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

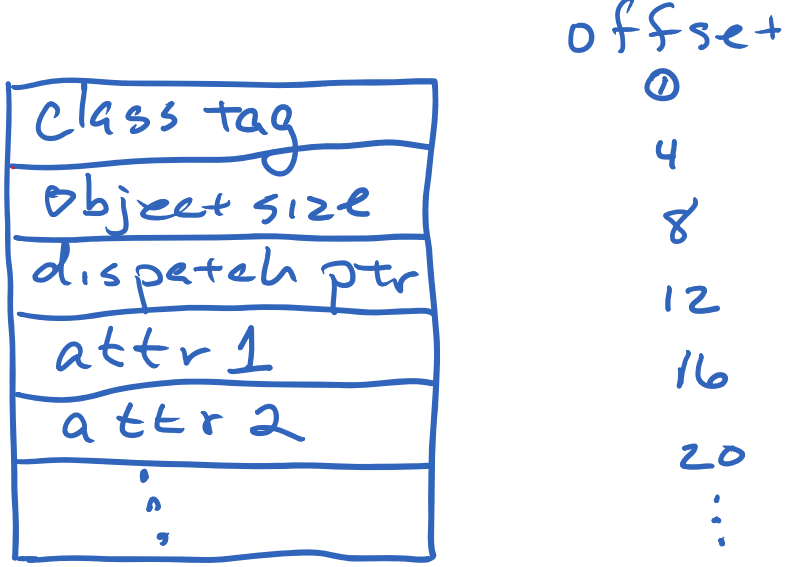
Objects

Principle

If class B $\leq$ 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



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 B inherits A {
    b : Int ← 2;
    f() : { ~~ };
    g() : { ~~ };
};
```

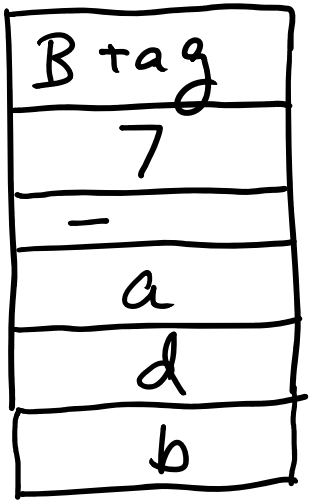
```
class C inherits A {
    c : Int ← 3;
    h() : Int { ~~ };
};
```

Layout and Inheritance

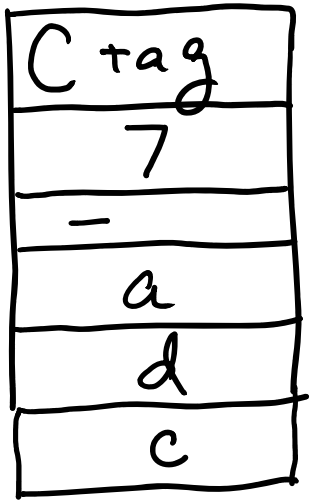
Class B inherits A ...

Instance of A

Instance of B



Instance of C



Example

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

```
class B inherits A {
    b : Int ← 2;
    f() : { ~~ };
    g() : { ~~ };
};
```

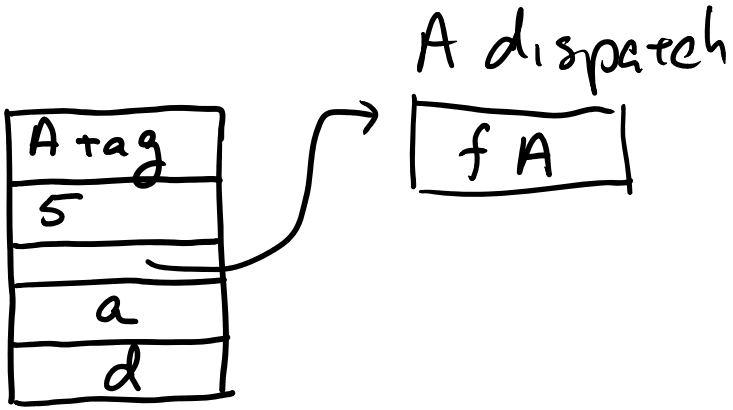
```
class C inherits A {
    c : Int ← 3;
    h() : Int { ~~ };
};
```

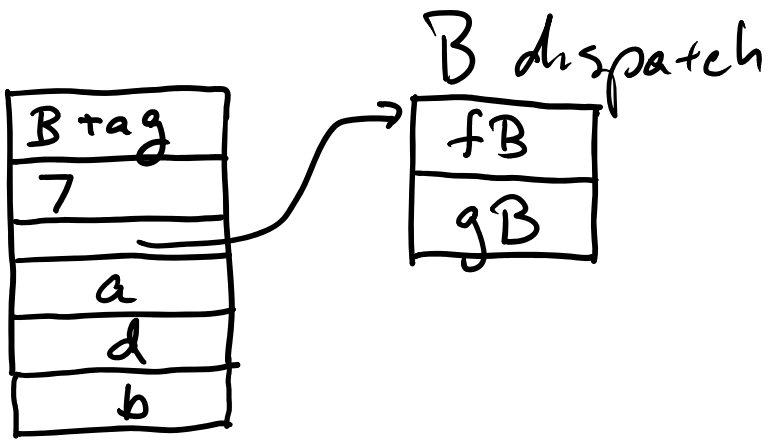
Dispatch

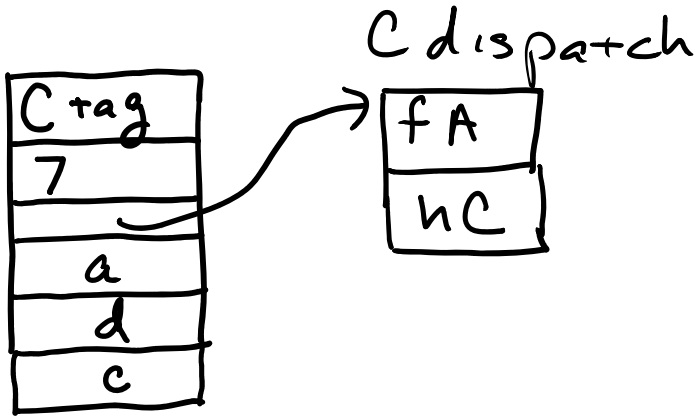
Want f at same offset in dispatch table, whether inherited or redefined.

Dispatch

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







Using Dispatch Tables

Dynamic dispatch e.f()

Code:

evaluate e → ptr to object
get ptr to dispatch table
get ptr to method from dispatch table
call method with 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: $Context \vdash e : T$

Operational Semantics:

$Context \vdash e : v$

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")

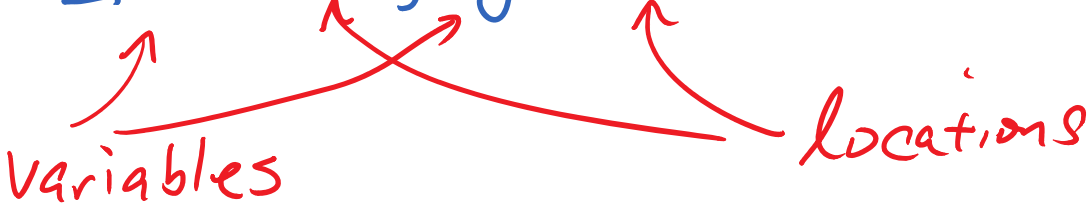
Maps "locations" to values

Updating a location makes a new copy

E – maps variables to locations ("environment")

Environment

$$E = [x \mapsto \ell_1, y \mapsto \ell_2]$$



$$E[x] = \ell_1$$

$$E[y] = \ell_2$$

Store

Maps memory locations to values

$$S = [l_1 \mapsto 5,\ l_2 \mapsto 7]$$

$$S(l_1) = 5$$

$$S(s_2) = 7$$

Why separate E, S?

Objects

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

Class of object

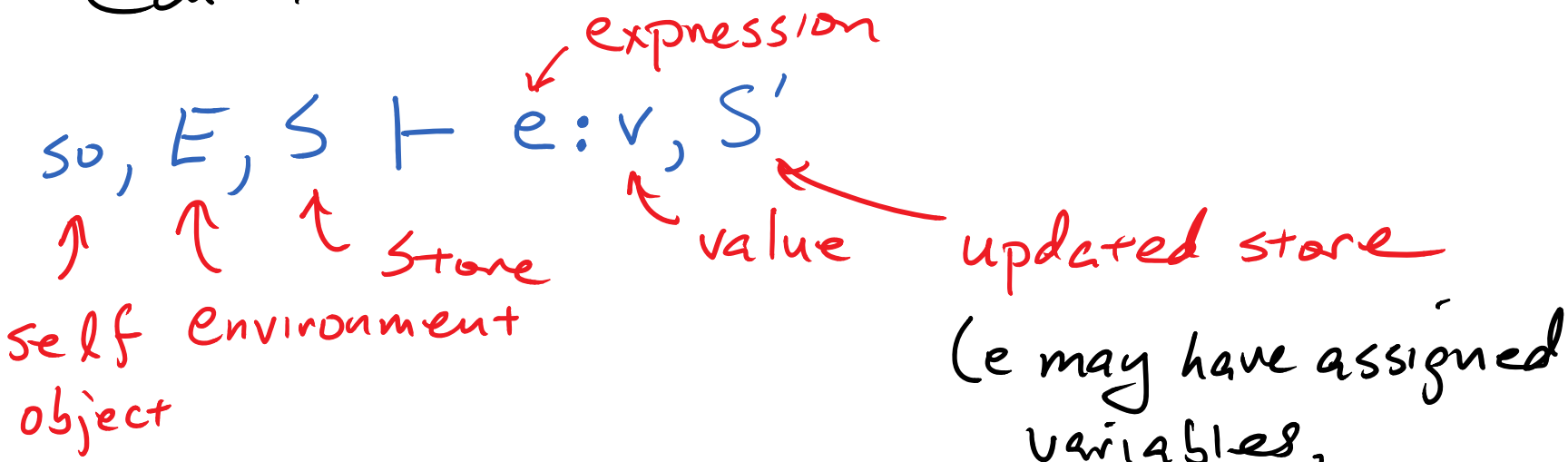
Attribute

Location of attribute

Basic Classes

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

Context



(evaluation of e may not terminate. That's ok.)

Constants and Variables

$$\frac{}{so, E, S \vdash true : Bool(true), S}$$

Expression true does not have side effects

so S is unchanged.

$$\frac{}{so, E, S \vdash true : Bool(true), S}$$

$$\frac{i \text{ is an integer literal}}{so, E, S \vdash i : Int(i), S}$$

$$\frac{\begin{array}{c}s \text{ is a string literal} \\ n \text{ is the length of } s\end{array}}{so, E, S \vdash String(n, s), S}$$

Value of a Variable

$$\frac{E(id) = l_{id} \quad \text{look up location of id} \qquad S(l_{id}) = v \quad \text{look up contents of "address" } l}{so, E, S \vdash id : v, S}$$

Value of a Variable

$$\frac{E(id) = l_{id} \quad \text{look up location of id} \qquad S(l_{id}) = v \quad \text{look up contents of "address" } l}{so, E, S \vdash id : v, S}$$

no side effects

$$\frac{}{s_0, E, S \vdash \texttt{self} : s_0, S}$$

self evaluates to s_0

no side effects

Updating S

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

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

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

(S itself does not change)

Assignment

$$\frac{s_0, E, S \vdash e : v, S_1 \qquad E(id) = l_{id} \qquad S_2 = S_1[v / l_{id}]}{s_0, E, S \vdash id \leftarrow e : v, S_2}$$

Informally: (1) get value of e (may change S)

(2) get location of 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

$$\frac{s_0, E, S \vdash e_1 : \text{Bool}(\text{true}), S_1 \qquad s_0, E, S_1 \vdash e_2 : V, S_2}{s_0, E, S \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : V, S_2}$$

Only covers case where $e_1 = \text{true}$

e_1 might have side effects

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 Bool because we only run programs that type check.

$$\frac{\begin{array}{l} s_0, E, S \vdash e_1 : v_1, S_1 \\ s_0, E, S_1 \vdash e_2 : v_2, S_2 \\ \cdots \\ s_0, E, S_{n-1} \vdash e_n : v_n, S_n \end{array}}{s_0, E, S \vdash \{e_1; e_2; \cdots; e_n;\} : v_n, S_n}$$

changes to store accumulate

value of e_n is returned.

While

$$\frac{so, E, S \vdash Bool(false), S_1}{so, E, S \vdash \text{while } e_1 \text{ loop } e_2 \text{ pool} : void, S_1}$$

Loop terminates if e_1 is false

e_2 is not evaluated

loops always return void

$$\frac{s_0, E, S \vdash e_1 : \text{Bool(true)}, S_1 \quad s_0, E, S_1 \vdash e_2 : V, S_2 \quad s_0, E, S_2 \vdash \text{while } e_1 \text{ loop } e_2 \text{ pool} : \text{void}, S_3}{s_0, E, S \vdash \text{while } e_1 \text{ loop } e_2 \text{ pool} : \text{void}, S_3}$$

S_1 ← loop test is true

S_2 ← 1st iteration gives S_2

S_3 ← S_3 at termination

start loop in store resulting from first iteration.

Let

New Locations

Need a "mathematical" malloc.

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

$\ell_{new} = \text{newloc}(S)$

needs S so it can return a location that is not already in use in S.

$$\frac{s_0, E, S \vdash e_1 : v_1, S_1 \quad l_{new} = newloc(S_1) \quad s_0, E[l_{new}/id], S_1[v_1/l_{new}] \vdash e_2 : v_2, S_2}{s_0, E, S \vdash let\ id : T \leftarrow e_1\ in\ e_2 : v_2, S_2}$$

Informally: (1) Create a new location l_{new}

(2) $id \mapsto l_{new}$ in E

(3) $l_{new} \mapsto e_1$ 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_A — default value for class A

$D_{int} = Int(0)$

$D_{bool} = Bool(false)$

$D_{string} = String(0, "")$

$D_A = void$ (for other classes A)

Notation:

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

attributes types initializers

T_0 = if (T = SELF_TYPE and $s_0 = X(\ldots)$) then X else T

find the type of object

$V, E, S \vdash \text{new } T : V, S_2$

T_0 = if (T = SELF_TYPE and $s_0 = X(\ldots)$) then X else T

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

get the class definition

$V, E, S \vdash$ new $T : V, S_2$

$T_0 =$ if $(T =$ SELF_TYPE and $so = X(\ldots))$ then X else T

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

$l_i = newloc(S)$ for $i = 1, \ldots n$

↑ new locations for attributes

$V, E, S \vdash$ new $T : V, S_2$

$T_0 =$ if $(T =$ SELF_TYPE and $s_0 = X(\ldots))$ then X else T

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

$l_i = newloc(S)$ for $i = 1, \ldots n$

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

↑ make new object (attributes → locations)

$V, E, S \vdash$ new $T : V, S_2$

$T_0 =$ if $(T =$ SELF_TYPE and $s_0 = X(\ldots))$ then X else T

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

$l_i = newloc(S)$ for $i = 1, \ldots n$

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

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

↑ assign locations to default values

$V, E, S \vdash$ new $T : V, S_2$

$T_0 =$ if $(T =$ SELF_TYPE and $s_0 = X(\ldots))$ then X else T

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

$l_i = newloc(S)$ for $i = 1, \ldots n$

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

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

$E' = \{a_1 : l_1, \ldots, a_n : l_n\}$ ← bind attributes as variables

---

$V, E, S \vdash$ new $T : V, S_2$

$T_0 = \text{if } (T = \text{SELF\_TYPE and } so = X(\ldots)) \text{ then } X \text{ else } T$

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

$l_i = newloc(S) \text{ for } i = 1, \ldots n$

$V = T_0(a_1 = l_1, \ldots, a_n = l_n)$

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

$E' = \{a_1 : l_1, \ldots, a_n : l_n\}$

$V, E', S_1 \vdash \{a_1 \leftarrow e_1; \ldots; a_n \leftarrow e_n\} : V_n, S_2$

evaluate initializers in E', S_j

$V, E, S \vdash \text{new } T : V, S_2$

$$T_0 = \text{if } (T = \text{SELF\_TYPE and } so = X(\ldots)) \text{ then } X \text{ else } T$$

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

$$l_i = newloc(S) \text{ for } i = 1, \ldots n$$

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

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

$$E' = \{a_1 : l_1, \ldots, a_n : l_n\}$$

$$v, E', S_1 \vdash \{a_1 \leftarrow e_1; \ldots; a_n \leftarrow e_n\} : V_n, S_2$$

$$V, E, S \vdash \text{new } T : V, S_2$$

$T_0 = \text{if } (T = \text{SELF\_TYPE and } s_0 = X(\ldots)) \text{ then } X \text{ else } T$

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

$l_i = newloc(S) \text{ for } i = 1, \ldots n$

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

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

$E' = \{a_1 : l_1, \ldots, a_n : l_n\}$

$v, E', S_1 \vdash \{a_1 \leftarrow e_1; \ldots; a_n \leftarrow e_n\} : v_n, S_2$

$v, E, S \vdash \text{new } T : v, S_2$

What about v_n?

How do we find $v.a_1$ value?