Thread Checker Subtyping
Mathias Ricken and Corky Cartwright
mgricken|cork@cs.rice.edu

1 Introduction

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

2 Extended Syntax

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

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

3 Featherweight Java Subtyping Relation

$$C <: C$$

$$\frac{C <: D \quad D <: E}{C <: E}$$

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

4 Auxiliary Definitions

4.1 Extraction

Extract the method declarations from a class:

$$\frac{CT(C) = \overline{A}\ \texttt{class}\ C\ \texttt{extends}\ D\ \{\overline{C_f\ f};\ K\ \overline{M}\}}{extract_M(C) = \overline{M}}$$

Extract the annotations from a class:

$$\frac{CT(C) = \overline{A}\ \texttt{class}\ C\ \texttt{extends}\ D\ \{\overline{C_f\ f};\ K\ \overline{M}\}}{extract_A(C) = \overline{A}}$$

Extract the annotations from a method:

$$\frac{\overline{A}\ R\ m(\overline{P\ x}) \in extract_M(C)}{extract_A(m, C) = \overline{A}}$$

Extract the name from a method declaration:

$$\frac{M = \overline{A}\ R\ m(\overline{P\ x})}{name_M(M) = m}$$

Extract the annotations from a method:

$$\frac{\overline{A}\ R\ m(\overline{P\ x}) \in extract_M(C)}{extract_A(m, C) = \overline{A}}$$

Get only the `@NotRunBy(`x`)` or `@OnlyRunBy(`x`)` annotations of an annotations list, respectively:

$$\begin{array}{lcl}
notRunBy(\cdot) & = & \cdot \\
notRunBy(@\texttt{NotRunBy}(x), \overline{A}) & = & @\texttt{NotRunBy}(x), notRunBy(\overline{A}) \\
notRunBy(@\texttt{OnlyRunBy}(x), \overline{A}) & = & notRunBy(\overline{A})
\end{array}$$

$$\begin{array}{lcl}
onlyRunBy(\cdot) & = & \cdot \\
onlyRunBy(@\texttt{OnlyRunBy}(x), \overline{A}) & = & @\texttt{OnlyRunBy}(x), onlyRunBy(\overline{A}) \\
onlyRunBy(@\texttt{NotRunBy}(x), \overline{A}) & = & onlyRunBy(\overline{A})
\end{array}$$

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.

$$introduced(m, \texttt{Object}) = \cdot$$

$$\frac{\begin{array}{c} CT(C) = \overline{A}\ \texttt{class}\ C\ \texttt{extends}\ D\ \{...\} \\ m \text{ defined in } extract_M(C) \\ introduced(m, D) = \cdot \end{array}}{introduced(m, C) = C}$$

$$\frac{\begin{array}{c} CT(C) = \overline{A}\ \texttt{class}\ C\ \texttt{extends}\ D\ \{...\} \\ introduced(m, D) = S \\ S \neq \cdot \end{array}}{introduced(m, C) = S}$$

4.3 Annotation Lookup

Methods of class `Object` do not have any annotations.

$$annot_M(m, \texttt{Object}) = \cdot$$

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.

$$\frac{\begin{array}{c} introduced(m, C) = C \\ extract_A(C) = \overline{A_C} \\ extract_A(m, C) = \overline{A_M} \end{array}}{annot_M(m, C) = \overline{A_C}\ \cup\ \overline{A_M}}$$

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.

$$\frac{\begin{array}{c} \neg introduced(m, C) \\ CT(C) = \overline{A_C}\ \texttt{class}\ C\ \texttt{extends}\ D\ \{...\} \\ extract_A(m, C) = \overline{A_M} \\ annot_M(m, D) = \overline{A_D} \end{array}}{annot_M(m, C) = \overline{A_C}\ \cup\ \overline{A_M}\ \cup\ \overline{A_D}}$$

Check that a set of annotations does not contain contradictions.

$$\boxed{\overline{A}\ \texttt{OK}}$$

$$\frac{}{\cdot\ \texttt{OK}} \qquad \frac{\overline{A}\ \texttt{OK} \quad \texttt{@OnlyRunBy}(x) \notin \overline{A}}{\texttt{@NotRunBy}(x),\ \overline{A}\ \texttt{OK}} \qquad \frac{\overline{A}\ \texttt{OK} \quad \texttt{@NotRunBy}(x) \notin \overline{A}}{\texttt{@OnlyRunBy}(x),\ \overline{A}\ \texttt{OK}}$$

5 Thread Checker Subtyping Relation

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

Reflexive property

$$C <@ C$$

Transitive property

$$\frac{C <@ D \qquad D <@ E}{C <@ E}$$

$$\frac{\begin{array}{c} C <: D \\ extract_M(D) = \overline{M} \\ name_M(\overline{M}) = \overline{m} \\ notRunBy(extract_A(\overline{m}, C)) = \overline{\overline{A_C}} \\ notRunBy(extract_A(\overline{m}, D)) = \overline{\overline{A_D}} \\ \text{for each}(\overline{A_C}, \overline{A_D}) \in (\overline{\overline{A_C}}, \overline{\overline{A_D}}) : \overline{A_C} \subseteq \overline{A_D} \end{array}}{C <@ D}$$

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 $\boxed{M\ \texttt{OK in}\ C}$ and $\boxed{C\ \texttt{OK}}$ rules should be extended to also perform a $\overline{A}\ \texttt{OK}$ 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 n is not sufficient; instead, the full signature is used: $Cm(\overline{P\ x})$.
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 allos 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.