Object-Oriented Programs

Shangping Ren

Department of Computer Science
Illinois Institute of Technology

March 10, 2014
Outline

1. Basics
2. Syntax
3. Semantics
4. Verification
   - Partial Correctness
   - Total Correctness
OO Basics

- State
  - instance variable

- Encapsulation
  - instance variable is “invisible” to outside
  - access to instance variable through method call

- method call
  - control transfers from caller to callee
Local Expressions

- New type: **Object**;
- Operations: \( \text{=} \text{object} \), or abbreviated by \( \text{=} \);
- constant **null** of type **Object**: void reference
- variable **this** of type **Object**: current executing object;
- Shared variable set \( \text{Var} \);
- Instance variable set \( \text{IVar} \), variable owned by objects;
- Instance variable can be either simple variable or array variable.
- \( \text{Var} \cap \text{IVar} = \emptyset \);
- **this** \( \in \text{Var} \)
Local Expression Examples

Examples: Given an array variable \( a \in Var \cup IVar \) of type \( Integer \rightarrow Object \), and \( u \) and \( v \) are of type \( Object \)

- \( a[0] \) is a local expression of type \( Object \)
- \( a[1] = null \) is a local Boolean expression;
- \( this = a[0] \) and \( this \neq null \) are local Boolean expressions
- expression of type \( Object \) can only be compared for equality
- \( u := null \) is an assignment
- \( u := this \) is an assignment
- \( v := u \) is an assignment
Method call \((s.m)\) : \(s\) of \textbf{Object} and \(m\) a method of \(s\)

\[
S ::= \text{skip} \mid u ::= t \mid S_1; S_2 \mid \text{if } B \text{ then } S_1 \text{ else } S_2 \text{ fi} \mid \text{while } B \text{ do } S_1 \text{ od} \mid s.m
\]

Method definition \((m :: S)\) : \(m\) is the method name and \(S\) is method body which is defined by using the syntax given above.
Method Definition ($m :: S$)

$m :: S$

- instance variables appearing in the method definition body $S$ is owned by `this`.
- instance variables are permanent
- names of instance variables can coincide
- `this` is read-only

Ensuring permanency of instance variables

For every

$$
\begin{align*}
\text{begin local } \overline{u} := \overline{t}; S \text{ end}
\end{align*}
$$

we require $\text{var}(\overline{u}) \subseteq \text{Var} \setminus \{\text{this}\}$. 
**Syntax**

**OO-Program**

\[ D \] : a set of method definition

\[ S \] : main statement

Valid OO-program

- each method used has a unique declaration in \( D \);
- each method call refers to a method declared in \( D \);
- no local variable of \( S \) occurs in \( D \), i.e, \( IVar(S) \cap D = \phi \).
Example 1:

\[ \text{getx :: return} := x; \]

Example 2:

\[ y.\text{getx}; z := \text{return} \]

Example 3:

\[ y.\text{getx}; x := \text{return} \]
More OO-Program Examples

Method definitions:

\[ \text{inc}::\text{count} := \text{count} + 1, \]
\[ \text{reset}::\text{count} := 0 \]

Main statement

\[
\begin{align*}
i &:= 0; \\
\text{up}.\text{reset}; \\
\text{down}.\text{reset}; \\
\text{while} \quad i \leq n \\
\text{do} \quad &\text{if} \quad a[i] > j \\
&\quad \quad \text{then} \quad \text{up}.\text{inc} \\
&\quad \quad \text{else} \quad \text{if} \quad a[i] < k \quad \text{then} \quad \text{down}.\text{inc} \quad \text{fi} \\
&\quad \quad \text{fi}; \\
&\quad i := i + 1 \\
\text{od}
\end{align*}
\]
Add the value of instance variable `val` in a link list.
More OO-Program Examples

Method definitions:

```plaintext
add::sum := sum + val,
move::current := next.
```

Main statement

```plaintext
sum := 0;
current := first;
while current ≠ null
do
    current.add; current.move
od
```

What if `current.move` is changed to `current := next`?
Semantics

1. Extension of State Definition
2. Extension of State Update
3. Extension of Transition Axioms
Semantics of Local Expressions

Let $\mathcal{D}_{\text{object}}$ be the set of object identities and $\text{null} \in \mathcal{D}_{\text{object}}$, a proper state $\sigma$:

- All object variables are in the object identity set, i.e.,
  $\forall o \in \text{Object}, \exists id \in \mathcal{D}_{\text{object}},$ such that, $o = id$;

- Each object $o \in \text{Object}$ is assigned to its local state, i.e., $\sigma(o)$

- The local state $\sigma(o)$ of an object $o$ further assigns values of appropriate types to the object $o$’s instance variables $x \in IVar$, i.e.,
  - $x \in \text{Var}$, the value of $x$ is given by $\sigma(x)$
  - $x \in IVar$, the value of $x$ is given by $\sigma(o)(x)$

- An instance variable can also be of type $\text{Object}$ (object nesting) (such as $o3$ within $o1$ in the next slide).
Semantics of Local Expressions — Cont.

Let \( o = \sigma(\text{this}) \), we have

1. if \( s \equiv \text{null} \) then
   \[
   S[s](\sigma) = \text{null}
   \]

2. if \( s \equiv x \) for some simple instance variable \( x \) then
   \[
   S[s](\sigma) = \sigma(o)(x)
   \]

3. if \( s \equiv a[s_1, \ldots, s_n] \) for some instance array variable \( a \) then
   \[
   S[s](\sigma) = \sigma(o)(a)(S[s_1](\sigma), \ldots, S[s_n](\sigma))
   \]

Abbreviate \( S[s](\sigma) \) with \( \sigma(s) \). We have

\[
\forall x \in \text{IVar}, \quad \sigma(x) = \sigma(\sigma(\text{this}))(x).
\]
Semantics of Local Expressions — Examples

1. $\sigma(x) =$?
2. $\sigma(o1)(x) =$?
3. $\sigma(o2)(a[1]) =$?

Figure: A Proper State
Global Expressions

- if $s$ is a global expression type `Object` and $x$ is an instance variable of a basic type $T$, then $s.x$ is a global expression of type $T$

$$S[s.x](\sigma) = \sigma(o)(x)$$

- if $s$ is a global expression type `Object`, $s_1, \ldots, s_n$ are global expressions of type $T_1, \ldots, T_n$, and $a$ is an array instance variable of type $T_1 \times \ldots \times T \rightarrow T$, then $s.a[s_1, \ldots, s_n]$ is a global expression of type $T$

$$S[s.a[s_1, \ldots, s_n]](\sigma) = \sigma(o)(a)(S[s_1](\sigma), \ldots, S[s_1](\sigma))$$

where $S[s](\sigma) = o$.

The global expression of the form $s.u$ a navigation expression.
Assertions are constructed from *global* Boolean expressions. Only *normal* variables can be quantified. (Why?)
Let's see how we update the instance variable $x$ in previous example:

**Step 1:** from the proper state $\sigma$ (the gray circle), find the instance variable’s encapsulating object (the yellow circle) $o1 = \sigma(this)$ and its local state $\sigma(\sigma(this))$ or $\sigma(o1)$

**Step 2:** from the local state $\tau = \sigma(o1)$, we update the value of variable $x$ with new value, say 8, i.e., $\tau[x := 8]$.
Updates of States

Given the state as shown in the figure above,

1. \( o_1.x[x := 4] \equiv \) ?
2. \( o_2.a[2][a[2] := 5] \equiv \) ?
3. let \( \text{this} = o_3, \text{this} .z[z := t] \equiv \) ?
State Updates

In general, let $u$ be an instance variable of type $T$, and $d \in D_T$, $o \in D_{object}$, $\sigma$ be a proper state, $\tau$ be a local state, and let $o = \sigma(this)$ and $\tau = \sigma(o)$

$$\sigma[u := d] = \sigma[o := \tau[u := d]]$$

where the state update $\sigma[o := \tau]$ is defined by

$$\sigma[o := \tau](o') = \begin{cases} 
\tau & \text{if } o = o' \\
\sigma(o') & \text{otherwise}
\end{cases}$$
State Updates — Examples

Let \( x \in IVar, \ o = \sigma(this) \), and \( \tau = \sigma(o) \)

\[
\sigma[x := 1](x) = \sigma[x := 1](\sigma[x := 1](this))(x) = \sigma[x := 1](\sigma[x := 1](this))(x)
\]

\[
\{\text{state def} : \sigma(x) = \sigma(\sigma(this))(x)\}
\]

\[
= \sigma[x := 1](\sigma[x := 1](this))(x) = \sigma(\sigma(this)) = o
\]

\[
= \sigma[x := 1](o)(x) = \sigma[x := 1](o)(x)
\]

\[
\{\text{state update def} : \sigma[x := 1]\}
\]

\[
= \sigma[o := \tau[x := 1]](o)(x) = \sigma[o := \tau[x := 1]](o)(x)
\]

\[
\{\text{state update def} \ \sigma[o := \tau[x := 1]]\}
\]

\[
= \tau[x := 1](x) = \tau[x := 1](x)
\]

\[
\{\text{state update def} \ \tau[x := 1]\}
\]

\[
= 1
\]
Let $a \in IVar$, $o = \sigma(this)$, and $\tau = \sigma(o)$. The proper state $\sigma$ is as shown in Figure below.

$$
\sigma[a[1] := a[0] + 1](a[1]) = ?
$$
Semantics of Statements and Programs

Assignment Axiom

\[ \langle u := t, \sigma \rangle \rightarrow \langle E, \sigma[u := \sigma(t)] \rangle \]

Method Call Axiom

\[ \langle s.m, \sigma \rangle \rightarrow \]

\[
\begin{cases}
\langle E, \text{fail} \rangle & \text{if } \sigma(s) = \text{null} \\
\langle \text{begin local this} := s; S \text{ end }, \sigma \rangle & \text{if } \sigma(s) \neq \text{null} \text{ and } m :: S \in D,
\end{cases}
\]

Question: can we use the following to define the method call axiom?

\[ \langle s.m, \sigma \rangle \rightarrow \langle \text{this} := s; S; \text{this} := \sigma(\text{this}), \sigma \rangle \]
Semantics of Statements and Programs — Examples

\(< \ y.\text{getx}, \sigma \ >
\rightarrow \ < \ \text{begin local this} := y; \ \text{return} := x \ \text{end} , \sigma >
\rightarrow \ < \ \text{this} := y; \ \text{return} := x; \ \text{this} := \sigma(\text{this}) , \sigma >
\rightarrow \ < \ \text{return} := x; \ \text{this} := \sigma(\text{this}) , \sigma[\text{this} := y] >
\rightarrow \ < \ \text{this} := \sigma(\text{this}) , \sigma[\text{this} := y][\text{return} := \sigma(\sigma(y))(x)] >
\rightarrow \ < \ E, \sigma[\text{this} := y][\text{return} := \sigma(\sigma(y))(x)][\text{this} := \sigma(\text{this})] >
= \ < \ E, \sigma[\text{return} := \sigma(\sigma(y))(x)][\text{this} := y][\text{this} := \sigma(\text{this})] >
= \ < \ E, \sigma[\text{return} := \sigma(\sigma(y))(x)][\text{this} := \sigma(\text{this})] >
= \ < \ E, \sigma[\text{return} := \sigma(\sigma(y))(x)] >
Absence of Blocking

Lemma

For every $S$ that can arise during an execution of an OO-program, if $S \not\equiv E$ then for any proper state $\sigma$, such that $\sigma(\text{this}) \neq \text{null}$, there exists a configuration $< S_1, \tau >$ such that

$$< S, \sigma > \rightarrow < S_1, \tau >$$

where $\tau(\text{this}) \neq \text{null}$.
Partial/Total Correctness of OO-Programs

Partial Correctness  same as the while program
Total Correctness

\[ \mathcal{M}_{tot}[S](\sigma) = \mathcal{M}[S](\sigma) \]
\[ \cup \{ \bot \mid S \text{ can diverge from } \sigma \} \]
\[ \cup \{ \text{fail} \mid S \text{ can fail from } \sigma. \} \]
Partial/Total Correctness — cont

- if $\sigma(y) \neq \text{null}$ then
  
  $$\mathcal{M}[y\text{.getx}](\sigma) = \mathcal{M}_{\text{tot}}[y\text{.getx}](\sigma) = \{ \sigma[\text{return} := \sigma(\sigma(y))(x)] \}$$

- if $\sigma(y) = \text{null}$ then
  
  $$\mathcal{M}[y\text{.getx}](\sigma) = \phi$$

  and

  $$\mathcal{M}_{\text{tot}}[y\text{.getx}](\sigma) = \{ \text{fail} \}.$$
Substitution

Let $u$ be a (simple, or array) instance variable; $s$ and $t$ be global expressions.

1. if $s \equiv x \in \text{Var}$, then

$$s[u := t] \equiv x$$

2. if $s \equiv e.u$ and $u$ is a simple instance variable, then

$$s[u := t] \equiv \text{if } e' = \text{this then } t \text{ else } e'.u \text{ fi}$$

$$e' \equiv e[u := t]$$

3. if $s \equiv e.a[s_1, \ldots, s_n]$ and $u \equiv a[t_1, \ldots, t_n]$, then

$$s[u := t] \equiv \text{if } e' = \text{this } \land \bigwedge_{i=1}^{n} s'_i = t_i \text{ then } t \text{ else } e'.a[s'_1, \ldots, s'_n] \text{ fi}$$

$$e' \equiv e[u := t] \text{ and } \forall i \in \{1, \ldots, n\}, \, s'_i \equiv s_i[u := t]$$
Substitution — Examples

Let $s \equiv \text{this} \ . u$, then

$$\text{this} \ . u[u := t]$$

$$\equiv \text{if this} [u := t] = \text{this} \text{ then } t \text{ else } \text{this} [u := t] . u \text{ fi}$$

$$\equiv \text{if this} = \text{this} \text{ then } t \text{ else } \text{this} [u := t] . u \text{ fi}$$

$$\equiv t$$

Hence, for all proper states $s$, we have $\sigma(\text{this} \ . u[u := t]) = \sigma(t)$. 
Substitution — Examples

Let $s \equiv \text{this}.a[x]$, then

$$\text{this}.a[x][a[x] := t]$$

$\equiv$ if this $[a[x] := t] = \text{this} \land x[a[x] := t] = x$ then $t$ else this $[a[x] := t]$

$\equiv$ if this $= \text{this} \land x = x$ then $t$ else this $[a[x] := t].a[x]$ fi

$\equiv t$

So, for all proper states $s$, we have $\sigma(\text{this}.a[x][a[x] := t]) = \sigma(t)$. 
Substitution — Examples

Let $s \equiv x.u$, then

$$x.u[u := t]$$

$$\equiv \text{if } x[u := t] = \text{this} \text{ then } t \text{ else } x[u := t].u \text{ fi}$$

$$\equiv \text{if } x = \text{this} \text{ then } t \text{ else } x[u := t].u \text{ fi}$$
Substitution of Instance Variables

Recall the substitution of variable lemma: For all expressions $s$ and $t$, all assertions $p$, all simple or subscripted variables $u$ of the same type as $t$ and all proper states $\sigma$,

1. $\sigma(s[u := t]) = \sigma[u := \sigma(t)](s)$
2. $\sigma \models p[u := t] = \sigma[u := \sigma(t)] \models p$

We have similar lemma for instance variables

Lemma

For all global expressions $s$ and $t$, all assertions $p$, all simple or subscripted instance variables $u$ of the same type as $t$ and all proper states $\sigma$,

1. $\sigma(s[u := t]) = \sigma[u := \sigma(t)](s)$, and
2. $\sigma \models p[u := t] \iff \sigma[u := \sigma(t)] \models p$
Correctness

\[ [p] = \{ \sigma | \sigma \text{ is a proper state s.t. } \sigma(\textit{this}) \neq \textit{null} \text{ and } \sigma \models p \} \]

- \models \{ p \} S \{ q \} \text{ if } M[S]([p]) \subseteq [q].
- \models_{tot} \{ p \} S \{ q \} \text{ if } M_{tot}[S]([p]) \subseteq [q].
Rule 10: Block

\[
\begin{align*}
\{p\} \overline{x} := \overline{t}; S \{q\} \\
\{p\} \text{begin local } \overline{x} := \overline{t}; S \text{ end } \{q\}
\end{align*}
\]

where \( \text{var}(\overline{x}) \cap \text{free}(q) = \emptyset \).
Assignment to Instance Variables Axiom

Axiom 14: Assignment to Instance Variables

\[ \{ p[u := t] \} u := t[p] \]

where \( u \) is a (simple or subscripted) instance variable.
Method Call

Rule 15: Method Instantiation

\[
\begin{align*}
\{p\} y.m\{q\} & \quad \Rightarrow \quad \{p[y := s]\} s.m\{q[y := s]\}
\end{align*}
\]

where \( D \) is the set of method declarations, \( y \notin \text{var}(D) \) and \( \text{var}(s) \cap \text{change}(D) = \emptyset \).

Rule 16: Method Call

\[
\begin{align*}
\{p\}\begin{align*}
\text{begin local this} & := s; S \end{align*} \quad \text{end} \quad \{q\} & \quad \Rightarrow \quad \{p\} s.m\{q\}
\end{align*}
\]

where \( D = m :: S \).
Partial Correctness — Examples

Example:
Let $\text{count} \in \text{IVar}$, $z \in \text{Var}$, and $\text{other} \in \text{Object}$. The method $\text{inc}$ definition is given below:

$$\text{inc} :: \text{count} := \text{count} + 1$$

Prove the following invariant property:

$$\{ \text{this} \neq \text{other} \land \text{this} . \text{count} = z \} \text{other} . \text{inc} \{ \text{this} . \text{count} = z \}$$
Total Correctness

Rule 17: Recursion VI

\[
\begin{align*}
\{p_1\} s_1.m_1\{q_1\}, \ldots, \{p_n\} s_n.m_n\{q_n\} & \vdash \{p\} s\{q\}, \\
\{p_1 \land t < z\} s_1.m_1\{q_1\}, \ldots, \{p_n \land t < z\} s_n.m_n\{q_n\} & \vdash \\
\{p_i \land t = z\} & \begin{aligned}
& \text{begin local this := } s_i; S_i \text{ end} \\
& \{q_i\}, i \in \{1, \ldots, n\}, \\
& p_i \rightarrow t \geq 0 \land s_i \neq \text{null}, i \in \{1, \ldots, n\}
\end{aligned} \\
\{p\} S\{q\}
\end{align*}
\]

where \(m_i :: S_i \in D\), and \(z\) is an integer variable that does not occur in \(p_i, t, q_i, S_i\) for \(i \in \{1, \ldots, n\}\).
The rest of the slides are for your interest. They will not be covered in the class.
Method Call with Parameters

\[ S ::= s.m(t_1, \ldots, t_n) \]

and method is defined as

\[ m(u_1, \ldots, u_n) :: S \]
Method Call with Parameters — Examples

Method definition:

\[ \text{setx}(u) :: x := u. \]

Main statement

\[ y.\text{setx}(t) \]

Method definition:

\[ \text{setnext}(u) :: \text{next} := u. \]

Main statement

\[ x.\text{setnext}(\text{next}); \text{next} := x \]

(See Figure 6.2 and 6.3)
Method Call with Parameters — Semantics

\[< s.m(t), \sigma > \rightarrow < \text{begin local this , } \bar{u} := s, t; S \; \text{end} , \sigma > \]

where \( \sigma(s) \neq \text{null} \) and \( m(\bar{u}) :: S \in D \),

\[< s.m(t), \sigma > \rightarrow < E, \; \text{fail} > \quad \text{where } \sigma(s) = \text{null} \]
\[ \langle x.\text{setnext}(\text{next}), \sigma \rangle \]
\[ \rightarrow \langle \text{begin local this , } u := x, \text{next ; next } := u \text{ end} , \sigma \rangle \]
\[ \rightarrow \langle \text{this , } u := x, \text{next ; next } := u; \text{this , } u := \sigma(\text{this } ), \sigma(u), \sigma \rangle \]
\[ \rightarrow \langle \text{next } := u; \text{this , } u := \sigma(\text{this } ), \sigma(u), \sigma' \rangle \]
\[ \rightarrow \langle \text{this , } u := \sigma(\text{this } ), \sigma(u), \sigma'[\text{next } := \sigma(o)(\text{next})] \rangle \]
\[ \rightarrow \langle E, \sigma[\sigma' := \tau[\text{next } := \sigma(o)(\text{next})]] \rangle , \]

where \( \sigma' \) denotes the state

\[ \sigma[\text{this , } u := \sigma', \sigma(o)(\text{next})] \]
Method Call with Parameters — Partial Correctness

Rule 18: Instantiation III

\[ \{p\} y.m(\bar{x})\{q\} \]
\[ \{p[y, \bar{x} := s, \bar{t}]\} s.m(\bar{t})\{q[y, \bar{x} := s, \bar{t}]\} \]

where \(y, \bar{t}\) is a sequence of simple variables in \(Var\) which do not appear in \(D\) and \(\text{var}(s, \bar{t}) \cap \text{change}(D) = \phi\).

Rule 19: Recursion VII

\[ \{p_1\} s_1.m_1(\bar{t}_1)\{q_1\}, \ldots, \{p_n\} s_n.m_n(\bar{t}_n)\{q_n\} \vdash \{p\} s\{q\}, \]
\[ \{p_1\} s_1.m_1(\bar{t}_1)\{q_1\}, \ldots, \{p_n\} s_n.m_n(\bar{t}_n)\{q_n\} \vdash \]
\[ \{p_i\} \begin{array}{l} \text{begin local this }, \bar{u}_i := s_i, \bar{t}_i; S_i \end{array} \text{ end } \{q_i\}, i \in \{1, \ldots, n\}, \]
\[ \{p\} S\{q\} \]

where \(m_i(\bar{u}_i \:: \ S_i \in D\) for \(i \in \{1, \ldots, n\}\).
Proof System PoP: the system consists of the group of axiom and rules 1 – 6, 10, 14, 18, 19, and A2 – A7.

Example 6.18:

$$\{\text{true}\} y.setx(z) \{y.x = z\}$$
Rule 20: Recursion VIII

\[
\{p_1\} s_1 . m_1(\overline{e}_1)\{q_1\}, \ldots, \{p_n\} s_n . m_n(\overline{e}_n)\{q_n\} \vdash \{p\} s\{q\},
\]

\[
\{p_1 \land t < z\} s_1 . m_1(\overline{e}_1)\{q_1\}, \ldots, \{p_n \land t < z\} s_n . m_n(\overline{e}_n)\{q_n\} \vdash
\]

\[
\{p_i \land t = z\} \begin{aligned}
beginal local this, \overline{u}_i := s_i, \overline{e}_i; S_i \end{aligned} \end{local} \{q_i\}, i \in \{1, \ldots, n\}
\]

\[
p_i \rightarrow t \geq 0 \land s_i \neq \text{null} , i \in \{1, \ldots, n\}
\]

\[
\{p\} S\{q\}
\]

where \( m_i(\overline{u}_i) :: S_i \in D \), and \( z \) is an integer variable that does not occur in \( p_i, t, q_i, S_i \) for \( i \in \{1, \ldots, n\} \).
Proof system TOP: consists of the group of axioms and rules 1 – 4, 6, 7, 10, 14, 18, 20 and A3 – A7.
Example:

\[ \{ y \neq \text{null} \} y.setx(z)\{y.x = z\} \]
Object Creation

Syntax:  \( S ::= u := \text{new} \)

Semantics:  \(< u := \text{new} , \sigma > \rightarrow < E, \sigma[u := \text{new}] >\)

Axiom 21: Object Creation

\[
\{p[x := \text{new}]\} x := \text{new} \{p\}
\]

Rule 22: Object Creation

\[
p' \rightarrow p[u := x] \\
\{p'[x := \text{new}]\} u := \text{new} \{p\}
\]
What does the program do?

Method definition:

\[
\text{setnext}(u) :: \text{next} := u
\]

\[
\text{insert} :: \text{begin} \quad \text{local}
\]
\[
z := \text{next};
\]
\[
\text{next} := \text{new};
\]
\[
\text{next.setnext}(z)
\]
\[
\text{end}
\]

What does the program \text{insert} do?
Proof System with Object Creation

- **Partial correctness proof system (POC)** consists of rule 1 – 6, 10, 14, 18, 19, 21, 22, and A2 – A7.
- **Total correctness proof system (TOC)** consists of rule 1 – 4, 6, 7, 10, 14, 18, 20, 21, 22, and A3 – A7.