1 Introduction

The syntax and subtyping relation introduced here is based on Featherweight Java (Igarashi, Pierce, Wadler ’1999).

2 Extended Syntax

\[ A ::= \text{Thread Checker annotations} \]
\[ \text{@NotRunBy}(x) \]
\[ \text{@OnlyRunBy}(x) \]
\[ CL ::= \text{class declarations} \]
\[ \text{class } \overline{A} C \text{ extends } C \{ \overline{C f}; K \overline{M} \} \]
\[ K ::= \text{constructor declarations} \]
\[ \overline{A} C(\overline{C f}) \{ \text{super}(\overline{f}); \text{this.} \overline{f} = \overline{f}; \} \]
\[ M ::= \text{method declarations} \]
\[ \overline{A} C m(\overline{C x}) \{ \text{return } t; \} \]

The rest of the syntax is identical to that of Featherweight Java.

3 Featherweight Java Subtyping Relation

\[
\begin{align*}
C & <; C \\
C & <; D & D & <; E \\
\overline{C T(C)} & = \overline{A \text{ class } C \text{ extends } D\{...\}} \\
\overline{C} & <; \overline{D}
\end{align*}
\]
4 Auxiliary Definitions

4.1 Extraction

Extract the method declarations from a class:

\[
CT(C) = \overrightarrow{\text{class } C \text{ extends } D \{C_f, K M\}}
\]

\[
\text{extract}_M(C) = M
\]

Extract the annotations from a class:

\[
CT(C) = \overrightarrow{\text{class } C \text{ extends } D \{C_f, K M\}}
\]

\[
\text{extract}_A(C) = \overrightarrow{A}
\]

Extract the annotations from a method:

\[
\overrightarrow{A R m(P x)} \in \text{extract}_M(C)
\]

\[
\text{extract}_A(m, C) = A
\]

Extract the name from a method declaration:

\[
M = \overrightarrow{A R m(P x)}
\]

\[
\text{name}_M(M) = m
\]

Extract the annotations from a method:

\[
\overrightarrow{A R m(P x)} \in \text{extract}_M(C)
\]

\[
\text{extract}_A(m, C) = A
\]

Get only the \texttt{@NotRunBy}(x) or \texttt{@OnlyRunBy}(x) annotations of an annotations list, respectively:

\[
\text{notRunBy}(\cdot) = \cdot
\]

\[
\text{notRunBy}(\texttt{@NotRunBy}(x), \overrightarrow{A}) = \texttt{@NotRunBy}(x), \text{notRunBy}(\overrightarrow{A})
\]

\[
\text{notRunBy}(\texttt{@OnlyRunBy}(x), \overrightarrow{A}) = \text{notRunBy}(\overrightarrow{A})
\]

\[
\text{onlyRunBy}(\cdot) = \cdot
\]

\[
\text{onlyRunBy}(\texttt{@OnlyRunBy}(x), \overrightarrow{A}) = \texttt{@OnlyRunBy}(x), \text{onlyRunBy}(\overrightarrow{A})
\]

\[
\text{onlyRunBy}(\texttt{@NotRunBy}(x), \overrightarrow{A}) = \text{onlyRunBy}(\overrightarrow{A})
\]
4.2 Method Introduction

Find the class $D$, such that $C <: D$, that introduces a method $m$, i.e. the superclass of $C$ containing a declaration of $m$ that does not have any superclasses that also contains a declaration of $m$. If the method $m$ is not defined in $C$ or any of its superclasses, $\cdot$ is returned.

\[
\text{introduced}(m, \text{Object}) = \cdot
\]

\[
CT(C) = \overline{A} \text{ class } C \text{ extends } D \{...\}
\]

\[
\begin{align*}
\text{m defined in } \text{extract}_M(C) \\
\text{introduced}(m, D) = \cdot \\
\text{introduced}(m, C) = C
\end{align*}
\]

\[
CT(C) = \overline{A} \text{ class } C \text{ extends } D \{...\}
\]

\[
\begin{align*}
\text{introduced}(m, D) = S \\
S \neq \cdot \\
\text{introduced}(m, C) = S
\end{align*}
\]

4.3 Annotation Lookup

Methods of class $\text{Object}$ do not have any annotations.

\[
\text{annot}_M(m, \text{Object}) = \cdot
\]

The annotations of a method $m$ that is introduced in class $C$ are the union of the class annotations on $C$ and the annotations mentioned in the method declaration of $m$ in $C$.

\[
\begin{align*}
\text{introduced}(m, C) = C \\
\text{extract}_A(C) = \overline{A}_C \\
\text{extract}_A(m, C) = \overline{A}_M \\
\text{annot}_M(m, C) = \overline{A}_C \cup \overline{A}_M
\end{align*}
\]

The annotations of a method $m$ that is overridden in class $C$, i.e. method $m$ was introduced in a superclass of $C$, are the union of the class annotations on $C$, the annotations mentioned in the overriding method declaration of $m$ in $C$, and the annotations of the same method $m$ in the superclass of $C$.

\[
\begin{align*}
\overline{\text{introduced}}(m, C) \\
CT(C) = \overline{A}_C \text{ class } C \text{ extends } D \{...\} \\
\text{extract}_A(m, C) = \overline{A}_M \\
\text{annot}_M(m, D) = \overline{A}_D \\
\text{annot}_M(m, C) = \overline{A}_C \cup \overline{A}_M \cup \overline{A}_D
\end{align*}
\]
Check that a set of annotations does not contain contradictions.

\[
\begin{align*}
\text{A OK} & \quad \text{A OK} \\
\text{OK} & \quad \text{OK} \\
\text{OnlyRunBy}(x) \notin A & \quad \text{NotRunBy}(x) \notin A \\
\text{NotRunBy}(x), \ A \text{ OK} & \quad \text{OnlyRunBy}(x), \ A \text{ OK}
\end{align*}
\]

5 Thread Checker Subtyping Relation

Let \( <@ \) be the Thread Checker subtyping relation to distinguish it from the Featherweight Java subtyping relation \( <:: \).

<table>
<thead>
<tr>
<th>Reflexive property</th>
<th>Transitive property</th>
</tr>
</thead>
<tbody>
<tr>
<td>( @C \leq @C )</td>
<td>( @C \leq@ E \rightarrow @D \leq@ E )</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
C <@ D & \quad \text{extract}_M(D) = M \\
\text{name}_M(M) = m & \quad \text{notRunBy(extract}_A(m, C)) = A_C \\
\text{notRunBy(extract}_A(m, D)) = A_D & \quad \text{for each } (A_C, A_D) \in (\overline{A_C}, \overline{A_D}) : A_C \subseteq A_D \\
\end{align*}
\]

For \( C <@ D \), it is required that \( C <:: D \) and that the sets of @NotRunBy annotations of all of \( C \)'s methods are subsets of or equal to the corresponding sets of \( D \)'s methods. If this were not the case, if a method \( m \) in \( C \) contained a @NotRunBy('foo') annotation, but the same method \( m \) in \( D \) did not, then the set of threads allowed to invoke \( m \) has been narrowed. Code working only with \( D \) at an abstract level may expect that \( m \) can be invoked on thread “foo”, because the contract for \( m \) in \( D \) did not state otherwise. This situation is similar to the covariant nature of function arguments: The set of acceptable threads may be made smaller, but not bigger.

6 Extension of Featherweight Java Typing

To check whether a program is well-typed both using the original \( <:: \) subtyping relation and using the \( <@ \) subtyping relation defined above, the \( <@ \) relation should be used instead of \( <:: \): as \( <@ \) is more stringent and completely subsumes \( <:: \) as one if its antecedents.

Furthermore, the \( [ \text{M OK in C} ] \) and \( [ C \text{ OK} ] \) rules should be extended to also perform a \( A \text{ OK} \) check.
7 Implementation Differences

In the current implementation, a few things differ from the syntax and typing described above. Full Java allows overloaded types, so distinguishing methods just by name is not sufficient; instead, the full signature is used: $Cm(\overline{P \pi})$.

The class Object is treated specially in Featherweight Java and is assumed to contain no fields or methods; in the same spirit, we decided to disallow annotations on the class Object or its methods, even though our implementation allows it.

The syntax of annotation is also slightly different: As Java does not allow multiple annotations of the same type on the same class or method, @NotRunBy and OnlyRunBy actually contain arrays of annotations, which specify the thread. We felt it was unnecessary to reproduce this syntactic complexity here.

Finally, while we here identify threads only by a name, our implementation allows the specification by name, group name, ID number, or being an event thread.