

Lecture 13 — Objects and inheritance

- Practice with SOS rules
- SOS rules for objects
- Inheritance in Java
- SOS rules for inheritance

Some SOS rules from MP 5

(Plus-Int)

$$\begin{aligned} e_1 + e_2, \sigma, \pi &\Downarrow \text{IntV } (i_1 + i_2) \\ e_1, \sigma, \pi &\Downarrow \text{IntV } i_1 \\ e_2, \sigma, \pi &\Downarrow \text{IntV } i_2 \end{aligned}$$

(Var)

$$x, \sigma, \pi \Downarrow \sigma(x) \quad \text{if } \sigma(x) \neq \perp$$

(Method-Call)

$$\begin{aligned} x.f(e_1, \dots, e_n), \sigma, \pi &\Downarrow v \\ e_1, \sigma, \pi &\Downarrow v_1 \\ &\vdots \\ e_n, \sigma, \pi &\Downarrow v_n \\ sl, \sigma_0, \pi &\Rightarrow \sigma'_0 \\ e, \sigma'_0, \pi &\Downarrow v \end{aligned}$$

where $\sigma_0 = [(x_1, v_1); \dots; (y_0, \text{NullV}); \dots]$
and f is defined in π as
 $\text{Method}(t, f, [x_1; \dots], [y_1; \dots], sl, e)$

(Mult-Int)

$$\begin{aligned} e_1 * e_2, \sigma, \pi &\Downarrow \text{IntV } (i_1 * i_2) \\ e_1, \sigma, \pi &\Downarrow \text{IntV } i_1 \\ e_2, \sigma, \pi &\Downarrow \text{IntV } i_2 \end{aligned}$$

(Int-Const)

$$i, \sigma, \pi \Downarrow \text{IntV } i$$

(Stmt-List)

$$\begin{aligned} s_1; \dots; s_n, \sigma_1, \pi &\Rightarrow \sigma_{n+1} \\ s_1, \sigma_1, \pi &\Rightarrow \sigma_2 \\ &\vdots \\ s_n, \sigma_n, \pi &\Rightarrow \sigma_{n+1} \end{aligned}$$

(Stmt-List-0)

$$[], \sigma_1, \pi \Rightarrow \sigma_1$$

(Assignment)

$$\begin{aligned} x=e, \sigma, \pi &\Rightarrow \sigma[v/x], \quad \text{if } \sigma(x) \neq \perp \\ e, \sigma, \pi &\Downarrow v \end{aligned}$$

Proofs in SOS

- Use SOS rules to obtain results of entire computations.

$3+(4*5), \sigma, \pi \Downarrow \text{IntV } 23$	(PLUS-INT)
$3, \sigma, \pi \Downarrow \text{IntV } 3$	(INT-CONST)
$4*5, \sigma, \pi \Downarrow \text{IntV } 20$	(MULT-INT)
$4, \sigma, \pi \Downarrow \text{IntV } 4$	(INT-CONST)
$5, \sigma, \pi \Downarrow \text{IntV } 5$	(INT-CONST)

- Exercise:

$(3*4) + (5*2), \sigma, \pi \Downarrow \text{IntV } 22$	(INT-PLUS)
---	------------

Proofs in SOS (cont.)

- Suppose π contains class Main with method: `int f (int a)`
`{ return a; }`.

x=5;y=null.f(x),	$\sigma_0, \pi \Rightarrow \sigma_2$	(STMTLIST)
x=5,	$\sigma_0, \pi \Rightarrow \sigma_1$	(ASSIGNMENT)
5,	$\sigma_0, \pi \Downarrow \text{IntV } 5$	(INT-CONST)
y=null.f(x),	$\sigma_1, \pi \Rightarrow \sigma_2$	(ASSIGNMENT)
null.f(x),	$\sigma_1, \pi \Downarrow \text{IntV } 5$	(METHOD-CALL)
x,	$\sigma_1, \pi \Downarrow \text{IntV } 5$	(VAR)
[],	$\sigma_3, \pi \Rightarrow \sigma_3$	(STMTSEQ-0)
a,	$\sigma_3, \pi \Downarrow \text{IntV } 5$	(VAR)

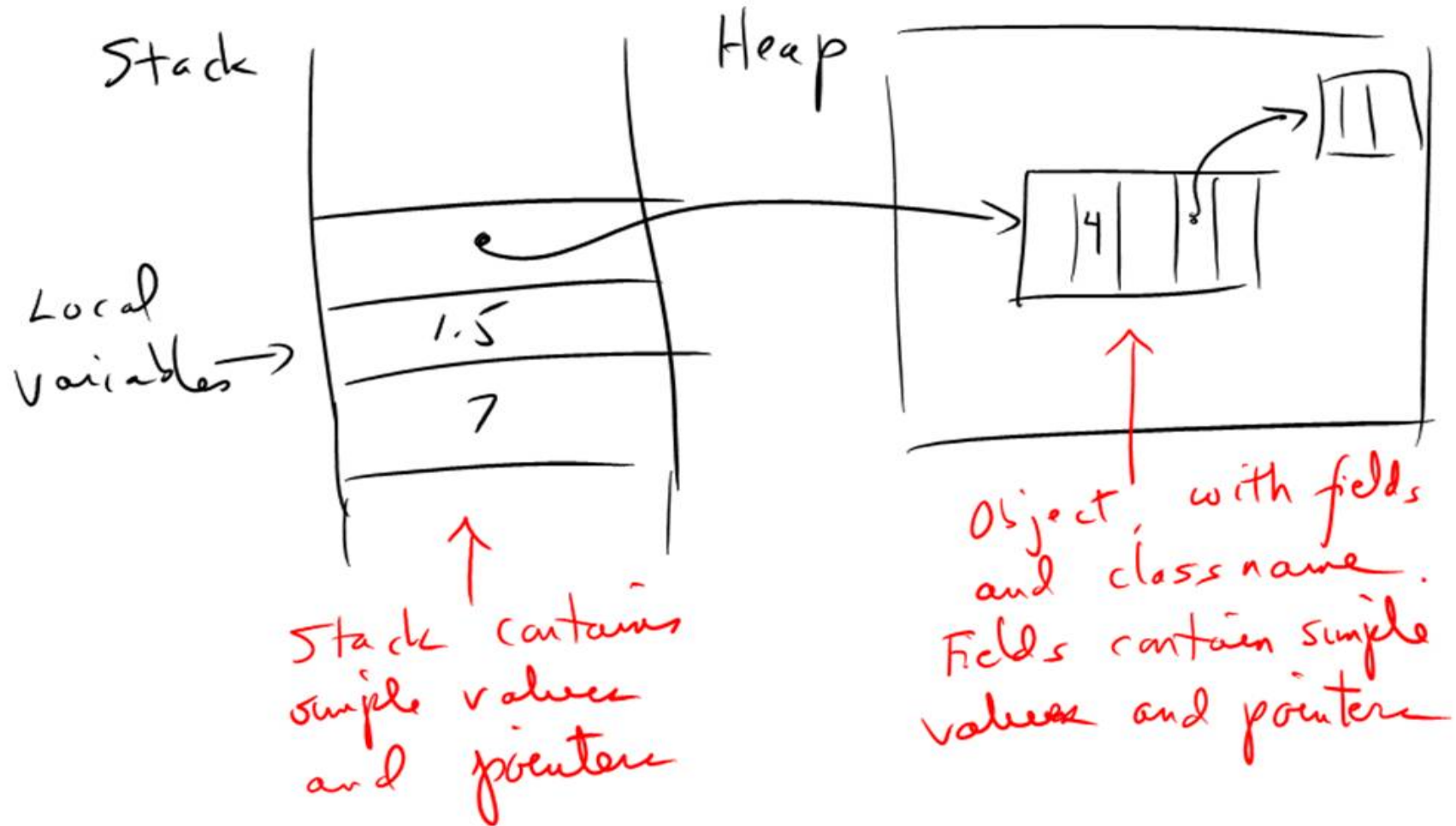
where $\sigma_0 = [(x, \text{IntV } 0); (y, \text{IntV } 0)]$, $\sigma_1 = [(x, \text{IntV } 5); (y, \text{IntV } 0)]$, $\sigma_2 = [(x, \text{IntV } 5); (y, \text{IntV } 5)]$, and $\sigma_3 = [(a, \text{IntV } 5)]$.

Proofs in SOS (cont.)

- **Exercise:** Same statements, but `int f (int x) return a+1;`

`x=5;y=null.f(x),  $\sigma_0, \pi \Rightarrow \sigma_5$  ( $\sigma_5 = [(x, \text{IntV } 5);(y, \text{IntV } 6)]$ )`

Stack and heap in Java



Two-level state in MJ

- Need to use same idea in implementation of MJ to implement objects correctly.
- Terminology: “stack” and “environment” synonyms; “heap” and “store” synonyms; “state” is a pair of an environment and a store.
- For MP6, have both environment and store. As in Java:
 - Environment contains simple value and pointers.
 - Store contains objects.
 - (*We will not implement arrays.*)

SOS rules (v. 2)

```
type varname = string
type classname = string
type stackvalue = IntV of int | StringV of string
                  | BoolV of bool | NullV | Location of location
and location = int
type environment = (varname * stackvalue) list
type heapvalue = Object of classname * environment
type store = heapvalue list
type state = environment * store
```

- In SOS rules, we still use σ for a state, but often write (ρ, η) instead, with ρ for environment and η for store.
- Consider evaluation of `new C`. Involves putting a new object in the store, so we must change the judgments for evaluation:

$$e, \sigma, \pi \Downarrow v, \eta$$

SOS rules (v. 2) (cont.)

```
type stackvalue = IntV of int | StringV of string
                  | BoolV of bool | NullV | Location of location
and location = int
type environment = (varname * stackvalue) list
type heapvalue = Object of classname * environment
type store = heapvalue list
type state = environment * store
```

- **Write an expression of type state for a state that contains variables x bound to 3 and y bound to an object of class C; C contains fields a and b , and in y these have integer values 4 and 5.**

SOS rules (v. 2) (cont.)

- Give the SOS rule for new:

(NEW) $\text{new } C(), (\rho, \eta), \pi \Downarrow$

- New form of SOS rules reflected in new type of eval:

```
let rec eval (e:exp) ((env,sto) as sigma:state) (prog:program)
  : stackvalue * store =
```

- and the corresponding clause in eval (you can assume any auxiliary functions you think useful):

```
| NewId c ->
```

Threading the store

- The major new thing is that evaluation of expressions can have side effects, in particular, they can cause the store to change. (Evaluation of an expression *cannot* cause the environment to change.)
- Therefore, need to take the store from any expression evaluation and pass it along to the next expression evaluation. Can never discard any changes that occur in the store.

$$\begin{array}{l} \text{(BINOP-LESS)} \quad e_1 < e_2, (\rho, \eta), \pi \Downarrow \text{BoolV } (i_1 < i_2), \eta'' \\ \quad \quad \quad e_1, (\rho, \eta), \pi \Downarrow \text{IntV } (i_1), \eta' \\ \quad \quad \quad e_2, (\rho, \eta'), \pi \Downarrow \text{IntV } (i_2), \eta'' \end{array}$$

SOS rules (v. 2) (cont.)

(NOT) $!e, (\rho, \eta), \pi \Downarrow$

(INT-MULT) $e_1 * e_2, (\rho, \eta), \pi \Downarrow$

SOS rules (v. 2) (cont.)

(VAR) $x, (\rho, \eta), \pi \Downarrow$

(FIELD) $x, (\rho, \eta), \pi \Downarrow$

(METHOD-CALL)

$e_0.f(e_1, \dots, e_n), (\rho, \eta), \sigma, \pi \Downarrow v$

SOS rules (v. 2) (cont.)

- Rules for statements actually don't change — they always passed the state along from one to the next — except for assignment.

$(\text{VARASGN}) \quad x=e, (\rho, \eta), \pi \Rightarrow$

$(\text{FIELDASGN}) \quad x=e, (\rho, \eta), \pi \Rightarrow$

Inheritance in Java

// EXAMPLE 1

```
class B {  
    string f() { return this.g(); }  
    string g() { return "B"; } }
```

```
class C extends B {  
    string g() { return "C"; } }
```

```
x = new B(); y = new C();  
x.f(); // ?  
y.f(); // ?
```

// EXAMPLE 2

```
class B { B aB;  
    void r() { aB = this; }  
    string s() { return aB.g(); }  
    string g() { return "B"; } }
```

```
class C extends B {  
    string g() { return "C"; } }
```

```
x = new B(); y = new C(); x.r(); y.r();  
x.s(); // ?  
y.s(); // ?
```

Inheritance in Java (cont.)

// EXAMPLE 3

```
class B {  
    B aB;  
    void q(B x) { aB = x; }  
    string s() { return aB.g(); }  
    string g() { return "B"; } }
```

```
class C extends B {  
    string g() { return "C"; } }
```

```
x = new B(); y = new C();  
x.q(x); x.s(); // ?  
x.q(y); x.s(); // ?  
y.q(y); y.s(); // ?  
y.q(x); y.s(); // ?
```

Inheritance in Java (cont.)

// EXAMPLE 4

```
class B {  
    string f() { return this.g(); } }  
    string g() { return "B"; } }
```

```
class C extends B {  
    B b;  
    string g() { return "C"; }  
    string f() { return b.g(); }  
    void h(B y) { b = y; } }
```

```
x = new B(); y = new C();  
y.h(y); y.f(); // ?  
y.h(x); y.f(); // ?
```

Principles of inheritance in Java and MJ

- Inheriting fields and methods:
 - Fields of all superclasses are included in the object.
 - The methods associated with an object include those of its class and all superclasses; if there is more than one method with the same name, the “closest” one is called.
- Changes in SOS rules (and in functions in MP 6):
 - To evaluate `new C()`, create an object consisting of all fields of `C` and inherited fields; class of the new object is still `C`.
 - To call `e0.f(e1, ..., en)`, evaluate `e0` and find its class; look for `f` in that class, or, if not found, in its superclass, and so on, until a definition of `f` is found.