Topic 1. Types for Featherweight Java. We now turn to the typing discipline for Featherweight Java, which is subsumed by Java’s type system. Our aim is to formally characterize the system using the tools we’ve developed in this course.

Failure semantics, revisited To distinguish FJ expressions that result in type error, we introduce a fail value and failure semantics as for D. However, we will add an elegant twist. Recall that one of the problems with defining a failure semantics is to ensure that fail aborts any “waiting” computations. For example, in D we want:

\[ (\text{Function } x \rightarrow x + 1)([\text{fail} + 5] - 3) \Rightarrow \text{fail} \]

For this reason, we need to introduce a slew of new rules to “bubble up” fail during computation:

- \( \text{PLUSABORTL} \)
  \[
  e_1 \Rightarrow \text{fail} \quad e_1 + e_2 \Rightarrow \text{fail}
  \]

- \( \text{PLUSABORTR} \)
  \[
  e_1 \Rightarrow n \quad e_2 \Rightarrow \text{fail} \quad e_1 + e_2 \Rightarrow \text{fail}
  \]

- \( \text{NOTABORT} \)
  \[
  e \Rightarrow \text{fail} \quad \text{Not } e \Rightarrow \text{fail}
  \]

and so on for the rest of the expression forms. This is very cumbersome and tedious. A simpler way to go about this is to formally characterize the idea of “waiting computations”, or “continuations of computation”, via the definition of evaluation contexts. In D, this would be as follows:

\[
\oplus ::= \text{And} | \text{Or} | + | - | = \quad \text{binary ops}
\]

\[
E ::= [\ ] | v \oplus E | E \oplus e | \text{Not } E | E e | v E \quad \text{D evaluation contexts}
\]

Then, we define a single rule to replace the myriad of ABORT rules:

\[ \text{FAILABORT} \]

\[ E[\text{fail}] \Rightarrow \text{fail} \]

For example:

\[
E \triangleq (\text{Function } x \rightarrow x + 1)(((\text{fail}) + 5) - 3) \quad E[\text{fail}] = (\text{Function } x \rightarrow x + 1)(((\text{fail}) + 5) - 3)
\]

In FJ, we define evaluation contexts as follows:

\[
E ::= [\ ] | E.f | E.m(\vec{e}) | v.m(\vec{e}, \vec{e}') | \text{new } C(\vec{e}, E, \vec{e}') | (C)E \quad \text{FJ evaluation contexts}
\]

And our failure semantics for FJ:

- \( \text{FIELDFAIL} \)
  \[
  e \Rightarrow \text{new } C(\vec{e}) \quad \text{fields}(C) = \vec{C} \quad f_i \not\in \vec{F}
  \]

- \( \text{INVOKEMETHFAIL} \)
  \[
  d \Rightarrow \text{new } C(\vec{e}) \quad mbody(m, C) \text{ undefined}
  \]

- \( \text{INVOKERGSSFAIL} \)
  \[
  d \Rightarrow \text{new } C(\vec{e}) \quad mbody(m, C) = \vec{x}.e \quad |\vec{x}| \neq |\vec{e}|
  \]

- \( \text{FAILABORT} \)
  \[
  E[\text{fail}] \Rightarrow \text{fail}
  \]

1
Method type lookup  Just as we’ve defined a function for looking up method bodies in the class table, we also define a function that will look up method types in a class table. Note that method types are analogous to D and OCaml function types; they specify the types of the domain and range of a given method. Method type lookup supports inheritance, and as we will see, the type system requires that the types of overridden methods must agree with the type of the superclass version, as in Java.

\[
\begin{align*}
\text{class } C &\text{ extends } D \{ \text{\bar{C} \bar{f}; \text{K \bar{R}}} \} \\
\text{B m(\bar{B} \bar{x})\{\text{return } e;\} } &\in \mathcal{R} \\
\frac{\text{mtype}(m, \bar{C}) = \bar{B} \rightarrow \bar{B}}{
\text{class } C &\text{ extends } D \{ \text{\bar{C} \bar{f}; \text{K \bar{R}}} \} \\
\text{m } &\not\in \mathcal{R} \\
\frac{\text{mtype}(m, \bar{C}) = \text{mtype}(m, D)}{
\end{align*}
\]

Expression typing  Note that the T-INVK and T-NEW rules allow arguments to be subtypes of the specified method or constructor type, realizing the flexibility of subtyping. Also, note that only up- or down-casts are allowed by the type system, not stupid casts.

\[
\begin{align*}
\text{T-VAR} &\quad \Gamma \vdash x : \Gamma(x) \\
\text{T-BADCAST} &\quad \Gamma \vdash \text{badcast} : \mathcal{C} \\
\text{T-FIELD} &\quad \Gamma \vdash e_0 : C_0 \\
\quad \text{fields}(C_0) = \bar{C} \bar{f} \\
\frac{\Gamma \vdash e_0.f_i : C_i}{\text{mtype}(m, C_0) = \bar{B} \rightarrow C} \\
\text{T-INVK} &\quad \Gamma \vdash e_0 : C_0 \\
\quad \text{mtype}(m, C_0) = \bar{B} \rightarrow C \\
\frac{\Gamma \vdash e_0.m(\bar{g}) : \mathcal{C}}{
\text{T-NEW} &\quad \text{fields}(C) = \bar{D} \bar{f} \\
\quad \Gamma \vdash \bar{a} : \bar{C} \\
\quad \bar{C} < : \bar{D} \\
\frac{\Gamma \vdash \text{new } C(\bar{g}) : C}{\text{T-UCAST} &\quad \Gamma \vdash e_0 : D \\
\quad D < : C \\
\frac{\Gamma \vdash e_0 : C}{\text{T-DCAST} &\quad \Gamma \vdash e_0 : D \\
\quad C < : D \\
\quad C \neq D}}
\end{align*}
\]

Method and class typing  Here are the rules for typing methods and classes. Note how the T-METH rule requires that all versions of the same method in a class hierarchy must agree in type.

\[
\begin{align*}
\text{T-METH} &\quad \Gamma \vdash \text{this} : C \rightarrow \bar{C} \\
\quad \text{this} : C \vdash e_0 : E_0 \\
\quad E_0 < : C_0 \\
\frac{\text{if } \text{mtype}(m, D) = \bar{B} \rightarrow D_0, \text{then } \bar{C} = \bar{D} \text{ and } C_0 = D_0}{
\text{class } C &\text{ extends } D \{ \text{\bar{C} \bar{f}; \text{K \bar{R}}} \} \{\text{return } e_0;\} \text{ OK IN C}}
\end{align*}
\]

\[
\begin{align*}
\text{T-CLASS} &\quad K = C(\bar{D} \bar{g} ; \bar{C} \bar{f})\{\text{super}(\bar{g}); \text{this.} \bar{f} = \bar{f};\} \\
\text{fields}(D) = \bar{D} \bar{g} \\
\frac{\text{fields}(D) = \bar{D} \bar{g}}{
\text{class } C &\text{ extends } D \{ \text{\bar{C} \bar{f}; \text{K \bar{R}}} \} \text{ OK}}
\end{align*}
\]

Properties  Given this formal development, we are able to obtain type preservation and safety results for FJ. Note that these results do allow for the possibility of a badcast to be raised.

\begin{enumerate}
\item[Definition 1.1] A class table \(CT\) is well-typed iff \((CT(C) \text{ OK})\) is derivable for all \(C \in \text{domain}(CT)\).
\item[Theorem 1.1 (FJ Type Preservation)] Assume given well-typed \(CT\). Then if \(\Gamma \vdash e : \mathcal{T}\) and \(e \Rightarrow v\), then \(\Gamma \vdash v : \mathcal{T}\).
\item[Theorem 1.2 (FJ Type Safety)] Assume given well-typed \(CT\). Then if \(\Gamma \vdash e : \mathcal{T}\), it is not the case that \(e \Rightarrow \text{fail}\).
\end{enumerate}