

 $C m(\overline{C} \overline{x}) \{ return t; \}$

CL ::= class C extends C { \overline{C} \overline{f} ; K \overline{M} }

v ::=

(C) t

new $C(\overline{v})$

cast

values

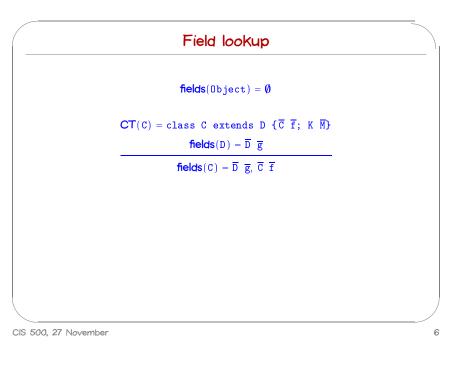
object creation

CIS 500, 27 November

class declarations

2

Subtyping Assume we have a (global, fixed) class table CT mapping class names to definitions. $CT(C) = class \ C \ extends \ D \ \{...\}$ C <: D C <: C C <: E

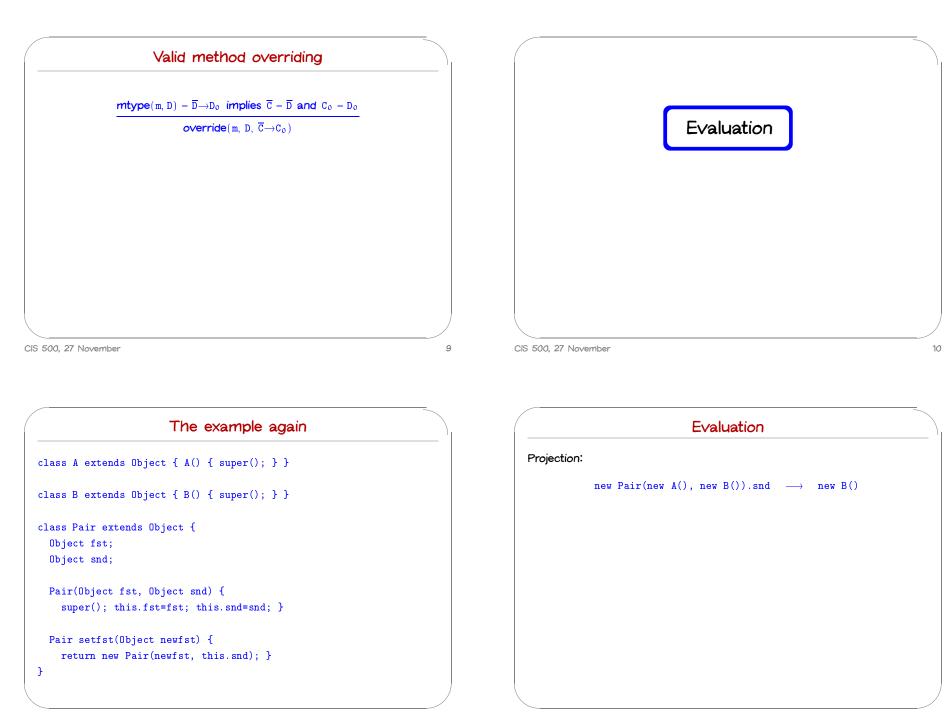


Method type lookup $CT(C) = class C extends D \{\overline{C} \ \overline{f}; K \ \overline{M}\}$ $B m (\overline{B} \ \overline{x}) \{return \ t;\} \in \overline{M}$ $mtype(m, C) = \overline{B} \rightarrow B$ $CT(C) = class C extends D \{\overline{C} \ \overline{f}; K \ \overline{M}\}$ $m is not defined in \ \overline{M}$ mtype(m, C) = mtype(m, D)

 $\begin{array}{l} \textbf{Method body lookup} \\ \textbf{CT}(C) = class C extends D \{\overline{C} \ \overline{f}; K \ \overline{M}\} \\ & B \ m \ (\overline{B} \ \overline{x}) \ \{return \ t;\} \in \overline{M} \\ \hline \textbf{mbody}(m, C) = (\overline{x}, t) \\ \textbf{CT}(C) = class C \ extends D \ \{\overline{C} \ \overline{f}; K \ \overline{M}\} \\ & \underline{m \ is \ not \ defined \ in \ \overline{M}} \\ \hline \textbf{mbody}(m, C) = \textbf{mbody}(m, D) \end{array}$

CIS 500, 27 November

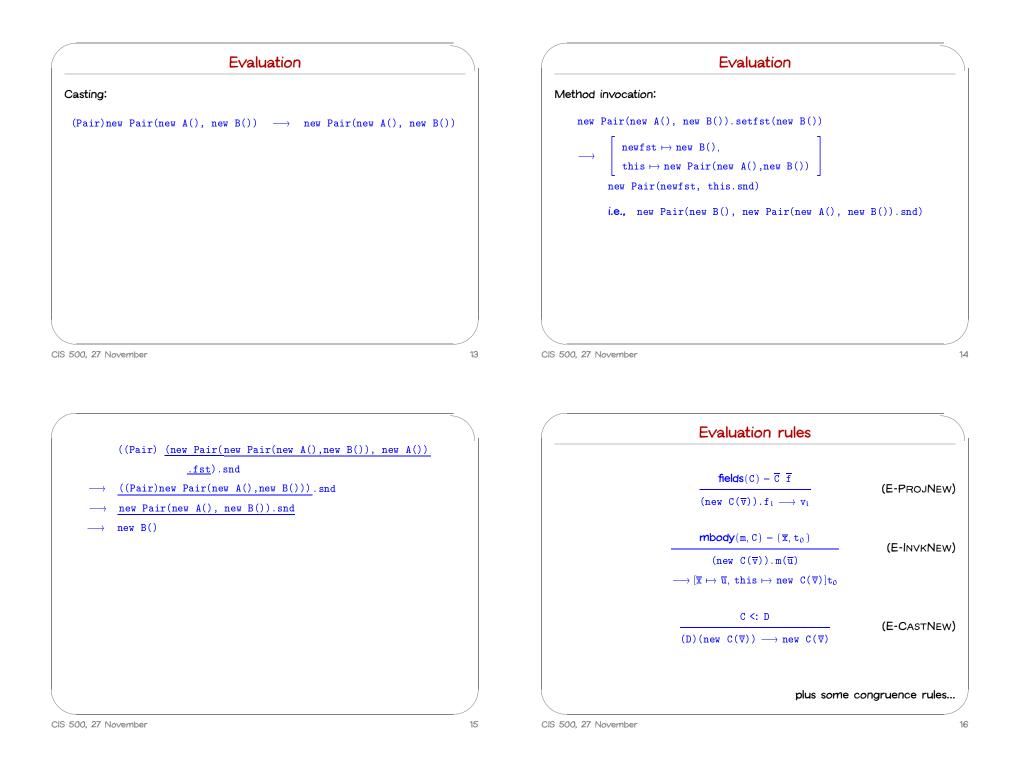
5

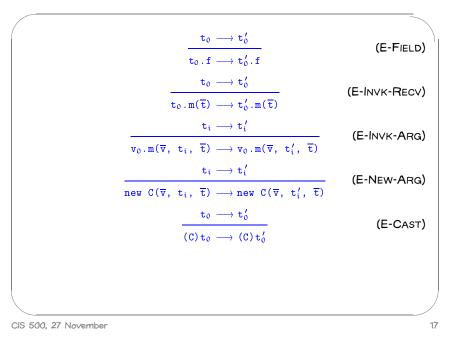


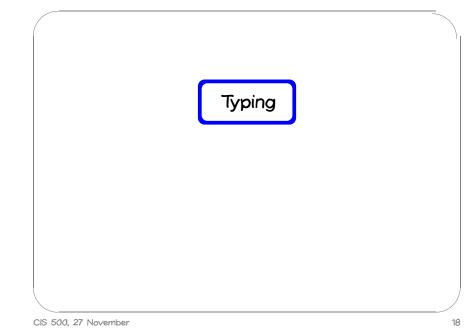
CIS 500, 27 November

11

CIS 500, 27 November



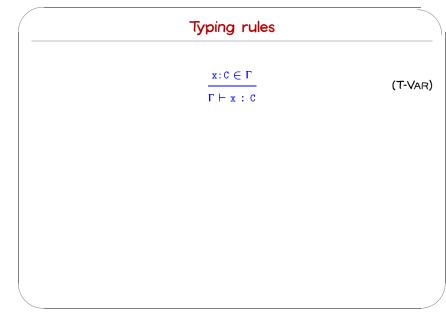


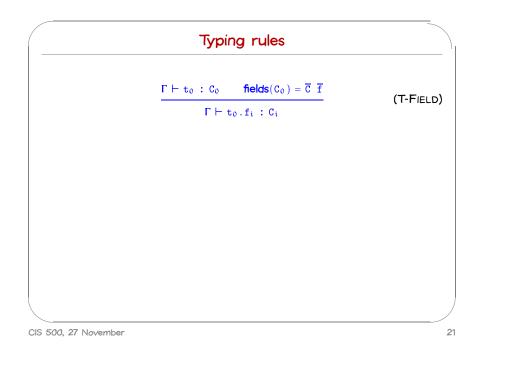


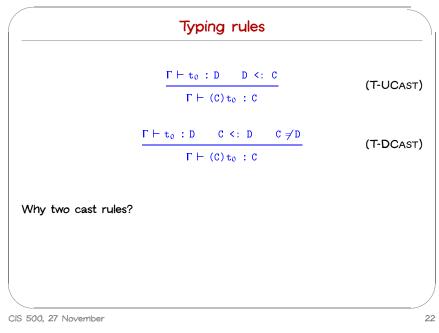


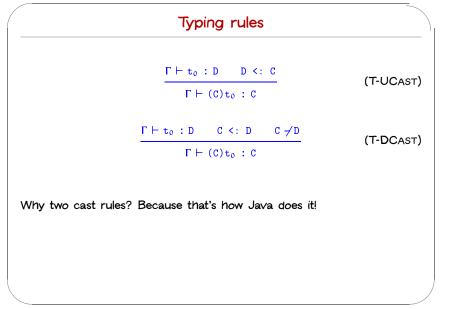
FJ has no rule of subsumption (because we want to follow Java). The typing rules are algorithmic.

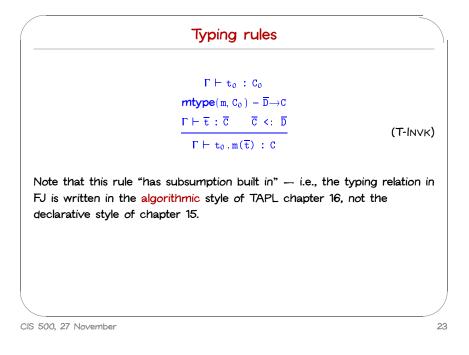
(Where would this make a difference?...)



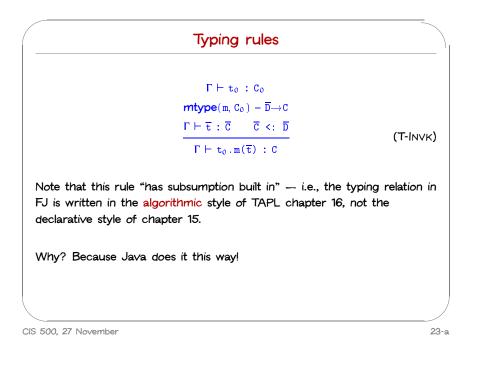








CIS 500, 27 November



$\begin{array}{l} \Gamma \vdash \mathbf{t}_{0} : \mathbf{C}_{0} \\ \text{mtype}(\mathbf{m}, \mathbf{C}_{0}) = \overline{\mathbf{D}} \rightarrow \mathbf{C} \\ \overline{\Gamma \vdash \overline{\mathbf{t}} : \overline{\mathbf{C}} \quad \overline{\mathbf{C}} <: \overline{\mathbf{D}} \\ \overline{\Gamma \vdash \mathbf{t}_{0} . \mathbf{m}(\overline{\mathbf{t}}) : \mathbf{C}} \end{array} \qquad (T-\text{INVK}) \end{array}$ Note that this rule "has subsumption built in" — i.e., the typing relation in FJ is written in the algorithmic style of TAPL chapter 16, not the declarative style of chapter 15. Why? Because Java does it this way! But why does Java do it this way??

Typing rules

Java typing is algorithmic

The Java typing relation is defined in the algorithmic style, for (at least) two reasons:

- 1. In order to perform static overloading resolution, we need to be able to speak of "the type" of an expression
- 2. We would otherwise run into trouble with typing of conditional expressions

Java typing must be algorithmic

We haven't included them in FJ, but full Java has both interfaces and conditional expressions.

The two together actually make the declarative style of typing rules unworkable!

Let's look at the second in more detail...



Java has no joins

But, in full Java (with interfaces), there are types that have no join! E.g.:

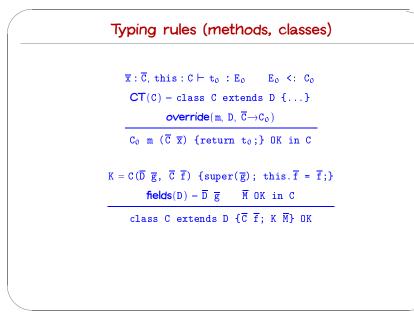
```
interface I {...}
interface J {...}
interface K extends I,J {...}
interface L extends I,J {...}
```

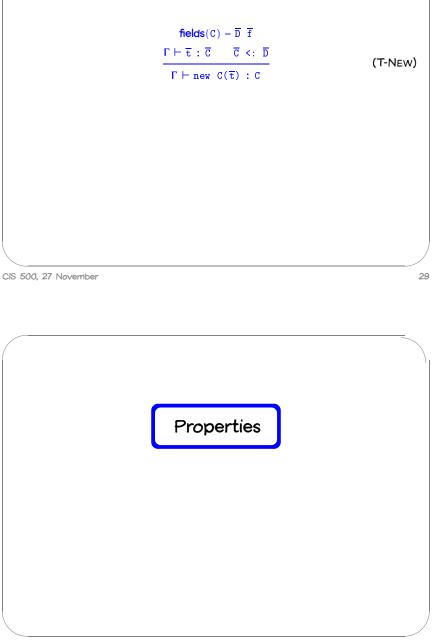
K and L have no join (least upper bound) — both I and J are common upper bounds, but neither of these is less than the other.

So: algorithmic typing rules are really our only option.

CIS 500, 27 November

28





FJ Typing rules

Preservation

Theorem [Preservation]: If $\Gamma \vdash t : C$ and $t \longrightarrow t'$, then $\Gamma \vdash t' : C'$ for some $C' \leq C$.

Proof: Straightforward induction.

Preservation

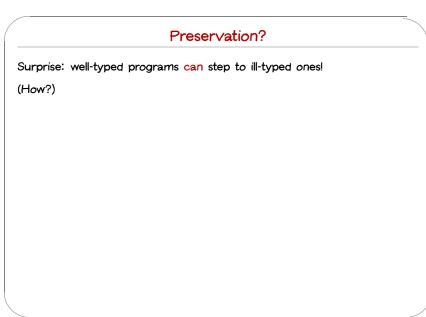
Theorem [Preservation]: If $\Gamma \vdash t : C$ and $t \longrightarrow t'$, then $\Gamma \vdash t' : C'$ for some $C' \leq C$.

Proof: Straightforward induction. ???

CIS 500, 27 November

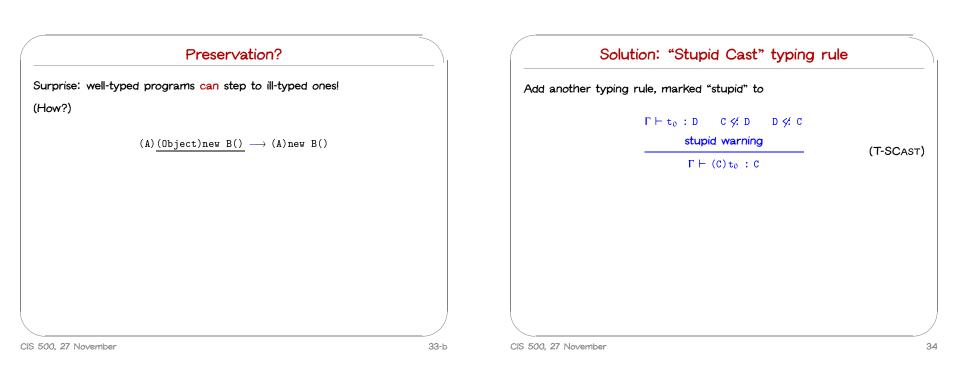
32

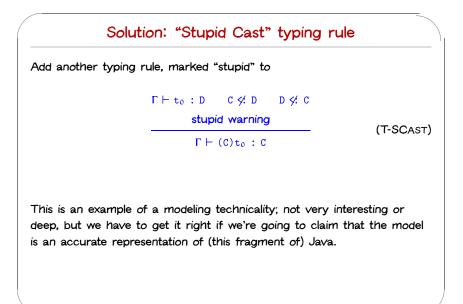
Preservation?

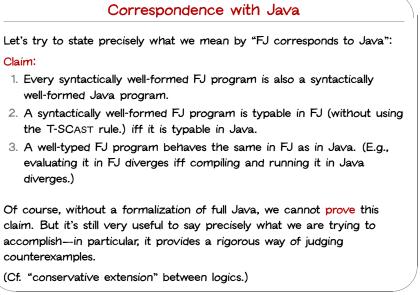


CIS 500, 27 November

32-a

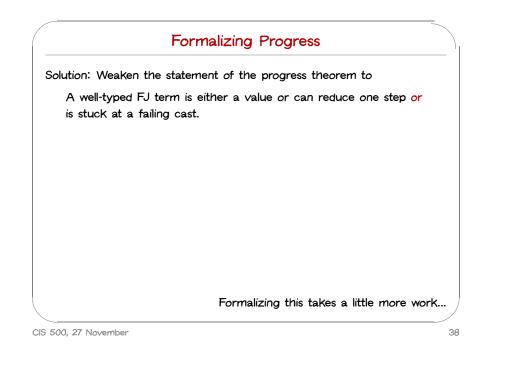








CIS 500, 27 November



E ::=	evaluation contexts
[]	hole
E.f	field access
$E.m(\overline{t})$	method invocation (receive
$v.m(\overline{v}, E, \overline{t})$	method invocation (arg)
new $C(\overline{v}, E, \overline{t})$	object creation (arg)
(C)E	cast
	\rightarrow t', then we can express t and t' as e E, r, and r', with r \rightarrow r' by one of

Progress

Theorem [Progress]: Suppose t is a closed, well-typed normal form. Then either (1) t is a value, or (2) t \longrightarrow t' for some t', or (3) for some evaluation context E, we can express t as t = E[(C) (new D(\overline{v}))], with D \leq C.