University of Bordeaux
Year 2015-2016

Master ESSV
MtoCPL

MtoCPL Assignment
Due date : November the 17th, 2015

In this assignment, we propose to add new constructions to FeatherWeight Java, a subset of the Java programming language designed [1] for the simplicity of its type system. Most of the constructions coming from Java have been removed from FeatherWeight Java, most notably the notion of assignment. That simplicity motivated the choice of this language as a base for writing extensions to the Java language.

First, all FeatherWeight Java code is also valid Java code. Here is a snippet of code taken from the original article [1] (the comments highlight the most notable differences from classical Java) :

```
class A extends Object {      // Explicit mention of the superclass
    A() { super(); }          // Explicit declaration of constructor + super call
}
class B extends Object {
    B() { super(); }
}
class Pair extends Object {
    Object fst;
    Object snd;
    Pair(Object fst, Object snd) { // Init of attributes inside the constructor
        super(); this.fst=fst; this.snd=snd;
    }
    Pair setfst(Object newfst) {   // Method, that can only return an expression
        return new Pair(newfst, this.snd);
    }
}
```

The grammar of the language is given on Fig. 1, and the type checking rules appear on Fig 1 and 2. Moreover, they are explained at length in [1] and in chapter 19 of [2].

In 2009, Allwood [3] proposed a technique to construct a suite of test programs that covered a sufficiently large subset of the language features, this in order to test the behavior of the different Java compilers. He programmed a FeatherWeight Java static analyzer in order to compare these different behaviors.

In this assignment, you are given a code derived from his analyzer, that can parse and type-check an expression in the FeatherWeight Java language. All the questions are related to the reading and modification of this code.

1. Write a set of FeatherWeight Java programs that allow a coverage of at least 95% of the files in the `checker/typecheck` directory. These tests should serve as non-

regression tests for the following questions. The coverage will be computed using the Eclemma plugin of Eclipse.

2. Propose a set of grammar rules so as to add to FeatherWeight Java the possibility for sequence of instructions and assignments in the method bodies, and a set of typing rules in order for these new constructs to type-check. We limit ourselves to assigning only to the attributes of a class, and only *within the body of a method*. As an exemple, the following example should type-check :

```
class Void extends Object {
    Void() { super(); }
}
class A extends Object {
    A a;
    A(A a) { super(); this.a = a; }                // Init , not an assignment
    Void setA(A a) {
      this.a = a;                                   // This is an assignment
      return new Void();                            // followed by a return
    }
    A get() { return this.a; }
}
```

Structure of the code

The code given as a source for this assignment contains the following directories :

- `checker` : the main Java package containing :
  - `analysis`, `lexer` and `parser` : utility classes for parsing the code ;
  - `model` : classes modelizing the AST of the code ;
  - `node` : classes auto-generated by SableCC
  - `passes` : code building the model from the parser ;
  - `typecheck` : code handling the type-checking ;
  - `util` : utility classes ;
- `examples` : a list of FeatherWeight Java program examples ;
- `grammar` : a SableCC grammar for the language ;
- `patches` : patches in order to compile without warnings.

In addition, it contains a `Makefile` and a script `fjrun.sh` to run the examples, as well as a `Main.java` that type-checks a single file and a `Test.java` that launches a series of tests.

Syntax:

$$\texttt{L} ::= \texttt{class C extends C \{}\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{; K }\overline{\texttt{M}}\texttt{\}}$$

$$\texttt{K} ::= \texttt{C(}\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{)\{super(}\overline{\texttt{f}}\texttt{); this.}\overline{\texttt{f}}\texttt{=}\overline{\texttt{f}}\texttt{;\}}$$

$$\texttt{M} ::= \texttt{C m(}\overline{\texttt{C}}\ \overline{\texttt{x}}\texttt{)\{ return e; \}}$$

$$\texttt{e} ::= \texttt{x} \mid \texttt{e.f} \mid \texttt{e.m(}\overline{\texttt{e}}\texttt{)} \mid \texttt{new C(}\overline{\texttt{e}}\texttt{)} \mid \texttt{(C)e}$$

Subtyping:

$$\texttt{C <: C} \qquad \frac{\texttt{C <: D} \quad \texttt{D <: E}}{\texttt{C <: E}} \qquad \frac{\texttt{class C extends D \{...\}}}{\texttt{C <: D}}$$

Field lookup:

$$\mathit{fields}(\texttt{Object}) = \bullet$$

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

Method type lookup:

$$\frac{\texttt{class C extends D \{}\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{; K }\overline{\texttt{M}}\texttt{\}} \quad \texttt{B m(}\overline{\texttt{B}}\ \overline{\texttt{x}}\texttt{)\{ return e; \}} \in \overline{\texttt{M}}}{\mathit{mtype}(\texttt{m}, \texttt{C}) = \overline{\texttt{B}} \rightarrow \texttt{B}}$$

$$\frac{\texttt{class C extends D \{}\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{; K }\overline{\texttt{M}}\texttt{\}} \quad \texttt{m} \notin \overline{\texttt{M}}}{\mathit{mtype}(\texttt{m}, \texttt{C}) = \mathit{mtype}(\texttt{m}, \texttt{D})}$$

Method body lookup:

$$\frac{\texttt{class C extends D \{}\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{; K }\overline{\texttt{M}}\texttt{\}} \quad \texttt{B m(}\overline{\texttt{B}}\ \overline{\texttt{x}}\texttt{)\{ return e; \}} \in \overline{\texttt{M}}}{\mathit{mbody}(\texttt{m}, \texttt{C}) = \overline{\texttt{x}}\texttt{.e}}$$

$$\frac{\texttt{class C extends D \{}\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{; K }\overline{\texttt{M}}\texttt{\}} \quad \texttt{m} \notin \overline{\texttt{M}}}{\mathit{mbody}(\texttt{m}, \texttt{C}) = \mathit{mbody}(\texttt{m}, \texttt{D})}$$

FIGURE 1 – Grammar rules for FeatherWeight Java and definition of the fields, mtype and mbody functions.

Expression typing:

$$\Gamma \vdash \mathtt{x} : \Gamma(\mathtt{x}) \qquad \text{(T-VAR)}$$

$$\frac{\Gamma \vdash \mathtt{e}_0 : \mathtt{C}_0 \qquad \mathit{fields}(\mathtt{C}_0) = \overline{\mathtt{C}}\ \overline{\mathtt{f}}}{\Gamma \vdash \mathtt{e}_0.\mathtt{f}_i : \mathtt{C}_i} \qquad \text{(T-FIELD)}$$

$$\frac{\Gamma \vdash \mathtt{e}_0 : \mathtt{C}_0 \qquad \mathit{mtype}(\mathtt{m}, \mathtt{C}_0) = \overline{\mathtt{D}} \to \mathtt{C} \qquad \Gamma \vdash \overline{\mathtt{e}} : \overline{\mathtt{C}} \qquad \overline{\mathtt{C}} <: \overline{\mathtt{D}}}{\Gamma \vdash \mathtt{e}_0.\mathtt{m}(\overline{\mathtt{e}}) : \mathtt{C}} \qquad \text{(T-INVK)}$$

$$\frac{\mathit{fields}(\mathtt{C}) = \overline{\mathtt{D}}\ \overline{\mathtt{f}} \qquad \Gamma \vdash \overline{\mathtt{e}} : \overline{\mathtt{C}} \qquad \overline{\mathtt{C}} <: \overline{\mathtt{D}}}{\Gamma \vdash \mathtt{new}\ \mathtt{C}(\overline{\mathtt{e}}) : \mathtt{C}} \qquad \text{(T-NEW)}$$

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

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

$$\frac{\Gamma \vdash \mathtt{e}_0 : \mathtt{D} \qquad \mathtt{C} \not<: \mathtt{D} \qquad \mathtt{D} \not<: \mathtt{C} \qquad \mathit{stupid\ warning}}{\Gamma \vdash (\mathtt{C})\mathtt{e}_0 : \mathtt{C}} \qquad \text{(T-SCAST)}$$

Method typing:

$$\frac{\begin{array}{c}\overline{\mathtt{x}} : \overline{\mathtt{C}}, \mathtt{this} : \mathtt{C} \vdash \mathtt{e}_0 : \mathtt{E}_0 \qquad \mathtt{E}_0 <: \mathtt{C}_0 \\ \mathtt{class\ C\ extends\ D\ \{...\}} \\ \text{if } \mathit{mtype}(\mathtt{m}, \mathtt{D}) = \overline{\mathtt{D}} \to \mathtt{D}_0, \text{ then } \overline{\mathtt{C}} = \overline{\mathtt{D}} \text{ and } \mathtt{C}_0 = \mathtt{D}_0\end{array}}{\mathtt{C}_0\ \mathtt{m}(\overline{\mathtt{C}}\ \overline{\mathtt{x}})\{\ \mathtt{return}\ \mathtt{e}_0;\ \}\ \mathtt{OK\ IN\ C}} \qquad \text{(T-METHOD)}$$

Class typing:

$$\frac{\mathtt{K} = \mathtt{C}(\overline{\mathtt{D}}\ \overline{\mathtt{g}},\ \overline{\mathtt{C}}\ \overline{\mathtt{f}})\{\mathtt{super}(\overline{\mathtt{g}});\ \mathtt{this}.\overline{\mathtt{f}}=\overline{\mathtt{f}};\} \qquad \mathit{fields}(\mathtt{D}) = \overline{\mathtt{D}}\ \overline{\mathtt{g}} \qquad \overline{\mathtt{M}}\ \mathtt{OK\ IN\ C}}{\mathtt{class\ C\ extends\ D}\ \{\overline{\mathtt{C}}\ \overline{\mathtt{f}};\ \mathtt{K}\ \overline{\mathtt{M}}\}\ \mathtt{OK}} \qquad \text{(T-CLASS)}$$

FIGURE 2 – Typing rules for FeatherWeight Java.

Références

[1] A. Igarashi, B.C. Pierce and P. Wadler, *Featherweight Java : A Minimal Core Calculus for Java and GJ.* ACM Transactions on Programming Languages ans Systems (TOPLAS), Pages 396-450, May 2001.
Available at `http://www.cis.upenn.edu/~bcpierce/papers/fj-toplas.pdf`

[2] B. C. Pierce, *Types and programming languages.* MIT Press, 2002.

[3] T.O.R. Allwood, S. Eisenbach, *Tickling Java with a Feather.* Electronic Notes in Theoretical Computer Science Volume 238, Pages 3-16, October 2009.
Available at `http://pubs.doc.ic.ac.uk/testing-Java-with-fj/`

Modalities of submission : This assignment is due for the November the 17th, 2015. It must be sent by email to `renault@labri.fr`, with a subject containing the identifier `[Submission MtoCPL]` and the names of the persons submitting the work.

The email is supposed to contain a `tar.gz` file (containing the full code with a `Makefile` to compile it) and a `.pdf` file for the answers to the questions. This assignment may be done by teams of at most 2 people.