INSTITUT FÜR INFORMATIK

Establishing static scope name binding and direct superclassing in the external language of the object oriented Java with inner classes is a difficult and subtle task

Hans Langmaack, Andrzej Salwicki
Marek Warpechowski

Bericht Nr. 1111
Dezember 2011
ISSN 2192-6247

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

ZU KIEL

Institut für Informatik der
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40
D – 24098 Kiel

Establishing static scope name binding and direct superclassing in the external language of the object oriented Java with inner classes is a difficult and subtle task

Hans Langmaack, Andrzej Salwicki
Marek Warpechowski

Bericht Nr. 1111
Dezember 2011
ISSN 2192-6247

e-mail: hl@informatik.uni-kiel.de, salwicki@mimuw.edu.pl,
warp@mimuw.edu.pl

Technischer Bericht 1111(2011) 3–46
Christian-Albrechts-Universität zu Kiel, Institut für Informatik

Establishing static scope name binding and direct superclassing in the external language of the object oriented Java with inner classes is a difficult and subtle task

Hans Langmaack
Institut für Informatik,
Christian-Albrechts-Universität zu Kiel
Christian-Albrechts-Platz 4, D-24098 Kiel, Germany
hl@informatik.uni-kiel.de

Andrzej Salwicki
Faculty of Mathematics and Informatics, University
Cardinal Stefan Wyszyński, Warsaw
Wóycickiego 1-3, 01-938 Warszawa, Poland
salwicki@mimuw.edu.pl

Marek Warpechowski
Institute of Informatics, Warsaw University
Banacha 2, 02-092 Warszawa, Poland
warp@mimuw.edu.pl

Abstract. In [IP02] an axiomatic approach towards the semantics of FJI, Featherweight Java with Inner classes, essentially a subset of the Java-programming language, is presented. In this way the authors contribute to an ambitious project: to give an axiomatic definition of the semantics of programming language Java. A similar project with a partly axiomatic flavour, with so called Abstract State Machines ASM, was initiated by E. Boerger and his colleagues[Boe01] in 2001, but did not yet include inner classes. At a first glance the approach of reducing Java's semantics to that of FJI seems promising. We are going to show that several questions have been left unanswered. It turns out that the theory how to elaborate or bind types and thus to determine direct superclasses as proposed in [IP02] has different models. Therefore the suggestion that the formal system of [IP02] defines the (exactly one) semantics of Java is not justified. We present our contribution to the project showing that it must be attacked from another starting point. Quite frequently one encounters a set of inference rules and a claim that a semantics is defined by the rules. Such a claim should be proved. One should present arguments: 1^0 that the system has a model and hence it is a consistent system, and 2^0 that all models are isomorphic. Sometimes such a proposed system contains a rule with a premise which reads: there is no proof of something. One should notice that this is a metatheoretic property. It seems strange to accept a metatheorem as a premise, especially if such a system does not offer any other inference rules which would enable a proof of the premise. We are going to study the system in [IP02]. We shall show that it has many non-isomorphic models. We present a repair of Igarashi's and Pierce's calculus such that their ideas are preserved as close as possible.

Key words: object oriented programming, semantics, inheritance, inner classes, direct superclass, static semantics analysis, static binding, derivation calculus, model, minimal resp. least model

1. Introduction

The Java-programming language is one of a few languages which allow *inheritance* and *inner classes*. The combination of these two features makes the language interesting for software engineers. To make a very short resume: two classes A and B nested in a class C share the resources of C, two classes D and E extending (inheriting) a class F obtain each a private copy of resources defined in F. It is not astonishing that it is a challenge to define the *semantics* of Java. In [IP02] Igarashi and Pierce presented an *axiomatic* approach towards the semantics of the language Java, namely an axiomatic way to reduce Java's semantics to that one of FJI (Featherweight Java with Inner classes). One inference Rule (ET-SimpEncl) works with a *metatheoretic* property as a premise, whereas the system does not offer any rules which would enable a proof of the premise.

A *declaration* of a class may contain the keyword **extends** followed by the *type* X naming the *direct superclass*. An example declaration may look like this:

class A **extends** $B.C\ \{\ldots\}$.

Now, since classes may be declared inside classes (and methods), it may happen that there are several classes named B resp. C in one program. Which of the classes named C is the *direct superclass* of class A? Which of the classes named B should be used in the process of identification of the direct superclass of class A? Notice, it may happen that no correct direct superclass exists, even if there are many candidates.

Subsection 5.2.1 of Section 5 of [IP02] is devoted to the *elaboration of types* which shall enable the *identification of direct superclasses*. Table Fig. 14 of paper [IP02, section 5.2.1] cites six *inference rules*. The authors define a *calculus*; we name it IPET-calculus and analyze it. The calculus' aim is to help identifying the direct superclasses in any *syntactically correct* Java-program. This identification is required to check I&P's *sanity conditions* so that these *static semantically correct* resp. *well-formed* (in the sense of I&P) programs can be assigned reasonable *dynamic semantics* as I&P do in [IP02].
We present some observations:

- The calculus is not *determinate*. It means that it is possible to derive two or more different classes as a direct superclass of a certain class.
- Moreover, there exist at least two different *models* of the calculus.
- Moreover, the models do not enjoy properties of this kind: *the intersection of two models is or contains a model;* or *there is a least model;* or *there is at most one minimal model*.

Therefore it is difficult to say what the meaning of the calculus is. The authors of [IP02] are aware that a straightforward *elaboration algorithm* obtained by reading the rules in a *bottom-up* manner might *diverge*. But a supplemented check for such divergent *recursive calls* is not an obvious method: Is every divergence always generated by a circular call from a recognizable finite set of patterns as the authors of [IP02] suggest? Does a divergent call mean that every former unfinished call has an undefined result as we know this phenomenon from *classical recursive functions*? Or does a divergent call mean that the algorithm proceeds with the most recent (or with a certain earlier) not yet finished application of the

critical Rule (ET-SimpEncl)? We shall show that the method can be specified in at least two different manners, i.e. the IPET-calculus may be used to define resp. deduce two different *inheritance* resp. *direct superclassing functions inh* from classes to classes.

We can go another approach and ask: has the IPET-calculus one or more models? It turns out that it has several *non-isomorphic* models. (Let us remark that every model can be constructed by a corresponding algorithm.) Hence it is necessary to add some hints of metatheoretical nature. Frequently, a calculus (or a theory) is accompanied by the metatheoretical hint: *choose the least model*. We are going to show that this does not work easily. For the intersection of two models needs not contain any model and there are at least two different *minimal* models.

The main source of the problems is in admitting a special inference Rule (ET-SimpEncl) in combination with Rule (ET-Long Sup). One of the premises of (ET-SimpEncl) is a metatheorem: $P \vdash X.D \Uparrow$. The formula $P \vdash X.D \Uparrow$ expresses the following property: for every class T there is no proof of the formula $P \vdash X.D \Rightarrow T$ or, more general, in a position of a premise, there is no valid formula $P \vdash X.D \Rightarrow T$. The last formula $P \vdash X.D \Rightarrow T$ says: Type *X.D* in (i.e. directly enclosed by the body of) class occurrence *P* elaborates (is bound) to class occurrence *T*. One remedy would be to eliminate the rule and to replace it by some rules that do not introduce metatheoretic premises and such that the premises are positive formulas. Another approach would consist in extending the language of the theory such that the expression $P \vdash X.D \Uparrow$ were a well-formed expression of the language and in adding some inference rules to deduce formulas of this kind. Nothing of this kind happens in [IP02].

Since a long time *expression nesting* and *static scoping* are well established notions in *predicate logics* [Fre1879] and *lambda calculus* [Chu41]. The notions were transferred to programming essentially by the Algol60-Report [Nau$^+$60/63]. In order to move Java into a direction where *object orientation* is in concordance with nesting of program structures, static scoping and *embedded software design* [Bjo09] and thus to follow the lines of Simula67 [DaNy67], Loglan82/88 [Bar$^+$82, KSW88] and Beta [MMPN93] the authors of Java[GJS96] have created their new Java Language Specification in 2000 [GJSB00] and allow inner classes. Igarashi and Pierce supported this development by their article [IP02] and earlier contributions.

Understanding and implementing nested program structures combined with static scoping has turned out to be quite a subtle topic in Algol- and Lisp-like languages [Dij60, GHL67, McG72, Lan73, Kan74, Ich80, Old81, WaGo84, WiMa92/97, Lan10, McC$^+$65, Sto84, Ste84]. Establishing static scope name binding and direct superclassing in the *external language* of the object oriented Java with inner classes is an as difficult and subtle task as the present article demonstrates.

The structure of our paper is as follows: Section 2 presents the calculus IPET of Igarashi and Pierce and raises questions. A decisive one is: does $P \vdash X \Rightarrow T$ denote a relation or a function? We present a seemingly evident, properly relational model of IPET, but realistic programming cannot accept multi valued types elaboration.

In Section 3 we translate the inference rules of IPET in such a way that the phrase "*the meaning of type X in environment P is class T*" is now expressed by the formula $bindfn(X \text{ in } P) = T$. We show the Examples 5 and 6 of well-formed programs, each one with different models, even minimal models, so that an only one least model cannot exist. So IPET resp. the equivalent calculus BIPET does not guarantee unique language semantics even if we restrict to functional (single valued) models.