Static Scoping and Name Resolution for Mobile Processes with Polymorphic Interfaces

Matt B. Pedersen & Matthew Sowders

University of Nevada, Las Vegas
What Have we Done?

- Added mobile processes to ProcessJ
  - With polymorphic interfaces
    - Multiple interfaces to the same process
    - Different set of formal parameters per interface

```java
mobile void foo (int x, int y) {
    ...  
    while (...) {
        ...  
        suspend resume with (int z)  
        ...  
    }
}
```
Polymorphic Interfaces

```java
mobile void foo (int x, int y) {
    ...
    while (...) {
        ...
        suspend resume with (int z)
        ...
    }
}
```

- `foo` has 2 interfaces
  - `(int x, int y)`
  - `(int z)`
- When `foo` is started `(int x, int y)` is used; subsequently `(int z)` is used.
Why Have we Done That?

MOBILE PROC reindelf (CHAN AGENT.INITIALIZE initialize?,
    SHARED CHAN AGENT.MESSAGE report!,
    SHARED CHAN INT santa.a!, santa.b!)
    IMPLEMENTS AGENT

... local state declarations
SEQ
... in station compound (initialise local state)
WHILE TRUE
SEQ
... in station compound
SUSPEND -- move to gathering place
... in the gathering place
SUSPEND -- move to santa’s grotto
... in santa’s grotto
SUSPEND -- move to compound

From: Santa Claus – with mobile reindeer and elves, CPA Fringe presentation 2008
MOBILE PROC reindef

(CHAN AGENT.INITIALIZE initialize?,
  SHARED CHAN AGENT.MESSAGE report!,
  SHARED CHAN INT santa.a!, santa.b!)

IMPLEMENTS AGENT

These are all the same interface:

(CHAN AGENT.INITIALIZE initialize?,
  SHARED CHAN AGENT.MESSAGE report!,
  SHARED CHAN INT santa.a!, santa.b!)

... local state declarations

SEQ

... in station compound (initialise local state)

WHILE TRUE

SEQ

... in station compound

SUSPEND -- move to gathering place

... in the gathering place

SUSPEND -- move to santa’s grotto

... in santa’s grotto

SUSPEND -- move to compound

:
Why Have we Done That?

MOBILE PROC reindelf (CHAN AGENT.INITIALIZE initialize?,
SHARED CHAN AGENT.MESSAGE report!,
SHARED CHAN INT santa.a!, santa.b!)
IMPLEMENTS AGENT

... local state declarations
SEQ
... in station compound (initialise local state)
WHILE TRUE
SEQ
... in station compound
SUSPEND -- move to gathering place
... in the gathering place
SUSPEND -- move to santa’s grotto
... in santa’s grotto
SUSPEND -- move to compound

The Initialize channel is only used in

... local state declaration

Subsequent re-animations of reindelf must thus provide ‘dummy’ values for this channel.
Why Have we Done That?

- Channel ends (or other parameters) not used in code following a resumption must still be passed
  - A dummy reading end passed could cause deadlock if ever read.
  - Made up actual parameter values must be passed to satisfy the compiler.
Ok, so now what?

- What is the semantics of this?
  - Parameters do not retain their values between invocations.

```java
mobile void foo (int x, int y) {
  B_1
  while (B_2) {
    B_3
    suspend resume with (int z)
    B_4
  }
  B_5
}
```

x & y can be referenced in $B_1$ (first invocation); z in $B_4$ (subsequent invocations)
Invocation of Mobiles

```java
mobile void foo (int x, int y) {
    B1
    while (B2) {
        B3
        suspend resume with (int z)
        B4
    }
    B5
}
```

- Example of execution of foo:
  foo(4,5); foo(4), foo(5), foo(7), ....
- Only the first time (when foo is started) is the procedure interface used.
- All other resumptions use the suspend/resume interface.
Possible Executions

```c
mobile void foo (int x, int y) {
    B_1
    while (B_2) {
        B_3
        suspend resume with (int z)
        B_4
    }
    B_5
}
```

foo(x,y): $B_1, B_2, B_5$, done!

foo(x,y): $B_1, B_2, B_3$, suspend/foo(z): $B_4, B_2, B_5$

We see that e.g. $B_2$ ($&$ $B_5$) can be executed ‘with’ both $x$ & $y$ as well as $z$. 
Possible Executions

```c
mobile void foo (int x, int y) {
    B_1
    while (B_2) {
        B_3
        suspend resume with (int z)
        B_4
    }
    B_5
}
```

The first time $B_2$ is executed $x$ seems to be ‘a valid parameter’, but the second time it does not; only $z$ does.
What should we do?

- Determine which parameters can be referenced in all program blocks
  - Create a control flow graph (CFG) based on the source
  - Massage it a little
  - Perform an analysis using In and Out sets (to be defined shortly)
Remember:
The following concerns parameters only; no local variables are considered (yet)
Remember: Only the parameters from the most recent invocation may be referenced
Initial Control Flow Graph

```
mobile void foo (int x, int y) {
    B₁
    while (B₂) {
        B₃
        suspend resume with (int z)
        B₄
    }
    B₅
}
```

I₀ represents the original interface:
```
foo (int x, int y)
```

I₁ represents the resume interface:
```
foo (int z)
```
Interfaces are separated out and given their own nodes
Transformed CFG

```plaintext
mobile void foo (int x, int y) {
    B_1
    while (B_2) {
        B_3
        suspend resume with (int z)
        B_4
    }
    B_5
}
```
CFG with interface information

\[
\{(x \text{ int } l_0) \ (y \text{ int } l_0)\}
\]

\[
\{(z \text{ int } l_1)\}
\]
Interface Nodes

\[ I_0 \{(x \text{ int } I_0) (y \text{ int } I_0)\} \]

\[ I_1 \{(z \text{ int } I_1)\} \]
So Far So Good

- Let us define In and Out sets (loosely):
  - For interface nodes:
    - In Set: Not interesting as an interface defines a new set of **parameters**
    - Out Set: The set of **parameters** defined by the interface
  - For Code nodes:
    - In Set: The set of **parameters** that can be referenced in the node (at least for the final generation of In set)
    - Out Set: a copy of the In set
Generation 0 In and Out Sets

- **Interface Nodes**
  - \( \text{In}_0(l_i) = \{ \} \)
  - \( \text{Out}_0(l_i) = \{(n_{i,1}, t_{i,1}, l_i) \ldots (n_{i,k_i}, t_{i,k_i}, l_i)\} \)
    - The Outset of an interface is the set of triples (name type interface) defined by it

- **Code Nodes**
  - \( \text{In}_0(B_j) = \{ \} \)
  - \( \text{Out}_0(B_j) = \{ \} \)
Example

```plaintext
mobile void foo (int x, int y) {
    B_1
    while (B_2) {
        B_3
        suspend resume with (int z)
        B_4
    }
    B_5
}
```

- For Example
  - $Out_0(l_0) = \{ (x \text{ int } l_0) \ (y \text{ int } l_0) \}$
  - $Out_0(l_1) = \{ (z \text{ int } l_1) \}$
In and Out Sets

- We generate generations of these sets until no sets change, after which we have the set of parameters that can be referenced for a node in its In set
  - We start out with empty sets except for Out sets of interface nodes
CFG with In and Out Sets (Gen. 0)

- $\text{In}(I_0) = \{ \}$
  $\text{Out}(I_0) = \{ (x \text{ int } I_0), (y \text{ int } I_0) \}$
- $\text{In}(B_1) = \{ \}$
  $\text{Out}(B_1) = \{ \}$
- $\text{In}(B_2) = \{ \}$
  $\text{Out}(B_2) = \{ \}$
- $\text{In}(B_3) = \{ \}$
  $\text{Out}(B_3) = \{ \}$
- $\text{In}(B_4) = \{ \}$
  $\text{Out}(B_4) = \{ \}$
- $\text{In}(B_5) = \{ \}$
  $\text{Out}(B_5) = \{ \}$
- $\text{In}(I_1) = \{ \}$
  $\text{Out}(I_1) = \{ (z \text{ int } I_1) \}$
Generation k+1 (Interface Nodes)

- $\text{In}_{k+1}(I_i) = \{ \}$
  - Interface nodes define a new interface, In sets can be ignored.

- $\text{Out}_{k+1}(I_i) = \text{Out}_k(I_i) = \{(n_{i,1}, t_{i,1} \bot I_i) \ldots (n_{i,k_i}, t_{i,k_i} \bot I_i)\}$
  - Interface nodes always define the same interface.
Generation $k+1$ (Code Nodes)

- $\text{In}_{k+1}(B_j) = \bigcap_{(N,B_i) \in E_{\text{CFG}}} \text{Out}_k(N)$
  - New In set is the **intersection** of all the Out sets of the code node’s **predecessors** in the CFG

- $\text{Out}_{k+1}(B_j) = \text{In}_{k+1}(B_j)$
  - The Out set of a code node is the same as its In set, as it cannot define a new interface (Technically not needed but nice to have)
Generation $k+1$ (Code Nodes)

- $\text{In}_{k+1}(B_j) = \bigcap_{(N,B_i) \in E_{\text{CFG}}} \text{Out}_k(N)$
  - New In set is the **intersection** of all the Out sets of the code nodes **predecessors** in the CFG

- $\text{Out}_{k+1}(B_j) = \text{In}_{k+1}(B_j)$
  - The Out set of a code node is the same as its In set, as it cannot define a new interface

\[(n_i \ t_i \ I_i) \equiv (n_j \ t_j \ I_i) \iff (n_i \equiv n_j) \land (t_i \equiv t_j)\]
Generation 1 Sets

In(I_0) = \{ \}  
Out(I_0) = \{(x \text{ int } I_0) (y \text{ int } I_0)\}

In(B_1) = \{(x \text{ int } I_0) (y \text{ int } I_0)\}  
Out(B_1) = \{(x \text{ int } I_0) (y \text{ int } I_0)\}

In(B_4) = \{ \}  
Out(B_4) = \{ \}

In(B_2) = \{ \}  
Out(B_2) = \{ \}

In(I_1) = \{ \}  
Out(I_1) = \{(z \text{ int } I_1)\}

In(B_3) = \{ \}  
Out(B_3) = \{ \}

In(B_5) = \{ \}  
Out(B_5) = \{ \}
No Changes in $B_2$ since
$\text{In}(B_2) = \text{Out}(B_1) \cap \text{Out}(B_4)$
$= \emptyset \cap \{(x \text{ int } l_0) \ (y \text{ int } l_0)\}$

$\text{In}(B_4) = \emptyset$
$\text{Out}(B_4) = \emptyset$

$\text{In}(B_1) = \{(x \text{ int } l_0) \ (y \text{ int } l_0)\}$
$\text{Out}(B_1) = \{(x \text{ int } l_0) \ (y \text{ int } l_0)\}$

$\text{In}(I_1) = \emptyset$
$\text{Out}(I_1) = \{(z \text{ int } I_1)\}$

$\text{In}(B_3) = \emptyset$
$\text{Out}(B_3) = \emptyset$

$\text{In}(B_5) = \emptyset$
$\text{Out}(B_5) = \emptyset$
Generation I Sets

**I₀**
- \( \text{In}(I₀) = \emptyset \)
- \( \text{Out}(I₀) = \{(x \text{ int } I₀) (y \text{ int } I₀)\} \)

**B₁**
- \( \text{In}(B₁) = \{(x \text{ int } I₀) (y \text{ int } I₀)\} \)
- \( \text{Out}(B₁) = \{(x \text{ int } I₀) (y \text{ int } I₀)\} \)

**B₄**
- \( \text{In}(B₄) = \emptyset \)
- \( \text{Out}(B₄) = \emptyset \)

**I₁**
- \( \text{In}(I₁) = \emptyset \)
- \( \text{Out}(I₁) = \{(z \text{ int } I₁)\} \)

**B₂**
- \( \text{In}(B₂) = \emptyset \)
- \( \text{Out}(B₂) = \emptyset \)

**B₃**
- \( \text{In}(B₃) = \emptyset \)
- \( \text{Out}(B₃) = \emptyset \)

**B₅**
- \( \text{In}(B₅) = \emptyset \)
- \( \text{Out}(B₅) = \emptyset \)
Generation 1 Sets

\[
\text{In}(I_0) = \{\}
\]
\[
\text{Out}(I_0) = \{(x \text{ int } I_0) (y \text{ int } I_0)\}
\]

\[
\text{In}(B_1) = \{(x \text{ int } I_0) (y \text{ int } I_0)\}
\]
\[
\text{Out}(B_1) = \{(x \text{ int } I_0) (y \text{ int } I_0)\}
\]

\[
\text{In}(B_2) = \{\}
\]
\[
\text{Out}(B_2) = \{\}
\]

\[
\text{In}(B_3) = \{\}
\]
\[
\text{Out}(B_3) = \{\}
\]

\[
\text{In}(B_4) = \{(z \text{ int } I_1)\}
\]
\[
\text{Out}(B_4) = \{(z \text{ int } I_1)\}
\]

\[
\text{In}(B_5) = \{\}
\]
\[
\text{Out}(B_5) = \{\}
\]

\[
\text{In}(I_1) = \{\}
\]
\[
\text{Out}(I_1) = \{(z \text{ int } I_1)\}
\]
Generation 1 Sets

In(I_0) = \{ \} 
Out(I_0) = \{(x \text{ int } I_0) (y \text{ int } I_0)\}

In(B_1) = \{(x \text{ int } I_0) (y \text{ int } I_0)\} 
Out(B_1) = \{(x \text{ int } I_0) (y \text{ int } I_0)\}

In(B_2) = \{\} 
Out(B_2) = \{\}

In(B_3) = \{\} 
Out(B_3) = \{\}

In(B_4) = \{(z \text{ int } I_1)\} 
Out(B_4) = \{(z \text{ int } I_1)\}

In(B_5) = \{\} 
Out(B_5) = \{\}
In & Out Sets after Generation 1

- **I₀**
  - In(I₀) = {}  
  - Out(I₀) = {(x int I₀) (y int I₀)}

- **B₁**
  - In(B₁) = {(x int I₀) (y int I₀)}  
  - Out(B₁) = {(x int I₀) (y int I₀)}

- **B₂**
  - In(B₂) = {}  
  - Out(B₂) = {}  

- **B₃**
  - In(B₃) = {}  
  - Out(B₃) = {}  

- **B₄**
  - In(B₄) = {(z int I₁)}  
  - Out(B₄) = {(z int I₁)}

- **B₅**
  - In(B₅) = {}  
  - Out(B₅) = {}
Generation 2 In and Out Sets

- Nothing changes when computing generation 2.
Final In and Out Sets

- $\text{In}(I_0) = \{\}$
  $\text{Out}(I_0) = \{(x \text{ int } I_0) (y \text{ int } I_0)\}$

- $\text{In}(B_1) = \{(x \text{ int } I_0) (y \text{ int } I_0)\}$
  $\text{Out}(B_1) = \{(x \text{ int } I_0) (y \text{ int } I_0)\}$

- $\text{In}(B_4) = \{(z \text{ int } I_1)\}$
  $\text{Out}(B_4) = \{(z \text{ int } I_1)\}$

- $\text{In}(B_2) = \{\}$
  $\text{Out}(B_2) = \{\}$

- $\text{In}(I_1) = \{\}$
  $\text{Out}(I_1) = \{(z \text{ int } I_1)\}$

- $\text{In}(B_3) = \{\}$
  $\text{Out}(B_3) = \{\}$

- $\text{In}(B_5) = \{\}$
  $\text{Out}(B_5) = \{\}$
Final Result of Analysis
Final Result of Analysis

- During all possible executions we can only guarantee that
  - $x$ and $y$ are always available in $B_1$
  - $z$ is always available in $B_4$
Name Resolution

- We have **not** considered local variables or how to resolve name usage in the code
  - For locals, regular scoping rules apply
  - Locals can hide parameters
  - One symbol table for an interface
  - One symbol table for its body
Name Resolution

- A suspend/resume point acts like the procedure interface
  - One symbol table for the interface
  - One symbol table for the implicit ‘body’ following
    - End of scope determined by the closest enclosing scope of the suspend/resume statement.

- A block { } and a for-statement opens a new scope as well
Now with Locals ;-)  

```c
mobile void foo (int x, int y) 
{
    int a;
    B_1
    while (B_2)
    {
        int q;
        B_3
        suspend resume with (int z)
        int w,z;
        B_4
    }  \textcolor{blue}{\textit{Implicit end of scopes opened by suspend}}
    B_5
} 
```
mobile void foo
    
    (int x, int y)  
    
    int a;  
    B₁  
    while (B₂)  
    
    int q;  
    B₃  
    suspend resume with  
    
    (int z)  
    
    int w,z;  
    B₄  

    }-T₄  

    }-T₃  

    }-T₂  

    }-T₁  

    }-T₀

Implicit scopes added in red

• Parameter scope for procedure interface (T₀)
• Parameter scope for suspend/resume interface (T₃)
• Body scope for suspend/resume interface (T₄)
Name Resolution

- A symbol Table now has an ‘access list’
  - Only name-uses in code blocks listed in the access list are allowed to perform a look up in the table – if not listed, move on to the parent table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>z</td>
</tr>
</tbody>
</table>

Access List: \{1,2,3,4,5\}
Symbol Tables

- Only symbol tables associated with the **implicit scopes** of interfaces have limited access lists; all others have full access.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>x</td>
</tr>
<tr>
<td>int</td>
<td>y</td>
</tr>
</tbody>
</table>

Access List: `{ 1 }`

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>z</td>
</tr>
</tbody>
</table>

Access List: `{ 4 }`
Errata: Access lists of $T_1$, $T_2$ & $T_4$ in the paper are incorrect. They read \{1,2,3,4\} they should be \{1,2,3,4,5\} (the 5 has gone missing)
The access list contains only 1, because \( x \) and \( y \) can only be referenced in \( B_1 \).
<table>
<thead>
<tr>
<th>N</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>x</td>
</tr>
<tr>
<td>int</td>
<td>y</td>
</tr>
<tr>
<td>{ 1 }</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>a</td>
</tr>
<tr>
<td>{ 1, 2, 3, 4, 5 }</td>
<td></td>
</tr>
</tbody>
</table>
$z$ from $I_1$ can only be referenced in $B_4$
Red arrows indicate in which table resolution starts.

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₀</td>
<td>int</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>int</td>
<td>y</td>
</tr>
<tr>
<td></td>
<td>{ 1 }</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₁</td>
<td>int</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>{ 1, 2, 3, 4, 5 }</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₂</td>
<td>int</td>
<td>q</td>
</tr>
<tr>
<td></td>
<td>{ 1, 2, 3, 4, 5 }</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₃</td>
<td>int</td>
<td>z</td>
</tr>
<tr>
<td></td>
<td>{ 4 }</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₄</td>
<td>int</td>
<td>w</td>
</tr>
<tr>
<td></td>
<td>int</td>
<td>z</td>
</tr>
<tr>
<td></td>
<td>{ 1, 2, 3, 4, 5 }</td>
<td></td>
</tr>
</tbody>
</table>
We get the following table of blocks and which parameters and locals they can reference.

<table>
<thead>
<tr>
<th>Block</th>
<th>Locals</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_1$</td>
<td>$a \in T_1$</td>
<td>$x \in T_0, y \in T_0$</td>
</tr>
<tr>
<td>$B_2$</td>
<td>$a \in T_1$</td>
<td></td>
</tr>
<tr>
<td>$B_3$</td>
<td>$q \in T_2, a \in T_1$</td>
<td></td>
</tr>
<tr>
<td>$B_4$</td>
<td>$w \in T_4, z \in T_4, q \in T_2, a \in T_1$</td>
<td>$z \in T_3$</td>
</tr>
<tr>
<td>$B_5$</td>
<td>$a \in T_1$</td>
<td></td>
</tr>
</tbody>
</table>

Errata: Table 4 Parameter for $B_4$ should read $z \in T_3$ and **not** $z \in T_4$
Conclusion

- We have defined and implemented mobile procedures with polymorphic interfaces in ProcessJ
- Provided a new scope resolution mechanism for polymorphic mobiles that performs correct name resolution
Conclusion

- Provided an implementation in Java/JCSP (ProcessJ translated to Java with JCSP)
  - Paper on the implementation (similar to our 2009 paper at CPA but without byte code rewriting) is being presented at PDPTA 2011 in July.

processj.cs.unlv.edu (currently turned off cause of hackers but we will be back up soon)
Questions or Lunch?