CS143: Operational Semantics

David L. Dill
Stanford University
Operational Semantics

• Semantic Analysis (wrap up)
  • Objects

• Operational Semantics
  • Introduction
  • Constants and Variables
  • Conditionals and Loops
  • Let
  • New
Objects
Principle

If class B ≤ class A, then any code that operates on an object of type A must work on an object of type B.

- Attributes inherited from A must be in same position
- Dispatch must find correct method (even if method is redefined in B)
Object Layout

<table>
<thead>
<tr>
<th>Class tag</th>
<th>Offset 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object size</td>
<td>4</td>
</tr>
<tr>
<td>dispatch ptr</td>
<td>8</td>
</tr>
<tr>
<td>attr 1</td>
<td>12</td>
</tr>
<tr>
<td>attr 2</td>
<td>16</td>
</tr>
<tr>
<td>...</td>
<td>20</td>
</tr>
</tbody>
</table>

Offsets are known at compile time

Size & offset of each attribute are computed by analyzing class definition
Example

class A {
    a: Int ← 0;
    d: Int ← 1;
    f(): { ← };
}

class C inherits A {
    c: Int ← 3;
    h(): Int { ← 3 };
}

class B inherits A {
    b: Int ← 2;
    f(): { ← };
    g(): { ← };
}
Layout and Inheritance

Class B inherits A ...

Instance of A   Instance of B   Instance of C

A: tag 5
- a
- d

B: tag 7
- a
- d
- b

C: tag 7
- a
- d
- c
Example

class A \{ 
    a: Int ← 0;
    d: Int ← 1;
    f(): \{ \_ \};
\}

class C inherits A \{ 
    c: Int ← 3;
    h(): Int \{ \_ \};
\}

class B inherits A \{ 
    b: Int ← 2;
    f(): \{ \_ \};
    g(): \{ \_ \};
\}
Dispatch

Want f at same offset in dispatch table, whether inherited or redefined.
Dispatch

Want f at same offset in dispatch table, whether inherited or redefined.

A dispatch

B dispatch

C dispatch
Using Dispatch Tables

Dynamic dispatch \( e.f() \)

Code:

evaluate \( e \) \rightarrow \) ptr to object
get ptr to dispatch table
get ptr to method from dispatch table
call method with \( \text{self} = e \) value
Operational Semantics
Goal: Precise Mathematical Definition of Runtime Semantics

Why not generate code?

Too much detail
Machine-specific
Rules of Inference

Type checking: \[ \text{Context} \vdash e : T \]

Operational semantics:

\[ \text{Context} \vdash e : v \]

\(e\) has value \(v\) in the context "Context"

"Context" - state of a mathematical virtual machine.
Problem: Assignment

In math, values of variables can’t change.

How to handle assignment in an expression?

S — models memory ("store")
Maps "locations" to values
Updating a location makes a new copy

E — maps variables to locations ("environment")
Environment

\[ E = [x \mapsto l_1, y \mapsto l_2] \]

\[ E[x] = l_1 \]
\[ E[y] = l_2 \]
Store

Maps memory locations to values

\[ S = \{ l_1 \rightarrow 5, \ l_2 \rightarrow 7 \} \]

\[ S(l_1) = 5 \]

\[ S(s_2) = 7 \]

Why separate \( E_1 \), \( S \)?
Objects

\( x (a_1 = l_1, a_2 = l_2, \ldots, a_n = l_n) \)

- Class of object
- Attribute
- Location of attribute
Basic Classes

Int(5)     The integer 5
Bool(true)  The Boolean true
String(4l, "cool")  The string "cool" of length 4
Context

so, $E, S \vdash e : v, S'$

↑ ↑ ↑ Store self environment object

expression

value updated store (e may have assigned variables, allocated new locations)

(evaluation of e may not terminate.
That's ok.)
Constants and Variables
so, $E, S \vdash \text{true} : \text{Bool}(\text{true}), S$

Expression $\text{true}$ does not have side effects so $S$ is unchanged.
So, $E, S |- \text{true} : \text{Bool}(\text{true}), S$

---

$i$ is an integer literal

So, $E, S |- i : \text{Int}(i), S$

---

$n$ is the length of $s$

So, $E, S |- \text{String}(n, s), S$

---

$s$ is a string literal
Value of a Variable

\[ E(id) = \text{lid} \quad \text{← look up location of id} \]
\[ S(\text{lid}) = v \quad \text{← look up contents of "address" lid} \]

so, \[ E, S \vdash id : v, S \]
Value of a Variable

\[ E(id) = \text{lid} \quad \text{← look up location of id} \]
\[ S(\text{lid}) = v \quad \text{← look up contents of "address" l} \]

so, \( E, S \vdash id : v, S \)

\[ \text{no side effects} \]
so, E, S T self: so, S

Self evaluates to so

no side effects
Updating $S$

$S[v/l]$ is a new store, which is exactly the same as $S$, except $S(l) = v$

$S = [a \mapsto 1, \ b \mapsto 2]$

$S[3/a] = [a \mapsto 3, \ b \mapsto 2]$

($S$ itself does not change)
Assignment

\[ s_0, E, S \vdash e : v, S_1 \]

\[ E(\text{id}) = \text{id} \]

\[ S_2 = S_1 [v/\text{id}] \]

\[ s_0, E, S \vdash \text{id} \leftarrow e : v, S_2 \]

Informally:
1. get value of \( e \) (may change \( S \))
2. get location of \( \text{id} \)
3. update location with value of \( e \).
Announcements

• PA4/PA5 assigned
  • Very hard – start immediately
  • PA5 is completely optional. Don’t do it unless your PA4 is nearly perfect.
  • Submit PA4/PA5 separately so optimizations in PA5 don’t break PA4.
Conditionals and Loops
Conditional

\[ \text{so, } E, S \vdash e_1 : \text{Bool(}\text{true}\text{)}, S_1 \]
\[ \text{so, } E, S_1 \vdash e_2 : V, S_2 \]
\[ \text{so, } E, S \vdash \text{if } e, \text{ then } e_2 \text{ else } e_3 : V, S_2 \]

Only covers case where \( e_1 = \text{true} \)

\( e_1 \) might have side effects

\( \text{eval then part} \)

Value of \( e_2 \)

Results of assignments in \( e_1, e_2 \)

\( e_1 \) is guaranteed to return a value of type \( \text{Bool} \) because we only run programs that type check.
\[ \text{so, } E, S \vdash e_1 : v_1, S_1 \]
\[ \text{so, } E, S_1 \vdash e_2 : v_2, S_2 \]
\[ \ldots \]
\[ \text{so, } E, S_{n-1} \vdash e_n : v_n, S_n \]
\[ \text{so, } E, S \vdash \{ e_1, e_2, \ldots, e_n \} : v_n, S_n \]

*changes to store accumulate*

\text{Value of } e_n \text{ is returned.}
While

so, E, S \leftarrow \text{Bool}(\text{false}), S_1

so, E, S \leftarrow \text{while } e_1 \text{ loop } e_2 \text{ pool : void, } S_1

Loop terminates if $e_1$ is false
$e_2$ is not evaluated
loops always return void
\[ s_0, E, S_1 \leftarrow e_1: \text{Bool}(\text{true}), S_1 \]
\[ s_0, E, S_1 \leftarrow e_2: v, S_2 \]
\[ s_0, E, S_2 \leftarrow \text{while } e_1 \text{ loop } e_2 \text{ pool: } \text{void}, S_3 \]
\[ s_0, E, S_1 \leftarrow \text{while } e_1 \text{ loop } e_2 \text{ pool: } \text{void}, S_3 \]

\text{loop test is true} \quad \text{1st iteration gives } S_2
\text{start loop in store resulting from first iteration.}
\text{termination}
Let
New Locations

Need a "mathematical" malloc.

Get a new location that does not already appear in S.

\[ \text{new} = \text{newloc}(S) \]

\[ \text{needs } S \text{ so it can return a location that is not already in use in } S. \]
so, \( E, S \vdash e_1 : v_1, S_1 \)
\[
l_{\text{new}} = \text{newloc}(S_1)
\]
so, \( E[l_{\text{new}}/\text{id}], S_1[v_1/l_{\text{new}}] \vdash e_2 : v_2, S_2 \)
so, \( E, S \vdash \text{let id : } T \leftarrow e_1 \text{ in } e_2 : v_2, S_2 \)

Informally:
1. Create a new location \( l_{\text{new}} \)
2. \( \text{id} \rightarrow l_{\text{new}} \) in \( E \)
3. \( l_{\text{new}} \rightarrow e_1 \text{ value} \) in store
new
new $T$

Make new locations for attributes of $T$

Set attributes to default values

Evaluate initializers and assign to attributes

Return the new object
Default Value

D\_int = Int(0)
Db\_ool = Bool(false)
D\_string = String(0, "")
D\_a = Void (for other classes A)
Notation:

\[ \text{class } \langle x \rangle = (a_i : T_i \leftarrow e_i, \ldots, a_n : T_n \leftarrow e_n) \]
\[ T_0 = \text{if } (T = \text{SELF\_TYPE} \text{ and } s0 = X(\ldots)) \text{ then } X \text{ else } T \]

find the type of object

\[ V, E, S \vdash \text{new } T : V, S \]
\( T_0 = \text{if } (T = \text{SELF-TYPE and } s_0 = X(\ldots)) \text{ then } X \text{ else } T \)

class \( (T_0) = (a_1: T_1 \leftarrow e_1, \ldots, a_n: T_n \leftarrow e_n) \)

\text{get the class definition}

\[
V, E, S \vdash \text{new } T : V, S, T_2
\]
\[ T_0 = \begin{cases} \text{LET } (T = \text{SELF-TYPE and } s_0 = X(\ldots)) \text{ then } X \text{ else } T \\ \text{class } (T_0) = (a_1 : T_1 \leftarrow e_1, \ldots, a_n : T_n \leftarrow e_n) \\ k_i = \text{new loc } (s) \text{ for } i = 1, \ldots, n \\ \uparrow \text{ new locations for attributes} \end{cases} \]

\[ V, E, S \vdash \text{new } T : V, S \]
\( T_0 = \begin{cases} \text{if} & (T = \text{SELF\_TYPE} \text{ and } s_0 = X(...) ) \text{ then } X \text{ else } T \\ \text{class} & (T_0) = (a_1 : T_1 \leftarrow e_1, \ldots, a_n : T_n \leftarrow e_n) \\ l_i = \text{new loc}(s) \text{ for } i = 1, \ldots, n \\ v = T_0 (a_1 = l_1, \ldots, a_n = l_n) \\ \uparrow \text{ make new object } (\text{attributes } \rightarrow \text{locations}) \\ \end{cases} \)

\( \forall, E, S \vdash \text{new } T : V, S_2 \)
\[ T_0 = \text{if } (T = \text{SELF\_TYPE and } s_0 = X(\ldots)) \text{ then } X \text{ else } T \]

\[ \text{class } (T_0) = (a_1 : T_1 \leftarrow e_1, \ldots, a_n : T_n \leftarrow e_n) \]

\[ l_i = \text{new loc}(S) \text{ for } i = 1, \ldots, n \]

\[ v = T_0(a_1 = l_1, \ldots, a_n = l_n) \]

\[ S_1 = S(D_{T_1} / l_1, \ldots, D_{T_n} / l_n) \]

\[ \uparrow \text{ assign locations to default values} \]

\[ V, E, S \vdash \text{new } T : V, S_2 \]
\[ T_0 = \begin{cases} A & (T = \text{SELF-TYPE} \text{ and } s_0 = X(\ldots)) \text{ then } X \text{ else } T \\ \text{class } (T_0) = (a_1 : T_1 \leftarrow e_1, \ldots, a_n : T_n \leftarrow e_n) \\ l_i = \text{new loc}(s) \text{ for } i = 1, \ldots, n \\ v = T_0 (a_1 = l_1, \ldots, a_n = l_n) \\ S_1 = S(D_{T_1}/l_1, \ldots, D_{T_n}/l_n) \\ E' = \{ a_i : l_1, \ldots, a_n : l_n \} \leftarrow \text{bind attributes as variables} \\ \end{cases} \]

\[ V, E, S \vdash \text{new } T : V, S, \]
\[ T_0 = \text{if } (T = \text{SELF-TYPE and } s_0 = X(...)) \text{ then } X \text{ else } T \]

\text{class } (T_0) = (a_1 : T_1 \leftarrow e_1, \ldots, a_n : T_n \leftarrow e_n)

\[ k_i = \text{new loc}(S) \text{ for } i = 1, \ldots, n \]

\[ v = T_0(a_1 = l_1, \ldots, a_n = l_n) \]

\[ S_1 = S(D_{T_1}/l_1, \ldots, D_{T_n}/l_n) \]

\[ E' = \{ a_i : l_1, \ldots, a_n : l_n \} \]

\[ v, E', S_1 \vdash \{ a_i \leftarrow e_i, \ldots, a_n \leftarrow e_n \} : V_n, S_2 \]

\[ v, E, S \vdash \text{new } T : V_2, S_2 \]
$T_0 = \begin{array}{l} \text{if} (T = \text{SELF}-\text{TYPE and } s_0 = X(\ldots)) \text{ then } X \text{ else } T \\ \text{class } (T_0) = (a_1 : T_1 \leftarrow e_1, \ldots, a_n : T_n \leftarrow e_n) \\
\hat{l}_i = \text{new loc}(S) \text{ for } i = 1, \ldots, n \\
\nu = T_0 (a_1 = \hat{l}_1, \ldots, a_n = \hat{l}_n) \\
S_1 = S(D_{T_1} / \hat{l}_1, \ldots, D_{T_n} / \hat{l}_n) \\
E' = \{a_i : l_1, \ldots, a_n : l_n\} \\
v, E', S_1 \vdash \{a_i \leftarrow e_i, \ldots, a_n \leftarrow e_n\} : V_n, S_2 \\
v, E, S \vdash \text{new T} : V_n, S_2 \\
\end{array}$
\[ T_0 = \text{if } (T = \text{SELF-TYPE and } s_0 = X(\ldots)) \text{ then } X \text{ else } T \]

class \((T_0) = (a_1 : T_i \leftarrow e_i, \ldots, a_n : T_n \leftarrow e_n)\)

\[ l_i = \text{new loc}(S) \text{ for } i = 1, \ldots, n \]

\[ v = T_0 \ (a_1 = l_1, \ldots, a_n = l_n) \]

\[ S_1 = S \ (D_{T_i} / l_1, \ldots, D_{T_n} / l_n) \]

\[ E' = \{a_i : l_1, \ldots, a_n : l_n\} \]

\[ v, E' , S_1 \vdash \{a_i \leftarrow e_i, \ldots, j \leftarrow e_{n3} : V_n, S_2\} \]

\[ v, E_j, S \vdash \text{new } T : V_j, S_2 \]

What about \(V_n\)?

How do we find \(V_0, A_1\) value?