Imperial College London

Department of Computing

Intersection Types for Class-based Object Oriented Programming

by

Reuben N. S. Rowe

Submitted in partial fulfilment of the requirements for the MSc Degree in Advanced Computing of Imperial College London

September 2008

Abstract

Intersection type systems have been well studied in the context of the Lambda Calculus and functional programming over the last quarter of a century or so. Recently, the principles of intersection types have been successfully applied in an *object oriented* context [5]. This work was done using the ς object calculus of Abadi and Cardelli [1], however we note that this calculus leans more towards the *object-based* approach to object orientation, and wish to investigate how intersection types interact with a quintessentially *class-based* approach. In order to do this, we define a small functional calculus that expresses class-based object oriented features and is modelled on the similar calculi of Featherweight Java [18] and Middleweight Java [8], which are ultimately based upon the Java programming language. We define a *predicate* system, similar to the one defined by van Bakel and de'Liguoro [5], and show subject reduction and expansion. We discuss the implications that this has for the characterisation of expressions in our calculus, and define a restriction of the predicate system which we informally argue is decidable.

Acknowledgements

I gratefully acknowledge the help and guidance that I have received throughout the last few months from my supervisor, Dr. Steffen van Bakel, who was responsible for my interest in intersection types in the first place.

Contents

Chapter 1

Introduction

The relationship between computing theory and computing practice is not a straightforward one. At times theoretical principles are developed first and subsequently put to practical use. At others, practical applications arise in the industry without the underlying foundations being well understood.

We see this dynamic at work in the theory of programming languages and type systems. In the 1930s, two theoretical models of computation were developed by Church and Turing, before the first stored-program computer was even thought of. This work, on the Lambda Calculus (λ-calculus) [6] and Turing machines [24], led to a deeper understanding of the fundamental notion of what a computation actually is, as well as the limits of what can be computed. When the electronic computer was invented and programming languages were developed, it turned out that the λ-calculus provided an ideal framework upon which programming languages could be based. This led to the development of the *functional* programming paradigm, so called because it is based on the theory of *functions*, and exemplified in languages such as ML [19].

In parallel with work on functional programming, the *object oriented* paradigm was being developed. The first example of this style of programming is the language Simula [13], developed in the 1960s in response to a need for implementing simulation software. This mode of programming became widespread when it was implemented in the language C++ [23], and its popularity has been maintained by more recent languages such as Java [17] and C# [21]. Despite its evident practical usefulness, a formal theoretical framework to describe object orientation had been lacking until recently, when Abadi and Cardelli carried out their seminal work on an *object calculus*, the ς-calculus [1]. This calculus is a highly abstract view of object oriented programming, and describes many features found in a multiplicity of programming languages. This generality notwithstanding, the ς-calculus was formulated using a particular view of the object oriented methodology – the *object-based* variety as opposed to the *class-based* variety. For this reason, other work was carried out to develop similar formal models describing class-based languages. Notable efforts are Featherweight Java [18] and its successor Middleweight Java [8]. We will examine the basic difference between the two approaches in Chapter 2.

An integral aspect of the theory of programming languages is *type theory* which allows abstract reasoning about programs to be carried out, and certain guarantees to be given about their behaviour. Type theory arose side-by-side with the formal models we have already seen, one of the earliest being Curry's system for the λ-calculus [12]. The *intersection type* discipline was first developed in the early 1980s [10, 11, 7] to extend Curry's system and address certain inadequacies therein. With the advent of the ς-calculus, work was carried out by van Bakel and de'Liguoro to apply the principles of intersection types to object oriented programming [5].

In this work, we aim to follow up these latest efforts and apply the principles of intersection types, and the system of [5] specifically, to a formal model of *class-based* object oriented programming. The formal model that we use is based on [18] and [8]. We find that we would like to use a slightly richer calculus than [18], but that the collection of features in [8] is too complex for our purposes. Therefore we define a new calculus, which we call *Lightweight Java*. Having defined the calculus, we will then prove subject reduction and expansion results. Subject reduction is a standard result for type systems leading to *type soundness*. Subject expansion is a standard property of *intersection* type systems, thus we demonstrate that our predicate system is as expressive as [10, 11, 7, 5]. Since intersection type assignment is typically undecidable, we look at two restriction of our predicate system which we believe *are* decidable. They are inspired by the decidable restrictions of the intersection type system for term rewriting systems in [3]. However, our restrictions do not directly descend from that work, since our predicate system displays significant differences to that one.

This report is organised as follows: in Chapter 2 we survey some relevant work. We first describe intersection types as they have been formulated for the Lambda Calculus, and then look at the ς-calculus and how intersection types have been applied to it in the form of the predicate system of [5]. We then briefly touch upon two calculi that have been developed specifically to describe the Java programming language, namely Featherweight Java and Middleweight Java. Chapter 3 then builds upon these two calculi, and defines Lightweight Java. In Chapter 4 we formulate both a standard type system (like those of Featherweight Java and Middleweight Java), and an intersection type system based upon the aforementioned predicate system. In Chapter 5, we derive subject reduction and expansion results for the type systems, before defining two decidable restrictions of the predicate system in Chapter 6. Finally, we draw conclusions and look at future work in Chapter 7.

Chapter 2

Background

In this chapter, we undertake a short survey of some relevant formal systems in order to place the work that follows in context. We begin by looking at the Lambda Calculus (λ-calculus) [6], the earliest such system for studying the formal notion of computation and also the first system for which intersection types were developed. We then move on to look at Cardelli and Abadi's Object Calculus (ς-calculus) [1], which is significant as it was the first attempt at placing the *object oriented* programming paradigm on the same theoretical footing as functional programming. Finally, we examine two other calculi which build upon this approach to describe class-based object oriented features, specifically those of Java [17].

2.1 The λ-calculus and Intersection Types

The λ-calculus was developed in the 1930s by A. Church as a formal model for studying the notions of functions, computability and recursion. It forms a theoretical basis for the *functional programming paradigm*, and is an extremely well studied system. It is also an extremely simple system, capturing the two basic notions of function construction and application, yet it is enormously expressive: concepts such as numerals and arithmetic, Boolean logic and numerical predicates (equal, less than, greater than, etc.) and tuples (or records), which are all found universally in everyday programming, can all be encoded in the λ-calculus. We will now define the λ-calculus.

Definition 2.1.1 (λ-TERMS)**.** Terms in the λ-calculus (ranged over by M, N) are defined by the following grammar:

$$M, N ::= x \mid (\lambda x.M) \mid (MN)$$

where x ranges over a set of term variables. Repeated abstractions can be abbreviated (i.e. $\lambda x.\lambda y.\lambda z.M$ is written as $\lambda xyz.M$) and left-most, outer-most brackets in function applications can be omitted.

The function constructor, λ, is a *variable binder*. So, in a term $\lambda x.M$, λ binds (or captures) the variable x, and any occurrences of x in the sub-term M are 'bound'. We can therefore talk about free and bound variables in a term:

Definition 2.1.2 (FREE & BOUND VARIABLES)**.** For a λ-term M, the free variables of M, denoted by $fv(M)$, and the bound variables of M, denoted by $bv(M)$, are defined by:

$$\begin{array}{lcllcl} fv(x) & = & \{x\} & bv(x) & = & \emptyset \\ fv(M_1M_2) & = & fv(M_1) \cup fv(M_2) & bv(M_1M_2) & = & bv(M_1) \cup bv(M_2) \\ fv(\lambda x.M) & = & fv(M) \setminus \{x\} & bv(\lambda x.M) & = & bv(M) \cup \{x\} \end{array}$$

Replacement of a variable x by a term N (called *substitution* and denoted by $[N/x]$), forms the foundation of *reduction* (or computation). When defining reduction, we refer to a notion of equivalence on terms based on renaming of bound variables, called α-conversion. This equivalence is defined from $(\lambda y.M) =_\alpha (\lambda z.M[z/y])$ and is extended to all terms so that, for example, $\lambda xyz.xz(yz) =_\alpha \lambda abc.ac(bc)$. If we assume that bound and free variables of a term are always different (Barendregt's convention), then substitution can be defined as follows:

Definition 2.1.3 (SUBSTITUTION).

$$\begin{array}{rcl} x[N/x] & = & N \\ y[N/x] & = & y \qquad (\text{if } y \neq x) \\ (M_1 M_2)[N/x] & = & (M_1[N/x]\, M_2[N/x]) \\ (\lambda y.M)[N/x] & = & (\lambda y.(M[N/x])) \end{array}$$

In the last case since y is bound in $\lambda y.M$, by Barendregt's convention, it is not free in N so y is not captured during the substitution.

Using substitution, we define the (β-)reduction relation $\rightarrow_\beta$ on λ-terms as follows:

Definition 2.1.4 (β-REDUCTION).

$$\begin{array}{rcl} (\lambda x.M)N & \rightarrow_\beta & M[N/x] \\ M \rightarrow_\beta N & \Rightarrow & \left\{ \begin{array}{lcl} M'M & \rightarrow_\beta & M'N \\ MM' & \rightarrow_\beta & NM' \\ \lambda x.M & \rightarrow_\beta & (\lambda x.N) \end{array} \right. \end{array}$$

During β-reduction, α-conversion takes place automatically in order to avoid variable capture. The reflexive and transitive closure of $\rightarrow_\beta$ is denoted by $\twoheadrightarrow_\beta$. Having defined this notion of reduction, we can then characterise terms by their structure and reduction properties:

Definition 2.1.5 (TERM CHARACTERISATION).

1. A term of the form $(\lambda x.M)N$ is called a *reducible expression*, or redex. The term $M[N/x]$, obtained by performing a single reduction step, is called the *reduct*.

2. A term is in *head-normal form* if it is in the form $\lambda x_1 \cdots x_n.yM_1 \cdots M_n$. Terms in head-normal form can be defined by the grammar:

$$H ::= x \mid \lambda x.H \mid xM_1 \cdots M_n (n \geq 0, M_i)$$

   where M is any λ-term.

3. A term is in *normal form* if it does not contain a redex. Terms in normal form can be defined by the grammar:

$$N ::= x \mid \lambda x.N \mid xN_1 \cdots N_n (n \geq 0)$$

   By definition, a term in normal form is also in head-normal form.

4. A term is *normalisable* if it has a normal form, i.e. if there exists a term N in normal form such that $M \twoheadrightarrow_\beta N$. Similarly, a term is *head-normalisable* if it has a head-normal form. By definition, all normalisable terms are head-normalisable.

The definitions above constitute the *un-typed* λ-calculus. Type systems allow us to build a layer of abstraction over this basic calculus, allowing us to talk about the properties of terms, and their execution (i.e reduction), in a more general way. Many varieties of *type system* have been developed for the λ-calculus and its successors. The first type system for the λ-calculus was that of Curry [12]. In this system, types are formed from a set of type variables Φ ($\{\alpha, \beta, \sigma, \tau, \ldots\}$), and the type constructor $\rightarrow$, which is used to indicate function abstraction. So, for example, the type $\sigma \rightarrow \tau$ denotes a function that takes an input of type σ and returns an output of type τ.

Definition 2.1.6 (CURRY TYPES). The set of types in the Curry type assignment system, ranged over by ϕ, is defined by the following grammar:

$$\phi ::= \varphi \mid \phi \rightarrow \phi$$

where φ ranges over Φ.

Type assignment is represented by a ternary relation on λ-terms M, types ϕ and contexts (or bases) B. Bases are partial mappings from term variables to types, and are written as a set of statements, $\{x : \phi_1, y : \phi_2, z : \phi_3\ , \ldots\}$, each one asserting that a variable x is mapped to a type ϕ. $B, x{:}\,\phi$ will be written for $B \cup \{x{:}\,\phi\}$ where B does not contain a statement about x.

Definition 2.1.7 (CURRY TYPE ASSIGNMENT)**.** Curry type assignment is defined by the following natural deduction system:

$$(Ax):\quad \frac{}{B \vdash_C x : \phi}\ x : \phi \in B$$

$$(\to I):\quad \frac{B, x : \phi \vdash_C M : \tau}{B \vdash_C \lambda x.M : \phi \to \tau}$$

$$(\to E):\quad \frac{B \vdash_C M_1 : \phi \to \tau \quad B \vdash_C M_2 : \phi}{B \vdash_C M_1 M_2 : \tau}$$

The statement $B \vdash_C M : \phi$ indicates that the type ϕ can be assigned to the term M under the basis B using the rules of the natural deduction system above. Curry's system exhibits a *subject reduction* property, that is if $B \vdash_C M : \phi$ and $M \to_\beta N$ then $B \vdash_C N : \phi$. There are λ-terms which cannot be typed in this system, however. These are the terms that contain some form of *self-application*, the simplest example of which is the term xx. In order for this term to be typeable, the variable x must have a both a function type $\phi \to \sigma$ and the type ϕ, which is not possible in Curry's system. The *intersection type* discipline addresses this problem by allowing terms to have more than one type. We will briefly look at one variant in particular - the strict intersection type system.

Definition 2.1.8 (STRICT INTERSECTION TYPES)**.** The set of intersection types (ranged over by σ), and the set of strict types (ranged over by ϕ) are defined by the following grammar:

$$\begin{array}{lcll} \phi & ::= & \varphi \mid (\sigma \to \phi) & \\ \sigma & ::= & (\phi_1 \cap \ldots \cap \phi_n) & \qquad (n \geq 0) \end{array}$$

We call $\phi_1 \cap \ldots \cap \phi_n$ an *intersection*, and when this type is assigned to a term, it denotes that the term also has each of the individual types ϕ_i. An intersection type is permitted to be empty ($n = 0$), in which case we write ω.

Definition 2.1.9 (STRICT INTERSECTION TYPE ASSIGNMENT)**.** Strict intersection type assignment is defined by the following natural deduction system:

$$(\cap E):\quad \frac{}{B, x : \phi_1 \cap \ldots \cap \phi_n \vDash_S x : \phi_i}\ n \geq 1, i \in \underline{n} \qquad (\to I):\quad \frac{B, x : \sigma \vDash_S M : \phi}{B \vDash_S \lambda x.M : \sigma \to \phi}$$

$$(\cap I):\quad \frac{B \vDash_S M{:}\,\phi_1 \ \ldots\ B \vDash_S M : \phi_n}{B \vDash_S M : \phi_1 \cap \ldots \cap \phi_n}\ n \geq 0 \qquad (\to E):\quad \frac{B \vDash_S M : \sigma \to \phi \quad B \vDash_S N : \sigma}{B \vDash_S MN : \phi}$$

Self-application is now typeable:

$$\dfrac{\dfrac{}{\{x : \phi \to \sigma \cap \phi\} \vDash_S M : \phi \to \sigma}\,(\cap E) \qquad \dfrac{}{\{x : \phi \to \sigma \cap \phi\} \vDash_S M : \phi}\,(\cap E)}{\{x : \phi \to \sigma \cap \phi\} \vDash_S M : \sigma}\,(\to E)$$

In fact, the strict intersection system is able to type *all* terms. In addition, it exhibits a *subject expansion* property, that is, if $B \vdash_C N : \phi$ and $M \to_\beta N$ then $B \vdash_C M : \phi$. A further result of this intersection type system (and many other variations) is that it is able to *characterise* terms by their assignable types:

1. (HEAD-NORMALISATION) If $B \vdash M{:}\,\sigma$ & $\sigma \neq \omega$ then M has a head-normal form

2. (NORMALISATION) If $B \vdash M{:}\,\sigma$ and B, σ are ω-free then M has a normal form

This is shown in [7, 4]. From these results, we also see that terms with no normal form at all (so-called *unsolvable* terms) can only be assigned the type ω. These characterisation results are significant, and we will compare them with our own system in Chapter 5. This great expressive power comes at a price, however: whereas Curry type assignment is decidable, intersection type assignment is not.

2.2 The ς-calculus and Predicates

The ς-calculus [1] was developed by L. Cardelli and M. Abadi in the mid-1990s as an effort to provide a theoretical foundation for the object oriented programming paradigm, similar to that provided for functional programming by the λ-calculus. Object oriented programming began with the languages Simula 67 [13] and Smalltalk [16], and was popularised by the languages C++ [23] and Java [17]. The central concept of object oriented programming is that of the *object*. Computation is defined in terms of objects, which pass messages between one another. Objects are essentially records, which encapsulate computational behaviour by exposing *methods* which can be invoked by other objects. Objects may also store values in a series of *fields*.

There are two main 'flavours' of object oriented programming: the first is the object-based approach in which the programmer is free to modify objects (e.g. by adding or removing methods and fields) on an object-by-object basis. In this form of object oriented programming, each object is an independent entity. The ς-calculus mainly describes this approach to object orientation. The alternative approach is that of *class-based* object orientation, in which each object is an *instance* of a particular template, which the programmer defines in the form of a *class*. Both C++ and Java are examples of this latter form of object orientation.

In its entirety, the object calculus of Abadi and Cardelli is extensive and comprises many different fragments, which may be combined in different ways. For example, one fragment is itself the λ-calculus, and so the ς-calculus can be taken to be a superset of that system. Here, we will attempt to describe the essence of the calculus by looking at the simplest fragment, which is called $\mathbf{Ob_1}$ in [1] and deals only with objects in their most primitive form. Thus, this overview does not attempt to be exhaustive. We will also try to follow the notation in [1] as closely as possible.

Definition 2.2.1 (ς-calculus syntax)**.** Let l range over a set of (method) labels. Also, let x, y, z range over a set of variables. Types and terms in the ς-calculus are defined as follows:

$$\begin{array}{rcl}
\textbf{Types} & & \\
A, B & ::= & [l_1{:}B_1, \ldots, l_n{:}B_n] \\
\textbf{Terms} & & \\
a, b, c & ::= & x \mid [l_1{:}\varsigma(x_1^{A_1})b_1, \ldots, l_n{:}\varsigma(x_n^{A_n})b_n] \mid a.l \mid a.l \Leftarrow \varsigma(x^A)b \\
\textbf{Values} & & \\
v & ::= & [l_1{:}\varsigma(x_1^{A_1})b_1, \ldots, l_n{:}\varsigma(x_n^{A_n})b_n]
\end{array}$$

We use $[l_i{:}B_i{}^{i \in 1..n}]$ to abbreviate the type $[l_1{:}B_1, \ldots, l_n{:}B_n]$, and $[l_i{:}\varsigma(x^{A_i})b_i{}^{i \in 1..n}]$ to abbreviate the term $[l_1{:}\varsigma(x_1^{A_1})b_1, \ldots, l_n{:}\varsigma(x_n^{A_n})b_n]$.

Thus, we have objects $[l_i{:}\varsigma(x^{A_i})b_i{}^{i \in 1..n}]$ which are a sequence of methods $\varsigma(x^A)b$. Methods can be invoked by the syntax $a.l$, or overridden with a new method using the syntax $a.l \Leftarrow \varsigma(x^A)b$. Notice that types are embedded into the syntax of terms. ς is a *binder*, much like λ in the Lambda Calculus, so that the variable x is bound in the term $\varsigma(x^A)b$. Again, as in the λ-calculus, bound (or equivalently, free) variables provide the mechanism by which reduction takes place:

Definition 2.2.2 (free variables)**.** The set of free variables of a term, $FV(a)$, is defined inductively as follows:

$$\begin{array}{lcl}
FV(x) & \triangleq & \{x\} \\
FV(\varsigma(x^A)b) & \triangleq & FV(b) \setminus \{x\} \\
FV([l_i{:}\varsigma(x^{A_i})b_i{}^{i \in 1..n}]) & \triangleq & FV(\varsigma(x^A)b_1) \cup \ldots \cup FV(\varsigma(x^A)b_n) \\
FV(a.l) & \triangleq & FV(a) \\
FV(a.l \Leftarrow \varsigma(x^A)b) & \triangleq & FV(a) \cup FV(\varsigma(x^A)b)
\end{array}$$

Definition 2.2.3 (OBJECT SUBSTITUTION)**.** The notation $a\{x \leftarrow b\}$ denotes the replacement of all occurrences of the variable x in a by the term c. It is defined inductively as follows:

$$\begin{array}{lcll}
x\{x \leftarrow c\} & \triangleq & c & \\
x\{x \leftarrow c\} & \triangleq & y & \text{for } y \neq x \\
\varsigma(y^A)b\{x \leftarrow c\} & \triangleq & \varsigma(z^A)(b\{y \leftarrow z\}\{x \leftarrow c\}) & \text{for } z \notin FV(\varsigma(y^A)b) \cup FV(c) \cup FV(x) \\
[l_i{:}\varsigma(x^{A_i})b_i{}^{i \in 1..n}]\{x \leftarrow c\} & \triangleq & [l_i{:}(\varsigma(x^A)b_i)\{x \leftarrow c\}^{i \in 1..n}] & \\
(a.l)\{x \leftarrow b\} & \triangleq & (a\{x \leftarrow b\}).l & \\
(a.l \Leftarrow \varsigma(y^A)b)\{x \leftarrow b\} & \triangleq & (a\{x \leftarrow b\}).l \Leftarrow ((\varsigma(y^A)b)\{x \leftarrow b\}) &
\end{array}$$

Notice the interesting case of substituion on a method $\varsigma(x^A)b$. Here, we explicitly choose a fresh variable that does not occur in either b or c and rename the bound variable, thus avoiding free variable capture in c. Using this notion of substitution, a reduction relation is defined on terms:

Definition 2.2.4 (REDUCTION)**.**

1. An *evaluating context* is a term with a hole $[\_]$, and is defined by the following grammar:

$$\mathcal{E}[\_] ::= \_ \mid \mathcal{E}[\_].l \mid \mathcal{E}[\_].l \Leftarrow \varsigma(x^A)b$$

   $\mathcal{E}[a]$ denotes filling the hole in $\mathcal{E}$ with a.

2. The one-step reduction relation on terms is the binary relation defined by the following rules:

$$\begin{array}{rcll}
[l_i{:}\varsigma(x^{A_i})b_i{}^{i \in 1..n}].l_j & \to & b_j\{x_j \leftarrow [l_i{:}\varsigma(x^{A_i})b_i{}^{i \in 1..n}]\} & j \in 1..n \\
[l_i{:}\varsigma(x^{A_i})b_i{}^{i \in 1..n}].l_j \Leftarrow \varsigma(x^A)b & \to & [l_1{:}\varsigma(x_1^{A_1})b_1, \ldots, l_j{:}\varsigma(x^A)b, \ldots, l_n{:}\varsigma(x_n^{A_n})b_n] & j \in 1..n \\
a \to b & \Rightarrow & \mathcal{E}[a] \to \mathcal{E}[b] &
\end{array}$$

3. The relation $\to^*$ is the reflexive and transitive closure of $\to$.

4. If $a \to^* v$ then we say that *a converges* to the value v, and write $a \Downarrow v$.

Like the λ-calculus, the following type assignment system for $\mathbf{Ob_1}$ has a subject reduction property: if $E \vdash_O a : A$ and $a \to b$ then $E \vdash_O b : A$. Subject expansion, however, does not hold.

Definition 2.2.5 (OBJECT TYPE ASSIGNMENT)**.**

1. An object environment, E, is a sequence of statements of the form $x{:}A$.

2. Object types are assigned to terms using the following natural deduction system:

(Val x)

$$\frac{}{E, x{:}A, E' \vdash_O x : A}$$

(Val Object) (where $A \equiv [l_i{:}B_i{}^{i \in 1..n}]$)

$$\frac{E, x{:}A \vdash_O b_i : B_i \quad \forall\, i \in 1..n}{E \vdash_O [l_i{:}\varsigma(x^A)b_i{}^{i \in 1..n}] : A}$$

(Val Select)

$$\frac{E \vdash_O a : [l_i{:}B_i{}^{i \in 1..n}]}{E \vdash_O a.l_j : B_j}\ (j \in 1..n)$$

(Val Update) (where $A \equiv [l_i{:}B_i{}^{i \in 1..n}]$)

$$\frac{E \vdash_O a : A \quad E, x{:}A \vdash_O b : B_j}{E \vdash_O a.l_j \Leftarrow \varsigma(x^A)b : A}\ (j \in 1..n)$$

The work of S. van Bakel and U. de'Liguoro [5] has taken the principles of intersection type systems, as outlined in the previous section, and applied it to the object calculus of Abadi and Cardelli, obtaining similar results. Here, we will briefly outline the basic elements of the *predicate* system, as it is called in [5].

Definition 2.2.6 (PREDICATES)**.**

1. The set of predicates, ranged over by $\sigma, \tau, \ldots$ and its subset of *strict* predicates ranged over by $\phi, \psi, \ldots$ are defined by the following grammar:

$$\begin{array}{rcl}
\phi & ::= & \omega \mid (\sigma \to \phi) \mid \langle l{:}\phi \rangle \\
\sigma, \tau & ::= & \phi \mid (\sigma \wedge \tau)
\end{array}$$

2. A relation $\leq$ is defined on predicates as the least pre-order (reflexive and transitive relation) such that for any predicate σ, τ and strict predicate ϕ, ψ:

   (a) $\sigma \leq \omega$;

   (b) $(\sigma \wedge \tau) \leq \sigma$ and $(\sigma \wedge \tau) \leq \tau$;

   (c) $(\sigma \to \omega) \leq (\omega \to \omega)$;

   (d) $\tau \leq \sigma, \phi \leq \psi \Rightarrow (\sigma \to \phi) \leq (\tau \to \psi)$;

   (e) $\phi \leq \psi \Rightarrow \langle l{:}\phi\rangle \leq \langle l{:}\psi\rangle$ for any label l.

The predicate assignment system, then, assigns predicates to typeable terms. Part of that system defined in [5] is a separate notion of assignment of predicates to *types*, however for simplicity we will omit this definition from the discussion (thus slightly modifying the definition of predicate assignment), as it is not immediately relevant to the work that is presented in later chapters.

Definition 2.2.7 (PREDICATE ASSIGNMENT). 1. A predicate environment, Γ, is a sequence of statements of the form $x{:}A{:}\sigma$. $\widehat{\Gamma}$ denotes the object environment obtained by discarding the predicate information from each statement in Γ.

2. Predicates can be assigned to terms using the following natural deduction system, in which we take $A \equiv [l_i{:}B_i{}^{i \in 1..n}]$:

(Val x)
$$\frac{}{\Gamma, x{:}B{:}\sigma, \Gamma' \vdash_{\mathrm{P}} x : B : \psi}\ (\sigma \leq \psi)$$

(ω)
$$\frac{\widehat{\Gamma} \vdash_{\mathrm{O}} a : B}{\Gamma \vdash_{\mathrm{P}} a : B : \omega}$$

(Val Object)
$$\frac{\Gamma, x_j{:}A{:}\sigma \vdash_{\mathrm{P}} b_j : B_j : \phi \quad \widehat{\Gamma}, x_i{:}A \vdash_{\mathrm{O}} b_i : B_i \quad (\forall\, i \in 1..n \text{ such that } i \neq j)}{\Gamma \vdash_{\mathrm{P}} [l_i{:}\varsigma(x^A)b_i{}^{i \in 1..n}] : A : \langle l_j{:}\sigma \to \phi\rangle}\ (j \in 1..n)$$

(Val Update$_1$)
$$\frac{\Gamma \vdash_{\mathrm{P}} a : A : \sigma \quad \Gamma, y{:}A{:}\tau \vdash_{\mathrm{P}} b : B_j : \phi}{\Gamma \vdash_{\mathrm{P}} (a.l_j \Leftarrow \varsigma(y^A)b) : A : \langle l_j{:}\tau \to \phi\rangle}\ (\sigma \neq \omega, j \in 1..n)$$

(Val Select)
$$\frac{\Gamma \vdash_{\mathrm{P}} a : A : \langle l_j{:}\sigma \to \phi\rangle}{\Gamma \vdash_{\mathrm{P}} a.l_j : B_j : \phi}\ (j \in 1..n)$$

(Val Update$_2$)
$$\frac{\Gamma \vdash_{\mathrm{P}} a : A : \langle l_k{:}\phi\rangle \quad \widehat{\Gamma}, y{:}A \vdash_{\mathrm{O}} b : B_j}{\Gamma \vdash_{\mathrm{P}} (a.l_j \Leftarrow \varsigma(y^A)b) : A : \langle l_k{:}\phi\rangle}\ (j \in 1..n, k \neq j)$$

($\wedge I$)
$$\frac{\Gamma \vdash_{\mathrm{P}} a : B : \sigma_i \quad (\forall\, i \in 1..n)}{\Gamma \vdash_{\mathrm{P}} a : B : \sigma_1 \wedge \ldots \wedge \sigma_n}$$

It is clear that the predicate system shares things in common with the intersection type system of §2.1. For example, we have the rule ($\wedge I$), which is the direct analog of the ($\cap I$) rule in the strict intersection type system. There is also a rule (ω) that allows any typeable term to be assigned the predicate ω, which plays the same role here as in the intersection type systems for the λ-calculus (i.e. to type non-terminating terms). Indeed, the predicate system is essentially an intersection system for the ς-calculus. As such, we see that it also shares results in common:

1. (SUBJECT REDUCTION): If $\Gamma \vdash_{\mathrm{P}} a : A : \phi$ and $a \to b$, then $\Gamma \vdash_{\mathrm{P}} b : A : \phi$.

2. (SUBJECT EXPANSION): If $\Gamma \vdash_{\mathrm{P}} b : B : \phi$ and $\widehat{\Gamma} \vdash_{\mathrm{O}} a : B$ with $a \to b$, then $\Gamma \vdash_{\mathrm{P}} a : B : \phi$.

3. (CHARACTERISATION OF CONVERGENCE): Let a be a term and v a value, then $a \Downarrow v$ if and only if $\Gamma \vdash_{\mathrm{P}} a : A : \sigma$ for some $\sigma \neq \omega$.

In Chapter 4 we will take the ideas of this system and define a similar one for a class-based, rather than an object-based calculus.

2.3 Featherweight Java and Middleweight Java

Although Abadi and Cardelli show how the class-based approach to object oriented programming can be represented in their system, the ς-calculus seems firmly rooted in the object-based approach. Their syntax for terms takes independent objects as a primitive, and the notion of 'class' is described in terms of these. Subsequent work ([18, 8]) has followed in the footsteps of Abadi and Cardelli, but with the aim of developing formal calculi specifically modelled on Java - a class-based programming language. In these calculi, if the reader will forgive the unintentional pun, classes are first-class entities and are built into the calculus at a primitive level.

Featherweight Java (FJ) [18], so called because of its minimal composition, defines a *core* subset of the features of the Java programming language. The objective in designing FJ was to produce a formal model of a class-based language that was simpler than its predecessors [14, 20, 15], allowing a proof of type soundness to be as concise as possible. The calculus defines the notion of a class, and expresses inheritance between classes. Classes contain fields and methods, and each class has a constructor, which allows field values to be initialised when a new object is created. Inherited methods can be overridden (redefined), but there is no notion of method *overloading* (multiple methods defined with the same name, but with differing numbers and types of parameters) as there is in full Java. There are five types of expression: variables, field access, method invocation, new object creation and casting. Field assignment is purposefully omitted for simplicity. Therefore, fields can only be initialised with values on object creation, making FJ a *functional* subset of Java. This is equivalent to marking all fields with the `final` modifier in full Java.

We now give the definition of FJ syntax as it is presented in [18]:

Definition 2.3.1 (FJ SYNTAX)**.** FJ programs consist of a sequence of class definitions and an expression to be evaluated (which corresponds to the body of the `main` method in a full Java program). These are defined by the following grammar:

Class Definitions

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

Constructors

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

Method Declarations

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

Expressions

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

Values

$$\mathtt{v} ::= \mathtt{new\ C}(\overline{\mathtt{v}})$$

where `C` ranges over a set of class names, `f` ranges over a set of field names, and `x` ranges over a set of variables.

In Definition 2.3.1, the notation $\overline{\mathtt{M}}$, $\overline{\mathtt{f}}$, $\overline{\mathtt{e}}$ and $\overline{\mathtt{v}}$ denote sequences of method definitions, field names, expressions and values respectively. This notation is extended so that $\overline{\mathtt{C}}\ \overline{\mathtt{f}}$ is shorthand for the sequence $\mathtt{C}_1$ $\mathtt{f}_1, \ldots, \mathtt{C}_n\ \mathtt{f}_n$. Similarly $\overline{\mathtt{C}}\ \overline{\mathtt{x}}$ is shorthand for the sequence $\mathtt{C}_1\ \mathtt{x}_1, \ldots, \mathtt{C}_n\ \mathtt{x}_n$ and $\mathtt{this}.\overline{\mathtt{f}} = \overline{\mathtt{f}}$ represents the sequence $\mathtt{this.f}_1 = \mathtt{f}_1, \ldots, \mathtt{this.f}_n = \mathtt{f}_n$. We borrow this sequence notation for our calculus, but aim to use it in a more consistent and transparent manner.

Types in FJ consist of class names, since expressions all result in instances of some class. A notion of subtyping, $\mathtt{C} <: \mathtt{D}$, is defined which reflects the inheritance hierarchy. Thus, if a class `C` inherits from class `D`, then $\mathtt{C} <: \mathtt{D}$. Reduction in FJ is based upon substitution, as in the λ-calculus. However the notion is loosely defined in [18], with the authors simply stating that the notation $[\overline{\mathtt{d}}/\overline{\mathtt{x}}, \mathtt{e}/\mathtt{y}]\mathtt{e}_0$ stands for the result of replacing $\mathtt{x}_1$ by $\mathtt{d}_1, \ldots, \mathtt{x}_n$ by $\mathtt{d}_n$ and `y` by `e` in the expression $\mathtt{e}_0$. Reduction takes place in the context of a program (sequence of class definitions), however since the program does not change, it is assumed to be constant and so is omitted from the definition.

Definition 2.3.2 (FJ REDUCTION)**.** The reduction relation $\rightarrow$ on FJ expressions is defined by the following natural deduction system:

Computation

$$\frac{\mathit{fields}(\texttt{C}) = \overline{\texttt{C}}\,\overline{\texttt{f}}}{(\texttt{new C}(\overline{\texttt{e}})).\texttt{f}_i \to \texttt{e}_i} \qquad \text{(R-Field)}$$

$$\frac{\mathit{mbody}(\texttt{m, C}) = \overline{\texttt{x}}.\texttt{e}_0}{(\texttt{new C}(\overline{\texttt{e}})).\texttt{m}(\overline{\texttt{d}}) \to [\overline{\texttt{d}}/\overline{\texttt{x}}, \texttt{new C}(\overline{\texttt{e}})/\texttt{this}]} \qquad \text{(R-Invk)}$$

$$\frac{\texttt{C} <: \texttt{D}}{(\texttt{D})(\texttt{new C}(\overline{\texttt{e}})) \to \texttt{new C}(\overline{\texttt{e}})} \qquad \text{(R-Cast)}$$

Congruence

$$\frac{\texttt{e}_0 \to \texttt{e}'_0}{\texttt{e}_0.f \to \texttt{e}'_0.f} \qquad \text{(RC-Field)}$$

$$\frac{\texttt{e}_0 \to \texttt{e}'_0}{\texttt{e}_0.m(\overline{\texttt{e}}) \to \texttt{e}'_0.m(\overline{\texttt{e}})} \qquad \text{(RC-Invk-Recv)}$$

$$\frac{\texttt{e}_i \to \texttt{e}'_i}{\texttt{e}_0.m(\ldots, \texttt{e}_i, \ldots) \to \texttt{e}_0.m(\ldots, \texttt{e}'_i, \ldots)} \qquad \text{(RC-Invk-Arg)}$$

$$\frac{\texttt{e}_i \to \texttt{e}'_i}{\texttt{new C}(\ldots, \texttt{e}_i, \ldots) \to \texttt{new C}(\ldots, \texttt{e}'_i, \ldots)} \qquad \text{(RC-New-Arg)}$$

$$\frac{\texttt{e}_0 \to \texttt{e}'_0}{(\texttt{C})\texttt{e}_0 \to (\texttt{C})\texttt{e}'_0} \qquad \text{(RC-Cast)}$$

where *fields*(C) and *mbody*(m, C) are appropriately defined look-up functions that return a list of fields and a method body respectively.

A type system is also defined for FJ with a judgement $\Gamma \vdash \texttt{e} : \texttt{C}$, denoting that the expression e can be assigned the type C under the type assumptions on variables contained in the set Γ. However we omit the definition here and refer to the reader to [18] for the details, since it is very similar to the type system that we will define in Chapter 4.

One point that we will make about the type system, though, is that the presence of casts poses a complication for type soundness. Upcasting is a natural operation in which we allow an object to be treated as an instance of one of its supertypes, and so it seems reasonable to permit downcasting, where an object becomes a subtype of its current type. This is because a downcast may return an object to its actual type after an upcast. Moreover, an obvious restriction that one would want to impose is to disallow expressions that attempt to cast an object to an unrelated type: one that is neither a supertype nor a subtype of the expression's current type. [18] calls this a *stupid* cast. We see, however, that the cast operation is not sound in the sense that an expression that does not contain any stupid casts may reduce to one that does. Consider the following example in which classes A and B are both subclasses of Object but are otherwise unrelated:

$$(\texttt{A})(\texttt{Object})\texttt{new B}() \to (\texttt{A})\texttt{new B}()$$

The left-hand expression (before reduction) contains a valid upcast and a valid downcast, but after reduction the computation is now stuck because the reduced expression contains a stupid cast: B is *not* a subtype of A, neither is the converse true. To maintain subject reduction, stupid casts *are* typeable in FJ. However, a special property of programs is defined: that of *cast-safety*. An program is said to be cast-safe if none of the method bodies or the expression to be executed contain either downcasts or stupid casts. It is shown that reduction preserves cast-safety.

The main result for LJ is a type soundness theorem which, due to the complication introduced by the cast operation, is stated in two stages:

1. (FJ TYPE SOUNDNESS): If $\emptyset \vdash \texttt{e} : \texttt{C}$ and $\texttt{e} \rightarrow^* \texttt{e}'$ with $\texttt{e}'$ a normal form, then $\texttt{e}'$ is either a value $\texttt{v}$ with $\emptyset \vdash \texttt{v} : \texttt{D}$ and $\texttt{D} <: \texttt{C}$, or an expression containing $\texttt{(D)new C(}\overline{\texttt{e}}\texttt{)}$ where $\texttt{C} \not<: \texttt{D}$.

2. (NO TYPECAST ERRORS IN CAST-SAFE PROGRAMS): If $\texttt{e}$ is cast-safe and $\texttt{e} \rightarrow^* \texttt{e}'$ with $\texttt{e}'$ a normal form, then $\texttt{e}'$ is a value $\texttt{v}$.

Middleweight Java (MJ) [8], inspired by FJ, also defines a calculus that is a pure subset of Java. Unlike FJ however, the authors of [8] were as much interested in the *operational* issues of the Java language as much as its type system. For this reason they find FJ, as a *functional* calculus, too simplistic and define MJ as an *imperative* calculus, able to express such concepts as object identity (comparison), field assignment, `null` pointers and sequences of statements (possibly resulting in side-effects) with local variables and block-structured scoping. The reduction semantics of MJ are based around the concept of a *store* (or heap) where objects persist, and a stack of execution frames. We present the syntax of MJ so that the reader may get a feel for the complexity that it introduces over FJ, and because we will borrow some of the notation for the definition of our calculus. However we will not examine MJ in any further detail (e.g. its operational semantics) since we do not use most of its features in our calculus.

Definition 2.3.3 (MJ SYNTAX)**.** MJ programs are defined by the following grammar, where C ranges over a set of class names:

Program			
p	::=	$cd_1 \dots cd_n; s_1 \dots s_n$	
Class Definition			
cd	::=	$\texttt{class}\ C\ \texttt{extends}\ C$	
		$\{\ fd_1 \dots fd_k$	
		cnd	
		$md_1 \dots md_n\ \}$	
Field Definition			
fd	::=	$C\ f;$	
Constructor Definition			
cnd	::=	$C(C_1\ x_1, \dots, C_j\ x_j)\ \{\ \texttt{super}\ (e_1, \dots, e_k);\ s_1 \dots s_n\ \}$	
Method Definition			
md	::=	$\tau\ m(C_1\ x_1, \dots, C_n\ x_n)\ \{\ s_1 \dots s_k\ \}$	
Return Type			
τ	::=	$C \mid \texttt{void}$	
Expression			
e	::=	x	Variable
		$\mid \texttt{null}$	Null
		$\mid e.f$	Field access
		$\mid (C)e$	Cast
		$\mid pe$	Promotable Expression
Promotable Expression			
pe	::=	$e.m(e_1, \dots, e_k)$	Method invocation
		$\mid \texttt{new}\ C(e_1, \dots, e_k)$	Object creation
Statement			
s	::=	$;$	No-op
		$\mid pe;$	Promoted Expression
		$\mid \texttt{if}\ (e == e)\ \{\ s_1 \dots s_k\ \}\ \texttt{else}\ \{\ s_{k+1} \dots s_n\ \}$	Conditional
		$\mid e.f = e;$	Field assignment
		$\mid C\ x;$	Local variable declaration

$\mid x = e;$	Variable assignment
$\mid \texttt{return}\ e;$	Return
$\mid \{\ s_1 \ldots s_n\ \}$	Block

We have now completed our survey of calculi relevant to our current study, and will begin to define a calculus in the next chapter to which we can apply the principals of intersection types and the predicate system of [5].

Chapter 3

A Class-based Calculus: Lightweight Java

In this chapter, we formally define the calculus that we study. It is based on aspects of both Featherweight Java and Middleweight Java, and so to continue with the theme we have named it *Lightweight Java* (LJ). We retain the functional nature of FJ, but add some features from MJ (namely field assignment expressions and `null` objects) which do not conflict with this.

The following notational conventions will be used within the remainder of this document:

Notation (SEQUENCES)**.** We use the following notation for abbreviating and manipulating sequences of elements

1. A sequence of n elements $a_1, \ldots, a_n$ is denoted by $\overline{a}_n$. The subscript can be omitted when the exact number of elements in the sequence is not relevant. Both comma-separated and space-separated sequences may be abbreviated in this way.

2. We write $a \in \overline{a}_n$ whenever there exists some $i \in \{1, \ldots, n\}$ such that $a_i = a$. Similarly, we write $a \notin \overline{a}_n$ whenever there does *not* exist an $i \in \{1, \ldots, n\}$ such that $a_i = a$. Again, the subscript n may be omitted as implicit.

3. $\overline{a}_n \cdot \overline{a'}_m = a_1, \ldots, a_n, a'_1, \ldots, a'_m$ denotes the concatenation of two sequences.

4. The empty sequence is denoted by ϵ

5. We use the special sequence $\overline{n}$ (where n is a natural number) to represent the list $1, \ \ldots \ , n$.

Notation (FUNCTIONS)**.** If F is a partial function defined over n arguments, then:

- $F(\text{arg}_1, \ldots, \text{arg}_n)\Downarrow$ denotes that F is defined on the arguments $\text{arg}_1, \ldots, \text{arg}_n$.
- $F(\text{arg}_1, \ldots, \text{arg}_n)\Uparrow$ denotes that F is *not* defined on the arguments $\text{arg}_1, \ldots, \text{arg}_n$.

3.1 Language Definition

Definition 3.1.1 (IDENTIFIERS)**.** We define the following sets of identifiers:

1. C, D range over a set of class names, $\mathbb{CLASS\text{-}NAME}$, which includes the distinguished element `Object`.

2. f ranges over a set of field identifiers, $\mathbb{FIELD\text{-}ID}$.

3. m ranges over a set of method names, $\mathbb{METHOD\text{-}NAME}$.

4. l ranges over the union of the set of field identifiers and method names, which we call the set of *class member labels*.

5. x ranges over a set of variables, $\mathbb{VARIABLE}$, which includes the special variable `this`.

As in FJ and MJ (and indeed full Java [17]), types are embedded within the syntax of the language itself, allowing the programmer to explicitly specify the type of each field and method. Thus, in order to define the syntax of our calculus, we must first define types.

Definition 3.1.2 (CLASS & METHOD TYPES).

1. The set of types that can be assigned to LJ expressions is denoted by $\mathbb{TYPE}_{\mathrm{C}}$ and we call the types in this set *class* types, corresponding to the intuition that each expression results in an object which is an instance of some class. The set of class types is identical to the set of class names and, as such, we will also use C and D to range over these types. As will be seen in the syntax definition below, we also use class types as annotations in field definitions since fields are used to store (object) values.

2. We define a set of types, $\mathbb{TYPE}_{\mathrm{m}}$, encapsulating information pertaining to the behaviour of methods. These *method types* are defined by the following grammar:

   Method type
   $$\mu \quad ::= \quad \mathrm{C}_1, \ldots, \mathrm{C}_n \to \mathrm{D}$$

   Thus, each method takes a sequence of arguments of types $\mathrm{C}_1, \ \ldots \ , \mathrm{C}_n$, and returns a result of type D.

Definition 3.1.3 (PROGRAM SYNTAX). The following syntactic elements comprise LJ programs:

1. We define a set of *expressions*, $\mathbb{EXPR}$. We also define a subset of expressions, $\mathbb{OBJECT}$, which is the set of *objects*:

   Expression
   $$\begin{array}{lll} \mathrm{e} & ::= & x \mid (\mathrm{C})\,\texttt{null} \mid \mathrm{e}.f \mid \mathrm{e}.f = \mathrm{e}' \mid \\ & & \mathrm{e}.m(\mathrm{e}_1, \ldots, \mathrm{e}_n) \mid \texttt{new}\ \mathrm{C}(\mathrm{e}_1, \ldots, \mathrm{e}_n) \qquad (n \geq 0) \end{array}$$

   Object
   $$\begin{array}{lll} \mathrm{o} & ::= & x \mid (\mathrm{C})\,\texttt{null} \mid \\ & & \texttt{new}\ \mathrm{C}(\mathrm{o}_1, \ldots, \mathrm{o}_n) \qquad (n \geq 0) \end{array}$$

2. *Classes* are defined by the following grammar:

   Field definition
   $$\mathrm{fd} \quad ::= \quad \mathrm{C}\ f$$

   Method definition
   $$\mathrm{md} \quad ::= \quad \mathrm{D}\ m(\mathrm{C}_1\ x_1, \ldots, \mathrm{C}_n\ x_n)\ \{\ \mathrm{e}\ \} \qquad (n \geq 0)$$

   Class definition
   $$\mathrm{cd} \quad ::= \quad \texttt{class}\ \mathrm{C}\ \texttt{extends}\ \mathrm{C}'\ \{\ \mathrm{fd}_1\ \ldots\ \mathrm{fd}_k\ \mathrm{md}_1\ \ldots\ \mathrm{md}_n\ \} \qquad (k, n \geq 0)$$

3. LJ Programs consist of an *execution context* (which is a sequence of class definitions), and an expression that is evaluated when the program is run:

   Execution Context
   $$\chi \quad ::= \quad \mathrm{cd}_1\ \ldots\ \mathrm{cd}_n \qquad (n \geq 0)$$

   Program
   $$\mathrm{P} \quad ::= \quad (\chi, \mathrm{e})$$

   $\mathbb{PROGRAM}$ denotes the set of all programs, and $\mathbb{CONTEXT}$ denotes the set of all execution contexts.

We now discuss these various syntactic elements. Expressions may create objects that conform to a specified class template using the `new` keyword. They may also invoke methods, and retrieve or assign field values. Expressions may also refer to variables (method parameters) and the null value.

Classes contain a list of fields and a list of methods, the types of which must be declared. Methods may take multiple arguments, and method bodies consist of a single expression. Classes may also inherit from one another, meaning that they share field and method definitions. There is a lack of symmetry between the structure of field definitions and method definitions: in a field definition, the type of the field precedes the field identifier, whereas the sequence of parameter types comprising the method type is mixed in with the sequence of formal parameters. However this is an intentional decision, motivated by the desire to have the calculus conform to its predecessors (LJ and MJ), as well as its namesake language.

Note that the syntax of Defintion 3.1.3 does not disallow the *redeclaration* of any methods and fields that have be defined in a superclass. In §3.2 we will specify that for an execution context to be well formed (and so in turn for expressions to be typeable), classes must not redeclare fields in this way. We will also specify that if methods are redeclared, then the method type must match that of the superclass, although the names of the formal parameters may differ. We will not place any restrictions on the body of the redeclared method, thus allowing a limited form of method *override*.

Unlike in FJ and MJ, and indeed full Java itself, we do not include object constructor methods in the language definition. Full Java allows objects to be created in multiple ways, thereby requiring the ability to define multiple constructor methods. Both FJ and MJ enforce a single constructor method for each class to which initial values for all fields must be passed as parameters. There is no loss of generality in this approach and so we adopt it for LJ. However, in FJ and MJ, constructor methods are made explicit and must appear in the method definition list of each class. This constructor method is distinguished from the other methods through the use of separate typing rules. Since there can only be a single constructor, we feel that this is an unnecessary complication of the language syntax, and so make the constructor method *implicit* by requiring in the type rule for `new` expressions that the types of the sub-expressions appearing in the object creation construct match the types for the sequence of fields defined by the class of the object being created.

Furthermore, we have chosen to omit cast expressions from our language. It appears that casts were included in FJ in order to support the compilation of FGJ programs to FJ [18, §3]. Since that is not an objective of the current work, and the presence of downcasts makes the system unsound (in the sense that well-typed expressions can get stuck), they are omitted. Upcasts are replaced by subsumption rules in the type system. It is necessary to point out at this stage the unusual syntax that we have defined for `null` expressions. We have borrowed the traditional syntax for denoting casts but intend it instead to merely act as a tag indicating the desired type that the `null` object should have. This choice has been made in order to be able to define a simple type inference algorithm, in which all expressions have a *most specific* type, in the sense that any other type assignable to that expression (in a given typing environment) will be a supertype of this most specific one.

Definition 3.1.4 (SYNTAX LOOK-UP FUNCTIONS)**.** We define the following look-up functions to retrieve the various syntactic elements of a program:

1. The following three functions retrieve the names of a class, a method and a field from their respective definitions:

$$\begin{array}{rcll} \textsc{cname}(\mathrm{cd}) & \triangleq & \mathrm{C} & \text{where } \mathrm{cd} = \texttt{class}\ \mathrm{C}\ \texttt{extends}\ \mathrm{C}'\ \{\ \overline{\mathrm{fd}}\ \overline{\mathrm{md}}\ \} \\ \textsc{mname}(\mathrm{md}) & \triangleq & m & \text{where } \mathrm{md} = \mathrm{D}\ m(\mathrm{C}_1\ x_1, \ldots, \mathrm{C}_n\ x_n) \\ \textsc{fname}(\mathrm{fd}) & \triangleq & f & \text{where } \mathrm{fd} = \mathrm{C}\ f \end{array}$$

2. The class table, Δ, is a partial map from execution contexts and class names to class definitions:

$$\Delta(\chi, \mathrm{C}) \triangleq \begin{cases} \mathrm{cd} & \text{if } \textsc{cname}(\mathrm{cd}) = \mathrm{C}\ \&\ \mathrm{cd} \in \chi \\ \text{Undefined} & \text{otherwise} \end{cases}$$

   We specify explicitly that the class table should be undefined on the special class `Object`, since this class should only serve as the root of the class hierarchy and contain no fields and methods:

$$\Delta(\chi, \texttt{Object}) \triangleq \text{Undefined}$$

3. The SUPERCLASS function is a partial map from execution contexts and class names to class names, returning the direct superclass of a given class within the given context:

$$\textsc{superclass}(\chi, \mathrm{C}) \triangleq \begin{cases} \mathrm{C}' & \text{if } \Delta(\chi, \mathrm{C}) = \texttt{class}\ \mathrm{C}\ \texttt{extends}\ \mathrm{C}'\ \{\ \overline{\mathrm{fd}}\ \overline{\mathrm{md}}\ \} \\ \text{Undefined} & \text{otherwise} \end{cases}$$

4. The list of fields belonging to a class C in an execution context χ is given by the following function:

$$\mathcal{F}(\chi, \mathrm{C}) \triangleq \begin{cases} \mathcal{F}(\chi, \mathrm{C}') \cdot \overline{f}_n & \text{if } \Delta(\chi, \mathrm{C}) = \texttt{class}\ \mathrm{C}\ \texttt{extends}\ \mathrm{C}'\ \{\ \overline{\mathrm{fd}}_n\ \overline{\mathrm{md}}\ \} \\ & \text{and } f_i = \textsc{fname}(\mathrm{fd}_i) \text{ for each } i \in \overline{n} \\ \epsilon & \text{otherwise} \end{cases}$$

Thus, the sequence returned by this function contains not only the fields declared in the specified class, but all the fields that it inherits from its superclasses. Note that we have defined this function to return a list, rather than a set, since the order of the fields is important. When typing `new` expressions, we must make sure that values are given for *all* the fields belonging to a class, and that they are given in the correct order so that field selection and update expressions reduce correctly.

5. The function $\mathcal{M}$ returns a set of method names, corresponding to the methods declared and inherited by a given class. It is defined as follows:

$$\mathcal{M}(\chi, \mathrm{C}) \triangleq \begin{cases} \mathcal{M}(\chi, \mathrm{C}') \cup \{\textsc{mname}(\mathrm{md}) \mid \mathrm{md} \in \overline{\mathrm{md}}\} & \text{if } \Delta(\chi, \mathrm{C}) = \texttt{class}\ \mathrm{C}\ \texttt{extends}\ \mathrm{C}'\ \{\ \overline{\mathrm{fd}}\ \overline{\mathrm{md}}\ \} \\ \emptyset & \text{otherwise} \end{cases}$$

Note that we have defined this function to return a set rather than a list since, unlike fields, the order of methods is *not* important. Also, the set returned by this function contains not only the names of methods defined in the specified class, but the names of all methods that it inherits from its superclasses. We allow this set to be notationally referred to as a sequence, and its elements to be indexed and subscripted in the same way, in order to increase the readability of the notation.

6. The function $\textsc{mbody}$, when given an execution context χ, class name C and method name m, returns a tuple $(\overline{x}, \mathrm{e})$, consisting of a sequence of the method's formal parameters and the method body:

$$\textsc{mbody}(\chi, \mathrm{C}, m) \triangleq \begin{cases} (\overline{x}_n, \mathrm{e}) & \text{if } \Delta(\chi, \mathrm{C}) = \texttt{class}\ \mathrm{C}\ \texttt{extends}\ \mathrm{C}'\ \{\ \overline{\mathrm{fd}}\ \overline{\mathrm{md}}\ \} \\ & \&\ \mathrm{C}_0\ m(\mathrm{C}_1\ x_1, \ldots, \mathrm{C}_n\ x_n)\ \{\ \mathrm{e}\ \} \in \overline{\mathrm{md}} \\ \textsc{mbody}(\chi, \mathrm{C}', m) & \text{if } \Delta(\chi, \mathrm{C}) = \texttt{class}\ \mathrm{C}\ \texttt{extends}\ \mathrm{C}'\ \{\ \overline{\mathrm{fd}}\ \overline{\mathrm{md}}\ \} \\ & \&\ \mathrm{C}_0\ m(\mathrm{C}_1\ x_1, \ldots, \mathrm{C}_n\ x_n)\ \{\ \mathrm{e}\ \} \notin \overline{\mathrm{md}} \\ \text{Undefined} & \text{if } \Delta(\chi, \mathrm{C})\Uparrow \end{cases}$$

7. The function $\textsc{vars} : \mathbb{EXPR} \rightarrow \wp(\mathbb{VARIABLE})$ returns the set of all variables used in a given expression. It is defined inductively as follows:

$$\begin{aligned} \textsc{vars}((\mathrm{C})\ \texttt{null}) &= \emptyset \\ \textsc{vars}(x) &= \{x\} \\ \textsc{vars}(\mathrm{e}.f) &= \textsc{vars}(\mathrm{e}) \\ \textsc{vars}(\mathrm{e}_0.f = \mathrm{e}_1) &= \textsc{vars}(\mathrm{e}_0) \cup \textsc{vars}(\mathrm{e}_1) \\ \textsc{vars}(\mathrm{e}_0.m(\overline{\mathrm{e}}_n)) &= \textsc{vars}(\mathrm{e}_0) \cup \textsc{vars}(\mathrm{e}_1) \cup \ \ldots\ \cup \textsc{vars}(\mathrm{e}_n) \\ \textsc{vars}(\texttt{new}\ \mathrm{C}(\overline{\mathrm{e}}_n)) &= \textsc{vars}(\mathrm{e}_1) \cup \ \ldots\ \cup \textsc{vars}(\mathrm{e}_n) \end{aligned}$$

We now define look-up functions which allow us to extract the type information that is defined in a given class:

Definition 3.1.5 (MEMBER TYPE LOOKUP)**.** The *field table* Δ_f and *method table* Δ_m are functions which return type information about the elements of a given class within an execution context:

$$\begin{aligned} \Delta_\mathrm{f} &\in (\mathbb{CONTEXT} \times \mathbb{CLASS\text{-}NAME} \times \mathbb{FIELD\text{-}ID}) \rightarrow \mathbb{TYPE}_\mathrm{C} \\ \Delta_\mathrm{m} &\in (\mathbb{CONTEXT} \times \mathbb{CLASS\text{-}NAME} \times \mathbb{METHOD\text{-}NAME}) \rightarrow \mathbb{TYPE}_\mathrm{m} \end{aligned}$$

These functions allow us to retrieve the types of any given field f or method m declared in a particular class C in the context χ:

$$\Delta_{\mathrm{f}}(\chi, \mathrm{C}, f) = \begin{cases} \mathrm{D} & \text{if } \Delta(\chi, \mathrm{C}) = \texttt{class}\ \mathrm{C}\ \texttt{extends}\ \mathrm{C}'\ \{\ \overline{\mathrm{fd}}\ \overline{\mathrm{md}}\ \} \\ & \&\ \mathrm{D}\ f \in \overline{\mathrm{fd}} \\ \Delta_{\mathrm{f}}(\chi, \mathrm{C}', f) & \text{if } \Delta(\chi, \mathrm{C}) = \texttt{class}\ \mathrm{C}\ \texttt{extends}\ \mathrm{C}'\ \{\ \overline{\mathrm{fd}}\ \overline{\mathrm{md}}\ \} \\ & \&\ \mathrm{D}\ f \notin \overline{\mathrm{fd}} \\ \text{Undefined} & \text{if } \Delta(\chi, \mathrm{C})\Uparrow \end{cases}$$

$$\Delta_{\mathrm{m}}(\chi, \mathrm{C}, m) = \begin{cases} \overline{\mathrm{C}}_n \to \mathrm{D} & \text{if } \Delta(\chi, \mathrm{C}) = \texttt{class}\ \mathrm{C}\ \texttt{extends}\ \mathrm{C}'\ \{\ \overline{\mathrm{fd}}\ \overline{\mathrm{md}}\ \} \\ & \&\ \mathrm{D}\ m(\mathrm{C}_1\ x_1, \ldots, \mathrm{C}_n\ x_n)\ \{\ \mathrm{e}\ \} \in \overline{\mathrm{md}} \\ \Delta_{\mathrm{m}}(\chi, \mathrm{C}', m) & \text{if } \Delta(\chi, \mathrm{C}) = \texttt{class}\ \mathrm{C}\ \texttt{extends}\ \mathrm{C}'\ \{\ \overline{\mathrm{fd}}\ \overline{\mathrm{md}}\ \} \\ & \&\ \mathrm{D}\ m(\mathrm{C}_1\ x_1, \ldots, \mathrm{C}_n\ x_n)\ \{\ \mathrm{e}\ \} \notin \overline{\mathrm{md}} \\ \text{Undefined} & \text{if } \Delta(\chi, \mathrm{C})\Uparrow \end{cases}$$

We also define these lookup functions so that the special class `Object` does not have any fields or methods. For all contexts χ, field identifiers f and method names m:

$$\begin{aligned} \Delta_{\mathrm{f}}(\chi, \texttt{Object}, f) &= \text{Undefined} \\ \Delta_{\mathrm{m}}(\chi, \texttt{Object}, m) &= \text{Undefined} \end{aligned}$$

We say that C *extends* C′ in the context χ when $\Delta(\chi, \mathrm{C}) = \texttt{class}\ \mathrm{C}\ \texttt{extends}\ \mathrm{C}'\ \{\ \overline{\mathrm{fd}}\ \overline{\mathrm{md}}\ \}$. We say that C is a *subclass* of C′ in the context χ when C extends C′, or there is a non-empty sequence of classes $\mathrm{C}_1, \ldots, \mathrm{C}_n$ in χ such that C extends C_1, each C_i extends C_{i+1} for all $i \in \{1, \ldots, n-1\}$, and C_n extends C′. We formalise this notion by defining a family of relations, with each relation representing subclassing in a particular context.

Definition 3.1.6 (SUBCLASS RELATION)**.** The function $\prec$ returns the subclass relation for a given execution context:

$$\prec\ \in \mathbb{CONTEXT} \to \wp(\mathbb{CLASS\text{-}NAME} \times \mathbb{CLASS\text{-}NAME})$$

Then, for all contexts, χ, the corresponding subclass relation is defined as the smallest transitive relation satisfying the following condition:

$$\textsc{superclass}(\chi, \mathrm{C}) = \mathrm{C}' \Rightarrow (\mathrm{C}, \mathrm{C}') \in \prec(\chi)$$

We will use the abbreviation $\prec_\chi$ to denote $\prec(\chi)$, and use infix notation; thus $\mathrm{C} \prec_\chi \mathrm{C}'$ represents that $(\mathrm{C}, \mathrm{C}') \in \prec(\chi)$.

Using the subclass relation, we now define a relation over *types*. The subtype relation is simply the reflexive projection of the subclass relation onto types.

Definition 3.1.7 (SUBTYPES)**.** As for subclassing, we define a family of *subtype* relations, and a total function from programs to this set of relations:

$$<:\ \in \mathbb{CONTEXT} \to \wp(\mathbb{TYPE}_{\mathrm{C}} \times \mathbb{TYPE}_{\mathrm{C}})$$

For each context χ, its corresponding subtype relation $<:(\chi)$ is defined as the smallest partial order satisfying the following condition:

$$\mathrm{C} \prec_\chi \mathrm{C}' \Rightarrow (\mathrm{C}, \mathrm{C}') \in\ <:(\chi)$$

Again, we will use the abbreviation $<:_\chi$ to denote $<:(\chi)$ and use infix notion; thus $\mathrm{C} <:_\chi \mathrm{C}'$ represents that $(\mathrm{C}, \mathrm{C}') \in\ <:(\chi)$.

Finally, we define *validity* of a type. We say that a type is *valid* with respect to an execution context χ, written $\vdash_\chi \mathrm{C}$, when the corresponding class is defined in the context.

Definition 3.1.8 (VALID TYPES)**.** We define type validity through the following judgements:

$$\frac{}{\vdash_\chi \texttt{Object}} \qquad \frac{}{\vdash_\chi \mathrm{C}}\ (\Delta(\chi, \mathrm{C})\Downarrow)$$

3.2 Well-Formed Contexts

In this section, we define the conditions necessary for an execution context to be *well formed*. Well formedness of a context is itself a necessary condition for the correct reduction (execution) of LJ programs.

Definition 3.2.1 (WELL-FORMED CONTEXT)**.** The notation $\vdash \chi$ denotes a well formed execution context. An execution context χ is well formed if, and only if, it satisfies the following conditions:

1. There are no duplicate class definitions:

$$\chi = \overline{\mathrm{cd}}_n \Rightarrow \forall\, i, j \in \overline{n}\ [i \neq j \Rightarrow \textsc{cname}(\mathrm{cd}_i) \neq \textsc{cname}(\mathrm{cd}_j)]$$

2. The class hierarchy is *acyclic*; that is, there are no two classes defined in the execution context such that they are both subclasses of one another:

$$\neg\exists\, \mathrm{C}, \mathrm{D}\ [\Delta(\chi, \mathrm{C})\!\downarrow\ \&\ \Delta(\chi, \mathrm{D})\!\downarrow\ \Rightarrow \mathrm{C} \prec_\chi \mathrm{D}\ \&\ \mathrm{D} \prec_\chi \mathrm{C}]$$

   Note that this also precludes a class inheriting from itself, i.e. the subclass relation must be *antireflexive*. We require this property to ensure that the field and method list look-up functions are well defined for classes that are defined in the class table.

3. All fields defined in a particular branch of the class hierarchy are uniquely named, so there are no duplicate field definitions or field hiding:

$$\forall\, \mathrm{C}\ [\Delta(\chi, \mathrm{C})\!\downarrow\ \&\ \mathcal{F}(\chi, \mathrm{C}) = \overline{f}_n \Rightarrow \neg\exists\, i, j \in \overline{n}\ [i \neq j \Rightarrow f_i = f_j]]$$

4. There are no duplicate method declarations within a given class, and the types of any overridden methods must match.

$$\begin{array}{ll} \forall\, \mathrm{C}\ [& \Delta(\chi, \mathrm{C}) = \texttt{class}\ \mathrm{C}\ \texttt{extends}\ \mathrm{C}'\ \{\overline{\mathrm{fd}}\ \overline{\mathrm{md}}_n\} \\ & \Rightarrow \neg\exists\, i, j \in \overline{n}\ [i \neq j \Rightarrow \textsc{mname}(\mathrm{md}_i) = \textsc{mname}(\mathrm{md}_j)] \quad] \\ \forall\, \mathrm{C}, \mathrm{C}', m\ [& \Delta_{\mathrm{m}}(\chi, \mathrm{C}, m) = \overline{\mathrm{C}}_{n_1} \to \mathrm{D}_1\ \&\ \Delta_{\mathrm{m}}(\chi, \mathrm{C}', m) = \overline{\mathrm{C}'}_{n_2} \to \mathrm{D}_2 \\ & \&\ \textsc{superclass}(\chi, \mathrm{C}) = \mathrm{C}' \Rightarrow n_1 = n_2\ \&\ \mathrm{D}_1 = \mathrm{D}_2\ \&\ \forall\, i \in \overline{n_1}\ [\mathrm{C}_i = \mathrm{C}'_i] \quad] \end{array}$$

5. The special variable `this` must not appear as a parameter in any method definition:

$$\forall\, \mathrm{C}, m\ [\textsc{mbody}(\chi, \mathrm{C}, m) = (\overline{x}, \mathrm{e}) \Rightarrow \texttt{this} \notin \overline{x}]$$

6. All types declared in field and method types must be valid types with respect to the execution context, as must all classes that are inherited from:

$$\begin{array}{rcl} \forall\, \mathrm{C}, f\ [\Delta_{\mathrm{f}}(\chi, \mathrm{C}, f) = \mathrm{D} & \Rightarrow & \vdash_\chi \mathrm{D}] \\ \forall\, \mathrm{C}, m\ [\Delta_{\mathrm{m}}(\chi, \mathrm{C}, m) = \overline{\mathrm{C}}_n \to \mathrm{D} & \Rightarrow & \vdash_\chi \mathrm{D}\ \&\ \forall\, i \in \overline{n}\ [\vdash_\chi \mathrm{C}_i]] \\ \forall\, \mathrm{C}\ [\textsc{superclass}(\chi, \mathrm{C}) = \mathrm{D} & \Rightarrow & \vdash_\chi \mathrm{D}] \end{array}$$

   In other words, this condition ensures that all elements of the program correspond to a well defined class.

Note, that we could have specified that a well formed execution context does not define the class `Object`, since we reserve this class name as denoting a special empty object. This condition is not strictly necessary however, since we have defined the class table to be undefined on the `Object` class. Such a condition would serve merely as a sanity check for the programmer, ensuring that they had not defined other classes in such a way as to rely on properties of the `Object` class that do not exist.

Property 3.2.2 (TYPE CONSISTENCY)**.** We note that well formed execution contexts exhibit the following consistency properties:

1. The type of an inherited field in a subclass is consistent with its type in the superclass:

$$\vdash \chi \;\&\; \Delta_{\mathrm{f}}(\chi, \mathrm{C}, f) = \mathrm{C}' \;\&\; \Delta_{\mathrm{f}}(\chi, \mathrm{D}, f) = \mathrm{D}' \;\&\; \mathrm{C} <:_{\chi} \mathrm{D} \Rightarrow \mathrm{C}' = \mathrm{D}'$$

2. The type of an inherited or overridden method in a subclass are consistent with its type in the superclass:

$$\vdash \chi \;\&\; \Delta_{\mathrm{m}}(\chi, \mathrm{C}, m) = \overline{\mathrm{C}}_{n_1} \to \mathrm{C}_0 \;\&\; \Delta_{\mathrm{m}}(\chi, \mathrm{D}, m) = \overline{\mathrm{D}}_{n_2} \to \mathrm{D}_0 \;\&\; \mathrm{C} <:_{\chi} \mathrm{D} \Rightarrow$$
$$\mathrm{C}_0 = \mathrm{D}_0 \;\&\; n_1 = n_2 \;\&\; \forall\, i \in \overline{n}\, [\mathrm{C}_i = \mathrm{D}_i]$$

Proof. From Definition 3.1.6, we see that $\mathrm{C} <:_{\chi} \mathrm{D}$ implies that either $\mathrm{C} = \mathrm{D}$ or $\mathrm{C} \prec_{\chi} \mathrm{D}$. The results then follow immediately, or by induction on the derivation of $\mathrm{C} \prec_{\chi} \mathrm{D}$, using the properties of well formed execution contexts and the definition of the field table Δ_{f} and method table Δ_{m}. We show the case for fields (1).

From the definitions of Δ_{f} and $\mathcal{F}$ we see that if $\Delta_{\mathrm{f}}(\chi, \mathrm{C}, f) = \mathrm{C}'$, then $f \in \mathcal{F}(\chi, \mathrm{C})$. Part (3) of Definition 3.2.1 asserts that each field in $\mathcal{F}(\chi, \mathrm{C})$ is unique. Then, if $\mathrm{C} = \mathrm{D}$ it follows immediately that $\Delta_{\mathrm{f}}(\chi, \mathrm{C}, f) = \Delta_{\mathrm{f}}(\chi, \mathrm{D}, f)$ since the types C' and D' must come from the same field definition. If $\mathrm{C} \prec_{\chi} \mathrm{D}$, then by induction on the derivation of $\mathrm{C} \prec_{\chi} \mathrm{D}$ it follows that there is a sequence of classes $\overline{\mathrm{C}}_n$ with $n \geq 2$ such that $\mathrm{C}_1 = \mathrm{C}$ and $\mathrm{C}_n = \mathrm{D}$ with $\textsc{superclass}(\mathrm{C}_i) = \mathrm{C}_{i+1}$ for each $i \in \overline{n}$. Then, since $f \in \mathcal{F}(\chi, \mathrm{D})$ it follows by the definition of Δ_{f} (Definition 3.1.4) and part (3) of Definition 3.2.1 that the field f is not defined in any of the classes $\mathrm{C} = \mathrm{C}_1, \ldots, \mathrm{C}_{n-1}$, and so the type returned by $\Delta_{\mathrm{f}}(\chi, \mathrm{C}, f)$ is the type declared in the field definition for f in class D. Therefore, $\mathrm{C}' = \mathrm{D}'$. $\square$

3.3 Reduction

In this section, we define a reduction relation, $\to$, on LJ programs. The reduction rules given below are a simple modification of those in §2.3 of [18], and are defined through the notion of *substitution*:

Definition 3.3.1 (substitution). 1. A *substitution* is a construction of the form $[\mathrm{e}/x]$ where e is an LJ expression, and x is a variable. We define the *application* of a substitution to an expression, $(\mathrm{e})[\mathrm{e}'/x]$, inductively as follows:

$$\begin{array}{rcll}
((\mathrm{C})\,\texttt{null})[\mathrm{e}'/x] & = & (\mathrm{C})\,\texttt{null} & \\
(x)[\mathrm{e}'/x] & = & \mathrm{e}' & \\
(y)[\mathrm{e}'/x] & = & y & \text{if } y \neq x \\
(\mathrm{e}.f)[\mathrm{e}'/x] & = & (\mathrm{e}[\mathrm{e}'/x]).f & \\
(\mathrm{e}.f = \mathrm{e}'')[\mathrm{e}'/x] & = & (\mathrm{e}[\mathrm{e}'/x]).f = (\mathrm{e}''[\mathrm{e}'/x]) & \\
(\mathrm{e}_0.m(\mathrm{e}_1, \ldots, \mathrm{e}_n))[\mathrm{e}'/x] & = & (\mathrm{e}_0[\mathrm{e}'/x]).m((\mathrm{e}_1[\mathrm{e}'/x]), \ldots, (\mathrm{e}_n[\mathrm{e}'/x])) & \\
(\texttt{new}\;\mathrm{C}(\mathrm{e}_1, \ldots, \mathrm{e}_n))[\mathrm{e}'/x] & = & \texttt{new}\;\mathrm{C}((\mathrm{e}_1[\mathrm{e}'/x]), \ldots, (\mathrm{e}_n[\mathrm{e}'/x])) &
\end{array}$$

Thus, the result of applying the term substitution $[\mathrm{e}'/x]$ to expression e is the term e with all occurrences of the variable x replaced by the expression e'.

2. A sequence of n distinct substitutions $(\ldots\ ((\mathrm{e})[\mathrm{e}_1/x_1])\ \ldots)[\mathrm{e}_n/x_n]$ can be abbreviated by the notation $(\mathrm{e})[\mathrm{e}_1/x_1, \ldots, \mathrm{e}_n/x_n]$.

3. A *variable substitution* is a substitution $[\mathrm{e}/x]$ in which the expression e is itself a variable. When such a substitution is applied to an expression, the result is that all occurrences of one variable are replaced by another.

Corollary 3.3.2. A corollary of definition 3.3.1 is that if $[y/x]$ is a variable substitution and $[\mathrm{e}/y]$ is a term substitution, with $y \notin \textsc{vars}(\mathrm{e}') \setminus \{x\}$ then

$$\mathrm{e}'[y/x][\mathrm{e}/y] = \mathrm{e}'[\mathrm{e}/x]$$

We now define the reduction relation itself.

Definition 3.3.3 (REDUCTION). The one-step reduction relation $\rightarrow$ is a relation between programs:

$$\rightarrow \; \in \wp(\mathbb{PROGRAM} \times \mathbb{PROGRAM})$$

Since the execution context remains the same for all programs related by $\rightarrow$, to ease readability we again use a subscript notation, so that $\mathrm{e} \rightarrow_\chi \mathrm{e}'$ indicates that $(\chi, \mathrm{e}) \rightarrow (\chi, \mathrm{e}')$. The one-step reduction relation is defined by the following natural deduction system:

$$[\textsc{r-fld}] \quad \frac{}{(\texttt{new}\ \mathrm{C}(\overline{\mathrm{e}}_n).f_i) \rightarrow_\chi \mathrm{e}_i} \; (\mathcal{F}(\chi, \mathrm{C}) = \overline{f}_n \;\&\; i \in \overline{n})$$

$$[\textsc{r-ass}] \quad \frac{}{(\texttt{new}\ \mathrm{C}(\overline{\mathrm{e}}_n)).f_j = \mathrm{e}'_j \rightarrow_\chi \texttt{new}\ \mathrm{C}(\overline{\mathrm{e}'}_n)} \left(\begin{array}{c} \mathcal{F}(\chi, \mathrm{C}) = \overline{f}_n \;\&\; j \in \overline{n} \\ \forall i \in \overline{n}\ [i \neq j \Rightarrow \mathrm{e}'_i = \mathrm{e}_i] \end{array} \right)$$

$$[\textsc{r-invk}] \quad \frac{}{(\texttt{new}\ \mathrm{C}(\overline{\mathrm{e}})).m(\overline{\mathrm{e}'}_n) \rightarrow_\chi \mathrm{e}[\mathrm{e}'_1/x_1, \ldots, \mathrm{e}'_n/x_n, \texttt{new}\ \mathrm{C}(\overline{\mathrm{e}})/\texttt{this}]} \; (\textsc{mbody}(\chi, \mathrm{C}, m) = (\overline{x}_n, \mathrm{e}))$$

$$[\textsc{rc-fld}] \quad \frac{\mathrm{e} \rightarrow_\chi \mathrm{e}'}{\mathrm{e}.f \rightarrow_\chi \mathrm{e}'.f}$$

$$[\textsc{rc-ass}_1] \quad \frac{\mathrm{e} \rightarrow_\chi \mathrm{e}''}{(\mathrm{e}.f = \mathrm{e}') \rightarrow_\chi (\mathrm{e}''.f = \mathrm{e}')}$$

$$[\textsc{rc-ass}_2] \quad \frac{\mathrm{e}' \rightarrow_\chi \mathrm{e}''}{(\mathrm{e}.f = \mathrm{e}') \rightarrow_\chi (\mathrm{e}.f = \mathrm{e}'')}$$

$$[\textsc{rc-invk}_1] \quad \frac{\mathrm{e}_0 \rightarrow_\chi \mathrm{e}'_0}{\mathrm{e}_0.m(\overline{\mathrm{e}}) \rightarrow_\chi \mathrm{e}'_0.m(\overline{\mathrm{e}})}$$

$$[\textsc{rc-invk}_2] \quad \frac{\mathrm{e}_j \rightarrow_\chi \mathrm{e}'_j}{\mathrm{e}_0.m(\overline{\mathrm{e}}_n) \rightarrow_\chi \mathrm{e}_0.m(\overline{\mathrm{e}'}_n)} \left(\begin{array}{c} j \in \overline{n} \\ \forall i \in \overline{n}\ [i \neq j \Rightarrow \mathrm{e}'_i = \mathrm{e}_i] \end{array} \right)$$

$$[\textsc{rc-new}] \quad \frac{\mathrm{e}_j \rightarrow_\chi \mathrm{e}'_j}{\texttt{new}\ \mathrm{C}(\overline{\mathrm{e}}_n) \rightarrow_\chi \texttt{new}\ \mathrm{C}(\overline{\mathrm{e}'}_n)} \left(\begin{array}{c} j \in \overline{n} \\ \forall i \in \overline{n}\ [i \neq j \Rightarrow \mathrm{e}'_i = \mathrm{e}_i] \end{array} \right)$$

For two expressions, e and e', if $\mathrm{e} \rightarrow_\chi \mathrm{e}'$, then we call e the *redex*, and e' the *reduct*. We also say that e' *expands* to e. We denote the transitive closure of $\rightarrow$ by $\rightarrow^*$, and thus write $\mathrm{e} \rightarrow^*_\chi \mathrm{e}'$ if there exists a (possibly empty) sequence of expressions $\overline{\mathrm{e}}_n$ such that $\mathrm{e} \rightarrow_\chi \mathrm{e}_1 \rightarrow_\chi \ldots \rightarrow_\chi \mathrm{e}_n \rightarrow_\chi \mathrm{e}'$.

In the next chapter, we will define an intersection type system for LJ, and in Chapter 5 we will derive some results relating typeability to this notion of reduction.

3.4 Remarks on the Nature of the Calculus

Now that we have defined the calculus, and how it operates, we are in a position to make a few remarks on how it compares to the other calculi on which it is based. But for the fact that method bodies do not include the `return` keyword, or terminate with the '`;`' character, LJ is a valid subset of full Java (if we also consider the type annotations on null object expressions as casts), which similarly is the case for Featherweight Java and Middleweight Java. Otherwise, the most prominent feature of LJ is that it is *functional* in nature. In this respect, it is similar to FJ and, indeed, the λ-calculus. Furthermore, like the Lambda Calculus, we expect LJ to be *confluent*: that is, it exhibits the *Church-Rosser* property [9] that if an expression e reduces to both e_1 *and* e_2, then there is a further expression e' such that both $e_1 \rightarrow^* e'$ and $e_2 \rightarrow^* e'$.

Middleweight Java, on the other hand, is an *imperative* calculus. Method bodies consist of a sequence of *statements* which may, and indeed are most often intended to have side-effects. MJ defines the notion of a store, and newly created objects persist in this store with further execution operating on a *reference* to the object. Thus, the result of the execution of one sub-expression may be visible to another unrelated sub-expression. This is something which is not possible in LJ, since the substitution of an object expression into multiple sub-expressions effectively creates *copies* of that object expression. MJ also contains other syntactic elements that our calculus does not: local variable declaration and assignment, for instance, as well as conditional statements. It is thus a larger subset of Java than LJ.

Chapter 4

The Type Systems

We now define two type assignment systems for LJ expressions. The first directly corresponds to the type systems found in [18] and [8], which in turn are modelled on the full Java type system [17]. We call this the *class* type system. The second system is an extension of the first, which we call the *predicate* type system. For readability, when we refer to the type system from now on, we will mean the *class* type system, and the *predicate system* will refer to the predicate type system. When we refer to *types*, we will mean class types, and similarly *predicates* will refer to predicate types.

The predicate system takes after the system of the same name in [5], with predicates comprising sequences of statements, each of which describes a single behaviour exhibited by the expression to which it is assigned. It is equivalent to the intersection type systems studied for the Lambda Calculus [2], and similar results will be shown to hold.

4.1 The Class Type System

Standard notions of type assignment, which associate a type with an expression require an *environment* to provide assumptions for the types of any (free) variables that occur in the expression. Our system is no different, and so we now proceed to formally define this notion:

Definition 4.1.1 (TYPE ENVIRONMENTS).

1. A *type statement* is a construction of the form $\mathrm{e} : \mathrm{C}$, where e is an LJ expression, and C is a class type. The expression e is called the *subject* of the statement, and the type C is called the *conclusion* of the statement.

2. A *type environment*, Γ, is a set of type statements with term variables as subjects. Note that we do not require the term variables to be distinct. This is only the case for *well formed* environments, defined below.

3. We use the abbreviation $\Gamma, x : \mathrm{C}$ to represent $\Gamma \cup \{x : \mathrm{C}\}$. Similarly, we write Γ, Γ' to represent $\Gamma \cup \Gamma'$.

4. We define the function $\mathbb{VARS}^{\mathrm{T}}_{\mathrm{ENV}}$, which returns the set of variables used as subjects of the statements in a type environment as follows:

$$\mathbb{VARS}^{\mathrm{T}}_{\mathrm{ENV}}(\Gamma) = \{x \mid x : \mathrm{C} \in \Gamma\}$$

Definition 4.1.2 (WELL-FORMED TYPE ENVIRONMENTS). We say that a type environment Γ is *well formed*, with respect to some execution context χ, when the execution context is itself well formed and the statements in Γ all have distinct variables as subjects. Furthermore, the conclusion of each statement must be a valid class type with respect to χ. This notion is formalised through the following judgements:

$$\frac{\vdash \chi}{\chi \vdash \emptyset}$$

$$\frac{\chi \vdash \Gamma \qquad \vdash_{\chi} \mathrm{C}}{\chi \vdash \Gamma, x : \mathrm{C}} \; (\neg \exists \mathrm{D}\, [x : \mathrm{D} \in \Gamma])$$

Notice that, by simple induction on the derivation of $\chi \vdash \Gamma$, it is easy to see that $\chi \vdash \Gamma'$ for any $\Gamma' \subseteq \Gamma$.

Definition 4.1.3 (TYPE ENVIRONMENT SUBSTITUTION)**.** We can perform substitution on a type environment, which results in the renaming of variables. Given a variable substitution $[y/x]$, we define substitution on type environments inductively as follows:

$$\begin{array}{rcll} (\emptyset)[y/x] & = & \emptyset & \\ (\Gamma, z : \mathrm{C})[y/x] & = & (\Gamma)[y/x], z : \mathrm{C} & \text{if } z \neq x \\ (\Gamma, z : \mathrm{C})[y/x] & = & (\Gamma)[y/x], y : \mathrm{C} & \text{if } z = x \end{array}$$

Notice that, if Γ is well-formed with respect to some execution context χ, and there does not exist a class D such that $y : \mathrm{D} \in \Gamma$, then $(\Gamma)[y/x]$ will also be well formed with respect to χ.

Definition 4.1.4 (TYPE ASSIGNMENT)**.** 1. Type assignment $\vDash^{\mathrm{T}}$ is a ternary relation between execution contexts, type environments and type statements, written as $\Gamma \vDash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C}$. We say that the type C can be *assigned* to the expression e in the context χ using the type environment Γ. It is defined using the following natural deduction system:

[T-NULL] $$\frac{\chi \vdash \Gamma \qquad \vdash_{\chi} \mathrm{C}}{\Gamma \vDash^{\mathrm{T}}_{\chi} (\mathrm{C})\,\texttt{null} : \mathrm{C}}$$

[T-VAR] $$\frac{\chi \vdash \Gamma}{\Gamma \vDash^{\mathrm{T}}_{\chi} x : \mathrm{C}} \; (x : \mathrm{C} \in \Gamma)$$

[T-FLD] $$\frac{\Gamma \vDash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{D}}{\Gamma \vDash^{\mathrm{T}}_{\chi} \mathrm{e}.f : \mathrm{C}} \; (\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f) = \mathrm{C})$$

[T-ASS] $$\frac{\Gamma \vDash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{D} \qquad \Gamma \vDash^{\mathrm{T}}_{\chi} \mathrm{e}' : \mathrm{C}}{\Gamma \vDash^{\mathrm{T}}_{\chi} \mathrm{e}.f = \mathrm{e}' : \mathrm{D}} \; (\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f) = \mathrm{C})$$

[T-INVK] $$\frac{\Gamma \vDash^{\mathrm{T}}_{\chi} \mathrm{e}_0 : \mathrm{C}_0 \quad \Gamma \vDash^{\mathrm{T}}_{\chi} \mathrm{e}_1 : \mathrm{C}_1 \quad \ldots \quad \Gamma \vDash^{\mathrm{T}}_{\chi} \mathrm{e}_n : \mathrm{C}_n}{\Gamma \vDash^{\mathrm{T}}_{\chi} \mathrm{e}_0.m(\overline{\mathrm{e}}_n) : \mathrm{D}} \; (\Delta_{\mathrm{m}}(\chi, \mathrm{C}_0, m) = \overline{\mathrm{C}}_n \rightarrow \mathrm{D})$$

[T-NEW] $$\frac{\Gamma \vDash^{\mathrm{T}}_{\chi} \mathrm{e}_1 : \mathrm{C}_1 \quad \ldots \quad \Gamma \vDash^{\mathrm{T}}_{\chi} \mathrm{e}_n : \mathrm{C}_n}{\Gamma \vDash^{\mathrm{T}}_{\chi} \texttt{new}\ \mathrm{C}(\overline{\mathrm{e}}_n) : \mathrm{C}} \left(\begin{array}{c} \mathcal{F}(\chi, \mathrm{C}) = \overline{f}_n \\ \Delta_{\mathrm{f}}(\chi, \mathrm{C}, f_1) = \mathrm{C}_1, \ldots, \Delta_{\mathrm{f}}(\chi, \mathrm{C}, f_n) = \mathrm{C}_n \end{array} \right)$$

[T-SUB] $$\frac{\Gamma \vDash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C}'}{\Gamma \vDash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C}} \; (\mathrm{C}' <:_{\chi} \mathrm{C})$$

2. We write $\mathcal{D}{::}\Gamma \vDash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C}$ if the judgement $\Gamma \vDash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C}$ is witnessed by the derivation $\mathcal{D}$.

Lemma 4.1.5 (WELL-FORMED TYPE ENVIRONMENTS)**.** If there exists a type derivation using a type environment Γ, then Γ is well-formed:

$$\Gamma \vDash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C} \Rightarrow \chi \vdash \Gamma$$

Proof. By easy induction on the structure of type derivations. The base cases are instances of the rules (T-NULL) or (P-VAR), both of which include the premise that the environment is well-formed $\chi \vdash \Gamma$. The other cases follow by straightforward induction. □

Lemma 4.1.6 (TYPE GENERATION)**.** The type system given in definition 4.1.4 displays the following generation properties:

1. $\Gamma \vdash^{\mathrm{T}}_{\chi} (\mathrm{C})\,\texttt{null} : \mathrm{D} \Rightarrow \mathrm{C} <:_{\chi} \mathrm{D}$
2. $\Gamma \vdash^{\mathrm{T}}_{\chi} x : \mathrm{C} \Rightarrow \exists\, \mathrm{C}' <:_{\chi} \mathrm{C}\ [x : \mathrm{C}' \in \Gamma]$
3. $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}.f :\ \mathrm{C} \Rightarrow \exists\, \mathrm{D}, \mathrm{D}' <:_{\chi} \mathrm{C}\ [\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{D}\ \&\ \Delta_{\mathrm{f}}(\chi, \mathrm{D}, f) = \mathrm{D}']$
4. $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}.f = \mathrm{e}' : \mathrm{C} \Rightarrow \exists\, \mathrm{D} <:_{\chi} \mathrm{C}\ [\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{D}\ \&\ \Delta_{\mathrm{f}}(\chi, \mathrm{D}, f) = \mathrm{D}'\ \&\ \Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}' : \mathrm{D}']$
5. $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0.m(\overline{\mathrm{e}}_n) : \mathrm{D} \Rightarrow \exists\, \mathrm{C}_0, \overline{\mathrm{C}}_n, \mathrm{D}' <:_{\chi} \mathrm{D}\ [\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0 : \mathrm{C}_0\ \&\ \Delta_{\mathrm{m}}(\chi, \mathrm{C}_0, m) = \overline{\mathrm{C}}_n \to \mathrm{D}'\ \&\ \forall i \in \overline{n}\ [\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_i : \mathrm{C}_i]]$
6. $\Gamma \vdash^{\mathrm{T}}_{\chi} \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_n) : \mathrm{C} \Rightarrow \mathrm{D} <:_{\chi} \mathrm{C}\ \&\ \mathcal{F}(\chi, \mathrm{D}) = \overline{f}_n\ \&\ \forall i \in \overline{n}\ [\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f_i) = \mathrm{C}_i\ \&\ \Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_i : \mathrm{C}_i]$

Proof. By easy induction on the structure of type derivations. We show the case for method invocation. If $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0.m(\overline{\mathrm{e}}_n) : \mathrm{C}$ then the last step of the derivation must be an instance of either the (T-INVK) rule or the (T-SUB) rule. If it is (T-INVK), then we have immediately that $\exists\, \mathrm{C}_0$ and $\overline{\mathrm{C}}_n$ with $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0 : \mathrm{C}_0$ and $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_i : \mathrm{C}_i$ for each $i \in \overline{n}$. We also have that $\exists\, \mathrm{D} <:_{\chi} \mathrm{C}$, since the subtype relation is reflexive, and we can take $\mathrm{D} = \mathrm{C}$. If the rule is (T-SUB), then we have that $\exists\, \mathrm{C}' <:_{\chi} \mathrm{C}$ such that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0.m(\overline{\mathrm{e}}_n) : \mathrm{C}'$, and we can apply the same reasoning again, since the expression is unchanged. In general, there may be any (finite) number of instances of rule (T-SUB) before an instance of (T-INVK). Therefore, we will obtain a sequence of types $\overline{\mathrm{C}'}_{n'}$ such that $\mathrm{C}'_1 <:_{\chi} \mathrm{C}$ and $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0.m(\overline{\mathrm{e}}_n) : \mathrm{C}'_i$ with $\mathrm{C}'_{i+1} <:_{\chi} \mathrm{C}'_i$ for each $i \in \overline{n'}$, before we have that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0.m(\overline{\mathrm{e}}_n) : \mathrm{C}'_{n'+1}$ by a instance of rule (T-INVK). Then, since $<:_{\chi}$ is transitive we have that $\mathrm{C}'_{n'+1} <:_{\chi} \mathrm{C}$. $\square$

Definition 4.1.7 (TYPE CONSISTENT EXECUTION CONTEXTS)**.** We say that an execution context is *type consistent*, $\vdash^{\mathrm{T}}_{\chi} \diamond$, when the bodies of all methods defined in the context can be assigned their declared return type.

$$\begin{array}{lll}
\vdash^{\mathrm{T}}_{\chi} \diamond & \Leftrightarrow & \vdash \chi\ \&\ \forall\, \mathrm{C}\ [\\
 & & \quad \Delta(\chi, \mathrm{C})\!\Downarrow\ \Rightarrow \forall\, m\ [\ \ \Delta_{\mathrm{m}}(\chi, \mathrm{C}, m) = \overline{\mathrm{D}}_n \to \mathrm{D}_0\ \&\ \textsc{mbody}(\chi, \mathrm{C}, m) = (\overline{x}_n, \mathrm{e}_0) \\
 & & \qquad\qquad\qquad\qquad \Rightarrow \{x_1 : \mathrm{D}_1, \ldots, x_n : \mathrm{D}_n, \texttt{this} : \mathrm{C}\} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0 : \mathrm{D}_0] \\
 & &]
\end{array}$$

4.2 The Predicate Type System

We now define an extension to the class type system: the predicate type system. We will see that the predicate system types exactly the same set of terms as the class type system. This result is shown in Theorem 4.3.1.

Definition 4.2.1 (PREDICATE TYPES)**.** We define a set of *predicate types*, or simply *predicates*, in the spirit of the system of van Bakel and de'Liguoro [5]. The set $\mathbb{PRED}$ of predicates, ranged over by ϕ and ψ, consists of the set of *object* predicates, ranged over by σ, and the special predicate constant ω, which we call the *universal* predicate. Object predicates consist of a (possibly empty) sequence of *predicate member statements*. These are statements of the form $l : \tau$, where l ranges over the set of class member labels (see Definition 3.1.1) and τ ranges over the set of *member* predicates. We define these sets of predicates inductively as follows:

$$\begin{array}{rcll}
\phi, \psi & ::= & \omega \mid \sigma & \\
\sigma & ::= & \langle l_1 : \tau_1,\ \ldots\ , l_n : \tau_n \rangle & (n \geq 0) \\
\tau & ::= & \phi \mid \psi{::}\phi_1,\ \ldots\ , \phi_n \to \sigma & (n \geq 0)
\end{array}$$

We can abbreviate the object predicate $\langle l_1 : \tau_1,\ \ldots\ , l_n : \tau_n \rangle$, by writing $\langle l_i : \tau_i\ ^{i \in \overline{n}} \rangle$, and we call the object predicate consisting of the empty sequence, $\langle \epsilon \rangle$, the *empty* predicate. We call ω a *trivial* predicate; all other predicates are, correspondingly, *non-trivial*.

The aim in defining predicates in this way is that they should describe the behaviour of an object. The predicate member statements that comprise an object predicate each indicate that the object to which it is assigned behaves in a particular way. The class member label in each statement denotes either a field of method belonging to the object, and the member predicate then describes the result of accessing the field or invoking the method. In the case of a method member statement, the member predicate also indicates the *required* behaviour of the arguments, as well as the receiver. The universal predicate is intended to indicate non-terminating behaviour, as will be discussed in §5.4.

Definition 4.2.2 (SUBPREDICATES). 1. The relation $\trianglelefteq$ is defined as the least pre-order on predicates such that:

$$\begin{array}{rcll} \langle\epsilon\rangle & \trianglelefteq & \omega & \\ \langle l_i : \tau_i {}^{\,i \in \overline{n}}\rangle & \trianglelefteq & \langle l_j : \tau_j\rangle & \forall j \in \overline{n} \\ \phi \trianglelefteq \langle l_j : \tau_j\rangle \text{ for all } j \in \overline{n} & \Rightarrow & \phi \trianglelefteq \langle l_i : \tau_i {}^{\,i \in \overline{n}}\rangle & n \geq 1 \end{array}$$

2. The equivalence relation $\tau \sim \sigma$ is defined over member types by:

$$\phi \sim \psi \quad \Leftrightarrow \quad \phi \trianglelefteq \psi \trianglelefteq \phi$$

This definition captures the intuition that object predicates should be equivalent up to reordering of predicate member statements. Thus

$$\langle l_i : \tau_i {}^{\,i\in\overline{n_1}}\rangle \sim \langle l'_i : \tau'_i {}^{\,i\in\overline{n_2}}\rangle \Rightarrow n_1 = n_2 \;\&\; \forall i \in \overline{n_2}\; \exists j \in \overline{n_1}\; [l_j = l'_i \;\&\; \tau_j = \tau'_i]$$

Definition 4.2.3 (PREDICATE JOIN). The *join* of two predicates, $\phi_1 \sqcup \phi_2$, is defined as follows:

$$\begin{array}{rcl} \phi \sqcup \omega & = & \phi \\ \omega \sqcup \phi & = & \phi \\ \\ \langle l_i : \tau_i {}^{\,i \in \overline{n_1}}\rangle \sqcup \langle l'_i : \tau'_i {}^{\,i \in \overline{n_2}}\rangle & = & \langle l_1 : \tau_1, \ldots, l_{n_1} : \tau_{n_1}, l'_1 : \tau'_1, \ldots, l'_{n_2} : \tau'_{n_2}\rangle \end{array}$$

We also define the following shorthand notation:

$$\bigsqcup_{i\in\overline{n}} \phi_i \triangleq \phi_1 \sqcup \ldots \sqcup \phi_n$$

The join operation will be used in the proof of the subject expansion property for the predicate system in §5.3, and also in the type inference algorithms in Chapter 6.

Definition 4.2.4 (PREDICATE ENVIRONMENTS). 1. A *predicate statement* is a construction of the form e : C : ϕ, where e is an LJ expression, C is a class type and ϕ is a predicate. The expression e is called the *subject*, C is called the *type conclusion*, and ϕ is called the *predicate conclusion* of the statement.

2. A *predicate environment*, Π, is a set of predicate statements with term variables as subjects. As for type environments, we do not require the term variables to be distinct. Again, this is only a property of well formed environments, defined below.

3. As for type environments, we use the abbreviation $\Pi, x : \mathrm{C} : \phi$ to represent $\Pi \cup \{x : \mathrm{C} : \phi\}$. Similarly, we write Π, Π' to represent $\Pi \cup \Pi'$.

4. We define the function $\mathbb{VARS}^{\mathrm{P}}_{\mathrm{ENV}}$, which returns the set of variables used as subjects of the statements in a type environment as follows:

$$\mathbb{VARS}^{\mathrm{P}}_{\mathrm{ENV}}(\Pi) = \{x \mid x : \mathrm{C} : \phi \in \Pi\}$$

5. We say the a predicate environment Π is an *object* predicate environment when the predicate conclusion of each statement Π is an object predicate that does not contain ω.

6. We extend the subpredicate relation to predicate environments as follows:

$$\Pi \trianglelefteq \Pi' \Leftrightarrow \forall\, x \in \mathbb{VARS}^{\mathrm{P}}_{\mathrm{ENV}}(\Pi)\; [x : \mathrm{C} : \phi \in \Pi \Rightarrow x : \mathrm{C} : \phi' \in \Pi' \;\&\; \phi \trianglelefteq \phi']$$

Definition 4.2.5 (WELL-FORMED PREDICATE ENVIRONMENTS). We say that a predicate environment Π is *well formed*, with respect to some execution context χ, when the statements in Π all have distinct variables as subjects, and the type conclusion of each statement is a valid class type with respect to χ. This notion is formalised through the following judgements:

$$\frac{\vdash \chi}{\chi \vdash \emptyset}$$

$$\frac{\chi \vdash \Pi \qquad \vdash_{\chi} \mathrm{C}}{\chi \vdash \Pi, x : \mathrm{C} : \phi}\ (\neg\exists \mathrm{D}, \phi' \ [x : \mathrm{D} : \phi' \in \Pi])$$

Notice that, by simple induction on the derivation of $\chi \vdash \Pi$, it is easy to see that $\chi \vdash \Pi'$ for any $\Pi' \subseteq \Pi$.

Definition 4.2.6 (PREDICATE ENVIRONMENT SUBSTITUTION)**.** We can perform substitution on a predicate environment, which results in the renaming of variables. Given a variable substitution $[y/x]$, we define substitution on predicate environments in the same way as for type environments as follows:

$$\begin{array}{rcll}(\emptyset)[y/x] & = & \emptyset & \\ (\Pi, z : \mathrm{C} : \phi)[y/x] & = & (\Pi)[y/x], z : \mathrm{C} : \phi & \text{if } z \neq x \\ (\Pi, z : \mathrm{C} : \phi)[y/x] & = & (\Pi)[y/x], y : \mathrm{C} : \phi & \text{if } z = x\end{array}$$

Notice that, if Π is well-formed with respect to some execution context χ, and there does not exists a type D and a predicate ϕ' such that $y : \mathrm{D} : \phi' \in \Pi$, then $(\Pi)[y/x]$ will also be well-formed with respect to χ.

Definition 4.2.7 (ENVIRONMENT CONVERSION)**.** The notation $\widehat{\Pi}$ denotes the type environment obtained by discarding the predicate conclusions from the statements in Π:

$$\widehat{\Pi} \triangleq \{\, x : \mathrm{C} \mid x : \mathrm{C} : \phi \in \Pi \,\}$$

Notice that the following results are an immediate consequence of this definition:

$$\begin{array}{rcl}\Pi \subseteq \Pi' & \Leftrightarrow & \widehat{\Pi} \subseteq \widehat{\Pi'} \\ \chi \vdash \widehat{\Pi} & \Leftrightarrow & \chi \vdash \Pi \\ \widehat{\Pi[y/x]} & = & \widehat{\Pi}\,[y/x]\end{array}$$

Definition 4.2.8 (PREDICATE ASSIGNMENT)**.** 1. Predicate assignment $\Vdash^{\mathrm{P}}$ is a ternary relation between execution contexts, predicate environments and predicate statements, written as $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \phi$. We say that type-predicate pair $\mathrm{C} : \phi$ can be assigned to expression e in the context χ using the predicate environment Π. It is defined using the following natural deduction system:

[P-NULL]
$$\frac{\chi \vdash \Pi \qquad \vdash_{\chi} \mathrm{C}}{\Pi \Vdash^{\mathrm{P}}_{\chi} (\mathrm{C})\,\mathtt{null} : \mathrm{C} : \langle\epsilon\rangle}$$

[P-VAR]
$$\frac{\chi \vdash \Pi}{\Pi \Vdash^{\mathrm{P}}_{\chi} x : \mathrm{C} : \phi}\ (x : \mathrm{C} : \phi \in \Pi)$$

[P-NEWOBJ]
$$\frac{\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathtt{new}\ \mathrm{C}(\overline{\mathrm{e}}) : \mathrm{C}}{\Pi \Vdash^{\mathrm{P}}_{\chi} \mathtt{new}\ \mathrm{C}(\overline{\mathrm{e}}) : \mathrm{C} : \langle\epsilon\rangle}$$

[P-FLD]
$$\frac{\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{D} : \langle f : \phi\rangle}{\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}.f : \mathrm{C} : \phi}\ (\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f) = \mathrm{C})$$

[P-ASS$_1$]
$$\frac{\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \sigma \qquad \Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}' : \mathrm{D} : \phi}{\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}.f = \mathrm{e}' : \mathrm{C} : \langle f : \phi\rangle}\ (\Delta_{\mathrm{f}}(\chi, \mathrm{C}, f) = \mathrm{D})$$

[P-ASS$_2$]
$$\frac{\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \langle l_i : \tau_i \ ^{i \in \overline{n}}\rangle \qquad \widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}' : \mathrm{D}}{\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}.f = \mathrm{e}' : \mathrm{C} : \langle l_i : \tau_i \ ^{i \in \overline{n}}\rangle}\ \left(\begin{array}{c} f \notin \overline{l}_n \\ \Delta_{\mathrm{f}}(\chi, \mathrm{C}, f) = \mathrm{D}\end{array}\right)$$

[P-INVK]
$$\frac{\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0 : \mathrm{D} : \langle m : \psi{::}\overline{\phi}_n \to \sigma\rangle \qquad \Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_i : \mathrm{C}_i : \phi_i\ \ (\forall i \in \overline{n}) \qquad \Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0 : \mathrm{D} : \psi}{\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0.m(\overline{\mathrm{e}}_n) : \mathrm{C} : \sigma}\ (\Delta_{\mathrm{m}}(\chi, \mathrm{D}, m) = \overline{\mathrm{C}}_n \to \mathrm{C})$$

[P-NEWFLD]
$$\frac{\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_j : \mathrm{C}_j : \phi \qquad \widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_i : \mathrm{C}_i \quad (\forall i \in \overline{n}\ [i \neq j])}{\Pi \Vdash^{\mathrm{P}}_{\chi} \mathtt{new}\ \mathrm{C}(\overline{\mathrm{e}}_n) : \mathrm{C} : \langle f_j : \phi\rangle}\ \left(\begin{array}{c} \mathcal{F}(\chi, \mathrm{C}) = \overline{f}_n \ \&\ j \in \overline{n} \\ \forall\, i \in \overline{n}\ [\Delta_{\mathrm{f}}(\chi, \mathrm{C}, f_i) = \mathrm{C}_i]\end{array}\right)$$

[P-NEWMETH]
$$\frac{\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathtt{new}\ \mathrm{C}(\overline{\mathrm{e}}_n) : \mathrm{C} \qquad \Pi' \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0 : \mathrm{D} : \sigma}{\Pi \Vdash^{\mathrm{P}}_{\chi} \mathtt{new}\ \mathrm{C}(\overline{\mathrm{e}}_n) : \mathrm{C} : \langle m : \psi{::}\overline{\phi}_{n'} \to \sigma\rangle}\ \left(\begin{array}{c} \Delta_{\mathrm{m}}(\chi, \mathrm{C}, m) = \overline{\mathrm{C}}_{n'} \to \mathrm{D} \\ \textsc{mbody}(\chi, \mathrm{C}, m) = (\overline{x}_{n'}, \mathrm{e}_0) \\ \Pi' = \{x_1 : \mathrm{C}_1 : \phi_1,\ \ldots\ , x_{n'} : \mathrm{C}_{n'} : \phi_{n'}, \mathtt{this} : \mathrm{C} : \psi\}\end{array}\right)$$

$$[\textsc{p-subType}] \quad \frac{\Pi \vdash^{\mathsf{P}}_{\chi} \mathrm{e} : \mathrm{C}' : \phi}{\Pi \vdash^{\mathsf{P}}_{\chi} \mathrm{e} : \mathrm{C} : \phi}\ (\mathrm{C}' <:_{\chi} \mathrm{C}) \qquad [\textsc{p-omega}] \quad \frac{\widehat{\Pi} \vdash^{\mathsf{T}}_{\chi} \mathrm{e} : \mathrm{C}}{\Pi \vdash^{\mathsf{P}}_{\chi} \mathrm{e} : \mathrm{C} : \omega}$$

$$[\textsc{p-sel}_1] \quad \frac{\Pi \vdash^{\mathsf{P}}_{\chi} \mathrm{e} : \mathrm{C} : \langle l_i : \tau_i \ ^{i \in \overline{n}} \rangle}{\Pi \vdash^{\mathsf{P}}_{\chi} \mathrm{e} : \mathrm{C} : \langle l_j : \tau_j \rangle}\ (j \in \overline{n}) \qquad [\textsc{p-sel}_2] \quad \frac{\Pi \vdash^{\mathsf{P}}_{\chi} \mathrm{e} : \mathrm{C} : \langle l_i : \tau_i \ ^{i \in \overline{n}} \rangle}{\Pi \vdash^{\mathsf{P}}_{\chi} \mathrm{e} : \mathrm{C} : \langle \epsilon \rangle}$$

$$[\textsc{p-seq}] \quad \frac{\Pi \vdash^{\mathsf{P}}_{\chi} \mathrm{e} : \mathrm{C} : \langle l_1 : \tau_1 \rangle \ \ \ldots \ \ \Pi \vdash^{\mathsf{P}}_{\chi} \mathrm{e} : \mathrm{C} : \langle l_n : \tau_n \rangle}{\Pi \vdash^{\mathsf{P}}_{\chi} \mathrm{e} : \mathrm{C} : \langle l_i : \tau_i \ ^{i \in \overline{n}} \rangle}$$

2. As for type assignment, we write $\mathcal{D}{::}\Pi \vdash^{\mathsf{P}}_{\chi} \mathrm{e} : \mathrm{C} : \phi$ if the judgement $\Pi \vdash^{\mathsf{P}}_{\chi} \mathrm{e} : \mathrm{C} : \phi$ is witnessed by the derivation $\mathcal{D}$.

Lemma 4.2.9 (WELL-FORMED PREDICATE ENVIRONMENTS)**.** If there exists a predicate derivation using a predicate environment Π, then Π is well-formed:

$$\Pi \vdash^{\mathsf{P}}_{\chi} \mathrm{e} : \mathrm{C} : \phi \Rightarrow \chi \vdash \Pi$$

Proof. By easy induction on the structure of type derivations. We note that the leaves of any predicate derivation will be instances of the rules (P-VAR) and (P-NULL), or type derivations. In the case of the former, the premise of this rule is that the environment is well-formed, $\chi \vdash \Pi$. For the latter, we have by Lemma 4.1.5 that $\chi \vdash \widehat{\Pi}$, and then by Definition 4.2.7 that $\chi \vdash \Pi$. □

Lemma 4.2.10 (PREDICATE GENERATION)**.** The predicate system displays the following generation properties:

1. $\Pi \vdash^{\mathsf{P}}_{\chi} (\mathrm{C})\, \mathtt{null} : \mathrm{D} : \sigma \Rightarrow \mathrm{C} <:_{\chi} \mathrm{D}\ \&\ \vdash_{\chi} \mathrm{C}\ \&\ \sigma \equiv \langle \epsilon \rangle$
2. $\Pi \vdash^{\mathsf{P}}_{\chi} x : \mathrm{C} : \sigma \Rightarrow \exists\, \mathrm{D} <:_{\chi} \mathrm{C}, \psi \trianglelefteq \sigma\ [x : \mathrm{D} : \psi \in \Pi]$
3. $\Pi \vdash^{\mathsf{P}}_{\chi} \mathrm{e}.f : \mathrm{C} : \sigma \Rightarrow \exists\, \mathrm{D}, \mathrm{C}' <:_{\chi} \mathrm{C}, \psi \trianglelefteq \sigma\ [\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f) = \mathrm{C}'\ \&\ \Pi \vdash^{\mathsf{P}}_{\chi} \mathrm{e} : \mathrm{D} : \langle f : \psi \rangle]$
4. $\Pi \vdash^{\mathsf{P}}_{\chi} \mathrm{e}.f = \mathrm{e}' : \mathrm{C} : \sigma \Rightarrow \forall\, i \in \overline{n}\ [\exists\, \mathrm{D}_i <:_{\chi} \mathrm{C}\ [\Pi \vdash^{\mathsf{P}}_{\chi} \mathrm{e}.f = \mathrm{e}' : \mathrm{D}_i : \langle l_i : \tau_i \rangle]]$
5. $\Pi \vdash^{\mathsf{P}}_{\chi} \mathrm{e}_0.m(\overline{\mathrm{e}}_{n_1}) : \mathrm{C} : \sigma \Rightarrow \exists\, \mathrm{D}, \overline{\mathrm{C}}_{n_1}, \mathrm{C}' <:_{\chi} \mathrm{C}, \psi, \overline{\phi}_{n_1}, \sigma' \trianglelefteq \sigma\ [\Delta_{\mathrm{m}}(\chi, \mathrm{D}, m) = \overline{\mathrm{C}}_{n_1} \rightarrow \mathrm{C}'\ \&\ \Pi \vdash^{\mathsf{P}}_{\chi} \mathrm{e}_0 : \mathrm{D} : \langle m : \psi{::}\overline{\phi}_{n_1} \rightarrow \sigma' \rangle\ \&\ \Pi \vdash^{\mathsf{P}}_{\chi} \mathrm{e}_0 : \mathrm{D} : \psi\ \&\ \forall\, i \in \overline{n_1}\ [\Pi \vdash^{\mathsf{P}}_{\chi} \mathrm{e}_i : \mathrm{C}_i : \phi_i]]$
6. $\Pi \vdash^{\mathsf{P}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1}) : \mathrm{C} : \langle \epsilon \rangle \Rightarrow \mathrm{D} <:_{\chi} \mathrm{C}\ \&\ \widehat{\Pi} \vdash^{\mathsf{T}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1}) : \mathrm{D}$

In the following, we take $\sigma = \langle l_i : \tau_i \ ^{i \in \overline{n}} \rangle \not\equiv \langle \epsilon \rangle$:

7. $\Pi \vdash^{\mathsf{P}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1}) : \mathrm{C} : \sigma \Rightarrow \mathrm{D} <:_{\chi} \mathrm{C}\ \&\ \forall\, i \in \overline{n}\ [\Pi \vdash^{\mathsf{P}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1}) : \mathrm{D} : \langle l_i : \tau_i \rangle]$

Proof. By easy induction on the structure of predicate derivations. We show the case for new object creation, and a non-empty object predicate. If $\Pi \vdash^{\mathsf{P}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1}) : \mathrm{C} : \sigma$, with $\sigma \equiv \langle l_i : \tau_i \ ^{i \in \overline{n}} \rangle$ and $n > 0$, then there are two cases to consider. If $n = 1$, then $\sigma = \langle l : \tau \rangle$ and the last step of the derivation must be an instance of rule (P-SUBTYPE). In general, there will be a number of instances of (P-SUBTYPE), but without loss of generality, we may assume that there is a single instance, the premise of which is a derivation of $\Pi \vdash^{\mathsf{P}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1}) : \mathrm{D} : \langle l_i : \tau_i \ ^{i \in \overline{n}} \rangle$, with $\mathrm{D} <:_{\chi} \mathrm{C}$ by the side condition of the (P-SUBTYPE) rule.

If $n > 1$, then there will again be, in general, a sequence of instances of (P-SUBTYPE) with the premise $\Pi \vdash^{\mathsf{P}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1}) : \mathrm{C}' : \langle l_i : \tau_i \ ^{i \in \overline{n}} \rangle$ derived by an instance of (P-SEQ), with $\mathrm{C}' <:_{\chi} \mathrm{C}$. Then, it follows that $\Pi \vdash^{\mathsf{P}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1}) : \mathrm{C}' : \langle l_i : \tau_i \rangle$ for each $i \in \overline{n}$. Then, for each of these, there must be another sequence of instances of (P-SUBTYPE) with the premise being $\Pi \vdash^{\mathsf{P}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1}) : \mathrm{D} : \langle l_i : \tau_i \rangle$ derived by an instance of rule (P-NEWFLD) if $l_i \in \mathbb{FIELD\text{-}ID}$, and by an instance of rule (P-NEWMETH) if $l_i \in \mathbb{METHOD\text{-}NAME}$. It then follows by the side condition of (P-SUBTYPE) that $\mathrm{D} <:_{\chi} \mathrm{C}'$, and so by the transitivity of $<:_{\chi}$, we have that $\mathrm{D} <:_{\chi} \mathrm{C}$. □

4.3 Properties of the Type Systems

We now present some properties of the type systems defined in §4.1 and §4.2. We begin by showing that the set of expressions typeable by the class type system is exactly the same as the set of expressions typeable by the predicate system.

Theorem 4.3.1. $\exists\, \phi\, [\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \phi] \Leftrightarrow \widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C}$

Proof. ($\Leftarrow$) This direction is easy. If e is typeable then there certainly exists a predicate that we can assign to e: it is the universal predicate ω, which we can assign by rule (p-omega).

($\Rightarrow$) This direction proceeds by induction on the structure of predicate derivations. The base cases are (p-null), (p-var), (p-newObj), (p-newMeth) and (p-omega). We show some of these, and also a few inductive cases.

(p-null) Then $\mathrm{e} \equiv (\mathrm{C})\,\mathtt{null}$ with $\chi \vdash \Pi$ and $\vdash_{\chi} \mathrm{C}$. By Definition 4.2.7 we have that $\chi \vdash \widehat{\Pi}$ and so by rule (t-null) it follows that $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} (\mathrm{C})\,\mathtt{null} : \mathrm{C}$.

(p-var) Then $\mathrm{e} \equiv x$ with $\chi \vdash \Pi$ and $\vdash_{\chi} \mathrm{C}$ such that $x : \mathrm{C} : \phi \in \Pi$. By Definition 4.2.7 it follows that $x : \mathrm{C} \in \widehat{\Pi}$ and $\chi \vdash \widehat{\Pi}$. So by rule (t-var) we have that $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} x : \mathrm{C}$.

(p-newObj) Then $\mathrm{e} \equiv \mathtt{new}\ \mathrm{C}(\overline{\mathrm{e}})$ and the result follows immediately since the premise of this rule is that $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathtt{new}\ \mathrm{C}(\overline{\mathrm{e}}) : \mathrm{C}$.

(p-ass$_1$) Then $\mathrm{e} \equiv \mathrm{e}_0.f = \mathrm{e}_1$ and $\phi = \langle f : \phi' \rangle$ with $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0 : \mathrm{C} : \sigma$ for some σ and $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_1 : \mathrm{D} : \phi'$ such that $\Delta_{\mathrm{f}}(\chi, \mathrm{C}, f) = \mathrm{D}$. By the inductive hypothesis it follows that $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0 : \mathrm{C}$ and $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_1 : \mathrm{D}$. Then by rule (t-ass) we have that $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0.f = \mathrm{e}_1 : \mathrm{C}$.

(p-invk) Then $\mathrm{e} \equiv \mathrm{e}_0.m(\overline{\mathrm{e}}_n)$ with $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0 : \mathrm{D} : \langle m : \psi{::}\overline{\phi}_n \to \phi \rangle$ and $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0 : \mathrm{D} : \psi$. We also have that $\Delta_{\mathrm{m}}(\chi, \mathrm{D}, m) = \overline{\mathrm{C}}_n \to \mathrm{C}$ such that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_i : \mathrm{C}_i : \phi_i$ for each $i \in \overline{n}$. By the inductive hypothesis it follows that $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0 : \mathrm{D}$ and that $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_i : \mathrm{C}_i$ for each $i \in \overline{n}$. Then by rule (t-invk) we have that $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0.m(\overline{\mathrm{e}}_n) : \mathrm{C}$.

(p-newFld) Then $\mathrm{e} \equiv \mathtt{new}\ \mathrm{C}(\overline{\mathrm{e}}_n)$ and $\mathcal{F}(\chi, \mathrm{C}) = \overline{f}_n$ with $\phi = \langle f_j : \phi' \rangle$ for some $j \in \overline{n}$ such that $\Delta_{\mathrm{f}}(\chi, \mathrm{C}, f_j) = \mathrm{C}_j$ with $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_j : \mathrm{C}_j : \phi'$. Also, $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_i : \mathrm{C}_i$ with $\Delta_{\mathrm{f}}(\chi, \mathrm{C}) = \mathrm{C}_i$ for each $i \in \overline{n}$ such that $i \neq j$. By the inductive hypothesis it follows that $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_j : \mathrm{C}_j$ and so by rule (t-new) we have that $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathtt{new}\ \mathrm{C}(\overline{\mathrm{e}}_n) : \mathrm{C}$.

$\square$

A consequence of the preceding theorem is that the following rule is admissible in the predicate assignment system:

Theorem 4.3.2.

$$\frac{\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \psi}{\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \phi}\ (\psi \trianglelefteq \phi)$$

Proof. We break the proof down by considering the structure of ϕ:

($\phi \equiv \omega$) By assumption we have that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \psi$ and so, by Theorem 4.3.1, $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C}$. Then, by rule (p-omega), it follows that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \omega$.

($\phi \equiv \langle\epsilon\rangle$) By Definition 4.2.2, it must be that $\psi \equiv \langle l_i : \tau_i\ ^{i \in \overline{n}} \rangle$. If $n = 0$ then $\psi \equiv \langle\epsilon\rangle$ and the result follows immediately since then, by assumption, we have $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \langle\epsilon\rangle$. If $n > 0$, then we have by rule (p-sel$_2$) that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \langle\epsilon\rangle$.

($\phi \equiv \langle l_i : \tau_i\ ^{i \in \overline{n}} \rangle, n > 0$) By Definition 4.2.2, it must be that $\psi \equiv \langle l_i : \tau_i\ ^{i \in \overline{n'}} \rangle$ with $n' \geq n$ and for each $k \in \overline{n}$, $\exists\ j \in \overline{n'}$ such that $l_k = l'_j$ and $\tau_k = \tau'_j$. By assumption, $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \psi$ and so, by rule (p-sel$_1$) it follows that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \langle l'_k : \tau'_k \rangle$ for each $k \in \overline{n'}$. Then, since for all $k \in \overline{n}$, there is some $j \in \overline{n'}$ such that $l_k = l'_j$ and $\tau_k = \tau'_j$, we have that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \langle l_k : \tau_k \rangle$ for each $k \in \overline{n}$. And so, by rule (p-seq), it follows that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \langle l_i : \tau_i\ ^{i \in \overline{n}} \rangle$.

□

We also see in the following theorem that the set of predicates assignable to an expression does not depend on the set of *types* assignable to that expression. In other words, if there is a predicate derivation assigning the type-predicate pair $C : \phi$ to an expression e, and we can also assign the type C' to e, then there exists a predicate derivation assigning the type-predicate pair $C' : \phi$ to e also. This result is a crucial part of the proof for subject expansion.

Theorem 4.3.3. $\Pi \vdash^{P}_{\chi} e : C : \phi \ \& \ \widehat{\Pi} \vdash^{T}_{\chi} e : C' \Rightarrow \Pi \vdash^{P}_{\chi} e : C' : \phi$

Proof. By induction on the structure of predicate derivations. Again, we only show some of the cases:

(P-VAR) Then $e \equiv x$ with $x : C : \phi \in \Pi$. By assumption, $\widehat{\Pi} \vdash^{T}_{\chi} x : C'$ and so by Lemma 4.1.6(2) $\exists\, C'' <:_{\chi} C'\ [x : C''\widehat{\Pi}]$. By Definition 4.2.7 it follows that $x : C \in \widehat{\Pi}$, and so it must be the case that $C'' = C$ and therefore $C <:_{\chi} C'$. Since, by assumption, we have $\Pi \vdash^{P}_{\chi} x : C : \phi$ it then follows by rule (P-SUBTYPE) that $\Pi \vdash^{P}_{\chi} x : C' : \phi$.

(P-FLD) Then $e \equiv e'.f$ with $\Pi \vdash^{P}_{\chi} e' : D : \langle f : \phi\rangle$ such that $\Delta_{f}(\chi, D, f) = C$. By assumption we have that $\widehat{\Pi} \vdash^{T}_{\chi} e'.f : C'$ and so by Lemma 4.1.6(3), $\exists\, D', C'' <:_{\chi} C'$ such that $\widehat{\Pi} \vdash^{T}_{\chi} e' : D'$ and $\Delta_{f}(\chi, D', f) = C''$. By the inductive hypothesis it then follows that $\Pi \vdash^{P}_{\chi} e' : D' : \langle f : \phi\rangle$, and by rule (P-FLD) we have that $\Pi \vdash^{P}_{\chi} e'.f : C'' : \phi$. Then, since $C'' <:_{\chi} C'$, we have by rule (P-SUBTYPE) that $\Pi \vdash^{P}_{\chi} e'.f : C' : \phi$.

(P-ASS$_2$) Then $e \equiv e_1.f = e_2$ and $\phi = \langle l_i : \tau_i \ ^{i \in \overline{n}}\rangle$ with $\Pi \vdash^{P}_{\chi} e_1 : C : \langle l_i : \tau_i \ ^{i \in \overline{n}}\rangle$ and $\widehat{\Pi} \vdash^{T}_{\chi} e_2 : D$ such that $f \notin \overline{l}_n$ and $\Delta_{f}(\chi, C, f) = D$. By assumption we have that $\widehat{\Pi} \vdash^{T}_{\chi} e_1.f = e_2 : C'$ and so by Lemma 4.1.6(4), $\exists\, C'' <:_{\chi} C'$ such that $\widehat{\Pi} \vdash^{T}_{\chi} e_1 : C''$ and $\widehat{\Pi} \vdash^{T}_{\chi} e_2 : D'$ with $\Delta_{f}(\chi, C'', f) = D'$. By the inductive hypothesis it then follows that $\Pi \vdash^{P}_{\chi} e_1 : C'' : \langle l : \tau\rangle$, and so by rule (P-ASS$_2$) that $\Pi \vdash^{P}_{\chi} e_1.f = e_2 : C'' : \langle f : \phi'\rangle$. Then, since $C'' <:_{\chi} C'$ we have by rule (P-SUBTYPE) that $\Pi \vdash^{P}_{\chi} e_1.f = e_2 : C' : \langle f : \phi'\rangle$.

(P-NEWMETH) Then $e \equiv \texttt{new}\ C(\overline{e}_n)$ and $\phi = \langle m : \psi{::}\overline{\phi}_n \to \phi'\rangle$. By assumption we have that $\widehat{\Pi} \vdash^{T}_{\chi} \texttt{new}\ C(\overline{e}_n) : C'$ and so by Lemma 4.1.6(6), it follows that $C <:_{\chi} C'$. We then have by rule (P-SUBTYPE) that $\Pi \vdash^{P}_{\chi} \texttt{new}\ C(\overline{e}_n) : C' : \langle m : \psi{::}\overline{\phi}_n \to \phi'\rangle$.

(P-OMEGA) Then $\phi = \omega$ and since, by assumption, we have that $\widehat{\Pi} \vdash^{T}_{\chi} e : C'$, it follows immediately by rule (P-OMEGA) that $\Pi \vdash^{P}_{\chi} e : C' : \omega$.

□

Chapter 5

Subject Reduction & Expansion

In this chapter we discuss and prove results relating the type systems defined in Chapter 4 with the reduction relation (Definition 3.3.3). Specifically, we see that the reduction relation preserves the types assignable to terms. The results that we show in this chapter are subject reduction (for both the class and predicate type systems), and subject expansion (for the predicate type system only). We state these results now, and present their proofs later in the chapter.

Theorem. Both types and predicates are preserved by reduction:

1. $\Vdash^{\mathrm{T}}_{\chi} \diamond \;\&\; \Gamma \Vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C} \;\&\; \mathrm{e} \rightarrow_{\chi} \mathrm{e}' \Rightarrow \Gamma \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}' : \mathrm{C}$
2. $\Vdash^{\mathrm{T}}_{\chi} \diamond \;\&\; \Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \phi \;\&\; \mathrm{e} \rightarrow_{\chi} \mathrm{e}' \Rightarrow \Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}' : \mathrm{C} : \phi$

Predicates are preserved by expansion:

3. $\Vdash^{\mathrm{T}}_{\chi} \diamond \;\&\; \Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}' : \mathrm{C} : \phi \;\&\; \mathrm{e} \rightarrow_{\chi} \mathrm{e}' \;\&\; \widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C} \Rightarrow \Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \phi$

Before presenting the proofs of the subject reduction and expansion theorems, we must develop some other auxiliary lemmas and theorems.

5.1 Auxiliary Lemmas and Theorems

One thing that we notice about the following results is that is that the proofs of many of the lemmas for predicate assignment are almost identical to their corresponding proofs for type assignment. Indeed, the assignment systems are similar enough that we may obtain a proof for the type assignment system simply by removing all references to predicates from the proofs for the predicate assignment system. Notwithstanding, the situation is not quite as simple as this because in most cases the proofs for the predicate assignment system rely on the result holding for the type assignment system. On closer inspection, this is not so remarkable since some rules of the predicate system require typeability. Thus, we have that the results for the type assignment system imply the results for the predicate system.

Many of the results follow by straightforward induction. To convince the reader of their correctness, we will show only a few cases in each proof. We first show some standard environment widening and thinning lemmas. These hold for both type assignment and predicate assignment. These lemmas are used to show substitution results for both subject reduction and expansion.

Lemma 5.1.1 (WIDENING FOR TYPE ASSIGNMENT)**.** If Γ and Γ' are both type environments such that $\Gamma \subseteq \Gamma'$ with $\chi \vdash \Gamma$ and $\chi \vdash \Gamma'$, then

$$\Gamma \Vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C} \Rightarrow \Gamma' \Vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C}$$

Proof. By straightforward induction on the structure of type derivations. We show only a few cases.

(T-VAR) Then $\mathrm{e} \equiv x$ and $x : \mathrm{C} \in \Gamma$ with $\chi \vdash \Gamma$. Since $\Gamma \subseteq \Gamma'$, it follows that $x : \mathrm{C} \in \Gamma'$. Also, by assumption we have that $\chi \vdash \Gamma'$ and so it we have by rule (T-VAR) that $\Gamma' \Vdash^{\mathrm{T}}_{\chi} x : \mathrm{C}$.

(T-ASS) Then $\mathrm{e} \equiv \mathrm{e}_1.f = \mathrm{e}_2$ and $\Gamma \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_1 : \mathrm{C}$ and $\Gamma \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_2 : \mathrm{D}$ such that $\Delta_{\mathrm{f}}(\chi, \mathrm{C}, f) = \mathrm{D}$. By the inductive hypothesis it follows that $\Gamma' \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_1 : \mathrm{C}$ and $\Gamma' \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_2 : \mathrm{D}$. Then, by rule (T-ASS) we have that $\Gamma' \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_1.f = \mathrm{e}_2 : \mathrm{C}$.

(T-SUB) Then $\Gamma \Vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C}'$ with $\mathrm{C}' <:_{\chi} \mathrm{C}$. By the inductive hypothesis it follows that $\Gamma' \Vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C}'$. Then, by rule (T-SUB) we have that $\Gamma' \Vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C}$.

□

Lemma 5.1.2 (WIDENING FOR PREDICATE ASSIGNMENT)**.** If Π and Π' are both predicate environments such that $\Pi \subseteq \Pi'$ with $\chi \vdash \Pi$ and $\chi \vdash \Pi'$, then

$$\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \phi \Rightarrow \Pi' \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \phi$$

Proof. By straightforward induction on the structure of predicate derivations. We show only a few cases.

(P-NULL) Then $\mathrm{e} \equiv (\mathrm{C})\,\texttt{null}$ and $\phi \equiv \langle\epsilon\rangle$ with $\chi \vdash \Pi$ and $\vdash_{\chi} \mathrm{C}$. By assumption, we have that $\chi \vdash \Pi'$ and so it follows by rule (P-NULL) that $\Pi' \Vdash^{\mathrm{P}}_{\chi} (\mathrm{C})\,\texttt{null} : \mathrm{C} : \langle\epsilon\rangle$.

(P-ASS$_2$) Then $\mathrm{e} \equiv \mathrm{e}_0.f = \mathrm{e}_1$ and $\phi = \langle l_i : \tau_i \ ^{i \in \overline{n}}\rangle$ such that $f \notin \overline{l}_n$ with $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0 : \mathrm{C} : \langle l_i : \tau_i \ ^{i \in \overline{n}}\rangle$ and $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_1 : \mathrm{D}$ such that $\Delta_{\mathrm{f}}(\chi, \mathrm{C}, f) = \mathrm{D}$. By the inductive hypothesis we have that $\Pi' \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0 : \mathrm{C} : \langle l_i : \tau_i \ ^{i \in \overline{n}}\rangle$. Also, by Definition 4.2.7 we have that $\widehat{\Pi} \subseteq \widehat{\Pi'}$, and so by Lemma 5.1.1 it follows that $\widehat{\Pi'} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_1 : \mathrm{D}$. Then, by rule (P-ASS$_2$), we have that $\Pi' \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0.f = \mathrm{e}_1 : \mathrm{C} : \langle l_i : \tau_i \ ^{i \in \overline{n}}\rangle$.

(P-OMEGA) Then $\phi = \omega$ with $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C}$. By definition 4.2.7 we have that $\widehat{\Pi} \subseteq \widehat{\Pi'}$, and so by Lemma 5.1.1 it follows that $\widehat{\Pi'} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C}$. Then, by rule (P-OMEGA), we have that $\Pi' \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \omega$.

(P-SEL$_1$) Then $\phi = \langle l : \tau\rangle$ and there is a sequence of class member labels $\overline{l}_n$ and a sequence of member types $\overline{\tau}_n$ with $n \geq 1$ such that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \langle l_i : \tau_i \ ^{i \in \overline{n}}\rangle$ with $l = l_j$ and $\tau = \tau_j$ for some $j \in \overline{n}$. By the inductive hypothesis it follows that $\Pi' \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \langle l_i : \tau_i \ ^{i \in \overline{n}}\rangle$ and then by rule (P-SEL$_1$) we have that $\Pi' \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \langle l_j : \tau_j\rangle$.

□

The following thinning lemmas show that typeable expressions can be typed using environments containing only statements about the variables that appear in the expression.

Lemma 5.1.3 (THINNING FOR TYPE ASSIGNMENT)**.** If Γ and Γ' are type environments such that $\Gamma' = \{x : \mathrm{D} \in \Gamma \mid x \in \textsc{vars}(\mathrm{e})\}$ then

$$\Gamma \Vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C} \Rightarrow \Gamma' \Vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C}$$

Proof. By straightforward induction on the structure of type derivations. We show only a few cases.

(T-VAR) Then $\mathrm{e} \equiv x$ and $x : \mathrm{C} \in \Gamma$ with $\chi \vdash \Gamma$. Since $\mathbb{VARS}^{\mathrm{T}}_{\mathrm{ENV}}(\Gamma) = \{x\}$ we have by definition that $x : \mathrm{C} \in \Gamma'$. Also, we have by Definition 4.1.2 that $\chi \vdash \Gamma'$ since $\Gamma' \subseteq \Gamma$. Then, by rule (T-VAR), it follows that $\Gamma' \Vdash^{\mathrm{T}}_{\chi} x : \mathrm{C}$.

(T-ASS) Then $\mathrm{e} \equiv \mathrm{e}_1.f = \mathrm{e}_2$ and $\Gamma \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_1 : \mathrm{C}$ with $\Gamma \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_2 : \mathrm{D}$ such that $\Delta_{\mathrm{f}}(\chi, \mathrm{C}, f) = \mathrm{D}$. By the inductive hypothesis, it follows that $\Gamma_1 \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_1 : \mathrm{C}$ and $\Gamma_2 \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_2 : \mathrm{D}$ where $\Gamma_1 = \{x : \mathrm{C} \in \Gamma \mid x \in \textsc{vars}(\mathrm{e}_1)\}$ and $\Gamma_2 = \{x : \mathrm{C} \in \Gamma \mid x \in \textsc{vars}(\mathrm{e}_2)\}$. By Definition 3.1.4, we can see that $\textsc{vars}(\mathrm{e}_1.f = \mathrm{e}_2) = \textsc{vars}(\mathrm{e}_1) \cup \textsc{vars}(\mathrm{e}_2)$, and so it follows that $\Gamma' = \{x : \mathrm{C} \in \Gamma \mid x \in \textsc{vars}(\mathrm{e}_1.f = \mathrm{e}_2)\} = \Gamma_1 \cup \Gamma_2$. It is clear that both $\Gamma_1 \subseteq \Gamma'$ and $\Gamma_2 \subseteq \Gamma'$. Also, since by assumption we have $\Gamma \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_1.f = \mathrm{e}_2 : \mathrm{C}$ it follows by Lemma 4.1.5 that $\chi \vdash \Gamma$. Also notice that $\Gamma' \subseteq \Gamma$ so by Definition 4.1.2 it also follows that $\chi \vdash \Gamma'$. Then, by Lemma 5.1.1, we have that $\Gamma' \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_1 : \mathrm{C}$ and $\Gamma' \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_2 : \mathrm{D}$. So, by rule (T-ASS), it follows that $\Gamma' \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_1.f = \mathrm{e}_2 : \mathrm{C}$.

(T-NEW) Then $\mathrm{e} \equiv \texttt{new}\ \mathrm{C}(\overline{\mathrm{e}}_n)$. This case is a generalised version of that for rule (T-ASS) above. By rule (T-NEW) we have that $\mathcal{F}(\chi, \mathrm{D}) = \overline{f}_n$ and $\Gamma \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_i : \mathrm{C}_i$ such that $\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f_i) = \mathrm{C}_i$ for each $i \in \overline{n}$. By the inductive hypothesis it follows that $\Gamma_i \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_i : \mathrm{C}_i$ with $\Gamma_i = \{x : \mathrm{C} \in \Gamma \mid x \in \textsc{vars}(\mathrm{e}_i)\}$ for each $i \in \overline{n}$. By Definition 3.1.4 we have that $\Gamma' = \Gamma_1 \cup \ \ldots \ \cup \Gamma_n$. By Lemma 4.1.5 it follows that $\chi \vdash \Gamma$ and so by Definition 4.1.2 that $\chi \vdash \Gamma'$. Also, since $\Gamma_i \subseteq \Gamma'$ for each $i \in \overline{n}$, by Lemma 5.1.1 we have that $\Gamma' \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_i : \mathrm{C}_i$ for each $i \in \overline{n}$. Then, by rule (T-NEW) it follows that $\Gamma' \Vdash^{\mathrm{T}}_{\chi} \texttt{new}\ \mathrm{C}(\overline{\mathrm{e}}_n) : \mathrm{C}$.

□

Lemma 5.1.4 (THINNING FOR PREDICATE ASSIGNMENT)**.** If Π and Π' are predicate environments such that $\Pi' = \{x : \mathrm{D} : \psi \in \Pi \mid x \in \textsc{vars}(\mathrm{e})\}$ then

$$\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \phi \Rightarrow \Gamma' \vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \phi$$

Proof. By straightforward induction on predicate derivations. We show only a few cases.

(P-NULL) Then $\mathrm{e} \equiv (\mathrm{C})\,\texttt{null}$ and $\phi \equiv \langle\epsilon\rangle$ with $\chi \vdash \Pi$ and $\vdash_{\chi} \mathrm{C}$. By Definition 3.1.4 we have that $\textsc{vars}((\mathrm{C})\,\texttt{null}) = \emptyset$. By Definition 4.2.5 it follows immediately that $\chi \vdash \emptyset$, and so by rule (P-NULL) we have that $\emptyset \vdash^{\mathrm{P}}_{\chi} (\mathrm{C})\,\texttt{null} : \mathrm{C} : \langle\epsilon\rangle$.

(P-NEWOBJ) Then $\mathrm{e} \equiv \texttt{new}\,\mathrm{C}(\overline{\mathrm{e}})$ and $\phi \equiv \langle\epsilon\rangle$ with $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \texttt{new}\,\mathrm{C}(\overline{\mathrm{e}}) : \mathrm{C}$. By Lemma 5.1.3 it follows that $\Gamma' \vdash^{\mathrm{T}}_{\chi} \texttt{new}\,\mathrm{C}(\overline{\mathrm{e}}) : \mathrm{C}$ where $\Gamma' = \{x : \mathrm{C} \in \widehat{\Pi} \mid x \in \textsc{vars}(\texttt{new}\,\mathrm{C}(\overline{\mathrm{e}}))\}$. By Definition 4.2.7 we have that $\widehat{\Pi'} = \Gamma'$ and so we have $\widehat{\Pi'} \vdash^{\mathrm{T}}_{\chi} \texttt{new}\,\mathrm{C}(\overline{\mathrm{e}}) : \mathrm{C}$. Then, by rule (P-NEWOBJ), it follows that $\Pi' \vdash^{\mathrm{P}}_{\chi} \texttt{new}\,\mathrm{C}(\overline{\mathrm{e}}) : \mathrm{C} : \langle\epsilon\rangle$.

(P-FLD) Then $\mathrm{e} \equiv \mathrm{e}'.f$ and $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}' : \mathrm{D} : \langle f : \phi\rangle$ such that $\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f) = \mathrm{C}$. By the inductive hypothesis it follows that $\Pi'' \vdash^{\mathrm{P}}_{\chi} \mathrm{e}' : \mathrm{D} : \langle f : \phi\rangle$ where $\Pi'' = \{x : \mathrm{C} : \phi \mid x \in \textsc{vars}(\mathrm{e}'.f)\}$. By Definition 3.1.4, $\textsc{vars}(\mathrm{e}'.f) = \textsc{vars}(\mathrm{e}')$ and so it follows that $\Pi' = \Pi''$. Then by rule (P-FLD) we have that $\Pi' \vdash^{\mathrm{P}}_{\chi} \mathrm{e}'.f : \mathrm{C} : \phi$.

(P-INVK) Then $\mathrm{e} \equiv \mathrm{e}_0.m(\overline{\mathrm{e}}_n)$ and $\phi \equiv \sigma$ with $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0 : \mathrm{D} : \langle m : \psi{::}\overline{\phi}_n \to \sigma\rangle$ and $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0 : \mathrm{D} : \psi$ such that $\Delta_{\mathrm{m}}(\chi, \mathrm{D}, m) = \overline{\mathrm{C}}_n \to \mathrm{C}$ and $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_i : \mathrm{C}_i : \phi_i$ for each $i \in \overline{n}$. By the inductive hypothesis we have that $\Pi_0 \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0 : \mathrm{D} : \langle m : \psi{::}\overline{\phi}_n \to \sigma\rangle$ and $\Pi_0 \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0 : \mathrm{D} : \psi$ with $\Pi_i \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_i : \mathrm{C}_i : \phi_i$ for each $i \in \overline{n}$ where $\Pi_k = \{x : \mathrm{C} : \phi \in \Pi \mid x \in \textsc{vars}(\mathrm{e}_i)\}$ for $0 \le k \le n$. By Definition 3.1.4, $\textsc{vars}(\mathrm{e}_0.m(\overline{\mathrm{e}}_n)) = \textsc{vars}(\mathrm{e}_0) \cup \ldots \cup \textsc{vars}(\mathrm{e}_n)$ and so it follows that $\Pi' = \Pi_0 \cup \ldots \cup \Pi_n$. Also, by Lemma 4.2.9, we have that $\chi \vdash \Pi$, and so by Definition 4.2.5 it follows that $\chi \vdash \Pi'$ since $\Pi' \subseteq \Pi$. Then, by Lemma 5.1.2 we have that $\Pi' \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0 : \mathrm{D} : \langle m : \psi{::}\overline{\phi}_n \to \sigma\rangle$ and $\Pi' \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0 : \mathrm{D} : \psi$ with $\Pi' \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_i : \mathrm{C}_i : \phi_i$ for each $i \in \overline{n}$. So, by rule (P-INVK) it follows that $\Pi' \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0.m(\overline{\mathrm{e}}_n) : \mathrm{C} : \sigma$.

□

We now show that variable substitution (the substitution of one variable for another in both an environment and an expression) is a *sound* operation. That is, substituting variables in a type or predicate derivation preserves the assignable type or predicate. This result will be used to effectively 'alpha-convert' type and predicate derivations in order to show general substitution and expansion results for sequences of expressions (Corollaries 5.1.9 and 5.1.12).

Lemma 5.1.5 (SOUNDNESS OF VARIABLE SUBSTITUTION FOR TYPE ASSIGNMENT)**.** If $[y/x]$ is a variable substitution and Γ is a type environment such that $\neg\exists \mathrm{D}\,[y : \mathrm{D} \in \Gamma]$, then

$$\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C} \Rightarrow \Gamma[y/x] \vdash^{\mathrm{T}}_{\chi} \mathrm{e}[y/x] : \mathrm{C}$$

Proof. By straightforward induction on the structure of type derivations. We show only a few cases.

(T-NULL) Then $\mathrm{e} \equiv (\mathrm{C})\,\texttt{null}$ with $\chi \vdash \Gamma$ and $\vdash_{\chi} \mathrm{C}$. Since, by Definition 3.3.1(1), $(\mathrm{C})\,\texttt{null}[y/x] = (\mathrm{C})\,\texttt{null}$, it follows trivially by rule (T-NULL) that $\Gamma[y/x] \vdash^{\mathrm{T}}_{\chi} (\mathrm{C})\,\texttt{null} : \mathrm{C}$.

(T-VAR) Then $\mathrm{e} \equiv z$ with $z : \mathrm{C} \in \Gamma$ and $\chi \vdash \Gamma$. Notice that the substitution does not introduce a duplicate statement with y as the subject since, by assumption, no statement with y as the subject exists in Γ. Therefore, we have that $\chi \vdash \Gamma[y/x]$. There are now two possibilities:

1. If $z = x$, then by Definition 4.1.3 it follows that $y : \mathrm{C} \in \Gamma[y/x]$. Then, since $x[y/x] = y$ by Definition 3.3.1(1), we have by rule (T-VAR) that $\Gamma[y/x] \vdash^{\mathrm{T}}_{\chi} y : \mathrm{C}$.
2. If $z \neq x$, then by Definition 4.1.3 it follows that $z : \mathrm{C} \in \Gamma[y/x]$. Then, since $z[y/x] = z$ by Definition 3.3.1(1), we have by rule (T-VAR) that $\Gamma[y/x] \vdash^{\mathrm{T}}_{\chi} z : \mathrm{C}$.

(T-INVK) Then $e \equiv e_0.m(\overline{e}_n)$ and $\Gamma \vdash^{T}_{\chi} e_0 : D$ such that $\Delta_m(\chi, D, m) = \overline{C}_n \rightarrow C$ with $\Gamma \vdash^{T}_{\chi} e_i : C_i$ for each $i \in \overline{n}$. By the inductive hypothesis it follows that $\Gamma[y/x] \vdash^{T}_{\chi} e_0[y/x] : D$ and $\Gamma[y/x] \vdash^{T}_{\chi} e_i[y/x] : C_i$ for each $i \in \overline{n}$. Then, by rule (T-INVK), we have that $\Gamma[y/x] \vdash^{T}_{\chi} (e_0[y/x]).m(e_1[y/x], \ldots, e_n[y/x]) : C$. By Definition 3.3.1(1) we have that $(e_0[y/x]).m(e_1[y/x], \ldots, e_n[y/x]) = e_0.m(\overline{e}_n)[y/x]$ and so it follows that $\Gamma[y/x] \vdash^{T}_{\chi} e_0.m(\overline{e}_n)[y/x] : C$.

(T-SUB) Then $\Gamma \vdash^{T}_{\chi} e : C'$ with $C' <:_{\chi} C$. By the inductive hypothesis it follows that $\Gamma[y/x] \vdash^{T}_{\chi} e[y/x] : C'$ and so by rule (T-SUB) we have that $\Gamma[y/x] \vdash^{T}_{\chi} e[y/x] : C$.

□

Lemma 5.1.6 (SOUNDNESS OF VARIABLE SUBSTITUTION FOR PREDICATE ASSIGNMENT)**.** If $[y/x]$ is a variable substitution and Π is a predicate environment such that $\neg\exists D, \phi' \; [y : D : \phi' \in \Pi]$, then

$$\Pi \vdash^{P}_{\chi} e : C : \phi \Rightarrow \Pi[y/x] \vdash^{P}_{\chi} e[y/x] : C : \phi$$

Proof. By straightforward induction on the structure of predicate derivations. The base case of rule (P-VAR) is identical to the corresponding case in the proof of Lemma 5.1.5 but for the presence of predicate information. We therefore omit it in this proof, and show only some inductive cases.

(P-FLD) Then $e \equiv e'.f$ and $\Pi \vdash^{P}_{\chi} e' : D : \langle f : \phi \rangle$ such that $\Delta_f(\chi, D, f) = C$. By the inductive hypothesis it follows that $\Pi[y/x] \vdash^{P}_{\chi} e'[y/x] : D : \langle f : \phi \rangle$, and so by rule (P-FLD) we have that $\Pi[y/x] \vdash^{P}_{\chi} (e'[y/x]).f : C : \phi$. Then, since by Definition 3.3.1(1) $(e'[y/x]).f = e'.f[y/x]$, it follows that $\Pi[y/x] \vdash^{P}_{\chi} e'.f[y/x] : C : \phi$.

(P-ASS$_2$) Then $e \equiv e_0.f = e_1$ and $\phi = \langle l_i : \tau_i \; ^{i \in \overline{n}} \rangle$ with $\Pi \vdash^{P}_{\chi} e_0 : C : \langle l_i : \tau_i \; ^{i \in \overline{n}} \rangle$ and $\widehat{\Pi} \vdash^{T}_{\chi} e_1 : D$ such that $f \notin \overline{l}_n$ and $\Delta_f(\chi, C, f) = D$. By the inductive hypothesis it follows that $\Pi[y/x] \vdash^{P}_{\chi} e_0[y/x] : C : \langle l_i : \tau_i \; ^{i \in \overline{n}} \rangle$. Also, by Lemma 5.1.5 we have that $\widehat{\Pi}[y/x] \vdash^{T}_{\chi} e_1[y/x] : D$. By Definition 4.2.7 it follows that $\widehat{\Pi}[y/x] = \widehat{\Pi[y/x]}$ and so we have that $\widehat{\Pi[y/x]} \vdash^{T}_{\chi} e_1[y/x] : D$. Then, by rule (P-ASS$_2$) it follows that $\Pi[y/x] \vdash^{P}_{\chi} (e_0[y/x]).f = (e_1[y/x]) : C : \langle l_i : \tau_i \; ^{i \in \overline{n}} \rangle$. Finally, by Definition 3.3.1(1) it follows that $((e_0[y/x]).f = (e_1[y/x])) = (e_0.f = e_1)[y/x]$ and so we have that $\Pi[y/x] \vdash^{P}_{\chi} (e_0.f = e_1)[y/x] : C : \langle l_i : \tau_i \; ^{i \in \overline{n}} \rangle$.

(P-OMEGA) Then $\phi = \omega$ and $\widehat{\Pi} \vdash^{T}_{\chi} e : C$. By Lemma 5.1.5 it follows that $\widehat{\Pi}[y/x] \vdash^{T}_{\chi} e[y/x] : C$. By Definition 4.2.7 it follows that $\widehat{\Pi}[y/x] = \widehat{\Pi[y/x]}$ and so we have that $\widehat{\Pi[y/x]} \vdash^{T}_{\chi} e[y/x] : C$. Then, by rule (P-OMEGA) it follows that $\Pi[y/x] \vdash^{P}_{\chi} e[y/x] : C : \omega$.

(P-SEL$_1$) Then $\phi = \langle l : \tau \rangle$ and there is a sequence of class member labels $\overline{l}_n$ and a sequence of member types $\overline{\tau}_n$ with $n \geq 1$ such that $\Pi \vdash^{P}_{\chi} e : C : \langle l_i : \tau_i \; ^{i \in \overline{n}} \rangle$ with $l = l_j$ and $\tau = \tau_j$ for some $j \in \overline{n}$. By the inductive hypothesis it follows that $\Pi[y/x] \vdash^{P}_{\chi} e[y/x] : C : \langle l_i : \tau_i \; ^{i \in \overline{n}} \rangle$ and so by rule (P-SEL$_1$) we have that $\Pi[y/x] \vdash^{P}_{\chi} e[y/x] : C : \langle l_j : \tau_j \rangle$.

□

We now come to the result that forms the basis for the subject reduction theorem. This substitution lemma states that types are preserved when expressions are substituted for variables where the type assumed for the variable matches the type of the expression being substituted. This holds for both type assignment and predicate assignment.

Lemma 5.1.7 (TYPE SUBSTITUTION LEMMA)**.** $\Gamma, x : D \vdash^{T}_{\chi} e : C \;\&\; \Gamma \vdash^{T}_{\chi} e' : D \Rightarrow \Gamma \vdash^{T}_{\chi} e[e'/x] : C$

Proof. By straightforward induction on the structure of type derivations. We show only a few cases.

(T-VAR) There are two cases to consider:

1. If $e \equiv x$, then by rule (T-VAR) $C = D$. From definition 3.3.1(1) $x[e'/x] = e'$ and so this case follows immediately since, by assumption, we have $\Gamma \vdash^{T}_{\chi} e' : D$.

2. If $\mathrm{e} \equiv y \neq x$, then by rule (T-VAR) $\Gamma, x : \mathrm{D} \vdash^{\mathrm{T}}_{\chi} y : \mathrm{C}$ and $y : \mathrm{C} \in \Gamma, x : \mathrm{D}$ with $\chi \vdash \Gamma$. Since $y \neq x$, it follows from definition 4.1.2 that $y : \mathrm{C} \in \Gamma$. Then, since by definition 3.3.1(1) $y[\mathrm{e}'/x] = y$, we have by rule (T-VAR) that $\Gamma \vdash^{\mathrm{T}}_{\chi} y : \mathrm{C}$.

(T-FLD) Then $\mathrm{e} \equiv \mathrm{e}'.f$, and $\Gamma, x : \mathrm{D} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}'.f : \mathrm{C}$. By rule (T-FLD) we have that $\Gamma, x : \mathrm{D} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}' : \mathrm{D}'$ such that $\Delta_{\mathrm{f}}(\chi, \mathrm{D}', f) = \mathrm{C}$. By the inductive hypothesis, it follows that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}'[\mathrm{e}'/x] : \mathrm{D}'$ and so by (T-FLD) $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}'[\mathrm{e}'/x].f : \mathrm{C}$. Since, by definition 3.3.1(1), $(\mathrm{e}'[\mathrm{e}'/x]).f = \mathrm{e}'.f[\mathrm{e}'/x]$, we have that $\Gamma \mathrm{e}'.f[\mathrm{e}'/x] \mathrm{C}$.

(T-INVK) Then $\mathrm{e} \equiv \mathrm{e}_0.m(\overline{\mathrm{e}}_n)$ and $\Gamma, x : \mathrm{D} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0.m(\overline{\mathrm{e}}_n) : \mathrm{C}$. By rule (T-INVK) we have that $\Gamma, x : \mathrm{D} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0 : \mathrm{C}_0$ and $\Delta_{\mathrm{m}}(\chi, \mathrm{C}_0, m) = \overline{\mathrm{C}}_n \to \mathrm{D}$ for some C_0. Also, $\Gamma, x : \mathrm{D} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_i : \mathrm{C}_i$ for each $i \in \overline{n}$. By the inductive hypothesis it follows that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0[\mathrm{e}'/x] : \mathrm{C}_0$ and $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_i[\mathrm{e}'/x] : \mathrm{C}_i$ for each $i \in \overline{n}$. Then, by rule (T-INVK), we have that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0[\mathrm{e}'/x].m(\mathrm{e}_1[\mathrm{e}'/x], \ldots, \mathrm{e}_n[\mathrm{e}'/x]) : \mathrm{C}$. It then follows from definition 3.3.1(1) that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0.m(\overline{\mathrm{e}}_n)[\mathrm{e}'/x] : \mathrm{C}$.

(T-NEW) Then $\mathrm{e} \equiv \texttt{new}\ \mathrm{C}(\overline{\mathrm{e}}_n)$ and $\Gamma, x : \mathrm{D} \vdash^{\mathrm{T}}_{\chi} \texttt{new}\ \mathrm{C}(\overline{\mathrm{e}}_n) : \mathrm{C}$. By rule (T-NEW) we have that $\mathcal{F}(\chi, \mathrm{C}) = \overline{f}_n$ with $\Gamma, x : \mathrm{D} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_i : \mathrm{C}_i$ such that $\Delta_{\mathrm{f}}(\chi, \mathrm{C}, f_i) = \mathrm{C}_i$ for each $i \in \overline{n}$. By the inductive hypothesis it follows that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_i[\mathrm{e}'/x] : \mathrm{C}_i$ for each $i \in \overline{n}$ and so, by rule (T-NEW), we have that $\Gamma \vdash^{\mathrm{T}}_{\chi} \texttt{new}\ \mathrm{C}(\mathrm{e}_1[\mathrm{e}'/x], \ldots, \mathrm{e}_n[\mathrm{e}'/x]) : \mathrm{C}$. Then from definition 3.3.1(1) it follows that $\Gamma \vdash^{\mathrm{T}}_{\chi} \texttt{new}\ \mathrm{C}(\overline{\mathrm{e}}_n)[\mathrm{e}'/x] : \mathrm{C}$.

□

Lemma 5.1.8 (PREDICATE SUBSTITUTION LEMMA)**.** $\Pi, x : \mathrm{D} : \phi' \vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \phi \ \&\ \Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}' : \mathrm{D} : \phi' \Rightarrow \Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}[\mathrm{e}'/x] : \mathrm{C} : \phi$

Proof. By straightforward induction on the structure of predicate derivations. Again, the base case of rule (P-VAR) is exactly the same as in the proof for type substitution but for the addition of predicate information. Therefore we will omit it. We show some other base cases and a few inductive cases.

(P-NULL) Then $\mathrm{e} \equiv (\mathrm{C})\ \texttt{null}$ and $\phi \equiv \langle\epsilon\rangle$ with $\chi \vdash \Pi$ and $\vdash_{\chi} \mathrm{C}$. From Definition 3.3.1(1) we have that $((\mathrm{C})\ \texttt{null})[\mathrm{e}'/x] = (\mathrm{C})\ \texttt{null}$ and so the result follows immediately from rule (P-NULL).

(P-NEWOBJ) Then $\mathrm{e} \equiv \texttt{new}\ \mathrm{C}(\overline{\mathrm{e}})$ and $\phi \equiv \langle\epsilon\rangle$ with $\widehat{\Pi, x : \mathrm{D} : \phi'} \vdash^{\mathrm{T}}_{\chi} \texttt{new}\ \mathrm{C}(\overline{\mathrm{e}}) : \mathrm{C}$. By Definition 4.2.7 we have that $\widehat{\Pi, x : \mathrm{D} : \phi'} = \widehat{\Pi}, x : \mathrm{D}$ and so we have $\widehat{\Pi}, x : \mathrm{D} \vdash^{\mathrm{T}}_{\chi} \texttt{new}\ \mathrm{C}(\overline{\mathrm{e}}) : \mathrm{C}$. Since, by assumption, we have $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}' : \mathrm{D} : \phi'$ it follows from Theorem 4.3.3 that $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}' : \mathrm{D}$. Then, by Lemma 5.1.7, we have that $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \texttt{new}\ \mathrm{C}(\overline{\mathrm{e}})[\mathrm{e}'/x] : \mathrm{C}$, and by rule (P-NEWOBJ) that $\Pi \vdash^{\mathrm{P}}_{\chi} \texttt{new}\ \mathrm{C}(\overline{\mathrm{e}})[\mathrm{e}'/x] : \mathrm{C} : \langle\epsilon\rangle$.

(P-OMEGA) Then $\phi \equiv \omega$ and $\widehat{\Pi, x : \mathrm{D} : \phi'} \vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C}$. By Definition 4.2.7 we have that $\widehat{\Pi, x : \mathrm{D} : \phi'} = \widehat{\Pi}, x : \mathrm{D}$ and so we have $\widehat{\Pi}, x : \mathrm{D} \vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C}$. Since by assumption we have $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}' : \mathrm{D} : \phi'$, it follows from Theorem 4.3.3 that $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}' : \mathrm{D}$. Then by Lemma 5.1.7 we have that $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}[\mathrm{e}'/x] : \mathrm{C}$, and by rule (P-OMEGA), we have that $\widehat{\Pi} \vdash^{\mathrm{P}}_{\chi} \mathrm{e}[\mathrm{e}'/x] : \mathrm{C} : \omega$.

(P-ASS$_1$) Then $\mathrm{e} \equiv \mathrm{e}_0.f = \mathrm{e}_1$ and $\phi \equiv \langle f : \phi''\rangle$, so $\Pi, x : \mathrm{D} : \phi' \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0.f = \mathrm{e}_1 : \mathrm{C} : \langle f : \phi''\rangle$ for some ϕ''. By rule (P-ASS$_1$) we have that $\Pi, x : \mathrm{D} : \phi' \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0 : \mathrm{C} : \sigma$ for some σ and $\Pi, x : \mathrm{D} : \phi' \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_1 : \mathrm{C}' : \phi''$ with $\Delta_{\mathrm{f}}(\chi, \mathrm{C}, f) = \mathrm{C}'$. Since, by assumption, we have that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}' : \mathrm{D} : \phi'$, it follows by the inductive hypothesis that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0[\mathrm{e}'/x] : \mathrm{C} : \sigma$ and $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_1[\mathrm{e}'/x] : \mathrm{C}' : \phi$. So by rule (P-ASS$_1$) it follows that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0[\mathrm{e}'/x].f = \mathrm{e}_1[\mathrm{e}'/x] : \mathrm{C} : \langle f : \phi\rangle$. Then, by Definition 3.3.1(1), it follows that $\Pi \vdash^{\mathrm{P}}_{\chi} (\mathrm{e}_0.f = \mathrm{e}_1)[\mathrm{e}'/x] : \mathrm{C} : \langle f : \phi\rangle$.

(P-ASS$_2$) Then $\mathrm{e} \equiv \mathrm{e}_0.f = \mathrm{e}_1$ and $\phi \equiv \langle l_i : \tau_i \ ^{i \in \overline{n}}\rangle$, so $\Pi, x : \mathrm{D} : \phi' \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0.f = \mathrm{e}_1 : \mathrm{C} : \langle l_i : \tau_i \ ^{i \in \overline{n}}\rangle$. By rule (P-ASS$_2$) we have that $\Pi, x : \mathrm{D} : \phi' \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0 : \mathrm{C} : \langle l_i : \tau_i \ ^{i \in \overline{n}}\rangle$ with $f \notin \overline{l}_n$ and $\widehat{\Pi, x : \mathrm{D} : \phi'} \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_1 : \mathrm{C}'$ such that $\Delta_{\mathrm{f}}(\chi, \mathrm{C}, f) = \mathrm{C}'$. Since by assumption we have $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}' : \mathrm{D} : \phi'$, it follows by the inductive hypothesis that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0[\mathrm{e}'/x] : \mathrm{C} : \langle l_i : \tau_i \ ^{i \in \overline{n}}\rangle$. Also, by Definition 4.2.7, $\widehat{\Pi, x : \mathrm{D} : \phi'} = \widehat{\Pi}, x : \mathrm{D}$ and so we have $\widehat{\Pi}, x : \mathrm{D} \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_1 : \mathrm{C}'$. By Theorem 4.3.3 we have $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}' : \mathrm{D}$, and so it follows by Lemma 5.1.7 that $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_1[\mathrm{e}'/x] : \mathrm{C}'$. Then, by rule (P-ASS$_2$) we have that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0[\mathrm{e}'/x].f = \mathrm{e}_1[\mathrm{e}'/x] : \mathrm{C} : \langle f' : \phi\rangle$, and by Definition 3.3.1(1) we have that $\Pi \vdash^{\mathrm{P}}_{\chi} (\mathrm{e}_0.f = \mathrm{e}_1)[\mathrm{e}'/x] : \mathrm{C} : \langle f' : \phi\rangle$.

□

We now show how the results in Lemmas 5.1.7 and 5.1.8 can be generalised to sequences of substitutions. That is, substituting a sequence of appropriately typed expressions for all the variables in a given expression preserves typeability. This corollary will be used in the case for method invocation in the subject reduction proofs. Referring to Definition 3.3.3, we see that invoking a method amounts to substituting the expressions passed as arguments for the variables in the body of that method. Thus, the following corollary shows that this process is type (and predicate) preserving.

Corollary 5.1.9 (SUBSTITUTION SEQUENCE)**.** Substituting a sequence expressions for a sequence of variables preserves typeability:

1. $\Gamma' \vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C} \;\&\; \forall\, i \in \overline{n}\, [\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_i : \mathrm{C}_i] \Rightarrow \Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}[\mathrm{e}_1/x_1, \ldots, \mathrm{e}_n/x_n] : \mathrm{C}$

2. $\Pi' \vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \phi \;\&\; \forall\, i \in \overline{n}\, [\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_i : \mathrm{C}_i : \phi_i] \Rightarrow \Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}[\mathrm{e}_1/x_1, \ldots, \mathrm{e}_n/x_n] : \mathrm{C} : \phi$

where $\Gamma' = \{x_1 : \mathrm{C}_1, \ldots, x_n : \mathrm{C}_n\}$, and $\Pi' = \{x_1 : \mathrm{C}_1 : \phi_1, \ldots, x_n : \mathrm{C}_n : \phi_1\}$.

Proof. In the following proof we will only deal with the case for type assignment. The case for predicate assignment follows by symmetry, using the corresponding substitution results for predicate assignment.

We begin the proof by first ensuring that the two type environments, Γ and Γ', are *disjoint* from one another. That is, the sets of variables used as subjects of the statements in each of the two respective environments are disjoint from one another. We achieve this by first constructing a sequence of n substitutions, $[y_1/x_1, \ldots, y_n/x_n]$, where each variable y_i is a fresh variable not occurring in either Γ or Γ' if the corresponding variable $x_i \in \mathbb{VARS}^{\mathrm{T}}_{\mathrm{ENV}}(\Gamma) \cap \mathbb{VARS}^{\mathrm{T}}_{\mathrm{ENV}}(\Gamma')$, or is equal to x_i otherwise. By the soundness of variable substitution (Lemma 5.1.5) we have that $\Gamma'[y_1/x_1, \ldots, y_n/x_n] \vdash^{\mathrm{T}}_{\chi} \mathrm{e}[y_1/x_1, \ldots, y_n/x_n] : \mathrm{C}$. Furthermore, we now have that $\mathbb{VARS}^{\mathrm{T}}_{\mathrm{ENV}}(\Gamma) \cap \mathbb{VARS}^{\mathrm{T}}_{\mathrm{ENV}}(\Gamma'[y_1/x_1, \ldots, y_n/x_n]) = \emptyset$ since all shared variables in Γ' have been renamed with fresh variables. In the remainder of the proof, for brevity we will write Γ'_{S} for $\Gamma'[y_1/x_1, \ldots, y_n/x_n] = \{y_1 : \mathrm{C}_1, \ldots, y_n : \mathrm{C}_n\}$, and e_{S} for $\mathrm{e}[y_1/x_1, \ldots, y_n/x_n]$.

We now proceed apply Lemma 5.1.7 for each variable y_i. First, notice that since $\Gamma'_{\mathrm{S}} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_{\mathrm{S}} : \mathrm{C}$ and $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_i : \mathrm{C}_i$ for each $i \in \overline{n}$, it follows from lemma 4.1.5 that both $\chi \vdash \Gamma'_{\mathrm{S}}$ and $\chi \vdash \Gamma$. Then, since the two environments are disjoint, we have $\chi \vdash \Gamma, \Gamma'_{\mathrm{S}}$. It is also clear that $\Gamma'_{\mathrm{S}} \subseteq \Gamma, \Gamma'_{\mathrm{S}}$. So, by the widening lemma (5.1.1) it follows that $\Gamma, \Gamma'_{\mathrm{S}} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_{\mathrm{S}} : \mathrm{C}$. We now construct the type environment $\Gamma, \Gamma'_{\mathrm{S}} \setminus \{y_1 : \mathrm{C}_1\}$ which, since it is strictly smaller than $\Gamma, \Gamma'_{\mathrm{S}}$, is also well-formed with respect to χ. Then, by Lemma 5.1.1 again, we have that $\Gamma, \Gamma'_{\mathrm{S}} \setminus \{y_1 : \mathrm{C}_1\} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_1 : \mathrm{C}_1$. We are now in a position to apply Lemma 5.1.7, from which it follows that $\Gamma, \Gamma'_{\mathrm{S}} \setminus \{y_1 : \mathrm{C}_1\} \vdash^{\mathrm{T}}_{\chi} \Gamma'_{\mathrm{S}}[\mathrm{e}_1/y_1] : \mathrm{C}$.

We now follow a similar process on each subsequent variable $y_k, 1 < k \leq n$. We construct the type environment $\Gamma, \Gamma'_{\mathrm{S}} \setminus \{y_1 : \mathrm{C}_1, y_2 : \mathrm{C}_2\}$ which is well-formed and by widening obtain that $\Gamma, \Gamma'_{\mathrm{S}} \setminus \{y_1 : \mathrm{C}_1, y_2 : \mathrm{C}_2\} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_2 : \mathrm{C}_2$. So, using the previous substitution result and the substitution lemma it follows that $\Gamma, \Gamma'_{\mathrm{S}} \setminus \{y_1 : \mathrm{C}_1, y_2 : \mathrm{C}_2\} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_{\mathrm{S}}[\mathrm{e}_1/y_1, \mathrm{e}_2/y_2] : \mathrm{C}$, and so on such that we obtain a sequence of results as follows:

$$\begin{array}{c}\Gamma, \Gamma'_{\mathrm{S}} \setminus \{y_1 : \mathrm{C}_1, y_2 : \mathrm{C}_2, y_3 : \mathrm{C}_3\} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_{\mathrm{S}}[\mathrm{e}_1/y_1, \mathrm{e}_2/y_2, \mathrm{e}_3/y_3] : \mathrm{C} \\ \vdots \\ \Gamma, \Gamma'_{\mathrm{S}} \setminus \{y_1 : \mathrm{C}_1, \ldots, y_n : \mathrm{C}_n\} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_{\mathrm{S}}[\mathrm{e}_1/y_1, \ldots, \mathrm{e}_n/y_n] : \mathrm{C}\end{array}$$

Since $\Gamma, \Gamma'_{\mathrm{S}} \setminus \{y_1 : \mathrm{C}_1, \ldots, y_n : \mathrm{C}_n\} = \Gamma$ we have that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_{\mathrm{S}}[\mathrm{e}_1/y_1, \ldots, \mathrm{e}_n/y_n] : \mathrm{C}$. Finally, since by Corollary 3.3.2 we have that $\mathrm{e}_{\mathrm{S}}[\mathrm{e}_1/y_1, \ldots, \mathrm{e}_n/y_n] = \mathrm{e}[\mathrm{e}_1/x_1, \ldots, \mathrm{e}_n/x_n]$, it follows that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}[\mathrm{e}_1/x_1, \ldots, \mathrm{e}_n/x_n] : \mathrm{C}$. □

the predicate join operation (Definition 4.2.3) displays the following two properties. They are necessary for demonstrating subject expansion since in a predicate derivation a sub-expression may occur more than once, with a different predicate assigned to it each time. During expansion, the sub-expression will be replaced by a variable, which may only be associated with a single predicate. We generate this predicate by combining the predicates for each occurrence of the sub-expression using the join operator. The following lemma shows us that, once we join a number of predicates together, for each occurrence of a variable we are able to able to give the predicate originally assigned to the particular occurrence of the sub-expression which has been replaced.

Lemma 5.1.10 (PREDICATE JOIN LEMMA)**.**

1. $\exists\, \overline{\phi_n}\, [\forall\, i \in \overline{n}\, [\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \phi_i]] \Rightarrow \Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \bigsqcup_{i \in \overline{n}} \phi_i$

2. $\exists\, \overline{\psi_n}, j \in \overline{n}\, [\Pi, x : \mathrm{D} : \psi_j \vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \phi] \Rightarrow \Pi, x : \mathrm{D} : \bigsqcup_{i \in \overline{n}} \psi_i \vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \phi$

Proof. 1. We first discard all $\phi_i = \omega$, since ω is the identity element for the predicate join operation. We are then left with a subsequence of predicates, $\overline{\phi_{n'}}$ such that $\phi_k \in \overline{\phi_n}$ for all $k \in \overline{n'}$, with each $\phi_k \equiv \langle \overline{l^k_i : \tau^k_i}^{\,i \in \overline{n_k}} \rangle$. By Definition 4.2.3, we note that $\bigsqcup_{i \in \overline{n}} \phi = \psi = \langle l^1_1 : \tau^1_1, \ldots, l^{n'}_{n_{n'}} : \tau^{n'}_{n_{n'}} \rangle$. So, for each $k \in \overline{n'}$, by rule (P-SEL$_1$) we have that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \langle l^k_1 : \tau^k_1 \rangle, \ldots, \Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \langle l^k_{n_k} : \tau^k_{n_k} \rangle$. Then, it follows by rule (P-SEQ) that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \psi$, and thus that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \bigsqcup_{i \in \overline{n}} \phi i$.

2. This result follows by induction on the structure of predicate derivations. The only non-trivial case is that for the variable x. We note that, whenever $\Pi, x : \mathrm{D} : \psi_j \vdash^{\mathrm{P}}_{\chi} x : \mathrm{D} : \phi'$, we can also derive $\Pi, x : \mathrm{D} : \bigsqcup_{i \in \overline{n}} \psi_i \vdash^{\mathrm{P}}_{\chi} x : \mathrm{D} : \phi'$. There are three cases to consider:

   ($\phi' = \omega$) By Definition 4.2.7, we have that $x : \mathrm{D} \in \widehat{\Pi, x : \mathrm{D} : \bigsqcup_{i \in \overline{n}} \psi_i}$ and so we have by rule (T-VAR) that $\widehat{\Pi, x : \mathrm{D} : \bigsqcup_{i \in \overline{n}} \psi_i} \vdash^{\mathrm{T}}_{\chi} x : \mathrm{D}$. Then, by rule (P-OMEGA), it follows that $\Pi, x : \mathrm{D} : \bigsqcup_{i \in \overline{n}} \psi_i \vdash^{\mathrm{P}}_{\chi} x : \mathrm{D} : \omega$.

   ($\phi' = \langle \epsilon \rangle$) By rule (P-VAR) we have that $\Pi, x : \mathrm{D} : \bigsqcup_{i \in \overline{n}} \psi_i \vdash^{\mathrm{P}}_{\chi} x : \mathrm{D} : \bigsqcup_{i \in \overline{n}} \psi_i$. Then, by rule (P-SEL$_2$) it follows that $\Pi, x : \mathrm{D} : \bigsqcup_{i \in \overline{n}} \psi_i \vdash^{\mathrm{P}}_{\chi} x : \mathrm{D} : \langle \epsilon \rangle$.

   ($\phi' = \sigma \neq \langle \epsilon \rangle$) We first note that it must be the case that $\psi_j \trianglelefteq \sigma$. Then, by rule (P-VAR) we have that $\Pi, x : \mathrm{D} : \bigsqcup_{i \in \overline{n}} \psi_i \vdash^{\mathrm{P}}_{\chi} x : \mathrm{D} : \bigsqcup_{i \in \overline{n}} \psi_i$. Then, since by Definition 4.2.3, $\bigsqcup_{i \in \overline{n}} \psi_i \trianglelefteq \psi_j \trianglelefteq \sigma$, it follows by Theorem 4.3.2 that $\Pi, x : \mathrm{D} : \bigsqcup_{i \in \overline{n}} \psi_i \vdash^{\mathrm{P}}_{\chi} x : \mathrm{D} : \sigma$.

□

The following lemma forms the key step in the proof for subject expansion in the same way that the substitution lemmas (5.1.7 and 5.1.8) form the basis of the proof for subject reduction. It states that if we can assign a type-predicate pair to an expression e in which a sub-expression e′ has been substituted for a variable x, then we can also assign that type-predicate pair to the original expression in which the sub-expression has not been substituted, under an appropriate assumption about the type of the variable.

Lemma 5.1.11 (EXPANSION LEMMA).

$$\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}[\mathrm{e}'/x] : \mathrm{C} : \phi \;\&\; \widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}' : \mathrm{C}' \;\&\; \widehat{\Pi}, x : \mathrm{C}' \vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C} \Rightarrow \exists\, \psi\, [\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}' : \mathrm{C}' : \psi \;\&\; \Pi, x : \mathrm{C}' : \psi \vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \phi]$$

Proof. We deal with the cases where $\phi \equiv \omega$ and $\phi \equiv \sigma$ separately:

($\phi \equiv \omega$) This case follows immediately since we can take $\psi = \omega$: by assumption, $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}' : \mathrm{C}'$ and so by rule (P-OMEGA) it follows that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}' : \mathrm{C}' : \omega$. Similarly, since by assumption $\widehat{\Pi}, x : \mathrm{C}' \vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C}$ and by Definition 4.2.7 $\widehat{\Pi}, x : \mathrm{C}' = \widehat{\Pi, x : \mathrm{C}' : \omega}$, rule (P-OMEGA) gives us that $\Pi, x : \mathrm{C}' : \omega \vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \omega$.

($\phi \equiv \sigma$) This case proceeds by induction on the structure of e. We show the cases for variables, and field assignment. We also show method invocation, since it shows how the predicate join lemma (5.1.10) is used. The base case for the null object follows easily by taking $\psi = \omega$ since the substituted expression e′ does not appear. The case for field access follows straightforwardly by induction, and the case for object creation is similar to that for field assignment.

   ($\mathrm{e} \equiv x$) Then by Definition 3.3.1(1) we have that $\mathrm{e}[\mathrm{e}'/x] = \mathrm{e}'$, and so $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}' : \mathrm{C} : \phi$. Since we also have by assumption that $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}' : \mathrm{C}'$, it follows by Theorem 4.3.3 that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}' : \mathrm{C}' : \phi$, and so we can take $\psi = \phi$. Since $\mathrm{e} \equiv x$, and by assumption we have $\widehat{\Pi}, x : \mathrm{C}' \vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C}$, it follows by Lemma 4.1.6(2) that $\mathrm{C}' <:_{\chi} \mathrm{C}$. Furthermore, by Lemma 4.1.5 it follows that $\chi \vdash \widehat{\Pi}, x : \mathrm{C}'$ and so by Definition 4.2.7 that $\chi \vdash \Pi, x : \mathrm{C}' : \phi$. Then, by rule (P-VAR) it also follows that $\Pi, x : \mathrm{C}' : \phi \vdash^{\mathrm{P}}_{\chi} x : \mathrm{C}' : \phi$, and by rule (P-SUBTYPE) that $\Pi, x : \mathrm{C}' : \phi \vdash^{\mathrm{P}}_{\chi} x : \mathrm{C} : \phi$.

   ($\mathrm{e} \equiv y \neq x$) Then by Definition 3.3.1(1) we have that $\mathrm{e}[\mathrm{e}'/x] = y$, and so $\Pi \vdash^{\mathrm{P}}_{\chi} y : \mathrm{C} : \phi$. Then, by Lemma 4.2.10(2), it follows that $\exists\, \mathrm{C}'' <:_{\chi} \mathrm{C}$ and $\phi' \trianglelefteq \phi$ such that $y : \mathrm{C}'' : \phi' \in \Pi$. Since, by assumption, $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}' : \mathrm{C}'$, we can take $\psi = \omega$ by rule (P-OMEGA). Then, since by assumption we also have that $\widehat{\Pi}, x : \mathrm{C}' \vdash^{\mathrm{T}}_{\chi} y : \mathrm{C}$, it follows from Lemma 4.2.9 that $\chi \vdash \widehat{\Pi}, x : \mathrm{C}'$, and so by Definition 4.2.7 that $\chi \vdash \Pi, x : \mathrm{C}' : \omega$. Therefore, by rule (P-VAR) we have that $\Pi, x : \mathrm{C}' : \omega \vdash^{\mathrm{P}}_{\chi} y : \mathrm{C}'' : \phi'$. Then by rule (P-SUBTYPE) it follows that $\Pi, x : \mathrm{C}' : \omega \vdash^{\mathrm{P}}_{\chi} y : \mathrm{C} : \phi'$ and by Theorem 4.3.2 that $\Pi, x : \mathrm{C}' : \omega \vdash^{\mathrm{P}}_{\chi} y : \mathrm{C} : \phi$.

($e \equiv e_1.f = e_2$) Then by Definition 3.3.1(1) we have that $e[e'/x] \equiv e_1[e'/x].f = e_2[e'/x]$ and $\Pi \Vdash^{\mathrm{P}}_{\chi} e_1[e'/x].f = e_2[e'/x] : C : \phi$. By assumption we also have that $\widehat{\Pi, x : C'} \Vdash^{\mathrm{T}}_{\chi} e_1.f = e_2 : C$ and so by Lemma 4.1.6(4) it follows that $\exists\, D <:_{\chi} C$ such that $\widehat{\Pi, x : C'} \Vdash^{\mathrm{T}}_{\chi} e_1 : D$ with $\Delta_{\mathrm{f}}(\chi, D, f) = D'$ and $\widehat{\Pi, x : C'} \Vdash^{\mathrm{T}}_{\chi} e_2 : D'$. Then, since by assumption we have $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} e' : C'$, by Lemma 5.1.7 it also follows that $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} e_1[e'/x] : D$ and $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} e_2[e'/x] : D'$.

If $\sigma = \langle \epsilon \rangle$, then by rule (p-ass$_2$) we have that $\Pi \Vdash^{\mathrm{P}}_{\chi} e_1[e'/x] : C : \langle \epsilon \rangle$ with $\Delta_{\mathrm{f}}(\chi, C, f) = C''$ and $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} e_2[e'/x] : C''$. As shown above, we also have $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} e_1[e'/x] : D$ and so by Theorem 4.3.3 it follows that $\Pi \Vdash^{\mathrm{P}}_{\chi} e_1[e'/x] : D : \langle \epsilon \rangle$. Then by the inductive hypothesis, $\exists\, \psi$ such that $\Pi \Vdash^{\mathrm{P}}_{\chi} e' : C' : \psi$ and $\Pi, x : C' : \psi \Vdash^{\mathrm{P}}_{\chi} e_1 : D : \langle \epsilon \rangle$. Now, since by Definition 4.2.7, $\widehat{\Pi, x : C'} = \widehat{\Pi, x : C' : \psi}$, and $\widehat{\Pi, x : C'} \Vdash^{\mathrm{T}}_{\chi} e_2 : D'$ from above, it follows that $\widehat{\Pi, x : C' : \psi} \Vdash^{\mathrm{T}}_{\chi} e_2 : D'$. Then we have by rule (p-ass$_2$) that $\Pi, x : C' : \psi \Vdash^{\mathrm{P}}_{\chi} e_1.f = e_2 : D : \langle \epsilon \rangle$. Then, since $D <:_{\chi} C$ we have by rule (p-subType) that $\Pi, x : C' : \psi \Vdash^{\mathrm{P}}_{\chi} e_1.f = e_2 : C : \langle \epsilon \rangle$.

If $\sigma = \langle l_i : \tau_i\ ^{i \in \overline{n}} \rangle \not\equiv \langle \epsilon \rangle$, then by Lemma 4.2.10(4), $\exists\, \overline{D}_n$ such that $D_i <:_{\chi} C$ and $\Pi \Vdash^{\mathrm{P}}_{\chi} e_1[e'/x].f = e_2[e'/x] : D_i : \langle l_i : \tau_i \rangle$ for each $i \in \overline{n}$. Now, take any $i \in \overline{n}$; there are two possibilities:

($f = l_i$) Then $\tau_i = \phi'$ and by rule (p-ass$_1$) it follows that $\Pi \Vdash^{\mathrm{P}}_{\chi} e_1[e'/x] : D_i : \sigma'$ for some σ', with $\Delta_{\mathrm{f}}(\chi, D_i, f) = C_i$ and $\Pi \Vdash^{\mathrm{P}}_{\chi} e_2[e'/x] : C_i : \phi'$. Since we also have, from above, that $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} e_1[e'/x] : D$ and $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} e_2[e'/x] : D'$, it follows by Theorem 4.3.3 that $\Pi \Vdash^{\mathrm{P}}_{\chi} e_1[e'/x] : D : \sigma'$ and $\Pi \Vdash^{\mathrm{P}}_{\chi} e_2[e'/x] : D' : \phi'$. Then by the inductive hypothesis $\exists\, \psi_1$ and ψ_2 such that $\Pi \Vdash^{\mathrm{P}}_{\chi} e' : C' : \psi_1$ and $\Pi \Vdash^{\mathrm{P}}_{\chi} e' : C' : \psi_2$ with $\Pi, x : C' : \psi_1 \Vdash^{\mathrm{P}}_{\chi} e_1 : D : \sigma'$ and $\Pi, x : C' : \psi_2 \Vdash^{\mathrm{P}}_{\chi} e_2 : D' : \phi'$. We take $\psi_i = \psi_1 \sqcup \psi_2$. By lemma 5.1.10 we have that $\Pi \Vdash^{\mathrm{P}}_{\chi} e' : C' : \psi_i$, in addition to $\Pi, x : C' : \psi_i \Vdash^{\mathrm{P}}_{\chi} e_1 : D : \sigma'$ and $\Pi, x : C' : \psi_i \Vdash^{\mathrm{P}}_{\chi} e_2 : D' : \phi'$. So, by rule (p-ass$_1$) it follows that $\Pi, x : C' : \psi_i \Vdash^{\mathrm{P}}_{\chi} e_1.f = e_2 : D : \langle f : \phi' \rangle$, and by rule (p-subType) that $\Pi, x : C' : \psi_i \Vdash^{\mathrm{P}}_{\chi} e_1.f = e_2 : C : \langle f : \phi' \rangle$.

($f \neq l_i$) This case proceeds in exactly the same fashion as the case ($\sigma = \langle \epsilon \rangle$) above. So we have that $\exists\, \psi_i$ such that $\Pi \Vdash^{\mathrm{P}}_{\chi} e' : C' : \psi_i$ and $\Pi, x : C' : \psi_i \Vdash^{\mathrm{P}}_{\chi} e_1.f = e_2 : C : \langle l_i : \tau_i \rangle$.

Since the choice of i was arbitrary, we have that $\exists\, \overline{\psi}_n$ such that $\Pi \Vdash^{\mathrm{P}}_{\chi} e' : C' : \psi_i$ and $\Pi, x : C' : \psi_i \Vdash^{\mathrm{P}}_{\chi} e_1.f = e_2 : C : \langle l_i : \tau_i \rangle$ for each $i \in \overline{n}$. We take $\psi = \bigsqcup_{i \in \overline{n}} \psi_i$. Then by Lemma 5.1.10 it follows that $\Pi \Vdash^{\mathrm{P}}_{\chi} e' : C' : \psi$ and $\Pi, x : C' : \psi \Vdash^{\mathrm{P}}_{\chi} e_1.f = e_2 : C : \langle l_i : \tau_i \rangle$ for each $i \in \overline{n}$. We then have by rule (p-seq) that $\Pi, x : C' : \psi \Vdash^{\mathrm{P}}_{\chi} e_1.f = e_2 : C : \langle l_i : \tau_i\ ^{i \in \overline{n}} \rangle$.

($e \equiv e_0.m(\overline{e}_{n_1})$) Then by Definition 3.3.1(1), we have $e[e'/x] = e_0[e'/x].m(e_1[e'/x], \ldots, e_{n_1}[e'/x])$ and therefore that $\Pi \Vdash^{\mathrm{P}}_{\chi} e_0[e'/x].m(e_1[e'/x], \ldots, e_{n_1}[e'/x]) : C : \sigma$. Then, by Lemma 4.2.10(5) it follows that $\exists\, D. \overline{C}_{n_1}, C'' <:_{\chi} C, \psi_0, \overline{\phi}_{n_1}, \sigma' \trianglelefteq \sigma$ such that $\Pi \Vdash^{\mathrm{P}}_{\chi} e_0[e'/x] : D : \langle m : \psi_0 :: \overline{\phi}_{n_1} \to \sigma' \rangle$ and $\Pi \Vdash^{\mathrm{P}}_{\chi} e_0[e'/x] : D : \psi_0$ with $\Delta_{\mathrm{m}}(\chi, D, m) = \overline{C}_{n_1} \to C''$ and $\Pi \Vdash^{\mathrm{P}}_{\chi} e_k[e'/x] : C_k : \phi_k$ for each $k \in \overline{n_1}$. By assumption, we also have $\widehat{\Pi, x : C'} \Vdash^{\mathrm{T}}_{\chi} e_0.m(\overline{e}_{n_1}) : C$, and so by Lemma 4.1.6(5) it follows that $\exists\, D'\ \overline{C'}_{n_1}, C'''$ such that $\widehat{\Pi, x : C'} \Vdash^{\mathrm{T}}_{\chi} e_0 : D'$ with $\Delta_{\mathrm{m}}(\chi, D', m) = \overline{C'}_{n_1} \to C'''$ and $\widehat{\Pi, x : C'} \Vdash^{\mathrm{T}}_{\chi} e_k : C'_k$ for each $k \in \overline{n_1}$. Furthermore, since we have by assumption that $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} e' : C'$, it follows by Lemma 5.1.7 that $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} e_0[e'/x] : D'$ and $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} e_k[e'/x] : C'_k$ for each $k \in \overline{n_1}$. Then, by Theorem 4.3.3, we have that $\Pi \Vdash^{\mathrm{P}}_{\chi} e_0[e'/x] : D' : \langle m : \psi_0 :: \overline{\phi}_{n_1} \to \sigma' \rangle$ and $\Pi \Vdash^{\mathrm{P}}_{\chi} e_0[e'/x] : D' : \psi_0$ with $\Pi \Vdash^{\mathrm{P}}_{\chi} e_k[e'/x] : C'_k : \phi_k$ for each $k \in \overline{n_1}$. Now, by the inductive hypothesis it follows that $\exists\, \psi'$ and ψ'' such that $\Pi \Vdash^{\mathrm{P}}_{\chi} e' : C' : \psi'$ and $\Pi \Vdash^{\mathrm{P}}_{\chi} e' : C' : \psi''$ with $\Pi, x : C' : \psi' \Vdash^{\mathrm{P}}_{\chi} e_0 : D' : \langle m : \psi_0 :: \overline{\phi}_{n_1} \to \sigma' \rangle$ and $\Pi, x : C' : \psi'' \Vdash^{\mathrm{P}}_{\chi} e_0 : D' : \psi_0$. We also have by the inductive hypothesis that $\exists\, \overline{\psi}_{n_1}$ such that $\Pi \Vdash^{\mathrm{P}}_{\chi} e' : C' : \psi_k$ and $\Pi, x : C' : \psi_k \Vdash^{\mathrm{P}}_{\chi} e_k : C'_k : \phi_k$ for each $k \in \overline{n_1}$. Let us now take $\psi = \psi' \sqcup \psi'' \sqcup \bigsqcup_{i \in \overline{n_1}} \psi_i$. By Lemma 5.1.10(1) we have that $\Pi \Vdash^{\mathrm{P}}_{\chi} e' : C' : \psi$, and by Lemma 5.1.10(2) it also follows that $\Pi, x : C' : \psi \Vdash^{\mathrm{P}}_{\chi} e_0 : D' : \langle m : \psi_0 :: \overline{\phi}_{n_1} \to \sigma' \rangle$ and $\Pi, x : C' : \psi \Vdash^{\mathrm{P}}_{\chi} e_0 : D' : \psi_0$ with $\Pi, x : C' : \psi \Vdash^{\mathrm{P}}_{\chi} e_k : C'_k : \phi_k$ for each $k \in \overline{n_1}$. Then, by rule (p-invk) we have that $\Pi, x : C' : \psi \Vdash^{\mathrm{P}}_{\chi} e_0.m(\overline{e}_{n_1}) : C''' : \sigma'$. Since $C''' <:_{\chi} C$ it follows from rule (p-subType) that $\Pi, x : C' : \psi \Vdash^{\mathrm{P}}_{\chi} e_0.m(\overline{e}_{n_1}) : C : \sigma'$. Further more, since $\sigma' \trianglelefteq \sigma$ we have by Theorem 4.3.2 that $\Pi, x : C' : \psi \Vdash^{\mathrm{P}}_{\chi} e_0.m(\overline{e}_{n_1}) : C : \sigma$.

□

Just as we were able to generalise the substitution lemma to a sequence of expressions, we can do the same thing with the expansion lemma. This is necessary since, in general, methods take a sequence of arguments. The proof is very similar to the proof for Corollary 5.1.9. We again use the widening and thinning lemmas, and combine them with the expansion lemma proved above to make the generalisation to sequences of expressions.

Corollary 5.1.12 (EXPANSION SEQUENCE)**.** We now show how the result in Lemma 5.1.11 can be generalised to a sequence of substitutions. That is

$$\begin{array}{l}\Pi \models^{\mathrm{P}}_{\chi} \mathrm{e}[\mathrm{e}_1/x_1,\ldots,\mathrm{e}_n/x_n] : \mathrm{C} : \phi \;\&\; \forall\, i \in \overline{n}\; [\widehat{\Pi} \models^{\mathrm{T}}_{\chi} \mathrm{e}_i : \mathrm{C}_i] \;\&\; \Gamma' \models^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C} \\ \quad \Rightarrow \exists\, \overline{\phi}_n\; [\Pi' \models^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \phi \;\&\; \forall\, i \in \overline{n}\; [\Pi \models^{\mathrm{P}}_{\chi} \mathrm{e}_i : \mathrm{C}_i : \phi_i]]\end{array}$$

where $\Gamma' = \{x_1 : \mathrm{C}_1,\ldots,x_n : \mathrm{C}_n\}$ and $\Pi' = \{x_1 : \mathrm{C}_1 : \phi_1,\ldots,x_n : \mathrm{C}_n : \phi_n\}$.

Proof. We begin the proof by ensuring that the two type environments Γ' and $\widehat{\Pi}$ are disjoint, as in the proof of Corollary 5.1.9: we refer to that proof for the details. We thus construct the new expression e_{s} and the new type environment Γ'_{s} such that $\Gamma'_{\mathrm{s}} \models^{\mathrm{T}}_{\chi} \mathrm{e}_{\mathrm{s}} : \mathrm{C}$ and $\Pi \models^{\mathrm{P}}_{\chi} \mathrm{e}_{\mathrm{s}}[y_1/x_1,\ldots,y_n/x_n] : \mathrm{C} : \phi$.

In the proof of Corollary 5.1.9, we demonstrated how to derive a series of (intermediate) results as follows: $\widehat{\Pi}, \{y_n : \mathrm{C}_n\} \models^{\mathrm{T}}_{\chi} \mathrm{e}_{\mathrm{s}}[\mathrm{e}_1/y_1,\ldots,\mathrm{e}_{n-1}/y_{n-1}] : \mathrm{C}$, $\widehat{\Pi}, \{y_{n-1} : \mathrm{C}_{n-1}, y_n : \mathrm{C}_n\} \models^{\mathrm{T}}_{\chi} \mathrm{e}_{\mathrm{s}}[\mathrm{e}_1/y_1,\ldots,\mathrm{e}_{n-2}/y_{n-2}] : \mathrm{C}$, …, $\widehat{\Pi}, \{y_1 : \mathrm{C}_1,\ldots,y_n : \mathrm{C}_n\} \models^{\mathrm{T}}_{\chi} \mathrm{e}_{\mathrm{s}} : \mathrm{C}$. We will also use these results in the following proof, which proceeds by expanding each variable in turn, in reverse order. That is, we first expand y_n by substituting the expression e_n, and end by expanding y_1.

We start with the derived result $\widehat{\Pi}, \{y_n : \mathrm{C}_n\} \models^{\mathrm{T}}_{\chi} \mathrm{e}_{\mathrm{s}}[\mathrm{e}_1/y_1,\ldots,\mathrm{e}_{n-1}/y_{n-1}] : \mathrm{C}$ from Corollary 5.1.9. By assumption we also have $\widehat{\Pi} \models^{\mathrm{T}}_{\chi} \mathrm{e}_n : \mathrm{C}_n$ and $\Pi \models^{\mathrm{P}}_{\chi} [\mathrm{e}_1/y_1,\ldots,\mathrm{e}_n/y_n] : \mathrm{C} : \phi \equiv \Pi \models^{\mathrm{P}}_{\chi} \mathrm{e}_{\mathrm{s}}[\mathrm{e}_1/y_1,\ldots,\mathrm{e}_{n-1}/y_{n-1}][\mathrm{e}_n/y_n] : \mathrm{C} : \phi$. So, by Lemma 5.1.11 it follows that $\exists\, \phi_n$ such that $\Pi \models^{\mathrm{P}}_{\chi} \mathrm{e}_n : \mathrm{C}_n : \phi_n$ and $\Pi, y : \mathrm{C}_n : \phi_n \models^{\mathrm{P}}_{\chi} \mathrm{e}_{\mathrm{s}}[\mathrm{e}_1/y_1,\ldots,\mathrm{e}_{n-1}/y_{n-1}] : \mathrm{C} : \phi$.

We now expand the variable y_{n-1}. Notice that the last result is equivalent to the following $\Pi, y : \mathrm{C}_n : \phi_n \models^{\mathrm{P}}_{\chi} \mathrm{e}_{\mathrm{s}}[\mathrm{e}_1/y_1,\ldots,\mathrm{e}_{n-2}/y_{n-2}][\mathrm{e}_{n-1}/y_{n-1}] : \mathrm{C} : \phi$. By assumption, we have $\widehat{\Pi} \models^{\mathrm{T}}_{\chi} \mathrm{e}_{n-1} : \mathrm{C}_{n-1}$. Since both $\chi \vdash \widehat{\Pi}$ and $\chi \vdash \Gamma'_{\mathrm{s}}$, with $\widehat{\Pi}$ and Γ'_{s} disjoint, we have that $\chi \vdash \widehat{\Pi}, y_n : \mathrm{C}_n$, and $\widehat{\Pi} \subseteq \widehat{\Pi}, y_n : \mathrm{C}_n$. Then, by Lemma 5.1.1 it follows that $\widehat{\Pi}, y : \mathrm{C}_n \models^{\mathrm{T}}_{\chi} \mathrm{e}_{n-1} : \mathrm{C}_{n-1}$. By Definition 4.2.7, $\widehat{\Pi}, y : \mathrm{C}_n = \widehat{\Pi, y_n : \mathrm{C}_n : \phi_n}$ and so we have that $\widehat{\Pi, y : \mathrm{C}_n : \phi_n} \models^{\mathrm{T}}_{\chi} \mathrm{e}_{n-1} : \mathrm{C}_{n-1}$. We now use another derived result from Corollary 5.1.9: $(\widehat{\Pi}, y_n : \mathrm{C}_n), y_{n-1} : \mathrm{C}_{n-1} \models^{\mathrm{T}}_{\chi} \mathrm{e}_{\mathrm{s}}[\mathrm{e}_1/y_1,\ldots,\mathrm{e}_{n-2}/y_{n-2}] : \mathrm{C}$. From Lemma 5.1.11 it follows that $\exists\, \phi_{n-1}$ such that $\Pi, y_n : \mathrm{C}_n : \phi_n \models^{\mathrm{P}}_{\chi} \mathrm{e}_{n-1} : \mathrm{C}_{n-1} : \phi_{n-1}$ and $\Pi, \{y_n : \mathrm{C}_n : \phi_n, y_{n-1} : \mathrm{C}_{n-1} : \phi_{n-1}\} \models^{\mathrm{P}}_{\chi} \mathrm{e}_{\mathrm{s}}[\mathrm{e}_1/y_1,\ldots,\mathrm{e}_{n-2}/y_{n-2}] : \mathrm{C} : \phi$. Now, it remains to show that from these results, we can obtain $\Pi \models^{\mathrm{P}}_{\chi} \mathrm{e}_{n-1} : \mathrm{C}_{n-1} : \phi_{n-1}$. By Lemma 5.1.4 it follows that $\Pi_{n-1} \models^{\mathrm{P}}_{\chi} \mathrm{e}_{n-1} : \mathrm{C}_{n-1} : \phi_{n-1}$, where $\Pi_{n-1} = \{x : \mathrm{C} : \phi \in \Pi, y_n : \mathrm{C}_n : \phi_n \mid x \in \textsc{vars}(\mathrm{e}_{n-1})\}$ and also that $\Gamma_{n-1} \models^{\mathrm{T}}_{\chi} \mathrm{e}_{n-1} : \mathrm{C}_{n-1}$, where $\Gamma_{n-1} = \{x : \mathrm{C} \in \widehat{\Pi} \mid x \in \textsc{vars}(\mathrm{e}_{n-1})\}$. It follows by definition, then, that $\mathbb{VARS}^{\mathrm{T}}_{\mathrm{ENV}}(\Gamma_{n-1}) = \mathbb{VARS}^{\mathrm{P}}_{\mathrm{ENV}}(\Pi_{n-1})$ and so also that $\Gamma_{n-1} = \widehat{\Pi_{n-1}}$. Since $\Gamma_{n-1} \subseteq \widehat{\Pi}$ we have that $\widehat{\Pi_{n-1}} \subseteq \widehat{\Pi}$, and then by Definition 4.2.7 that $\Pi_{n-1} \subseteq \Pi$. So, by Lemma 5.1.2 that $\Pi \models^{\mathrm{P}}_{\chi} \mathrm{e}_{n-1} : \mathrm{C}_{n-1} : \phi_{n-1}$.

We repeat this process of expanding each variable, y_{n-2}, …, y_1. We thus obtain a sequence of predicates, $\overline{\phi}_n$, such that $\Pi \models^{\mathrm{P}}_{\chi} \mathrm{e}_n : \mathrm{C}_n : \phi_n$, …, $\Pi \models^{\mathrm{P}}_{\chi} \mathrm{e}_1 : \mathrm{C}_1 : \phi_1$, as well as the final result of $\Pi, \{y_1 : \mathrm{C}_1 : \phi_1,\ldots,y_n : \mathrm{C}_n : \phi_n\} \models^{\mathrm{P}}_{\chi} \mathrm{e}_{\mathrm{s}} : \mathrm{C} : \phi$. The last step is to use the process of thinning and widening on this final result, in an identical manner as for each of the expressions e_i. We have by Lemma 5.1.3 that $\Gamma''_{\mathrm{s}} \models^{\mathrm{T}}_{\chi} \mathrm{e}_{\mathrm{s}} : \mathrm{C}$ where $\Gamma''_{\mathrm{s}} = \{x : \mathrm{C} \in \Gamma'_{\mathrm{s}} \mid x \in \textsc{vars}(\mathrm{e}_{\mathrm{s}})\}$, and by Lemma 5.1.4 that $\Pi'' \models^{\mathrm{P}}_{\chi} \mathrm{e}_{\mathrm{s}} : \mathrm{C} : \phi$ where $\Pi'' = \{x : \mathrm{C} : \phi \in \Pi, \{y_1 : \mathrm{C}_1 : \phi_1,\ldots,y_n : \mathrm{C}_n : \phi_n\} \mid x \in \textsc{vars}(\mathrm{e}_{\mathrm{s}})\}$. Again, we have that $\Gamma''_{\mathrm{s}} = \widehat{\Pi''}$ and since $\Gamma''_{\mathrm{s}} \subseteq \{y_1 : \mathrm{C}_1,\ldots,y_n : \mathrm{C}_n\}$ it follows that $\Pi'' \subseteq \{y_1 : \mathrm{C}_1 : \phi_1,\ldots,y_n : \mathrm{C}_n : \phi_n\}$. Then, by Lemma 5.1.2 we have that $\{y_1 : \mathrm{C}_1 : \phi_1,\ldots,y_n : \mathrm{C}_n : \phi_n\} \models^{\mathrm{P}}_{\chi} \mathrm{e}_{\mathrm{s}} : \mathrm{C} : \phi$. We can now construct a variable substitution returning each y_i to its original x_i, and so by Lemma 5.1.6 we have that $\Pi''[x_1/y_1,\ldots,x_n/y_n] \models^{\mathrm{P}}_{\chi} \mathrm{e}_{\mathrm{s}}[x_1/y_1,\ldots,x_n/y_n] : \mathrm{C} : \phi$, which by Corollary 3.3.2 is equivalent to $\Pi' \models^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \phi$. □

5.2 Subject Reduction

We are now in a position to prove the subject reduction theorems which we stated at the beginning of the chapter.

Theorem 5.2.1 (SUBJECT REDUCTION FOR TYPE ASSIGNMENT)**.**

$$\models^{\mathrm{T}}_{\chi} \diamond \;\&\; \Gamma \models^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C} \;\&\; \mathrm{e} \rightarrow_{\chi} \mathrm{e}' \Rightarrow \Gamma \models^{\mathrm{T}}_{\chi} \mathrm{e}' : \mathrm{C}$$

Proof. By straightforward induction on the derivation of $\mathrm{e} \rightarrow_\chi \mathrm{e}'$. We show the base cases of field access, field assignment and method invocation, and some inductive cases.

(R-FLD) Then $\mathrm{e} \equiv (\mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_n)).f_j$ and $\mathrm{e}' \equiv \mathrm{e}_j$ with $\mathcal{F}(\chi, \mathrm{D}) = \overline{f}_n$ and $j \in \overline{n}$. By assumption, we have that $\Gamma \vdash^{\mathrm{T}}_{\chi} (\mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_n)).f_j : \mathrm{C}$ and so, by Lemma 4.1.6(3), $\exists\, \mathrm{D}', \mathrm{D}'' <:_\chi \mathrm{C}$ such that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_n) : \mathrm{D}'$ with $\Delta_{\mathrm{f}}(\chi, \mathrm{D}', f_j) = \mathrm{D}''$. Also, by Lemma 4.1.6(6), $\mathrm{D} <:_\chi \mathrm{D}'$ with $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_i : \mathrm{C}_i$ such that $\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f_i) = \mathrm{C}_i$ for each $i \in \overline{n}$. In particular, $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_j : \mathrm{C}_j$. By assumption we have that $\vdash^{\mathrm{T}}_{\chi} \diamond$, and so it follows from Definition 4.1.7 that $\vdash \chi$. Then, since $\mathrm{D} <:_\chi \mathrm{D}'$, it follows by Property 3.2.2(1) that $\mathrm{C}_j = \mathrm{D}''$ and therefore that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_j : \mathrm{D}''$. Therefore, since $\mathrm{D}'' <:_\chi \mathrm{C}$, by rule (T-SUB) we have that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_j : \mathrm{C}$.

(R-ASS) Then $\mathrm{e} \equiv (\mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_n)).f_j = \mathrm{e}'_j$ and $\mathrm{e}' \equiv \mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_n)$ with $\mathcal{F}(\mathrm{D}) = \overline{f}_n$ such that $j \in \overline{n}$. Also, $\mathrm{e}'_i = \mathrm{e}_i$ for all $i \in \overline{n}$ such that $i \neq j$. By assumption we have that $\Gamma \vdash^{\mathrm{T}}_{\chi} (\mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_n)).f_j = \mathrm{e}_j : \mathrm{C}$ and so by Lemma 4.1.6(4), $\exists\, \mathrm{D}' <:_\chi \mathrm{C}$ such that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_n) : \mathrm{D}'$ and $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}'_j : \mathrm{C}'$ such that $\Delta_{\mathrm{f}}(\chi, \mathrm{D}', f_j) = \mathrm{C}'$. Also, by Lemma 4.1.6(6), we have that $\mathrm{D} <:_\chi \mathrm{D}'$ and $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_i : \mathrm{C}_i$ such that $\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f_i) = \mathrm{C}_i$ for each $i \in \overline{n}$. Then, since $\mathrm{e}'_i = \mathrm{e}_i$ for all $i \in \overline{n}$ such that $i \neq j$, it follows that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}'_i : \mathrm{C}_i$ for each $i \in \overline{n}$ such that $i \neq j$. By assumption we have that $\vdash^{\mathrm{T}}_{\chi} \diamond$, and so it follows from Definition 4.1.7 that $\vdash \chi$. Then, since $\mathrm{D} <:_\chi \mathrm{D}'$, it follows by Property 3.2.2(1) that $\mathrm{C}' = \mathrm{C}_j$ and therefore that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}'_j : \mathrm{C}_j$. Then, by rule (T-NEW) we have $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}'}_n) : \mathrm{D}$ and since $\mathrm{D} <:_\chi \mathrm{D}' <:_\chi \mathrm{C}$ it follows by rule (T-SUB) that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}'}_n) : \mathrm{C}$.

(R-INVK) Then $\mathrm{e} \equiv (\mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_n)).m(\overline{\mathrm{e}'}_{n'})$ and $\mathrm{e}' \equiv \mathrm{e}_0[\mathrm{e}'_1/x_1, \ldots, \mathrm{e}'_{n'}/x_{n'}, \mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_n)/\mathtt{this}]$ with $\textsc{mbody}(\mathrm{D}, m) = (\overline{x}_{n'}, \mathrm{e}_0)$. By assumption we have that $\Gamma \vdash^{\mathrm{T}}_{\chi} (\mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_n)).m(\overline{\mathrm{e}'}_{n'}) : \mathrm{C}$ and so by Lemma 4.1.6(5) $\exists\, \mathrm{D}'$ such that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_n) : \mathrm{D}'$ and $\Delta_{\mathrm{m}}(\chi, \mathrm{D}', m) = \overline{\mathrm{C}}_{n'} \rightarrow \mathrm{C}_0$ with $\mathrm{C}_0 <:_\chi \mathrm{C}$. Furthermore, $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}'_i : \mathrm{C}_i$ for each $i \in \overline{n'}$. By Lemma 4.1.6(6) it follows that $\mathrm{D} <:_\chi \mathrm{D}'$ and $\mathcal{F}(\mathrm{D}) = \overline{f}_n$ with $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_i : \mathrm{D}_i$ where $\Delta^{\mathrm{f}}_{\mathrm{P}}(\mathrm{D}, f_i) = \mathrm{D}_i$ for each $i \in \overline{n}$, so by rule (T-NEW) we have that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_n) : \mathrm{D}$. By assumption we have that $\vdash^{\mathrm{T}}_{\chi} \diamond$, and so it follows from Definition 4.1.7 that $\vdash \chi$. So, by Property 3.2.2(2), we have that $\textsc{mbody}(\chi, \mathrm{D}, m) = \overline{\mathrm{C}}_n \rightarrow \mathrm{C}_0$, since $\mathrm{D} <:_\chi \mathrm{D}'$. It also follows from the definition of type consistent execution contexts (Definition 4.1.7) that $\{x_1 : \mathrm{C}_1, \ldots, x_{n'} : \mathrm{C}_{n'}, \mathtt{this} : \mathrm{D}\} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0 : \mathrm{C}_0$, and so by Corollary 5.1.9 we have that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0[\mathrm{e}'_1/x_1, \ldots, \mathrm{e}'_{n'}/x_{n'}, \mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_n)/\mathtt{this}] : \mathrm{C}_0$. Then since $\mathrm{C}_0 <:_\chi \mathrm{C}$ it follows that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0[\mathrm{e}'_1/x_1, \ldots, \mathrm{e}'_{n'}/x_{n'}, \mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_n)/\mathtt{this}] : \mathrm{C}$.

(RC-FLD) Then $\mathrm{e} \equiv \mathrm{e}_0.f$ and $\mathrm{e}' \equiv \mathrm{e}'_0.f$ with $\mathrm{e}_0 \rightarrow_\chi \mathrm{e}'_0$. By assumption $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0.f : \mathrm{C}$. So, by Lemma 4.1.6(3), $\exists\, \mathrm{D}, \mathrm{C}' <:_\chi \mathrm{C}$ such that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0 : \mathrm{D}$ and $\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f) = \mathrm{C}'$. By the inductive hypothesis we have that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}'_0 : \mathrm{D}$ and so by rule (T-FLD) it follows that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}'_0.f : \mathrm{C}'$. Then, by rule (T-SUB), we have that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}'_0.f : \mathrm{C}$.

(RC-ASS$_1$) Then $\mathrm{e} \equiv \mathrm{e}_0.f = \mathrm{e}_1$ and $\mathrm{e}' \equiv \mathrm{e}'_0.f = \mathrm{e}_1$ with $\mathrm{e}_0 \rightarrow_\chi \mathrm{e}'_0$. By assumption, $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0.f = \mathrm{e}_1 : \mathrm{C}$. So, by Lemma 4.1.6(4), $\exists\, \mathrm{D}, \mathrm{C}' <:_\chi \mathrm{C}$ such that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0 : \mathrm{C}'$ and $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_1 : \mathrm{D}$ with $\Delta_{\mathrm{f}}(\chi, \mathrm{C}', f) = \mathrm{D}$. By the inductive hypothesis we have that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}'_0 : \mathrm{C}'$ and so, by rule (T-ASS) it follows that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}'_0.f = \mathrm{e}_1 : \mathrm{C}'$. Then, by rule (T-SUB) we have that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}'_0.f = \mathrm{e}_1 : \mathrm{C}$.

(RC-NEW) Then $\mathrm{e} \equiv \mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_n)$ and $\mathrm{e}' \equiv \mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}'}_n)$ with $\mathrm{e}_j \rightarrow_\chi \mathrm{e}'_j$ for some $j \in \overline{n}$, and $\mathrm{e}'_i = \mathrm{e}_i$ for each $i \in \overline{n}$ such that $i \neq j$. By assumption $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_n) : \mathrm{C}$. So, by Lemma 4.1.6(6) $\mathrm{D} <:_\chi \mathrm{C}$ and $\mathcal{F}(\chi, \mathrm{D}) = \overline{f}_n$ with $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_i : \mathrm{C}_i$ such that $\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f_i) = \mathrm{C}_i$ for each $i \in \overline{n}$. By the inductive hypothesis we have that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}'_j : \mathrm{C}_j$ and we also have that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathrm{e}'_i : \mathrm{C}_i$ for each $i \in \overline{n}$ such that $i \neq j$ since $\mathrm{e}'_i = \mathrm{e}_i$ for each $i \in \overline{n}$ such that $i \neq j$. So, by rule (T-NEW) it follows that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}'}_n) : \mathrm{D}$. Then by rule (T-SUB) we have that $\Gamma \vdash^{\mathrm{T}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}'}_n) : \mathrm{C}$.

□

Theorem 5.2.2 (SUBJECT REDUCTION FOR PREDICATE ASSIGNMENT)**.**

$$\vdash^{\mathrm{T}}_{\chi} \diamond \ \&\ \Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \phi \ \&\ \mathrm{e} \rightarrow_\chi \mathrm{e}' \Rightarrow \Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}' : \mathrm{C} : \phi$$

Proof. We consider the cases where $\phi \equiv \omega$ and $\phi \equiv \langle l_i : \tau_i \ ^{i \in \overline{n}} \rangle$ separately:

($\phi \equiv \omega$) By assumption, $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \omega$ and so by Theorem 4.3.1 we have that $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C}$. By Theorem 5.2.1 it then follows that $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}' : \mathrm{C}$ and by rule (P-OMEGA) we have that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}' : \mathrm{C} : \omega$.

($\phi = \sigma \equiv \langle l_i : \tau_i \ ^{i \in \overline{n}} \rangle$) This case proceeds by induction on the derivation of $\mathsf{e} \rightarrow_\chi \mathsf{e}'$. Again, we show the base cases of field access, field assignment and method invocation, and some inductive cases.

(R-FLD) Then $\mathsf{e} \equiv (\mathtt{new}\ \mathrm{D}(\overline{\mathsf{e}}_{n_1})).f_j$ and $\mathsf{e}' \equiv \mathsf{e}_j$ with $\mathcal{F}(\mathrm{D}) = \overline{f}_{n_1}$ and $j \in \overline{n_1}$. By assumption, $\Pi \vdash^{\mathrm{P}}_{\chi} (\mathtt{new}\ \mathrm{D}(\overline{\mathsf{e}}_{n_1})).f_j : \mathrm{C} : \phi$. So, by Lemma 4.2.10(3), $\exists\ \mathrm{D}', \mathrm{C}' <:_\chi \mathrm{C}, \psi \trianglelefteq \phi$ such that $\Delta_{\mathrm{f}}(\chi, \mathrm{D}', f_j) = \mathrm{C}'$ and $\Pi \vdash^{\mathrm{P}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathsf{e}}_{n_1}) : \mathrm{D}' : \langle f_j : \psi \rangle$. Then, by Lemma 4.2.10(7), $\mathrm{D} <:_\chi \mathrm{D}'$ and $\Pi \vdash^{\mathrm{P}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathsf{e}}_{n_1}) : \mathrm{D} : \langle f_j : \psi \rangle$, and by rule (P-NEWFLD) it follows that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathsf{e}_j : \mathrm{C}'' : \psi$ with $\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f_j) = \mathrm{C}''$ for some C''. By assumption we have that $\vdash^{\mathrm{T}}_{\chi} \diamond$, and so it follows from Definition 4.1.7 that $\vdash \chi$. Then, since $\mathrm{D} <:_\chi \mathrm{D}'$, it follows from Property 3.2.2(1) that $\mathrm{C}' = \mathrm{C}''$ and so we have that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathsf{e}_j : \mathrm{C}' : \psi$. Then, since $\mathrm{C}' <:_\chi \mathrm{C}$, we have by rule (P-SUBTYPE) that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathsf{e}_j : \mathrm{C} : \psi$. Furthermore, since $\psi \trianglelefteq \phi$ it follows by Theorem 4.3.2 that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathsf{e}_j : \mathrm{C} : \phi$.

(R-ASS) Then $\mathsf{e} \equiv (\mathtt{new}\ \mathrm{D}(\overline{\mathsf{e}}_{n_1})).f_j = \mathsf{e}'_j$ and $\mathsf{e}' \equiv \mathtt{new}\ \mathrm{D}(\overline{\mathsf{e}'}_{n_1})$ with $\mathcal{F}(\chi, \mathrm{D}) = \overline{f}_{n_1}$ such that $j \in \overline{n_1}$. Also $\mathsf{e}'_k = \mathsf{e}_k$ for each $k \in \overline{n_1}$ such that $k \neq j$.

If $\sigma = \langle \epsilon \rangle$, then by rule (P-ASS$_2$) we have that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathsf{e}}_{n_1}) : \mathrm{C} : \langle \epsilon \rangle$. So, by Lemma 4.2.10(6) we have that $\mathrm{D}\ <:_\chi \mathrm{C}$ and $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathsf{e}}_{n_1}) : \mathrm{D}$. So, by rule (T-NEW) it follows that $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathsf{e}_k : \mathrm{C}_k$ such that $\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f) = \mathrm{C}_k$ for each $k \in \overline{n_1}$. By Theorem 5.2.1 it follows that $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathsf{e}'_j : \mathrm{C}_j$, since $\mathsf{e}_j \rightarrow_\chi \mathsf{e}'_j$. Also, we have that $\mathsf{e}'_k = \mathsf{e}_k$ for each $k \in \overline{n_1}$ such that $k \neq j$, and so it follows that $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathsf{e}'_k : \mathrm{C}_k$ for each $k \in \overline{n_1}$ such that $k \neq j$. Now, by rule (T-NEW), it follows that $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathsf{e}'}_{n_1}) : \mathrm{D}$ and so by rule (P-NEWOBJ) we have that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathsf{e}'}_{n_1}) : \mathrm{D} : \langle \epsilon \rangle$. Then, since $\mathrm{D}\ <:_\chi \mathrm{C}$ it follows by rule (P-SUBTYPE) that$\Pi \vdash^{\mathrm{P}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathsf{e}'}_{n_1}) : \mathrm{C} : \langle \epsilon \rangle$.

If $\sigma = \langle l_i : \tau_i \ ^{i \in \overline{n}} \rangle \neq \langle \epsilon \rangle$, then by Lemma 4.2.10(4), $\exists\ \overline{\mathrm{D}}_n$ such that $\mathrm{D}_i <:_\chi \mathrm{C}$ and $\Pi \vdash^{\mathrm{P}}_{\chi} (\mathtt{new}\ \mathrm{D}(\overline{\mathsf{e}}_{n_1})).f_j = \mathsf{e}'_j : \mathrm{D}_i : \langle l_i : \tau_i \rangle$ for each $i \in \overline{n}$. Now, take any $i \in \overline{n}$; there are three possibilities:

($l_i = f \in \mathbb{FIELD\text{-}ID}, f = f_j$) Then $\tau_i \equiv \phi'$, and by rule (P-ASS$_1$), $\Pi \vdash^{\mathrm{P}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathsf{e}}_{n_1}) : \mathrm{D}_i : \sigma'$ for some σ' and $\Pi \vdash^{\mathrm{P}}_{\chi} \mathsf{e}'_j : \mathrm{D}' : \phi'$ with $\Delta_{\mathrm{f}}(\chi, \mathrm{D}_i, f_j) = \mathrm{D}'$. By Theorem 4.3.3 it follows that $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathsf{e}}_{n_1}) : \mathrm{D}_i$ and so by Lemma 4.1.6(6) we have that $\mathrm{D} <:_\chi \mathrm{D}_i$ and $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathsf{e}_k : \mathrm{C}_k$ with $\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f_k) = \mathrm{C}_k$ for each $k \in \overline{n_1}$. Since $\mathsf{e}'_k = \mathsf{e}_k$ for each $k \in \overline{n_1}$ such that $k \neq j$, it follows that $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathsf{e}'_k : \mathrm{C}_k$ for each $k \in \overline{n_1}$ such that $k \neq j$. By assumption we have that $\vdash^{\mathrm{T}}_{\chi} \diamond$, and so it follows from Definition 4.1.7 that $\vdash \chi$. Then, since $\mathrm{D} <:_\chi \mathrm{D}_i$ it follows from Property 3.2.2(1) that $\mathrm{D}' = \mathrm{C}_j$ and therefore that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathsf{e}'_j : \mathrm{C}_j : \phi'$. So, by rule (P-NEWFLD) we have that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathsf{e}'}_{n_1}) : \mathrm{D} : \langle f_j : \phi' \rangle$. Then, since $\mathrm{D} <:_\chi \mathrm{D}_i$ it follows by rule (P-SUBTYPE) that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathsf{e}'}_{n_1}) : \mathrm{D}_i : \langle f_j : \phi' \rangle$.

($l_i = f \in \mathbb{FIELD\text{-}ID}, f \neq f_j$) Then $\tau_i = \phi'$ and $l_i = f_{j'}$ for some $j' \in \overline{n_1}$ such that $j' \neq j$. By rule (P-ASS$_2$) we have that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathsf{e}}_{n_1}) : \mathrm{D}_i : \langle f_{j'} : \phi' \rangle$ and $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathsf{e}'_j : \mathrm{D}'$ with $\Delta_{\mathrm{f}}(\chi, \mathrm{D}_i, f_j) = \mathrm{D}'$. By Lemma 4.2.10(7) $\mathrm{D} <:_\chi \mathrm{D}_i$ and $\Pi \vdash^{\mathrm{P}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathsf{e}}_{n_1}) : \mathrm{D} : \langle f_{j'} : \phi' \rangle$. By rule (P-NEWFLD) we then have that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathsf{e}_{j'} : \mathrm{C}_{j'} : \phi'$ with $\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f_{j'}) = \mathrm{C}_{j'}$, and $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathsf{e}_k : \mathrm{C}_k$ with $\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f_k) = \mathrm{C}_k$ for all $k \in \overline{n_1}$ such that $k \neq j'$. Since $\mathsf{e}'_k = \mathsf{e}_k$ for each $k \in \overline{n_1}$ such that $k \neq j$, it follows that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathsf{e}'_{j'} : \mathrm{C}_{j'} : \phi'$ and also that $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathsf{e}'_k : \mathrm{C}_k$ for each $k \in \overline{n_1}$ such that $k \neq j'$ and $k \neq j$. By assumption we have that $\vdash^{\mathrm{T}}_{\chi} \diamond$, and so it follows from Definition 4.1.7 that $\vdash \chi$. Then, since $\mathrm{D} <:_\chi \mathrm{D}_i$, it follows from property 3.2.2(1) that $\mathrm{D}' = \mathrm{C}_j$ and so we have that $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathsf{e}'_j : \mathrm{C}_j$. Then, by rule (P-NEWFLD) we have that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathsf{e}'}_{n_1}) : \mathrm{D} : \langle f_{j'} : \phi' \rangle$. Furthermore, since $\mathrm{D} <:_\chi \mathrm{D}_i$ it follows by rule (P-SUBTYPE) that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathsf{e}'}_{n_1}) : \mathrm{D}_i : \langle f_{j'} : \phi' \rangle$.

($l_i \in \mathbb{METHOD\text{-}NAME}$) Then $l_i = m$ and $\tau_i \equiv \psi{::}\overline{\phi}_{n_2} \rightarrow \phi'$. By rule (P-ASS$_2$), $\Pi \vdash^{\mathrm{P}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathsf{e}}_{n_1}) : \mathrm{D}_i : \langle m : \tau_i \psi{::}\overline{\phi}_{n_2} \rightarrow \phi' \rangle$ and $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathsf{e}'_j : \mathrm{D}'$ with $\Delta_{\mathrm{f}}(\chi, \mathrm{D}_i, f_j) = \mathrm{D}'$. By Lemma 4.2.10(7) $\mathrm{D} <:_\chi \mathrm{D}_i$ and $\Pi \vdash^{\mathrm{P}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathsf{e}}_{n_1}) : \mathrm{D} : \langle m : \psi{::}\overline{\phi}_{n_2} \rightarrow \phi' \rangle$. By rule (P-NEWMETH) we have that $\Pi' \vdash^{\mathrm{P}}_{\chi} \mathsf{e}_0 : \mathrm{D}_0 : \phi'$ with $\Delta_{\mathrm{m}}(\chi, \mathrm{D}, m) = \overline{\mathrm{C}}_{n_2} \rightarrow \mathrm{D}_0$ and $\Pi' = \{\ x_1 : \mathrm{C}_1 : \phi_1, \ldots, x_{n_2} : \mathrm{C}_{n_2} : \phi_{n_2}, \mathtt{this} : \mathrm{D} : \psi\ \}$ such that $\textsc{mbody}(\chi, \mathrm{D}, m) = (\overline{x}_{n_2}, \mathsf{e}_0)$. We also have that $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathtt{new}\ \mathrm{D}(\overline{\mathsf{e}}_{n_1}) : \mathrm{D}$. Then, by rule (T-NEW) it follows that $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathsf{e}_k : \mathrm{C}'_k$ with $\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f_k) = \mathrm{C}'_k$ for each $k \in \overline{n_1}$. Since $\mathsf{e}'_k = \mathsf{e}_k$ for each $k \in \overline{n_1}$ such that $k \neq j$, it follows that $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathsf{e}'_k : \mathrm{C}_k$ for each $k \in \overline{n_1}$ such that $k \neq j$. By assumption we have that $\vdash^{\mathrm{T}}_{\chi} \diamond$, and so it follows from Definition 4.1.7 that $\vdash \chi$. Then, since $\mathrm{D} <:_\chi \mathrm{D}_i$ it follows from Property 3.2.2(1) that $\mathrm{D}' = \mathrm{C}'_j$ and so $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}'_j : \mathrm{C}'_j$. So, by rule (T-NEW) we have that $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}'}_{n_1}) : \mathrm{D}$ and by rule (P-NEWMETH) it follows that $\Pi \Vdash^{\mathrm{P}}_{\chi} \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}'}_{n_1}) : \mathrm{D} : \langle m : \psi{::}\overline{\phi}_{n_2} \to \phi' \rangle$. Furthermore, since $\mathrm{D} <:_{\chi} \mathrm{D}_i$ it follows by rule (P-SUBTYPE) that $\Pi \Vdash^{\mathrm{P}}_{\chi} \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}'}_{n_1}) : \mathrm{D}_i : \langle m : \psi{::}\overline{\phi}_{n_2} \to \phi' \rangle$.

Since our choice of i was arbitrary, we have that $\Pi \Vdash^{\mathrm{P}}_{\chi} \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}'}_{n_1}) : \mathrm{D}_i : \langle l_i : \tau_i \rangle$ for all $i \in \overline{n}$. Then, since each $\mathrm{D}_i <:_{\chi} \mathrm{C}$ we have by rule (P-SUBTYPE) that $\Pi \Vdash^{\mathrm{P}}_{\chi} \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}'}_{n_1}) : \mathrm{C} : \langle l_i : \tau_i \rangle$ for each $i \in \overline{n}$ and by rule (P-SEQ) it then follows that $\Pi \Vdash^{\mathrm{P}}_{\chi} \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}'}_{n_1}) : \mathrm{C} : \langle l_i : \tau_i \ ^{i \in \overline{n}} \rangle$.

(R-INVK) Then $\mathrm{e} \equiv (\texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1})).m(\overline{\mathrm{e}'}_{n_2})$ and also $\mathrm{e}' \equiv \mathrm{e}_0[\mathrm{e}'_1/x_1, \ldots, \mathrm{e}'_{n_2}/x_{n_2}, \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1})/\texttt{this}]$ with $\mathrm{MBODY}(\mathrm{D}, m) = (\overline{x}_{n_2}, \mathrm{e}_0)$. By assumption we have $\Pi \Vdash^{\mathrm{P}}_{\chi} (\texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1})).m(\overline{\mathrm{e}'}_{n_2}) : \mathrm{C} : \phi$ and so by Lemma 4.2.10(5) $\exists$ D', $\overline{\mathrm{C}}_{n_2}$, $\mathrm{C}'' <:_{\chi} \mathrm{C}$, ψ, $\overline{\phi}_{n_2}$ and $\phi' \trianglelefteq \phi$ such that $\Pi \Vdash^{\mathrm{P}}_{\chi} \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1}) : \mathrm{D}' : \langle m : \psi{::}\overline{\phi}_{n_2} \to \phi' \rangle$ and $\Pi \Vdash^{\mathrm{P}}_{\chi} \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1}) : \mathrm{D}' : \psi$ with $\Delta_{\mathrm{m}}(\chi, \mathrm{D}', m) = \overline{\mathrm{C}}_{n_2} \to \mathrm{C}''$. Furthermore, we have that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_k : \mathrm{C}_k : \phi_k$ for each $k \in \overline{n_2}$. By Theorem 4.3.1 it follows that $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1}) : \mathrm{D}'$. Then, by Lemma 4.1.6(6) we have that $\mathrm{D} <:_{\chi} \mathrm{D}'$ and $\mathcal{F}(\chi, \mathrm{D}) = \overline{f}_{n_1}$ with $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_k : \mathrm{D}_k$ such that $\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f_k) = \mathrm{D}_k$ for each $k \in \overline{n_1}$. Therefore, by rule (T-NEW) it follows that $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1}) : \mathrm{D}$ and so by Theorem 4.3.3 we have that $\Pi \Vdash^{\mathrm{P}}_{\chi} \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1}) : \mathrm{D} : \psi$ and $\Pi \Vdash^{\mathrm{P}}_{\chi} \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1}) : \mathrm{D} : \langle m : \psi{::}\overline{\phi}_{n_2} \to \phi' \rangle$. Then, by rule (P-NEWMETH) it follows that $\Pi' \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0 : \mathrm{C}' : \phi'$ with $\Pi' = \{x_1 : \mathrm{C}'_1 : \phi_1, \ldots, x_{n_2} : \mathrm{C}'_{n_2} : \phi_{n_2}, \texttt{this} : \mathrm{D} : \psi\}$ such that $\Delta_{\mathrm{m}}(\chi, \mathrm{D}, m) = \overline{\mathrm{C}'}_{n_2} \to \mathrm{C}'$. Since, by assumption we have $\Vdash^{\mathrm{T}}_{\chi} \diamond$, it follows from Definition 4.1.7 that $\vdash \chi$ and so from Property 3.2.2(2) that $\mathrm{C}'' = \mathrm{C}'$ and $\mathrm{C}_k = \mathrm{C}'_k$ for each $k \in \overline{n_2}$, since $\mathrm{D} <:_{\chi} \mathrm{D}'$. Thus, $\Pi' = \{x_1 : \mathrm{C}_1 : \phi_1, \ldots, x_{n_2} : \mathrm{C}_{n_2} : \phi_{n_2}, \texttt{this} : \mathrm{D} : \psi\}$, and by Corollary 5.1.9(2) it follows that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0[\mathrm{e}_1/x_1, \ldots, \mathrm{e}_{n_2}/x_{n_2}, \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1})/\texttt{this}] : \mathrm{C}' : \phi'$. Then, since $\mathrm{C}' = \mathrm{C}'' <:_{\chi} \mathrm{C}$ we have by rule (P-SUBTYPE) that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0[\mathrm{e}_1/x_1, \ldots, \mathrm{e}_{n_2}/x_{n_2}, \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1})/\texttt{this}] : \mathrm{C} : \phi'$. Furthermore, since $\phi' \trianglelefteq \phi$ it follows from Theorem 4.3.2 that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0[\mathrm{e}_1/x_1, \ldots, \mathrm{e}_{n_2}/x_{n_2}, \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1})/\texttt{this}] : \mathrm{C} : \phi$.

(RC-ASS$_2$) Then $\mathrm{e} \equiv \mathrm{e}_0.f = \mathrm{e}_1$ and $\mathrm{e}' \equiv \mathrm{e}_0.f = \mathrm{e}'_1$ with $\mathrm{e}_1 \to_{\chi} \mathrm{e}'_1$. By assumption, $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0.f = \mathrm{e}_1 : \mathrm{C} : \sigma$. By Lemma 4.2.10(4), $\exists\ \overline{\mathrm{C}}_n$ such that $\mathrm{C}_i <:_{\chi} \mathrm{C}$ and $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0.f = \mathrm{e}_1 : \mathrm{C}_i : \langle l_i : \tau_i \rangle$ for each $i \in \overline{n}$. Now, take any $i \in \overline{n}$; there are two possibilities:

($f = l_i$) Then $\tau = \phi'$. By rule (P-ASS$_1$), $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0 : \mathrm{C}_i$ and $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_1 : \mathrm{D} : \phi'$ with $\Delta_{\mathrm{f}}(\chi, \mathrm{C}_i, f) = \mathrm{D}$. By the inductive hypothesis we have that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}'_1 : \mathrm{D} : \phi'$ and so by rule (P-ASS$_1$) it follows that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0.f = \mathrm{e}'_1 : \mathrm{C}_i : \langle f : \phi' \rangle$. Then, since $\mathrm{C}_i <:_{\chi} \mathrm{C}$, by rule (P-SUBTYPE) we have that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0.f = \mathrm{e}'_1 : \mathrm{C} : \langle f : \phi' \rangle$.

($f \neq l_i$) By rule (P-ASS$_2$), $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0 : \mathrm{C}_i : \langle l_i : \tau_i \rangle$ and $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_1 : \mathrm{D}$ with $\Delta_{\mathrm{f}}(\chi, \mathrm{C}_i, f) = \mathrm{D}$. By Theorem 5.2.1 we have that $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}'_1 : \mathrm{D}$ and so by rule (P-ASS$_2$) it follows that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0.f = \mathrm{e}'_1 : \mathrm{C}_i : \langle l_i : \tau_i \rangle$. Then, since $\mathrm{C}_i <:_{\chi} \mathrm{C}$, we have by rule (P-SUBTYPE) that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0.f = \mathrm{e}'_1 : \mathrm{C} : \langle l_i : \tau_i \rangle$.

Since our choice of i was arbitrary, we have that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0.f = \mathrm{e}'_1 : \mathrm{C} : \langle l_i : \tau_i \rangle$ for all $i \in \overline{n}$, and by rule (P-SEQ) that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0.f = \mathrm{e}'_1 : \mathrm{C} : \langle l_i : \tau_i \ ^{i \in \overline{n}} \rangle$.

(RC-INVK$_2$) Then $\mathrm{e} \equiv \mathrm{e}_0.m(\overline{\mathrm{e}}_{n_1})$ and $\mathrm{e}' \equiv \mathrm{e}_0.m(\overline{\mathrm{e}'}_{n_1})$ with $\mathrm{e}_j \to_{\chi} \mathrm{e}'_j$ for some $j \in \overline{n_1}$, and $\mathrm{e}'_i = \mathrm{e}_i$ for each $i \in \overline{n_1}$ such that $i \neq j$. By assumption $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0.m(\overline{\mathrm{e}}_{n_1}) : \mathrm{C} : \phi$. So, by Lemma 4.2.10(5), $\exists$ D, $\overline{\mathrm{C}}_{n_1}$, $\mathrm{C}' <:_{\chi} \mathrm{C}$, ψ, $\overline{\phi}_{n_1}$, $\phi' \trianglelefteq \phi$ such that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}'_0 : \mathrm{D} : \langle m : \psi{::}\overline{\phi}_{n_1} \to \phi' \rangle$ and $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}'_0 : \mathrm{D} : \psi$ with $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_i : \mathrm{C}_k : \phi_k$ for each $k \in \overline{n_1}$ such that $\Delta_{\mathrm{m}}(\chi, \mathrm{D}, m) = \overline{\mathrm{C}}_{n_1} \to \mathrm{C}'$. By the inductive hypothesis we then have that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}'_j : \mathrm{C}_j : \phi_j$, and we also have that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}'_i : \mathrm{C}_i : \phi_i$ for each $i \in \overline{n_1}$ such that $i \neq j$ since, in each case, $\mathrm{e}'_i = \mathrm{e}_i$. Thus, by rule (P-INVK), it follows that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0.m(\overline{\mathrm{e}'}_{n_1}) : \mathrm{C}' : \phi'$. Then, since $\mathrm{C}' <:_{\chi} \mathrm{C}$ we have by rule (T-SUBTYPE) that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0.m(\overline{\mathrm{e}'}_{n_1}) : \mathrm{C} : \psi$. Furthermore, since $\phi' \trianglelefteq \phi$, by Theorem 4.3.2 it follows that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0.m(\overline{\mathrm{e}'}_{n_1}) : \mathrm{C} : \phi$.

□

5.3 Subject Expansion

As is the case for the ς-calculus [1], subject expansion does not hold in general for type assignment. We can, however, show that subject expansion for predicate assignment holds under the assumption that both the redex and the reduct have the same type. This extra assumption is necessary, since predicates are assigned to *typeable* terms (see Theorem 4.3.1). This is also a requirement of the subject expansion result for the predicate system in [5]

Theorem 5.3.1 (SUBJECT EXPANSION FOR PREDICATE ASSIGNMENT)**.**

$$\Vdash^{\mathrm{T}}_{\chi} \diamond \;\&\; \Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}' : \mathrm{C} : \phi \;\&\; \mathrm{e} \rightarrow_{\chi} \mathrm{e}' \;\&\; \widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C} \Rightarrow \Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \phi$$

Proof. We consider the cases where $\phi \equiv \omega$ and $\phi \equiv \langle l_i : \tau_i \ ^{i \in \overline{n}}\rangle$ separately:

($\phi \equiv \omega$) This case is trivial, since by assumption we have that $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathrm{C}$ and so, by rule (P-OMEGA), it follows that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \omega$.

($\phi \equiv \langle l_i : \tau_i \ ^{i \in \overline{n}}\rangle$) This case proceeds by induction on the derivation of $\mathrm{e} \rightarrow_{\chi} \mathrm{e}'$. As for the proofs of subject reduction, we show the base cases of (R-FLD), (R-ASS) and (R-INVK), and some inductive cases.

(R-FLD) Then $\mathrm{e} \equiv (\texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1})).f_j$ and $\mathrm{e}' \equiv \mathrm{e}_j$ with $\mathcal{F}(\chi, \mathrm{D}) = \overline{f}_{n_1}$ and $j \in \overline{n_1}$. By assumption, $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} (\texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1})).f_j : \mathrm{C}$. So, by Lemma 4.1.6(3), $\exists\, \mathrm{D}', \mathrm{C}' <:_{\chi} \mathrm{C}$ such that $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1}) : \mathrm{D}'$ and $\Delta_{\mathrm{f}}(\chi, \mathrm{D}', f) = \mathrm{C}'$. Then, by Lemma 4.1.6(6) it follows that $\mathrm{D} <:_{\chi} \mathrm{D}'$ and $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_k : \mathrm{C}_k$ with $\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f_k) = \mathrm{C}_k$ for each $k \in \overline{n_1}$. Also by assumption we have that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_j : \mathrm{C} : \phi$, so by Theorem 4.3.3 it follows that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_j : \mathrm{C}_j : \phi$. Then by rule (P-NEWFLD) we have that $\Pi \Vdash^{\mathrm{P}}_{\chi} \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1}) : \mathrm{D} : \langle f_j : \phi\rangle$, and by rule (P-FLD) that $\Pi \Vdash^{\mathrm{P}}_{\chi} (\texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1})).f_j : \mathrm{C}_j : \phi$. By assumption we have that $\Vdash^{\mathrm{T}}_{\chi} \diamond$, and so it follows from Definition 4.1.7 that $\vdash \chi$. Then, since $\Delta_{\mathrm{f}}(\chi, \mathrm{D}', f_j) = \mathrm{C}'$ and $\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f_j) = \mathrm{C}_j$ with $\mathrm{D} <:_{\chi} \mathrm{D}'$, it follows from Property 3.2.2(1) that $\mathrm{C}_j = \mathrm{C}'$, and so $\Pi \Vdash^{\mathrm{P}}_{\chi} (\texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1})).f_j : \mathrm{C}' : \phi$. Then, since $\mathrm{C}' <:_{\chi} \mathrm{C}$, by rule (P-SUBTYPE) we have that $\Pi \Vdash^{\mathrm{P}}_{\chi} (\texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1})).f_j : \mathrm{C} : \phi$.

(R-ASS) Then $\mathrm{e} \equiv (\texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1})).f_j = \mathrm{e}'_j$ and $\mathrm{e}' \equiv \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}'}_{n_1})$ with $\mathcal{F}(\chi, \mathrm{D}) = \overline{f}_{n_1}$ such that $j \in \overline{n_1}$. Also $\mathrm{e}'_k = \mathrm{e}_k$ for each $k \in \overline{n_1}$ such that $k \neq j$.

If $\sigma = \langle\epsilon\rangle$, then we need to show that $\Pi \Vdash^{\mathrm{P}}_{\chi} (\texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1})).f_j = \mathrm{e}'_j : \mathrm{C} : \langle\epsilon\rangle$. By assumption we have that $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} (\texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1})).f_j = \mathrm{e}'_j : \mathrm{C}$, and so by Lemma 4.1.6(4) we have that $\exists\, \mathrm{D}' <:_{\chi} \mathrm{C}$ such that $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1}) : \mathrm{D}'$ and $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}'_j : \mathrm{C}'_j$ with $\Delta_{\mathrm{f}}(\chi, \mathrm{D}', f_j) = \mathrm{C}'_j$. By Lemma 4.1.6(6) is also follows that $\mathrm{D} <: \mathrm{D}'$ and $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_k : \mathrm{C}_k$ with $\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f_k) = \mathrm{C}_k$ for each $k \in \overline{n_1}$. So, by rule (T-NEW) it follows that $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1}) : \mathrm{D}$ and so by rule (P-NEWOBJ) that $\Pi \Vdash^{\mathrm{P}}_{\chi} \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1}) : \mathrm{D} : \langle\epsilon\rangle$. By assumption we have that $\Vdash^{\mathrm{T}}_{\chi} \diamond$, and so it follows from Definition 4.1.7 that $\vdash \chi$. Then, since $\mathrm{D} <:_{\chi} \mathrm{D}'$, it follows from Property 3.2.2(1) that $\mathrm{C}'_j = \mathrm{C}_j$, and so $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}'_j : \mathrm{C}_j$. So, we now have by rule (P-ASS$_2$) that $\Pi \Vdash^{\mathrm{P}}_{\chi} (\texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1})).f_j = \mathrm{e}'_j : \mathrm{D} : \langle\epsilon\rangle$. Lastly, since $\mathrm{D} <:_{\chi} \mathrm{D}' <:_{\chi} \mathrm{C}$, it follows by rule (P-SUBTYPE) that $\Pi \Vdash^{\mathrm{P}}_{\chi} (\texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1})).f_j = \mathrm{e}'_j : \mathrm{C} : \langle\epsilon\rangle$.

If $\sigma = \langle l_i : \tau_i \ ^{i \in \overline{n}}\rangle \neq \langle\epsilon\rangle$, then by Lemma 4.2.10(7) it follows that $\mathrm{D} <:_{\chi} \mathrm{C}$ and $\Pi \Vdash^{\mathrm{P}}_{\chi} \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}'}_{n_1}) : \mathrm{D} : \langle l_i : \tau_i\rangle$ for each $i \in \overline{n_1}$. Now, take any $i \in \overline{n_1}$; there are three possibilities:

($l_i = f \in \mathbb{FIELD}$-$\mathbb{ID}$, $f = f_j$) Then $\tau_i = \phi'$, and by rule (P-NEWFLD), $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}'_j : \mathrm{C}_j : \phi'$ and $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}'_k : \mathrm{C}_k$ with $\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f_k) = \mathrm{C}_k$ for each $k \in \overline{n_1}$ such that $k \neq j'$. By assumption, $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} (\texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1})).f_j = \mathrm{e}'_j : \mathrm{C}$. So by Lemma 4.1.6(4), $\exists\, \mathrm{D}' <:_{\chi} \mathrm{C}$ such that $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1}) : \mathrm{D}'$ and $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}'_j : \mathrm{C}'$ with $\Delta_{\mathrm{f}}(\chi, \mathrm{D}', f_j) = \mathrm{C}'$. By assumption we have that $\Vdash^{\mathrm{T}}_{\chi} \diamond$, and so it follows from Definition 4.1.7 that $\vdash \chi$. Then, since by Lemma 4.1.6(6) we have that $\mathrm{D} <:_{\chi} \mathrm{D}'$, it follows by Property 3.2.2(1) that $\mathrm{C}' = \mathrm{C}_j$, and so also that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}'_j : \mathrm{C}' : \phi'$. Therefore, by rule (P-ASS$_1$) we have that $\Pi \Vdash^{\mathrm{P}}_{\chi} (\texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1})).f_j = \mathrm{e}'_j : \mathrm{D}' : \langle f_j : \phi'\rangle$ and since $\mathrm{D}' <:_{\chi} \mathrm{C}$ it follows by rule (P-SUBTYPE) that $\Pi \Vdash^{\mathrm{P}}_{\chi} (\texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1})).f_j = \mathrm{e}'_j : \mathrm{C} : \langle f_j : \phi'\rangle$.

($l_i = f \in \mathbb{FIELD}$-$\mathbb{ID}$, $f \neq f_j$) Then $\tau_i = \phi'$, and by rule (P-NEWFLD) we have that $f = f_{j'}$ with $j \in \overline{n_1}$ such that $j' \neq j$. So, $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}'_{j'} : \mathrm{C}_{j'} : \phi'$ with $\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f_{j'}) = \mathrm{C}_{j'}$ and $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}'_k : \mathrm{C}_k$ with $\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f_k) = \mathrm{C}_k$ for each $k \in \overline{n_1}$ such that $k \neq j'$. By assumption, $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} (\texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1})).f_j = \mathrm{e}'_j : \mathrm{C}$. So by Lemma 4.1.6(4), $\exists\, \mathrm{D}' <:_{\chi} \mathrm{C}$ such that $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1}) : \mathrm{D}'$ and $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}'_j : \mathrm{C}'$ with $\Delta_{\mathrm{f}}(\chi, \mathrm{D}', f_j) = \mathrm{C}'$. Then, by Lemma 4.1.6(6), $\mathrm{D} <:_{\chi} \mathrm{D}'$ and $\widehat{\Pi} \Vdash^{\mathrm{T}}_{\chi} \mathrm{e}_k : \mathrm{C}_k$ such that $\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f_k) = \mathrm{C}_k$ for each $k \in \overline{n_1}$. Now, since $\mathrm{e}_{j'} = \mathrm{e}'_{j'}$, it follows immediately that $\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e}_{j'} : \mathrm{C}_{j'} : \phi'$. So, by rule (P-NEWFLD) we have that $\Pi \Vdash^{\mathrm{P}}_{\chi} \texttt{new}\ \mathrm{D}(\overline{\mathrm{e}}_{n_1}) : \mathrm{D} : \langle f_{j'} : \phi'\rangle$. By assumption we have that $\Vdash^{\mathrm{T}}_{\chi} \diamond$, and so it follows from Definition 4.1.7 that $\vdash \chi$. Then, since

D $<:_\chi$ D$'$, it follows from Property 3.2.2(1) that C$'$ = C$_j$, and so $\widehat{\Pi} \vDash^{\mathrm{T}}_{\chi} \mathrm{e}'_j : \mathrm{C}_j$. Then by rule (p-ass$_2$) we have that $\Pi \vDash^{\mathrm{P}}_{\chi}$ (`new` D($\overline{\mathrm{e}}_{n_1}$)).$f_j = \mathrm{e}'_j$: D : $\langle f_{j'} : \phi' \rangle$. Furthermore, since D $<:_\chi$ C it follows by rule (p-subType) that $\Pi \vDash^{\mathrm{P}}_{\chi}$ (`new` D($\overline{\mathrm{e}}_{n_1}$)).$f_j = \mathrm{e}'_j$: C : $\langle f_{j'} : \phi' \rangle$.

($l_i = m \in$ METHOD-NAME) Then $\tau_i = \langle m : \psi{::}\overline{\phi}_{n_2} \to \phi' \rangle$, and by rule (p-newMeth) we have that $\widehat{\Pi} \vDash^{\mathrm{T}}_{\chi}$ `new` D($\overline{\mathrm{e}'}_{n_1}$) : D and $\Pi' \vDash^{\mathrm{P}}_{\chi} \mathrm{e}_0 : \mathrm{C}' : \phi'$ such that $\Delta_{\mathrm{m}}(\chi, \mathrm{D}, m) = \overline{\mathrm{C}}_{n_2} \to \mathrm{C}'$ and mbody$(\chi, \mathrm{D}, m) = (\overline{x}_{n_2}, \mathrm{e}_0)$ where $\Pi' = \{x_1 : \mathrm{C}_1 : \phi_1, \ldots, x_{n_2} : \mathrm{C}_{n_2} : \phi_{n_2}$, `this` : D : $\psi\}$. By assumption, $\widehat{\Pi} \vDash^{\mathrm{T}}_{\chi}$ (`new` D($\overline{\mathrm{e}}_{n_1}$)).$f_j = \mathrm{e}'_j$: C. So by Lemma 4.1.6(4), $\exists$ D$' <:_\chi$ C such that $\widehat{\Pi} \vDash^{\mathrm{T}}_{\chi}$ `new` D($\overline{\mathrm{e}}_{n_1}$) : D$'$ and $\widehat{\Pi} \vDash^{\mathrm{T}}_{\chi} \mathrm{e}'_j$: C$'$ with $\Delta_{\mathrm{f}}(\chi, \mathrm{D}', f_j) = \mathrm{C}'$. Then, by Lemma 4.1.6(6), D $<:_\chi$ D$'$ and $\widehat{\Pi} \vDash^{\mathrm{T}}_{\chi} \mathrm{e}_k : \mathrm{C}_k$ such that $\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f_k) = \mathrm{C}_k$ for each $k \in \overline{n_1}$, and by rule (t-new) it follows that $\widehat{\Pi} \vDash^{\mathrm{T}}_{\chi}$ `new` D($\overline{\mathrm{e}}_{n_1}$) : D. By rule (p-newMeth) it now follows that $\Pi \vDash^{\mathrm{P}}_{\chi}$ `new` D($\overline{\mathrm{e}}_{n_1}$) : D : $\langle m : \psi{::}\overline{\phi}_{n_2} \to \phi' \rangle$, and since D $<:_\chi$ D$'$ we have by rule (p-subType) that $\Pi \vDash^{\mathrm{P}}_{\chi}$ `new` D($\overline{\mathrm{e}}_{n_1}$) : D$'$: $\langle m : \psi{::}\overline{\phi}_{n_2} \to \phi' \rangle$. Now, by rule (p-ass$_2$) it follows that $\Pi \vDash^{\mathrm{P}}_{\chi}$ (`new` D($\overline{\mathrm{e}}_{n_1}$)).$f_j = \mathrm{e}'_j$: D$'$: $\langle m : \psi{::}\overline{\phi}_{n_2} \to \phi' \rangle$. Furthermore, since D$' <:_\chi$ C we have by rule (p-subType) that $\Pi \vDash^{\mathrm{P}}_{\chi}$ (`new` D($\overline{\mathrm{e}}_{n_1}$)).$f_j = \mathrm{e}'_j$: C : $\langle m : \psi{::}\overline{\phi}_{n_2} \to \phi' \rangle$.

Since the choice of i was arbitrary, we have that $\Pi \vDash^{\mathrm{P}}_{\chi}$ (`new` D($\overline{\mathrm{e}}_{n_1}$)).$f_j = \mathrm{e}'_j$: C : $\langle l_i : \tau_i \rangle$ for each $i \in \overline{n}$. Then by rule (p-seq) it follows that $\Pi \vDash^{\mathrm{P}}_{\chi}$ (`new` D($\overline{\mathrm{e}}_{n_1}$)).$f_j = \mathrm{e}'_j$: C : $\langle l_i : \tau_i \ {}^{i \,\in\, \overline{n}} \rangle$.

(r-invk) Then e $\equiv$ (`new` D($\overline{\mathrm{e}}_{n_1}$)).$m$($\overline{\mathrm{e}'}_{n_2}$) and e$' \equiv \mathrm{e}_0[\mathrm{e}'_1/x_1, \ldots, \mathrm{e}'_{n_2}/x_{n_2}$, `new` D($\overline{\mathrm{e}}_{n_1}$)/`this`] with mbody (D, m) = ($\overline{x}_{n_2}$, e$_0$). By assumption, we have that $\Pi \vDash^{\mathrm{P}}_{\chi} \mathrm{e}_0[\mathrm{e}'_1/x_1, \ldots, \mathrm{e}'_{n_2}/x_{n_2}$, `new` D($\overline{\mathrm{e}}_{n_1}$)/`this`] : C : ϕ. Also, by assumption, we have that $\widehat{\Pi} \vDash^{\mathrm{T}}_{\chi}$ (`new` D($\overline{\mathrm{e}}_{n_1}$)).$m$($\overline{\mathrm{e}'}_{n_2}$) : C. So, by Lemma 4.1.6(5), $\exists$ D$'$, $\overline{\mathrm{C}}_{n_2}$, and C$'' <:_\chi$ C such that $\widehat{\Pi} \vDash^{\mathrm{T}}_{\chi}$ `new` D($\overline{\mathrm{e}}_{n_1}$) : D$'$ and $\Delta_{\mathrm{m}}(\chi, \mathrm{D}', m) = \overline{\mathrm{C}}_{n_2} \to \mathrm{C}''$ with $\widehat{\Pi} \vDash^{\mathrm{T}}_{\chi} \mathrm{e}_k : \mathrm{C}_k$ for each $k \in \overline{n_2}$. Also, by Lemma 4.1.6(6), it follows that D $<:_\chi$ D$'$ and $\mathcal{F}(\chi, \mathrm{D}) = \overline{f}_{n_1}$ with $\widehat{\Pi} \vDash^{\mathrm{T}}_{\chi} \mathrm{e}_k : \mathrm{D}_k$ such that $\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f_k) = \mathrm{D}_k$ for each $k \in \overline{n_1}$. Therefore, by rule (t-new) we have that $\widehat{\Pi} \vDash^{\mathrm{T}}_{\chi}$ `new` D($\overline{\mathrm{e}}_{n_1}$) : D. Since, by assumption, we have $\vDash^{\mathrm{T}}_{\chi} \diamond$ it follows from Definition 4.1.7 that $\Gamma' \vDash^{\mathrm{T}}_{\chi} \mathrm{e}_0 : \mathrm{C}'$ where $\Delta_{\mathrm{m}}(\chi, \mathrm{D}, m) = \overline{\mathrm{C}'}_{n_2} \to \mathrm{C}'$ and $\Gamma' = \{x_1 : \mathrm{C}'_1, \ldots, x_{n_2} : \mathrm{C}'_{n_2}$, `this` : D$\}$. We also have from Definition 4.1.7 that $\vdash \chi$ and so from Property 3.2.2(2) that C$''$ = C$'$, and C$_k$ = C$'_k$ for each $k \in \overline{n_2}$, since D $<:_\chi$ D$'$. Then it follows that $\widehat{\Pi} \vDash^{\mathrm{T}}_{\chi} \mathrm{e}'_k : \mathrm{C}'_k$ for each $k \in \overline{n_2}$. It also follows, then, that C$' <:_\chi$ C and so by rule (t-sub) we have that $\Gamma' \vDash^{\mathrm{T}}_{\chi} \mathrm{e}_0$: C. Now, it follows from Corollary 5.1.12 that $\exists\, \overline{\phi}_{n_2}$ and ψ such that $\Pi' \vDash^{\mathrm{P}}_{\chi} \mathrm{e}_0 : \mathrm{C} : \phi$ where $\Pi' = \{x_1 : \mathrm{C}'_1 : \phi_1, \ldots, x_{n_2} : \mathrm{C}'_{n_2} : \phi_{n_2}$, `this` : D : $\psi\}$, with $\Pi \vDash^{\mathrm{P}}_{\chi}$ (`new` D($\overline{\mathrm{e}}_{n_1}$)) : D : ψ and $\Pi \vDash^{\mathrm{P}}_{\chi} \mathrm{e}_k : \mathrm{C}'_k : \phi_k$ for each $k \in \overline{n_2}$. So, by rule (p-newMeth) we have that $\Pi \vDash^{\mathrm{P}}_{\chi}$ (`new` D($\overline{\mathrm{e}}_{n_1}$)) : D : $\langle m : \psi{::}\overline{\phi}_{n_2} \to \phi \rangle$, and by rule (p-invk), that $\Pi \vDash^{\mathrm{P}}_{\chi}$ (`new` D($\overline{\mathrm{e}}_{n_1}$)).$m$($\overline{\mathrm{e}'}_{n_2}$) : C$'$: ϕ. Then, since C$' <:$ C we have by rule (p-subType) that $\Pi \vDash^{\mathrm{P}}_{\chi}$ (`new` D($\overline{\mathrm{e}}_{n_1}$)).$m$($\overline{\mathrm{e}'}_{n_2}$) : C : ϕ.

(rc-fld) Then e $\equiv \mathrm{e}_0.f$ and e$' \equiv \mathrm{e}'_0.f$ with $\mathrm{e}_0 \to_\chi \mathrm{e}'_0$. By assumption we have that $\Pi \vDash^{\mathrm{P}}_{\chi} \mathrm{e}'_0 : \mathrm{C} : \phi$ and so by Lemma 4.2.10(3), $\exists$ D, C$' <:_\chi$ C and $\psi \trianglelefteq \phi$ such that $\Delta_{\mathrm{f}}(\chi, \mathrm{D}, f) = \mathrm{C}'$ and $\Pi \vDash^{\mathrm{P}}_{\chi} \mathrm{e}'_0$: D : $\langle f : \psi \rangle$. Also, by assumption we have that $\widehat{\Pi} \vDash^{\mathrm{T}}_{\chi} \mathrm{e}_0.f$: C and so by Lemma 4.1.6(3), $\exists$ D$'$, C$'' <:_\chi$ C such that $\widehat{\Pi} \vDash^{\mathrm{T}}_{\chi} \mathrm{e}_0 : \mathrm{D}'$ and $\Delta_{\mathrm{f}}(\chi, \mathrm{D}', f) = \mathrm{C}''$. Since $\mathrm{e}_0 \to_\chi \mathrm{e}_0$, by Theorem 5.2.1 it follows that $\widehat{\Pi} \vDash^{\mathrm{T}}_{\chi} \mathrm{e}'_0 : \mathrm{D}'$. Then, by Theorem 4.3.3 we have that $\Pi \vDash^{\mathrm{P}}_{\chi} \mathrm{e}'_0 : \mathrm{D}' : \langle f : \psi \rangle$. Now, by the inductive hypothesis it follows that $\Pi \vDash^{\mathrm{P}}_{\chi} \mathrm{e}_0 : \mathrm{D}' : \langle f : \psi \rangle$ and so by rule (p-fld) we have that $\Pi \vDash^{\mathrm{P}}_{\chi} \mathrm{e}_0.f : \mathrm{C}'' : \psi$. Then, since C$'' <:_\chi$ C it follows by rule (p-subType) that $\Pi \vDash^{\mathrm{P}}_{\chi} \mathrm{e}_0.f : \mathrm{C} : \psi$. Furthermore, since $\psi \trianglelefteq \phi$ we have by Theorem 4.3.2 that $\Pi \vDash^{\mathrm{P}}_{\chi} \mathrm{e}_0.f : \mathrm{C} : \phi$.

(rc-invk$_2$) Then e $\equiv \mathrm{e}_0.m(\overline{\mathrm{e}}_{n_1})$ and e$' \equiv \mathrm{e}_0.m(\overline{\mathrm{e}'}_{n_1})$ with $\mathrm{e}_j \to_\chi \mathrm{e}'_j$ for some $j \in \overline{n_1}$ and $\mathrm{e}'_k = \mathrm{e}_k$ for each $k \in \overline{n_1}$ such that $k \neq j$. By assumption we have that $\Pi \vDash^{\mathrm{P}}_{\chi} \mathrm{e}'_0.m(\overline{\mathrm{e}'}_{n_1}) : \mathrm{C} : \phi$ and so by Lemma 4.2.10(5), $\exists$ D, $\overline{\mathrm{C}}_{n_1}$, C$' <:_\chi$ C, ψ, $\overline{\phi}_{n_1}$, $\phi' \trianglelefteq \phi$ such that $\Pi \vDash^{\mathrm{P}}_{\chi} \mathrm{e}_0 : \mathrm{D} : \langle m : \psi{::}\overline{\phi}_{n_1} \to \phi' \rangle$ and $\Pi \vDash^{\mathrm{P}}_{\chi} \mathrm{e}_0 : \mathrm{D} : \psi$ with $\Pi \vDash^{\mathrm{P}}_{\chi} \mathrm{e}'_k : \mathrm{C}_k : \phi_k$ for each $k \in \overline{n_1}$ such that $\Delta_{\mathrm{m}}(\chi, \mathrm{D}, m) = \overline{\mathrm{C}}_{n_1} \to \mathrm{C}'$. Also by assumption we have that $\widehat{\Pi} \vDash^{\mathrm{T}}_{\chi} \mathrm{e}_0.m(\overline{\mathrm{e}}_{n_1})$: C and so by Lemma 4.1.6(5), $\exists$ D$'$, $\overline{\mathrm{C}'}_{n_1}$, C$'' <:_\chi$ C such that $\widehat{\Pi} \vDash^{\mathrm{T}}_{\chi} \mathrm{e}_0 : \mathrm{D}'$ and $\widehat{\Pi} \vDash^{\mathrm{T}}_{\chi} \mathrm{e}_k : \mathrm{C}'_k$ for each $k \in \overline{n_1}$ such that $\Delta_{\mathrm{m}}(\chi, \mathrm{D}', m) = \overline{\mathrm{C}'}_{n_1} \to \mathrm{C}''$. Since $\widehat{\Pi} \vDash^{\mathrm{T}}_{\chi} \mathrm{e}_j : \mathrm{C}'_j$ and $\mathrm{e}_j \to_\chi \mathrm{e}'_j$ it follows by Theorem 5.2.1 that $\widehat{\Pi} \vDash^{\mathrm{T}}_{\chi} \mathrm{e}'_j : \mathrm{C}'_j$, and so by Theorem 4.3.3 we have that $\Pi \vDash^{\mathrm{P}}_{\chi} \mathrm{e}'_j : \mathrm{C}'_j : \phi_j$. Then by the inductive hypothesis it follows that $\Pi \vDash^{\mathrm{P}}_{\chi} \mathrm{e}_j : \mathrm{C}'_j : \phi_j$. Also, since $\mathrm{e}_k = \mathrm{e}'_k$ for each $k \in \overline{n_1}$ such that $k \neq j$, it follows that $\Pi \vDash^{\mathrm{P}}_{\chi} \mathrm{e}_k : \mathrm{C}_k : \phi_k$ for each $k \in \overline{n_1}$ such that $k \neq j$, and then by Theorem 4.3.3, that $\Pi \vDash^{\mathrm{P}}_{\chi} \mathrm{e}_k : \mathrm{C}'_k : \phi_k$ for each $k \in \overline{n_1}$ such that $k \neq j$.

Lastly, since $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0 : \mathrm{D} : \langle m : \psi{::}\overline{\phi}_{n_1} \rightarrow \phi' \rangle$ and $\widehat{\Pi} \vdash^{\mathrm{T}}_{\chi} \mathrm{e}_0 : \mathrm{D}'$ it also follows by Theorem 4.3.3 that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0 : \mathrm{D}' : \langle m : \psi{::}\overline{\phi}_{n_1} \rightarrow \phi' \rangle$. So, by rule (P-INVK), we then have that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0.m(\overline{\mathrm{e}}_{n_1}) : \mathrm{C}'' : \phi'$ and since $\mathrm{C}'' <:_{\chi} \mathrm{C}$ it follows by rule (P-SUBTYPE) that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0.m(\overline{\mathrm{e}}_{n_1}) : \mathrm{C} : \phi'$. Furthermore, since $\phi' \trianglelefteq \phi$ we have by Theorem 4.3.2 that $\Pi \vdash^{\mathrm{P}}_{\chi} \mathrm{e}_0.m(\overline{\mathrm{e}}_{n_1}) : \mathrm{C} : \phi$

□

5.4 Characterisation of Expressions

As with the intersection type system for the λ-calculus, the subject expansion result for the predicate system allows us to characterise the behaviour of LJ expressions by the predicates that we can assign to them. We can investigate this property of the predicate system by comparing similar results from the λ-calculus. It is important to note that we have not proved these characterisation results for the LJ predicate system, however we believe that they will hold due to the similarity between this system and the intersection type systems for the λ-calculus.

We first see that (using an object predicate environment) any LJ expression that terminates in an object can be assigned a non-trivial predicate that *does not* include ω. To see this, we first notice that such an expression has a reduction sequence of the form $\mathrm{e} \rightarrow^{*}_{\chi} \mathrm{o} \equiv \mathrm{e} \rightarrow_{\chi} \mathrm{e}_n \rightarrow_{\chi} \ldots \rightarrow_{\chi} \mathrm{e}_1 \rightarrow_{\chi} \mathrm{o}$ for some (possibly empty) sequence of expressions $\overline{\mathrm{e}}_n$. It is easy to see that the object o can be assigned such a non-trivial predicate (using an object predicate environment). Then, using the subject expansion theorem, we can see that the same predicate is assignable to each of the expressions e_i in turn, and so then also to e. Such expressions correspond to λ-terms which have a normal form (a term on which no further reduction is possible). Any term in normal form may be assigned an intersection type without using ω, and by the expansion result for the intersection type assignment system, so can any term which reduces to it.

We now turn our attention to expressions which are non-terminating. These correspond to λ terms which do not have a normal form. An example of such a term is $(\lambda x.xx)(\lambda x.xx)$, which β-reduces to itself in a single step, and thus has neither a normal form, nor a head-normal form. Consider the following program:

```
χ  =  class C extends Object
      {
         C m() { this.m() }
      }

e  =  (new C()).m()
```

The expression e also runs to itself, thus $\mathrm{e} \rightarrow_{\chi} \mathrm{e} \rightarrow_{\chi} \ldots$ ad infinitum. Notice that $\emptyset \vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \mathtt{C}$, and so by rule (P-OMEGA) we have $\emptyset \vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathtt{C} : \omega$. However, ω is the *only* predicate that we can assign to e. This is also the case for λ-terms without a normal form or a head-normal form.

Finally we look at LJ expressions that correspond to those terms in the λ-calculus that have a head-normal form. These terms correspond to computations that are non-terminating, yet still return some form of meaningful result. An example of such a λ-term is the fixed point operator $\lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$. This term has the following sequence of reductions:

$$\begin{array}{ll}
 & \lambda f.(\lambda x.f(xx))(\lambda x.f(xx)) \\
\rightarrow_{\beta} & \lambda f.f(\lambda f.(\lambda x.f(xx))(\lambda x.f(xx))) \\
\rightarrow_{\beta} & \lambda f.f(f(\lambda f.(\lambda x.f(xx))(\lambda x.f(xx)))) \\
\rightarrow_{\beta} & \lambda f.f(f(f(\lambda f.(\lambda x.f(xx))(\lambda x.f(xx))))) \\
\rightarrow_{\beta} & \ldots
\end{array}$$

Now, consider the following LJ program:

$$\begin{array}{lll}
\chi & = & \texttt{class C extends Object} \\
& & \{ \\
& & \quad \texttt{C f} \\
& & \quad \texttt{C m() \{ this.f = this.m() \}} \\
& & \} \\
\\
\mathrm{e} & = & \texttt{(new C()).m()}
\end{array}$$

which results in this sequence of reductions:

$$\begin{array}{llll}
& \mathrm{e} & \equiv & \texttt{(new C((C) null)).m()} \\
\rightarrow_\chi & & & \texttt{(new C((C) null)).f = (new C((C) null)).m()} \\
\rightarrow_\chi & \mathrm{e}_1 & \equiv & \texttt{new C((new C((C) null)).m())} \\
\rightarrow_\chi & & & \texttt{new C((new C((C) null)).f = (new C((C) null)).m())} \\
\rightarrow_\chi & \mathrm{e}_2 & \equiv & \texttt{new C(new C((new C((C) null)).m()))} \\
\rightarrow^*_\chi & \mathrm{e}_3 & \equiv & \texttt{new C(new C(new C((new C((C) null)).m())))} \\
& & & \vdots
\end{array}$$

Thus, this expression constructs an ever increasingly nested object. Observe that $\emptyset \vdash^{\mathrm{T}}_{\chi} \mathrm{e} : \texttt{C}$ and so by rule (P-OMEGA) it follows that $\emptyset \vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \texttt{C} : \omega$. Given this, we can assign the following predicates to the sequence of expressions $\overline{\mathrm{e}}_n$:

$$\begin{array}{lll}
\emptyset & \vdash^{\mathrm{P}}_{\chi} & \mathrm{e}_1 : \texttt{C} : \langle \texttt{f} : \omega \rangle \\
\emptyset & \vdash^{\mathrm{P}}_{\chi} & \mathrm{e}_2 : \texttt{C} : \langle \texttt{f} : \langle \texttt{f} : \omega \rangle\rangle \\
\emptyset & \vdash^{\mathrm{P}}_{\chi} & \mathrm{e}_3 : \texttt{C} : \langle \texttt{f} : \langle \texttt{f} : \langle \texttt{f} : \omega \rangle\rangle\rangle \\
& & \vdots
\end{array}$$

Then, by subject expansion, we can assign all of these predicates to the expression e itself. Thus, we can assign a non-trivial predicate to e, but it must contain ω, indicating that there is non-termination somewhere. Again, this is the case for the intersection type assignment system for the λ-calculus.

A final point that we can make concerns the expressiveness of the predicate system over the type system. A result of the type system (and similarly of the type systems of FJ, MJ and Java itself) is that if an expression is typeable, then executing the expression will not result in any illegal field accesses or method invocations. In other words, whenever a field is accessed, or a method invoked, such a field or method will always exist in the receiving object. One thing that *may* happen, however, is a null reference exception. This occurs when a field is accessed or a method invoked on a null object. The type system does not distinguish between the types of null objects and the types of non-null objects; thus it cannot determine when such a mismatch will occur. The predicate system, on the other hand, *does* make such a distinction: the only non-trivial predicate that null objects may be assigned is the empty predicate $\langle \epsilon \rangle$. As such, a field access or a method invocation on a null object cannot be assigned any predicate other than ω, since the premise for such a predicate assignment is that the receiver have an appropriate non-empty object predicate. Thus, again by subject expansion, it follows that the execution of any expression which can be assigned a non-trivial predicate will not result in a null reference exception.

Chapter 6

Decidable Restrictions of the Predicate System

In Chapter 5 we showed a subject expansion result for the predicate assignment system. This suggests that predicate assignment is undecidable, as is the case for intersection type assignment systems for the λ-calculus and the predicate system for the ς-calculus in [5] (which also display subject expansion properties). In this chapter, we define a two-tier restriction of the predicate system which we believe *is* decidable. That is, there exists an algorithm which will terminate resulting in a *yes/no* answer indicating whether a non-trivial predicate can be assigned to that expression. We will present such an algorithm, which takes as input an execution context, a (type) environment and an expression, and argue that it is terminating. If the algorithm terminates in the affirmative, then it also returns a predicate. We conjecture that the algorithm is *sound* in that whenever it returns a predicate, there exists a predicate derivation which assigns that predicate to the input expression.

6.1 Decidability of Type Assignment

Since expressions are annotated with type information, type assignment is decidable, and collapses to the simple process of type checking. That is, in order to infer a type for an expression, we need only look at the types declared in the syntax when a new (possibly null) object is created, and then subsequently look up the types of its fields and methods (checking that objects assigned to fields or passed as arguments to methods have an appropriate type).

What we notice, however, is that open terms do not necessarily have a unique type: the type of an expression will depend upon the types of the variables. To see that this is the case, consider the following execution context:

```
χ  =  class A extends Object     class C extends Object
      {                          {
         B f                        D f
      }                          }

      class B extends Object  { }
      class D extends Object  { }
```

Then, when we come to assign a type to the expression `x.f`, the type that we can assign depends upon the type of `x` . So, *both* of the following are valid statements:

$$\{\mathtt{x} : \mathtt{A}\} \ \vdash^{\mathcal{T}}_{\chi} \ \mathtt{x.f} : \mathtt{B}$$
$$\{\mathtt{x} : \mathtt{C}\} \ \vdash^{\mathcal{T}}_{\chi} \ \mathtt{x.f} : \mathtt{D}$$

Thus, an open expression does not, in general, have a *unique* type. Finding each possible type environment and assignable type for an expression is still a decidable problem, however such an algorithm would need to utilise back-tracking techniques and would necessarily be more complex than one which is constrained to find a single solution. Therefore, to simplify the type inference algorithm, we require that a type environment is provided. This will also be the case for the predicate inference algorithms of §6.2 and §6.3.

We now present the type inference algorithm itself:

Definition 6.1.1 (TYPE INFERENCE ALGORITHM)**.** The class type inference algorithm, *Type*, takes an execution context and a type environment, and returns the type of the given expression. It is defined as follows:

$$
\begin{array}{lllllllll}
\mathit{Type} & \chi & \Gamma & (\mathrm{C})\,\mathtt{null} & = & \mathrm{C} & & & \\
\mathit{Type} & \chi & \Gamma & x & = & \mathrm{C} & \text{if} & x : \mathrm{C} \in \Gamma & \\
\mathit{Type} & \chi & \Gamma & \mathrm{e}.f & = & \mathrm{D} & \text{if} & \Delta_{\mathrm{f}}(\chi, \mathrm{C}, f) = \mathrm{D} & \\
 & & & & & \text{where} & \multicolumn{2}{l}{\mathrm{C} = \mathit{Type}\,(\chi, \Gamma, \mathrm{e})} & \\
\mathit{Type} & \chi & \Gamma & \mathrm{e}_1.f = \mathrm{e}_2 & = & \mathrm{C} & \text{if} & \mathrm{D} <:_{\chi} \Delta_{\mathrm{f}}(\chi, \mathrm{C}, f) & \\
 & & & & & \text{where} & \multicolumn{2}{l}{\mathrm{C} = \mathit{Type}\,(\chi, \Gamma, \mathrm{e}_1)} & \\
 & & & & & & \multicolumn{2}{l}{\mathrm{D} = \mathit{Type}\,(\chi, \Gamma, \mathrm{e}_2)} & \\
\mathit{Type} & \chi & \Gamma & \mathrm{e}_0.m(\overline{\mathrm{e}}_n) & = & \mathrm{D} & \text{if} & \mathrm{C}_i <:_{\chi} \mathrm{C}'_i & \forall\, i \in \overline{n} \\
 & & & & & \text{where} & \multicolumn{2}{l}{\mathrm{C} = \mathit{Type}\,(\chi, \Gamma, \mathrm{e}_0)} & \\
 & & & & & & \multicolumn{2}{l}{\mathrm{C}_i = \mathit{Type}\,(\chi, \Gamma, \mathrm{e}_i)} & \forall\, i \in \overline{n} \\
 & & & & & & \multicolumn{2}{l}{\Delta_{\mathrm{m}}(\chi, \mathrm{C}, m) = \overline{\mathrm{C}'}_n \rightarrow \mathrm{D}} & \\
\mathit{Type} & \chi & \Gamma & \mathtt{new}\,\mathrm{C}(\overline{\mathrm{e}}_n) & = & \mathrm{C} & \text{if} & \mathrm{C}_i <:_{\chi} \mathrm{C}'_i & \forall\, i \in \overline{n} \\
 & & & & & \text{where} & \multicolumn{2}{l}{\mathcal{F}(\chi, \mathrm{C}) = \overline{f}_n} & \\
 & & & & & & \multicolumn{2}{l}{\Delta_{\mathrm{f}}(\chi, \mathrm{C}, f_i) = \mathrm{C}'_i} & \forall\, i \in \overline{n} \\
 & & & & & & \multicolumn{2}{l}{\mathrm{C}_i = \mathit{Type}\,(\chi, \Gamma, \mathrm{e}_i)} & \forall\, i \in \overline{n}
\end{array}
$$

The algorithm fails whenever any of the side-conditions are not met, that is any of the following:

- Variable look-up in the type environment fails (i.e. no statement with the specified variable as the subject exists in the type environment),
- Type lookup fails to return a type from the field table Δ_{f} or method table Δ_{m} (i.e. no such field or method exists in the given class),
- The number of arguments to a method call does not match the length of the sequence of argument types returned by the method table Δ_{m},
- The number of sub-expressions in the sequence $\overline{\mathrm{e}}_n$ of a new object creation expression $\mathtt{new}\ \mathrm{D}(\overline{\mathrm{e}}_n)$ does not match the length of the sequence returned by the field list look-up function $\mathcal{F}$,
- A subtype check $\mathrm{C}_i <:_{\chi} \mathrm{C}'_i$ fails (i.e. C_i is *not* a subtype of C'_i).

This algorithm illustrates the rationale behind the choice to define null object expressions with a type annotation. Since the algorithm is defined inductively, without such type annotations we cannot know, at the point where we must assign a type to a null object expression, which object it must behave as (i.e. which fields and methods will be called on it). Therefore, we would have to incorporate back-tracking techniques into the algorithm in order that we could reassign the correct type once we have examined the context in which the null object appears, and the correct type can be inferred.

It is not difficult to see that this algorithm terminates, and we now argue informally to this effect. Firstly, we note that the algorithm operates recursively, i.e. by calling itself. At each recursive call, however, the size of the expression on which it operates becomes smaller. Therefore, the recursion is limited by the length of the expression upon which the algorithm operates. It then remains for us to check that the other elements of the algorithm are terminating, namely the type environment look-up, the checking whether one type is a subtype of another, the looking up of types in the field and method tables, and the looking up of field lists. The type environment look-up procedure is clearly terminating if the type environment contains a finite number of statements. As we will explain in a moment, the termination of the other operations is dependent upon the well formedness of the execution context passed to the algorithm, and one property of well formed contexts in particular: the acyclic nature of the class hierarchy. This property ensures that there are no cycles in the inheritance hierarchy of each class defined in the execution context. Since the field list look-up function, and the field and method tables are also defined recursively (see Definitions 3.1.4 and 3.1.5 respectively), the acyclic nature of the class hierarchy means that there will be no infinite looping in the implementation of these functions. Then, since there must be a finite number of classes in the execution context (and there are a finite number of fields and methods in each class), the level of recursion is limited by the number of classes defined.

To check whether the class hierarchy is acyclic is also a decidable problem, since the maximum length of an acyclic inheritance hierarchy is bounded by the number of classes defined in the context. To construct the inheritance hierarchy for a class, we extract the name of its superclass from the class definition, then look up the definition of the superclass and extract the name of *its* superclass, etc. We do this until we obtain a class name which is not defined in the context, *or* we have obtained a sequence whose length matches the number of classes defined in the context. Since we bound the length of the sequence in this way, we are sure that our look-up operation will terminate. Then, to check that the hierarchy is acyclic, we simply check that each class name in the sequence is *unique*, which again is a terminating operation, bounded by the number of class names in the sequence.

We assert that the type inference algorithm is both *sound* and *complete* with respect to type assignment:

Property 6.1.2 (soundness of type inference)**.** $\chi \vdash \Gamma \;\&\; \mathit{Type}\,(\chi, \Gamma, \mathrm{e}) = \mathrm{C} \Rightarrow \Gamma \vdash^{T}_{\chi} \mathrm{e} : \mathrm{C}$

Property 6.1.3 (completeness of type inference)**.** $\Gamma \vdash^{T}_{\chi} \mathrm{e} : \mathrm{C} \Rightarrow \exists\, \mathrm{D} <:_{\chi} \mathrm{C}\ [\mathit{Type}\,(\chi, \Gamma, \mathrm{e}) = \mathrm{D}]$

6.2 The Rank-0 Restriction

At a conceptual level, the undecidability of predicate assignment stems from the fact that non-terminating expressions *cannot* be assigned non-trivial predicates (see the discussion in §5.4). Thus, a terminating algorithm which is complete (in the sense that if some non-trivial predicate can be assigned to an expression then the algorithm will find it) would be a solution to the halting problem. A decidable system, therefore, can only be complete with respect to some subset of (non-trivial) predicates. We choose to define such subsets in terms of the number of nested method invocations, what would be termed 'stack depth' in conventional programming.

In this section, we define *Rank-0* predicates, which can only be assigned to object, field access or field assignment expressions. In other words, we restrict ourselves to only being able to assign predicates to expressions in which no methods are invoked.

Definition 6.2.1 (rank-0 predicates)**.**

1. The set of Rank-0 predicates, denoted by $\mathbb{PRED}_0$ and ranged over by α, is defined by the following grammar:

$$\begin{array}{rcll} \alpha & ::= & \varphi & \\ & \mid & \langle f_1 : \alpha_1,\ \ldots\ , f_n : \alpha_n \rangle & \quad (n \geq 0) \end{array}$$

   where φ ranges over a set of *predicate variables*. We say that a Rank-0 predicate is *closed* when it does not contain any predicate variables. Conversely, a predicate is said to be *open* if it *does* contain predicate variables. Notice that the set of closed Rank-0 predicates is a subset of the set of predicates, $\mathbb{PRED}$.

2. A Rank-0 predicate environment is a predicate environment in which the predicate conclusion of each statement is a Rank-0 predicate.

Note that we have now introduced predicate variables. These are required by the inference algorithm, and are used to construct the predicates that are assigned to variables that occur in expressions. Since the algorithm is defined inductively, we will not know at the point when we must assign a predicate to a variable what substructure that predicate will be required to have. Therefore, we assign a predicate containing variables, and bring a more detailed structure to the predicate when the context requires such. As usual, we do this through unification.

Since we have introduced predicate variables, we will now need to define a method for transforming these variables into the predicates that they represent. To achieve this, we define a notion of *predicate substitution*:

Definition 6.2.2 (PREDICATE SUBSTITUTION). 1. The predicate substitution $(\varphi \mapsto \alpha) : \mathbb{PRED}_0 \to \mathbb{PRED}_0$, where φ is a predicate variable and $\alpha \in \mathbb{PRED}_0$, is defined inductively as follows:

$$\begin{array}{lcl} (\varphi \mapsto \alpha)\,\varphi & = & \alpha \\ (\varphi \mapsto \alpha)\,\varphi' & = & \varphi', \text{ if } \varphi' \neq \varphi \\ (\varphi \mapsto \alpha)\,\langle f_i : \alpha_i \;^{i \in \overline{n}}\rangle & = & \langle f_1 : (\varphi \mapsto \alpha)\,\alpha_1, \ldots, f_n : (\varphi \mapsto \alpha)\,\alpha_n\rangle \end{array}$$

2. If $\mathcal{S}_1, \mathcal{S}_2$ are predicate substitutions, then so is the composition $\mathcal{S}_1 \circ \mathcal{S}_2$, where $\mathcal{S}_1 \circ \mathcal{S}_2\ \alpha = \mathcal{S}_1\ (\mathcal{S}_2\ \alpha)$

3. $\mathcal{S}\,\Pi = \{x : \mathrm{C} : \mathcal{S}\,\alpha \mid x : \mathrm{C} : \alpha \in \Pi\}$.

4. If for α_1, α_2 there is a predicate substitution $\mathcal{S}$ such that $\mathcal{S}\,\alpha_1 = \alpha_2$, then α_2 is a *(substitution) instance* of α_1.

5. If α is an open Rank-0 predicate and $\mathcal{S}$ is a substitution such that $\mathcal{S}\,\alpha$ is a closed predicate, then we say that $\mathcal{S}$ is a *closing substitution* for α.

We now define a notion of *unification*, similar to Robinson's notion of unification, as also used in the principal type algorithm of Curry type assignment for the λ-calculus [22]. Our notion of unification is not completely analogous to that of Robinson, however. Robinson's algorithm will fail if there is no substitution that maps both its arguments to a common instance. Our algorithm will *not* fail in this situation, rather it will return the identity substitution (i.e. the substitution that maps each predicate variable to itself). The objective of our notion of unification is simply to combine into a single predicate the information contained in two separate ones. The fail cases of our inference algorithm arise out the *matching* operation (defined below).

Definition 6.2.3 (PREDICATE UNIFICATION). Let $Id_{\mathcal{S}}$ be the substitution that replaces all predicate variable by themselves.

1. Unification is defined over Rank-0 predicates by:

$$\begin{array}{llll} \mathit{unify} & \varphi & \alpha & = (\varphi \mapsto \alpha) \\ \mathit{unify} & \langle f_i : \alpha_i \;^{i \in \overline{n}}\rangle & \varphi & = \mathit{unify}\ \varphi\ \langle f_i : \alpha_i \;^{i \in \overline{n}}\rangle \\ \mathit{unify} & \langle \epsilon \rangle & \langle f_i : \alpha_i \;^{i \in \overline{n}}\rangle & = Id_{\mathcal{S}} \\ \mathit{unify} & \langle f : \alpha \rangle & \langle \epsilon \rangle & = Id_{\mathcal{S}} \\ \mathit{unify} & \langle f : \alpha \rangle & \langle f' : \alpha' \rangle & = \begin{cases} \mathit{unify}\ \alpha\ \alpha' & \text{if } f = f' \\ Id_{\mathcal{S}} & \text{otherwise} \end{cases} \\ \mathit{unify} & \langle f : \alpha \rangle & \langle f_i : \alpha_i \;^{i \in \overline{n}}\rangle & = \mathit{unify}\ (\mathcal{S}\,\langle f : \alpha \rangle)\ (\mathcal{S}\,\langle f_i : \alpha_i \;^{i \in \overline{n-1}}\rangle) \quad \text{if } n > 1 \\ & & & \quad \text{where } \mathcal{S} = \mathit{unify}\ \langle f : \alpha \rangle\ \langle f_n : \alpha_n \rangle \\ \mathit{unify} & \langle f_i : \alpha_i \;^{i \in \overline{n_1}}\rangle & \langle f'_i : \alpha'_i \;^{i \in \overline{n_2}}\rangle & = \mathcal{S}_2 \circ \mathcal{S}_1 \quad \text{if } n_1 > 1 \\ & & & \quad \text{where } \mathcal{S}_1 = \mathit{unify}\ \langle f_n : \alpha_n \rangle\ \langle f'_i : \alpha'_i \;^{i \in \overline{n_2}}\rangle \\ & & & \quad \phantom{\text{where }} \mathcal{S}_2 = \mathit{unify}\ (\mathcal{S}_1\,\langle f_i : \alpha_i \;^{i \in \overline{n_1-1}}\rangle)\ (\mathcal{S}_1\,\langle f'_i : \alpha'_i \;^{i \in \overline{n_2}}\rangle) \end{array}$$

2. The unification operation can be extended to Rank-0 predicate environments as follows:

$$\begin{array}{llll} \mathit{UnifyEnv} & \emptyset & \Pi & = Id_{\mathcal{S}} \\ \mathit{UnifyEnv} & (\Pi_1, x : \mathrm{C} : \alpha_1) & \Pi_2 & = \mathit{UnifyEnv}\ \Pi_1\ \Pi_2 \qquad \text{if } x \notin \mathbb{VARS}^{\mathrm{P}}_{\mathrm{ENV}}(\Pi_2) \\ \mathit{UnifyEnv} & (\Pi_1, x : \mathrm{C} : \alpha_1) & (\Pi_2, x : \mathrm{C} : \alpha_2) & = \mathcal{S}_2 \circ \mathcal{S}_1 \\ & & & \quad \text{where } \mathcal{S}_1 = \mathit{unify}\ \alpha_1\ \alpha_2 \\ & & & \quad \phantom{\text{where }} \mathcal{S}_2 = \mathit{UnifyEnv}\ (\mathcal{S}_1\,\Pi_1)\ (\mathcal{S}_1\,\Pi_2) \end{array}$$

When searching for an appropriate predicate for an expression, the inference algorithm must be able to check that the predicate inferred for a sub-expression matches some expected form. For example, when inferring a predicate for an expression of the form $\mathrm{e}.f$, we must check that the inferred predicate for the sub-expression e has the form $\langle f : \phi \rangle$. Similarly, when we extend this system to be able to handle method invocation in the following section, we will need to check that the predicates inferred for expressions passed as arguments to method invocations match the corresponding predicates in the method type of the receiver. For this purpose, we define a *matching operation*:

Definition 6.2.4 (PREDICATE MATCHING)**.** The predicate matching operation is defined as follows:

$$\begin{array}{llll}
\mathit{match} \;\; \varphi & \varphi' & = & \begin{cases} \textsc{true} & \text{if } \varphi = \varphi' \\ \textsc{false} & \text{otherwise} \end{cases} \\
\mathit{match} \;\; \langle f_i : \alpha_i \;^{i \in \overline{n}} \rangle & \langle \epsilon \rangle & = & \textsc{true} \\
\mathit{match} \;\; \langle f_i : \alpha_i \;^{i \in \overline{n}} \rangle & \langle f : \alpha \rangle & = & \begin{cases} \textsc{true} & \text{if } \exists\, j \in \overline{n} \;\; [f = f_j \;\&\; \mathit{match}\; \alpha_j \; \alpha] \\ \textsc{false} & \text{otherwise} \end{cases} \\
\mathit{match} \;\; \langle f_i : \alpha_i \;^{i \in \overline{n_1}} \rangle & \langle f'_i : \alpha'_i \;^{i \in \overline{n_2}} \rangle & = & \mathit{match}\, \langle f_i : \alpha_i \;^{i \in \overline{n_1}} \rangle \, \langle f'_n : \alpha'_n \rangle \\
& & & \wedge\; \mathit{match}\, \langle f_i : \alpha_i \;^{i \in \overline{n_1}} \rangle \, \langle f'_i : \alpha'_i \;^{i \in \overline{n_2 - 1}} \rangle
\end{array}$$

We now define an *override* operation, which takes an object predicate and overwrites the predicate with which a given (field identifier) label is associated. This must be done when inferring a predicate for field assignment expressions, since in field assignment the previous value is overwritten with the new one.

Definition 6.2.5 (PREDICATE OVERRIDE)**.** The predicate override operation is defined as follows:

$$\begin{array}{lllll}
\mathit{override} & f & \langle \epsilon \rangle & \alpha & = \langle f : \alpha \rangle \\
\mathit{override} & f & \langle f' : \alpha' \rangle & \alpha & = \begin{cases} \langle f' : \alpha \rangle & \text{if } f = f' \\ \langle f' : \alpha' \rangle & \text{otherwise} \end{cases} \\
\mathit{override} & f & \langle f_i : \alpha_i \;^{i \in \overline{n}} \rangle & \alpha & = (\mathit{override}\;\, f \, \langle f_i : \alpha_i \;^{i \in \overline{n-1}} \rangle \; \alpha) \sqcup (\mathit{override}\;\, f \, \langle f_n : \alpha_n \rangle \; \alpha) \\
& & & & \quad \text{where } n > 1
\end{array}$$

In the algorithm, when we infer a predicate for a variable of a field access expression, we must construct a predicate that contains as much information as possible, since we do not know at that point the exact structure that the predicate should have. That is, we cannot be sure which fields will be accessed and assigned to, and which will not. For readability purposes, we define the following function, which returns a Rank-0 predicate assignable to a given type, which is new in the sense that all fields are assigned a fresh predicate variable.

Definition 6.2.6 (FRESH PREDICATE (RANK-0))**.** The function $\mathit{FreshPredicate}_0 : \mathbb{CONTEXT} \times \mathbb{TYPE}_{\mathrm{C}} \to \mathbb{PRED}_0$ returns a Rank-0 predicate containing all fields in a given class C, with each field associated with a fresh predicate variable:

$$\mathit{FreshPredicate}_0(\chi, \mathrm{C}) \;\; = \;\; \langle f_i : \varphi_i \;^{i \in \overline{n}} \rangle \quad \text{where} \quad \mathcal{F}(\chi, \mathrm{C}) = \overline{f}_n, \;\; \overline{\varphi}_n \text{ fresh}$$

The final operation we will define is that of *predicate merge*. This operation allows us to combine the predicate information inferred about a variable occurring in multiple sub-expressions to be combined into a single predicate.

Definition 6.2.7 (PREDICATE MERGE)**.** 1. The *merging* of two Rank-0 predicates, $\alpha_1 + \alpha_2$, is defined inductively as follows:

$$\begin{array}{lclcl}
\varphi & + & \varphi' & = & \varphi \quad \text{if } \varphi = \varphi' \\
\langle \epsilon \rangle & + & \langle f_i : \alpha_i \;^{i \in \overline{n}} \rangle & = & \langle f_i : \alpha_i \;^{i \in \overline{n}} \rangle \\
\langle f : \alpha \rangle & + & \langle f_i : \alpha_i \;^{i \in \overline{n}} \rangle & = & \begin{cases} \langle f_1 : \alpha_1, \ldots, f_j : \alpha_j + \alpha, \ldots, f_n : \alpha_n \rangle & f = f_j \\ \langle f : \alpha \rangle \sqcup \langle f_i : \alpha_i \;^{i \in \overline{n}} \rangle & f \notin \overline{f}_n \end{cases} \\
\langle f_i : \alpha_i \;^{i \in \overline{n}} \rangle & + & \langle f'_i : \alpha'_i \;^{i \in \overline{n'}} \rangle & = & \langle f_i : \alpha_i \;^{i \in \overline{n-1}} \rangle + (\langle f_n : \alpha_n \rangle + \langle f'_i : \alpha'_i \;^{i \in \overline{n'}} \rangle) \\
& & & & \text{where } n > 1
\end{array}$$

2. We extend the merge operation to predicate environments in a straightforward manner:

$$\begin{array}{llcll}
\emptyset & + \ \Pi & = & \Pi & \\
\Pi_1, x : \mathrm{C} : \alpha & + \ \Pi_2 & = & (\Pi_1 + \Pi_2), x : \mathrm{C} : \alpha & \text{if } x \notin \mathbb{VARS}^{\mathrm{P}}_{\mathrm{ENV}}(\Pi_2) \\
\Pi_1, x : \mathrm{C} : \alpha_1 & + \ \Pi_2, x : \mathrm{C} : \alpha_2 & = & (\Pi_1 + \Pi_2), x : \mathrm{C} : \alpha_1 + \alpha_2 &
\end{array}$$

3. We use the notation $\Sigma_{\overline{n}}\ \alpha_i$ to represent a sequence of consecutive merges: $(\ldots(\alpha_1 + \alpha_2) + \ldots) + \alpha_n$. Similarly we use $\Sigma_{\overline{n}}\ \Pi_i$ to denote $(\ldots(\Pi_1 + \Pi_2) + \ldots) + \Pi_n$.

We now give the Rank-0 predicate inference algorithm. It operates in two stages. Firstly, a Rank-0 predicate is inferred for the given expression, using the algorithm $\mathcal{R}_0$. This predicate will be open if the expression contains variables. Thus, to ensure that we return a valid predicate, that is, a (closed) predicate belonging to the set $\mathbb{PRED}$, the predicate variables are eliminated by applying a closing substitution. The closing substitution that we use is, in fact, the least such one: it replaces all predicate variables by the empty predicate.

Definition 6.2.8 (RANK-0 PREDICATE INFERENCE ALGORITHM)**.** Let $\mathcal{S}_\downarrow$ be the predicate substitution that replaces all predicate variables by $\langle\epsilon\rangle$. We define the algorithms *Inf*$_0$ and $\mathcal{R}_0$ as follows:

$$\begin{array}{lll}
\mathit{Inf}_0\ (\chi, \Gamma, \mathrm{e}) & = & \mathcal{S}_\downarrow\ (\Pi,\ \mathrm{C} : \alpha) \\
& & \text{where} \quad (\Pi,\ \mathrm{C} : \alpha) = \mathcal{R}_0\ (\chi, \Gamma, \mathrm{e})
\end{array}$$

$$\begin{array}{llllll}
\mathcal{R}_0\ \chi\ \Gamma\ (\mathrm{C})\,\mathtt{null} & = & (\emptyset,\ \mathrm{C} : \langle\epsilon\rangle) & & & \\
\mathcal{R}_0\ \chi\ \Gamma\ x & = & (\{x : \mathrm{C} : \alpha\},\ \mathrm{C} : \alpha) & & \text{if} & x : \mathrm{C} \in \Gamma \\
& & \text{where} \quad \alpha = \mathit{FreshPredicate}_0(\chi, \mathrm{C}) & & & \\
\mathcal{R}_0\ \chi\ \Gamma\ \mathrm{e}.f & = & (\mathcal{S}\,\Pi,\ \mathrm{D} : (\mathcal{S}\,\alpha')) & & \text{if} & \mathit{match}\ (\mathcal{S}\ \alpha)\ (\mathcal{S}\ \langle f : \varphi_j\rangle) \\
& & \text{where} \quad (\Pi,\ \mathrm{C} : \alpha) = \mathcal{R}_0\ (\chi, \Gamma, \mathrm{e}) & & & \\
& & \qquad \Delta_{\mathrm{f}}(\chi, \mathrm{C}, f) = \mathrm{D} & & & \\
& & \qquad \alpha' = \mathit{FreshPredicate}_0(\chi, \mathrm{D}) = \langle f_i : \varphi_i\ ^{i \,\in\, \overline{n}}\rangle \text{ such that } f = f_j & & & \\
& & \qquad \mathcal{S} = \mathit{unify}\ \alpha\ \alpha' & & & \\
\mathcal{R}_0\ \chi\ \Gamma\ \mathrm{e}_1.f = \mathrm{e}_2 & = & ((\mathcal{S}_2\ \Pi_1) + (\mathcal{S}_2\ \Pi_2),\ \mathrm{C} : \mathcal{S}_2\ \alpha) & & \text{if} & \mathrm{D} <:_\chi \Delta_{\mathrm{f}}(\chi, \mathrm{C}, f) \\
& & & & & \mathit{match}\ (\mathcal{S}_2\ \alpha)\ (\mathcal{S}_2\ \langle f : \varphi\rangle) \\
& & \text{where} \quad (\Pi_1,\ \mathrm{C} : \alpha_1) = \mathit{Inf}_0\ (\chi, \Gamma, \mathrm{e}_1) & & & \\
& & \qquad (\Pi_2,\ \mathrm{D} : \alpha_2) = \mathit{Inf}_0\ (\chi, \Gamma, \mathrm{e}_2) & & & \\
& & \qquad \alpha = \mathit{override}\ f\ \alpha_1\ \alpha_2 & & & \\
& & \qquad \mathcal{S}_1 = \mathit{UnifyEnv}\ \Pi_1\ \Pi_2 & & & \\
& & \qquad \mathcal{S}_2 = (\mathit{unify}\ (\mathcal{S}_1\ \alpha_1)\ \langle f : \varphi\rangle) \circ \mathcal{S}_1 & \varphi \text{ fresh} & & \\
\mathcal{R}_0\ \chi\ \Gamma\ \mathtt{new}\,\mathrm{C}(\overline{\mathrm{e}}_n) & = & (\Sigma_{\overline{n}}\ (\mathcal{S}_n\ \Pi_i),\ \mathrm{C} : \mathcal{S}_n\ \langle f_i : \alpha_i\ ^{i \,\in\, \overline{n}}\rangle) & & \text{if} & \mathrm{C}_i <:_\chi \mathrm{C}'_i\ \forall\, i \in \overline{n} \\
& & \text{where} \quad \mathcal{F}(\chi, \mathrm{C}) = \overline{f}_n & & & \\
& & \qquad \Delta_{\mathrm{f}}(\chi, \mathrm{C}, f_i) = \mathrm{C}'_i & \forall i \in \overline{n} & & \\
& & \qquad (\Pi_i,\ \mathrm{C}_i : \alpha_i) = \mathit{Inf}_0\ (\chi, \Gamma, \mathrm{e}_i) & \forall i \in \overline{n} & & \\
& & \qquad \mathcal{S}_1 = \mathit{Id}_{\mathcal{S}} & & & \\
& & \qquad \mathcal{S}_i = (\mathit{UnifyEnv}\ (\mathcal{S}_{i-1}\ \Pi_{i-1})\ (\mathcal{S}_{i-1}\ \Pi_i)) \circ \mathcal{S}_{i-1} & 1 < i \leq n & &
\end{array}$$

As for the type inference algorithm in §6.1, the Rank-0 predicate inference algorithm will fail whenever any of its side-conditions does. These conditions are the same as for the type inference algorithm, with the addition of the *match* operation. When considering the termination of the Rank-0 algorithm, we must consider the termination of the extra operations that we have defined. The *FreshPredicate*$_0$ function is defined using the field list look-up function $\mathcal{F}$, the termination of which we have already discussed in the context of type inference. Notice that the substitution, unification, matching, overriding and merging operations are defined recursively over the structure of *predicates*. At each recursive call, however, the length of the predicate being operated on is reduced and so the level of recursion is bounded by the size of the predicate. The substitution operation is technically also defined over the number of predicate variables that it acts upon. Thus, an argument for the termination of the substitution operation will also have to take this into account. It is clear, however, that termination is guaranteed for substitutions defined over a finite number of predicate variables. Moreover, the extensions of the unification and and merging operations to predicate environments are defined recursively over the structure of the environment itself. However, again, at each recursive call the size of the environment is reduced, and therefore the level of recursion is again bounded. Two final points to deal with are the special cases of the identity and closing substitutions, $Id_{\mathcal{S}}$ and $\mathcal{S}_{\downarrow}$. Formally, these are defined over the entire (infinite) set of predicate variables. However, in practice they only operate on the (finite number of) predicate variables present in the predicate that they act upon. An implementation of these substitutions can then clearly be made to terminate by defining them to treat *all* predicate variables that they encounter in the same manner.

We make the proposition that the Rank-0 predicate inference algorithm of Definition 6.2.8 is sound with respect to predicate assignment. We also assert that it is *complete* with respect to predicate assignment using only Rank-0 predicates. However, due to time constraints we have not been able to construct a proof for this.

Conjecture 6.2.9 (SOUNDNESS OF RANK-0 PREDICATE INFERENCE)**.** If the Rank-0 inference algorithm returns a predicate environment and a type-predicate pair for a given execution context, type environment and expression, then there exists a predicate derivation assigning the type-predicate pair to the expression using the execution context and predicate environment:

$$\chi \vdash \Gamma \;\&\; \mathit{Inf}_0\,(\chi, \Gamma, \mathrm{e}) = (\Pi,\ \mathrm{C} : \phi) \Rightarrow \Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \phi$$

Conjecture 6.2.10 (COMPLETENESS OF RANK-0 PREDICATE INFERENCE)**.** If $\mathcal{D}$::$\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \alpha$ such that $\mathcal{D}$ contains only Rank-0 predicates, then there exists a Rank-0 predicate α' and a predicate substitution $\mathcal{S}$ such that $\mathcal{R}_0\,(\chi, \widehat{\Pi}, \mathrm{e}) = (\Pi', \alpha')$ and $\mathcal{S}\,\Pi' \trianglelefteq \Pi$ with $\mathcal{S}\,\alpha' \trianglelefteq \alpha$.

6.3 The Rank-1 Restriction

A restriction that precludes assigning predicates to expressions using method invocation is undoubtedly *too* severe. Method invocation is at the heart of LJ, and so a meaningful system should be able to deal with it. For this reason, we now generalise the Rank-0 system to be able to type expressions with method invocations. The Rank-1 system, however, still places restrictions of the type of method invocations that those expressions can contain. The methods that may be invoked are restricted to those whose bodies can be typed with Rank-0 predicates. Thus, the Rank-1 system also cannot type expressions in which a method is invoked on the return value of another method invocation. We first define Rank-1 predicates:

Definition 6.3.1 (RANK-1 PREDICATES)**.**

1. The set of Rank-1 predicates, denoted by $\mathbb{PRED}_1$ and ranged over by β, is defined by the following grammar:

$$\begin{array}{lcll} \beta & ::= & \varphi & \\ & | & \langle l_1 : \tau_1,\ \ldots\ , l_n : \tau_n \rangle & (n \geq 0) \\[1ex] \tau & ::= & \beta & \\ & | & \alpha_0{::}\overline{\alpha}_n \to \alpha & (n \geq 0) \end{array}$$

   Again, we say that a Rank-1 predicate is *closed* when it does not contain any predicate variables. Also, as for closed Rank-0 predicates, the set of closed Rank-1 predicates is a subset of the set of predicates. Notice that $\mathbb{PRED}_0 \subset \mathbb{PRED}_1$.

2. A Rank-1 predicate environment is a predicate environment in which the predicate conclusion of each statement is a Rank-1 predicate.

We must now extend the notion of predicate substitution to operate over Rank-1 predicates, as well as extend the definition of the unification, matching and overriding operations:

Definition 6.3.2 (RANK-1 PREDICATE SUBSTITUTION)**.** $(\varphi \mapsto \beta) : \mathbb{PRED}_1 \to \mathbb{PRED}_1$, where φ is a predicate variable and $\beta \in \mathbb{PRED}_1$, is defined inductively as follows:

$$\begin{array}{lcl}
(\varphi \mapsto \beta)\,\varphi & = & \beta \\
(\varphi \mapsto \beta)\,\varphi' & = & \varphi',\ \text{if } \varphi' \neq \varphi \\
(\varphi \mapsto \beta)\,\alpha_0{::}\overline{\alpha}_n \to \alpha & = & ((\varphi \mapsto \beta)\,\alpha_0){::}((\varphi \mapsto \beta)\,\alpha_1),\ \ldots\ ,((\varphi \mapsto \beta)\,\alpha_n) \to ((\varphi \mapsto \beta)\,\alpha) \\
(\varphi \mapsto \beta)\,\langle l_i : \tau_i \ {}^{i \in \overline{n}}\rangle & = & \langle l_1 : (\varphi \mapsto \beta)\,\tau_1, \ldots, l_n : (\varphi \mapsto \beta)\,\tau_n\rangle
\end{array}$$

This extends to Rank-1 predicate environments in a similar way as for Rank-0 substitution (see Definition 6.2.2).

We now present modifications to the unification, matching and merging operations so that they can handle Rank-1 predicates. The extensions are straightforward and minor, amounting to extra cases in each operation for handling method predicate types, and generalising the cases for object predicates so that they may contain both field identifiers and method names. We do not give the extension for the override operation, since the definition is identical to the Rank-0 version, except that Rank-1 predicates are used instead of Rank-0 ones.

Definition 6.3.3 (UNIFICATION)**.** Unification is defined over Rank-1 predicates by:

$$\begin{array}{llllll}
\mathit{unify} & \varphi & \beta & = & (\varphi \mapsto \beta) \\
\mathit{unify} & \langle l_i : \tau_i \ {}^{i \in \overline{n}}\rangle & \varphi & = & \mathit{unify}\ \varphi\ \langle l_i : \tau_i \ {}^{i \in \overline{n}}\rangle \\[1ex]
\mathit{unify} & \alpha_0{::}\overline{\alpha}_n \to \alpha & \alpha'_0{::}\overline{\alpha'}_n \to \alpha' & = & \mathcal{S} \circ \mathcal{S}_n \circ \ldots \circ \mathcal{S}_0 \\
 & & & & \text{where } \mathcal{S} = \mathit{unify}\ \alpha\ \alpha' \\
 & & & & \mathcal{S}_i = \mathit{unify}\ \alpha_i\ \alpha'_i \text{ for all } 0 \leq i \leq n \\
\mathit{unify} & \langle\epsilon\rangle & \langle l_i : \tau_i \ {}^{i \in \overline{n}}\rangle & = & \mathit{Id}_{\mathcal{S}} \\[1ex]
\mathit{unify} & \langle l : \tau\rangle & \langle\epsilon\rangle & = & \mathit{Id}_{\mathcal{S}} \\
\mathit{unify} & \langle l : \tau\rangle & \langle l' : \tau'\rangle & = & \begin{cases} \mathit{unify}\ \tau\ \tau' & \text{if } l = l' \\ \mathit{Id}_{\mathcal{S}} & \text{otherwise} \end{cases} \\
\mathit{unify} & \langle l : \tau\rangle & \langle l_i : \tau_i \ {}^{i \in \overline{n}}\rangle & = & \mathit{unify}\ (\mathcal{S}\,\langle l : \tau\rangle)\ (\mathcal{S}\,\langle l_i : \tau_i \ {}^{i \in \overline{n-1}}\rangle) \qquad \text{if } n > 1 \\
 & & & & \text{where } \mathcal{S} = \mathit{unify}\ \langle l : \tau\rangle\ \langle l_n : \tau_n\rangle \\[1ex]
\mathit{unify} & \langle l_i : \tau_i \ {}^{i \in \overline{n_1}}\rangle & \langle l'_i : \tau'_i \ {}^{i \in \overline{n_2}}\rangle & = & \mathcal{S}_2 \circ \mathcal{S}_1 \qquad \text{if } n_1 > 1 \\
 & & & & \text{where } \mathcal{S}_1 = \mathit{unify}\ \langle l_n : \tau_n\rangle\ \langle l'_i : \tau'_i \ {}^{i \in \overline{n_2}}\rangle \\
 & & & & \mathcal{S}_2 = \mathit{unify}\ (\mathcal{S}_1\,\langle l_i : \tau_i \ {}^{i \in \overline{n_1-1}}\rangle)\ (\mathcal{S}_1\,\langle l'_i : \tau'_i \ {}^{i \in \overline{n_2}}\rangle)
\end{array}$$

This modification extends to the *UnifyEnv* function in the same way as the Rank-0 version. The definition is identical, but for the obvious fact that the Rank-1 *unify* is used.

Definition 6.3.4 (MATCHING)**.** We extend the *match* operation as follows:

$$\begin{array}{llllll}
\mathit{match} & \varphi & \varphi' & = & \begin{cases} \textsc{true} & \text{if } \varphi = \varphi' \\ \textsc{false} & \text{otherwise} \end{cases} \\
\mathit{match} & \alpha_0{::}\overline{\alpha}_n \to \alpha & \alpha'_0{::}\overline{\alpha'}_n \to \alpha' & = & \mathit{match}\ \alpha\ \alpha' \wedge \mathit{match}\ \alpha_0\ \alpha'_0 \wedge \\
 & & & & \ldots \wedge \mathit{match}\ \alpha_n\ \alpha'_n \\[1ex]
\mathit{match} & \langle l_i : \tau_i \ {}^{i \in \overline{n}}\rangle & \langle\epsilon\rangle & = & \textsc{true} \\
\mathit{match} & \langle l_i : \tau_i \ {}^{i \in \overline{n}}\rangle & \langle l : \tau\rangle & = & \begin{cases} \textsc{true} & \text{if } \exists\ j \in \overline{n}\ \ [l = l_j\ \&\ \mathit{match}\ \tau_j\ \tau] \\ \textsc{false} & \text{otherwise} \end{cases} \\
\mathit{match} & \langle l_i : \tau_i \ {}^{i \in \overline{n_1}}\rangle & \langle l'_i : \tau'_i \ {}^{i \in \overline{n_2}}\rangle & = & \mathit{match}\ \langle l_i : \tau_i \ {}^{i \in \overline{n_1}}\rangle\ \langle l'_n : \tau'_n\rangle \\
 & & & & \wedge\ \mathit{match}\ \langle l_i : \tau_i \ {}^{i \in \overline{n_1}}\rangle\ \langle l'_i : \tau'_i \ {}^{i \in \overline{n_2-1}}\rangle
\end{array}$$

Definition 6.3.5 (PREDICATE MERGE)**.** The predicate merge operation is extended to Rank-1 predicates as follows:

$$
\begin{array}{lclcl}
\varphi & + & \varphi' & = & \varphi \qquad \text{if } \varphi = \varphi' \\
\alpha_0{::}\overline{\alpha}_n \to \alpha & + & \alpha'_0{::}\overline{\alpha'}_n \to \alpha' & = & (\alpha_0 + \alpha'_0){::}(\alpha_1 + \alpha'_1), \ldots, (\alpha_n + \alpha'_n) \to (\alpha + \alpha') \\
\langle \epsilon \rangle & + & \langle l_i : \tau_i \ ^{i \in \overline{n}} \rangle & = & \langle l_i : \tau_i \ ^{i \in \overline{n}} \rangle \\
\langle l : \tau \rangle & + & \langle l_i : \tau_i \ ^{i \in \overline{n}} \rangle & = & \left\{ \begin{array}{ll} \langle l_1 : \tau_1, \ldots, l_j : \tau + \tau_j, \ldots, l_n : \tau_n \rangle & l = l_j \\ \langle l : \tau \rangle \sqcup \langle l_i : \tau_i \ ^{i \in \overline{n}} \rangle & l \notin \overline{l}_n \end{array} \right. \\
\langle l_i : \tau_i \ ^{i \in \overline{n}} \rangle & + & \langle l'_i : \tau'_i \ ^{i \in \overline{n}} \rangle & = & \langle l_i : \tau_i \ ^{i \in \overline{n-1}} \rangle + (\langle l_n : \tau_n \rangle + \langle l'_i : \tau'_i \ ^{i \in \overline{n'}} \rangle) \\
 & & & & \text{where } n > 1
\end{array}
$$

This definition of merging extends to Rank-1 predicate environments in exactly the same way as the Rank-0 merge operation does. We also retain the $\Sigma_{\overline{n}}$ notation.

The extension of the *FreshPredicate*$_0$ function is also straightforward. To generate a fresh Rank-1 predicate, we must also include the (Rank-1) methods for the given class.

Definition 6.3.6 (FRESH PREDICATE (RANK-1))**.** The function *FreshPredicate*$_1$: $\mathbb{CONTEXT} \times \mathbb{TYPE}_\mathrm{C} \to \mathbb{PRED}_1$ returns a Rank-1 predicate containing all fields in a given class C, with each field associated with a fresh predicate variable. It also contains all methods belonging to the class with member predicates constructed from fresh variables:

$$
\begin{array}{lcllr}
\mathit{FreshPredicate}_0(\chi, \mathrm{C}) & = & \multicolumn{3}{l}{\langle f_i : \varphi_i \ ^{i \in \overline{n_1}} \rangle \sqcup \langle m_i : \tau_i \ ^{i \in \overline{n_2}} \rangle} \\
 & & \text{where} & \mathcal{F}(\chi, \mathrm{C}) = \overline{f}_{n_1} & \overline{\varphi}_{n_1} \text{ fresh} \\
 & & & \mathcal{M}(\chi, \mathrm{C}) = \overline{m}_{n_2} & \\
 & & & \Delta_\mathrm{m}(\chi, \mathrm{C}, m) = \overline{\mathrm{C}^i}_{n_i} \to \mathrm{C}_i & \forall\, i \in \overline{n_2} \\
 & & & \tau_i = \varphi_0{::}\overline{\varphi}_{n_i} \to \varphi \qquad \varphi, \varphi_0, \overline{\varphi}_{n_i} \text{ fresh} & \forall\, i \in \overline{n_2}
\end{array}
$$

Note that this construction assumes that all the methods belonging to the class can be assigned Rank-1 predicates. This does not present a particular problem, however, since such fresh Rank-1 predicates will only be generated as part of a variable predicate. Then, the algorithm simply asserts that given an environment in which the variable satisfies such a predicate, the final result is valid. Any predicate inferred for an object constructed via a `new` expression will contain only those methods whose bodies *can* be typed with a Rank-0 predicate. We do not perform this check for variables since variables may represent objects of a *subtype* of their decared type. Therefore, we cannot be sure which method body will actually be executed when the method is invoked on an object substituted for the variable.

We must now define one extra operation on Rank-1 predicates before presenting the Rank-1 predicate inference algorithm itself. This operation is that of *flattening,* or converting a Rank-1 predicate into a Rank-0 predicate by discarding all the information regarding methods. This operation is necessary when defining the case for method invocation in the predicate inference algorithm. In the full predicate assignment system, as well as having a method predicate type, the receiver of a method invocation must also satisfy the self predicate of that method type. Since, in the Rank-1 system, the self predicate must be Rank-0, the algorithm takes the Rank-1 predicate that has been inferred for the receiver expression, and flattens it before matching it against the self predicate of its inferred method type.

Definition 6.3.7 (PREDICATE FLATTENING)**.** If β is a Rank-1 predicate, then we define the flattening operation, $\lfloor \beta \rfloor$, which returns a Rank-0 predicate, as follows:

$$
\begin{array}{lcl}
\lfloor \varphi \rfloor & = & \varphi \\
\lfloor \langle \epsilon \rangle \rfloor & = & \langle \epsilon \rangle \\
\lfloor \langle f : \beta \rangle \rfloor & = & \langle f : \lfloor \beta \rfloor \rangle \\
\lfloor \langle m : \alpha_0{::}\overline{\alpha}_n \to \alpha \rangle \rfloor & = & \langle \epsilon \rangle \\
\lfloor \langle l_i : \tau_i \ ^{i \in \overline{n}} \rangle \rfloor & = & \lfloor \langle l_i : \tau_i \ ^{i \in \overline{n-1}} \rangle \rfloor \sqcup \lfloor \langle l_n : \tau_n \rangle \rfloor
\end{array}
$$

We now present the Rank-1 predicate inference algorithm. The major differences between this algorithm and the Rank-0 algorithm are the extra case for method invocation, and an extension of the case for object creation in which types for its (Rank-0) methods are also inferred as well as fields.

Definition 6.3.8 (RANK-1 PREDICATE INFERENCE)**.** Again, let $\mathcal{S}_\downarrow$ be the subsitution that replaces all predicate variables with $\langle\epsilon\rangle$. We define the algorithms Inf_1 and $\mathcal{R}_1$ as follows:

$$\begin{array}{llll}
Inf_1\ (\chi, \Gamma, \mathrm{e}) & = & \mathcal{S}_\downarrow\ (\Pi\ ,\ \mathrm{C}{:}\ \beta) & \\
 & & \text{where} \quad (\Pi,\ \mathrm{C} : \beta) = \mathcal{R}_1\ (\chi, \Gamma, \mathrm{e}) &
\end{array}$$

$$\begin{array}{llllll}
\mathcal{R}_1 & \chi & \Gamma & (\mathrm{C})\, \mathtt{null} & = & (\emptyset,\ \mathrm{C} : \langle\epsilon\rangle)
\end{array}$$

$$\begin{array}{llllllll}
\mathcal{R}_1 & \chi & \Gamma & x & = & (\{x : \mathrm{C} : \beta\},\ \mathrm{C} : \beta) & \text{if} & x : \mathrm{C} \in \Gamma \\
 & & & & & \text{where} \quad \beta = FreshPredicate_1(\chi, \mathrm{C}) & &
\end{array}$$

$$\begin{array}{llllllll}
\mathcal{R}_1 & \chi & \Gamma & \mathrm{e}.f & = & (\mathcal{S}\,\Pi,\ \mathrm{D} : (\mathcal{S}\,\beta')) & \text{if} & match\ (\mathcal{S}\,\beta)\ (\mathcal{S}\ \langle f : \tau_j\rangle) \\
 & & & & & \text{where} \quad (\Pi,\ \mathrm{C} : \beta) = \mathcal{R}_1\ (\chi, \Gamma, \mathrm{e}) & & \\
 & & & & & \qquad \Delta_{\mathrm{f}}(\chi, \mathrm{C}, f) = \mathrm{D} & & \\
 & & & & & \qquad \beta' = FreshPredicate_1(\chi, \mathrm{D}) = \langle l_i : \tau_i\ ^{i \,\in\, \overline{n}}\rangle \text{ such that } f = l_j & & \\
 & & & & & \qquad \mathcal{S} = unify\ \beta\ \beta' & &
\end{array}$$

$$\begin{array}{llllllll}
\mathcal{R}_1 & \chi & \Gamma & \mathrm{e}_1.f = \mathrm{e}_2 & = & ((\mathcal{S}_2\ \Pi_1) + (\mathcal{S}_2\ \Pi_2),\ \mathrm{C} : \mathcal{S}_2\,\beta) & \text{if} & \mathrm{D} <:_\chi \Delta_{\mathrm{f}}(\chi, \mathrm{C}, f) \\
 & & & & & & & match\ (\mathcal{S}_2\,\beta)\ (\mathcal{S}_2\ \langle f : \varphi\rangle) \\
 & & & & & \text{where} \quad (\Pi_1,\ \mathrm{C} : \beta_1) = Inf_1\ (\chi, \Gamma, \mathrm{e}_1) & & \\
 & & & & & \qquad (\Pi_2,\ \mathrm{D} : \beta_2) = Inf_1\ (\chi, \Gamma, \mathrm{e}_2) & & \\
 & & & & & \qquad \beta = override\ \ f\ \beta_1\ \beta_2 & & \\
 & & & & & \qquad \mathcal{S}_1 = UnifyEnv\ \Pi_1\ \Pi_2 & & \\
 & & & & & \qquad \mathcal{S}_2 = (unify\ (\mathcal{S}_1\,\beta_1)\ \langle f : \varphi\rangle) \circ \mathcal{S}_1 & & \varphi \text{ fresh}
\end{array}$$

$$\begin{array}{llllllll}
\mathcal{R}_1 & \chi & \Gamma & \mathrm{e}_0.m(\overline{\mathrm{e}}_n) & = & ((\mathcal{S}\,\Pi_0) + \Sigma_{\overline{n}}\ (\mathcal{S}\,\Pi_i),\ \mathrm{D} : \mathcal{S}\,\varphi) & \text{if} & \mathrm{C}_i <:_\chi \mathrm{C}'_i\ \forall\ i \in \overline{n}, \\
 & & & & & & & match\ (\mathcal{S}\,\beta)\ (\mathcal{S}\ \langle m : \varphi_0{::}\overline{\varphi}_n \to \varphi\rangle), \\
 & & & & & & & match\ (\mathcal{S}\ \lfloor\beta\rfloor)\ (\mathcal{S}\,\varphi_0), \\
 & & & & & & & match\ (\mathcal{S}\,\alpha_i)\ (\mathcal{S}\,\varphi_i)\ \forall\ i \in \overline{n} \\
 & & & & & \text{where} \quad (\Pi_0,\ \mathrm{C}_0 : \beta) = \mathcal{R}_1\ (\chi, \Gamma, \mathrm{e}_0) & & \\
 & & & & & \qquad (\Pi_i,\ \mathrm{C}_i : \alpha_i) = \mathcal{R}_0\ (\chi, \Gamma, \mathrm{e}_i) & & \forall\ i \in \overline{n} \\
 & & & & & \qquad \Delta_{\mathrm{m}}(\chi, \mathrm{C}_0, m) = \overline{\mathrm{C}'}_n \to \mathrm{D} & & \\
 & & & & & \qquad \mathcal{S}_0 = UnifyEnv\ \Pi_0\ \Pi_1 & & \\
 & & & & & \qquad \mathcal{S}_i = (UnifyEnv\ (\mathcal{S}_{i-1}\ \Pi_i)\ (\mathcal{S}_{i-1}\ \Pi_{i+1})) \circ \mathcal{S}_{i-1} & & 1 \leq i < n \\
 & & & & & \qquad \mathcal{S}_n = (unify\ (\mathcal{S}_{n-1}\,\beta)\ (\mathcal{S}_{n-1}\ \langle m : \varphi_0{::}\overline{\varphi}_n \to \varphi\rangle)) \circ \mathcal{S}_{n-1} & & \varphi, \varphi_0, \overline{\varphi}_n \text{ fresh} \\
 & & & & & \qquad \mathcal{S}'_0 = (unify\ (\mathcal{S}_n\ \lfloor\beta\rfloor)\ (\mathcal{S}_n\,\varphi_0)) \circ \mathcal{S}_n & & \\
 & & & & & \qquad \mathcal{S}'_i = (unify\ (\mathcal{S}'_{i-1}\ \alpha_i)\ (\mathcal{S}'_{i-1}\ \varphi_i)) \circ \mathcal{S}'_{i-1} & & \forall\ i \in \overline{n} \\
 & & & & & \qquad \mathcal{S} = (unify\ (\mathcal{S}'_n\ \varphi)\ FreshPredicate_0(\chi, \mathrm{D})) \circ \mathcal{S}'_n & &
\end{array}$$

$$\begin{array}{llllllll}
\mathcal{R}_1 & \chi & \Gamma & \mathtt{new}\ \mathrm{C}(\overline{\mathrm{e}}_n) & = & (\Sigma_{\overline{n}}\ (\mathcal{S}_n\ \Pi_i),\ \mathrm{C} : (\mathcal{S}_n\ \langle f_i : \beta_i\ ^{i \,\in\, \overline{n}}\rangle) \sqcup \langle m_i : \alpha^i_0{::}\overline{\alpha^i}_{n_i} \to \alpha_i\ ^{i \,\in\, \overline{n'}}\rangle) & & \\
 & & & & & \qquad\qquad\qquad\qquad\qquad \text{if} \quad \mathrm{C}_i <:_\chi \mathrm{C}'_i\ \forall\ i \in \overline{n}, \mathrm{C}''_j <:_\chi \mathrm{C}^j\ \forall\ j \in \overline{n'} & & \\
 & & & & & \text{where} \quad \mathcal{F}(\chi, \mathrm{C}) = \overline{f}_n, \mathcal{M}(\chi, \mathrm{C}) = \overline{m}_{n'} & & \\
 & & & & & \qquad \Delta_{\mathrm{f}}(\chi, \mathrm{C}, f_i) = \mathrm{C}'_i & & \forall i \in \overline{n} \\
 & & & & & \qquad (\Pi_i, \mathrm{C}_i : \beta_i) = Inf_1\ (\chi, \Gamma, \mathrm{e}_i) & & \forall i \in \overline{n} \\
 & & & & & \qquad \mathcal{S}_1 = Id_{\mathcal{S}} & & \\
 & & & & & \qquad \mathcal{S}_i = (UnifyEnv\ (\mathcal{S}_{i-1}\ \Pi_{i-1})\ (\mathcal{S}_{i-1}\ \Pi_i)) \circ \mathcal{S}_{i-1} & & 1 < i \leq n \\
 & & & & & \qquad \Delta_{\mathrm{m}}(\chi, \mathrm{D}, m_i) = \overline{\mathrm{C}^i}_{n_i} \to \mathrm{C}^i & & \forall\ i \in \overline{n'} \\
 & & & & & \qquad \textsc{mbody}(\chi, \mathrm{D}, m_i) = (\overline{x}_{n_i}, \mathrm{e}'_i) & & \forall\ i \in \overline{n'} \\
 & & & & & \qquad \Gamma_i = \{x^i_1 : \mathrm{C}^i_1, \ldots, x^i_{n_i} : \mathrm{C}^i_{n_i}, \mathtt{this} : \mathrm{D}\} & & \forall\ i \in \overline{n'} \\
 & & & & & \qquad (\Pi'_i,\ \mathrm{C}''_i : \alpha_i) = \mathcal{R}_0\ (\chi, \Gamma_i, \mathrm{e}'_i) & & \forall\ i \in \overline{n'} \\
 & & & & & \text{such that} \quad x^i_1 : \mathrm{C}^i_1 : \alpha^i_1, \ldots, x^i_{n_1} : \mathrm{C}^i_{n_1} : \alpha^i_{n_1}, \mathtt{this} : \mathrm{D} : \alpha^i_0 \in \Pi'_i & & \forall\ i \in \overline{n'}
\end{array}$$

As for type inference and Rank-0 predicate inference, we now turn our attention to the termination of the Rank-1 predicate inference algorithm. Firstly, notice that the extensions to the substitution, unification, matching and merging operations are such that they are still defined recursively over the structure of (Rank-1) predicates with each recursive call reducing the size of the predicate. It therefore remains the case that they are terminating over finite predicates. This argument also applies to the newly defined *flattening* operation. The $\mathit{FreshPredicate}_1$ function is extended to create Rank-1 predicates (i.e. predicates with member statements about methods as well as fields), and so is defined using the method list look-up function. However, in the same way that the field list look-up function is terminating (see the discussion in §6.2), we can see that the method list look-up function is also terminating. Lastly, a similar argument holds for the *method body* look-up function, MBODY, which is used in the case for method invocation in the main algorithm.

Again, we make the proposition that Rank-1 predicate inference is *sound*, and also that it is *complete* with respect to predicate assignment using only Rank-1 predicates:

Conjecture 6.3.9 (SOUNDNESS OF RANK-1 PREDICATE INFERENCE)**.** If the Rank-1 inference algorithm returns a predicate environment and a type-predicate pair for a given execution context, type environment and expression, then there is a predicate derivation assigning the type-predicate pair to the expression using the execution context and predicate environment:

$$\chi \vdash \Gamma \;\&\; \mathit{Inf}_1\,(\chi, \Gamma, \mathrm{e}) = (\Pi,\ \mathrm{C} : \phi) \Rightarrow \Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \phi$$

Conjecture 6.3.10 (COMPLETENESS OF RANK-1 PREDICATE INFERENCE)**.** If $\mathcal{D}{::}\Pi \Vdash^{\mathrm{P}}_{\chi} \mathrm{e} : \mathrm{C} : \beta$ such that $\mathcal{D}$ contains only Rank-1 predicates, then there exists a Rank-1 predicate β' and a predicate substitution $\mathcal{S}$ such that $\mathcal{R}_1\,(\chi, \widehat{\Pi}, \mathrm{e}) = (\Pi', \beta')$ and $\mathcal{S}\,\Pi' \trianglelefteqslant \Pi$ with $\mathcal{S}\,\beta' \trianglelefteqslant \beta$

Chapter 7

Conclusions and Future Work

We have developed a formal model of a class-based object oriented programming language, inspired by other similar calculi. The motivation for doing this was to demonstrate that an intersection type assignment system can be applied to the class-based flavour of the object oriented paradigm, as well to the object-based variety. We have indeed shown that this is possible by taking the predicate system of van Bakel and de'Liguoro, as described in [5], and modifying it to apply to our calculus, LJ. We have demonstrated the success of this approach by proving subject reduction and expansion theorems.

Our calculus was inspired by two previous efforts: Featherweight Java and Middleweight Java. It incorporates features from both systems, although it is closer in nature to the former. Both Featherweight Java and our calculus, LJ, are functional in nature, while MJ describes a number of imperative features. This makes our calculus a great deal less complex that MJ, however it does mean that we have not been able to investigate how the intersection type system interacts with imperative features. We note that this is not a feature of the system in [5] either.

We feel that LJ, even though it is very similar to Featherweight Java, is more elegant than that calculus. We have chosen to omit casts from our system, and in doing so have avoided complications in proving subject reduction. Additionally, the reduction of LJ cannot become stuck with 'class cast exceptions' (stupid casts) in the way that FJ expressions can. We have also added null objects and field assignment capabilities to our calculus with what we feel is very little additional complexity.

We have also examined the characterisation capabilities of the predicate system that we have defined for LJ. We have seen that convergent expressions are characterised by the assignability of non-trivial predicates, and we have also seen how expressions that can be assigned non-trivial predicates will not result in 'null reference exceptions'. Although not rigorously proved, we hope the reader is convinced that such properties are likely to hold, given the similarities between our system, and the intersection type systems for the Lambda Calculus. Finally, we have defined two restrictions to the predicate assignment system that we assert are decidable.

There are many directions that future research in this area could go. A number of extensions are immediately obvious. Firstly the soundness and completeness of the Rank-0 and Rank-1 predicate inference algorithms remain to be proven. A general definition for a Rank-n restriction would also be useful. Another possible avenue of investigation that would further cement the theoretical foundations of the class-based object oriented paradigm is to define an encoding of the Lambda Calculus in LJ. This has been done for the ς-calculus in [1], and would demonstrate the expressive power and equivalence of LJ with the λ-calculus. On a broader note, semantic models for LJ, as well as other class-based calculi, could be developed. [5] addresses this issue for the ς-calculus, so it seems likely that a similar approach could be taken for LJ.

We have discussed above how LJ lacks imperative features, such as the ones expressed in MJ. A further step might be to add these features to LJ, and also extend the predicate system to handle them. It would be interesting to see if they can easily subsumed into the predicate system, or whether the presence of side-effects will necessitate more drastic changes. Taking another lead from [8], we could also incorporate an effects system into our calculus. Again, one would hope that this extension would dovetail easily with the predicate system.

Bibliography

[1] M. Abadi and L. Cardelli. *A Theory Of Objects*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.

[2] S. van Bakel. Intersection Type Assignment Systems. *Theoretical Computer Science*, 151(2):385–435, 1995.

[3] S. van Bakel. Rank 2 Intersection Type Assignment in Term Rewriting Systems. *Fundamenta Informaticae*, 2(26):141–166, 1996.

[4] S. van Bakel. Cut-Elimination in the Strict Intersection Type Assignment System is Strongly Normalising. *Notre Dame Journal of Formal Logic*, 45(1):35–63, 2004.

[5] S. van Bakel and U. de'Liguoro. Logical Equivalence for Subtyping Object and Recursive Types. *Theory of Computing Systems*, 42(3):306–348, 2008.

[6] H. Barendregt. *The Lambda Calculus, Its Syntax and Semantics*. North-Holland, Amsterdam, 1981.

[7] H. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A Filter Lambda Model and the Completeness of Type Assignment. *Journal of Symbolic Logic*, 48(4):931–940, 1983.

[8] G. Bierman, M. J. Parkinson, and A. Pitts. MJ: An Imperative Core Calculus for Java and Java with Effects. Technical Report 563, University of Cambridge Computer Laboratory, 15 JJ Thompson Ave., Cambridge, CB3 0FD, UK, April 2003.

[9] A. Church and J. B. Rosser. Some Properties of Conversion. *Transactions of the American Mathematical Society*, 39:472–482, 1936.

[10] M. Coppo and M. Dezani-Ciancaglini. An Extension of the Basic Functionality Theory for the λ-Calculus. *Notre Dame, Journal of Formal Logic*, 21(4):685–693, 1980.

[11] M. Coppo, M. Dezani-Ciancaglini, and B. Venneri. Functional Characters of Solvable Terms. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 27:45–58, 1981.

[12] H. B. Curry and R. Feys. *Combinatory Logic*, volume 1. North-Holland, Amsterdam, 1958.

[13] O-J. Dahl and K. Nygaard. SIMULA: an ALGOL-based Simulation Language. *Commun. ACM*, 9(9):671–678, 1966.

[14] S Drossopoulou and S Eisenbach. Is The Java Type System Sound. In *In Proceedings of the Fourth International Workshop on Foundations of Object-Oriented Languages*, 1997.

[15] M. Flatt, S. Krishnamurthi, and M. Felleisen. Classes and Mixins. In *In Principles of Programming Languages (POPL*, pages 171–183. ACM Press, 1998.

[16] A. Goldberg and D. Robson. *Smalltalk-80: The Language and its Implementation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1983.

[17] J. Gosling, B. Joy, G. Steele, and G. Bracha. *The Java Language Specification (3rd Edition)*. Prentice Hall, 2005.

[18] A. Igarashi, B. Pierce, and P. Wadler. Featherweight Java: A Minimal Core Calculus for Java and GJ. In *ACM SIGPLAN Conference on Object Oriented Programming: Systems, Languages, and Applications (OOPSLA)*, October 1999. Full version in ACM Transactions on Programming Languages and Systems (TOPLAS), 23(3), May 2001.

[19] R. Milner, M. Tofte, R. Harper, and D. MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, 1997.

[20] T. Nipkow and D. von Oheimb. Java$_{\text{light}}$ Is Type-Safe—Definitely. In *POPL '98: Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 161–170, New York, NY, USA, 1998. ACM.

[21] Microsoft Press. *C# Language Specifications*. Microsoft Press, U.S., May 2001.

[22] J. A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *J. ACM*, 12(1):23–41, 1965.

[23] B. Stroustrup. *The C++ Programming Language, Second Edition*. Addison-Wesley, 1991.

[24] A. M. Turing. Computability and Lambda-Definability. *J. Symb. Log.*, 2(4):153–163, 1937.