CIS 500
Software Foundations
Fall 2002

27 November

Administrivia

- People taking the 380 final (or another) early on the 20th can start the CIS500 exam at noon. Let me know by email if you plan to do this.

FJ syntax (terms and values)

`t` ::=		terms
	`x`	variable
	`t.f`	field access
	$\mathtt{t.m(\overline{t})}$	method invocation
	$\mathtt{new\ C(\overline{t})}$	object creation
	`(C) t`	cast
`v` ::=		values
	$\mathtt{new\ C(\overline{v})}$	object creation

Syntax (methods and classes)

`K` ::=		constructor declarations
	$\mathtt{C(\overline{C}\ \overline{f})\ \{super(\overline{f});\ this.\overline{f}=\overline{f};\}}$	
`M` ::=		method declarations
	$\mathtt{C\ m(\overline{C}\ \overline{x})\ \{return\ t;\}}$	
`CL` ::=		class declarations
	$\mathtt{class\ C\ extends\ C\ \{\overline{C}\ \overline{f};\ K\ \overline{M}\}}$	

Subtyping

Assume we have a (global, fixed) class table CT mapping class names to definitions.

$$\frac{CT(\texttt{C}) = \texttt{class C extends D \{...\}}}{\texttt{C <: D}}$$

$$\texttt{C <: C}$$

$$\frac{\texttt{C <: D} \qquad \texttt{D <: E}}{\texttt{C <: E}}$$

Field lookup

$$\textit{fields}(\texttt{Object}) = \emptyset$$

$$\frac{CT(\texttt{C}) = \texttt{class C extends D \{}\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{; K }\overline{\texttt{M}}\texttt{\}} \qquad \textit{fields}(\texttt{D}) = \overline{\texttt{D}}\ \overline{\texttt{g}}}{\textit{fields}(\texttt{C}) = \overline{\texttt{D}}\ \overline{\texttt{g}}, \overline{\texttt{C}}\ \overline{\texttt{f}}}$$

Method type lookup

$$\frac{CT(\texttt{C}) = \texttt{class C extends D \{}\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{; K }\overline{\texttt{M}}\texttt{\}} \qquad \texttt{B m (}\overline{\texttt{B}}\ \overline{\texttt{x}}\texttt{) \{return t;\}} \in \overline{\texttt{M}}}{\textit{mtype}(\texttt{m}, \texttt{C}) = \overline{\texttt{B}} \rightarrow \texttt{B}}$$

$$\frac{CT(\texttt{C}) = \texttt{class C extends D \{}\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{; K }\overline{\texttt{M}}\texttt{\}} \qquad \texttt{m} \text{ is not defined in } \overline{\texttt{M}}}{\textit{mtype}(\texttt{m}, \texttt{C}) = \textit{mtype}(\texttt{m}, \texttt{D})}$$

Method body lookup

$$\frac{CT(\texttt{C}) = \texttt{class C extends D \{}\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{; K }\overline{\texttt{M}}\texttt{\}} \qquad \texttt{B m (}\overline{\texttt{B}}\ \overline{\texttt{x}}\texttt{) \{return t;\}} \in \overline{\texttt{M}}}{\textit{mbody}(\texttt{m}, \texttt{C}) = (\overline{\texttt{x}}, \texttt{t})}$$

$$\frac{CT(\texttt{C}) = \texttt{class C extends D \{}\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{; K }\overline{\texttt{M}}\texttt{\}} \qquad \texttt{m} \text{ is not defined in } \overline{\texttt{M}}}{\textit{mbody}(\texttt{m}, \texttt{C}) = \textit{mbody}(\texttt{m}, \texttt{D})}$$

Valid method overriding

$$\frac{\mathit{mtype}(\mathtt{m}, \mathtt{D}) = \overline{\mathtt{D}} \to \mathtt{D_0} \text{ implies } \overline{\mathtt{C}} = \overline{\mathtt{D}} \text{ and } \mathtt{C_0} = \mathtt{D_0}}{\mathit{override}(\mathtt{m}, \mathtt{D}, \overline{\mathtt{C}} \to \mathtt{C_0})}$$

Evaluation

The example again

```
class A extends Object { A() { super(); } }

class B extends Object { B() { super(); } }

class Pair extends Object {
  Object fst;
  Object snd;

  Pair(Object fst, Object snd) {
    super(); this.fst=fst; this.snd=snd; }

  Pair setfst(Object newfst) {
    return new Pair(newfst, this.snd); }
}
```


Evaluation

Projection:

$$\mathtt{new\ Pair(new\ A(),\ new\ B()).snd} \longrightarrow \mathtt{new\ B()}$$

Evaluation

Casting:

$\texttt{(Pair)new Pair(new A(), new B())} \longrightarrow \texttt{new Pair(new A(), new B())}$

Evaluation

Method invocation:

```
new Pair(new A(), new B()).setfst(new B())
```

$\longrightarrow \begin{bmatrix} \texttt{newfst} \mapsto \texttt{new B()}, \\ \texttt{this} \mapsto \texttt{new Pair(new A(),new B())} \end{bmatrix}$
$\texttt{new Pair(newfst, this.snd)}$

i.e., `new Pair(new B(), new Pair(new A(), new B()).snd)`

$\texttt{((Pair)}\ \underline{\texttt{(new Pair(new Pair(new A(),new B()), new A())}}$
$\underline{\texttt{.fst}}\texttt{).snd}$

$\longrightarrow \underline{\texttt{((Pair)new Pair(new A(),new B()))}}\texttt{.snd}$

$\longrightarrow \underline{\texttt{new Pair(new A(), new B()).snd}}$

$\longrightarrow \texttt{new B()}$

Evaluation rules

$$\frac{\textit{fields}(\texttt{C}) = \overline{\texttt{C}}\ \overline{\texttt{f}}}{(\texttt{new C}(\overline{\texttt{v}}))\texttt{.f}_i \longrightarrow \texttt{v}_i} \quad \text{(E-ProjNew)}$$

$$\frac{\textit{mbody}(\texttt{m}, \texttt{C}) = (\overline{\texttt{x}}, \texttt{t}_0)}{(\texttt{new C}(\overline{\texttt{v}}))\texttt{.m}(\overline{\texttt{u}}) \longrightarrow [\overline{\texttt{x}} \mapsto \overline{\texttt{u}}, \texttt{this} \mapsto \texttt{new C}(\overline{\texttt{v}})]\texttt{t}_0} \quad \text{(E-InvkNew)}$$

$$\frac{\texttt{C} <: \texttt{D}}{(\texttt{D})(\texttt{new C}(\overline{\texttt{v}})) \longrightarrow \texttt{new C}(\overline{\texttt{v}})} \quad \text{(E-CastNew)}$$

plus some congruence rules...

$$\frac{t_0 \longrightarrow t_0'}{t_0.f \longrightarrow t_0'.f} \quad \text{(E-Field)}$$

$$\frac{t_0 \longrightarrow t_0'}{t_0.m(\overline{t}) \longrightarrow t_0'.m(\overline{t})} \quad \text{(E-Invk-Recv)}$$

$$\frac{t_i \longrightarrow t_i'}{v_0.m(\overline{v},\ t_i,\ \overline{t}) \longrightarrow v_0.m(\overline{v},\ t_i',\ \overline{t})} \quad \text{(E-Invk-Arg)}$$

$$\frac{t_i \longrightarrow t_i'}{\texttt{new}\ C(\overline{v},\ t_i,\ \overline{t}) \longrightarrow \texttt{new}\ C(\overline{v},\ t_i',\ \overline{t})} \quad \text{(E-New-Arg)}$$

$$\frac{t_0 \longrightarrow t_0'}{(C)t_0 \longrightarrow (C)t_0'} \quad \text{(E-Cast)}$$

Typing

Notes

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

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

Typing rules

$$\frac{x{:}C \in \Gamma}{\Gamma \vdash x : C} \quad \text{(T-Var)}$$

Typing rules

$$\frac{\Gamma \vdash \mathtt{t}_0 : \mathtt{C}_0 \qquad \textit{fields}(\mathtt{C}_0) = \overline{\mathtt{C}}\ \overline{\mathtt{f}}}{\Gamma \vdash \mathtt{t}_0.\mathtt{f}_i : \mathtt{C}_i} \qquad \text{(T-Field)}$$

Typing rules

$$\frac{\Gamma \vdash \mathtt{t}_0 : \mathtt{D} \qquad \mathtt{D} <: \mathtt{C}}{\Gamma \vdash (\mathtt{C})\mathtt{t}_0 : \mathtt{C}} \qquad \text{(T-UCast)}$$

$$\frac{\Gamma \vdash \mathtt{t}_0 : \mathtt{D} \qquad \mathtt{C} <: \mathtt{D} \qquad \mathtt{C} \neq \mathtt{D}}{\Gamma \vdash (\mathtt{C})\mathtt{t}_0 : \mathtt{C}} \qquad \text{(T-DCast)}$$

Why two cast rules?

Typing rules

$$\frac{\Gamma \vdash \mathtt{t}_0 : \mathtt{D} \qquad \mathtt{D} <: \mathtt{C}}{\Gamma \vdash (\mathtt{C})\mathtt{t}_0 : \mathtt{C}} \qquad \text{(T-UCast)}$$

$$\frac{\Gamma \vdash \mathtt{t}_0 : \mathtt{D} \qquad \mathtt{C} <: \mathtt{D} \qquad \mathtt{C} \neq \mathtt{D}}{\Gamma \vdash (\mathtt{C})\mathtt{t}_0 : \mathtt{C}} \qquad \text{(T-DCast)}$$

Why two cast rules? Because that's how Java does it!

Typing rules

$$\frac{\begin{array}{c}\Gamma \vdash \mathtt{t}_0 : \mathtt{C}_0 \\ \textit{mtype}(\mathtt{m}, \mathtt{C}_0) = \overline{\mathtt{D}} \rightarrow \mathtt{C} \\ \Gamma \vdash \overline{\mathtt{t}} : \overline{\mathtt{C}} \qquad \overline{\mathtt{C}} <: \overline{\mathtt{D}}\end{array}}{\Gamma \vdash \mathtt{t}_0.\mathtt{m}(\overline{\mathtt{t}}) : \mathtt{C}} \qquad \text{(T-Invk)}$$

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.

Typing rules

$$\frac{\begin{array}{c}\Gamma \vdash \mathtt{t}_0 : \mathtt{C}_0 \\ \textit{mtype}(\mathtt{m}, \mathtt{C}_0) = \overline{\mathtt{D}} \rightarrow \mathtt{C} \\ \Gamma \vdash \overline{\mathtt{t}} : \overline{\mathtt{C}} \qquad \overline{\mathtt{C}} <: \overline{\mathtt{D}}\end{array}}{\Gamma \vdash \mathtt{t}_0.\mathtt{m}(\overline{\mathtt{t}}) : \mathtt{C}} \quad \text{(T-INVK)}$$

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!

Typing rules

$$\frac{\begin{array}{c}\Gamma \vdash \mathtt{t}_0 : \mathtt{C}_0 \\ \textit{mtype}(\mathtt{m}, \mathtt{C}_0) = \overline{\mathtt{D}} \rightarrow \mathtt{C} \\ \Gamma \vdash \overline{\mathtt{t}} : \overline{\mathtt{C}} \qquad \overline{\mathtt{C}} <: \overline{\mathtt{D}}\end{array}}{\Gamma \vdash \mathtt{t}_0.\mathtt{m}(\overline{\mathtt{t}}) : \mathtt{C}} \quad \text{(T-INVK)}$$

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??

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

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

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!

Java conditionals

$$\frac{t_1 \in \texttt{bool} \qquad t_2 \in T_2 \qquad t_3 \in T_3}{t_1 \;?\; t_2 \;:\; t_3 \in \;?}$$

Java conditionals

$$\frac{t_1 \in \texttt{bool} \qquad t_2 \in T_2 \qquad t_3 \in T_3}{t_1 \;?\; t_2 \;:\; t_3 \in \;?}$$

Actual Java rule (algorithmic):

$$\frac{t_1 \in \texttt{bool} \qquad t_2 \in T_2 \qquad t_3 \in T_3}{t_1 \;?\; t_2 \;:\; t_3 \in \min(T_2, T_3)}$$

More standard (declarative) rule:

$$\frac{t_1 \in \texttt{bool} \qquad t_2 \in T \qquad t_3 \in T}{t_1 \;?\; t_2 \;:\; t_3 \in T}$$

More standard (declarative) rule:

$$\frac{t_1 \in \texttt{bool} \qquad t_2 \in T \qquad t_3 \in T}{t_1 \;?\; t_2 \;:\; t_3 \in T}$$

Algorithmic version:

$$\frac{t_1 \in \texttt{bool} \qquad t_2 \in T_2 \qquad t_3 \in T_3}{t_1 \;?\; t_2 \;:\; t_3 \in T_2 \vee T_3}$$

Requires joins!

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.

FJ Typing rules

$$\frac{\textit{fields}(\mathtt{C}) = \overline{\mathtt{D}}\ \overline{\mathtt{f}} \qquad \Gamma \vdash \overline{\mathtt{t}} : \overline{\mathtt{C}} \qquad \overline{\mathtt{C}} \mathrel{\mathtt{<:}} \overline{\mathtt{D}}}{\Gamma \vdash \mathtt{new}\ \mathtt{C}(\overline{\mathtt{t}}) : \mathtt{C}} \qquad \text{(T-New)}$$

Typing rules (methods, classes)

$$\frac{\overline{\mathtt{x}} : \overline{\mathtt{C}}, \mathtt{this} : \mathtt{C} \vdash \mathtt{t}_0 : \mathtt{E}_0 \qquad \mathtt{E}_0 \mathrel{\mathtt{<:}} \mathtt{C}_0 \qquad \textit{CT}(\mathtt{C}) = \mathtt{class\ C\ extends\ D\ \{...\}} \qquad \textit{override}(\mathtt{m}, \mathtt{D}, \overline{\mathtt{C}} \to \mathtt{C}_0)}{\mathtt{C}_0\ \mathtt{m}\ (\overline{\mathtt{C}}\ \overline{\mathtt{x}})\ \{\mathtt{return}\ \mathtt{t}_0;\}\ \mathtt{OK\ in\ C}}$$

$$\frac{\mathtt{K} = \mathtt{C}(\overline{\mathtt{D}}\ \overline{\mathtt{g}},\ \overline{\mathtt{C}}\ \overline{\mathtt{f}})\ \{\mathtt{super}(\overline{\mathtt{g}});\ \mathtt{this}.\overline{\mathtt{f}} = \overline{\mathtt{f}};\} \qquad \textit{fields}(\mathtt{D}) = \overline{\mathtt{D}}\ \overline{\mathtt{g}} \qquad \overline{\mathtt{M}}\ \mathtt{OK\ in\ C}}{\mathtt{class\ C\ extends\ D}\ \{\overline{\mathtt{C}}\ \overline{\mathtt{f}};\ \mathtt{K}\ \overline{\mathtt{M}}\}\ \mathtt{OK}}$$

Properties

Preservation

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

Proof: Straightforward induction.

Preservation

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

Proof: Straightforward induction. ???

Preservation?

Preservation?

Surprise: well-typed programs can step to ill-typed *ones*!

(How?)

Preservation?

Surprise: well-typed programs can step to ill-typed ones!

(How?)

$$\texttt{(A)}\underline{\texttt{(Object)new B()}} \longrightarrow \texttt{(A)new B()}$$

Solution: "Stupid Cast" typing rule

Add another typing rule, marked "stupid" to

$$\frac{\Gamma \vdash \texttt{t}_0 : \texttt{D} \qquad \texttt{C} \not<: \texttt{D} \qquad \texttt{D} \not<: \texttt{C} \qquad \textit{stupid warning}}{\Gamma \vdash \texttt{(C)t}_0 : \texttt{C}} \qquad \text{(T-SCast)}$$

Solution: "Stupid Cast" typing rule

Add another typing rule, marked "stupid" to

$$\frac{\Gamma \vdash \texttt{t}_0 : \texttt{D} \qquad \texttt{C} \not<: \texttt{D} \qquad \texttt{D} \not<: \texttt{C} \qquad \textit{stupid warning}}{\Gamma \vdash \texttt{(C)t}_0 : \texttt{C}} \qquad \text{(T-SCast)}$$

This is an example of a modeling technicality; not very interesting or deep, but we have to get it right if we're going to claim that the model is an accurate representation of (this fragment of) Java.

Correspondence with Java

Let's try to state precisely what we mean by "FJ corresponds to Java":

Claim:

1. Every syntactically well-formed FJ program is also a syntactically well-formed Java program.
2. A syntactically well-formed FJ program is typable in FJ (without using the T-SCast rule.) iff it is typable in Java.
3. A well-typed FJ program behaves the same in FJ as in Java. (E.g., evaluating it in FJ diverges iff compiling and running it in Java diverges.)

Of course, without a formalization of full Java, we cannot prove this claim. But it's still very useful to say precisely what we are trying to accomplish—in particular, it provides a rigorous way of judging counterexamples.

(Cf. "conservative extension" between logics.)

Alternative approaches to casting

- Loosen preservation theorem
- Use big-step semantics

Progress

Progress

Problem: well-typed programs can get stuck.

How?

Progress

Problem: well-typed programs can get stuck.

How?

Cast failure:

```
(A)new Object()
```

Formalizing Progress

Solution: Weaken the statement of the progress theorem to

> A well-typed FJ term is either a value or can reduce one step *or* is stuck at a failing cast.

Formalizing this takes a little more work...

Evaluation Contexts

E	::=	evaluation contexts
	[ ]	hole
	E.f	field access
	E.m($\overline{\mathtt{t}}$)	method invocation (receiver)
	v.m($\overline{\mathtt{v}}$,E,$\overline{\mathtt{t}}$)	method invocation (arg)
	new C($\overline{\mathtt{v}}$,E,$\overline{\mathtt{t}}$)	object creation (arg)
	(C)E	cast

Evaluation contexts capture the notion of the "next subterm to be reduced," in the sense that, if $\mathtt{t} \longrightarrow \mathtt{t}'$, then we can express t and t′ as $\mathtt{t} = \mathtt{E}[\mathtt{r}]$ and $\mathtt{t}' = \mathtt{E}[\mathtt{r}']$ for a unique E, r, and r′, with $\mathtt{r} \longrightarrow \mathtt{r}'$ by one of the computation rules E-ProjNew, E-InvkNew, or E-CastNew.

Progress

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