CS 476 – Programming Language Design

William Mansky
Java-Like Language: Semantics

• Values: ints, objects

• How should we represent an object?

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>5</td>
</tr>
<tr>
<td>getx()</td>
<td>return x;</td>
</tr>
<tr>
<td>sety(n)</td>
<td>y := n;</td>
</tr>
</tbody>
</table>

new Point(3, 5)
Java-Like Language: Semantics

class A extends Object {
    int x;
    A(int x) {
        super(); this.x = x;
    }
    int getx() {
        return this.x;
    }
}

class B extends A {
    int y;
    B(int x, int y) {
        super(x); this.y = y;
    }
    int gety() {
        return this.y;
    }
}
Java-Like Language: Semantics

• Values: ints, objects

• How should we represent an object?

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>5</td>
</tr>
<tr>
<td>getx()</td>
<td>return x;</td>
</tr>
<tr>
<td>sety(n)</td>
<td>y := n;</td>
</tr>
</tbody>
</table>

new Point(3, 5)
Java-Like Language: Semantics

\[
\begin{align*}
(e_1, \sigma) \downarrow v_1 \quad &\quad \ldots \quad (e_n, \sigma) \downarrow v_n \\
\text{(new } C(e_1, \ldots, e_n), \sigma) \downarrow \text{new } C(v_1, \ldots, v_n) \\
(e, \sigma) \downarrow \text{new } C(v_1, \ldots, v_n) \quad &\quad \text{fields}(C) = \tau_1 f_1, \ldots, \tau_n f_n \\
(e.f_i, \sigma) \downarrow v_i
\end{align*}
\]
Functions: Semantics of Calls

\[
\frac{(e_i, \sigma) \Downarrow v}{(x := f(e_1, \ldots, e_i, \ldots, e_n), \sigma) \rightarrow (x := f(e_1, \ldots, v, \ldots, e_n), \sigma)}
\]

\[
\frac{\sigma(f) = ((x_1, \ldots, x_n), c)}{(x := f(v_1, \ldots, v_n), k, \sigma) \rightarrow (c, (\sigma, x) :: k, \text{funs}(\sigma) \cup \{x_1 = v_1, \ldots, x_n = v_n\})}
\]
Java-Like Language: Semantics

\[
(e, \sigma) \downarrow v \\
(x = e. m(e_1, \ldots, e_n); , k, \sigma) \rightarrow (x = v. m(e_1, \ldots, e_n), k, \sigma)
\]

\[
(e_i, \sigma) \downarrow v \\
(x = e. m(e_1, \ldots, e_i, \ldots, e_n); , k, \sigma) \rightarrow (e. m(e_1, \ldots, v, \ldots, e_n), k, \sigma)
\]

\[
\text{methods}(C) = \cdots, \tau m(\tau_1 x_1, \ldots, \tau_n x_n)\{ c \} \\
(x = (\text{new } C(v_1, \ldots, v_n). m(u_1, \ldots, u_n); , k, \sigma) \rightarrow \\
(c, (\sigma, x) :: k, \{ \text{this } = \text{new } C(v_1, \ldots, v_n), x_1 = u_1, \ldots, x_n = u_n \})
\]
Java-Like Language: Contexts

• In our simple imperative language, declarations could be stored in the type context and the runtime state

\[
\text{funs}(\Gamma) \cup \{x_1 : \tau_1, \ldots, x_n : \tau_n\} \vdash c : \text{ok} \\
\Gamma \vdash \tau f(\tau_1 x_1, \ldots, \tau_n x_n)\{c\} : \Gamma[f \mapsto \tau(\tau_1 x_1, \ldots, \tau_n x_n)]
\]

\[
(\tau f(\tau_1 x_1, \ldots, \tau_n x_n)\{c\}, \sigma) \to (\text{skip}, \sigma[f \mapsto ((x_1, \ldots, x_n), c)])
\]

• In Java, both typing and semantics need the full declaration of each class

• We implicitly store class declarations in a class table
Java-Like Language: Contexts

• In Java, both typing and semantics need the full declaration of each class
• We implicitly store class declarations in a *class table*
• Java uses type information at runtime!
  — Every object is tagged with its class
  — The class determines which fields and methods are defined
  — The class determines which version of a method is called
• Whereas in C, types disappear at runtime
Java-Like Language: Contexts

• In Java, both typing and semantics need the full declaration of each class
• We implicitly store class declarations in a class table

$$\Gamma(C) = \text{class } C \text{ extends } D \{ \tau_1 f_1; \ldots; \tau_n f_n; \ldots \}$$

$$\text{fields}(\Gamma, C) = \text{fields}(\Gamma, D), \tau_1 f_1, \ldots, \tau_n f_n$$
Java-Like Language: Contexts

- In Java, both typing and semantics need the full declaration of each class
- We implicitly store class declarations in a class table

\[
\begin{align*}
\text{CT}(C) &= \text{class } C \text{ extends } D \{ \tau_1 f_1; \ldots; \tau_n f_n; \ldots \} \\
\text{fields}(C) &= \text{fields}(D), \tau_1 f_1, \ldots, \tau_n f_n
\end{align*}
\]

- CT is a global structure used by both the type system and the semantics, and computed in preprocessing
Java-Like Language: Casts

School s = new School();
s2 = w.build((Building) s);
((School) s2).getCourse();

where build takes a Building and School extends Building
Java-Like Language: Casts

School s = new School();
s2 = w.build((Building) s);  // upcast from School to Building
((School) s2).getCourse();  // downcast from Building to School

where build takes a Building and School extends Building
Java-Like Language: Casts

- Upcast: always safe, doesn’t do anything
- Downcast: safe only if object actually has the right type
- Other casts: ??

```java
B b = new B();
A a = (A) b;
```
Java-Like Language: Casts

• Upcast: always safe, doesn’t do anything
• Downcast: safe only if object actually has the right type
• Other casts: ??

B b = new B();
A a = (A) ((Object) b);
\[ \Gamma \vdash (C) \ e : C \]
Java-Like Language: Casts

• Upcast: always safe, doesn’t do anything
• Downcast: safe only if object actually has the right type
• Other casts: ??

B b = new B();
A a = (A) ((Object) b);

\[
\Gamma \vdash (C) \ e : C
\]

check for "type" error
Java-Like Language: Casts

• Upcast: always safe, doesn’t do anything
• Downcast: safe only if object actually has the right type
• Other casts: ??

B b = new B();
A a = (A) ((Object) b);

\[
\begin{align*}
\text{(e, } \sigma \text{)} \downarrow \text{new } C(v_1, \ldots, v_n) & \quad C <: D \\
((D) \ e , \sigma) \downarrow \text{new } C(v_1, \ldots, v_n) & \\
\text{(e, } \sigma \text{)} \downarrow \text{new } C(v_1, \ldots, v_n) & \quad \text{not } C <: D \\
((D) \ e , \sigma) \downarrow \text{exc}
\end{align*}
\]
Java-Like Language: Mutable Objects

• Our language so far has no field-set operation!

A a1 = new A(3, 5);
A a2 = a1;
a1.x = 4;
int result = a2.x; // should be 4
Java-Like Language: Mutable Objects

• Our language so far has no field-set operation!

```java
A a1 = new A(3, 5);  // {a1 = new A(3, 5)}
A a2 = a1;           // {a1 = new A(3, 5), a2 = new A(3,5)}
a1.x = 4;            // {a1 = new A(4, 5), a2 = new A(3,5)}
int result = a2.x;   // should be 4
```
Java-Like Language: Mutable Objects

• Our language so far has no field-set operation!

A a1 = new A(3, 5);       {a1 = r1, r1 -> new A(3, 5)}
A a2 = a1;               {a1 = r1, a2 = r1, r1 -> new A(3, 5)}
a1.x = 4;               {a1 = r1, a2 = r1, r1 -> new A(4, 5)}
int result = a2.x; // should be 4
a1 = new A(6, 7);       {a1 = r2, a2 = r1, r1 -> new A(4, 5),
                           r2 -> new A(6, 7)}

• Two-level model: variables hold references, references point to values
Java-Like Language: Mutable Objects

• Split the state $\sigma$ into two levels

• Program state is now a tuple $(c, k, \rho, \sigma)$ where:
  — $c$ is the currently executing command
  — $k$ is the call stack
  — $\rho$ is the environment, mapping variables to either primitive values (int, bool) or references
  — $\sigma$ is the store, mapping references to object values
Java-Like Language: Syntax

\[ CL ::= \text{class } <\text{id}> \text{ extends } <\text{id}> \{ T <\text{id}> , \ldots , T <\text{id}> ; K M \ldots M \} \]

\[ K ::= <\text{id}>(T <\text{id}> , \ldots , T <\text{id}> ) \{ \]

\[
\quad \text{super}(<\text{id}> , \ldots , <\text{id}> ); \text{this.<id>} = <\text{id}> ; \ldots ; \text{this.<id>} = <\text{id}> ; \} 
\]

\[ M ::= T <\text{id}>(T <\text{id}> , \ldots , T <\text{id}> ) \{ C \} \]

\[ E ::= \ldots \mid E.<\text{id}> \]

\[ C ::= \ldots \mid <\text{id}> = \text{new } <\text{id}>(E , \ldots , E) ; \mid E.<\text{id}> = E ; \mid <\text{id}> = E.<\text{id}>(E , \ldots , E) ; \]

\[ T ::= \text{int} \mid <\text{id}> \]
Java-Like Language: Semantics

\[(e, \sigma) \Downarrow \text{new } C(v_1, \ldots, v_n) \quad \text{fields}(C) = \tau_1 f_1, \ldots, \tau_n f_n \]

\[(e.f_i, \sigma) \Downarrow v_i\]
Java-Like Language: Semantics

\[(e, \rho, \sigma) \downarrow r \quad \sigma(r) = \text{new } C(\nu_1, \ldots, \nu_n) \quad \text{fields}(C) = \tau_1 f_1, \ldots, \tau_n f_n\]

\[(e. f_i, \rho, \sigma) \downarrow v_i\]

\[\rho(x) = v\]

\[(x, \rho, \sigma) \downarrow v\]

\[(e, \rho, \sigma) \downarrow v\]

\[(x = e; , \rho, \sigma) \rightarrow (\text{skip}, \rho[x \mapsto v], \sigma)\]
Java-Like Language: Semantics

\[
(e_1, \sigma) \Downarrow v_1 \quad \ldots \quad (e_n, \sigma) \Downarrow v_n
\]

\[
(new\ C(e_1, \ldots, e_n), \sigma) \Downarrow new\ C(v_1, \ldots, v_n)
\]
Java-Like Language: Semantics

\[
\begin{align*}
(e_1, \rho, \sigma) \downarrow v_1 & \quad \cdots \quad (e_n, \rho, \sigma) \downarrow v_n \quad r \notin \text{dom}(\sigma) \\
(x = \text{new } C(e_1, \ldots, e_n, \rho, \sigma) & \rightarrow \\
\text{(skip, } \rho[x \mapsto r], \sigma[r \mapsto \text{new } C(v_1, \ldots, v_n)]\text{)})
\end{align*}
\]

\[
\begin{align*}
(e, \rho, \sigma) \downarrow r & \quad \sigma(r) = \text{new } C(v_1, \ldots, v_n) \quad \text{fields}(C) = f_1, \ldots, f_n \\
(e_1, \rho, \sigma) \downarrow v & \quad (e_1, \rho, \sigma) \downarrow v \\
(e. f_i = e_1;, \rho, \sigma) & \rightarrow \text{(skip, } \rho, \sigma[r \mapsto \text{new } C(v_1, \ldots, v, \ldots, v_n)]\text{)}
\end{align*}
\]
Java-Like Language: Mutable Objects

A a1 = new A(3, 5);
A a2 = a1;
a1.x = 4;
int result = a2.x; // should be 4
a1 = new A(6, 7);

{a1 = r1, r1 -> new A(3, 5)}
{a1 = r1, a2 = r1, r1 -> new A(3, 5)}
{a1 = r1, a2 = r1, r1 -> new A(4, 5)}
{a1 = r2, a2 = r1,
  r1 -> new A(4, 5), r2 -> new A(6, 7)}