overline{AV}} (\sum_j a_j^2 * \bar{s}_j + c_2) &= \\ &= (\sum_j \min(a_j^1, a_j^2) * \bar{s}_j + \min(c_1, c_2)) \end{aligned}$$

$$\begin{aligned} (\sum_j a_j^1 * \bar{s}_j + c_1) \sqcap_{\overline{AV}} (\sum_j a_j^2 * \bar{s}_j + c_2) &= \\ &= (\sum_j \max(a_j^1, a_j^2) * \bar{s}_j + \max(c_1, c_2)) \end{aligned}$$

What could be specified?

- We do not have annotation
 - > Inhale/exhale everything is “reachable”
- E.g., method call `x.m(y)`:
 - > Permissions on any reachable location from
 - `x`
 - `y`
 - > `x` renamed to `this`
 - > `y` renamed to `a1` (1st argument of `m`)
 - > Many (infinite?) paths to reach a location

Reachability Analysis

$$\overline{reach} : (\overline{L} \times \overline{H} \times \wp(\overline{L}) \times \text{Path}) \rightarrow \wp(\overline{L} \times \overline{F} \times \text{Path})$$

$$\overline{reach}(\overline{r}, \overline{h}, \overline{R}, p) = \{(\overline{r}_1, f, p) : (\overline{r}_1, f) \in \overline{reach1}(\overline{h}, p) \wedge \overline{r}_1 \notin \overline{R}\} \cup \\ \cup \{(\overline{r}_2, f_1, p_1) \in \overline{reach}(\overline{r}_1, \overline{h}, \overline{R} \cup \downarrow_1(\overline{reach1}(\overline{h}, p))), p.f) : (\overline{r}_1, f) \in \overline{reach1}(\overline{h}, p)\}$$

$$\overline{reach1} : (\overline{H} \times \text{Path}) \rightarrow \wp(\overline{L} \times \overline{F})$$

$$\overline{reach1}(\overline{h}, p) = \{(\overline{r}, f) : \overline{\mathbb{E}}(\overline{h}, p.f) = \overline{r} \wedge f \in \overline{fields}(\overline{\mathbb{E}}(\overline{h}, p))\}$$

- *reach1*: what is reachable in one step
- *reach* iterates *reach1*
 - > Until we have visited all abstract locations
 - Suppose they are finite
 - > Take one of the shortest paths to reach it

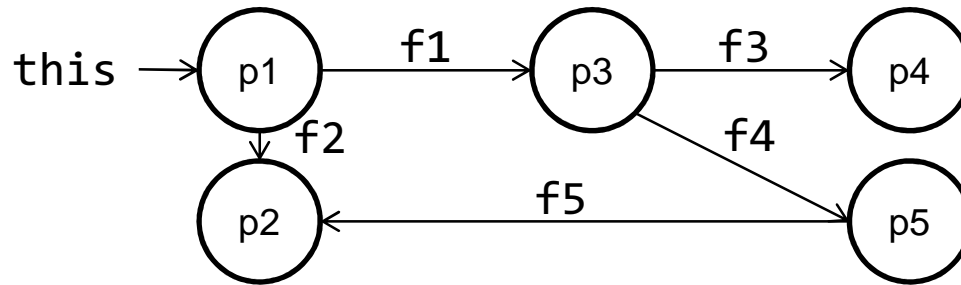
Reachability Analysis

$$\overline{reach} : (\overline{L} \times \overline{H} \times \wp(\overline{L}) \times \text{Path}) \rightarrow \wp(\overline{L} \times \overline{F} \times \text{Path})$$

$$\overline{reach}(\overline{r}, \overline{h}, \overline{R}, p) = \{(\overline{r}_1, f, p) : (\overline{r}_1, f) \in \overline{reach1}(\overline{h}, p) \wedge \overline{r}_1 \notin \overline{R}\} \cup \\ \cup \{(\overline{r}_2, f_1, p_1) \in \overline{reach}(\overline{r}_1, \overline{h}, \overline{R} \cup \downarrow_1(\overline{reach1}(\overline{h}, p))), p.f) : (\overline{r}_1, f) \in \overline{reach1}(\overline{h}, p)\}$$

$$\overline{reach1} : (\overline{H} \times \text{Path}) \rightarrow \wp(\overline{L} \times \overline{F})$$

$$\overline{reach1}(\overline{h}, p) = \{(\overline{r}, f) : \mathbb{E}(\overline{h}, p.f) = \overline{r} \wedge f \in \overline{fields}(\mathbb{E}(\overline{h}, p))\}$$



$$\overline{reach}(p1, \overline{h}, \emptyset, \text{this}) = \{(p2, f2, \text{this}), (p3, f1, \text{this})\} \\ \cup \{(p4, f3, \text{this.f1}), (p5, f4, \text{this.f1})\}$$

Abstract Semantics

- Inhale/exhale following Chalice semantics
- Rely on the reachability analysis
 - > Inhale/exhale what is returned by \overline{reach}
 - > Local paths injected through \overline{rep}

$$\overline{S}(x := \text{new } T, \overline{\sigma}, \overline{h}) = \overline{\sigma}[\overline{r} \mapsto 100 : (\overline{r}, p) \in \overline{reach1}(\overline{h}', x)]$$

where \overline{h}' is the abstract heap obtained after $x := \text{new } T$

$$\overline{S}(x.m(), \overline{\sigma}, \overline{h}) = \overline{\sigma}_2 : \overline{\sigma}_1 = \overline{exh}(\sigma, \overline{rep}(\overline{reach}(\overline{\mathbb{E}}(\overline{h}, x), \overline{h}, \emptyset, \text{this}), \text{Pre}(\text{class}(\overline{\mathbb{E}}(\overline{h}, x)), m, \emptyset))) \wedge$$
$$\overline{\sigma}_2 = \overline{inh}(\overline{\sigma}_1, \overline{rep}(\overline{reach}(\overline{\mathbb{E}}(\overline{h}, x), \overline{h}, \emptyset, \text{this}), \text{Post}(\text{class}(\overline{\mathbb{E}}(\overline{h}, x)), m, \emptyset)))$$

$$\overline{S}(\text{acquire } x, \overline{\sigma}, \overline{h}) = \overline{inh}(\sigma, \overline{rep}(\overline{reach}(\overline{\mathbb{E}}(\overline{h}, x), \overline{h}, \emptyset, \text{this}), \text{MI}(\text{class}(\overline{\mathbb{E}}(\overline{h}, x)), \emptyset)))$$

$$\overline{S}(\text{release } x, \overline{\sigma}, \overline{h}) = \overline{exh}(\sigma, \overline{rep}(\overline{reach}(\overline{\mathbb{E}}(\overline{h}, x), \overline{h}, \emptyset, \text{this}), \text{MI}(\text{class}(\overline{\mathbb{E}}(\overline{h}, x)), \emptyset)))$$

Owicki-Gries Example

```
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;
}
```

$c.* \rightarrow 100$

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

```
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;
  }
}
```

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

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

Pietro Ferrara: "Automatic Inference of Access Permissions"

Sémantique et Interprétation Abstraite, École Normale Supérieure, Paris, France

Outline

1. Access permissions
2. Chalice
3. Abstract domain
4. Inference of permissions
5. Sample and experimental results
6. Conclusion

Constraint Inference

- Accessing locations imposes constraints:
 - > $\geq \epsilon$ when reading, $= 100$ when writing
- Inhaling and exhaling as well:
 - > ≥ 0 when exhaling, ≤ 100 when inhaling
- $\forall \bar{s} \in \overline{SV} : 0 \leq \bar{s} \leq 100$
- Symbolic level \bar{s} at the exit point
 - > Impose $Post(C, m, p.f) = \bar{s}$
 - > It would be enough $Post(C, m, p.f) \leq \bar{s}$
 - More permissions returned by the method

Owicki-Gries Example

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

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

```
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;
  }
```

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

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

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

$$1 * \text{Post}(W1, \text{Inc}, c.c1) \geq \epsilon$$

$$1 * \text{Pre}(W1, \text{Inc}, c.x) + 1 * MI(\text{Cell}, x) = 100$$

$$1 * \text{Pre}(W1, \text{Inc}, c.c1) + 1 * MI(\text{Cell}, c1) = 100$$

$$1 * MI(\text{Cell}, *) \geq \epsilon$$

$$100 - 1 * MI(\text{Cell}, *) - 1 * \text{Pre}(W1, \text{Inc}, c.*) - 1 * \text{Pre}(W2, \text{Inc}, c.*) \geq 0$$

Pietro Ferrara: "Automatic Inference of Access Permissions"

Sémantique et Interprétation Abstraite, École Normale Supérieure, Paris, France

Linear Programming

$$1 * Post(W1, Inc, c.c1) \geq \epsilon$$

$$1 * Pre(W1, Inc, c.x) + 1 * MI(Cell, x) = 100$$

$$1 * Pre(W1, Inc, c.c1) + 1 * MI(Cell, c1) = 100$$

$$1 * MI(Cell, *) \geq \epsilon$$

$$100 - 1 * MI(Cell, *) - 1 * Pre(W1, Inc, c.*) - 1 * Pre(W2, Inc, c.*) \geq 0$$

- System solved using linear programming

- Objective function:

> $\sum_i n_i * s_i$ where

- n is an integer coefficient
- s is a symbolic value

> Minimize or maximize

Priorities and Objective Function

- We minimize the objective function
 - > We want the minimal levels
 - to own enough permissions
 - > Otherwise 100% on what is not accessed!
- In the objective function:
 - > Smaller coefficient → higher priority
- Two types of annotation:
 - > Pre- and post-conditions
 - > Monitor invariants

Epsilon's Representation

- Special symbolic value for ϵ
 - > $0 < \epsilon < 1$
 - > $n * \epsilon < 0.5$
 - where n is the maximal coefficient for ϵ
- Numerical value $n.f$ represents:
 - > $n\% + f/\epsilon$ if $f < 0.5$
 - > $(n + 1)\% - (1 - f)/\epsilon$ if $f > 0.5$
- Negative coefficient in the obj function
 - > Maximize its value
 - Avoid rounding problems

Pietro Ferrara: "Automatic Inference of Access Permissions"

Sémantique et Interprétation Abstraite, École Normale Supérieure, Paris, France

Output

- Two possible outputs:
 - > The system is infeasible, because of
 - Approximation (false alarm)

- “Wrong” program
acquire this;

> The system is feasible

```
class Node {  
    if(i>j) this.x=5;  
    int y;  
    void update(int x) {  
        }  
}  
void main() {  
    Node n = new Node();  
    t1=fork n.update(10);  
    t2=fork n.update(5);  
}
```

- Numerical access permissions
Can be translated to contracts

- Through them we obtain loop invariants

$$1 * Pre(Node, update, x) = 100$$

$$100 - 2 * Pre(Node, update, x) \geq 0$$

Loop Invariant

- Entry and exit abstract states for each loop
 - > Obtained through the static analysis
 - > Sum of symbolic permissions
- If the system is feasible
 - > Numerical value for each symbolic permission
- Compute loop invariants
 - > Instead of using other symbolic values
 - Propagate what we own
 - ... not infer what we need!

Owicki-Gries Example

$$1 * Post(W1, Inc, c.c1) \geq \epsilon$$

$$1 * Pre(W1, Inc, c.x) + 1 * MI(Cell, x) = 100$$

$$1 * Pre(W1, Inc, c.c1) + 1 * MI(Cell, c1) = 100$$

$$1 * MI(Cell, *) \geq \epsilon$$

$$100 - 1 * MI(Cell, *) - 1 * Pre(W1, Inc, c.*) - 1 * Pre(W2, Inc, c.*) \geq 0$$

- Three possible scenarios:
 - > Maximize monitor invariants

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

Pietro Ferrara, Automatic Inference of Access Permissions

Owicki-Gries Example

$$1 * Post(W1, Inc, c.c1) \geq \epsilon$$

$$1 * Pre(W1, Inc, c.x) + 1 * MI(Cell, x) = 100$$

$$1 * Pre(W1, Inc, c.c1) + 1 * MI(Cell, c1) = 100$$

$$1 * MI(Cell, *) \geq \epsilon$$

$$100 - 1 * MI(Cell, *) - 1 * Pre(W1, Inc, c.*) - 1 * Pre(W2, Inc, c.*) \geq 0$$

- Three possible scenarios:
 - > 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- ϵ
$MI(Cell, Inc, c1)$	ϵ
$MI(Cell, Inc, x)$	100

Peter Chen: Automatic Inference of Access Permissions

Owicki-Gries Example

$$1 * Post(W1, Inc, c.c1) \geq \epsilon$$

$$1 * Pre(W1, Inc, c.x) + 1 * MI(Cell, x) = 100$$

$$1 * Pre(W1, Inc, c.c1) + 1 * MI(Cell, c1) = 100$$

$$1 * MI(Cell, *) \geq \epsilon$$

$$100 - 1 * MI(Cell, *) - 1 * Pre(W1, Inc, c.*) - 1 * Pre(W2, Inc, c.*) \geq 0$$

- Three possible scenarios:
 - > Distribute equally the permissions

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
$MI(Cell, Inc, c1)$	50
$MI(Cell, Inc, x)$	100

Pietro Ferrara, Automatic Inference of Access Permissions

Unsoundness

- Improve the amount of inferred contracts
 - > If summary node → cannot inhale
 - > If two arguments of a method have compatible types → aliases, summary node
- Unsoundness:
 - > Inhale even if summary node
 - > Arguments cannot be aliases
- Better results in practice
- I think we can obtain similar results with better heap abstractions, e.g., TVLA

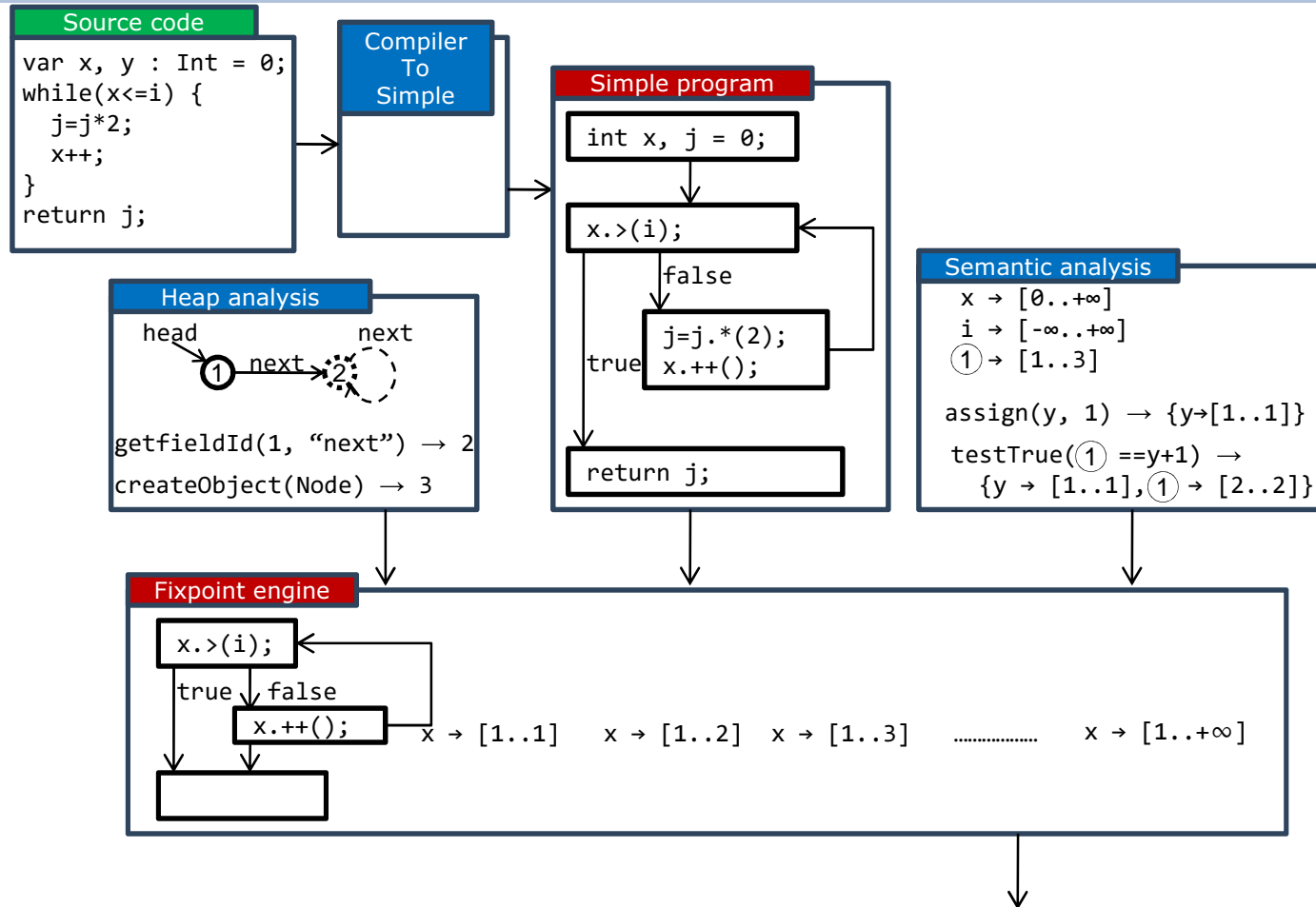
Pietro Ferrara: "Automatic Inference of Access Permissions"

Sémantique et Interprétation Abstraite, École Normale Supérieure, Paris, France

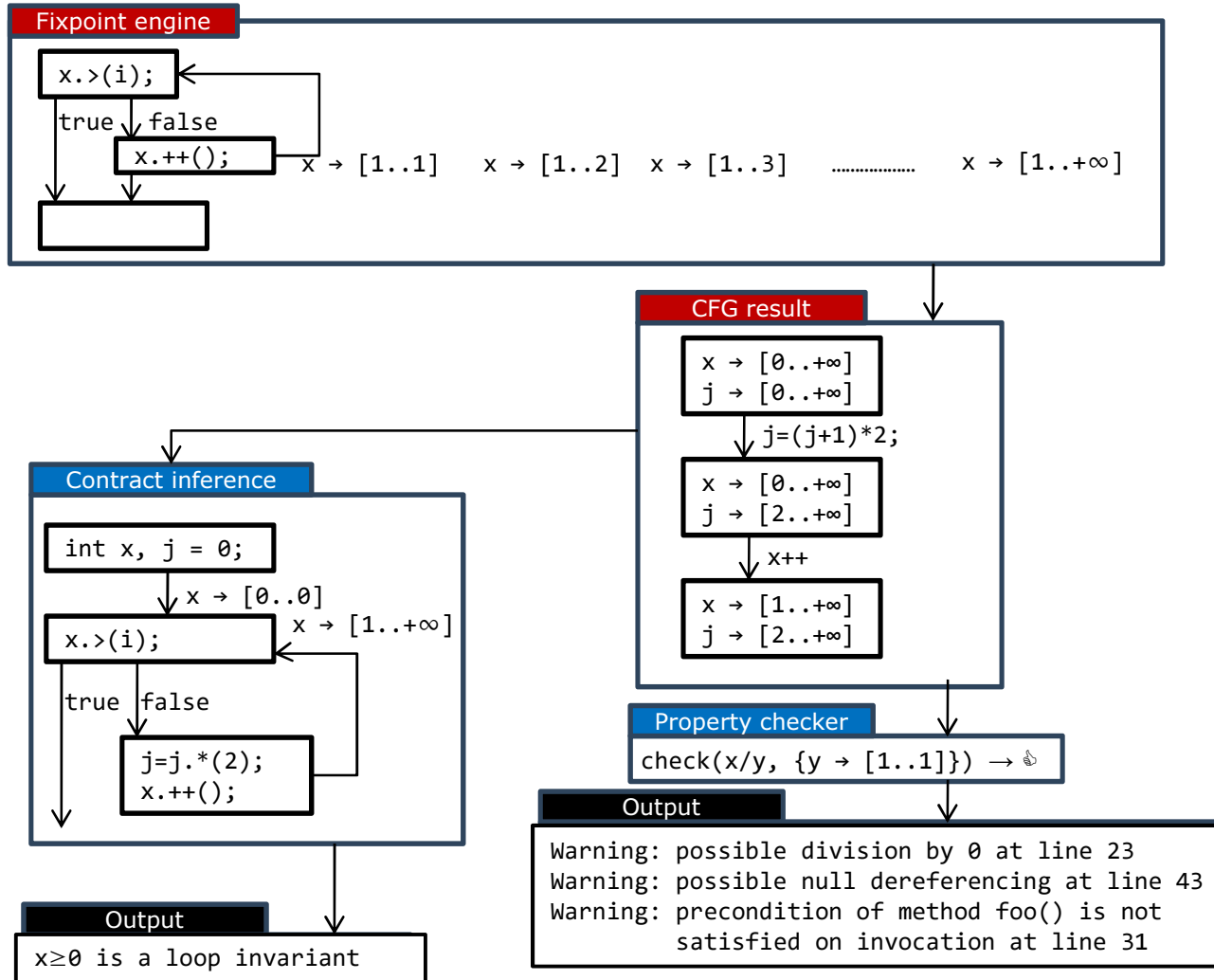
Outline

1. Access permissions
2. Chalice
3. Abstract domain
4. Inference of permissions
5. Sample and experimental results
6. Conclusion

Sample's Structure



Sample's Structure



Pietro Ferrara: "Automatic Inference of Access Permissions"

Sémantique et Interprétation Abstraite, École Normale Supérieure, Paris, France

Experimental Results

- Analysis implemented in Sample
- Executed on a
 - > Intel Core 2 Quad CPU 2.83 Ghz
 - > 4 GB RAM
 - > Windows 7
- Applied to several case studies
 - > Chalice /example directory
 - > Chalice's tutorial
 - K. R. M. Leino, P. Mueller, and J. Smans. Verification of concurrent programs with Chalice. In FOSAD '09, pages 195-222. Springer, 2009.

Pietro Ferrara: "Automatic Inference of Access Permissions"

Sémantique et Interprétation Abstraite, École Normale Supérieure, Paris, France

Experimental Results

Program	#C	Sound		Uns.entry		Uns.inhale		Unsound	
		#C	t	#C	t	#C	t	#C	t
Fig1	4	3	166	✗	173	3	169	<u>4</u>	181
Fig2	3	1	122	<u>3</u>	125	2	123	<u>3</u>	124
Fig3	4	2	74	<u>4</u>	76	3	77	<u>4</u>	79
Fig4	4	✗	132	✗	131	3	131	<u>4</u>	134
Fig5	3	✗	220	✗	219	2	227	<u>3</u>	251
Fig6	14	7	140	<u>14</u>	143	7	142	<u>14</u>	143
Fig11	7	3	117	<u>7</u>	122	<u>7</u>	114	<u>7</u>	117
Fig12	7	3	109	<u>7</u>	119	<u>7</u>	115	<u>7</u>	118
Fig13	12	5	354	<u>12</u>	810	5	359	<u>12</u>	802
AssociationList	11	1	1271	✗	2056	3	1266	✗	2063
cell – defaults	25	✗	221	✗	234	14	222	<u>25</u>	245
HandOverHand	25	6	1635	✗	2752	8	1632	✗	2788
linkedlist	4	2	271	<u>4</u>	910	2	274	<u>4</u>	918
swap	4	2	95	<u>4</u>	110	2	110	<u>4</u>	113

Pietro Ferrara: “Automatic Inference of Access Permissions”

Sémantique et Interprétation Abstraite, École Normale Supérieure, Paris, France

Outline

1. Access permissions
2. Chalice
3. Abstract domain
4. Inference of permissions
5. Sample and experimental results
6. Conclusion

Conclusion

- Analysis to infer symbolic permissions
- System of constraints
 - > Imposed by the semantics
- Solved using linear programming
 - > Many possible solutions
 - Various priorities through the objective function
- Unsoundness to obtain more contracts
- Fast and precise

Future Work

- Automatic inference of the wait order
- Intuition
 - > Monitors, channels, threads
 - > Each time we wait, we have to:
 - Be below the level of what we will wait for
 - Raise our level to that one after the wait
- Order manually specified when sharing
- Can dynamically change
- Avoid deadlock

Bibliography

- K. R. M. Leino and P. Müller. A basis for verifying multi-threaded programs. In *ESOP '09*
 - > The initial paper on Chalice
 - > Sharing, monitors, join/fork, abstract predicates
- K. R. M. Leino, P. Müller, and J. Smans. Deadlock-free channels and locks. In *ESOP '10*
 - > Channels, wait order
- K. R. M. Leino, P. Müller, and J. Smans. Verification of concurrent programs with Chalice. In *FOSAD '09*
 - > Chalice by examples