CSC547: Type Systems for OO Languages

Untyped calculi
Abadi and Cardelli, Chapter 6

The ζ-calculus

Semantics of the ζ-calculus

Functions as objects

Classes

Objects as functions?
Object primitives

Primitives are:

- Objects
- Methods
- Method call
- Method update

Abadi and Cardelli define a $\varsigma$-calculus with:

- Grammars for objects and methods
- Reductions $a \rightarrow b$ (a ‘takes one step to become’ b)
- Equations $a \leftrightarrow b$ (a ‘is the same program as’ b)
- Interpreter $a \hookrightarrow v$ (a ‘produces result’ v)

(Why all three?)

For the moment, this is all without types... they are in the next chapter.
Primitives of the ς-calculus

Variables \( x \)

Methods \( ς(x)b \)

Objects \([l_1=M_1, ..., l_n=M_n]\)

Method invocation \( a.l \)

Method update \( a.l \leftarrow M \)

Note that all methods are nullary (take no arguments)!
Primitives of the $\varsigma$-calculus

The object:

\[
[ \\
  \text{method}_1 = \varsigma(\text{this}) \ \text{body}_1, \\
  \ldots \\
  \text{method}_n = \varsigma(\text{this}) \ \text{body}_n \\
]
\]

is similar to the pseudo-Java:

\[
\text{new ObjectType} () \{ \\
  \text{ReturnType}_1 \ \text{method}_1 () \{ \text{return body}_1; \} \\
  \ldots \\
  \text{ReturnType}_n \ \text{method}_n () \{ \text{return body}_n; \} \\
\}
\]

(but note Java is typed!)

Method invocation \texttt{a.l} is similar to the pseudo-Java \texttt{a.l()}.

Method update \texttt{a.l}⇐\texttt{M} doesn’t have a Java equivalent! It’s similar to the object \texttt{b extends a { ... }} pseudo-Java we saw in Chapter 4.
Examples

interface Point {
    int getX ();
    int getY ();
    Point move ();
}

Point origin = new Point () {
    int getX () { return 0; }
    int getY () { return 0; }
    Point move () {
        Point current = this;
        Point result extends current {
            int getX () { return current.getX () + 1; }
            int getY () { return current.getY () + 1; }
        }
        return result;
    }
};

How can this be written in the $\xi$-calculus?
Examples

interface Integer {
    boolean isZero ();
    Integer succ ();
    Integer pred ();
}

Integer zero = new Integer () {
    boolean isZero () {
        return true;
    }

    Integer pred () {
        return this;
    }

    Integer succ () {
        Integer current = this;
        Integer result extends current {
            boolean isZero () {
                return false;
            }

            Integer pred () {
                return current;
            }
        }
    }

    return result;
}

one = zero.succ ();
two = one.succ ();

How can this be written in the $\varsigma$-calculus?
Examples

We can treat fields as methods with no use of this.

\[
\text{person} \triangleq [ \\
\quad \text{name} = "Unknown", \\
\quad \text{age} = 0, \\
\quad \text{birthday} = \varsigma(\text{this}) ( \\
\quad \quad \text{this.age} := \text{this.age.succ}
\quad )
\]
\[
\text{fred} \triangleq \text{person}
\quad .\text{name} := "Fred Flintstone"
\quad .\text{age} := 37
\]

How can we translate this into the ‘pure’ \( \varsigma \)-calculus?
Syntax of the $\varsigma$-calculus

Methods:

\[ M ::= \varsigma(x) \, b \]

Objects:

\[ O ::= [ \, l_1 = M_1, \ldots, l_n = M_n \, ] \]

Method bodies:

\[ b ::= x \]
\[ \mid O \]
\[ \mid b.l \]
\[ \mid b.l \leftarrow M \]
Primitive semantics of the $\varsigma$-calculus

A method $M_i = \varsigma(x_i)b_i$ ‘doesn’t do anything by itself’.

An object $O = [l_1=M_1, ..., l_n=M_n]$ ‘doesn’t do anything by itself’.

A method call $O.l_i$ calls $b_i$ with $x_i$ bound to $O$.

A method update $O.l_i \leftarrow M$ returns a new object ‘like $O$ but with $M_i$ replaced by $M$’.

```
person ≜ [ 
    name = "Unknown",
    age = 0,
    birthday = $\varsigma$(this) ( 
        this.age := this.age.succ
    )
] 
fred ≜ person
    .name := "Fred Flintstone"
    .age := 37
```

What is fred.birthday.age?
Primitive semantics of the $\omega$-calculus

That was in English, now in maths...

Define a relation $a \rightarrow b$ meaning ‘program $a$ can become program $b$’ as:

\[
O.l_i \rightarrow b_i \{ \{ x_i \leftarrow O \} \}
\]

\[
O.l_i \triangleleft M
\rightarrow [l_1= M_1, \ldots, l_{i-1}= M_{i-1}, l_i= M, l_{i+1}= M_{i+1}, \ldots, l_n= M_n]
\]

where $O$ is $[l_1= M_1, \ldots, l_n= M_n]$ and $M_i$ is $\omega(x_i) b_i$

Now what about fred.birthday.age?

$b \{\{ x \leftarrow O \}\}$ is $b$ with $O$ substituted for $x$.

How can we define substitution formally?
Reduction semantics of the $\varsigma$-calculus

We are only allowed to use $\rightarrow$ ‘at top-level’, for example:

\[
\text{[contents=zero].contents:=one} \rightarrow [\text{contents=one}]
\]

but:

\[
(\text{[contents=zero].contents:=one}).\text{contents} 
\not\rightarrow [\text{contents=one}]\text{.contents}
\]

A context $C[-]$ is a term with a hole (written ‘-’) in it, for example $C[-]$ might be $-.m$

We can fill a context with a term $C[b]$ for example if $C[-]$ is $-.m$ then $C[O.l]$ is $O.l.m$

Define $a \rightarrow b$ whenever:

\[
a = C[a'] \quad b = C[b'] \quad a' \rightarrow b'
\]

For example:

\[
\text{[contents=zero].contents:=one}.\text{contents} \rightarrow [\text{contents=one}]\text{.contents} 
\rightarrow \text{one}
\]
Reduction semantics of the $\varsigma$-calculus

Let $a \rightarrow b$ whenever $a \rightarrow \cdots \rightarrow b$

For example:

\[
[\text{contents=zero}].\text{contents}:=\text{one.contents} \\
\rightarrow \text{one}
\]

but oh dear...

\[
[\ x = [\text{contents=zero}].\text{contents}, \ y = \ \\
[\text{contents=one}].\text{contents] } \\
\rightarrow [\ x = \text{zero}, \ y = [\text{contents=one}].\text{contents} ]
\]

and...

\[
[\ x = [\text{contents=zero}].\text{contents}, \ y = \ \\
[\text{contents=one}].\text{contents] } \\
\rightarrow [\ x = [\text{contents=zero}].\text{contents}, \ y = \text{one} ]
\]

Gulp, nondeterminism!

Is the $\varsigma$-calculus really nondeterministic?

**Theorem (Church-Rosser)**

If $a \rightarrow b$ and $a \rightarrow c$ then there exists $d$ such that $b \rightarrow d$ and $c \rightarrow d$
Equations for the $\varsigma$-calculus

In section 6.2.3, Abadi and Cardelli give a collection of equations for the $\varsigma$-calculus.

The equivalence is given as the smallest relation $\leftrightarrow$ such that:

- If $a \rightarrow b$ then $a \leftrightarrow b$
- $a \leftrightarrow a$
- If $a \leftrightarrow b$ then $b \leftrightarrow a$
- If $a \leftrightarrow b$ and $b \leftrightarrow c$ then $a \leftrightarrow c$
- If $a \leftrightarrow b$ then $C[a] \leftrightarrow C[b]$

Example: show one.succ.pred $\leftrightarrow$ one.pred.succ

Proposition

$b \leftrightarrow c$ iff there exists $d$ such that $b \rightarrow d$ and $c \rightarrow d$
Operational semantics for the \(\varsigma\)-calculus

In section 6.2.4 Abadi and Cardelli define an operational semantics \(a \rightsquigarrow v\) where \(v\) is a value (in this case just an object \(O\)).

The operational semantics is given as the smallest relation \(\rightsquigarrow\) such that:

- \(v \rightsquigarrow v\)
- If \(a \rightsquigarrow O\), \(O.l \rightarrow b\) and \(b \rightsquigarrow v\) then \(a.l \rightsquigarrow v\)
- If \(a \rightsquigarrow O\) and \(O.l \leftarrow M \rightarrow v\) then \(a.l \leftarrow M \rightsquigarrow v\)

Example: show \(\text{one.pred} \rightsquigarrow \text{zero}\)

**Proposition (\(\rightsquigarrow\) is deterministic)**
If \(a \rightsquigarrow v\) and \(a \rightsquigarrow w\) then \(v \equiv w\).

**Proposition (Soundness of \(\rightsquigarrow\))**
If \(a \rightsquigarrow v\) then \(a \rightarrow v\).

**Proposition (Completeness of \(\rightsquigarrow\))**
If \(a \rightarrow v\) then \(a \rightsquigarrow w\) where \(w \rightarrow v\).

From this operational semantics, it is easy to read off an interpreter.
Functions as objects

So far, we’ve only looked at methods without arguments.

Not very practical!

Write $\lambda(x)b$ for a function with argument $x$ and body $b$.

For example:

```plaintext
cell ≜ [ contents = zero,
set = λ(this) λ(n) ( this.contents := n ),
get = λ(this) ( this.contents ) ]
```

What is cell.set (one).get?

How do we code a point object?

How do we code booleans?

How do we code linked lists?
Functions as objects

Functions can be implemented as objects!

\[
\lambda(x)b \triangleq [
\begin{align*}
\text{arg} &= \varsigma(x) ( \\
&\quad x.\text{arg} \\
\text{val} &= \varsigma(x) ( \\
&\quad b \{\{ x \leftarrow x.\text{arg} \}\} \\
&\quad )
\end{align*}
\]

Function application is update followed by method call:

\[
b (a) \triangleq (b.\text{arg}:=a).\text{val}
\]

Let’s just check this works:

\[
(\lambda(x)b) (a) \rightarrow b \{\{ x \leftarrow a \}\}
\]
Fixpoints

Recursive functions are quite useful!

$$\text{fact } \triangleq \mu(f) \lambda(x) \ ( \begin{array}{l}
\text{if } (x == 0) \ (1) \\
\text{else } (x \ast (f \ (x - 1)))
\end{array} )$$

What is fact (3)?
Functions as objects

Recursion can also be implemented as objects:

\[
\mu(x)b \triangleq [ \\
\text{rec} = \varsigma(x) ( \\
\quad b \{\{ x \leftarrow x.\text{rec} \}\} \\
\quad ) \\
\].\text{rec}
\]

Let’s just check this works:

\[
\mu(x)b \rightarrow b \{\{ x \leftarrow \mu(x)b \}\}
\]
Classes and traits

Given a class definition in pseudo-Java (ignoring types for now):

class a {
    foo () { return 1; }
    bar () { return this.foo () + 1; }
}

we can represent this in the $\varsigma$-calculus:

\[
a \triangleq \left[ \begin{array}{l}
    \text{new} = \varsigma(\text{This}) \left[ \begin{array}{l}
        \text{foo} = \varsigma(\text{this}) \text{This.foo (this)}, \\
        \text{bar} = \varsigma(\text{this}) \text{This.bar (this)}
    \end{array} \right], \\
    \text{foo} = \lambda(\text{this}) ( 1 ), \\
    \text{bar} = \lambda(\text{this}) ( \text{this.foo + 1} )
\end{array} \right]
\]

and new a () becomes a.new

What does a.new.bar do?
Classes and traits

Given a class definition in pseudo-Java (ignoring types for now):

```java
class b extends a {
    foo () { return 2; }
    baz () { return this.foo () + super.foo (); }
}
```

we can represent this in the $\varsigma$-calculus:

```
b ≜ [
    new = $\varsigma$(This) [
        foo = $\varsigma$(this) This.foo (this),
        bar = $\varsigma$(this) This.bar (this),
        baz = $\varsigma$(this) This.baz (this)
    ],
    foo = $\lambda$(this) ( 2 ),
    bar = a.bar,
    baz = $\lambda$(this) ( this.foo + a.foo(this) )
]
```

What does b.new.baz do?
Objects as functions

We can try to code objects as records of functions:

\[
[ l_1 = \varsigma(x_1) b_1, \ldots, l_n = \varsigma(x_n) b_n ]
\]

\[
\triangleq \langle l_1 = \lambda(x_1) b_1, \ldots, l_n = \lambda(x_n) b_n \rangle
\]

\[
a.l \triangleq a.l(a)
\]

\[
a.l \leftarrow \varsigma(x)b
\]

\[
\triangleq a.l := \lambda(x)b
\]

It’s easy to check that this has the same semantics as the \(\varsigma\)-calculus.

So why bother with the \(\varsigma\)-calculus?