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.

The succeeding (Sub-)Sections justify our claims on the program examples: Subsections 4.1 and 4.2 construct an infinite (!) family of *binding functions* $bind^{\nu}_{inh^{\nu}_0}, 1 \leq \nu \leq \infty$, so that Subsection 4.3 can show: each one is a model of IPET resp. BIPET. Subsection 5.1 goes further: each model has a minimal submodel and there are different minimal models. Every program which is well-formed w.r.t. one of these binding functions satisfies Igarashi's and Pierce's *sanity conditions*. So these conditions are no criterion to single out the right binding function and model. Reviewers of a former version of this article have claimed that the sanity conditions single out the right model.

Subsection 5.2 presents a first repair of IPET's resp. BIPET's shortcomings by a modified calculus BIPET´. The essential idea is to decompose undefinednesses (failures) of binding function applications into *finite failures*, represented by the so called *finite failure class* Fc, fictitiously joined to every program, and properly *infinite failures*, represented by *impossible derivability*. Strict $Fc - extension$ of Java's *official* binding function $bind^1_{inh^1_0}$ (named $bind_{inh_0}$ in [LSW09]) turns out to be the least model of BIPET´(Theorems 32,38, Corollary 39) in case the considered programs are binding well-formed.

Section 6 represents calculus BIPET´and its least model by an equivalent *recursive function definition* (or *recursive program* in the sense of [LoSi84]), Definition 40. If a Java-program is well-formed every valid formula $bind^1_{inh^1_0}(X\ in\ P) = T$ can be calculated by a successfully terminating call of $bindfn(X\ in\ P)$ and vice versa. Above this every call of $bindfn(X'\ in\ P)$ is terminating even if type X' does not denote any class occurrence. But the recursive program does not *decide* whether a Java-program is well-formed (i.e. the domain $dom_{inh^1_0}$ is the full set $\mathcal{C}$ of user declared classes) or not. Algorithm LSWA in [LSW08, LSW09] does so. A second, more profound repair BIPET˝simulates ideas in LSWA, BIPET˝'s least model is even the only model and the associated recursive function definition is a *semideciding algorithm* which can be readily transformed towards a deciding algorithm (Theorems 43,5, Corollary 46, Definitions 47,49).

2. Igarashi's and Pierce's calculus IPET for elaboration of types

Igarashi and Pierce [IP02, 5.2.1] are presenting a calculus IPET of derivation rules for a so called *elaboration relation of types*. The formulae of the calculus have the form (are written as) $\boxed{\mathtt{P} \vdash \mathtt{X} \Rightarrow \mathtt{T}}$ to be read: *The simple or qualified class type* X (i.e. a non-empty sequence of class identifiers separated by periods) *occurring inside the directly enclosing body of class declaration occurrence* P *is elaborated to* (resp. *is bound to*) *class declaration occurrence* T. In other and shorter words: the meaning of type X in class P is class T. For clarification: we have to differ between a *syntactical entity* and its *occurrences*, see the Algol68-report [Wij+68], because one and the same class declaration text (class for short) may occur several times at different places in a given program .

Observe that there is a bijection between class occurrences like P (or T) and their so called *absolute types (paths)* $\mathtt{C}_1.\cdots.\mathtt{C}_n$ where $\mathtt{C}_n$ is the *name* of class P, $\mathtt{C}_{n-1},\cdots,\mathtt{C}_1$ are the names of the successive class occurrences which enclose class occurrence P and $\mathtt{C}_1$ names a *top-level class*. To understand this phenomenon one should notice that the classes of a program form a *tree*. The *root* of the tree is a fictitious class *Root* which directly encloses all the top level classes of the program. Let nd be an internal *node* of the tree. It can be identified with the path leading from the root to it. Such a path consists of the names of enclosing classes. All direct inner classes declared in the class which is node nd are the *sons* of node nd. Therefore we are entitled to identify an occurrence of a class declaration and the absolute path of it. FJI requires that the *extends clause* has an *extends type* which is the absolute path of the denoted class occurrence whereas the external language of Java allows abbreviated extends types which are not necessarily absolute paths. Beside the user declared class occurrences in a Java-program there are two implicit, fictitious class occurrences:

1. $Root = \{\cdots\}$, which is enclosing all top level classes (and implicitly all other class occurrences) of the Java-program and which has no name nor extends clause. The authors of [IP02] represent $Root$ by its fictitious name $\star$ which users are not allowed to write. $\star.\mathtt{C}_1.\cdots.\mathtt{C}_n$ is identified with $\mathtt{C}_1.\cdots.\mathtt{C}_n$.

2. $Object$ = `class Object` $\{\cdots\}$ the name of which is `Object`, which is directly enclosed by $Root$ (so it is a top level class) and which has no extends clause also. There are no classes declared inside the body of $Object$ (see [GJSB00, GJSB05]).

3. To relieve our present investigations we neglect the implicit class occurrences of the Java utility package resp. we consider its classes as user declared ones.

Let us explain the meaning of some premises in the inference rules. In three rules one finds a premise of the form $CT(P.C) =$ **class** `C` **extends** `X` $\{\cdots\}$. In this way the authors Igarashi and Pierce express the fact that user declared class $P.C$ extends type X, i.e. extends that class which is the meaning of type X in the place where the declaration of class $P.C$ occurs[1]. Formulas of the form $P.C \in Dom(CT)$ mean: the program contains the class named C in its directly enclosing class which is identified with path P. Obviously, the formula of the form $P.C.D \notin dom(CT)$ expresses the fact that the class to be identified with the path $P.C$ does not contain any class named D. In Table 1 we present Igarashi's and Pierce's calculus IPET for elaboration of types. Below we collect some observations and comments.

1. It is not fully clear what the denotation $P \vdash X \Rightarrow T$ denotes! Should it be a *binary function* or a *ternary relation*? Observe that the above system has an unexpected model. Let π be a Java program. By $\mathcal{C}^\pi$ we denote the set of all user declared class occurrences of program π. By $\mathcal{SCT}^\pi$ we denote the set of *simple class types* occuring in π (`Object` included) and by $\mathcal{CT}^\pi = \mathcal{SCT}^{\pi+}$ the set of (*simple or qualified) class types* of the program π. Consider the following *subrelation* of the product
$$\mathcal{CT}^\pi \times (\mathcal{C}^\pi \cup \{Root, Object\}) \times (\mathcal{C}^\pi \cup \{Root, Object\}),$$
namely the set of all triples (X, P, T) with $X \in \mathcal{CT}^\pi$, $P, T \in \mathcal{C}^\pi \cup \{Root, Object\}$ where the name of class T coincides with the rightmost simple type in X. This subrelation satisfies all six rules. Hence one acceptable meaning of the predicate $P \vdash X \Rightarrow T$ is this subrelation. In the context of Java (or any programming language) such interpretation is of no worth as different classes with the same name D are allowed in a program and so every applied occurrence of D denotes different classes simultanuously. Practical programmers and compiler builders expect that predicate $P \vdash X \Rightarrow T$ denotes a single valued function. Some extra mechanism must be added

[1] Igarashi and Pierce require that every user declared class has an extends clause with a non-empty extends type. Java Language Specification [GJS96, GJSB00, GJSB05] allows empty extends clauses, and [LSW09] does so as well. In the following text we join Igarashi's and Pierce's practice.

Table 1. Igarashi's & Pierce's rules of elaboration

I. (ET-Object)	$P \vdash \texttt{Object} \Rightarrow Object$
II. (ET - In CT)	$\dfrac{P.C \in dom(CT)}{P \vdash C \Rightarrow P.C}$
III. (ET-SimpEncl)	$\dfrac{P.C.D \notin dom(CT) \qquad P \vdash D \Rightarrow T \\ CT(P.C) = \textbf{class } C \textbf{ extends } X \{\cdots\} \qquad P \vdash X.D \Uparrow}{P.C \vdash D \Rightarrow T}$
IV. (ET-SimpSup)	$\dfrac{P.C.D \notin dom(CT) \quad CT(P.C) = \textbf{class } C \textbf{ extends } X \{\cdots\} \\ P \vdash X.D \Rightarrow T}{P.C \vdash D \Rightarrow T}$
V. (ET-Long)	$\dfrac{P \vdash X \Rightarrow T \qquad T.C \in dom(CT)}{P \vdash X.C \Rightarrow T.C}$
VI. (ET-LongSup)	$\dfrac{P \vdash X \Rightarrow P'.D \qquad P'.D.C \notin dom(CT) \\ CT(P'.D) = \textbf{class } D \textbf{ extends } Y \{\cdots\} \qquad P' \vdash Y.C \Rightarrow U}{P \vdash X.C \Rightarrow U}$

to the six rules which reminds us to interpret the denotation in a functional sense. Featherweight Java with Inner classes FJI does so by using absolute paths as applied occurrences of class types (names).

2. Rule I.(ET-Object) reveals a slight inconsistency resp. restriction w.r.t. official Java with inner classes. I.o.w. [IP02] requires that the user is not allowed to choose the name `Object` for anyone of his declared classes. We have to respect this in our considerations and to discuss.

3. Rule III. (ET-SimpEncl) has four premises. The fourth premise of the form $\boxed{P \vdash X \Uparrow}$ is in fact a metatheorem "there is no class T such that the triplet $P \vdash X \Rightarrow T$ has a formal proof" or "there is no class T such that the triplet $P \vdash X \Rightarrow T$ is valid". This rule in combination with Rule VI. (ET-Long Sup) is a source of severe problems as we shall see below. Closer investigation of the calculus shows up that the more appropriate meaning of $P \vdash X \Uparrow$ is not a *general failure* of binding to any class T, but is a *specific binding* to an additional fictitious finite failure class (see Subsection 5.2) in concordance with theory of recursive programs [LoSi84], see Section 6.

4. There is **no** definition of the notion of *(formal) proof* in the system IPET of inference rules. Should one accept the classical definition of the notion of formal proof then the lack of possibilities to derive premises of the form $P \vdash X \Uparrow$ becomes evident. We know, the standard answer to this remark is: "*but everything is finite and therefore one can control the situation*". Is this one person

added to the definition of proof? What instructions are given to her/him which enable the task to recognize the impossibility of any proof?

5. A reader may hesitate to perceive what the following sentence is meaning: "*A straightforward elaboration algorithm obtained by reading the rules in a bottom-up manner might diverge.*" [IP02, 5.2.1, p.82]. Section 1 has already posed decisive critical questions.

6. The authors of [IP02] are aware that proof construction is not always possible. They make evident that their method may loop without exit [IP02, 5.2.1, p.82].

7. In fact, the task of type elaboration is divided in three subtasks: a) to find out whether the program is a well-formed one, b) to define a function inh which for every user declared class *P* returns the direct superclass of *P*, c) to find a binding function such that inheritance function inh is determined by the extends clauses of class declarations. It turns out that IPET does not help to solve task a) and to detect the possible errors in typing.

8. Seeing the incompleteness of the IPET-calculus (c.f. Rule III. together with Rule VI.) one may ask a slightly different question: is it true that IPET has exactly one model? We shall see that there are several models. In Sections 3 and 4 we prove that algorithm LSWA proposed in [LSW09] defines one IPET-model $\mathcal{M}_1 = bind^1_{inh^1_0}$ which is the official Java-binding function propagated in [GJSB00, GJSB05]. Above that in these sections we show up even further methods which lead to entirely different models $\mathcal{M}_\nu = bind^\nu_{inh^\nu_0}, 2 \leq \nu \leq \infty$, of IPET.

9. The next question: Is it possible to equip the calculus with an extra hint of the kind: consider the least one of all models as THE model of the IPET-calculus? This hope should be abandoned in the light of Section 5.

10. Our investigations lead to reasonable repairs of IPET which are conform to the type elaboration in the official Java Language Specification with inner classes [GJSB00, GJSB05, LSW09], see Subsection 5.2 and Section 6.

3. Binding functions, well-formedness, Igarashi's and Pierce's sanity conditions, models of calculus IPET resp. BIPET

Section 2 has pointed out that *type elaboration* or *binding relations* $P \vdash X \Rightarrow T$ in

$$\mathcal{P}(\mathcal{C}^{\pi RO} \times \mathcal{CT}^{\pi} \times \mathcal{C}^{\pi RO})$$

with $\mathcal{C}^{\pi RO} \stackrel{df}{=} \mathcal{C}^{\pi} \cup \{Root, Object\}$ should be *partial single valued binary functions* $bindfn^{\pi}(X\ in\ P) = T$ in

$$\mathcal{CT}^{\pi} \times \mathcal{C}^{\pi RO} \xrightarrow{part} \mathcal{C}^{\pi RO}$$

which for given class type X and class occurrence P determine class occurrence T – namely the meaning of class type occurrence X directly (immediately) enclosed by class occurrence P. We may say also: type (name) X, considered to be directly contained in the body of class (occurrence) P, is *bound to* (is *elaborated to*) class (occurrence) T. If binding function $bindfn^{\pi}$, applied to X and P, is undefined (has no result value) then no correct meaning of class type X inside class occurrence P can be found. In the sequel we shall omit the superscript π as always at most one program will be discussed.

Beside binding functions we consider *inheritance* or *direct* (immediate) *superclassing functions*

$$inh : \mathcal{C}^{RO} \xrightarrow{part} \mathcal{C}^{RO}$$

with its inherent conditions

$inh(Root)$ and $inh(Object)$ undefined,

$inh(P) \neq Root$

for all $P \in \mathcal{C}$ with $inh(P) \in \mathcal{C}^O$ in case $inh(P)$ is defined. These inheritance functions form a *subcpo*

$$\mathcal{INH} \stackrel{df}{=} \mathcal{C}^{RO} \xrightarrow{inherit} \mathcal{C}^{RO}$$

of the *full cpo*

$$\mathcal{C}^{RO} \xrightarrow{part} \mathcal{C}^{RO}.$$

The *domain* of inh is

$$dom_{inh} \stackrel{df}{=} \{K \in \mathcal{C} : inh(K) \in \mathcal{C}^O\},$$

$\mathcal{C}^O \stackrel{df}{=} \mathcal{C} \cup \{Object\}$, with its following extensions

$$dom^R_{inh} \stackrel{df}{=} dom_{inh} \cup \{Root\},$$
$$dom^O_{inh} \stackrel{df}{=} dom_{inh} \cup \{Object\},$$
$$dom^{RO}_{inh} \stackrel{df}{=} dom^R_{inh} \cup \{Object\}.$$

Definition 1: The *structure* of *a syntactically correct program* π is the tree $\mathcal{C}^{RO}$ of all its class occurrences, including $Root$ and $Object$, together with the operations $decl$ and ext. Operation $decl$ represents the tree's edges; $P' = decl(P)$ reads: P' *is directly declared in* P; $decl(Object)$ is $Root$ and $decl(Root)$ is undefined. $ext(P)$ is the type (name) in class P's extends-clause. ext is defined for all $P \in \mathcal{C}$ and is undefined for $Root$ and $Object$. □

Definition 2: A program π resp. its structure is called *binding well-formed w.r.t. binding function* $bindfn$ iff three conditions are fulfilled:

J_1) on *totality* of *induced inheritance function* inh_{bindfn}:

$$inh_{bindfn}(K) \stackrel{df}{=} bindfn(ext(K)\ in\ decl(K))$$

is total on all user declared classes $K \in \mathcal{C}$, $dom_{inh_{bindfn}} = \mathcal{C}$ (If $ext(K)$ or $decl(K)$ is undefined then so is $inh_{bindfn}(K)$ as is the usual understanding of function application).

J_2) on *non-existence of cycles* in the *induced dependency relation* dep_{bindfn} augmented (united) by the *directly declared in-relation decl* where:

$$dep_{bindfn} \stackrel{df}{=} \{\langle K, bindfn(ext(K)|^i\ in\ decl(K))\rangle : K \in \mathcal{C}, 1 \leq i \leq length(ext(K))\}$$

and all values $bindfn(ext(K)|^i\ in\ decl(K))$ have to exist as elements of $\mathcal{C}^O$.

J_3) on *non-paradoxical binding*: Let type $X \in \mathcal{CT}$ explicitly occur or be thought to occur applied directly in the body of class $K \in \mathcal{C}^{RO}$ and let $bindfn(X\ in\ K)$ be defined to be $T \in \mathcal{C}^{RO}$. Then T is different from $Root$ ($Root$ has no name) and T's name C is the rightmost simple type in X. If X is explicitly written down in the program or its structure then $bindfn(X\ in\ K)$ must be defined $\in \mathcal{C}^O$. □

Definition 2 generalizes the definition of well-formedness of a program structure as we know it from the official Java Language Specification JLS with inner classes [GJSB00, GJSB05], formalized in [LSW04, LSW09]. In case of JLS-Java the definition in the literature and our Definition 2 w.r.t. the JLS-Java-binding function are equivalent; a proof will be given in Subsection 4.2, Theorem 25.

Binding well-formedness implies the *sanity conditions* (1) to (7) in [IP02], Definition 2 is stronger than binding well-formedness in the sense of I&P where in FJI every applied occurrence of a class type X in any P is the absolute path of the denoted class $bindfn(X \ in \ P) \in \mathcal{C}^O$:
(1), (4), (5) are immediate implications of this latter fact.
(2) expresses: If L is an inner class named D and directly nested in the body of P then $L = P.D$ what follows from the definition of the selection oprator . (dot).
(3) expresses: `Object` $\notin name(\mathcal{C})$ what expresses I&P 's language restriction which does not allow `Object` as a user declared class name.
(6) says: there are no cycles in the subtyping relation `<:` what is an implication of J_2) because
$K{<:}\, L$ means $inh^{\star}_{bindfn}(K) = L$.
(7) is prohibiting a class from extending one of its inner classes, i.e. $decl^+(inh^+_{bindfn}(T)) \neq T$, what is an implication of J_2). (7) implies especially $T \not<: T.U$.

Binding well-formedness is a necessary condition of *well-formed* (i.o.w. *static semantically correct) programs* π, i.e. those syntactically correct programs which *language specificaiton has assigned dynamic semantics to*. Often we shall drop the word “binding” in “binding well-formedness”.

Since calculus IPET shall be employed to establish a *distinguished binding function* and because a system of rules might have several *complying functions* or *models* we are interested in extreme ones, preferably least ones. As IPET turns out not to have just one least complying function or model we try to search for minimal models, then to see which one is especially appropriate and to look for repaired calculi which come up with exactly one least model or even exactly one model.

In order to have an easier way of comparison we translate IPET's rules to the mode of expression in [LSW08, LSW09] what is yielding calculus BIPET in Table 2.

Definition 3: Let us consider a program π which is binding well-formed w.r.t. $bindfn$. We say $bindfn$ is a *model of calculus* BIPET iff $bindfn$ satisfies all six BIPET-rules. □

Although a definition of the family of different binding functions $bind^{\nu}_{inh^{\nu}_0}$, $1 \leq \nu \leq \infty$, will be presented only later in Definition 8 and Corollary 22 we anticipate Theorem 4 because it is central due to its consequences:

Theorem 4: Let a program be well-formed w.r.t. $bind^{\nu}_{inh^{\nu}_0}$. Then the single valued function $bind^{\nu}_{inh^{\nu}_0}$ is satisfying all six rules of BIPET, i.e. is a model. We may even say: $bind^{\nu}_{inh^{\nu}_0}$ is a *uniform model* of BIPET. We speak of a uniform model because the model property is not restricted to some special programs, but refers to the variety of all program structures well-formed w.r.t. $bind^{\nu}_{inh^{\nu}_0}$.

Theorem 4's proof will be given in Subsection 4.3. Among the considered models there is Java's official binding function $bind^1_{inh^1_0}$ [GJSB00, GJSB05, LSW08, LSW09], see the later Definition 23, Lemma 24

Table 2. Rules of calculus IPET are interpreted into calculus BIPET

I. (BET-Object)	$bindfn(\texttt{Object} \text{ in } P) = Object$
II. (BET - InCT)	$\dfrac{\text{class } P \text{ has a direct inner class} \in \mathcal{C}^O \text{ named } C}{bindfn(C \text{ in } P) = P.C}$
III. (BET-SimpEncl)	$\dfrac{\begin{array}{l} bindfn(D \text{ in } P) = T \\ \text{class } P.C \in \mathcal{C} \text{ has no direct inner class named } D \\ bindfn(ext(P.C).D \text{ in } P) \text{ undefined} \end{array}}{bindfn(D \text{ in } P.C) = T}$
IV. (BET-SimpSup)	$\dfrac{\begin{array}{l} \text{class } P.C \in \mathcal{C} \text{ has no direct inner class named } D \\ bindfn(ext(P.C).D \text{ in } P) = T \end{array}}{bindfn(D \text{ in } P.C) = T}$
V. (BET-Long)	$\dfrac{\begin{array}{l} bindfn(X \text{ in } P) = T \\ \text{class } T \text{ has a direct inner class named } C \end{array}}{bindfn(X.C \text{ in } P) = T.C}$
VI. (BET-LongSup)	$\dfrac{\begin{array}{l} bindfn(X \text{ in } P) = P'.D \\ \text{class } P'.D \in \mathcal{C} \text{ has no direct inner class named } C \\ bindfn(ext(P'.D).C \text{ in } P') = U \end{array}}{bindfn(X.C \text{ in } P) = U}$

Variables P, P' range over $\mathcal{C}^{RO}$, T, U over $\mathcal{C}^O$, X over $\mathcal{CT}$ and C, D over simple types $\mathcal{SCT}$

and Theorem 25. In order to understand better the translation from IPET to BIPET the following discussion might be a good exercise.

Let us shortly consider the following premise of IPET in the three Rules III., IV., VI.:

$CT(P.C) \ = \ \texttt{class}\ C\ \texttt{extends}\ X\{\dots\}$ resp.

$CT(P'.D) = \ \texttt{class}\ D\ \texttt{extends}\ Y\{\dots\}$.

In BIPET this premise is to be formulated fully as follows:

class $P.C$ has a defined extends type $X \in \mathcal{CT}$ which is $ext(P.C)$ resp.

class $P'.D$ has a defined extends type $Y \in \mathcal{CT}$ which is $ext(P'.D)$.

Since ext is undefined only if applied to $Root$ or $Object$ and since $P.C$ resp. $P'.D$ is different from $Root$ the premise in BIPET is equivalent to

class $P.C$ is different from $Object$ or $P.C \in \mathcal{C}$ resp.

class $P'.C$ is different from $Object$ or $P'.C \in \mathcal{C}$.

In the two Rules IV., VI. this being different from $Object$ may be deleted as it is an implication of the other premises:

Since $bindfn(ext(P.C).D \ in \ P)$ is defined $= T \in \mathcal{C}^O$ $\quad ext(P.C)$ is also defined $\in \mathcal{CT}$ and so $P.C$ is different from $Object$.

Similar reasoning holds for $P'.D$.

In Rule III. this implication is not valid. Consider the following user written part of a program structure

class `D` **extends** `Object` $\{\ldots\}$,

which obeys I&P's language restriction, and the official Java-binding function $bind^1_{inh^1_0}$. The structure is well-formed w.r.t. this binding function which complies with all six rules. Three premises in Rule III. are holding:

$bind^1_{inh^1_0}(\mathtt{D}\ in\ Root)$ is class D above;

class $Root.\mathtt{Object}$ is $Object$ and has no direct inner class named `D` ;

$bind^1_{inh^1_0}(ext(Root.\mathtt{Object}).\mathtt{D}\ in\ Root)$ is undefined as $ext(Object)$ is undefined.

But the fourth premise

$Root.\mathtt{Object} \in \mathcal{C} = \{D\}$

does not hold.

The serious dilemma with IPET resp. BIPET is that there are programs where each one of them has different models, even different minimal models, so that there is no least model. Consequence: IPET resp. BIPET in its present shape is no recommendable help to assign appropriate dynamic semantics to a Java-program with inner classes.

Example 5:

π_a: **class** `A` **extends** `Object` {
        class `E` **extends** `Object` { }
        class `C` **extends** `Object` { }
    }
    class `B` **extends** `A` {
        class `E` **extends** `Object` { }
        class `D` **extends** `C` {
            class `F` **extends** `E` { }
        }
    }

π_a has at least two different models, namely $bind^1_{inh^1_0}$ and $bind^2_{inh^2_0}$ $(= bind^{\nu}_{inh^{\nu}_0}, 2 \le \nu \le \infty)$, defined in Subsection 4.2, Corollary 22. These are two binding functions such that their induced inheritance functions (see totality property J_1) in Definition 2 are

$inh_{bind^1_{inh^1_0}} = inh^1_0$ and $inh_{bind^2_{inh^2_0}} = inh^2_0$.

We have identical inheritances

$inh^1_0(A) = inh^2_0(A) = Object$
$inh^1_0(B) = inh^2_0(B) = A$
$inh^1_0(A.\mathtt{E}) = inh^2_0(A.\mathtt{E}) = Object$
$inh^1_0(C) = inh^2_0(C) = Object$
$inh^1_0(B.\mathtt{E}) = inh^2_0(B.\mathtt{E}) = Object$
$inh^1_0(D) = inh^2_0(D) = C$,

but also different ones

$inh^1_0(F) = B.\mathtt{E} \neq inh^2_0(F) = A.\mathtt{E}$

(see Definitions 8,13 and Corollary 22. $A, B, C, D, A.$E, $B.$E, F denote the seven classes named A, B, C, D, E, E, F). As all these inheritances remain preserved (see Definition 2) as soon as we go to minimal submodels (which exist due to Theorem 29) we have even at least two different minimal models. □

This statement remains true even if all inheritances are identical. To see this fact we change Example 5 towards

Example 6: We exchange the body of class D in π_a by a body without any class declarations inside. We delete class F and assume there is an applied occurrence of simple type E in D which is not declared inside the body of D. So we get program

π_b: **class** A **extends** Object {
    class E **extends** Object { }
    class C **extends** Object { }
  }
  class B **extends** A {
    class E **extends** Object { }
    class D **extends** C {
        ...E...
    }
  }.

Here again we have an inequality
$(\star)\qquad bind^1_{inh^1_0}(\texttt{E}\ in\ D) = B.\texttt{E} \neq bind^2_{inh^2_0}(\texttt{E}\ in\ D) = A.\texttt{E}$
(for analogous reasons as above in Example 5).
In order to conclude that once again we have different minimal models we can reason as follows with the help of the Theorems 4 and 29: If we had only one minimal model $bindfn^{min}$, this would be a minimal submodel both of $bind^1_{inh^1_0}$ and of $bind^2_{inh^2_0}$. So due to $(\star)$ $bindfn^{min}(\texttt{E}\ in\ D)$ is necessarily undefined $(\star\star)$. But in Subsection 5.2 we shall demonstrate that there is a $bindfn^1$ which is a minimal submodel of $bind^1_{inh^1_0}$; so $bindfn^{min}$ is identical $bindfn^1$. We calculate $bindfn^1(\texttt{E}\ in\ D) = B.\texttt{E}$ what is contradicting the undefinedness $(\star\star)$ above. □

4. Langmaack's, Salwicki's and Warpechowski's way to construct type elaboration or binding functions and their property to be models of IPET resp. BIPET

4.1. Definition of a family of binding functions $bind^{\nu}_{inh}$

Consider the ordered alphabet $\mathcal{A}$ of the two operator symbols in and de, where we define in to be *less than* de: $in \prec de$. This order is inducing a *lexicographical (from the right) order* in the set $\mathcal{A}^\star$ of all words over $\mathcal{A}$. E.g., the words

$in \prec\ de^\frown in \prec de \prec in^\frown de$

are in this order. Notice, this order is total, but not well-founded. So we have to be careful; an infinite set of words might have no least one.

Let $w = id_1^\frown id_2^\frown \cdots ^\frown id_n$ be a word $\in \mathcal{A}^\star, n \geq 0,\ id_j \in \mathcal{A}$. Let P be a class. The word w applied to class P w.r.t. the semantical operation inh is the class

$$w(P) \stackrel{df}{=} id_1(id_2(\cdots(id_n(P))\cdots)) \in \mathcal{C}^{RO}$$

in case of definedness where $id_j = de$ is interpreted by the semantical "directly declared in" function $decl$ and $id_j = in$ by a semantical inheritance function inh. The empty word λ with $n = 0$ yields $\lambda(P) = P$.

Now we consider the following pairs $(\mathcal{A}^{d\nu}, \mathcal{A}^{i\nu})$ of special subsets of $\mathcal{A}^\star$ such that each pair will induce a model of BIPET:

1. $(\mathcal{A}^{d1}, \mathcal{A}^{i1}) = (in^{\star\frown} de^\star, in^\star)$ is responsible for the BIPET-model $bind_{inh_0}$ in [LSW09], i.e. for the official language specification of Java with inner classes [GJSB00, GJSB05].
2. $(\mathcal{A}^{d\infty}, \mathcal{A}^{i\infty}) = (\mathcal{A}^\star, \mathcal{A}^\star)$ is responsible for the BIPET-model $Bind_{inh_{B0}}$ in [LSW08b].
3. $(\mathcal{A}^{d\nu}, \mathcal{A}^{i\nu}) = ((in^{\star\frown} de^\star)^\nu, (in^{\star\frown} de^\star)^{\nu-1\frown} in^\star)$ is responsible for further models of BIPET for every natural number $2 \leq \nu < \infty$.

These pairs $(\mathcal{A}^{d\nu}, \mathcal{A}^{i\nu})$ have characteristic properties which are used in later proofs.

1. $\mathcal{A}^{d\nu}$ is de-closed, i.e. $\lambda \in \mathcal{A}^{d\nu}$ and $w \in \mathcal{A}^{d\nu}$ implies $w^\frown de \in \mathcal{A}^{d\nu}$.
2. $\mathcal{A}^{i\nu}$ is in-closed, i.e. $\lambda \in \mathcal{A}^{i\nu}$ and $w \in \mathcal{A}^{i\nu}$ implies $w^\frown in \in \mathcal{A}^{i\nu}$.
3. $\mathcal{A}^{d\nu} = \mathcal{A}^{i\nu\frown} de^\star$.
4. $\mathcal{A}^{i\nu\frown} in \subseteq \mathcal{A}^{i\nu} \cap \mathcal{A}^{d\nu}$.
5. $w^\frown in \in \mathcal{A}^{i\nu}$ implies $w \in \mathcal{A}^{i\nu}$.
6. $w^\frown de \in \mathcal{A}^{d\nu}$ implies $w \in \mathcal{A}^{d\nu}$.

Every such pair $(\mathcal{A}^{d\nu}, \mathcal{A}^{i\nu})$ *induces* a *binding function*

$$bind^\nu_{inh} : \mathcal{CT} \times \mathcal{C}^{RO} \xrightarrow{part} \mathcal{C}^{RO}$$

associated to and parameterized by a given inheritance function inh.

We need *auxiliary binding functions*

$$bd^{\iota\nu}_{inh} : \mathcal{SCT} \times \mathcal{C}^{RO} \xrightarrow{part} \mathcal{C}^{RO}$$

where $\mathcal{A}^{\iota\nu}$ is a subset (later equal $\mathcal{A}^{d\nu}$ or $\mathcal{A}^{i\nu}$) of $\mathcal{A}^\star$:

Definition 7:

$$bd^{\iota\nu}_{inh}(C\ in\ P) \overset{df}{=} \begin{cases} T.C & \text{if } C \in \mathcal{SCT}, P \in dom^{RO}_{inh} \text{ and} \\ & \bullet \text{ there is a lexicographically least word} \\ & \quad w \in \mathcal{A}^{\iota\nu} \text{ such that} \\ & \bullet \text{ the } w \text{-associated path from } P \text{ to } T \text{ has no} \\ & \quad \text{repeated nodes and is fully located in } dom^{RO}_{inh}, \\ & \quad w(P) = T, \text{and} \\ & \bullet\ T.C \text{ is defined } \in \mathcal{C}^O \text{ for the end node } T \text{ of the} \\ & \quad \text{path } (T.C \text{ is not necessarily in } dom^{RO}_{inh}) \text{ and} \\ & \bullet \text{ there are only finitely many words } v \in \mathcal{A}^{\iota\nu} \\ & \quad \text{lexicographically less than } w \text{ with } v \text{-associated} \\ & \quad \text{paths from } P \text{ fully located in } dom^{RO}_{inh}, \\ & \quad v(P) \in dom^{RO}_{inh} \\ \text{undefined} & \text{otherwise} \end{cases} \qquad \square$$

We call a whole w-associated path from $P \in dom^{RO}_{inh}$ via $T \in dom^{RO}_{inh}$ to $T.C \in \mathcal{C}^O$ *least* C*-admissible w.r.t.* $\mathcal{A}^{\iota\nu}$.

Definition 8 inductively over the length of types X: Let a pair $(\mathcal{A}^{d\nu}, \mathcal{A}^{i\nu})$ be given:[2]

$$bind^{\nu}_{inh}(X\ in\ P) \overset{df}{=} \begin{cases} bd^{d\nu}_{inh}(X\ in\ P) & \text{if } length(X) = 1 \text{ and} \\ & bd^{d\nu}_{inh}(X\ in\ P) \text{ is defined} \in \mathcal{C}^O \\ bd^{i\nu}_{inh}(C\ in\ P') & \text{if } X = X'.C, length(X') \geq 1, length(C) = 1, \\ & P' = bind^{\nu}_{inh}(X'\ in\ P) \text{ is defined } \in \mathcal{C}^O \text{ and} \\ & bd^{i\nu}_{inh}(C\ in\ P') \text{ is defined } \in \mathcal{C}^O \\ \text{undefined} & \text{otherwise} \end{cases} \qquad \square$$

Remark 9: Every resulting class $T.C = bind^{\nu}_{inh}(X\ in\ P) \in \mathcal{C}^O$ has as its name the rightmost simple type C in (qualified) type X ; so $bind^{\nu}_{inh}$ is *non-paradoxical*, see J_3) in Definition 2. $Root$ cannot occur as a result as $Root$ has no name. This new and more general binding function $bind^{\nu}_{inh}$ should not be intermixed with the old $bind_{inh}$ in [LSW09]. The latter one corresponds to the newer one with parameters $\mathcal{A}^{d1}, \mathcal{A}^{i1}, \nu = 1$. □

[2] We could try to define $bind^{\nu}_{inh}$ with the help of recursive function definitions. But such endeavour requires careful preparations and it is difficult to formulate sound proofs.

Remark 10: Let $X = C_1. \cdots .C_n, C_i$ simple types $\in \mathcal{SCT}$, $n \geq 1$, $C = C_n$. Then

$$T.C = bind^{\nu}_{inh}(X\ in\ P) \in \mathcal{C}^O$$

holds if and only if there is a chain of n least C_i-admissible paths, i.e. the first path from $P = P_1$ via T_1 to $T_1.C_1 = P_2$ is least C_1-admissible w.r.t. $\mathcal{A}^{d\nu}$ and the i-th path, $2 \leq i \leq n$, from $T_{i-1}.C_{i-1} = P_i$ via T_i to $T_i.C_i = P_{i+1}$ is least C_i-admissible w.r.t. $\mathcal{A}^{i\nu}$ with $T_n = T, T_n.C_n = P_{n+1} = T.C$. All nodes are in dom^{RO}_{inh} with the possible exception of $T.C$. We call such whole path from $P = P_1$ via $T_1, T_1.C_1 = P_2, \cdots, T_n$ to $T_n.C_n = P_{n+1} = T.C$ *least* X*-admissible*. □

Remark 11: The styles of definitions for $bind_{inh}$ in [LSW09] and $Bind_{inh}$ in [LSW08b] deviate slightly from the present style. Adaptation to the present style leads to slightly different functions. But these variants lead to the same monotonous functionals $Bdfl' = bdfl'^1$ and $BDfl' = bdfl'^{\infty}$, see Definition 13, and fixed points $inh_0 = inh^1_0$ and $inh_{B0} = inh^{\infty}_0$, see Corollary 22. The two Lemmas 11 and 12 in [LSW09] turn out to be immediately evident in our present presentation (due to stronger usage of the notion "admissibility"). □

4.2. Continuous binding functionals $bdfl'^{\nu}$ and their least fixed points inh^{ν}_0, shortly inh_0

The binding functions, which are to be singled out as models of IPET resp. BIPET, are determined by specific inheritance functions inh^{ν}_0. We might try to do so via a fixed point of the so called *natural functional* $bdfl^{\nu}$ with

$$bdfl^{\nu}(inh)(A) \overset{df}{=} bind^{\nu}_{inh}(ext(A)\ in\ decl(A))$$

for $A \in \mathcal{C}$ and $bdfl^{\nu}(inh)(A)$ undefined for $A \in \{Root, Object\}$. But these $bdfl^{\nu}$ are endangered to be not continuous functionals so that we might have no "natural" least fixed points in the sense of D. Scott's fixed point theory.

So we consider specific inheritance functions, called states[3], the set

$$\mathcal{STS} = \mathcal{C}^{RO} \overset{state}{\longrightarrow} \mathcal{C}^{RO}$$

of which is a subcpo of the subcpo

$$\mathcal{INH} = \mathcal{C}^{RO} \overset{inherit}{\longrightarrow} \mathcal{C}^{RO}$$

of the full cpo

$$\mathcal{C}^{RO} \overset{part}{\longrightarrow} \mathcal{C}^{RO}.$$

Definition 12: An inheritance function inh is called a *state* iff for all classes $K \in dom_{inh}$ the following two relations

$$inh(K) \in dom^{O}_{inh},$$
$$decl(K) \in dom^{R}_{inh}$$

and the equation

$$inh(K) = bind^{\nu}_{inh}(ext(K)\ in\ decl(K))$$

are holding. So all node classes in an associated least $ext(K)$-admissible path are in dom^{RO}_{inh}. □

Let's come to the desired functional $bdfl'^{\nu}$. We introduce the following logical formula $\alpha^{\nu}_{inh}(A)$:

[3] Algorithm LSWA [LSW09] is running through states which are special inheritance functions. This fact has motivated us towards the notion of "state".

$$decl(A) \in dom^R_{inh} \wedge A \neq Root \wedge A \neq Object \wedge$$
$$bind^{\nu}_{inh}(ext(A)\ in\ decl(A)) \in dom^O_{inh}$$

with $A \in \mathcal{C}^{RO}$.

Definition 13: The desired functional is

$$bdfl'^{\nu}(inh)(A) \overset{df}{=} \begin{cases} bind^{\nu}_{inh}(ext(A)\ in\ decl(A)) & \text{if } \alpha^{\nu}_{inh}(A) \\ \text{undefined} & \text{otherwise} \end{cases}$$ □

Lemma 14: If inh is a state then $inh' \overset{df}{=} bdfl'^{\nu}(inh)$ is an inheritance function and is an extension of inh.

Proof: Let $A \in dom_{inh}$. Claim: $inh(A) = inh'(A)$.

Then $A \neq Root$, $A \neq Object$, $decl(A) \in dom^R_{inh}$, $inh(A) = bind^{\nu}_{inh}(ext(A)\ in\ decl(A)) \in dom^O_{inh}$ because inh is a state. So $\alpha^{\nu}_{inh}(A)$ is holding and $inh'(A)$ is equal $bind^{\nu}_{inh}(ext(A)\ in\ decl(A))$ by definition of $bdfl'^{\nu}$ and inh'. So $inh'(A) = inh(A)$, i.e. inh' is an extension of inh. ■

Remark 15: Let inh be a state and $A \in \mathcal{C} \setminus dom_{inh}$ with $decl(A) \in dom^R_{inh}$ (i.e. A is a so called *candidate* w.r.t. algorithm LSWA, see [LSW09]) and $bind^{\nu}_{inh}(ext(A)\ in\ decl(A)) \in dom^O_{inh}$ (i.e. A is a so called *generating candidate*). Then let us denote the extension inh' of inh by inh^A where the undefined resulting value $inh(A)$ $(A \notin dom_{inh}\ !)$ is replaced by $inh'(A) \overset{df}{=} bind^{\nu}_{inh}(ext(A)\ in\ decl(A))$. □

Lemma 16: If inh is a state then $inh' \overset{df}{=} bdfl'^{\nu}(inh)$ is also a state.

Proof: Let $A \in dom_{inh'}$. We have to show that

$inh'(A) \in dom^O_{inh'}$ and
$decl(A) \in dom^R_{inh'}$ and
$inh'(A) = bind^{\nu}_{inh'}(ext(A)\ in\ decl(A))$.

Because $inh'(A)$ is defined $\alpha^{\nu}_{inh}(A)$ is holding and

$inh'(A) = bind^{\nu}_{inh}(ext(A)\ in\ decl(A)) \in dom^O_{inh}$.

Since inh' is an extension of inh we have

$inh'(A) \in dom^O_{inh'}$.

Since $decl(A) \in dom^R_{inh}$ we have

$decl(A) \in dom^R_{inh'}$.

As $decl(A) \in dom^R_{inh}$ and inh' is an extension of inh and $bind^{\nu}_{inh}(ext(A)\ in\ decl(A)) \in dom^O_{inh}$ all least $ext(A)$-admissible paths from $decl(A)$ w.r.t. inh' are also least $ext(A)$-admissible w.r.t. inh (and trivially vice versa)

$bind^{\nu}_{inh}(ext(A)\ in\ decl(A)) = bind^{\nu}_{inh'}(ext(A)\ in\ decl(A))$

is holding. So we have

$inh'(A) = bind^{\nu}_{inh'}(ext(A)\ in\ decl(A))$. ■

Remark 17 on direct and indirect successors of states: If in this proof of Lemma 16 A is a generating candidate and if we replace inh' by inh^A then we have a proof for: inh^A is a state. We call inh^A a *direct successor state* of inh and write $inh \prec^{DS} inh^A$ with the transitive closure $\prec^S$ of $\prec^{DS}$ which is an irreflexive partial order in the set of states

$$\mathcal{STS} = \mathcal{C}^{RO} \overset{state}{\longrightarrow} \mathcal{C}^{RO}.$$ □

Theorem 18: $bdfl'^{\nu}$ is a monotonous functional (and consequently is continuous because $\mathcal{STS}$ is finite).

Proof: Let inh_1, inh_2 be two states and inh_2 an extension of inh_1.
Claim: $bdfl'^{\nu}(inh_2) = inh'_2$ is an extension of $bdfl'^{\nu}(inh_1) = inh'_1$.
I.e. let $A \in dom_{inh'_1}$.
Claim: $inh'_2(A) = inh'_1(A)$.
As $inh'_1(A)$ is defined $\alpha^{\nu}_{inh_1}(A)$ is holding. So

$decl(A) \in dom^R_{inh_1} \subseteq dom^R_{inh_2}$,

$inh'_1(A) = bind^{\nu}_{inh_1}(ext(A)\ in\ decl(A)) \in dom^O_{inh_1} \subseteq dom^O_{inh_2}$.

Case 1: $A \in dom_{inh_1}$.
Then $A \in dom_{inh_2},\ inh_1(A) = inh_2(A) = inh'_1(A) = inh'_2(A)$.
Case 2: $A \in dom_{inh'_1} \setminus dom_{inh_1}$.
Then $inh'_1(A) = bind^{\nu}_{inh_1}(ext(A)\ in\ decl(A))$
$= bind^{\nu}_{inh_2}(ext(A)\ in\ decl(A)) = inh'_2(A)$
for the same reasons as in Lemma 16. ■

Remark 19 on modular confluence: The relation $\prec^{DS}$ is *modularly confluent*, i.e. if $inh \prec^{DS} inh^A, inh \prec^{DS} inh^{A'}$ and $inh^A \neq inh^{A'}$ then there is a common direct successor state sst with $inh^A \prec^{DS} sst$ and $inh^{A'} \prec^{DS} sst$, especially $sst = inh^{AA'} = inh^{A'A}$ (due to an easy consideration on admissible paths). If a $\prec^{DS}$-chain

$inh = sst_0 \prec^{DS} sst_1 \prec^{DS} \cdots \prec^{DS} sst_n, n \geq 0,$

ends up in a maximal sst_n then sst_n is uniquely determined by inh. Every state inh has such a uniquely determined maximal successor state inh^{max}. Obviously

$bdfl'^{\nu}(inh) = inh \cup \bigcup_{inh \prec^{DS} sst} sst$

is holding. Therefore a state inh is maximal w.r.t. $\prec^S$ if and only if inh is a fixed point of $bdfl'^{\nu}$. The maximal successor state $inh^{max}_{\perp}$ is the least fixed point of $bdfl'^{\nu}$, obviously, where $inh_{\perp}(A)$ is undefined for all $A \in \mathcal{C}^{RO}$ ($inh^{max}_{\perp}$ depends on ν implicitly!). □

Remark 20: If state inh has no cycle then inh^A has none as well since A is a generating candidate. $inh^{max}_{\perp}$ is single valued, has no cycles and there are even no repeated nodes in paths associated to $w \in \mathcal{A}^{\star}$ and starting from $P \in dom^{RO}_{inh^{max}_{\perp}}$. Especially no inheritance chain of such class P leads to an enclosed (inner) class of it. □

Remark 21 on the dependency relation: If inh is an inheritance function then the associated *induced dependency relation* $dep_{bind^{\nu}_{inh}}$ is defined due to J$_2$) in Definition 2 as

$\{\langle A, bind^{\nu}_{inh}(ext(A) \mid^i in\ decl(A))\rangle : A \in dom_{inh}, 1 \leq i \leq length(ext(A))\}$.

Relation $dep_{bind^{\nu}_{inh}}$ may be multi valued even if inh is single valued.
If inh is a state and $dep_{bind^{\nu}_{inh}}$ has no cycle then so it is for relation $dep_{bind^{\nu}_{inh^A}}$, because we may easily deduce

$$dep_{bind^{\nu}_{inh^A}} = dep_{bind^{\nu}_{inh}} \cup \{\langle A, bind^{\nu}_{inh}(ext(A) \mid^i in\ decl(A))\rangle : 1 \leq i \leq length(ext(A))\}$$

where $A \in \mathcal{C} \setminus dom_{inh}$ with $decl(A) \in dom^{RO}_{inh} \subseteq \mathcal{C}^{RO}$ is the generating candidate. Relation $dep_{bind^{\nu}_{inh^{max}_{\perp}}}$ has no cycles and even the augmented dependency relation $decl \cup dep_{bind^{\nu}_{inh^{max}_{\perp}}}$ has no cycles. This statement will be important in Corollary 22 and in Theorems 38 and 45. □

D.Scott's fixed point theorem [LoSi84] applied to continuous functional $bdfl'^{\nu}$ of Theorem 18 assures the existence of (the) least fixed point $\mu\, bdfl'^{\nu}$ (we name it inh_0^{ν}) in cpo $\mathcal{STS} = (\mathcal{C}^{RO} \stackrel{state}{\longrightarrow} \mathcal{C}^{RO})$ which can be approximated by repeated applications of $bdfl'^{\nu}$ to the bottom inheritance function $inh_{\perp}$.

Corollary 22 of Theorem 18: The functional

$$bdfl'^{\nu} : (\mathcal{C}^{RO} \stackrel{state}{\longrightarrow} \mathcal{C}^{RO}) \stackrel{tot,cont}{\longrightarrow} (\mathcal{C}^{RO} \stackrel{state}{\longrightarrow} \mathcal{C}^{RO})$$

has exactly one least fixed point ($\kappa = card(\mathcal{C})$)

$$inh_0^{\nu} \stackrel{df}{=} \mu\, bdfl'^{\nu} = \bigcup_{j\, \in Nat_0} bdfl'^{\nu\, j}(inh_{\perp}) = bdfl'^{\nu\, \kappa}(inh_{\perp})$$

which is, due to Remark 19,

$$= \bigcup_{inh_{\perp} \stackrel{\prec S}{=} inh} inh \;=\; inh_{\perp}^{max}$$

and which allows an influential characterization of well-formedness:
A Java-program is binding well-formed w.r.t. binding function $bind^{\nu}_{inh_0^{\nu}}$ iff $dom_{inh_0^{\nu}} = \mathcal{C}$.
Supplement: In case of binding well-formedness inh_0^{ν} is the inheritance function $inh_{bind^{\nu}_{inh_0^{\nu}}}$ induced by $bind^{\nu}_{inh_0^{\nu}}$.

Proof: Let $dom_{inh_0^{\nu}} = \mathcal{C}$ and $K \in \mathcal{C}$. Then

$$\begin{aligned}\mathcal{C}^{O} \ni inh_0^{\nu}(K) &= bdfl'^{\nu}(inh_0^{\nu})(K)\\ &= bind^{\nu}_{inh_0^{\nu}}(ext(K)\ in\ decl(K))\\ &= inh_{bind^{\nu}_{inh_0^{\nu}}}(K)\end{aligned}$$

due to fixed point property, Definition 13 and induced inheritance in J_1). So

$$inh_0^{\nu} = inh_{bind^{\nu}_{inh_0^{\nu}}}$$

and J_1) holds. J_2) and J_3) hold due to Remarks 21 and 9.
Vice versa, let the program be well-formed w.r.t. $bind^{\nu}_{inh_0^{\nu}}$. Then for the induced inheritance $dom_{inh_{bind^{\nu}_{inh_0^{\nu}}}} = \mathcal{C}$ holds and $inh_{bind^{\nu}_{inh_0^{\nu}}}$ is a state. Assume $dom_{inh_0^{\nu}}$ were strictly smaller than $\mathcal{C}$. Consider any candidate $K \in \mathcal{C} \setminus dom_{inh_0^{\nu}}$ with $decl(K) \in dom^{R}_{inh_0^{\nu}}$ (there is at least one such K) and $inh_{bind^{\nu}_{inh_0^{\nu}}}(K) = bind^{\nu}_{inh_0^{\nu}}(ext(K)\ in\ decl(K)) = M \in \mathcal{C}$. M is $\notin dom^{O}_{inh_0^{\nu}}$; otherwise $inh_{\perp}^{max}$ were strictly larger than inh_0^{ν}. So M is also a candidate. So $inh_{bind^{\nu}_{inh_0^{\nu}}}$ and $dep_{bind^{\nu}_{inh_0^{\nu}}}$ had a cycle what is contradicting J_2).
Now a proof of the supplement can easily be done by ideas in the first direction's proof. ■

Due to Remark 19 algorithm LSWA in [LSW08, LSW09] is a refined algorithm of the fixed point approximation to compute function inh_0^1 in case $\nu = 1$. Analoguous algorithms LSWA$^{\nu}$ are available for all indices $1 \leq \nu \leq \infty$ to determine the least fixed points inh_0^{ν}, see Appendix.

In the sequel we sometimes drop the superscript of inh_0^{ν} and write simply inh_0. We would like to avoid overloading but should not forget the dependency on ν.

We see: Every pair $\mathcal{A}^{d\nu}, \mathcal{A}^{i\nu}$ of word sets induces a binding functional $bind^{\nu}$, a continuous functional $bdfl'^{\nu}$, its least fixed point inh_0^{ν}, its domain $dom_{inh_0^{\nu}} \subseteq \mathcal{C}$ and binding function $bind^{\nu}_{inh_0^{\nu}}$.

Now we would like to clarify the relation between well-formedness of programs and program structures w.r.t. binding functions (Definition 2) and well-formedness in the sense of the official Java Language Specification with inner classes JLS.

Definition 23: Official Java Language Specification, case $\nu = 1$, defines a program π's structure to be *well-formed* (so called *JLS-well-formed*) iff the following holds:
There is an inheritance function inh_{wf} with the two properties:
$I_1)$ $\quad inh_{wf}(K) = bind^1_{inh_{wf}}(ext(K)\ in\ decl(K)) \in \mathcal{C}^O$
is valid for all $K \in \mathcal{C}$;
$I_2)$ the induced dependency relation $dep_{bind^1_{inh_{wf}}}$ (defined in J_2 of Definition 2) has no cycles. □

Lemma 24: In a JLS-well-formed program the structure inh_{wf} is uniquely determined and is equal to the least fixed point inh^1_0 of $bdfl'^1$ and is equal to the result of algorithm LSWA^1 applied to the program structure.

Proof: Uses similar ideas of proof of characterization in Corollary 22 (see Theorem 32 and Remark 33 in [LSW09]). ■

Theorem 25: A program structure is JLS-well-formed iff it is binding well-formed w.r.t. the concrete binding function $bindfn = bind^1_{inh^1_0}$.

Proof: Let binding well-formedness w.r.t $bind^1_{inh^1_0}$ be given. Then due to Corollary 22 inh^1_0 is a state with $dom_{inh^1_0} = \mathcal{C}$. With $inh_{wf} \overset{df}{=} inh^1_0$ I_1) and I_2) are immediate implications of J_1) and J_2).
Let JLS-well-formedness be given. Then I_1) implies J_1), Remark 21 implies J_2), Remark 19 implies J_3). ■

Let us repeat: All these different inheritance and binding functions inh^ν_0 and $bind^\nu_{inh^\nu_0}$, induced by $\mathcal{A}^{d\nu}, \mathcal{A}^{i\nu}$ in well-formed programs, satisfy the sanity conditions (1) to (7) of [IP02]. The well-formedness conditions J_1), J_2) and J_3) are *stronger* than the sanity conditions (reviewers have overlooked this fact in an earlier version of this article).

4.3. Every binding function $bind^\nu_{inh^\nu_0}$ is a BIPET-model

Theorem 4 is formulated already in Section 3. Here is the

Proof of Theorem 4: We begin with a

> **Remark 26**: Rule I. (BET-Object) works restrictively compared to official Java with inner classes. Satisfaction of Rule I. requires that standard class *Object* is the only class named `Object`. I.e. there is no user declared class allowed to be named `Object`. It is agreeing with official Java with inner classes to drop Rule I. and to subsume it under Rule II. (BET-InCT). But at this moment we procede as Igarashi and Pierce do, until we repair their calculus in Subsection 5.2 and Section 6. □

I. (BET-Object)
Claim: $bind^\nu_{inh^\nu_0}(\texttt{Object}\ in\ P) = Object.$

There is an `Object`-admissible path from P via $Root$ to $Root.$`Object` = $Object$ because $de^{\star} \subseteq \mathcal{A}^{d\nu}$. Only the "least"-condition might be violated. But as there are no node repetitions (see Remark 20) only finitely many words in $\mathcal{A}^{d\nu}$ are involved and so there is also a least `Object`-admissible path from P via $Root$ to $Object$ w.r.t. $\mathcal{A}^{d\nu}$.

II. (BET-InCT)
Let P have the direct inner class named C, i.e. $P.C \in \mathcal{C}^O$.
Claim: $bind^{\nu}_{inh^{\nu}_0}(C\ in\ P) = P.C$.
λ is the least word in $\mathcal{A}^{d\nu}$. So we have the λ-associated path from P via P to $P.C$ and this path is least C-admissible w.r.t. $\mathcal{A}^{d\nu}$.

III. (BET-SimpEncl)
Let ($\star$) $bind^{\nu}_{inh^{\nu}_0}(D\ in\ P) = T$,
($\star\star$) there be no direct inner class $\in \mathcal{C}^{RO}$ named D in $P.C$ $\in \mathcal{C}$, i.e. $(P.C).D$ is undefined for the simple type D,
($\star\star\star$) $bind^{\nu}_{inh^{\nu}_0}(ext(P.C).D\ in\ P)$ is undefined.
Claim: $bind^{\nu}_{inh^{\nu}_0}(D\ in\ P.C) = T$.
Due to ($\star$) there is a least D-admissible path from P via P_1 to $P_1.D = T$ w.r.t. $\mathcal{A}^{d\nu}$. So there is a D-admissible path from $P.C$ via P and P_1 to $P_1.D = T$. The path from $P.C$ via P to P_1 is in $\mathcal{A}^{d\nu}$ because $\mathcal{A}^{d\nu}$ is de-closed. Claim: This path from $P.C$ via P and P_1 to T is least w.r.t. $\mathcal{A}^{d\nu}$.

The very least word $\lambda \in \mathcal{A}^{d\nu}$ does not lead to any D-admissible path from $P.C$ via $P_1 = P.C$ to $P_1.D = (P.C).D = T$ because of ($\star\star$).

Assume we had a least D-admissible path from $P.C$ via $\tilde{P}$ to $\tilde{P}.D$ which is beginning with $inh^{\nu}_0(P.C) = P'$. Then

$$P' = bind^{\nu}_{inh^{\nu}_0}(ext(P.C)\ in\ decl(P.C))$$

due to $\mathcal{C} = dom_{inh^{\nu}_0}$ and we had a least $ext(P.C)$-admissible path from $decl(P.C) = P$ to P'. Then we had a least $ext(P.C).D$-admissible path from P via P' and $\tilde{P}$ to $\tilde{P}.D$ (because of $w^\frown in \in \mathcal{A}^{d\nu} \Rightarrow w \in \mathcal{A}^{i\nu}$) what would mean

$$\tilde{P}.D = bind^{\nu}_{inh^{\nu}_0}(ext(P.C).D\ in\ P)$$

what is impossible due to ($\star\star\star$). So the least D-admissible path from $P.C$ starts with $decl(P.C) = P$ and is that one considered above.

> **Remark 27**: Please realize that the premise $P.C$ is different from $Object$ or $P.C \in \mathcal{C}$ has not been employed. Hence all binding functions satisfy the stronger Rule III without this premise, i.e. with $P.C \in \mathcal{C}^O$. □

IV. (BET-SimpSup)
Let there be no direct inner class named D in $P.C \in \mathcal{C}$ (i.e. $(P.C).D$ is undefined),
$bind^{\nu}_{inh^{\nu}_0}(ext(P.C).D\ in\ P) = T$.
Claim: $bind^{\nu}_{inh^{\nu}_0}(D\ in\ P.C) = T$.
There is a least $ext(P.C).D$-admissible path from P via T' and P' to $P'.D = T$, where the prefix path from P to T' is least $ext(P.C)$-admissible and the postfix path from T' via P' to $P'.D = T$ is least

D-admissible w.r.t. $\mathcal{A}^{i\nu}$. As $P = decl(P.C)$ we have

$$T' = bind^{\nu}_{inh^{\nu}_0}(ext(P.C)\ in\ P) = inh^{\nu}_0(P.C).$$

Claim: The path from $P.C$ via T' and P' to $P'.D = T$ is least D-admissible.
Namely the only "less" path would be the one from $P.C$ via $P.C$ to $(P.C).D$, but that is impossible.

V. (BET-Long)
Let $bind^{\nu}_{inh^{\nu}_0}(X\ in\ P) = T$,
class T have a direct inner class named C, i.e. $T.C \in \mathcal{C}^{RO}$.
Claim: $bind^{\nu}_{inh^{\nu}_0}(X.C\ in\ P) = T.C$.
There is a least X-admissible path from P to T.
Claim: If we prolong this path to $T.C$ then we have a least $X.C$-admissible path from P via T to $T.C$.
This is true because $\lambda \in \mathcal{A}^{i\nu}$.

VI. (BET-LongSup)
Let $(\star)\ bind^{\nu}_{inh^{\nu}_0}(X\ in\ P) = P'.D$,
$(\star\star)$ class $P'.D \in \mathcal{C}$ have no direct inner class named C, i.e.
$(P'.D).C$ is undefined,
$(\star\star\star)\ bind^{\nu}_{inh^{\nu}_0}(ext(P'.D).C\ in\ P') = U$.
Claim: $bind^{\nu}_{inh^{\nu}_0}(X.C\ in\ P) = U$.
There is a least X-admissible path from P via P' to $P'.D$ where D is the rightmost simple type in X $(\star)$. There is a least $ext(P'.D).C$-admissible path from P' via T' to U where there is a prefixing least $ext(P'.D)$-admissible path from P to T' $(\star\star\star)$. Since $P' = decl(P'.D)$ we have $inh^{\nu}_0(P'.D) = T'$ due to $dom^{RO}_{inh^{\nu}_0} = \mathcal{C}^{RO}$.
Claim: The path from P via P' and $P'.D$ and T' to U is least $X.C$-admissible, i.e. the path from $P'.D$ via T' to U is least C-admissible w.r.t. $\mathcal{A}^{i\nu}$.
The path from T' to U is least C-admissible w.r.t. $\mathcal{A}^{i\nu}$ $(\star\star\star)$. So the path from $P'.D$ via T' to U is also least C-admissible because $\mathcal{A}^{i\nu}$ is in-closed and the second premise $(\star\star)$ is holding. ∎

We are now in a position to refute a claim of reviewers of a former version of this article. Their claim: Igarashi's and Pierce's sanity conditions in [IP02] are sufficient means to determine which definition of binding resp. inheritance is the appropriate one. Our refutation: Program Examples 5 and 6 (π_a and π_b) are well-formed in at least two different senses (different definitions of binding) and, due to Theorem 4, have different models, namely $bind^1_{inh^1_0}$ and $bind^2_{inh^2_0}$ with different bindings $bind^1_{inh^1_0}(\mathtt{E}\ in\ D) = B.\mathtt{E} \neq bind^2_{inh^2_0}(\mathtt{E}\ in\ D) = A.\mathtt{E}$. I&P's sanity conditions are implications of well-formedness in every sense and so cannot determine which binding is right.

5. The dilemma with BIPET's Rule III. (BET-SimpEncl) in combination with Rule VI. (BET-LongSup)

5.1. General existence of different minimal models

For a given index ν and a well-formed Java-program π's structure the binding function $bind^{\nu}_{inh^{\nu}_0}$ might be infinite. $bind^{\nu}_{inh^{\nu}_0}$ is a BIPET-model (Theorem 4), but might not be minimal. A proof of existence of a minimal submodel would be trivial if $bind^{\nu}_{inh^{\nu}_0}$ were finite. So we have to procede with a little care towards a minimal submodel.

We consider those restricted subfunctions $restrsubbdfn$ of $bind^{\nu}_{inh^{\nu}_0}$ the first arguments of which are no longer than the maximum M of

$$length(ext(K)) + 1$$

for all $K \in \mathcal{C}$. We look at those finitely many $restrsubbdfn$ which satisfy the equations

$$restrsubbdfn(ext(K)|^i \; in \; decl(K)) = bind^{\nu}_{inh^{\nu}_0}((ext(K)|^i \; in \; decl(K)),$$

$K \in \mathcal{C}, 1 \leq i \leq length(ext(K))$, and fulfill the Rules I'. to VI'., where the rules I'. to IV'. are the same as I. to IV. and the Rules V'. and VI'. have got the additional premise

$$1 \leq length(X) \leq M - 1.$$

Consequence: A minimal $restrsubbdfn^{min}$ exists and can be found effectively in finitely many steps.

Now we apply the Rules V. and VI. for $length(X) \geq M$ and derive all triples $(X.C, P, T)$, C a simple type $\in \mathcal{SCT}$, from the "axioms", namely all triples in $restrsubbdfn^{min}$; each triple $(X.C, P, T)$ is derived in finitely many steps. The resulting relation is a single valued function due to Theorem 4, it is a minimal submodel $subbdfn^{min}$ of $bind^{\nu}_{inh^{\nu}_0}$ which satisfies all Rules I. to VI. and fullfills the equations

$$subbdfn^{min}(ext(K)|^i \; in \; decl(K)) = bind^{\nu}_{inh^{\nu}_0}((ext(K)|^i \; in \; decl(K)),$$

$K \in \mathcal{C}, 1 \leq i \leq length(ext(K))$. So we have:

Theorem 28: Every model $bind^{\nu}_{inh^{\nu}_0}$, $1 \leq \nu \leq \infty$, contains a minimal submodel.

Program Example 5 shows that $bind^{1}_{inh^{1}_0}$ and $bind^{2}_{inh^{2}_0}$ have two different minimal submodels. Hence satisfaction of IPET resp. BIPET and minimality is no sufficient prescription to prefer a specific model as the most appropriate one. Especially Java's official binding function $bind^{1}_{inh^{1}_0}$ is not a result of I&P's election process with the help of their calculus IPET. We have different minimal models even if all inheritances in a program coincide w.r.t. these different models. Program Example 6 and the following Subsection 5.2 show this.

5.2. Extension of Java's official binding function $bind^{1}_{inh^{1}_0}$ which is the least model of a preliminarily repaired calculus BIPET´

In Subsections 4.3 and 5.1 we have taken into consideration that Igarashi and Pierce work with Java-programs where no user declared class is named `Object`; only the standard class $Object$ is named so, see Remark 26. Subsections 4.3 and 5.1 show that there is an infinite family of minimal models of IPET resp. BIPET, namely minimal submodels of $bind^{\nu}_{inh^{\nu}_0}$, $1 \leq \nu \leq \infty$.

Question 29: Can we drop Igarashi's and Pierce's language restriction and modify BIPET towards a new calculus BIPET´such that Java's official binding function $bind^{1}_{inh^{1}_{0}}$ turns out to be the (exactly one) least model of BIPET´?

For an answer we extend every Java-program by a third standard class (beside $Root$ and $Object$), the so called *finite failure class Fc*:

$$\mathcal{C}^{ROF} \stackrel{df}{=} \mathcal{C}^{RO} \cup \{Fc\}$$

and we extend $\mathcal{CT}$ and $\mathcal{SCT}$ by the so called *finite failure type Ft*:

$$\mathcal{CT}^{F} \stackrel{df}{=} \mathcal{CT} \cup \{Ft\}, \quad \mathcal{SCT}^{F} \stackrel{df}{=} \mathcal{SCT} \cup \{Ft\}.$$

Then we extend every partially defined function $bd^{\iota\nu}_{inh}$ resp. $bind^{\nu}_{inh}$: In case an old application

$bd^{\iota\nu}_{inh}(C\ in\ P)$ resp. $bind^{\nu}_{inh}(X\ in\ P)$

is undefined the new application

($\star$) $bd^{\iota\nu}_{inh}(C\ in\ P)$ resp. $bind^{\nu}_{inh}(X\ in\ P)$

is allowed to yield Fc as its result for certain arguments $C \in \mathcal{SCT}^{F}, X \in \mathcal{CT}^{F}, P \in \mathcal{C}^{ROF}$. We denote the extended functions ($\star$) by the same designators $bd^{\iota\nu}_{inh}$ resp. $bind^{\nu}_{inh}$ as the unextended ones; the circumstances will indicate implicitly which functions are meant. inh is a variable for any partially defined inheritance function as before in Sections 3 and 4.

Firstly we define:

Definition 30: $bd^{\iota\nu}_{inh} : \mathcal{SCT}^{F} \times \mathcal{C}^{ROF} \stackrel{part}{\longrightarrow} \mathcal{C}^{ROF}$

$$bd^{\iota\nu}_{inh}(C\ in\ P) \stackrel{df}{=} \begin{cases} Fc & \text{if } C = Ft \text{ or } P = Fc \\ bd^{\iota\nu}_{inh}(C\ in\ P) & \text{otherwise if the old } bd^{\iota\nu}_{inh}(C\ in\ P) \\ & \text{is defined to be } \in \mathcal{C}^{O} \\ Fc & \text{otherwise if there are only finitely many words} \\ & v \in \mathcal{A}^{\iota\nu} \text{ with } v\text{-associated paths from } P \in dom^{RO}_{inh} \\ & \text{fully located in } dom^{RO}_{inh}, v(P) \in dom^{RO}_{inh} \\ \text{undefined} & \text{otherwise} \end{cases} \quad \square$$

Secondly we define:

Definition 31: $bind^{\nu}_{inh} : \mathcal{CT}^{F} \times \mathcal{C}^{ROF} \stackrel{part}{\longrightarrow} \mathcal{C}^{ROF}$

$$bind^{\nu}_{inh}(X\ in\ P) \stackrel{df}{=} \begin{cases} Fc & \text{if } X = Ft \text{ or } P = Fc \\ bd^{d\nu}_{inh}(X\ in\ P) & \text{otherwise if } length(X) = 1 \text{ and } bd^{d\nu}_{inh}(X\ in\ P) \\ & \text{is defined } \in \mathcal{C}^{OF} \\ bd^{i\nu}_{inh}(C\ in\ P') & \text{otherwise if } X = X'.C,\ length(X') \geq 1, \\ & length(C) = 1,\ P' = bind^{\nu}_{inh}(X'\ in\ P) \text{ is defined} \\ & \in \mathcal{C}^{OF} \text{ and } bd^{i\nu}_{inh}(C\ in\ P') \text{ is defined } \in \mathcal{C}^{OF} \\ \text{undefined} & \text{otherwise} \end{cases} \quad \square$$

We formulate the rules of calculus BIPET´ in Table 3. BIPET´ seems to be long. The calculus can be condensed, but this needs some preparations, see Table 4 in Section 6. The longer version is better for didactical reasons, for a good understanding of the necessary proofs. We drop Igarashi's and Pierce's language restriction and prove the following Theorem 32:

Table 3. Rules of calculus BIPET'

0.1. (BET'-Fc1)	$bindfn(X \text{ in } Fc) = Fc$
0.2. (BET'-Fc2)	$bindfn(Fn \text{ in } P) = Fc$
0.3. (BET'-Fc3)	$bindfn(Fn \text{ in } Fc) = Fc$
II. (BET' - InCT)	$\dfrac{\text{class } P \text{ has a direct inner class } \in \mathcal{C}^O \text{ named } C}{bindfn(C \text{ in } P) = P.C}$
II.2. (BET' - InCT2)	$\dfrac{\text{class } Root \text{ has no direct inner class } \in \mathcal{C}^O \text{ named } C}{bindfn(C \text{ in } Root) = Fc}$
III. (BET'-SimpEncl)	$\dfrac{bindfn(D \text{ in } P) = T \quad \text{class } P.C \in \mathcal{C}^O \text{ has no direct inner class named } D \quad bindfn(ext(P.C).D \text{ in } P) = Fc}{bindfn(D \text{ in } P.C) = T}$
III.2. (BET'-SimpEncl2)	$\dfrac{bindfn(D \text{ in } P) = Fc \quad \text{class } P.C \in \mathcal{C}^O \text{ has no direct inner class named } D \quad bindfn(ext(P.C).D \text{ in } P) = Fc}{bindfn(D \text{ in } P.C) = Fc}$
IV. (BET'-SimpSup)	$\dfrac{\text{class } P.C \in \mathcal{C}^O \text{ has no direct inner class named } D \quad bindfn(ext(P.C).D \text{ in } P) = T}{bindfn(D \text{ in } P.C) = T}$
V. (BET'-Long)	$\dfrac{bindfn(X \text{ in } P) = T \quad \text{class } T \text{ has a direct inner class } \in \mathcal{C}^O \text{ named } C}{bindfn(X.C \text{ in } P) = T.C}$
V.2. (BET'-Long2)	$\dfrac{bindfn(X \text{ in } P) = Fc}{bindfn(X.C \text{ in } P) = Fc}$
VI. (BET'-LongSup)	$\dfrac{bindfn(X \text{ in } P) = P'.D \quad \text{class } P'.D \in \mathcal{C}^O \text{ has no direct inner class named } C \quad bindfn(ext(P'.D).C \text{ in } P') = U}{bindfn(X.C \text{ in } P) = U}$
VI.2. (BET'-LongSup2)	$\dfrac{bindfn(X \text{ in } P) = P'.D \quad \text{class } P'.D \in \mathcal{C}^O \text{ has no direct inner class named } C \quad bindfn(ext(P'.D).C \text{ in } P') = Fc}{bindfn(X.C \text{ in } P) = Fc}$

- Variables P, P' range over $\mathcal{C}^{RO}$, T, U over $\mathcal{C}^O$, X over $\mathcal{CT}$ (types) and C, D over $\mathcal{SCT}$ (simple types).

Theorem 32: Let a program be well-formed w.r.t. $bind^1_{inh^1_0}$ (extended or not, both views amount to the same notion of well-formedness). Then the extended single valued function $bind^1_{inh^1_0}$ is satisfying all twelve rules of BIPET´, i.e. it is a model of BIPET´ (see Table 3).

Remark 33: We are trying to prove Theorem 32 for all functions $bind^\nu_{inh^\nu_0}, 1 \leq \nu \leq \infty$, in order to find out which rules are not satisfied by all functions. □

Proof: 0.1. (BET'-Fc1)
0.2. (BET'-Fc2)
0.3. (BET'-Fc3)
These rules are axioms and hold because, due to Definition 31, $bind^{\nu}_{inh^{\nu}_0}$ is strict in Ft, Fc.

II. (BET'-InCT)
See the corresponding place in the proof of Theorem 4

II.2. (BET'-InCT2)
Let class $Root$ have no direct inner class $\in \mathcal{C}^O$ named C , i.e. $Root.C$ is undefined.
Claim: $bind^{\nu}_{inh^{\nu}_0}(C\ in\ Root) = Fc$.
There is only one path from $Root$ associated to $\mathcal{A}^{d\nu}$ inside $\mathcal{C}^{RO}$, namely that one associated to $\lambda \in \mathcal{A}^{d\nu}$. As $Root.C$ is undefined $bd^{d\nu}_{inh^{\nu}_0}(C\ in\ Root)$ is defined to be Fc and so is $bind^{\nu}_{inh^{\nu}_0}(C\ in\ Root)$.

III. (BET'-SimpEncl)
See the proof of Theorem 4. The premise "class $P.C$ is different from $Object$" of Rule III. of BIPET is not exploited in that proof. So the changed premise $P.C \in \mathcal{C}^O$ is allowed in BIPET´. See Remark 27.

III.2. (BET'-SimpEncl2)
Let ($\star$) $bind^{\nu}_{inh^{\nu}_0}(D\ in\ P) = Fc$,
($\star\star$) there be no direct inner class named D in
$P.C \in \mathcal{C}^O$ (equivalent: $P.C \in \mathcal{C}^{RO}$),
($\star\star\star$) $bind^{\nu}_{inh^{\nu}_0}(ext(P.C).D\ in\ P) = Fc$.
Claim: $bind^{\nu}_{inh^{\nu}_0}(D\ in\ P.C) = Fc$.
Due to well-formedness there are only finitely many $\mathcal{A}^{d\nu}$-associated paths from $P.C$ inside $\mathcal{C}^{RO}$. Assume the claim were wrong, i.e. there were a least D-admissible path from $P.C$ via P_1 to $T \in \mathcal{C}^O$ with $P_1.D = T$. Due to ($\star\star$) P_1 is different from $P.C$. So the path starts with $inh^{\nu}_0(P.C) = P'$ or $decl(P.C) = P$. The latter case is impossible due to ($\star$). So we have

$$P' = bind^{\nu}_{inh^{\nu}_0}(ext(P.C)\ in\ P)$$

due to $\mathcal{C} = dom_{inh^{\nu}_0}$. Because the $\mathcal{A}^{d\nu}$ -associated path from $P.C$ via P' to P_1 starts with inh^{ν}_0 the path is necessarily also an $\mathcal{A}^{i\nu}$-associated path and so is the path from P' to P_1. But then we had

$$bind^{\nu}_{inh^{\nu}_0}(ext(P.C).D\ in\ P) = T$$

what is contradicting ($\star\star\star$). So the assumption is wrong, the claim is holding.

IV. (BET'-SimpSup)
See the corresponding place in the proof of Theorem 4. The premises $P.C \in \mathcal{C}^O$ and $P.C \in \mathcal{C}$ are equivalent here.

V. (BET'-Long)
See Proof of Theorem 4.

V.2. (BET'-LongSup2)
This is an immediate consequence of Definition 31 of $bind^{\nu}_{inh^{\nu}_0}$.

VI. (BET'-LongSup)
See the proof of Theorem 4. The premises $P'.D \in \mathcal{C}^O$ and $P'.D \in \mathcal{C}$ are equivalent here.

VI.2. (BET'-LongSup2)
Let (⋆) $bind^{\nu}_{inh^{\nu}_0}(X\ in\ P) = P'.D$,
(⋆⋆) class $P'.D \in \mathcal{C}^O$ have no direct inner class named C,
(⋆⋆⋆) $bind^{\nu}_{inh^{\nu}_0}(ext(P'.C).D\ in\ P') = Fc$.
Claim: $bind^{\nu}_{inh^{\nu}_0}(X.C\ in\ P) = Fc$.
Due to well-formedness there are only finitely many $\mathcal{A}^{d\nu}$-, $\mathcal{A}^{i\nu}$-associated paths from any class in $\mathcal{C}^{RO}$. Assume the claim were wrong, i.e. there were a least $X.C$-admissible path from P via P' to $P'.D$ and the postfixing path from $P'.D$ via T^0 to U is least C-admissible w.r.t. $\mathcal{A}^{i\nu}$. Because of premise (⋆⋆) this postfixing path is starting with $inh^{\nu}_0(P'.D) = T'$ or $decl(P'.D) = P'$. We have due to wellformedness

$$inh^{\nu}_0(P'.D) = bind^{\nu}_{inh^{\nu}_0}(ext(P'.C)\ in\ P') = T'.$$

$inh^{\nu}_0(P'.D) = T'$ is impossible due to in-closedness of $\mathcal{A}^{i\nu}$ and premise(⋆⋆⋆). In case $\nu = 1$ $decl(P'.D) = P'$ is also impossible because $\mathcal{A}^{i1}$ is equal $in^{\star}$ and the postfixing path cannot start with $decl$. ■

Remark 34: In case $2 \leq \nu \leq \infty$ this last impossibility cannot be verified. Namely we have a disproof of Rule VI.2. (BET'-LongSup2) inside program Example 5. All three premises are fulfilled for $2 \leq \nu \leq \infty$:
(⋆) $bind^{\nu}_{inh^{\nu}_0}(\mathtt{F}\ in\ D) = D.\mathtt{F} = F$,
(⋆⋆) class F has no direct inner class named `D`,
(⋆⋆⋆) $bind^{\nu}_{inh^{\nu}_0}(ext(F).\mathtt{D}\ in\ D)$
$= bind^{\nu}_{inh^{\nu}_0}(\mathtt{E.D}\ in\ D)$
$= bd^{i\nu}_{inh^{\nu}_0}(\mathtt{D}\ in\ bd^{d\nu}_{inh^{\nu}_0}(\mathtt{E}\ in\ D))$
$= bd^{i\nu}_{inh^{\nu}_0}(\mathtt{D}\ in\ A.\mathtt{E})$
$= Fc.$
Nevertheless, the claim is wrong:

$$bind^{\nu}_{inh^{\nu}_0}(\mathtt{F.D}\ in\ D) = D.$$ □

Theorem 32 (which is a correctness proposition on $bind^{1}_{inh^{1}_0}$ w.r.t. calculus BIPET´) can be extended by a completeness proposition:

Theorem 35 (*on completeness*): In case index ν is 1 and $dom_{inh^1_0} = \mathcal{C}$ every triple $(X, P, T) \in \mathcal{CT}^F \times \mathcal{C}^{ROF} \times \mathcal{C}^{ROF}$ with

$$bind^{1}_{inh^{1}_0}(X\ in\ P) = T$$

is the conclusion of a Rule of BIPET´such that all premises are satisfied.

Remark 36: We try to prove Theorem 35 for all indices ν. Eleven Rules 0.1. to V.2. and VI.2. work out. But Rule VI. is an obstacle in case $2 \leq \nu \leq \infty$. So only case $\nu = 1$ works out for all twelve Rules. □

Proof: **Case 0**: $X = Ft$ or $P = Fc$.
One of the Rules 0.1. to 0.3. is applying.
From now on $X \in \mathcal{CT}$ and $P \in \mathcal{C}^{RO}$.
Case 1: $length(X) = 1$.
Case 1.1: $P.X \in \mathcal{C}^{RO}$.
Due to Theorem 32 the conclusion of Rule II.

$$bind^{\nu}_{inh^{\nu}_0}(X\ in\ P) = P.X$$

holds. Since $bind^{\nu}_{inh^{\nu}_0}$ is single valued we have $P.X = T$.
Case 1.2: $P.X$ is undefined and $P = Root$.
Due to Theorem 32 the conclusion of Rule II.2.

$$bind^{\nu}_{inh^{\nu}_0}(X\ in\ Root) = Fc$$

holds. Single valuedness yields $Fc = T$.
Case 1.3: $P.X$ is undefined and $P \in \mathcal{C}^{O}$.
Then $P = \overline{P}.C$ for an appropriate simple type C and $(\overline{P}.C).X$ is undefined ($\star\star$).
Case 1.3.1: $bind^{\nu}_{inh^{\nu}_0}(ext(\overline{P}.C).X\ in\ \overline{P}) = \overline{T} \in \mathcal{C}^{O}$.
Due to Theorem 32

$$bind^{\nu}_{inh^{\nu}_0}(X\ in\ \overline{P}.C) = \overline{T}$$

holds as the conclusion of Rule IV. Because $bind^{\nu}_{inh^{\nu}_0}$ is single valued we have $\overline{T} = T$.
Case 1.3.2: $bind^{\nu}_{inh^{\nu}_0}(ext(\overline{P}.C).X\ in\ \overline{P})$ is Fc ($\star\star\star$).
Case 1.3.2.1: $ext(\overline{P}.C)$ is undefined.
Then $\overline{P}.C = Object, \overline{P} = Root,\ C =$ `Object`. Due to definition of $bind^{\nu}_{inh^{\nu}_0}$ we have

$$T = bind^{\nu}_{inh^{\nu}_0}(X\ in\ P) = bind^{\nu}_{inh^{\nu}_0}(X\ in\ \overline{P})\ (\star),$$

let $T \in \mathcal{C}^{O}$ or $T = Fc$. Due to Theorem 32 and ($\star$), ($\star\star$), ($\star\star\star$) we have as the conclusion of Rule III. resp. III.2.

$$bind^{\nu}_{inh^{\nu}_0}(X\ in\ P) = T.$$

Case 1.3.2.2: $ext(\overline{P}.C)$ is defined $\in \mathcal{CT}$.
So $\overline{P}.C = P$ is user declared and different from $Object$. We have due to assumption $dom^{RO}_{inh^{\nu}_0} = \mathcal{C}^{RO}$ and state inh^{ν}_0

($\circ$) $\quad inh^{\nu}_0(\overline{P}.C) = bind^{\nu}_{inh^{\nu}_0}(ext(\overline{P}.C)\ in\ \overline{P}) = \tilde{T} \in \mathcal{C}^{O}$.

So there is no least X-admissible path from $\tilde{T}$ to any $\hat{T}$ such that $\hat{T}.X$ is defined $\in \mathcal{C}^{O}$ w.r.t. $\mathcal{A}^{i\nu}$. As we want to derive

$$bind^{\nu}_{inh^{\nu}_0}(X\ in\ P) = T$$

with $T \in \mathcal{C}^{OF}$ by the help of Rule III. or Rule III.2. we ask: What is the result of

$$bind^{\nu}_{inh^{\nu}_0}(X\ in\ \overline{P})?$$

We have

$$bind^{\nu}_{inh^{\nu}_0}(X\ in\ \overline{P}.C) = T.$$

Case 1.3.2.2.1: Let $T \in \mathcal{C}^{O}$. Since ($\circ$) and $(\overline{P}.C).X$ is undefined the least X-admissible path from $\overline{P}.C$ via a $\overline{T}$ to $\overline{T}.X = T$ starts with $decl(\overline{P}.C) = \overline{P}$ or with $inh^{\nu}_0(\overline{P}.C) = \tilde{T}$. In the latter case the path is not only associated to $\mathcal{A}^{d\nu}$ but also to $\mathcal{A}^{i\nu}$ which contradicts our statement above. So we conclude

$$bind^{\nu}_{inh^{\nu}_0}(X\ in\ \overline{P}) = T\ \ (\star^1).$$

Due to Theorem 32 and ($\star^1$), ($\star\star$), ($\star\star\star$) we have as the conclusion of Rule III.

$$bind^{\nu}_{inh^{\nu}_0}(X\ in\ \overline{P}.C) = T$$
what is to be shown due to $\overline{P}.C = P$.
Case 1.3.2.2.2.: Let $T = Fc$.
Claim: $bind^{\nu}_{inh^{\nu}_0}(X\ in\ \overline{P})$ is Fc ($\star^3$).
Assume we had a least X-admissible path from $\overline{P}$ to $T^\star \in \mathcal{C}^O$ in $\mathcal{C}^{RO}$ (w.r.t. $\mathcal{A}^{d\nu}$), i.e. we had
$$bind^{\nu}_{inh^{\nu}_0}(X\ in\ \overline{P}) = T^\star \in \mathcal{C}^O \quad (\star^2)$$
then due to Theorem 32, Rule III. with $(\star^2), (\star\star), (\star\star\star)$, we conclude
$$bind^{\nu}_{inh^{\nu}_0}(X\ in\ \overline{P}.C) = T^\star.$$
But this contradicts
$$bind^{\nu}_{inh^{\nu}_0}(X\ in\ \overline{P}.C) = T = Fc.$$
So the claim above holds.
Again due to Theorem 32, Rule III.2. with $(\star^3), (\star\star), (\star\star\star)$, we conclude
$$bind^{\nu}_{inh^{\nu}_0}(X\ in\ \overline{P}.C) = Fc$$
what is to be shown due to $\overline{P}.C = P$ and $T = Fc$.
Case 2: $length(X) \geq 2, X = \overline{X}.C$.
We have
$$(\star) \qquad bind^{\nu}_{inh^{\nu}_0}(\overline{X}\ in\ P) = \overline{T} \in \mathcal{C}^{OF}.$$
Case 2.1: $\overline{T} = Fc$.
Then
$$bind^{\nu}_{inh^{\nu}_0}(\overline{X}.C\ in\ P) = Fc$$
and Rule V.2. applies.
Case 2.2: $\overline{T} \in \mathcal{C}^O$ ($\star^2$), i.e. $\overline{T} = P'.D$ for appropriate P' and D.
Case 2.2.1: $\overline{T}.C \in \mathcal{C}^O$.
Due to Theorem 32 we have as the conclusion of Rule V.
$$bind^{\nu}_{inh^{\nu}_0}(\overline{X}.C\ in\ P) = \overline{T}.C,$$
so we have also derived
$$bind^{\nu}_{inh^{\nu}_0}(X\ in\ P) = T$$
due to single valuedness of $bind^{\nu}_{inh^{\nu}_0}$.
Case 2.2.2: $\overline{T}.C$ is Fc and $T \in \mathcal{C}^O$ $\quad (\star\star^2)$.
There is a least C-admissible path from $\overline{T}$ via a $\tilde{T}$ to $\tilde{T}.C = T$ w.r.t. $\mathcal{A}^{i\nu}$. In case $\nu = 1$ we are sure $\overline{T}$ is not equal $Object$.
Otherwise T were $= Fc$. We have due to assumption $dom_{inh^1_0} = \mathcal{C}$ and inh^1_0 being a state
$$inh^1_0(P'.D) = bind^1_{inh^1_0}(ext(P'.D)\ in\ P') = \hat{T}.$$
What is $bd^{i1}_{inh^1_0}(C\ in\ \hat{T})$?
As $\overline{T}.C$ is undefined the least $\mathcal{A}^{i1}$-associated path from $\overline{T}$ via $\tilde{T}$ to $\tilde{T}.C = T$ starts with $inh^1_0(\overline{T}) = \hat{T}$ because, due to $\mathcal{A}^{i1} = in^\star$, we are sure it is an inheritance chain (for $\mathcal{A}^{i\nu}$ with $\nu \geq 2$ we are not sure!).
So the path from $\hat{T}$ via $\tilde{T}$ to $\tilde{T}.C = T$ is least C-admissible w.r.t. $\mathcal{A}^{i1} = in^\star$. So
$$bd^{i1}_{inh^1_0}(C\ in\ \hat{T}) = T.$$
So
$$(\star\star\star^2) \quad bind^1_{inh^1_0}(ext(P'.D).C\ in\ P') = T$$
and

$$bind^1_{inh^1_0}(X\ in\ P) = T$$

holds due to Theorem 32 as the conclusion of Rule VI. with valid premises $(\star), (\star^2), (\star\star^2), (\star\star\star^2)$.
Case 2.2.3: $\overline{T}.C$ is Fc and $T = Fc \quad (\star\star^3)$.
If $\overline{T}$ is $Object$ then P' is $Root$ and

$bind^\nu_{inh^\nu_0}(ext(P'.D).C\ in\ P')$ is
$bind^\nu_{inh^\nu_0}(Ft\ in\ Root)$ is $Fc \quad (\star\star\star^3)$

Rule VI.2. with premises $(\star), (\star^2), (\star\star^3), (\star\star\star^3)$ yields due to Theorem 32

$$bind^\nu_{inh^\nu_0}(\overline{X}.C\ in\ P) = Fc$$

what is to be shown.
Now let $\overline{T} \neq Object$. As earlier we have

$$inh^\nu_0(P'.D) = bind^\nu_{inh^\nu_0}(ext(P'.D)\ in\ P') = \hat{T} \in \mathcal{C}^O.$$

What is the result U of

$$bd^{i\nu}_{inh^\nu_0}(C\ in\ \hat{T})?$$

Is it Fc as we hope, as we want to apply Rule VI.2. ?
Claim: $bd^{i\nu}_{inh^\nu_0}(C\ in\ \hat{T}) = Fc$.

Assume we had a least C-admissible path from $\hat{T}$ to a result $U \in \mathcal{C}^O$ w.r.t. $\mathcal{A}^{i\nu}$. Then we have a least C-admissible path from $\overline{T} = P'.D$ to a result $U \in \mathcal{C}^O$ w.r.t. $\mathcal{A}^{i\nu}$ and

$bind^\nu_{inh^\nu_0}(\overline{X}.C\ in\ P)$ were $= U$.

Contradiction!
So the claim holds and Rule VI.2. applies with valid premises $(\star), (\star^2), (\star\star^3), (\star\star\star^4)$ where
$(\star\star\star^4) \quad bind^\nu_{inh^\nu_0}(ext(P'.D).C\ in\ P') = Fc.$
The conclusion is

$$bind^\nu_{inh^\nu_0}(\overline{X}.C\ in\ P) = Fc.$$ ■

Remark 37: Let us return to Case 2.2.2 in the proof of Theorem 32. In case $\nu \geq 2$ we would like to see a program example where our reasoning does not work, i.e. where the least C-admissible path w.r.t. $\mathcal{A}^{i\nu}$ from $\overline{T}$ via $\tilde{T}$ to $\tilde{T}.C = T$ does not start with $inh^\nu_0(\overline{T})$ but with $decl(\overline{T})$.
Look at program Example 5. The corresponding classes are

abstract in the proof	concrete in the Example
$\overline{T}$	F
$\tilde{T}$	B
$\tilde{T}.C = T$	$B.\mathtt{D} = D$

So the least $\mathtt{D}$-admissible path from F to D w.r.t. $\mathcal{A}^{i\nu}$

$$decl(F) = D,\ decl(D) = B,\ B.\mathtt{D} = D$$

starts with $decl(F)$ and not with $inh^\nu_0(F)$. □

Theorem 38: In case $\nu = 1$ and $dom_{inh^1_0} = \mathcal{C}$ every valid triple $(X, P, T) \in \mathcal{CT}^F \times \mathcal{C}^{ROF} \times \mathcal{C}^{ROF}$ with

$$bind^1_{inh^1_0}(X\ in\ P) = T$$

is derivable by BIPET´ with a finite derivation tree.

Proof: In this proof we drop the superscript 1 and write simply $bind_{inh_0}$ or even bfn for $bind^1_{inh^1_0}$.

Theorem 32 allows to construct a (possibly infinite) proof tree for every valid formula

$bfn(X\ in\ P) = T$
with $(X, P, T) \in \mathcal{CT}^F \times \mathcal{C}^{ROF} \times \mathcal{C}^{ROF}$ in case $dom_{inh_0} = \mathcal{C}$. T is necessarily $\in \mathcal{C}^{OF}$.

Claim: The proof tree is finite.
Due to König's Lemma it is sufficing to prove: There is no infinite path from the root node
$bfn(X\ in\ P) = T$.

If we assume the contrary then only the Rules III., III.2., IV., V.,V.2., VI.,VI.2. are applied in such a path. It is obvous that there is at least one occurring of a node of the kind
$bfn(ext(P).C\ in\ decl(P)) = U$ or $= Fc$
as a premise of a Rule III. or III.2. or IV. or V. or V.2. or VI. or VI.2., $P \in \mathcal{C},\ U \in \mathcal{C}^O,\ C \in \mathcal{SCT}$.

Let such a node be given. We are looking for a next node of this kind in the path upward. There are two possibilities:
First:

$bfn(ext(P))\ |^i\ in\ decl(P)) = T^i \quad bfn(ext(T^i).ext(P)_{i+1}\ in\ decl(T^i)) = T^{i+1}$ or $= Fc$
\ | Rules VI. or VI.2.
$bfn(ext(P)\ |^{i+1}\ in\ decl(P)) = T^{i+1}$ or $= Fc$
|
⋮ Rules V. or V.2. or VI. or VI.2. (1 or more times)
|
$bfn(ext(P).C\ in\ decl(P)) = U$ or $= Fc$

with $1 \leq i \leq length(ext(P))$. This implies the existence of an edge from P to T^i in the dependency relation dep_{bfn}.
Second:

$bfn(ext(decl^{\nu+1}(P)).ext(P)\ |^1\ in\ decl(decl^{\nu+1}(P))) = T^\nu$ or $= Fc$
| Rules IV. or III. or III.2.
$bfn(ext(P)\ |^1\ \ in\ decl^{\nu+1}(P)) = T^\nu$ or $= Fc$
|
⋮ Rules III. or III.2. (0 or more times)
|
$bfn(ext(P)\ |^1\ \ in\ decl(P)) = T^0$ or $= Fc$
|
⋮ Rules V. or V.2. or VI. or VI.2. (1 or more times)
|
$bfn(ext(P).C\ in\ decl(P)) = U$ or $= Fc$

with $\nu \geq 0$. This implies the existence of a chain of edges from
P via $decl(P)$ via ... to $decl^{\nu+1}(P)$
in the relation $decl$.

Because the path is infinitely long there is necessarily a cycle in the united relation

$$decl \cup dep_{bfn}$$

what is contradicting Remark 21. So every valid

$$bfn(X\ in\ P) = T$$

is derivable by BIPET´with a finite derivation tree. ■

Corollary 39: Let a program be well-formed w.r.t. $bind^1_{inh^1_0}$. Then the extended binding function $bind^1_{inh^1_0}$ is the least model of BIPET´(is even the only one model of BIPET´) and the unextended $bind^1_{inh^1_0}$ is a minimal model of a calculus BIPET^{sm}. BIPET^{sm} is defined as a slight modification of BIPET:

Rule I. is deleted from BIPET and
Rule III. has deleted premise
class $P.C$ is different from $Object$
from Rule III. of BIPET.

So BIPET^{sm} is BIPET´ minus its Rules 0.1., 0.2., 0.3., II.2., III.2., V.2., VI.2..

Continuation Example 6: Now we are in a position to prove the claim in Example 6, namely the existence of $bindfn^1$ which in program π_b is a minimal submodel of the unextended $bind^1_{inh^1_0}$ of calculus BIPET.

We restrict the extended total function ($dom_{inh^1_0} = \mathcal{C}$!)

$$bind^1_{inh^1_0} : \mathcal{CT}^F \times \mathcal{C}^{ROF} \xrightarrow{tot} \mathcal{C}^{ROF}$$

to the unextended partial function

$$bind^1_{inh^1_0} : \mathcal{CT} \times \mathcal{C}^{RO} \xrightarrow{part} \mathcal{C}^{RO},$$

obviously a minimal model of BIPET^{sm} because all triples are derivable in BIPET^{sm} from the leaf nodes $bind^1_{inh^1_0}(X\ in\ P) = FC$ or, more specifically, $bind^1_{inh^1_0}(ext(P.C).D\ in\ P) = Fc$. Now we delete from the function table of $bind^1_{inh^1_0}$ all those triples (X, P, T) the derivation trees of which apply Rule III. where the premise

class $P.C$ is not different from (is equal) $Object$

is needed. So the remaining collection of triples, a subfunction $bindfn^1$, is a minimal submodel of $bind^1_{inh^1_0}$ w.r.t. BIPET if we restrict Java in the sense of Igarashi and Pierce and do not allow `Object` as a name of a user declared class.

$bindfn^1$ is different from the unextended $bind^1_{inh^1_0}$ but not very different. Their induced inheritances are the same:

$$inh_{bindfn^1} = inh_{bind^1_{inh^1_0}}, \text{ namely} = inh^1_0.$$

$bindfn^1$ and $bind^1_{inh^1_0}$ differ at most for types X in the body of class $Object$:

$bindfn^1(\texttt{Object}\ in\ Object)$
is $Object$ equal $bind^1_{inh^1_0}(\texttt{Object}\ in\ Object)$

$bindfn^1(X\ in\ Object)$ is undefined

for all $X \in \mathcal{CT} \setminus \{\texttt{Object}\}$, whereas $bind^1_{inh^1_0}(X\ in\ Object)$ may be defined $\in \mathcal{C}^O$. □

So if the authors of [IP02] (due to the fact that IPET resp. BIPET cannot have a least model) would have liked to present $bindfn^1$ at least as a minimal model of BIPET then they must concede that the body of $Object$ cannot have applied occurrences of class types X different from `Object`. On the other hand: we see in the definition of class $Object$ in the official Java Language Specification [GJSB00, GJSB05] that there are indeed applied occurrences of simple class types in $Object$ referring to classes in the Java-utility package.

6. The repaired calculi BIPET´ and BIPET˝ and their equivalent recursive function definitions resp. recursive programs

BIPET´ in Table 3 is a proper calculus which allows to derive exactly those formulas

$$bindfn(X\ in\ P) = T$$

for which formula

$$bind^1_{inh^1_0}(X\ in\ P) = T$$

is valid in case of binding well-formedness of the Java-program, i.e. $dom_{inh^1_0} = \mathcal{C}$. Theory of *recursive function definitions* (resp. *recursive programs* in the sense of [LoSi84], see also [Man74, BaWo82]) allows to rewrite BIPET´as a recursive function definition because the runtime stack contents of a regularly terminating call $bindfn(X\ in\ P)$ with a result T represent the finite derivation tree of $bindfn(X\ in\ P) = T$ in a 1-1-manner. Precise treatment requires that all standard operations are total. So it is advised to extend the four standard operations $decl$, ext, . (dot as a selector), . (dot as a concatenator) in a strict manner:

$decl(Root)$	is	Fc	
$decl(Fc)$	is	Fc	
$ext(Root)$	is	Ft	
$ext(Object)$	is	Ft	
$ext(Fc)$	is	Ft	
$P.C$	is	Fc	for all $P \in \mathcal{C}^{RO}, C \in \mathcal{CT}$ where the original $P.C$ is undefined
$Fc.C$	is	Fc	for all $C \in \mathcal{CT}^F$
$P.Ft$	is	Fc	for all $P \in \mathcal{C}^{ROF}$
$X.Ft$	is	Ft	for all $X \in \mathcal{CT}^F$
$Ft.C$	is	Ft	for all $C \in \mathcal{SCT}^F$

The set of Boolean values $\mathbb{B} \stackrel{df}{=} \{true, false\}$ is not extended and so the Boolean standard operations $\neg$, $\wedge$, $\vee$ are not changed. The standard operations like $=$, $\in \mathcal{SCT}$, $\in \mathcal{C}^O$, **let** $=$ **in endlet**, **if then else fi** keep their naturally known meanings.

Before we write $bindfn$ as a recursive function we may write BIPET´in a condensed form in Table 4 because we may exploit the extension of standard operations:

Table 4. Rules of calculus BIPET´condensed

0. (BET'-Fc)	$\dfrac{\tilde{P} = Fc \text{ or } \tilde{X} = Ft}{bindfn(\tilde{X} \text{ in } \tilde{P}) = Fc}$
II. (BET' - InCT)	$\dfrac{P = Root \text{ and } P.C = Fc \text{ or } P.C \in \mathcal{C}^O}{bindfn(C \text{ in } P) = P.C}$
III. (BET'-SimpEncl)	$\dfrac{\begin{array}{l} bindfn(D \text{ in } P) = T \\ P.C \in \mathcal{C}^O\ ,\ P.C.D = Fc \\ bindfn(ext(P.C).D \text{ in } P) = Fc \end{array}}{bindfn(D \text{ in } P.C) = T}$
IV. (BET'-SimpSup)	$\dfrac{\begin{array}{l} P.C \in \mathcal{C}^O\ ,\ P.C.D = Fc \\ bindfn(ext(P.C).D \text{ in } P) = \tilde{T} \end{array}}{bindfn(D \text{ in } P.C) = \tilde{T}}$
V. (BET'-Long)	$\dfrac{\begin{array}{l} bindfn(X \text{ in } P) = T \\ T = Fc \text{ or } T.C \in \mathcal{C}^O \end{array}}{bindfn(X.C \text{ in } P) = T.C}$
VI. (BET'-LongSup)	$\dfrac{\begin{array}{l} bindfn(X \text{ in } P) = P'.D \\ P'.D \in \mathcal{C}^O\ ,\ P'.D.C = Fc \\ bindfn(ext(P'.D).C \text{ in } P') = U \end{array}}{bindfn(X.C \text{ in } P) = U}$

Variables P, P' range over $\mathcal{C}^{RO}$, $\tilde{P}$ over $\mathcal{C}^{ROF}$,T, U over $\mathcal{C}^{OF}$, $\tilde{T}$ over $\mathcal{C}^O$, X over $\mathcal{CT}$, $\tilde{X}$ over $\mathcal{CT}^F$ and C, D over simple types $\mathcal{SCT}$.

Non-Boolean standard operations are extended towards total operations, Boolean standard operations remain total.

Definition 40:	Recursive definition of the function $bindfn$ is programmed below.
Rules	Code

```
                      bindfn(X in P) =
                      if P = Fc ∨ X = Ft
Rule 0.1.-0.3.        then Fc
                      else if X ∈ SCT
                         then if P.X ∈ C^O ∨ P = Root ∧ P.X = Fc
Rule II.,II.2.              then P.X
                            else let T = bindfn(ext(P).X in decl(P))
                               in if T ∈ C^O
Rule IV.                          then T
Rule III.,III.2.                  else bindfn(X in decl(P))
                                  fi
                               endlet
                            fi
                         else let X = X'.C, C ∈ SCT
                               T' = bindfn(X' in P)
                            in if T'.C ∈ C^O ∨ T' = Fc
Rule V.,V.2.                  then T'.C
Rule VI.,VI.2.                else bindfn(ext(T').C in decl(T'))
                              fi
                            endlet
                         fi
                      fi
                      end bindfn                                    □
```

See how elegantly the twelve rules of BIPET´ are mirrored in this recursive function definition.

Theorem 41: The recursive function $bindfn$ is an algorithm which determines the associated class (declaration occurrence) $bind^1_{inh^1_0}(X\ in\ P) \in \mathcal{C}^{OF}$ for every type $X \in \mathcal{CT}^F$ directly occurring in the body of class (declaration occurrence) $P \in \mathcal{C}^{ROF}$ in a given binding well-formed Java-program with $dom_{inh^1_0} = \mathcal{C}$. Especially for every user declared class $P \in \mathcal{C}$ a call of $bindfn(ext(P)\ in\ decl(P))$ terminates regularly with the result $inh^1_0(P) \in \mathcal{C}^O$.

Proof: To Definition 40 of $bindfn$ there is associated a formal execution or call tree [Lan73, Old81] generated by copy rule expansion of the body of $bindfn$, compare also [Man74, BaWo82, LoSi84]. Every computation of $bindfn$ applied to arguments $\tilde{X}$ and $\tilde{P}$ has a computation path through the tree from left to right which starts at the root and either finishes successfully at the root with a result $T \in \mathcal{C}^{OF}$ or is not successful and infinitely long and ascends an infinite copy rule applications path of the tree. There is no non-successful finite computation because all standard operators are interpreted as total operations.

A successful computation describes a finite initial tree of the whole formal execution or call tree (the **then**- resp. **else**-branches which are not entered are simply deleted) and represents in a 1-1-manner the BIPET´-derivation tree of a valid formula

$$bind^1_{inh^1_0}(\tilde{X}\ in\ \tilde{P}) = T, \tilde{X} \in \mathcal{CT}^F, \tilde{P} \in \mathcal{C}^{ROF}, T \in \mathcal{C}^{OF}$$

and vice versa. As we have assumed well-formedness of the considered Java-program there are no non-successful computations of $bindfn$. ■

Let us apply our recursive function $bindfn$ to a syntactically correct, but not binding well-formed Java-program so that $dom_{inh_0^1} \neq \mathcal{C}$. Again for $P \in dom_{inh_0^1}$ a call of $bindfn(ext(P)\ in\ decl(P))$ terminates regularly with the result $inh_0^1(P) \in dom^O_{inh_0^1}$.

As we would like to have this recursive function $bindfn$ as a semideciding algorithm $bindfn(ext(P)\ in\ decl(P))$ should either terminate regularly with a result Fc or run infinitely long for at least one user declared class $P \in \mathcal{C} \setminus dom_{inh_0^1}$. But this conjecture is not true as the following program Example 42 demonstrates:

Example 42:

π_{nwf}: **class** `A` **extends** `B` { }
class `B` **extends** `A` { }
class `C` **extends** `Object` { }

inh_0^1 is the function $\{\langle C, Object\rangle\}$, so $dom_{inh_0^1} = \{C\} \overset{\subset}{\neq} \mathcal{C} = \{A, B, C\}$,
$bindfn(ext(A)\ in\ decl(A)) = bindfn(\texttt{B}\ in\ Root) = B \in \mathcal{C}$,
$bindfn(ext(B)\ in\ decl(B)) = bindfn(\texttt{A}\ in\ Root) = A \in \mathcal{C}$.
$bindfn(ext(C)\ in\ decl(C)) = bindfn(\texttt{Object}\ in\ Root) = Object \in \mathcal{C}^O$. □

Because $bindfn$ does not decide whether a Java-program is binding well-formed with $dom_{inh_0^1} = \mathcal{C}$ and because $bindfn$ is not even a semideciding algorithm we want to modify BIPET´towards BIPET´´ and to make corresponding modifications of function $bindfn$ together with an auxiliary predicate $indom$ so that the following equivalences hold for all $P \in \mathcal{C}^{ROF}$:

1. $P \in dom^{RO}_{inh_0^1}$
2. $indom(P)$ is derivable in BIPET´´
3. predicate call $indom(P)$ terminates successfully (regularly) to $true$

$P \notin dom^{RO}_{inh_0^1}$ means non-derivability of $indom(P)$ resp. a calculation which is not terminating successfully with result $true$. Let's mention: If construction of $indom$ is going to be done such that the set of possible results is only $\{true\}$ then $P \notin dom^{RO}_{inh_0^1}$ means infinite calculation since all standard operations are total. Even $P = Fc$ means infinite calculation.

Also the following equivalences should hold for all $P \in \mathcal{C}^{ROF}$:

1. $bind^1_{inh_0^1}(ext(P)\ in\ decl(P)) = T \in dom^{OF}_{inh_0^1}$
2. $bindfn(ext(P)\ in\ decl(P)) = T$ is derivable in BIPET´´
3. function call $bindfn(ext(P)\ in\ decl(P))$ terminates successfully to result T.

The calculus BIPET´´ has the predicate

$$indom : \mathcal{C}^{ROF} \overset{part}{\longrightarrow} \mathbb{B}$$

beside the function $bindfn$. The rules are in Table 5.

Table 5. Rules of calculus BIPET˝

0. (BET"-Fc)	$\dfrac{\tilde{P} = Fc \text{ or } \tilde{X} = Ft}{bindfn(\tilde{X} \text{ in } \tilde{P}) = Fc}$
II. (BET"-InCT)	$\dfrac{P = Root \text{ and } P.C = Fc \text{ or } P.C \in \mathcal{C}^O \text{and } indom(P)}{bindfn(C \text{ in } P) = P.C}$
III. (BET"-SimpEncl)	$\dfrac{\begin{array}{l} bindfn(D \text{ in } P) = T \\ P.C \in \mathcal{C}^O,\ P.C.D = Fc,\ indom(P.C) \\ bindfn(ext(P.C).D \text{ in } P) = Fc \end{array}}{bindfn(D \text{ in } P.C) = T}$
IV. (BET"-SimpSup)	$\dfrac{\begin{array}{l} P.C \in \mathcal{C}^O,\ P.C.D = Fc,\ indom(P.C) \\ bindfn(ext(P.C).D \text{ in } P) = T \end{array}}{bindfn(D \text{ in } P.C) = \tilde{T}}$
V. (BET"-Long)	$\dfrac{\begin{array}{l} bindfn(X \text{ in } P) = T \\ T = Fc \text{ or } T.C \in \mathcal{C}^O \text{ and } indom(T) \end{array}}{bindfn(X.C \text{ in } P) = T.C}$
VI. (BET"-LongSup)	$\dfrac{\begin{array}{l} bindfn(X \text{ in } P) = P'.D \\ P'.D \in \mathcal{C}^O,\ P'.D.C = Fc,\ indom(P'.D) \\ bindfn(ext(P'.D).C \text{ in } P') = U \end{array}}{bindfn(X.C \text{ in } P) = U}$
VII.1. (BET"-Ind1)	$indom(Root)$
VII.2. (BET"-Ind2)	$indom(Object)$
VII.3. (BET"-Ind3)	$\dfrac{\begin{array}{l} bindfn(ext(P) \ in \ decl(P)) = T \\ indom(T) \end{array}}{indom(P)}$

Variables P, P' range over $\mathcal{C}^{RO}$, $\tilde{P}$ over $\mathcal{C}^{ROF}$,T, U over $\mathcal{C}^{OF}$, $\tilde{T}$ over $\mathcal{C}^O$, X over $\mathcal{CT}$, $\tilde{X}$ over $\mathcal{CT}^F$ and C, D over simple types $\mathcal{SCT}$.
Non-Boolean standard operations are extended towards total operations, Boolean standard operations remain total.

We can prove analoguously to Theorem 32:

Theorem 43: Let a syntactically correct program be given. Then the extended single valued binding function $bind^1_{inh^1_0}$ and the predicate $\in dom^{RO}_{inh^1_0}$ satisfy all nine rules of BIPET˝, i.e. are a model of BIPET˝where the binding well-formedness condition is restricted to the subset $dom_{inh^1_0}$ of $\mathcal{C}$ instead of the whole set $\mathcal{C}$ of user declared classes.

Proof: Restricted well-formedness holds due to Remark 21 and Corollary 22.
Satisfaction of Rules VII.1. and VII.2. is trivial.
Satisfaction of Rule VII.3. holds due to Remarks 15, 19 and Corollary 22.
Satisfaction of Rule 0. is due to Definition 31 of $bind^1_{inh^1_0}$ and due to properties of the parameterizing inheritance function inh^1_0 given by its fixed point construction in Corollary 22.
Satisfaction of Rules II. to VI. can be proved as for Thereom 32 where the added $indom$-premises are helping in our generalized situation of syntactically correct instead of well-formed programs. ■

We can prove analogouosly to Theorem 35:

Theorem 44: Every valid equation

$$bind^1_{inh^1_0}(X\ in\ P) = T$$

with $X \in \mathcal{CT}^F, P, T \in \mathcal{C}^{ROF}$ and every valid predicate

$$P \in dom^{RO}_{inh^1_0}$$

with $P \in \mathcal{C}^{ROF}$ is conclusion of a rule of BIPET˝such that all premises are satisfied where $bindfn$ is interpreted by $bind^1_{inh^1_0}$ and $indom$ by $\in dom^{RO}_{inh^1_0}$.

Proof: If $bind^1_{inh^1_0}(X\ in\ P) = T$ holds then P is necessarily $\in dom^{ROF}_{inh^1_0}$ and the proof works as for Theorem 35 including the extra premises on $indom$ in BIPET˝.
If $P \in dom^{RO}_{inh^1_0}$ then in the first two cases $P = Root$ or $P = Object$ the Rules VII.1. and VII.2. apply. The third case $P \in dom^1_{inh^1_0}$ can exploit the fact that inh^1_0 is a state (see Definition 12, Lemma 16 and Corollary 22) so that Rule VII.3. is applicable. ■

We can prove analoguously to Theorem 38:

Theorem 45: Every valid equation

$$bind^1_{inh^1_0}(X\ in\ P) = T$$

with $X \in \mathcal{CT}^F, P, T \in \mathcal{C}^{ROF}$ and every valid predicate

$$P \in dom^{RO}_{inh^1_0}$$

with $P \in \mathcal{C}^{ROF}$ is derivable by BIPET˝ with a finite derivation tree.

Proof: If we have an infinite derivation tree then there is an infinite path. There is at least one node of the kind

$$bfn(ext(P).C\ in\ decl(P)) = T \quad \text{or}$$
$$bfn(ext(P)\ |^i\ \ in\ decl(P)) = T, 1 \leq i \leq length(ext(P)).$$

As in the proof of Theorem 38 there is always a next node of this kind in the path upward

$$bfn(ext(P').C'\ in\ decl(P')) = T \quad \text{or}$$
$$bfn(ext(P')\ |^{i'}\ \ in\ decl(P')) = T', 1 \leq i' \leq length(ext(P'))$$

such that

$$\langle P, P' \rangle \in decl \cup dep_{bfn}$$

$(bfn = bind^1_{inh^1_0})$ and $decl \cup dep_{bfn}$ has a cycle what is contradicting Remark 21. ■

Corollary 46: Let a syntactically correct program be given. Then the extended binding function $bind^1_{inh^1_0}$ and the predicate $\in dom^{RO}_{inh^1_0}$ are the least model of BIPET˝ (and even the only model of BIPET˝) where the binding well-formedness condition is restricted to the subset $dom_{inh^1_0}$ of $\mathcal{C}$.

The BIPET˝-corresponding recursive function $bindfn$ and predicate $indom$ look as follows:

Definition 47:

Rules	Code
	$bindfn(X \text{ in } P) =$
	if $P = Fc \vee X = Ft$
Rule 0.	**then** Fc
	else if $X \in \mathcal{SCT}$
	then let $b = indom(P)$
	in if $P.X \in \mathcal{C}^O \vee P = Root \wedge P.X = Fc$
Rule II.	**then** $P.X$
	else let $T = bindfn(ext(P).X\ in\ decl(P))$
	in if $T \in \mathcal{C}^O$
Rule IV.	**then** T
Rule III.	**else** $bindfn(X\ in\ decl(P))$
	fi
	endlet
	fi
	endlet
	else let $X = X'.C, C \in \mathcal{SCT}$
	$T' = bindfn(X'\ in\ P),$
	$b = indom(T')$
	in if $T'.C \in \mathcal{C}^O \vee T' = Fc$
Rule V.	**then** $T'.C$
Rule VI.	**else** $bindfn(ext(T').C\ in\ decl(T'))$
	fi
	endlet
	fi
	fi
	end $bindfn$
	$indom(P) =$
	if $P = Root \vee P = Object$
Rule VII.1,VII.2	**then** $true$
	else let $T = bindfn(ext(P)\ in\ decl(P)),$
	$b = indom(T)$
Rule VIII.3	**in** $true$
	endlet
	fi
	end $indom$

There are occurrences of the letter b which simply denote local Boolean variables just as T, T' denote local class variables and X', C denote type variables. □

The proof of Theorem 48 is analoguous to that one of Theorem 41:

Theorem 48: The recursive function $bindfn$ plus predicate $indom$ form an algorithm which computes the associated class $bind^1_{inh^1_0}(X\ in\ P) \in \mathcal{C}^{OF}$ for every type $X \in \mathcal{CT}^F$ directly occurring in the body of class $P \in dom^{ROF}_{inh^1_0}$ (or exceptionally $X = Ft$ in class $P \in \mathcal{C} \setminus dom_{inh^1_0}$) in a given syntactically correct Java-program with $dom_{inh^1_0} \subseteq \mathcal{C}$. In case $X \in \mathcal{CT}$ in class $P \in \mathcal{C} \setminus dom_{inh^1_0}$ $\quad bind^1_{inh^1_0}(X\ in\ P)$ is undefined and call of $bindfn$, applied to X and P, does not terminate.

Continuation Example 42: Let us check $bindfn$ and $indom$ of Definition 47. Call of
$indom(C)$ yields $true$.
Call of
$indom(A)$ yields $indom(B)$ yields $indom(A) \ldots$,
so this calculation is running infinitely long.
These calculations confirm our equivalences mentioned in connection with Example 42. □

Now we would like to turn our function $bindfn$ and predicate $indom$ over into ones which terminate for all arguments X and P. $bindfn$ and $indom$ are provided with an additional call by value integer parameter d which keeps track of the depth of calls of $indom(\tilde{P})$ in the run time stack.

Definition 49:
$indom(P, d) =$
    if $d > card(\mathcal{C}) + 2$
    then error: original calculation is infinitely long
    else if $P = Root \vee P = Object$
        then $true$
        else let $T = bindfn(ext(P)\ in\ decl(P), d)$,
                $b = indom(T, d + 1)$
            in $true$
            endlet
        fi
    fi
end $indom$
$bindfn(X\ in\ P, d) =$
    if
        ⋮
      as in $indom$ every call of $bindfn$ is augmented by an
      actual parameter d and every call of $indom$ by $d + 1$
        ⋮
    fi
end $bindfn$ □

Usages of $bindfn$ and $indom$ are started by calls $bindfn(X\ in\ P,\ 0)$ and $indom(P, 0)$ instead of $bindfn(X\ in\ P)$ and $indom(P)$.

7. Concluding remarks

The identification of a declaring occurrence T of a class which is binding an applied occurrence of a (class) type X within a class P is basic for the understanding how a program works. The paper [IP02] offers the IPET-calculus for deducing the values of the function $bind(X \text{ in } P) = T$; in the original paper it is written $P \vdash X \Rightarrow T$. It has turned out that the formal system of IPET has many models, even minimal models, hence, the system does not define the binding function.

The discussion of the present paper shows how important it is to state a few questions known already in metamathematics:

1. (*determinacy or consistency*) It is obvious that a formal system may allow to prove a sentence in many alternative ways. However, a sound system does not allow to deduce mutually negating answers. In this case the question should be: is it true that for every class P and for every type X if calculus IPET allows to deduce two triplets $P \vdash X \Rightarrow T$ and $P \vdash X \Rightarrow U$ then $T = U$? We should be sure that the relation $P \vdash X \Rightarrow T$ is a function, which binds an applied occurrence of type X inside class P to just one declaration T of a class.

2. (*categoricity or completeness*) How many models has a proposed formal system? In our case the question is: are there different functions *bindfn* which are models of the IPET-calculus? The positive answer tells us that something important has escaped our attention, in our case the existence of the different models $bind^{\nu}_{inh^{\nu}_0}, 1 \leq \nu \leq \infty$, and minimal submodels in them.

3. (*repairing an incomplete system*) If there are several (minimal) models, one should try to repair the formal specification either by adding and changing axioms and inference rules (this way, we believe, is the correct one; so we have presented calculus BIPET´and BIPET´´) or by adding some metatheoretic rule like, for example, among all possible models choose the least one. Or better, among all possible models choose the one calculated by a certain algorithm, e.g. LSWA which delivers the inheritance function inh^1_0 either with $dom_{inh^1_0} = \mathcal{C}$ or delivers "given program is not well-formed" because $dom_{inh^1_0} \overset{\subset}{\neq} \mathcal{C}$.

These questions were not addressed in paper [IP02].

A few words on the problem formulated in the previous section

We stop here with one additional remark: one should consider the requirement that the formal theory of binding should allow to distinguish between well-formed programs and those which are not well-formed. The present authors do not know how to formulate an appropriate condition in terms of metamathematics. A candidate formulation like: "*if there exists a type X and class P such that the formula $\ulcorner bindfn^{\pi}(X, P) = null \urcorner$ has a proof then the program π is not well-formed (does not satisfy the sanity conditions)*" is far from being satisfactory. History of implementations of programming languages since 1960 has shown that decent understanding of the meanings of nested program structures is a great problem, not only for users, but even for language designers and compiler builders who are expected to have a higher education in informatics than users. A thorough pervasion of static binding of names, most natural since the origins of predicate logic and lambda calculus, by concepts of theoretical informatics, mathematics and mathematical logics is an absolute must. The more theoretical knowledges of binding we have the higher is the chance that both – users and compilers – conceive program semantics in the same manner. Strong theoretical connections assure that ideas of programming language designers and practicioners will achieve lasting importance.

Acknowledgement. We would like to thank the anonymous reviewers of article [LSW09] who have encouraged us to write a full paper on our observations of types elaboration in Java with inner classes in Igarashi's and Pierce's article [IP02].

Appendix: Algorithm LSWA$^\nu$ to construct inheritance function inh_0^ν and to decide binding well-formedness

Algorithm LSWA$^\nu$, $0 \leq \nu \leq \infty$, computes for a given Java-program structure a chain
$inh_\perp^\nu = sst_0^\nu \prec^{DS} sst_1^\nu \prec^{DS} \cdots \prec^{DS} sst_n^\nu = inh_\perp^{\nu\, max} = inh_0^\nu$,
$0 \leq n \leq card(\mathcal{C})$, of direct successor states beginning with the bottom(empty) state and ending with the unique maximal successor state which is the least fixed point $inh_0^\nu = \mu bdfl'^\nu$ (Remark 19 and Corollary 22). The main part of algorithm LSWA$^\nu$ is the same as of LSWA in [LSW09] for all $1 \leq \nu \leq \infty$:

var $\mathcal{INH}\ inh,\ \mathcal{P}(\mathcal{C})\ Candidates,\ \mathcal{C}\ K,\ \mathcal{C}^O\ M$;
$inh := \emptyset$;
while $dom_{inh} \neq \mathcal{C}$
do $Candidates := \{K \in \mathcal{C} : decl(K) \in dom_{inh}^R \wedge K \notin dom_{inh}\}$;
  if $(\exists K \in Candidates)\ bind_{inh}^{rst\,\nu}(ext(K)\ in\ decl(K)) \in dom_{inh}^O$
  then $K :=$ one of such generating candidates;
    $M := bind_{inh}^{rst\,\nu}(ext(K)\ in\ decl(K))$;
    $inh := inh \cup \{\langle K, M\rangle\}$
  else error: irregular termination with a final value of
    inh which is the maximal successor state $inh_\perp^{\nu\, max} =$
    inh_0^ν with $dom_{inh_0^\nu} \neq \mathcal{C}$
  fi
endwhile
regular, successful termination of LSWA$^\nu$ with a final value of inh which is $inh_\perp^{\nu\, max} = inh_0^\nu$ with $dom_{inh_0^\nu} = \mathcal{C}$, i.e. the given program structure is binding well-formed w.r.t. binding function $bind_{inh_0^\nu}^\nu$.

We write down the programmed restricted binding function $bind_{inh}^{rst\,\nu}(X\ in\ P)$ only for index $\nu = \infty$ because $bind_{inh_0^\infty}^\infty$ is the most surprizing model IPET of resp. BIPET which is essentially different from Java's official binding function and model $bind_{inh_0^1}^1$. If the two preconditions 1) inh is a state and 2) $P \in dom_{inh}^{RO}$ hold then invocation of $bind_{inh}^{rst\,\infty}(X\ in\ P)$ terminates regularly such that result $T \in dom_{inh}^{OF}$ satisfies the postcondition $T = bind_{inh}^\infty(X\ in\ P)$. I.o.w. $bind_{inh}^{rst\,\infty}$ is totally correct w.r.t. these pre- and postconditions.

$bind_{inh}^{rst\,\infty}(X\ in\ P) =$
if $X \in \mathcal{SCT}$
then if $P.X$ is defined $\in \mathcal{C}^O$, i.e. $\neq Fc$
  then $P.X$
  else if $P \in dom_{inh}$
    then let $T = bind_{inh}^{rst\infty}(X\ in\ inh(P))$;
      in if T is defined $\in \mathcal{C}^O$, i.e. $\neq Fc$
        then T
        else $bind_{inh}^{rst\,\infty}(X\ in\ decl(P))$
        fi
      endlet
    else **if** $P = Object$
      then $bind_{inh}^{rst\,\infty}(X\ in\ Root)$
      else Fc
      fi
    fi
  fi
else let $X = X'.C, C \in \mathcal{SCT}$
  $T' = bind_{inh}^{rst\,\infty}(X'\ in\ P)$
  in $bind_{inh}^{rst\,\infty}(C\ in\ T')$
  endlet
fi
end $bind_{inh}^{rst\,\infty}$

References

[Bar+82] W. M. Bartol et al.. The Report on the Loglan'82 Programming Language. PWN, Warszawa, 1984

[BaWo82] F.L.Bauer, H.Wössner. Algorithmic Language and Program Development. Springer, 1982

[Bjo09] D.Bjoerner. Domain Engineering – Technology Management, Research and Engineering. COE Research Monograph Series, Vol. 4, JAIST Japan, 2009

[Boe01] R. Staerk, J. Schmid, E. Boerger. Java and the Java Virtual Machine- Definition, Verification, Validation, Springer-Verlag, June 2001

[Chu41] A.Church. The Calculi of Lambda-Conversion. Princeton University Press, 1941

[DaNy67] O.-J.Dahl, K.Nygaard. Class and Subclass Declarations. In: J.N.Buxton (ed.). Simulation Programming Languages. Proc. IFIP Work. Conf. Oslo 1967, North Holland, Amsterdam, 158-174, 1968

[Dij60] E.W. Dijkstra. Recursive Programming. Numerische Mathematik **2**, 312-318, 1960

[Fre1879] G.Frege. Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens. Halle a.S., 1879

[GHL67] A.Grau, U.Hill, H.Langmaack. Translation of ALGOL60. Handbook for Automatic Computation, Vol. I, Part b (chief ed. K.Samelson), Springer 1967

[GJS96] J. Gosling, B. Joy, G. Steele. The Java Language Specification. First edition, Addison-Wesley 1996

[GJSB00] J. Gosling, B. Joy, G. Steele, G. Bracha. The Java Language Specification. Second edition, Addison-Wesley 2000

[GJSB05] J. Gosling, B. Joy, G. Steele, G. Bracha. The Java Language Specification. Third edition, Addison-Wesley 2005

[Ich80] J.D. Ichbiah. Ada Reference Manual. LNCS 106, Springer-Verlag, Berlin, Heidelberg, New York 1980

[IP02] A. Igarashi, B. Pierce. On inner classes. Information and Computation **177**, 56-89, 2002

[Kan74] P.Kandzia. On the "most recent"-property of ALGOL-like programs. In Proc. 2nd Coll. Automata Languages and Programming (J.Loeckx, ed.). LNCS 14, 97-111. Berlin, Heidelberg, New York, Springer 1974

[KSW88] A.Kreczmar, A.Salwicki, M.Warpechowski. Loglan'88 - Report on the Programming Language. LNCS 414, Springer, Berlin 1990

[Lan73] H.Langmaack. On Correct Procedure Parameter Transmission in Higher Programming Languages. Acta Informatica **2**, 2, 110-142, 1973

[Lan10] H.Langmaack. Dijkstras fruchtbarer, folgenreicher Irrtum. Informatik Spektrum Band 33, Heft 3, 302-308, Heft 4, 384-392, Heft 6, 634-646, 2010

[LoSi84] J. Loeckx, K. Sieber. The Foundation of Program Verification. Wiley-Teubner 1984

[LSW04] H.Langmaack, A. Salwicki, M.Warpechowski. On correctness and completeness of an algorithm determining inherited classes and on uniqueness of solutions. In: G.Lindemann et al., Proc. CS&P'2004, Caputh Sept. 24-26, Vol. 2, 319-329, Informatik-Berichte, Humboldt Univ. Berlin, 2004

[LSW08] H.Langmaack, A.Salwicki, M.Warpechowski. On a deterministic algorithm identifying direct superclasses in Java, Fundamenta Informaticae **85**, 343-357, 2008

[LSW08b] H.Langmaack, A.Salwicki, M.Warpechowski.Some methodological remarks inspired by the paper "On inner classes" by A.Igarashi and B.Pierce. Proc. CS&P'2008, Groß Väter See. Informatik-Berichte, Humboldt-Univ. Berlin, 448-462, 2008

[LSW09] H.Langmaack, A.Salwicki, M.Warpechowski. On an algorithm determining direct superclasses in Java-like languages with inner classes – its correctness, completeness and uniqueness of solutions. Information and Computation **207**, 389-410, 2009

[Man74] Z.Manna. Mathematical Theory of Computation. McGraw-Hill, 1974

[McC$^+$65] J.McCarthy et al.. LISP 1.5 Programmer's Manual. The M.I.T.Press, Cambridge, Mass., 1965

[McG72] C.L.McGowan. The "most recent"-error: its causes and correction. In: Proc. ACM Conf. on Proving assertions about programs. SIGPLAN Notices 7, No.1, 191-202, 1972

[MMPN93] O.L.Madsen, B.Moeller-Pedersen, K.Nygaard. Object Oriented Programming in the BETA Programming Language. Addison Wesley / ACM Presss, 1993, see also: Beta Programming Language, 2001, available from: http://www.daimi.au.dk/∼beta/

[MSST2001] G.Mirkowska, A.Salwicki, M. Srebrny, A. Tarlecki. First-order Specifications of Programmable Data Types, SIAM Journal on Computing, **30**, pp. 2084-2096, 2001

[Nau+60/63] Naur, P. (ed.) et al.: Report on the Algorithmic Language ALGOL60. Num. Math. 2, 106-136 (1960); Revised, Num. Math.4, 420-453, 1963

[Old81] E.R.Olderog. Charakterisierung Hoarescher Systeme für ALGOL-ähnliche Programmiersprachen. Dissertation, Inst. F. Informatik u. Prakt. Math., Univ. Kiel, Bericht 5/81, 1981

[Ste84] G.L.Steele jr.. Common LISP - The Language. Digital Press 1984

[Sto84] H.Stoyan. Early LISP History (1956-1959). LFP'84 Proceedings of the 1984 ACM Symposium on LISP and Functional Programming, Austin, 5-8 August, 299-310. ACM 1984

[WaGo84] W.M.Waite, G.Goos. Compiler Construction. Springer, New York Berlin Heidelberg Tokyo 1984

[Wij+68] A.van Wijngaarden et al. (eds.). Report on the Algorithmic Language ALGOL68. Numerische Mathematik **14**, 79-218, 1969

[WiMa92/97] R.Wilhelm, D.Maurer. Übersetzerbau - Theorie, Konstruktion, Generierung. Springer, Berlin Heidelberg New York 1992, 2. Aufl. 1997