Reasoning about Imperative and Higher-Order Programs

A dissertation presented
by

Vasileios Koutavas

to the Faculty of the Graduate School
of the College of Computer and Information Science
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Northeastern University
Boston, Massachusetts

September, 2008

Abstract

Contextual equivalence, namely the property that two expressions are indistinguishable inside any program context, is a fundamental property of program expressions. Discovering methods that enable formal reasoning about contextual equivalence is hard and highly dependent on the features of the programming language.

In this dissertation we present a technique for systematically deriving reasoning methods for contextual equivalence, which are sound and complete in a variety of languages, but also useful for proving many equivalences. The advantages of the derived reasoning methods are that they successfully deal with imperative as well as higher-order features.

We demonstrate our technique by deriving sound and complete methods for proving contextual equivalence in the call-by-value lambda calculus, a lambda calculus with higher-order store, the nu-calculus, an imperative object calculus, and an imperative core of Java.

Acknowledgments

I would like to thank my advisor, Mitchell Wand, for his guidance, inspiration, and support during my years in Northeastern. His instruction was invaluable in developing my skills as a researcher and contributing in the field of programming languages. His comments on this dissertation greatly improved its quality in technical content and style.

I would also like to express my appreciation to the rest of my thesis committee: Matthias Felleisen, Riccardo Pucella, and Radha Jagadeesan. Their useful comments and suggestions revealed connections between this work and others, and interesting directions of future work. They also contributed considerably in improving the quality of this dissertation.

I'm grateful to professors, friends, and colleagues of the Programming Resarch Lab at Northeastern University for creating a stimulating atmosphere for research and exposing me through classes, seminars, and discussions to many interesting aspects of our field.

My work has greatly benefited from working and discussing with many colleagues, among which are Nick Benton (with whom I collaborated during and after my intership in Microsoft Research Cambridge), Eijiro Sumii, Soren Lassen, Dimitrios Vytiniotis, Davide Sangiorgi, Andrew Kennedy, Lars Birkedal, Derek Dreyer, and Amal Ahmed.

I'm also grateful to my dear friends and colleagues in Boston: Christos Dimoulas, Evangelos Kanoulas, Therapon Skotiniotis, and my roomate Dimitrios Vardoulakis. They were always there for me, in good and bad times, always encouraging, and a joyful company.

Last but certainly not least, I would like to thank my family in Greece and New York for their support. Especially I thank my mother Efterpi, my sister Ioulia, and my brother Nikos, who always believed so much in me, supported me and advised me, even though they were thousands of miles away. Most of all, I'm grateful to my late father Marinos, who was always an inspiration to me and the first to believe that I could pursue and successfully complete a doctorate.

Contents

List of Figures

Main Notations

In this dissertation we use the following notations:

Sequences We use an overbar as a syntactic meta-operator to denote a comma-separated sequence of syntax fragments. For example we write $e[\overline{v/x}^{\,n}]$ for $e[v_1/x_1, \ldots, v_n/x_n]$, and $\overline{(u, u')}^{\,n} \in R$ for $(u_1, u'_1), \ldots, (u_n, u'_n) \in R$. We omit the length of sequences (e.g., writing $e[\overline{v/x}]$ and $\overline{(u, u')} \in R$) when it is implicit from the context or arbitrary.

Free Variables The meta-identifiers Φ and Ψ range over finite sets of variables. The (finite) set of free variables of an expression e is given by $fv(e)$. We write $\Phi \vdash e$ to mean $fv(e) \subseteq \Phi$, and $\vdash e$ to mean $\emptyset \vdash e$. We also write $\Phi \vdash \overline{e}$ to mean $\Phi \vdash e_1, \ldots, \Phi \vdash e_n$, for some n.

Typing Environments A *typing environment* is a mapping from identifiers to types of the language. Γ ranges over typing environments.

Contexts A *context* $C[\cdot]$ is an expression C of the language containing a hole $[\cdot]$. The variable-capturing substitution of expression e for the hole inside the context $C[\cdot]$ is written as $C[e]$.

For untyped languages we write $C[\cdot]_\Phi$ when at least the variables in Φ are in scope at the position of the hole of C.

For typed languages we write $C[\cdot]^T_\Gamma$ when only expressions of type T (or subtype of T) can be placed in the hole of C, and Γ is a part of the typing environment at the place of the hole.

Binary Relations The meta-identifiers R and Q range over binary relations on expressions.

For untyped languages we write $\Phi \vdash e\ R\ e'$ when $\Phi \vdash e, e'$ and $(e, e') \in R$; we write $\vdash e\ R\ e'$ when $\vdash e, e'$ and $(e, e') \in R$. We also write $\Phi \vdash \overline{e}\ R\ \overline{e'}$ to mean $\Phi \vdash e_1\ R\ e'_1, \ldots, \Phi \vdash e_n\ R\ e'_n$, for some n.

R^{T} denotes the inverse of relation R; i.e. $\Phi \vdash e'\ R^{\mathsf{T}}\ e$ if and only if $\Phi \vdash e\ R\ e'$.

For typed languages we write $\Sigma, \Sigma'; \Gamma \vdash e\ R\ e' : T$ when $(e, e') \in R$, e has type T under the typing environment Γ and store typing Σ, and e' has type T under the typing environment Γ and store typing Σ'. When only one store typing is enough for typing e and e' we write $\Sigma; \Gamma \vdash e\ R\ e' : T$.

Similarly to the case of untyped relations, R^{T} is the relation such that $\Sigma', \Sigma; \Gamma \vdash e'\ R^{\mathsf{T}}\ e : T$ if and only if $\Sigma, \Sigma'; \Gamma \vdash e\ R\ e' : T$.

Lexicographic Ordering If A and B are partially ordered sets (posets) with orderings $<_A$ and $<_B$, respectively, then their cartesian product $A \times B$ is a poset with a *lexicographic order* defined as:

$$(a_1, b_1) \prec_{\text{lex}} (a_2, b_2) \quad \Longleftrightarrow \quad (a_1 <_A a_2) \vee ((a_1 = a_2) \wedge (b_1 <_B b_2))$$

If A and B are totally ordered sets, then $A \times B$ is totally ordered, as well. In this dissertation we use lexicographic orders as the measure in lexicographic inductions.

CHAPTER 1

Introduction

Contextual equivalence [38] is a fundamental property of program expressions. It formalizes the notion that expressions have identical behavior if and only if they are indistinguishable inside any program context.

Formal proofs of contextual equivalence are useful for proving software correctness in several situations. Only by such a proof can one be certain that a change in the code of a program module that is potentially used by an unbounded number of programs will not break the behavior of those programs. Moreover, contextual equivalence can be used to show the correctness of program transformations performed by compilers, as well as the correctness of algorithms that solve a particular problem (e.g., sorting) by proving its implementation equivalent to the implementation of a known, prototype algorithm that solves the same problem.[1] A benefit of proving software correctness via contextual equivalence is that it avoids the hard task of writing down a formal specification of the desired behavior, and that the prototype is executable.

Nevertheless, reasoning about contextual equivalence is not an easy task. The definition of contextual equivalence does not offer much help in proofs of equivalence, thus auxiliary proof methods are necessary. Moreover, it is hard to find universal methods that work equally well in different languages.

[1] Other methods proposed in the literature for proving an implementation correct include the use of expressive type systems (e.g., [39]), denotational semantics (e.g., [62, 36]), or appropriate logics (e.g., [22, 44, 23]).

In this dissertation we give a technique for *producing* methods of proving contextual equivalence in sequential languages with higher-order and imperative features. These include pure functional, higher-order languages, functional languages with imperative effects, object-based languages, and class-based languages.

The main advantage of our technique is that the methods derived by it are not only sound and complete with respect to contextual equivalence, but also useful in proving many equivalences for the languages to which it has been applied [10, 28, 29, 30]. By using these methods it is possible to prove known equivalences, among which the ones given by Meyer and Sieber [35] involving higher-order and imperative procedures, where previous reasoning methods are harder to use [11, 41, 56].

As said above, the technique is not merely applicable to a single language, but a range of languages. The technique is successful in deriving reasoning principles about contextual equivalence in the λ-calculus, an untyped imperative λ-calculus, one of Abadi and Cardelli's imperative object calculus [1], a core Java calculus, and Pitt and Stark's ν-calculus [53]. Reasoning in these languages has a common source of complications, namely the imperative and higher-order features. Each language, though, has unique features that introduce extra complications in reasoning (e.g., inheritance). The methods derived from the proposed technique successfully deal with all these complications and are sound and complete with respect to contextual equivalence.

The reasoning methods produced by this technique follow from, and improve upon, Sumii and Pierce's bisimulation proof method [57, 58]. By following this approach we were able to make use of the idea of Kripke-style relations for reasoning about equivalence in imperative languages.

Sumii and Pierce's method breaks down bisimulations into parts. Each part corresponds to a "world", representing the conditions of knowledge, or the state, in which it holds. Similar ideas have previously been used

in process calculi (e.g., [15, 40]) and suggested for imperative languages [32]. Their method, as presented, has limited applicability when dealing with some equivalences of higher-order procedures [35]. In [56], though, they discuss a variation of their method that better deals with higher-order procedures, and is similar to the ones derived by the technique described in this dissertation. They dismiss it, though, because it is hard to show closure under unions for the bisimulations. We overcome this problem by proving completeness of the derived methods without using closure under unions.[2]

The reasoning methods derived by our technique use Kripke-style relations to encode the reasoning invariants that hold in the store of imperative languages. The benefits of the derived methods are the following:

- The derived methods are intuitive: They are close to the programmer's intuition, by encoding the invariant that holds in the equivalence of two expressions. Many of the conditions these relations resemble the tests one would write to test the two implementations (e.g., applying related procedures to related values). The rest are closure conditions; they represent the possible setups of the store that one would think to test the implementations with.

- The derived methods admit constructive proofs: To prove an equivalence one needs to construct a relation between the important intermediate states of related computations. These are states that satisfy the necessary invariant of the equivalence. Thus, there is no need for developing extra technical machinery (such as the Principle of Local Invariants in [41]) to facilitate proofs of equivalence.

- The construction of each relation that proves an equivalence can be partially automated: Invariants need the programmer's intuition and are hard to come up with automatically. Nonetheless, large portions

[2] Since the theory of our relations is not based on coinduction and closure under unions, we choose to avoid the term "bisimulation".

of the relations are constructed to satisfy trivial closure conditions, something that can be done automatically. We use an *up-to context* technique [45, 46, 51] to account for large parts of the relations, and in Chapter 5, where the benefit is significant, an up-to store technique to account for parts of the related stores.

- In many proofs of equivalence, reasoning about the behavior of the context is effortless: the conditions that test related computations (e.g., applications of related procedures to related arguments) include an *induction hypothesis* that can be used to immediately reason about smaller related computations, including computations that "leak" from the context into the procedures. Such computations are the applications of higher-order arguments in a functional language (Chapters 2 and 3), or invocations of callbacks in an object-oriented language (Chapters 5 and 6).

- The derived reasoning methods are useful for reasoning about the equivalence of imperative, higher-order procedures [35] where previous operational techniques had limited applicability [11, 41, 56].

The last three points are the main improvements of the derived reasoning methods over Sumii and Pierce's bisimulation method.

The technique described in this dissertation for deriving such reasoning methods consists of the following steps that need to be followed *once for each language*.

1. First, the technique requires us to define a notion of *adequacy*—a restatement of contextual equivalence, formulated as a property of the appropriate kind of relations for the language we reason about. Adequacy must be shown to be sound and complete with respect to contextual equivalence, but, unlike contextual equivalence, admits inductive proofs.

2. We then find necessary and sufficient conditions for a relation to be adequate, which give a guide for writing such relations and simplify proofs of adequacy. These conditions constitute the derived method for proving equivalences in a particular language.

   Finding such conditions for an arbitrary language is generally a non-trivial task. Our technique simplifies this task by attempting an inductive proof of adequacy, parameterized over the relation we want to prove adequate. This general proof is being carried out until the points where the actual contents of the relation are needed to continue the proof. At these points it is relatively easy to find necessary and sufficient conditions on the structure of the relation to continue the proof.

We give more detail and intuition about our technique in Chapter 2 where we derive a proof method of equivalence for a simple language: the call-by-value λ-calculus. In Chapter 3 we adapt our technique for an imperative λ-calculus. In Chapter 4 we develop the technique for the ν-calculus, and show how it can be used to prove the equivalences in Stark's dissertation [53]. In Chapter 5 we develop the technique for Abadi and Cardelli's imperative object language [1] and demonstrate the use of an up-to store technique for simplifying the relations. In Chapter 6 we study our technique for three Java-like calculi, each augmenting the previous with extra features and show how the resulting reasoning methods are affected. The final chapter discusses the related work and gives directions of future work.

CHAPTER 2

Lambda Calculus

This chapter illustrates our technique for deriving a sound and complete method for reasoning about contextual equivalence in a simple setting: the call-by-value (CBV) λ-calculus.

2.1 The Language

We use the syntax, big-step operational semantics, and beta substitution for the CBV λ-calculus shown in Figure 2.1; alpha-equivalent expressions are identified. Each big-step evaluation $e \Downarrow^k w$ consists of an initial closed expression e and a final value w. The natural number k is an upper bound on the number of derivations contained in the evaluation tree. We write $e \Downarrow w$ to mean that there exists k such that $e \Downarrow^k w$. This measure of big-step evaluation trees corresponds to the length of reductions in a small-step semantics, and it will be a stronger measure for the induction principle that we define in the following sections than the height of the evaluation trees.[1] We write $e \Downarrow$ when there exists a value w such that $e \Downarrow w$, and $e \Uparrow$ when no such value exists.

[1] This measure is necessary for proving the example in Section 2.7.2. All other examples in this dissertation can also be proved by taking the height of the evaluation trees as a measure.

$$
\begin{array}{rll}
e, d ::= x \mid \lambda x.\, e \mid (e\ e) & & \text{Expressions} \\
u, v, w ::= \lambda x.\, e & & \text{Values}
\end{array}
$$

$\boxed{e[v/x]}$

$$
\begin{array}{rll}
x[v/x] = v & & \\
y[v/x] = y & & (x \neq y) \\
(e_1\ e_2)[v/x] = (e_1[v/x]\ \ e_2[v/x]) & & \\
(\lambda y.\, e)[v/x] = \lambda y.\, (e[v/x]) & & (x \neq y, y \notin fv(v))
\end{array}
$$

$\boxed{e \Downarrow w}$

$$
\frac{k > 0}{w \Downarrow^{k} w} \qquad \frac{e_1 \Downarrow^{k_1} \lambda x.\, e_3 \quad e_2 \Downarrow^{k_2} v_2 \quad e_3[v_2/x] \Downarrow^{k_3} w}{(e_1\ e_2) \Downarrow^{1+k_1+k_2+k_3} w}
$$

Figure 2.1: Syntax, beta substitution, and operational semantics of the CBV λ-calculus

2.2 Contextual Equivalence

We want a method for proving two expressions of the CBV λ-calculus equivalent, according to the standard definition of contextual equivalence.

Definition 2.2.1 (Standard Contextual Equivalence ($\equiv$))**.** *Two expressions e and e', with $\Phi \vdash e, e'$, are* contextually equivalent*, and we write $\Phi \vdash e \equiv e'$, if and only if for all contexts $\vdash C[\cdot]_\Phi$,*

$$C[e]\Downarrow \iff C[e']\Downarrow$$

Using this definition directly as a basis for a proof by induction of a non-trivial equivalence fails for a number of reasons:

- *Open expressions.* Binding of free variables in e and e' by the context discriminates alpha-equivalent contexts, which complicates the infinite quantification over them.

$\boxed{\Phi \vdash e \preccurlyeq e'}$

$$\frac{x \in \Phi}{\Phi \vdash x \preccurlyeq x} \qquad \frac{\Phi \uplus \{x\} \vdash e \preccurlyeq e'}{\Phi \vdash \lambda x.e \preccurlyeq \lambda x.e'} \qquad \frac{\Phi \vdash e_1 \preccurlyeq e_1' \qquad \Phi \vdash e_2 \preccurlyeq e_2'}{\Phi \vdash (e_1\ e_2) \preccurlyeq (e_1'\ e_2')}$$

$$\frac{\Phi \uplus \{x\} \vdash e \preccurlyeq e'}{\Phi \uplus \{x\} \vdash e \preccurlyeq (\lambda x.e'\ x)} \qquad \frac{\Phi \uplus \{x\} \vdash e \preccurlyeq e' \qquad \vdash v \preccurlyeq v'}{\Phi \vdash e[v/x] \preccurlyeq e'[v'/x]}$$

Figure 2.2: Definition of $(\preccurlyeq)$ for the CBV λ-calculus

- *Multiple substitutions.* Evaluation may duplicate related values, thus the inductive principle will have to take into account the substitution of not just one, but an arbitrary number of equivalent values inside identical contexts.
- *Intermediate evaluations.* Any inductive proof involves not only the evaluation of related programs, but also the evaluation of related intermediate expressions and their resulting values.

We overcome the first complication by reasoning only about *closed values*; by the following theorem we extend this reasoning to arbitrary expressions.

Theorem 2.2.2 (Value Restriction)**.** *For any two expressions* $\{\overline{x}\} \vdash e, e'$*,*

$$\{\overline{x}\} \vdash e \equiv e' \iff \vdash \lambda\overline{x}.e \equiv \lambda\overline{x}.e'$$

To prove this theorem we define the relation $(\preccurlyeq)$, shown in Figure 2.2. This relation is the smallest substitutive congruence containing the beta-equivalent expressions $\Phi \uplus \{x\} \vdash e \preccurlyeq (\lambda x.e'\ x)$, for all Φ, x, and e. We will show that expressions related in $(\preccurlyeq)$ co-terminate.

Lemma 2.2.3 (Properties of $(\preccurlyeq)$)**.**

1. *If* $\Phi \vdash e \preccurlyeq e'$ *then* $\Phi \vdash e$ *and* $\Phi \vdash e'$*.*

2. *If* $\Phi \vdash e$ *then* $\Phi \vdash e \preccurlyeq e$.

3. *If* $\{\overline{x}\} \vdash e \preccurlyeq e'$ *and* $\vdash \overline{v} \preccurlyeq \overline{v'}$ *then* $\vdash e[\overline{v/x}] \preccurlyeq e'[\overline{v'/x}]$.

4. *If* $\Phi \vdash e \preccurlyeq e'$ *and* $\Psi \vdash C[\cdot]_\Phi$ *then* $\Psi \vdash C[e] \preccurlyeq C[e']$.

Proof. We prove the first property by induction on the derivation $\Phi \vdash e \preccurlyeq e'$, and the second property by induction on e. We prove the third property by induction on the sequence $\overline{x}$.

We prove the final property by structural induction on C, using the second property and the following induction hypothesis:

$$IH(C) \stackrel{\text{def}}{=} \forall \Psi.\ (\Psi \vdash C[\cdot]_\Phi \implies \Psi \vdash C[e] \preccurlyeq C[e'])$$

□

Expressions in $(\preccurlyeq)$, closed by values in the same relation, co-terminate.

Lemma 2.2.4. *If* $\{\overline{x}\} \vdash e \preccurlyeq e'$*, and* $\vdash \overline{v} \preccurlyeq \overline{v'}$*, then*

$$e[\overline{v/x}] \Downarrow \iff e'[\overline{v'/x}] \Downarrow$$

Proof. We prove this lemma by proving the following two properties:

$$\begin{aligned} &(\{\overline{x}\} \vdash e \preccurlyeq e') \wedge (\vdash \overline{v} \preccurlyeq \overline{v'}) \wedge (e[\overline{v/x}] \Downarrow w) \\ &\implies \exists w'.\ (e'[\overline{v'/x}] \Downarrow w') \wedge (\vdash w \preccurlyeq w') \end{aligned} \tag{2.1}$$

$$\begin{aligned} &(\{\overline{x}\} \vdash e \preccurlyeq e') \wedge (\vdash \overline{v} \preccurlyeq \overline{v'}) \wedge (e'[\overline{v'/x}] \Downarrow w') \\ &\implies \exists w.\ (e[\overline{v/x}] \Downarrow w) \wedge (\vdash w \preccurlyeq w') \end{aligned} \tag{2.2}$$

Here we show only the proof of (2.1); the proof of (2.2) is similar. We proceed by induction on the height of the derivations $e[\overline{v/x}] \Downarrow w$ and $\{\overline{x}\} \vdash e \preccurlyeq e'$, ordered lexicographically. The induction hypothesis is the following:

$$\begin{aligned} IH(k,m) \stackrel{\text{def}}{=} \forall &\overline{x, v, v'}, e, e', w. \\ &(\{\overline{x}\} \vdash e \preccurlyeq^m e')) \wedge (\vdash \overline{v} \preccurlyeq \overline{v'}) \wedge (e[\overline{v/x}] \Downarrow^k w) \\ &\implies \exists w'.\ (e'[\overline{v'/x}] \Downarrow w') \wedge (\vdash w \preccurlyeq w') \end{aligned}$$

We will prove that for all k and m, $IH(k,m)$ holds by proving that for all k and m

$$(\forall j, n.\ (j,n) \prec_{\text{lex}} (k,m) \implies IH(j,n)) \implies IH(k,m)$$

We assume

$$\forall j, n.\ (j,n) \prec_{\text{lex}} (k,m) \implies IH(j,n)$$

and $\{\overline{x}\} \vdash e \preccurlyeq^m e'$, $\vdash \overline{v} \preccurlyeq \overline{v'}$, and $e[\overline{v/x}] \Downarrow^k w$. We proceed by cases on $\{\overline{x}\} \vdash e \preccurlyeq^m e'$.

Case $\dfrac{x_i \in \{\overline{x}\}}{\{\overline{x}\} \vdash x_i \preccurlyeq^m x_i}$

We have $x_i[\overline{v/x}] = v_i \Downarrow^k v_i$; moreover $v_i' \Downarrow v_i'$ and $\vdash v_i \preccurlyeq v_i'$.

Case $\dfrac{\{\overline{x}\} \uplus \{y\} \vdash e_0 \preccurlyeq^{m-1} e_0'}{\{\overline{x}\} \vdash \lambda y.e_0 \preccurlyeq^m \lambda y.e_0'}$

We have $\lambda y.e_0[\overline{v/x}] \Downarrow^k \lambda y.e_0[\overline{v/x}]$; moreover $\lambda y.e_0'[\overline{v'/x}] \Downarrow \lambda y.e_0'[\overline{v'/x}]$, and by Lemma 2.2.3 (3), $\vdash \lambda y.e_0[\overline{v/x}] \preccurlyeq \lambda y.e_0'[\overline{v'/x}]$.

Case $\dfrac{\{\overline{x}\} \vdash e_1 \preccurlyeq^{m_1} e_1' \qquad \{\overline{x}\} \vdash e_2 \preccurlyeq^{m_2} e_2'}{\{\overline{x}\} \vdash (e_1\ e_2) \preccurlyeq^m (e_1'\ e_2')}, \quad m_1 < m, \quad m_2 < m$

We have $(e_1\ e_2)[\overline{v/z}] \Downarrow^k w$, thus for some $\lambda y.e_3$, w_2, w, and $\overline{k}$:

$$e_1[\overline{v/z}] \Downarrow^{k_1} \lambda y.e_3 \qquad e_2[\overline{v/z}] \Downarrow^{k_2} w_2 \qquad e_3[w_2/y] \Downarrow^{k_3} w \qquad \overline{k} < k$$

By $IH(k_1, m_1)$ and $IH(k_2, m_2)$ we get that there exist $\lambda y.e_3'$ and w_2' such that:

$$e_1'[\overline{v'/z}] \Downarrow \lambda y.e_3' \qquad \vdash \lambda y.e_3 \preccurlyeq \lambda y.e_3'$$
$$e_2'[\overline{v'/z}] \Downarrow w_2' \qquad \vdash w_2 \preccurlyeq w_2'$$

By the definition of $(\preccurlyeq)$ we get that $\{y\} \vdash e_3 \preccurlyeq e_3'$, and there exists m_3 such that $\vdash e_3[w_2/y] \preccurlyeq^{m_3} e_3'[w_2'/y]$. Thus, by $IH(k_3, m_3)$, there exists w' such that:

$$e_3'[w_2'/y] \Downarrow w' \qquad \vdash w \preccurlyeq w'$$

and therefore, by the evaluation rule of application, $(e_1'\ e_2')[\overline{v'/z}] \Downarrow w'$.

Case $\dfrac{\{\overline{x}\} \uplus \{y\} \vdash e \preccurlyeq^{m-1} e'}{\{\overline{x}\} \uplus \{y\} \vdash e \preccurlyeq^{m} (\lambda y.e'\ y)}$

We have $e[\overline{v/x}, u/y] \Downarrow^k w$. By $IH(k, m-1)$ we get that there exists w', such that:

$$e'[\overline{v'/x}, u/y] \Downarrow w' \qquad \vdash w \preccurlyeq w'$$

and therefore $(\lambda y.e'\ y)[\overline{v'/x}, u/y] \Downarrow w'$.

Case $\dfrac{\{\overline{x}\} \uplus \{y\} \vdash e \preccurlyeq^{m_1} e' \qquad \vdash u \preccurlyeq^{m_2} u'}{\{\overline{x}\} \vdash e[u/y] \preccurlyeq^{m} e'[u'/y]}, \quad m_1 < m,\ m_2 < m$

We have $e[\overline{v/x}, u/y] \Downarrow^k w$. By $IH(k, m_1)$ we get that there exists w', such that:

$$e'[\overline{v'/x}, u'/y] \Downarrow w' \qquad \vdash w \preccurlyeq w'$$

□

By using $(\preccurlyeq)$ we are able to prove the following lemma.

Lemma 2.2.5. *For any two expressions* $\Phi \uplus \{x\} \vdash e, e'$*,*

$$\Phi \uplus \{x\} \vdash e \equiv e' \iff \Phi \vdash \lambda x.e \equiv \lambda x.e'$$

Proof. The forward direction is easily shown by the definition of $(\equiv)$: for all contexts $\vdash C[\cdot]_{\Phi \uplus \{x\}}$,

$$C[e] \Downarrow \iff C[e'] \Downarrow$$

By taking $C[\cdot]$ to be $d[\lambda x.[\cdot]_{\Phi \uplus \{x\}}]_{\Phi}$, we get:

$$d[\lambda x.[e]] \Downarrow \iff d[\lambda x.[e']] \Downarrow$$

thus, $\Phi \vdash \lambda x.e \equiv \lambda x.e'$.

For the reverse direction we need to show that for all contexts $\vdash C[\cdot]_{\Phi \uplus \{x\}}$,

$$C[e] \Downarrow \iff C[e'] \Downarrow$$

assuming $\Phi \vdash \lambda x.e \equiv \lambda x.e'$.

By the definition of $(\preccurlyeq)$ and Lemma 2.2.3 (4) we have

$$\vdash C[e] \preccurlyeq C[(\lambda x.e\ x)] \tag{2.3}$$

$$\vdash C[e'] \preccurlyeq C[(\lambda x.e'\ x)] \tag{2.4}$$

Hence we have:

$$\begin{array}{ll} & C[e]\Downarrow \\ \textit{iff} & C[(\lambda x.e\ x)]\Downarrow \text{ (by Lemma 2.2.4, and (2.3))} \\ \textit{iff} & C[(\lambda x.e'\ x)]\Downarrow (\Phi \vdash \lambda x.e \equiv \lambda x.e') \\ \textit{iff} & C[e']\Downarrow \qquad \text{(by Lemma 2.2.4, and (2.4))} \end{array}$$

□

The Value Restriction theorem (Theorem 2.2.2) is immediate from Lemma 2.2.5.

We, therefore, consider only relations on closed values for the rest of this chapter. We extend such relations to relations on open expressions as follows.

Definition 2.2.6 (Open Extension)**.** *If R is a binary relation on closed values, then R° is the relation such that $\{\overline{x}\} \vdash e\ R^\circ\ e'$ if and only if $\vdash \lambda\overline{x}.e\ R\ \lambda\overline{x}.e'$.*

2.3 Derivation of Adequacy

2.3.1 Pre-Adequacy

By considering only relations on closed values we overcome the first complication of page 8. We overcome the second complication by allowing contexts to have multiple holes that are filled with multiple (closed) values. We place these values in the holes of the contexts using β-substitution. The following definition gives the relation between identical contexts with R-related values in their holes.

Definition 2.3.1 (Context Closure)**.** *If R is a binary relation on closed values, then R^{cxt} is the relation defined by*

$$\frac{\{\overline{x}\} \vdash d \qquad \vdash \overline{u}\ R\ \overline{u'}}{\vdash d[\overline{u/x}]\ R^{\mathsf{cxt}}\ d[\overline{u'/x}]}$$

Context closure satisfies the following properties.

Lemma 2.3.2 (Properties of $(\ )^{\mathsf{cxt}}$)**.**

1. *If $R \subseteq Q$, then $R^{\mathsf{cxt}} \subseteq Q^{\mathsf{cxt}}$.*
2. *$(R^{\mathsf{cxt}})^{\mathsf{cxt}} = R^{\mathsf{cxt}}$.*
3. *If $R \subseteq Q \subseteq R^{\mathsf{cxt}}$, then $R^{\mathsf{cxt}} = Q^{\mathsf{cxt}}$.*

Proof. By elementary properties of substitution. □

Next, we give a definition of *pre-adequacy*. A relation is pre-adequate when its context closure relates co-terminating programs

Definition 2.3.3 (Pre-Adequate Relations)**.** *A binary relation on closed values, R, is* pre-adequate *if and only if for all expressions $\vdash e\ R^{\mathsf{cxt}}\ e'$*

$$e\Downarrow \iff e'\Downarrow$$

The definition of pre-adequate relations contains a negative occurrence of R. Therefore it is not clear that the generating functional monotone and that the largest pre-adequate relation exists. However, we show that such a relation exists and is the union of all pre-adequate relations.

Lemma 2.3.4. *If the relations R_1 and R_2 are pre-adequate, then $(R_1 \cup R_2)$ is pre-adequate.*

Proof. Let $\vdash e\ (R_1 \cup R_2)^{\mathsf{cxt}}\ e'$. We will show

$$e\Downarrow \iff e'\Downarrow$$

By Definition 2.3.1 there must be some $\{\overline{x}, \overline{y}\} \vdash d$, $\vdash \overline{v}\ R_1\ \overline{v'}$, $\vdash \overline{u}\ R_2\ \overline{u'}$, such that

$$e = d[\overline{v/x}, \overline{u/y}] \qquad e' = d[\overline{v'/x}, \overline{u'/y}]$$

Hence, we have:

$$\begin{aligned}
& e \Downarrow \\
\textit{iff}\ & d[\overline{v/x}, \overline{u/y}] \Downarrow \\
\textit{iff}\ & d[\overline{v'/x}, \overline{u/y}] \Downarrow\ (\vdash d[\overline{v/x}, \overline{u/y}]\ R_1{}^{\mathsf{cxt}}\ d[\overline{v'/x}, \overline{u/y}], R_1 \text{ adequate}) \\
\textit{iff}\ & d[\overline{v'/x}, \overline{u'/y}] \Downarrow\ (\vdash d[\overline{v'/x}, \overline{u/y}]\ R_2{}^{\mathsf{cxt}}\ d[\overline{v'/x}, \overline{u'/y}], R_2 \text{ adequate}) \\
\textit{iff}\ & e' \Downarrow
\end{aligned}$$

□

Definition 2.3.5 (Pre-Adequacy $(\cong)$)**.** *Pre-adequacy is the union of all pre-adequate relations; namely:*

$$(\cong) \stackrel{\text{def}}{=} \bigcup_{R \textit{ pre-adequate}} R$$

Lemma 2.3.6. $(\cong)$ *is pre-adequate.*

Proof. Let $\vdash e\ (\cong)^{\mathsf{cxt}}\ e'$. We will show

$$e \Downarrow \iff e' \Downarrow$$

By Definition 2.3.1 there must be some $x_1, \ldots, x_n \vdash d$, $\vdash \overline{v} \cong \overline{v'}$, such that

$$e = d[\overline{v/x}] \qquad e' = d[\overline{v'/x}]$$

By Definition 2.3.5 there must be pre-adequate relations $R_1, \ldots, R_n$ such that

$$\vdash v_1\ R_1\ v_1' \quad \ldots \quad \vdash v_n\ R_n\ v_n'$$

Therefore

$$\vdash e\ (R_1 \cup \ldots \cup R_n)\ e'$$

and by Lemma 2.3.4 we get

$$e \Downarrow \iff e' \Downarrow$$

□

By the above lemma we conclude that $(\cong)$ is the largest pre-adequate relation. Furthermore, we show that $(\cong^\circ)$ coincides with $(\equiv)$.

Theorem 2.3.7 (Soundness and Completeness of $(\cong)$)**.** $(\cong)^\circ = (\equiv)$

Proof. Let $\overline{x}$ be a non-empty sequence of variables, and $\Phi = \{\overline{x}\}$. Then:

$$
\begin{array}{ll}
\quad \{\overline{x}\} \vdash e \equiv e' & \\
\textit{iff} \ \vdash \lambda\overline{x}.e \equiv \lambda\overline{x}.e' & \text{(by Theorem 2.2.2)} \\
\textit{iff} \ \forall \vdash C[\cdot]_\emptyset.\ C[\lambda\overline{x}.e]\Downarrow \iff C[\lambda\overline{x}.e']\Downarrow & \text{(by Definition 2.2.1)} \\
\textit{iff} \ \forall \{y\} \vdash d. & \\
\qquad ((\lambda y.d)\ [\lambda\overline{x}.e])\Downarrow \iff ((\lambda y.d)\ [\lambda\overline{x}.e'])\Downarrow & \\
\textit{iff} \ \forall \{y\} \vdash d.\ d[\lambda\overline{x}.e/y]\Downarrow \iff d[\lambda\overline{x}.e'/y]\Downarrow &
\end{array}
$$

Let $R = \{(\lambda\overline{x}.e, \lambda\overline{x}.e')\}$. By Definition 2.3.1 and the properties of substitution, the above can be written equivalently:

$$
\begin{array}{ll}
\quad \forall \vdash e_0\ R^{\mathsf{cxt}}\ e_0'.\ e_0\Downarrow \iff e_0'\Downarrow & \\
\textit{iff} \ R \subset (\cong) & \text{(by Definition 2.3.5)} \\
\textit{iff} \ \vdash \lambda\overline{x}.e \cong \lambda\overline{x}.e' & \\
\textit{iff} \ \{\overline{x}\} \vdash e \cong^\circ e' & \text{(by Definition 2.2.6)}
\end{array}
$$

□

2.3.2 Adequacy

Theorem 2.3.7 shows that pre-adequacy is the same as contextual equivalence. However, even by using the definition of pre-adequate relations we still do not get a good induction principle due to the final complication described on page 8. To get a definition that will give us a good induction principle we relate the final values of evaluation in the same relation that relates the initial expressions. The definition we end up with is the following:

Definition 2.3.8 (Adequate Relations)**.** *A binary relation R on closed values is* adequate *if and only if, for all $\vdash e\ R^{\mathsf{cxt}}\ e'$:*

$$\forall w.\ (e \Downarrow w) \implies \exists w'.\ (e' \Downarrow w') \wedge (\vdash w\ R^{\mathsf{cxt}}\ w')$$

and

$$\forall w'.\ (e' \Downarrow w') \implies \exists w.\ (e \Downarrow w) \wedge (\vdash w\ R^{\mathsf{cxt}}\ w')$$

If R is adequate, then R^{cxt} is a *big-step bisimulation*. In other words, the above definition requires R^{cxt} to relate expressions that co-terminate, and to be *closed under evaluation*.

As in the definition of pre-adequate relations, adequacy contains a negative occurrence of R. Here we show the existence of the largest adequate relation through the soundness and completeness of adequate relations with respect to pre-adequacy.

Theorem 2.3.9 (Soundness of Adequate Relations)**.** *If R is adequate then it is also pre-adequate.*

Proof. Trivial by the definitions of adequate and pre-adequate relations. □

Theorem 2.3.10 (Completeness of Adequate Relations)**.** $(\cong)$ *is adequate.*

Proof. Let $\vdash e \cong^{\mathsf{cxt}} e'$. We will show that

$$\forall w.\ (e \Downarrow w) \implies \exists w'.\ (e' \Downarrow w') \wedge (\vdash w \cong^{\mathsf{cxt}} w')$$

and

$$\forall w'.\ (e' \Downarrow w') \implies \exists w.\ (e \Downarrow w) \wedge (\vdash w \cong^{\mathsf{cxt}} w')$$

By the definition of pre-adequate relations (Definition 2.3.3) and the determinacy of the language it suffices to show

$$\forall w, w'.\ (e \Downarrow w) \wedge (e' \Downarrow w') \implies (\vdash w \cong^{\mathsf{cxt}} w')$$

Let $e \Downarrow w$, and $e' \Downarrow w'$; we will show that $\vdash w \cong^{\mathsf{cxt}} w'$.

For all $\overline{x, v, v'}$, y, and d such that $\{\overline{x}, y\} \vdash d$ and $\vdash \overline{v} \cong \overline{v'}$ we have:

$$\begin{array}{rll} & d[\overline{v/x}, w/y] \Downarrow & \\ \iff & (\lambda y.\, d[\overline{v/x}]\ e) \Downarrow & \text{(by the properties of evaluation)} \\ \iff & (\lambda y.\, d[\overline{v'/x}]\ e') \Downarrow & (\vdash (\lambda y.\, d[\overline{v/x}]\ e) \cong^{\mathsf{cxt}} (\lambda y.\, d[\overline{v'/x}]\ e')) \\ \iff & d[\overline{v'/x}, w'/y] \Downarrow & \text{(by the properties of evaluation)} \end{array}$$

Therefore, by Definitions 2.3.3 and 2.3.5, $\vdash w \cong^{\mathsf{cxt}} w'$. □

Definition 2.3.11 (Adequacy)**.** *Adequacy, $(\approx)$, is the union of all adequate relations; namely:*

$$(\approx) \stackrel{\text{def}}{=} \bigcup_{R \text{ adequate}} R$$

Theorem 2.3.12. $(\cong) = (\approx)$.

Proof. By Theorem 2.3.9 we have $(\approx) \subseteq (\cong)$ and by Theorem 2.3.10 we have $(\cong) \subseteq (\approx)$. Thus $(\cong) = (\approx)$. □

Theorem 2.3.13. $(\approx)$ *is adequate.*

Proof. Immediate by Theorems 2.3.10 and 2.3.12. □

Moreover, from the above we conclude that the open extension of adequacy coincides with the standard definition of contextual equivalence.

Theorem 2.3.14. $(\approx^\circ) = (\equiv)$.

Proof. By Theorems 2.3.7 and 2.3.12. □

2.4 Inductive Proofs of Equivalence

We can now prove two expressions $\{\overline{x}\} \vdash e, e'$ contextually equivalent by using the following proof method:

1. Find a binary relation on values R, such that $\vdash \lambda\overline{x}.\, e\ R\ \lambda\overline{x}.\, e'$,

2. show that R is adequate, and

3. invoke Theorem 2.3.14 to show $\{\overline{x}\} \vdash e \equiv e'$.

In this three-step proof method, the first step depends greatly on the second. It requires intuition to identify the necessary expressions that need to be related in R for proving it adequate. The second step can be done by induction; the definition of adequate relations (Definition 2.3.8) gives us a good induction principle for doing such a proof. The third step is immediate.

Here we show an induction principle for showing R adequate. The induction hypothesis that we will use for each direction of Definition 2.3.8 is the following:

Definition 2.4.1.

$$IH_R(k) \stackrel{\text{def}}{=} \forall e, e', w.\ (\vdash e\ R^{\mathsf{cxt}}\ e') \wedge (e \Downarrow^k w) \implies \exists w'.\ (e' \Downarrow w') \wedge (\vdash w\ R^{\mathsf{cxt}}\ w')$$

Proving that some relation R is adequate, as in the second step of the aforementioned proof method, is equivalent to proving the following two properties:

$$\forall k.\ IH_R(k) \qquad \forall k.\ IH_{R^\intercal}(k)$$

For $k = 0$ it is trivial; for the rest, it can be shown by induction:

$$\boxed{\begin{array}{l} \forall k.\ IH_R(k-1) \implies IH_R(k) \\ \forall k.\ IH_{R^\intercal}(k-1) \implies IH_{R^\intercal}(k) \end{array}}$$

These two inductions are *necessary and sufficient proof obligations* for showing a set R adequate. In the next section we show how these can be simplified for the CBV λ-calculus.

2.5 Deriving Smaller Proof Obligations for Adequate Relations

The two inductive proof obligations of the previous section for showing a relation R adequate are mostly standard and independent from R. A large part of these proof obligations is carried out by using just the induction hypothesis. We factor out these parts by considering an arbitrary R and unfolding as much as possible the two inductions. This needs only be done once for the language under consideration (here the CBV λ-calculus). What remain are the parts of the proof that depend on the contents of R. We can think of these parts as smaller proof obligations for showing R adequate.

Here we show the unfolding of the forward proof obligation. The reverse is symmetric. Let $k \in I\!N$; we assume $IH_R(k-1)$. We will show that $IH_R(k)$ holds; i.e.:

$$\begin{aligned} &\forall e, e', w.\ (\vdash e\ R^{\mathsf{cxt}}\ e') \wedge (e \Downarrow^k w) \\ &\quad \Longrightarrow \exists w'.\ (e' \Downarrow w') \wedge (\vdash w\ R^{\mathsf{cxt}}\ w') \end{aligned} \tag{2.5}$$

Let $\vdash e\ R^{\mathsf{cxt}}\ e'$ and $e \Downarrow^k w$. By Definition 2.3.1, there exist d, $\overline{u, x}$ such that:

$$e = d[\overline{u/x}] \qquad e' = d[\overline{u'/x}] \qquad \{\overline{x}\} \vdash d \qquad \vdash \overline{u}\ R\ \overline{u'}$$

We take cases on d. For the cases where $d = x$ and $d = \lambda y.d_1$, (2.5) is trivially satisfied. The only interesting case is when $\{\overline{x}\} \vdash d = (d_1\ d_2)$. We have $(d_1[\overline{u/x}]\ d_2[\overline{u/x}]) \Downarrow^k w$; by the evaluation rule for application we get:

$$d_1[\overline{u/x}] \Downarrow^{k-1} \lambda y.e_1 \tag{2.6}$$

$$d_2[\overline{u/x}] \Downarrow^{k-1} w_2 \tag{2.7}$$

$$e_1[w_2/y] \Downarrow^{k-1} w \tag{2.8}$$

From (2.6), (2.7), and $IH_R(k-1)$, there exist $\lambda y.e_3'$ and w_2' such that:

$$d_1[\overline{u'/x}] \Downarrow \lambda y.e_1' \qquad \vdash \lambda y.e_1\ R^{\mathsf{cxt}}\ \lambda y.e_1'$$

$$d_2[\overline{u'/x}] \Downarrow w_2' \qquad \vdash w_2\ R^{\mathsf{cxt}}\ w_2'$$

We now take cases on $\vdash \lambda y.e_1\ R^{\mathsf{cxt}}\ \lambda y.e_1'$.

Case $e_1 = d_3[\overline{u_3/x}]$, $e_1' = d_3[\overline{u_3'/x}]$, $\{\overline{x}\} \vdash \lambda y.d_3$, and $\vdash \overline{u_3}\ R^{\mathsf{cxt}}\ \overline{u_3'}$.

By (2.8) and $IH_R(k-1)$, there exists w' such that:

$$d_3[\overline{u_3'/x}, w_2'/y] \Downarrow w' \qquad \vdash w\ R\ w'$$

Hence, by the evaluation rule for application, $e' \Downarrow w'$, which concludes this case of the proof.

Case $\vdash \lambda y.e_1\ R\ \lambda y.e_1'$.

We need to show that there exists w' such that

$$e_1'[w_2'/y] \Downarrow w' \qquad \vdash w\ R^{\mathsf{cxt}}\ w'$$

This is a proof obligation that we can not unfold further since it depends on the values related by a particular R. Together with the assumptions at this point of the proof, this proof obligation can be summarized as follows.

$$\begin{array}{l}\forall k, w.\ IH_R(k-1)\\ \quad \wedge\ \vdash \lambda y.e\ R\ \lambda y.e'\\ \quad \wedge\ \vdash v\ R^{\mathsf{cxt}}\ v'\\ \quad \wedge\ (\lambda y.e\ v) \Downarrow^k w\\ \qquad \Longrightarrow\ \exists w'.\ (\lambda y.e'\ v') \Downarrow w'\\ \qquad\qquad \wedge\ \vdash w\ R^{\mathsf{cxt}}\ w'\end{array}$$

This proof obligation (together with its symmetric one) is a necessary and sufficient condition for a particular R to be adequate for the CBV λ-calculus.

Theorem 2.5.1. *A binary relation on values, R, is adequate if and only if for all k, when $IH_R(k-1)$ holds, the following condition is satisfied:*

- *For all $\vdash \lambda y.e\ R\ \lambda y.e'$ and all $\vdash v\ R^{\mathsf{cxt}}\ v'$ and w, such that $(\lambda y.e\ v) \Downarrow^k w$, there exists w' such that*

$$(\lambda y.e'\ v') \Downarrow w' \qquad \vdash w\ R^{\mathsf{cxt}}\ w'$$

and, moreover, the above holds for R^{T}.

Proof. Forward direction: by showing that the two proof obligations of the theorem are special cases of the proof obligations of the definition of adequate relations (Definition 2.3.8).

Reverse direction: by recapitulating the above proof. □

The conditions of the above theorem are simpler and easier to prove that the conditions in the definition of adequate relations (Definition 2.3.8). To prove a relation R adequate using Definition 2.3.8 we need to prove that R^{cxt} is closed under any evaluation of R^{cxt}-related expressions. By using Theorem 2.5.1, though, it suffices to show that R^{cxt} is closed only under applications of procedures related in R to arguments related in R^{cxt}. Moreover, the induction hypothesis makes this proof possible in cases where R contains higher-order procedures.

2.6 A Discussion of Completeness of the Proof Method and Limitations

The proof method at the beginning of Section 2.4 is theoretically sound and complete with respect to contextual equivalence: if one finds a relation that relates the lambda-closed versions of two expressions, and proves it is adequate, then the two expressions are contextually equivalent. Conversely, if two expressions are contextually equivalent, then there is always an adequate relation that relates the corresponding closed values. Theorem 2.3.14 states this soundness and completeness result, and Theorem 2.5.1 extends the result to the smaller proof obligations of the previous section.

In practice, constructing a convenient relation that relates two terms and provably satisfies the proof obligations of Theorem 2.5.1 is often easy and can be done by an iterative approach due to the *induction hypotheses* of the theorem: first we start with a relation R that only relates the values that

we want to show equivalent. Then we attempt to prove the proof obligations for each pair of values in the relation. To prove these conditions for each pair, we may choose to advance the computations involved until reaching a *synchronization point* with fewer computational steps left—a pair of expressions related in R^{cxt} that can be shown equivalent by the induction hypothesis. Finding such a point depends on the values related by R and it may require to add more values in R. Then we prove the proof obligations for the newly added pairs of values and repeat the process until a fixpoint is reached and no more pairs need to be added in the relation. This proof process is sufficient for constructing and proving adequate relations for many equivalences, including some that were known to be hard to prove [35]. This process is similar to proofs of equivalences using bisimulations "up-to", since $(\ )^{\mathsf{cxt}}$ is a form of up-to context closure and the induction hypotheses a form of up-to number of steps.

For some equivalences, though, this iterative proof is not possible. These are equivalences where the induction hypotheses do not apply in any subcomputation, and, thus, adding more pairs of values in the relation does not make the proof easier. In such cases constructing adequate relations is again possible, but in order to prove their adequacy we need to reason about the operational behavior of each expression separately.

In this dissertation we show three such equivalences. The first, in Section 2.7.3 uses entirely external operational reasoning to reason about the behavior of one of the terms and prove the singleton relation containing only the related terms adequate. In Section 3.6.4, we show an equivalence, the proof of which requires the combination of the iterative approach with an induction on the operational behavior of part of the context. In Section 4.6.2 we make a double use of the usual iterative proof method for the main proof of adequacy and for reasoning about the behavior of part of the terms and the context. The rest of the equivalences in this and following chapters are proved solely by a single use of the iterative approach discussed above.

$$\frac{}{\vdash Y_{\text{val}}\ R\ \Theta_{\text{val}}} \qquad \frac{\vdash v\ R^{\mathsf{cxt}}\ v'}{\vdash \lambda z.((\Delta_v\ \Delta_v)\ z)\ R\ \lambda z.(((A\ A)\ v')\ z)}$$

Figure 2.3: Construction of an adequate relation for proving the equivalence of Y_{val} and Θ_{val}

2.7 Examples

2.7.1 Fixpoint Combinators

We use the method described in Section 2.4 to prove the equivalence between the call-by-value versions of Church's and Turing's fixpoint operators ($\vdash Y_{\text{val}} \equiv \Theta_{\text{val}}$).

$$Y_{\text{val}} \stackrel{\text{def}}{=} \lambda f.(\Delta_f\ \Delta_f) \quad \text{where } \Delta_f \stackrel{\text{def}}{=} \lambda x.(f\ \lambda z.((x\ x)\ z))$$
$$\Theta_{\text{val}} \stackrel{\text{def}}{=} \lambda f.((A\ A)\ f) \quad \text{where } A \stackrel{\text{def}}{=} \lambda x.\lambda y.(y\ \lambda z.(((x\ x)\ y)\ z))$$

Proof. We construct an appropriate relation R and prove it adequate iteratively. Initially, we relate only Y_{val} and Θ_{val} in R by the first rule of Figure 2.3. We need to prove that the two symmetric conditions of Theorem 2.5.1 are satisfied for this pair of values. Here we show the proof of the first condition; the proof of the second is symmetric.

Let $\vdash v\ R^{\mathsf{cxt}}\ v'$ and $(Y_{\text{val}}\ v) \Downarrow^k w$. We need to show that for some w' $(\Theta_{\text{val}}\ v') \Downarrow w'$ and $\vdash w\ R^{\mathsf{cxt}}\ w'$. By the properties of evaluation we have:

$$\begin{aligned} & (Y_{\text{val}}\ v) \Downarrow^k w \\ \iff\ & (\Delta_v\ \Delta_v) \Downarrow^{k-1} w \\ \implies\ & (v\ \lambda z.((\Delta_v\ \Delta_v)\ z)) \Downarrow^{k-1} w \end{aligned} \tag{2.9}$$

and

$$\begin{aligned} & (\Theta_{\text{val}}\ v') \Downarrow w' \\ \iff & ((A\ A)\ v') \Downarrow w' \\ \iff & (v'\ \lambda z.(((A\ A)\ v')\ z)) \Downarrow w' \end{aligned} \tag{2.10}$$

Normally, we would not be able to continue this proof further without an induction on the common context of v and v'. Instead, we choose to relate more pairs in R that enable us to invoke the induction hypothesis. Hence, we add to our construction of R the second rule of Figure 2.3. By adding this rule, we do not need to revisit our proof so far; we only need to prove the conditions of adequacy for the newly related values.

R is a well-founded relation even after the addition of the second inference rule. This is because, by Definition 2.3.1, the premise of that rule amounts to:

$$v = d[\overline{u/x}] \qquad v' = d[\overline{u'/x}] \qquad \{\overline{x}\} \vdash d \qquad \vdash \overline{u}\ R\ \overline{u'}$$

Hence, by the second rule of the construction of R we have that:

$$\vdash \lambda z.((\Delta_v\ \Delta_v)\ z)\ R\ \lambda z.(((A\ A)\ v')\ z)$$

and therefore:

$$\frac{\{\overline{x}, y\} \vdash (d\ y) \qquad \vdash \overline{u}\ R\ \overline{u'} \qquad \vdash \lambda z.((\Delta_v\ \Delta_v)\ z)\ R\ \lambda z.(((A\ A)\ v')\ z)}{\vdash (v\ \lambda z.((\Delta_v\ \Delta_v)\ z))\ R^{\mathsf{cxt}}\ (v'\ \lambda z.(((A\ A)\ v')\ z))}$$

By the above, $IH_R(k-1)$, and (2.9) we get

$$(v'\ \lambda z.(((A\ A)\ v')\ z)) \Downarrow w' \qquad \vdash w\ R^{\mathsf{cxt}}\ w'$$

and, therefore, by (2.10) we get

$$(\Theta_{\text{val}}\ v') \Downarrow w' \qquad \vdash w\ R^{\mathsf{cxt}}\ w'$$

which concludes this part of the proof.

To finish the proof we need to prove that the conditions of Theorem 2.5.1 hold for the pairs of values added in the relation by the second rule of Figure 2.3. Again, we show only the proof of the first condition.

Let $\vdash \lambda z.((\Delta_v\ \Delta_v)\ z)\ R\ \lambda z.(((A\ A)\ v')\ z)$, with $\vdash v\ R^{\mathsf{cxt}}\ v'$, and let $\vdash u\ R^{\mathsf{cxt}}\ u'$ and $(\lambda z.((\Delta_v\ \Delta_v)\ z)\ u) \Downarrow^k w$. We need to show that, for some w', $(\lambda z.(((A\ A)\ v')\ z)\ u') \Downarrow w'$ and $\vdash w\ R^{\mathsf{cxt}}\ w'$. By the properties of evaluation we have

$$\begin{aligned} & (\lambda z.((\Delta_v\ \Delta_v)\ z)\ u) \Downarrow^k w \\ \iff & ((\Delta_v\ \Delta_v)\ u) \Downarrow^{k-1} w \\ \implies & ((v\ \lambda z.((\Delta_v\ \Delta_v)\ z))\ u) \Downarrow^{k-1} w \end{aligned} \tag{2.11}$$

and

$$\begin{aligned} & (\lambda z.(((A\ A)\ v')\ z)\ u') \Downarrow w' \\ \iff & (((A\ A)\ v')\ u') \Downarrow w' \\ \iff & ((v'\ \lambda z.(((A\ A)\ v')\ z))\ u') \Downarrow w' \end{aligned} \tag{2.12}$$

We also have that

$$\vdash ((v\ \lambda z.((\Delta_v\ \Delta_v)\ z))\ u)\ R^{\mathsf{cxt}}\ ((v'\ \lambda z.(((A\ A)\ v')\ z))\ u')$$

By the above, $IH_R(k-1)$, and (2.11) we get:

$$((v'\ \lambda z.(((A\ A)\ v')\ z))\ u') \Downarrow w' \qquad \vdash w\ R^{\mathsf{cxt}}\ w'$$

and thus, by (2.11):

$$(\lambda z.(((A\ A)\ v')\ z)\ u') \Downarrow w' \qquad \vdash w\ R^{\mathsf{cxt}}\ w'$$

This concludes the case of the proof for values added in the relation by the second rule of Figure 2.3. Since there are no more values added in the relation, the proof that R is an adequate relation is finished. By Theorem 2.3.14 we get that $\vdash Y_{\text{val}} \equiv \Theta_{\text{val}}$. □

2.7.2 Tail Recursion

We prove the equivalence of two recursive procedures, one that uses tail recursion one that does not, borrowed from [50]. For this example we consider the straightforward extension of the language with integer constants, operators on constants, and the following conditional expression:

$$\texttt{ifzero}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3$$

The evaluation rules for this expression are:

$$\frac{e_1 \Downarrow^{k_1} \mathsf{0} \qquad e_2 \Downarrow^{k_2} w}{\texttt{ifzero}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \Downarrow^{1+k_1+k_2} w}$$

$$\frac{e_1 \Downarrow^{k_1} v \qquad v \neq \mathsf{0} \qquad e_3 \Downarrow^{k_3} w}{\texttt{ifzero}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \Downarrow^{1+k_1+k_3} w}$$

If e evaluates to zero then the above conditional evaluates to the same value as e_2. If e evaluates to a value different than zero then the conditional evaluates to the same value as e_3.

We consider the following terms:

$$P \stackrel{\text{def}}{=} \lambda f.\lambda g.\lambda x.\lambda y.\texttt{ifzero}\ x\ \texttt{then}\ y\ \texttt{else}\ (g\ (f\ g\ (x-1)\ y))$$

$$P' \stackrel{\text{def}}{=} \lambda f.\lambda g.\lambda x.\lambda y.\texttt{ifzero}\ x\ \texttt{then}\ y\ \texttt{else}\ (f\ g\ (x-1)\ (g\ y))$$

$$F \stackrel{\text{def}}{=} \lambda z.(Y_{\text{val}}\ P\ z)$$

$$F' \stackrel{\text{def}}{=} \lambda z.(Y_{\text{val}}\ P'\ z)$$

Where Y_{val} is the CBV version of Church's fixpoint combinator, given in Section 2.7.1. For any integers n and m, and total function g from integers to integers, the following terms are contextually equivalent:

$$M \stackrel{\text{def}}{=} (F\ g\ n\ m)$$
$$M' \stackrel{\text{def}}{=} (F'\ g\ n\ m)$$

$$\frac{}{\vdash \lambda x.M\ R\ \lambda x.M'}$$

$$\frac{k > 0}{\vdash \lambda x.(g^k\ (F\ g\ n\ m))\ R\ \lambda x.(F'\ g\ n\ |g^k(m)|)}$$

Figure 2.4: Construction of an adequate relation for proving the equivalence of two recursive computations.

Both M and M' compute n iterations of g around m when $n \geq 0$, otherwise they both diverge. M' is a tail recursive computation. For example, if g is the successor function, both of the above computations return $n + m$ (when $n > 0$).

We define $(g^k\ e)$ to be the expression that wraps g times k around e, and, since g is a total function from integers to integers, $|g^k(m)|$ to be the integer result of such a wrapping around the integer m:

$$g^k(m) \stackrel{\text{def}}{=} \underbrace{(g \ldots (g}_{k}\ m) \ldots)$$

$$g^k(m) \Downarrow |g^k(m)| \qquad m \text{ integer}$$

Proof. For some x, we show that $\{x\} \vdash M \equiv M'$ using the method of Section 2.4. We construct a relation R and prove it adequate. Initially, we relate $\vdash \lambda x.M\ R\ \lambda x.M'$ by the first rule of Figure 2.4. We need to prove that the two symmetric conditions of Theorem 2.5.1 are satisfied for this pair of values. Here we show the proof of the first condition; the proof of the second is symmetric.

Let $IH_R(k-1)$ be true and $\vdash v\ R^{\mathsf{cxt}}\ v'$ and $(\lambda x.M\ v) \Downarrow^k w$. We need to show that for some w' $(\lambda x.M'\ v') \Downarrow w'$ and $\vdash w\ R^{\mathsf{cxt}}\ w'$. By the properties of

evaluation we have:

$$\begin{aligned} & (\lambda x.M\ v) \Downarrow^k w \\ \iff & (F\ g\ n\ m) \Downarrow^{k-1} w \\ \implies & \texttt{ifzero}\ n\ \texttt{then}\ m\ \texttt{else}\ (g\ (F\ g\ (n-1)\ m)) \Downarrow^{k-2} w \end{aligned} \tag{2.13}$$

Case $n = \texttt{0}$

Then $w = m$ and $(\lambda x.M'\ v') \Downarrow m$ and $\vdash m\ R^{\mathsf{cxt}}\ m$.

Case $n \neq \texttt{0}$

Then from (2.13) we get:

$$(g\ (F\ g\ (n-1)\ m)) \Downarrow^{k-3} w$$

and by the properties of evaluation:

$$(\lambda x.(g\ (F\ g\ (n-1)\ m))\ \texttt{0}) \Downarrow^{k-1} w$$

Similarly we have:

$$\begin{aligned} & (\lambda x.M'\ v') \Downarrow w' \\ \iff & (F'\ g\ n\ m) \Downarrow w' \\ \iff & \texttt{ifzero}\ n\ \texttt{then}\ m\ \texttt{else}\ (F'\ g\ (n-1)\ (g\ m)) \Downarrow w' \\ \iff & (F'\ g\ (n-1)\ (g\ m)) \Downarrow w' \end{aligned}$$

$$\begin{aligned} \iff & (F'\ g\ (n-1)\ |g^1(m)|) \Downarrow w' \\ \iff & (\lambda x.(F'\ g\ (n-1)\ |g^1(m)|)\ \texttt{0}) \Downarrow w' \end{aligned}$$

Because of $IH_R(k-1)$, it suffices to show:

$$\vdash \lambda x.(g\ (F\ g\ (n-1)\ m))\ R^{\mathsf{cxt}}\ \lambda x.(F'\ g\ (n-1)\ |g^1(m)|)$$

Hence, we add in our construction of R the second rule of Figure 2.4, which completes this case of the proof.

We now have to prove the condition of Theorem 2.5.1 for any $k > 0$ and any related procedures:

$$\vdash \lambda x.(g^k\ (F\ g\ n\ m))\ R\ \lambda x.(F'\ g\ n\ |g^k(m)|)$$

We follow the same process as before: let $IH_R(k-1)$ be true and $\vdash v\ R^{\mathsf{cxt}}\ v'$ and $(\lambda x.(g^k\ (F\ g\ n\ m))\ v) \Downarrow^k w$. We need to show that there exists w such that $(\lambda x.(F'\ g\ n\ |g^k(m)|)\ v') \Downarrow^k w$ and $\vdash w\ R^{\mathsf{cxt}}\ w'$.

By the properties of evaluation we have:

$$\begin{aligned} & (\lambda x.(g^k\ (F\ g\ n\ m))\ v) \Downarrow^k w \\ \iff & (g^k\ (F\ g\ n\ m)) \Downarrow^{k-1} w \\ \implies & (g^k\ (\texttt{ifzero}\ n\ \texttt{then}\ m\ \texttt{else}\ (g\ (F\ g\ (n-1)\ m)))) \Downarrow^{k-2} w \end{aligned} \tag{2.14}$$

Similarly we have:

$$\begin{aligned} & (\lambda x.(F'\ g\ n\ |g^k(m)|)\ v') \Downarrow w' \\ \iff & (F'\ g\ n\ |g^k(m)|) \Downarrow w' \\ \iff & \texttt{ifzero}\ n\ \texttt{then}\ |g^k(m)|\ \texttt{else}\ (F'\ g\ (n-1)\ (g\ |g^k(m)|)) \Downarrow w' \end{aligned} \tag{2.15}$$

Case $n = \texttt{0}$

From (2.14) we get: $(g^k\ m) \Downarrow^{k-3} w$ and thus $w = |g^k(m)|$. Moreover, from (2.15) we get $|g^k(m)| \Downarrow w'$ and thus $w' = |g^k(m)|$, and $\vdash |g^k(m)|\ R^{\mathsf{cxt}}\ |g^k(m)|$.

Case $n \neq \texttt{0}$

From (2.14) we get:

$$\begin{aligned} & (g^k\ (g\ (F\ g\ (n-1)\ m))) \Downarrow^{k-3} w \\ \iff & (g^{k+1}\ (F\ g\ (n-1)\ m)) \Downarrow^{k-3} w \\ \iff & (\lambda x.(g^{k+1}\ (F\ g\ (n-1)\ m))\ \texttt{0}) \Downarrow^{k-1} w \end{aligned}$$

and (2.15) is written equivalently:

$$\begin{aligned} & (F'\ g\ (n-1)\ (g\ |g^k(m)|)) \Downarrow w' \\ \iff & (F'\ g\ (n-1)\ |g^{k+1}(m)|) \Downarrow w' \\ \iff & (\lambda x.(F'\ g\ (n-1)\ |g^{k+1}(m)|)\ \texttt{0}) \Downarrow w' \end{aligned}$$

Because of $IH_R(k-1)$, it suffices to show:

$$\vdash \lambda x.(g^{k+1}\ (F\ g\ (n-1)\ m))\ R^{\mathsf{cxt}}\ \lambda x.(F'\ g\ (n-1)\ |g^{k+1}(m)|)$$

But this is true because by the second rule of Figure 2.4 we have:

$$\vdash \lambda x.(g^{k+1}\ (F\ g\ (n-1)\ m))\ R\ \lambda x.(F'\ g\ (n-1)\ |g^{k+1}(m)|)$$

Therefore R is adequate, and by Theorem 2.3.14 we get that $\{x\} \vdash M \equiv M'$, and, since M and M' are closed terms, $\vdash M \equiv M'$. □

2.7.3 Parallel-OR

Here we give two procedures and again prove them equivalent using the method discussed in Section 2.4: via an adequate relation. This proof, though, relies not on the iterative method of discussed in Section 2.6, but on operational reasoning on the terms.

This equivalence relies on the non-definability of the "parallel-or" operator in the CBV λ-calculus ([37], Section 2.5.6) and can also be proved using logical relations ([37], Example 8.6.3, pg. 589). Let M and M' be the following procedures:

$$\begin{array}{l}
M \stackrel{\text{def}}{=} \lambda f.\texttt{if}\ ((f\ \texttt{true})\ \lambda x.\Omega) \\
\qquad \texttt{then if}\ ((f\ \lambda x.\Omega)\ \texttt{true}) \\
\qquad\qquad \texttt{then if}\ ((f\ \texttt{false})\ \texttt{false}) \\
\qquad\qquad\qquad \texttt{then}\ \Omega \\
\qquad\qquad\qquad \texttt{else true} \\
\qquad\qquad \texttt{else}\ \Omega \\
\qquad \texttt{else}\ \Omega \\
M' \stackrel{\text{def}}{=} \lambda f.\Omega
\end{array}$$

where:

$$\Omega \stackrel{\text{def}}{=} (\lambda x.(x\ x)\ \lambda x.(x\ x))$$
$$\texttt{true} \stackrel{\text{def}}{=} \lambda x.\lambda y.(x\ \lambda z.z)$$
$$\texttt{false} \stackrel{\text{def}}{=} \lambda x.\lambda y.(y\ \lambda z.z)$$
$$\texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \stackrel{\text{def}}{=} ((e_1\ \lambda\_.e_2)\ \lambda\_.e_3)$$

Clearly M' diverges when applied to any argument. M also diverges because any lambda term that is given to it as an argument would have to either immediately diverge, or inspect its first argument first, or inspect its second argument first, or be a constant. M would only terminate when passed a function implementing parallel-OR—such a function is not definable in the lambda calculus.

To show M and M' equivalent, it suffices to show that the following set is adequate.

$$R = \{(M, M')\}$$

By Theorem 2.5.1, we need to show that M and M' co-terminate when applied to related arguments from R^{cxt}, and, if they do terminate, their final values are related in R^{cxt}. Unfortunately, the applications of the two procedures have no convenient synchronization points in their computations to help us with this proof. Instead, we will prove that both M and M' diverge when applied to any argument. For the case of M' we show this by showing that for any w, there is no natural number n such that $\Omega \Downarrow^n w$. For the case of M, the only way that there could be f and w, such that $(M\ f) \Downarrow w$, is if

$$((f\ \texttt{true})\ \lambda x.\Omega) \Downarrow \texttt{true}$$
$$((f\ \lambda x.\Omega)\ \texttt{true}) \Downarrow \texttt{true}$$
$$((f\ \texttt{false})\ \texttt{false}) \Downarrow \texttt{false}$$

which means that f behaves as a parallel-or operator. We can show that there is no procedure with such behavior by a proof similar to Mitchell's ([37], proof of Theorem 2.5.19).

CHAPTER 3

Lambda Calculus with Higher-Order Store

In this chapter we derive a sound and complete method for reasoning about contextual equivalence in an imperative extension of the lambda calculus, where functions are storable.

3.1 The Language

The term language is the untyped, CBV lambda calculus extended with constants, primitive operations, a conditional construct, tuples, and store operations; alpha-equivalent expressions are identified. The syntax of the language is shown in Figure 3.1.

The meta-variable c ranges over the set of natural numbers and booleans; op ranges over the arithmetic and boolean operators. Tuple construction is denoted by $(\overline{e})$, and projection of the i-th element of a tuple by $\pi_i(e)$. The values are constants, abstractions, locations, and tuples of values.

The expression $\nu x := c.\, e$ creates a new location in the store, assigns the constant c in that location, and binds x to the location inside e; expressions $!e$ and $(e_1 := e_2)$ return and update, respectively, the contents of a location in the store.

The domain of locations is an infinite countable set; the meta-variable l ranges over this set and L ranges over its finite subsets. A *store* is a finite function from locations to closed values; s and t range over stores. The value

EXPRESSION:	$e, d ::= x$	Identifier
	$\mid c$	Constant
	$\mid \lambda x.\, e$	Abstraction
	$\mid (e\ e)$	Application
	$\mid op(\overline{e})$	Primitive
	$\mid$ `if` e `then` e `else` e	Conditional
	$\mid (\overline{e})$	Tuple
	$\mid \pi_i(e)$	Projection
	$\mid l$	Location
	$\mid \nu x := c.\, e$	Allocation
	$\mid\ !e$	Dereferencing
	$\mid (e := e)$	Assignment
VALUE:	$u, v, w ::= c \mid \lambda x.\, e \mid (\overline{v}) \mid l$	
LOCATION:	l	
STORE:	$s, t \in \text{LOCATIONS} \xrightarrow{\text{fin}} \text{VALUES}$	
CONFIGURATION:	$\langle s,\, e\rangle \in \text{STORES} \times \text{EXPRESSIONS}$	

Figure 3.1: Syntactic domains of the imperative, CBV λ-calculus.

in location l of store s is given by $s(l)$; $s[l \mapsto v]$ assigns the value v in location l of store s, extending the domain of s if necessary. The domain of store s is given by $dom(s)$; $locs(e)$ denotes the set of locations that occur inside e.

A well-formed store contains closed values with locations within the domain of the store.

Definition 3.1.1 (Well-Formed Store)**.**

$wf(s) \stackrel{\text{def}}{=} \forall l \in dom(s).\ \vdash s(l) \ \wedge\ locs(s(l)) \subseteq dom(s)$

A *configuration*, $\langle s,\, e\rangle$, is a pair of store and expression. A well-formed configuration contains a well-formed store and a closed expression with locations within the domain of that store.

Definition 3.1.2 (Well-Formed Configuration)**.**

$wf\langle s,\, e\rangle \stackrel{\text{def}}{=} wf(s) \ \wedge\ \vdash e \ \wedge\ locs(e) \subseteq dom(s)$

$\boxed{\langle s,\, e\rangle \Downarrow^{k} \langle t,\, w\rangle}$

EVAL-VAL
$$\frac{k > 0}{\langle s,\, v\rangle \Downarrow^{k} \langle s,\, v\rangle}$$

EVAL-APP
$$\frac{\langle s,\, e_1\rangle \Downarrow^{k_1} \langle s_1,\, \lambda x.e_3\rangle \quad \langle s_1,\, e_2\rangle \Downarrow^{k_2} \langle s_2,\, w_2\rangle \quad \langle s_2,\, e_3[w_2/x]\rangle \Downarrow^{k_3} \langle t,\, w\rangle}{\langle s,\, (e_1\ e_2)\rangle \Downarrow^{1+k_1+k_2+k_3} \langle t,\, w\rangle}$$

EVAL-OP
$$\frac{\langle s,\, e_1\rangle \Downarrow^{k_1} \langle s_1,\, c_1\rangle \quad \ldots \quad \langle s_{n-1},\, e_n\rangle \Downarrow^{k_n} \langle t,\, c_n\rangle \quad op^{arith}(c_1, \ldots, c_n) = c}{\langle s,\, op(e_1, \ldots, e_n)\rangle \Downarrow^{1+\sum_{i\le n} k_i} \langle t,\, c\rangle}$$

EVAL-IF
$$\frac{\langle s,\, e_1\rangle \Downarrow^{k_1} \langle s_1,\, w_1\rangle \quad \langle s_1,\, e_i\rangle \Downarrow^{k_i} \langle t,\, w\rangle \quad (w_1, i) \in \{(\texttt{true}, 2), (\texttt{false}, 3)\}}{\langle s,\, \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3\rangle \Downarrow^{1+k_1+k_i} \langle t,\, w\rangle}$$

EVAL-PROJ
$$\frac{\langle s,\, e\rangle \Downarrow^{k} \langle t,\, (w_1, \ldots, w_n)\rangle \quad 1 \le i \le n}{\langle s,\, \pi_i(e)\rangle \Downarrow^{1+k} \langle t,\, w_i\rangle}$$

EVAL-TUPLE
$$\frac{\langle s,\, e_1\rangle \Downarrow^{k_1} \langle s_1,\, w_1\rangle \quad \ldots \quad \langle s_{n-1},\, e_n\rangle \Downarrow^{k_n} \langle t,\, w_n\rangle}{\langle s,\, (e_1, \ldots, e_n)\rangle \Downarrow^{1+\sum_{i\le n} k} \langle t,\, (w_1, \ldots, w_n)\rangle}$$

EVAL-NEW
$$\frac{\langle s[l{\mapsto}c],\, e[l/x]\rangle \Downarrow^{k} \langle t,\, w\rangle \quad l \notin dom(s)}{\langle s,\, \nu x{:=}c.\, e\rangle \Downarrow^{1+k} \langle t,\, w\rangle}$$

EVAL-DEREF
$$\frac{\langle s,\, e\rangle \Downarrow^{k} \langle t,\, l\rangle \quad t(l) = w}{\langle s,\, !e\rangle \Downarrow^{1+k} \langle t,\, w\rangle}$$

EVAL-ASSIGN
$$\frac{\langle s,\, e_1\rangle \Downarrow^{k_1} \langle s_1,\, l\rangle \quad \langle s_1,\, e_2\rangle \Downarrow^{k_2} \langle t,\, w\rangle}{\langle s,\, (e_1 := e_2)\rangle \Downarrow^{1+k_1+k_2} \langle t[l{\mapsto}w],\, 0\rangle}$$

Figure 3.2: Operational semantics of the imperative, CBV λ-calculus

For the rest of this chapter we consider only well-formed stores and configurations.

The big-step operational semantics of the language is defined for well-formed configurations, and is shown in Figure 3.2. Each big-step evaluation $\langle s,\, e\rangle \Downarrow^{k} \langle t,\, w\rangle$ consists of an initial well-formed configuration that contains an expression e and the store s, under which e evaluates, and a final configu-

ration that contains the final value w and store t. The natural number k is an upper bound on the number of derivations contained in the evaluation tree. We write $\langle s,\, e\rangle \Downarrow \langle t,\, v\rangle$ to mean that there is a k such that $\langle s,\, e\rangle \Downarrow^k \langle t,\, v\rangle$. This measure of big-step evaluation trees corresponds to the length of reductions in a small-step semantics, and it will be a stronger measure for the induction principle that we define in the following sections than the height of the evaluation trees.

Evaluation only increases the domain of the store and only yields well-formed configurations.

Lemma 3.1.3. *If* $\langle s,\, e\rangle \Downarrow \langle t,\, w\rangle$ *then* $dom(s) \subseteq dom(t)$.

Proof. By straightforward induction on the derivation $\langle s,\, e\rangle \Downarrow \langle t,\, w\rangle$. □

Lemma 3.1.4. *If* $wf\langle s,\, e\rangle$ *and* $\langle s,\, e\rangle \Downarrow \langle t,\, w\rangle$ *then* $wf\langle t,\, w\rangle$.

Proof. By straightforward induction on the derivation $\langle s,\, e\rangle \Downarrow \langle t,\, w\rangle$, using Lemma 3.1.3, and the definition of well-formed configuration (Definition 3.1.2). □

We say that a well-formed configuration $\langle s,\, e\rangle$ *terminates*, and we write $\langle s,\, e\rangle \Downarrow$, if and only if there is a finite derivation tree from the rules in Figure 3.2 that has $\langle s,\, e\rangle$ as initial configuration, and some $\langle t,\, w\rangle$ as final configuration. Conversely, if there is no such derivation, we say that $\langle s,\, e\rangle$ *diverges*, and we write $\langle s,\, e\rangle \Uparrow$.[1]

As for the λ-calculus in the previous chapter, a *context* $C[\cdot]$ is an expression C containing a hole. The variable-capturing substitution of expression e for the hole inside the context $C[\cdot]$ is written as $C[e]$. We write $C[\,]_\Phi$ when *at least* the variables in Φ are in scope at the position of the hole of C.

For the rest of this chapter we will use $\texttt{let}\, x = e_1\, \texttt{in}\, e_2$ and $(e_1;\, e_2)$ as syntactic sugar for $((\lambda x.\, e_2)\ e_1)$, depending on whether x is free in e_2.

[1] $\langle s,\, e\rangle \Uparrow$ denotes a diverging or stuck configuration.

3.2 Contextual Equivalence

We begin with the standard notion of contextual equivalence:

Definition 3.2.1 (Standard Contextual Equivalence $(\equiv)$)**.** $\Phi \vdash e \equiv e'$ *if and only if for all stores* s *and expression contexts* $C[\cdot]_\Phi$*, such that* $wf\langle s,\ C[e]\rangle$ *and* $wf\langle s,\ C[e']\rangle$*, we have*

$$\langle s,\ C[e]\rangle \Downarrow \iff \langle s,\ C[e']\rangle \Downarrow$$

Reasoning about contextual equivalence using the standard definition exhibits the same difficulties as for the case of the CBV λ-calculus, discussed in Section 2.2, but also an additional one involving the store.

- The definition deals with expressions in the same store, but since closures are storable, any inductive proof must somehow involve generalization to programs operating in different stores.

We restrict our reasoning to closed values, and extend it to open expressions via the following theorem.

Theorem 3.2.2 (Value Restriction)**.** *For any two expressions* $\{\overline{x}\} \vdash e, e'$*,*

$$\{\overline{x}\} \vdash e \equiv e' \iff \vdash \lambda\overline{x}.\, e \equiv \lambda\overline{x}.\, e'$$

To prove this theorem we define the relation $(\preccurlyeq)$ in Figure 3.3. This relation is the smallest substitutive congruence containing the beta-equivalent expressions $\Phi \uplus x \vdash e\ ,\ (\lambda x.\, e\ x)$ for all Φ, x, and e. We will show that expressions related in $(\preccurlyeq)$ co-terminate.

Lemma 3.2.3 (Properties of $(\preccurlyeq)$)**.**

1. *If* $\Phi \vdash e \preccurlyeq e'$ *then* $\Phi \vdash e$*, and* $\Phi \vdash e'$*.*
2. *If* $\Phi \vdash e$ *then* $\Phi \vdash e \preccurlyeq e$*.*
3. *If* $\{\overline{x}\} \vdash e \preccurlyeq e'$ *and* $\vdash \overline{v} \preccurlyeq \overline{v'}$ *then* $\vdash e[\overline{v/x}] \preccurlyeq e'[\overline{v'/x}]$*.*

$\boxed{\Phi \vdash e \preccurlyeq e'}$

$$\Phi \vdash c \preccurlyeq c \qquad \Phi \vdash l \preccurlyeq l \qquad \frac{x \in \Phi}{\Phi \vdash x \preccurlyeq x} \qquad \frac{\Phi \vdash \overline{e} \preccurlyeq \overline{e'}}{\Phi \vdash op(\overline{e}) \preccurlyeq op(\overline{e'})}$$

$$\frac{\Phi \uplus \{x\} \vdash e \preccurlyeq e'}{\Phi \vdash \lambda x.e \preccurlyeq \lambda x.e'} \qquad \frac{\Phi \vdash e_1 \preccurlyeq e_1' \qquad \Phi \vdash e_2 \preccurlyeq e_2'}{\Phi \vdash (e_1\ e_2) \preccurlyeq (e_1'\ e_2')} \qquad \frac{\Phi \vdash \overline{e} \preccurlyeq \overline{e'}}{\Phi \vdash (\overline{e}) \preccurlyeq (\overline{e'})}$$

$$\frac{\Phi \vdash e \preccurlyeq e'}{\Phi \vdash \pi_i(e) \preccurlyeq \pi_i(e')} \qquad \frac{\Phi \vdash e_1 \preccurlyeq e_1' \qquad \Phi \vdash e_2 \preccurlyeq e_2' \qquad \Phi \vdash e_3 \preccurlyeq e_3'}{\Phi \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \preccurlyeq \texttt{if}\ e_1'\ \texttt{then}\ e_2'\ \texttt{else}\ e_3'}$$

$$\frac{\Phi \uplus \{x\} \vdash e \preccurlyeq e'}{\Phi \vdash \nu x{:=}c.e \preccurlyeq \nu x{:=}c.e'} \qquad \frac{\Phi \vdash e \preccurlyeq e'}{\Phi \vdash\ !e \preccurlyeq\ !e'} \qquad \frac{\Phi \vdash e_1 \preccurlyeq e_1' \qquad \Phi \vdash e_2 \preccurlyeq e_2'}{\Phi \vdash (e_1 := e_2) \preccurlyeq (e_1' := e_2')}$$

$$\frac{\Phi \uplus \{x\} \vdash e \preccurlyeq e'}{\Phi \uplus \{x\} \vdash e \preccurlyeq (\lambda x.e'\ x)} \qquad \frac{\Phi \uplus \{x\} \vdash e \preccurlyeq e' \qquad \vdash v \preccurlyeq v'}{\Phi \vdash e[v/x] \preccurlyeq e'[v'/x]}$$

$\boxed{\vdash s \preccurlyeq s'}$

$$\frac{}{\vdash [] \preccurlyeq []} \qquad \frac{\vdash s \preccurlyeq s' \qquad \vdash v \preccurlyeq v' \qquad l \notin dom(s)}{\vdash s[l \mapsto v] \preccurlyeq s'[l \mapsto v']}$$

Figure 3.3: Definition of $(\preccurlyeq)$ for the imperative λ-calculus

4. *If* $\Phi \vdash e \preccurlyeq e'$ *and* $\Psi \vdash C[\cdot]_\Phi$ *then* $\Psi \vdash C[e] \preccurlyeq C[e']$.

5. *If* $\Phi \vdash v \preccurlyeq v'$ *then* v *and* v' *have the same top-level language constructor.*

6. *If* $\Phi \vdash v \preccurlyeq v'$ *then for any location* l, $v = l$ *iff* $v' = l$.

Proof. We prove the first, fifth, and sixth property by induction on the derivation $\Phi \vdash e \preccurlyeq e'$, and the second property by structural induction on e. We prove the third property by induction on the size of the sequence $\overline{x}$.

We prove the fifth property by structural induction on C, using the second property and the following induction hypothesis:

$$IH(C) \stackrel{\text{def}}{=} \forall \Psi.\ (\Psi \vdash C[\cdot]_\Phi \implies \Psi \vdash C[e] \preccurlyeq C[e'])$$

□

Configurations constructed by parts in $(\preccurlyeq)$ co-terminate.

Lemma 3.2.4. *If* $\{\overline{x}\} \vdash e \preccurlyeq e'$, $\vdash \overline{v} \preccurlyeq \overline{v'}$, *and* $\vdash s \preccurlyeq s'$, *then*

$$\langle s,\, e[\overline{v/x}]\rangle \Downarrow \iff \langle s',\, e'[\overline{v'/x}]\rangle \Downarrow$$

Proof. We prove this lemma by proving the following two properties:

$$(\{\overline{x}\} \vdash e \preccurlyeq e') \wedge (\vdash \overline{v} \preccurlyeq \overline{v'}) \wedge (\vdash s \preccurlyeq s') \wedge (\langle s,\, e[\overline{v/x}]\rangle \Downarrow \langle t,\, w\rangle) \tag{3.1}$$
$$\implies \exists t', w'.\ (\langle s',\, e'[\overline{v'/x}]\rangle \Downarrow \langle t',\, w'\rangle) \wedge (\vdash t \preccurlyeq t') \wedge (\vdash w \preccurlyeq w')$$

$$(\{\overline{x}\} \vdash e \preccurlyeq e') \wedge (\vdash \overline{v} \preccurlyeq \overline{v'}) \wedge (\vdash s \preccurlyeq s') \wedge (\langle s,\, e'[\overline{v'/x}]\rangle \Downarrow \langle t',\, w'\rangle) \tag{3.2}$$
$$\implies \exists t, w.\ (\langle s,\, e[\overline{v/x}]\rangle \Downarrow \langle t,\, w\rangle) \wedge (\vdash t \preccurlyeq t') \wedge (\vdash w \preccurlyeq w')$$

Here we show the proof of (3.1); the proof of (3.2) is similar. We proceed by induction on the height of the derivations $\langle s,\, e[\overline{v/x}]\rangle \Downarrow \langle t,\, w\rangle$ and $\{\overline{x}\} \vdash e \preccurlyeq e'$, ordered lexicographically. The induction hypothesis is the following.

$$IH(k,m) \stackrel{\text{def}}{=}$$
$$\forall \overline{x, v, v'}, e, e', s, s', w, t.$$
$$(\{\overline{x}\} \vdash e \preccurlyeq^m e')) \wedge (\vdash \overline{v} \preccurlyeq \overline{v'}) \wedge (\vdash s \preccurlyeq s') \wedge (\langle s,\, e[\overline{v/x}]\rangle \Downarrow^k \langle t,\, w\rangle)$$
$$\implies \exists t', w'.\ (\langle s',\, e'[\overline{v'/x}]\rangle \Downarrow \langle t',\, w'\rangle) \wedge (\vdash t \preccurlyeq t') \wedge (\vdash w \preccurlyeq w')$$

We will prove that for all k and m, $IH(k,m)$ holds by proving that for any k, and m

$$(\forall j, n.\ (j,n) \prec_{\text{lex}} (k,m) \implies IH(j,n)) \implies IH(k,m)$$

We assume

$$\forall j, n.\ (j,n) \prec_{\text{lex}} (k,m) \implies IH(j,n)$$

and $\{\overline{x}\} \vdash e \preccurlyeq^m e'$, $\vdash \overline{v} \preccurlyeq \overline{v'}$, $\vdash s \preccurlyeq s'$, and $\langle s,\, e[\overline{v/x}]\rangle \Downarrow^k \langle t,\, w\rangle$. We proceed by cases on $\{\overline{x}\} \vdash e \preccurlyeq^m e'$.

Cases $\{\overline{x}\} \vdash c \preccurlyeq^m c$ and $\{\overline{x}\} \vdash l \preccurlyeq^m l$

Let $u = c$ or $u = l$, and $u' = u$. We have $u[\overline{v/x}] = u$, $t = s$, and $\langle s,\, u\rangle \Downarrow^k \langle s,\, u\rangle$; moreover $\langle s',\, u'\rangle \Downarrow \langle s',\, u'\rangle$, $\vdash s \preccurlyeq s'$, and $\vdash u \preccurlyeq u'$.

Case $\dfrac{x_i \in \{\overline{x}\}}{\{\overline{x}\} \vdash x_i \preccurlyeq^m x_i}$

We have $x_i[\overline{v/x}] = v_i$, $t = s$, and $\langle s,\, v_i\rangle \Downarrow^k \langle s,\, v_i\rangle$; moreover $\langle s',\, v_i'\rangle \Downarrow \langle s',\, v_i'\rangle$, $\vdash s \preccurlyeq s'$, and $\vdash v_i \preccurlyeq v_i'$.

Case $\dfrac{\{\overline{x}\} \uplus \{y\} \vdash e_0 \preccurlyeq^{m-1} e_0'}{\{\overline{x}\} \vdash \lambda y.e_0 \preccurlyeq^m \lambda y.e_0'}$

We have $t = s$ and $\langle s,\, \lambda y.e_0[\overline{v/x}]\rangle \Downarrow^k \langle s,\, \lambda y.e_0[\overline{v/x}]\rangle$; moreover $\langle s',\, \lambda y.e_0'[\overline{v'/x}]\rangle \Downarrow \langle s',\, \lambda y.e_0'[\overline{v'/x}]\rangle$, $\vdash s \preccurlyeq s'$, and by Lemma 3.2.3 (3):

$$\vdash \lambda y.e_0[\overline{v/x}] \preccurlyeq \lambda y.e_0'[\overline{v'/x}].$$

Case $\dfrac{\{\overline{x}\} \vdash e_1 \preccurlyeq^{m_1} e_1' \qquad \{\overline{x}\} \vdash e_2 \preccurlyeq^{m_2} e_2'}{\{\overline{x}\} \vdash (e_1\ e_2) \preccurlyeq^m (e_1'\ e_2')}, \quad m_1 < m, \quad m_2 < m$

We have $\langle s,\, (e_1\ e_2)[\overline{v/x}]\rangle \Downarrow^k \langle t,\, w\rangle$, thus for some s_1, s_2, $\lambda y.e_3$, w_2, and $\overline{k} < k$:

$$\begin{aligned}\langle s,\, e_1[\overline{v/x}]\rangle &\Downarrow^{k_1} \langle s_1,\, \lambda y.e_3\rangle\\ \langle s_1,\, e_2[\overline{v/x}]\rangle &\Downarrow^{k_2} \langle s_2,\, w_2\rangle\\ \langle s_2,\, e_3[\overline{w_2/y}]\rangle &\Downarrow^{k_3} \langle t,\, w\rangle\end{aligned}$$

By $IH(k_1, m_1)$ and $IH(k_2, m_2)$ we get that there exist s_1', s_2', $\lambda y.e_3'$ and w_2', such that:

$$\begin{array}{lll}\langle s',\, e_1'[\overline{v'/x}]\rangle \Downarrow \langle s_1',\, \lambda y.e_3'\rangle & \vdash s_1 \preccurlyeq s_1' & \vdash \lambda y.e_3 \preccurlyeq \lambda y.e_3'\\ \langle s_1',\, e_2'[\overline{v'/x}]\rangle \Downarrow \langle s_2',\, w_2'\rangle & \vdash s_2 \preccurlyeq s_2' & \vdash w_2 \preccurlyeq w_2'\end{array}$$

By the construction of $(\preccurlyeq)$ we get that $\{y\} \vdash e_3 \preccurlyeq e_3'$, and $\vdash e_3[w_2/y] \preccurlyeq e_3'[w_2'/y]$. Thus, by $IH(k_3, m_3)$, for any m_3, we get that there exist t' and w', such that:

$$\langle s_2',\, e_3'[w_2'/y]\rangle \Downarrow \langle t',\, w'\rangle \qquad \vdash t \preccurlyeq t' \qquad \vdash w \preccurlyeq w'$$

and therefore, by the evaluation rule of application: $\langle s',\, (e_1'\ e_2')[\overline{v'/z}]\rangle \Downarrow \langle t',\, w'\rangle$.

Case $\dfrac{\{\overline{x}\} \vdash \overline{e} \preccurlyeq^{\overline{n}} \overline{e'}}{\{\overline{x}\} \vdash (\overline{e}) \preccurlyeq^{m} (\overline{e'})}$

This case is carried out by applying the induction hypothesis once for each premise.

Cases $\dfrac{\{\overline{x}\} \vdash \overline{e} \preccurlyeq^{\overline{n}} \overline{e'}}{\{\overline{x}\} \vdash op(\overline{e}) \preccurlyeq op(\overline{e'})}$, $\dfrac{\{\overline{x}\} \vdash e \preccurlyeq^{m-1} e'}{\{\overline{x}\} \vdash \pi_i(e) \preccurlyeq \pi_i(e')}$ and

$$\frac{\{\overline{x}\} \vdash e_1 \preccurlyeq^{m_1} e_1' \qquad \{\overline{x}\} \vdash e_2 \preccurlyeq^{m_2} e_2' \qquad \{\overline{x}\} \vdash e_3 \preccurlyeq^{m_3} e_3'}{\{\overline{x}\} \vdash \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \preccurlyeq^{m} \texttt{if } e_1' \texttt{ then } e_2' \texttt{ else } e_3'}$$

These cases are carried out by the induction hypothesis and Lemma 3.2.3 (5) to show that the reducts of subexpressions are identical constants or values of the same kind.

Case $\dfrac{\{\overline{x}\} \vdash e \preccurlyeq^{m-1} e'}{\{\overline{x}\} \vdash !e \preccurlyeq^{m} !e'}$

We have $\langle s,\, !e\rangle \Downarrow^{k} \langle t,\, w\rangle$, thus $\langle s,\, e\rangle \Downarrow^{k-1} \langle t,\, l\rangle$ and $t(l) = w$. By $IH(k-1, m-1)$ we get that there exist t' and u', such that

$$\langle s',\, e'\rangle \Downarrow \langle t',\, u'\rangle \qquad \vdash t \preccurlyeq t' \qquad \vdash l \preccurlyeq u'$$

By Lemma 3.2.3 (6) we get that $u' = l$. Furthermore, by the definition of $\vdash t \preccurlyeq t'$ we get that there exists w', such that $t'(l) = w'$ and $\vdash w \preccurlyeq w'$. Finally, by the operational semantics, $\langle s',\, !e\rangle \Downarrow \langle t',\, w'\rangle$.

Case $\dfrac{\{\overline{x}\} \uplus \{y\} \vdash e \preccurlyeq^{m-1} e'}{\{\overline{x}\} \vdash \nu y{:=}c.\, e \preccurlyeq^{m} \nu y{:=}c.\, e'}$

We have $\langle s,\, \nu y{:=}c.\, e\rangle \Downarrow^{k} \langle t,\, w\rangle$, thus, for $l \notin dom(s)$, $\langle s[l{\mapsto}c],\, e[l/x]\rangle \Downarrow^{k-1} \langle t,\, w\rangle$. By the definition of $(\preccurlyeq)$ for stores, we get that $\vdash s[l{\mapsto}c] \preccurlyeq s'[l{\mapsto}c]$, and by the construction of $(\preccurlyeq)$ for expressions we get $\vdash e[l/x] \preccurlyeq e'[l/x]$. Moreover, by $IH(k-1, m-1)$ we get that there exist t' and w', such that:

$$\langle s'[l{\mapsto}c],\, e'[l/x]\rangle \Downarrow \langle t',\, w'\rangle \qquad \vdash t \preccurlyeq t' \qquad \vdash w \preccurlyeq w'$$

By the operational semantics we get $\langle s',\, \nu y{:=}c.\, e'\rangle \Downarrow \langle t',\, w'\rangle$.

Case $\dfrac{\{\overline{x}\} \vdash e_1 \preccurlyeq^{m_1} e_1' \qquad \{\overline{x}\} \vdash e_2 \preccurlyeq^{m_2} e_2'}{\{\overline{x}\} \vdash (e_1 := e_2) \preccurlyeq^{m} (e_1' := e_2')}$, $m_1 < m$, and $m_2 < m$.

This case is carried out by the induction hypothesis and the properties of $(\preccurlyeq)$.

Case $\dfrac{\{\overline{x}\} \uplus \{y\} \vdash e \preccurlyeq e'}{\{\overline{x}\} \uplus \{y\} \vdash e \preccurlyeq (\lambda y.e'\ y)}$

We have $\langle s,\ e[\overline{v/x}, u/y]\rangle \Downarrow^k \langle t,\ w\rangle$. By $IH(k, m-1)$ we get that there exist t' and w', such that:

$$\langle s',\ e'[\overline{v'/x}, u/y]\rangle \Downarrow \langle t',\ w'\rangle \qquad \vdash t \preccurlyeq t' \qquad \vdash w \preccurlyeq w'$$

and therefore $\langle s',\ (\lambda y.e'\ y)[\overline{v'/x}, u/y]\rangle \Downarrow \langle t',\ w'\rangle$.

Case $\dfrac{\{\overline{x}\} \uplus \{y\} \vdash e \preccurlyeq^{m_1} e' \qquad \vdash u \preccurlyeq^{m_2} u'}{\{\overline{x}\} \vdash e[u/y] \preccurlyeq^{m} e'[u'/y]}$, $m_1 < m$, and $m_2 < m$.

We have $\langle s,\ e[\overline{v/x}, u/y]\rangle \Downarrow^k \langle t,\ w\rangle$. By $IH(k, m_1)$ we get that there exist t' and w', such that:

$$\langle s',\ e'[\overline{v'/x}, u'/y]\rangle \Downarrow \langle t',\ w'\rangle \qquad \vdash t \preccurlyeq t' \qquad \vdash w \preccurlyeq w'$$

□

Lemma 3.2.5. *For any two expressions* $\Phi \uplus \{x\} \vdash e, e'$*,*

$$\Phi \uplus \{x\} \vdash e \equiv e' \iff \Phi \vdash \lambda x.e \equiv \lambda x.e'$$

Proof. The forward direction is easily shown by the definition of $(\equiv)$: for all contexts $\vdash C[\cdot]_{\Phi \uplus \{x\}}$ and stores s, such that $wf\langle s,\ C[e]\rangle$ and $wf\langle s,\ C[e]\rangle$,

$$\langle s,\ C[e]\rangle \Downarrow \iff \langle s,\ C[e']\rangle \Downarrow$$

Hence, for all contexts $\vdash d[\lambda x.[\cdot]_{\Phi \uplus \{x\}}]_{\Phi}$, with $d[\lambda x.[\cdot]] = C[\cdot]$,

$$\langle s,\ d[\lambda x.[e]]\rangle \Downarrow \iff \langle s,\ d[\lambda x.[e']]\rangle \Downarrow$$

thus, $\Phi \vdash \lambda x.e \equiv \lambda x.e'$.

For the reverse direction we need to show that for all contexts $\vdash C[\cdot]_{\Phi \uplus \{x\}}$ and stores s, such that $wf\langle s,\ C[e]\rangle$ and $wf\langle s,\ C[e]\rangle$,

$$\langle s,\ C[e]\rangle \Downarrow \iff \langle s,\ C[e']\rangle \Downarrow$$

assuming $\Phi \vdash \lambda x.e \equiv \lambda x.e'$.

By the construction of $(\preccurlyeq)$, and Lemma 3.2.3 (4) we have:

$$\vdash C[e] \preccurlyeq C[(\lambda x.e\ x)] \tag{3.3}$$

$$\vdash C[e'] \preccurlyeq C[(\lambda x.e'\ x)] \tag{3.4}$$

$$\vdash s \preccurlyeq s \tag{3.5}$$

Hence we have:

$$\begin{array}{ll} & \langle s,\ C[e]\rangle \Downarrow \\ \textit{iff} & \langle s,\ C[(\lambda x.e\ x)]\rangle \Downarrow \text{ (by Lemma 3.2.4, (3.3), and (3.5))} \\ \textit{iff} & \langle s,\ C[(\lambda x.e'\ x)]\rangle \Downarrow \ (\Phi \vdash \lambda x.e \equiv \lambda x.e') \\ \textit{iff} & \langle s,\ C[e']\rangle \Downarrow \qquad \text{(by Lemma 3.2.4, (3.4), and (3.5)} \end{array}$$

□

The Value Restriction theorem (Theorem 3.2.2) is immediate from Lemma 3.2.5.

As a consequence of the Value Restriction theorem, instead of reasoning about the equivalence of two, possibly open, expressions e and e', we will close them under an appropriate number of abstractions and reason about the equivalence of the corresponding closed values. If e and e' are closed non-values, then we turn them into the values $\lambda x.e$ and $\lambda x.e'$, for an arbitrary identifier x, and reason about them instead.

3.3 Derivation of Adequacy

3.3.1 Pre-Adequacy

In this language two expressions are equivalent under pairs of stores. Contextual equivalence (Definition 3.2.1) relates terms that are equivalent un-

der any pair of *identical* stores. Any inductive method for proving contextual equivalence, though, would have to relate expressions under possibly pairs of *different* stores, since during execution related expressions may change part of the store differently, but in a manner not observable by the context. Thus, we (1) relate expressions under different stores, and (2) keep track of the part of the store that is *local* to each expression. We do so by following the method of Sumii and Pierce [57, 58].[2]

We will annotate each relation on values with a pair of stores, one for each side of the relation under which the relation is intended to hold. We write such annotated relations as:

$$(s, s', R) \in \textsc{Store} \times \textsc{Store} \times \mathscr{P}(\textsc{Value} \times \textsc{Value})$$

An annotated relation (s, s', R) is well-formed when

$$wf(s) \qquad wf(s') \qquad \forall \vdash v\ R\ v'.\ (locs(v) \subseteq dom(s)) \wedge (locs(v') \subseteq dom(s'))$$

For the rest of this chapter we consider only well-formed annotated relations.

Since evaluation of two equivalent expressions can lead to pairs of stores other than the initial one—under which more equivalences may hold—we need to reason about sets of such relations:

$$\mathcal{X} \subseteq \textsc{Store} \times \textsc{Store} \times \mathscr{P}(\textsc{Value} \times \textsc{Value})$$

The inverse of a set of annotated relations is defined as follows.

Definition 3.3.1. *If $\mathcal{X}$ is a set of annotated relations, the inverse of $\mathcal{X}$, written $\mathcal{X}^{\mathsf{T}}$, is defined as:*

$$(s', s, R^{\mathsf{T}}) \in \mathcal{X}^{\mathsf{T}} \quad \textit{iff} \quad (s, s', R) \in \mathcal{X}$$

We also define the open extension of sets of annotated relations.

Definition 3.3.2 (Open Extension of Sets of Annotated Relations)**.** *If $\mathcal{X}$ is a set of annotated relations, then $\mathcal{X}^{\circ}$ is the relation such that $\{\overline{x}\} \vdash e\ \mathcal{X}^{\circ}\ e'$*

[2] Similar ideas were also introduced in previous work [15, 32, 40]

if and only if for all stores s *such that* $wf(s)$ *and* $locs(e) \cup locs(e') \subseteq dom(s)$*, there exists* R *such that:*

$$(s, s, R) \in \mathcal{X} \qquad \vdash \lambda\overline{x}.e\ R\ \lambda\overline{x}.e' \qquad \forall l \in dom(s).\ \vdash l\ R\ l$$

Locations related by a relation act as *public* locations; i.e. locations known to the context. Thus, if we have (s, s', R), and $R|_{loc}$ is the largest subset of R relating locations, then $dom(R|_{loc})$ and $codom(R|_{loc})$ are the public locations of the left and right side of the relation, respectively. Similarly, $dom(s)\backslash dom(R|_{loc})$ and $dom(s')\backslash codom(R|_{loc})$ are the *private*, or local, locations of each side of the relation.

The distinction of locations to public and private is encoded in the definition of the context closure of relations.

Definition 3.3.3 (Context Closure)**.** *If* R *is a binary relation on closed values, then* R^{cxt} *is the relation defined by*

$$\frac{\{\overline{x}\} \vdash d \qquad \vdash \overline{u}\ R\ \overline{u'} \qquad locs(d) = \emptyset}{\vdash d[\overline{u/x}]\ R^{\mathsf{cxt}}\ d[\overline{u'/x}]}$$

This relation describes all contexts dXS containing related values. By disallowing the context to contain a priori arbitrary store locations we make sure that the only locations it refers to are the "public" ones from R.

As in the case of the CBV λ-calculus, R contains only closed values, thus permitting the use of the β-substitution $d[v/x]$, instead of the capturing substitution $d[v]$, where x is a free identifier in the hole of d. Moreover, the generalization to contexts with multiple holes is needed by the induction principle we will introduce later.

Context closure satisfies the following properties.

Lemma 3.3.4 (Properties of $(\ )^{\mathsf{cxt}}$)**.**

1. *If* $R \subseteq Q$*, then* $R^{\mathsf{cxt}} \subseteq Q^{\mathsf{cxt}}$.
2. $(R^{\mathsf{cxt}})^{\mathsf{cxt}} = R^{\mathsf{cxt}}$.

3. If $R \subseteq Q \subseteq R^{\mathsf{cxt}}$, then $R^{\mathsf{cxt}} = Q^{\mathsf{cxt}}$.

Proof. By elementary properties of substitution. □

We now give the definition of *pre-adequate annotated relations*. These relate closed values which the context is unable to distinguish under a pair of (possibly different) stores.

Definition 3.3.5 (Pre-Adequate Annotated Relations)**.** *An annotated relation, (s, s', R), is pre-adequate if and only if for all expressions $\vdash e\ R^{\mathsf{cxt}}\ e'$*

$$\langle s,\, e\rangle \Downarrow \iff \langle s',\, e'\rangle \Downarrow$$

The set containing all pre-adequate annotated relations is called *pre-adequacy*.

Definition 3.3.6 (Pre-Adequacy $(\cong)$)**.** $(\cong)$ *is the set of all pre-adequate annotated relations.*

We show that the open extension of pre-adequacy is sound and complete with respect to contextual equivalence.

Theorem 3.3.7 (Soundness and Completeness of $(\cong)$)**.** $(\cong)^\circ = (\equiv)$

Proof. Let $\overline{x}$ be a non-empty sequence of variables. Then:

$$\{\overline{x}\} \vdash e \equiv e'$$

or equivalently, by Theorem 3.2.2,

$$\vdash \lambda\overline{x}.\, e \equiv \lambda\overline{x}.\, e'$$

or equivalently, by the definition of $(\equiv)$

$$\begin{aligned}
&\forall s, C.\ \vdash C[\cdot]_\emptyset \\
&\quad \wedge\ \mathit{wf}\,\langle s,\ C[\lambda\overline{x}.\, e]\rangle \\
&\quad \wedge\ \mathit{wf}\,\langle s,\ C[\lambda\overline{x}.\, e']\rangle \\
&\qquad \Longrightarrow\ (\langle s,\ C[\lambda\overline{x}.\, e]\rangle \Downarrow \iff \langle s,\ C[\lambda\overline{x}.\, e']\rangle \Downarrow)
\end{aligned}$$

or equivalently, by choosing the appropriate $\{\overline{y}, z\} \vdash d$ and $\overline{l}$ for the forward direction (and the appropriate $\vdash C[\cdot]_\emptyset$ for the reverse), such that $locs(d) = \emptyset$, $\{\overline{l}\} = dom(s)$, and $\{z\} \vdash C[z] = d[\overline{l/y}]$:

$$\begin{aligned}\forall s, \overline{l,y}, z, d.\ & \{\overline{y}, z\} \vdash d \wedge locs(d) = \emptyset \wedge \{\overline{l}\} = dom(s) \\ & \wedge\ wf\langle s,\ d[\overline{l/y}, \lambda\overline{x}.e/z]\rangle \\ & \wedge\ wf\langle s,\ d[\overline{l/y}, \lambda\overline{x}.e'/z]\rangle \\ & \quad \Longrightarrow\ (\langle s,\ d[\overline{l/y}, \lambda\overline{x}.e/z]\rangle \Downarrow \iff \langle s,\ d[\overline{l/y}, \lambda\overline{x}.e'/z]\rangle \Downarrow)\end{aligned}$$

or equivalently:

$$\begin{aligned}\forall s, \overline{l}, \{\overline{y}, z\} \vdash d.\ & locs(d) = \emptyset \wedge \{\overline{l}\} = dom(s) \\ & \wedge\ wf(s) \wedge locs(e) \cup locs(e') \subseteq dom(s) \\ & \quad \Longrightarrow\ (\langle s,\ d[\overline{l/y}, \lambda\overline{x}.e/z]\rangle \Downarrow \iff \langle s,\ d[\overline{l/y}, \lambda\overline{x}.e'/z]\rangle \Downarrow)\end{aligned}$$

or equivalently, by choosing $R = \{(\lambda\overline{x}.e, \lambda\overline{x}.e'), \overline{(l,l)}\}$ for the forward direction:

$$\begin{aligned}\forall s.\ & wf(s) \wedge locs(e) \cup locs(e') \subseteq dom(s) \\ & \Longrightarrow\ \exists R.\ \vdash \lambda\overline{x}.e\ R\ \lambda\overline{x}.e' \\ & \qquad \wedge\ \forall l \in dom(s).\ \vdash l\ R\ l \\ & \qquad \wedge\ \forall \vdash e_d\ R^{\mathsf{cxt}}\ e'_d.\ (\langle s,\ e_d\rangle \Downarrow \iff \langle s,\ e'_d\rangle \Downarrow)\end{aligned}$$

or equivalently, by Definition 3.3.5 and the definition of $(\cong)$:

$$\begin{aligned}\forall s.\ & wf(s) \wedge locs(e) \cup locs(e') \subseteq dom(s) \\ & \Longrightarrow\ \exists R.\ \vdash \lambda\overline{x}.e\ R\ \lambda\overline{x}.e' \\ & \qquad \wedge\ \forall l \in dom(s).\ \vdash l\ R\ l \\ & \qquad \wedge\ (s, s, R) \in (\cong)\end{aligned}$$

or equivalently, by Definition 3.3.2:

$$\{\overline{x}\} \vdash e \cong e'$$

□

3.3.2 Adequacy

In order to carry out an induction, we need to show not merely that $\langle s,\, e\rangle \Downarrow$ if and only if $\langle s',\, e'\rangle \Downarrow$, but also that if both terminate the terminal configurations are related in some state of $\mathcal{X}$. We express this in the definition for adequate sets of relations.

Definition 3.3.8 (Adequate Sets of Annotated Relations)**.** *A set of annotated relations $\mathcal{X}$ is* adequate *if an only if for all $(s, s', R) \in \mathcal{X}$ we have:*

$$\begin{aligned}
\forall e, e', t, w.\ \vdash e\ R^{\mathsf{cxt}}\ e' \\
\quad \wedge\ \langle s,\, e\rangle \Downarrow \langle t,\, w\rangle \\
\quad\quad \Longrightarrow\ \exists t', w', Q.\ \langle s',\, e'\rangle \Downarrow \langle t',\, w'\rangle \\
\quad\quad\quad\quad \wedge\ \vdash w\ Q^{\mathsf{cxt}}\ w' \\
\quad\quad\quad\quad \wedge\ (t, t', Q) \in \mathcal{X} \\
\quad\quad\quad\quad \wedge\ R \subseteq Q
\end{aligned}$$

and similarly for all $(s, s', R) \in \mathcal{X}^{\mathsf{T}}$.

Adequate sets of relations ensure co-termination of related configurations (similar to pre-adequacy), but are also *closed under evaluation*. The terminal configurations are related in some (t, t', Q), which relates the reduced values under the pair of final stores t and t'.

The requirement that $R \subseteq Q$ is sufficient for successfully carrying out an induction. An alternative, more intuitive, explanation of this requirement is that equivalences known to the context before evaluation are still known to it after evaluation of the terms.

It is easy to see that the union of adequate sets is an adequate set. Thus, the union of all adequate sets is the largest adequate set.

Definition 3.3.9 (Adequacy $(\approx)$)**.** *$(\approx)$ is the largest adequate set of annotated relations.*

Theorem 3.3.10 (Soundness of Adequate Sets)**.** *If $\mathcal{X}$ is adequate then it is included in pre-adequacy.*

Proof. Trivial by the definitions of pre-adequate annotated relations and adequate sets of annotated relations. □

Theorem 3.3.11 (Completeness of Adequate Sets)**.** $(\cong)$ *is adequate.*

Proof. Let $(s, s', R) \in (\cong)$ and $\vdash e\ R^{\mathsf{cxt}}\ e'$. We will show that

$$\begin{aligned}\forall t, w.\ \langle s,\, e\rangle \Downarrow \langle t,\, w\rangle \\ \implies \exists t', w', Q.\ \langle s',\, e'\rangle \Downarrow \langle t',\, w'\rangle \\ \wedge\ \vdash w\ Q^{\mathsf{cxt}}\ w' \\ \wedge\ (t, t', Q) \in (\cong) \\ \wedge\ R \subseteq Q\end{aligned}$$

By the definition of pre-adequate annotated relations (Definition 3.3.5) and the determinacy of the semantics, it suffices to show that

$$\begin{aligned}\forall t, t', w, w'.\ \langle s,\, e\rangle \Downarrow \langle t,\, w\rangle \\ \wedge\ \langle s',\, e'\rangle \Downarrow \langle t',\, w'\rangle \\ \implies \exists Q.\ \vdash w\ Q^{\mathsf{cxt}}\ w' \\ \wedge\ (t, t', Q) \in (\cong) \\ \wedge\ R \subseteq Q\end{aligned}$$

Let $\langle s,\, e\rangle \Downarrow \langle t,\, w\rangle$ and $\langle s',\, e'\rangle \Downarrow \langle t',\, w'\rangle$. We will show that

$$(t, t', R \cup \{(w, w')\}) \in (\cong)$$

For any $\overline{x, v, v'}$, y, and d, such that $\{\overline{x}, y\} \vdash d$, and $\vdash \overline{v}\ R\ \overline{v'}$ we have:

$$\begin{aligned}&\langle t,\, d[\overline{v/x}, w/y]\rangle \Downarrow \\ \iff\ &\langle s,\, (\lambda y.\, d[\overline{v/x}]\ e)\rangle \Downarrow && \text{(by properties of evaluation)} \\ \iff\ &\langle s',\, (\lambda y.\, d[\overline{v'/x}]\ e')\rangle \Downarrow && ((s, s', R) \in (\cong)) \\ \iff\ &\langle t',\, d[\overline{v'/x}, w'/y]\rangle \Downarrow && \text{(by properties of evaluation)}\end{aligned}$$

Therefore, by Definitions 3.3.5 and 3.3.6:

$$(t, t', R \cup \{(w, w')\}) \in (\cong)$$

□

Theorem 3.3.12. $(\cong) = (\approx)$.

Proof. By Theorem 3.3.10 we have $(\approx) \subseteq (\cong)$ and by Theorem 3.3.11 we have $(\cong) \subseteq (\approx)$. Thus $(\cong) = (\approx)$. □

From the above we conclude that the open extension of adequacy coincides with the standard definition of contextual equivalence.

Theorem 3.3.13. $(\approx)^\circ = (\equiv)$.

Proof. By Theorems 3.3.7 and 3.3.12. □

3.4 Inductive Proofs of Equivalence

We can prove two expressions of the imperative, CBV λ-calculus, $\{\overline{x}\} \vdash e, e'$, contextually equivalent by the following proof method:

1. Find a set $\mathcal{X}$, such that for all stores s, there exists R with $(s, s, R) \in \mathcal{X}$, $\vdash \lambda\overline{x}.e\ R\ \lambda\overline{x}.e'$, and for all $l \in dom(s)$, $\vdash l\ R\ l$,
2. show that $\mathcal{X}$ is adequate, and
3. invoke Theorem 3.3.13 to show $\{\overline{x}\} \vdash e \equiv e'$.

As in the case of the CBV λ-calculus, we give an induction principle for proving $\mathcal{X}$ adequate. The induction hypothesis that we will use is the following.

Definition 3.4.1.

$$
\begin{aligned}
IH_{\mathcal{X}}(k) \stackrel{\text{def}}{=} \forall (s, s', R) \in \mathcal{X}.\ & \\
\forall e, e', t, w.\ & \vdash e\ R^{\mathsf{cxt}}\ e' \\
& \wedge\ \langle s,\, e\rangle \Downarrow^{k} \langle t,\, w\rangle \\
& \quad \Longrightarrow\ \exists t', w', Q.\ \langle s',\, e'\rangle \Downarrow \langle t',\, w'\rangle \\
& \qquad \wedge\ \vdash w\ Q^{\mathsf{cxt}}\ w' \\
& \qquad \wedge\ (t, t', Q) \in \mathcal{X} \\
& \qquad \wedge\ R \subseteq Q
\end{aligned}
$$

The measure of the induction is the size, k, of the evaluation $\langle s,\, e\rangle \Downarrow^k \langle t,\, w\rangle$. Hence, proving a set $\mathcal{X}$ of annotated relations adequate amounts to proving that for all k, $IH_{\mathcal{X}}(k)$ and $IH_{\mathcal{X}^\intercal}(k)$ hold. For $k = 0$ it is trivial; for $k > 0$ it can be shown by induction:

$$\boxed{\begin{array}{l}\forall k.\ IH_{\mathcal{X}}(k-1) \implies IH_{\mathcal{X}}(k)\\ \forall k.\ IH_{\mathcal{X}^\intercal}(k-1) \implies IH_{\mathcal{X}^\intercal}(k)\end{array}}$$

3.5 Deriving Smaller Proof Obligations for Adequate Sets

The two inductive proof obligations at the end of the preceding section have proofs that are mostly standard and independent of $\mathcal{X}$. We factor out the common parts of all instances of these proofs by a *proof construction scheme*—a hypothetical proof of the inductive proof obligations for an arbitrary set. By this method we discover smaller, necessary and sufficient, proof obligations of adequacy.

We show this process for the forward proof obligation; the reverse is symmetric. For all k we assume $IH_{\mathcal{X}}(k-1)$ holds. We need to prove that $IH_{\mathcal{X}}(k)$ also holds; i.e. we need to prove:

$$\begin{array}{l}\forall (s, s', R) \in \mathcal{X}. \qquad (3.6)\\ \quad \forall e, e', t, w.\ \vdash e\ R^{\mathsf{cxt}}\ e'\\ \qquad\qquad \wedge\ \langle s,\, e\rangle \Downarrow^k \langle t,\, w\rangle\\ \qquad\qquad\quad \implies \exists t', w', Q.\ \langle s',\, e'\rangle \Downarrow \langle t',\, w'\rangle\\ \qquad\qquad\qquad\qquad \wedge\ \vdash w\ Q^{\mathsf{cxt}}\ w'\\ \qquad\qquad\qquad\qquad \wedge\ (t, t', Q) \in \mathcal{X}\\ \qquad\qquad\qquad\qquad \wedge\ R \subseteq Q\end{array}$$

Let $(s, s', R) \in \mathcal{X}$, $\vdash e\ R^{\mathsf{cxt}}\ e'$, and $\langle s,\, e\rangle \Downarrow^k \langle t,\, w\rangle$. By Definition 3.3.3 we have:

$$e = d[\overline{u/x}] \qquad e' = d[\overline{u'/x}] \qquad \{\overline{x}\} \vdash d \qquad locs(d) = \emptyset \qquad \vdash \overline{u}\ R\ \overline{u'}$$

We take cases on d. For the cases where $d = x$, $d = c$, and $d = \lambda y.\, d_1$, (3.6) is trivially satisfied. For the case where $d = (\overline{d})$, the proof goes through entirely by the induction hypothesis.

For the case where $d = op(\overline{d})$ the proof goes through only when R on constants is the identity. Similar conditions are required for the rest of the cases. We write all these conditions as the following general one:

$$\boxed{\begin{array}{l} R \text{ relates values with the same top-level language con-} \\ \text{structor; i.e. (integer and boolean) constants are related} \\ \text{to themselves, procedures are related to procedures, lo-} \\ \text{cations are related to locations, and tuples are related to} \\ \text{tuples of same cardinality.} \end{array}} \tag{3.7}$$

Here we assume that each constant, each tuple constructor of different arity, and each allocation expression with different initial constant are all different language constructors.

For the case where $d = \texttt{if}\ d_1\ \texttt{then}\ d_2\ \texttt{else}\ d_3$ the proof goes through by condition (3.7) and the induction hypothesis.

For the case where $d = \pi_i(d_1)$ the proof goes through by condition (3.7), the induction hypothesis, and an extra condition that requires related tuples to have pairwise related parts; i.e:

$$\boxed{\begin{array}{l} \forall (s, s', R) \in \mathcal{X}. \\ \quad \forall \overline{v}, \overline{v'}.\ \vdash (\overline{v})\ R\ (\overline{v'}) \\ \qquad\qquad \Longrightarrow\ \exists Q.\ \vdash \overline{v}\ Q^{\mathsf{cxt}}\ \overline{v'} \\ \qquad\qquad\qquad\qquad \wedge\ (s, s', Q) \in \mathcal{X} \\ \qquad\qquad\qquad\qquad \wedge\ R \subseteq Q \end{array}} \tag{3.8}$$

For the case of application, where $d = (d_1\ d_2)$, the proof proceeds as for the corresponding case of the CBV λ-calculus in Section 2.5, and goes

through by condition (3.7), the induction hypothesis, and the following:

$$
\begin{array}{l}
\forall (s, s', R) \in \mathcal{X}. \\
\quad \forall e, e', v, v', t, w. \;\; \vdash \lambda x.\, e \; R \; \lambda x.\, e' \\
\qquad\qquad \wedge \;\; \vdash v \; R^{\mathsf{cxt}} \; v' \\
\qquad\qquad \wedge \;\; \langle s,\, (\lambda x.\, e \; v) \rangle \Downarrow^{k} \langle t,\, w \rangle \\
\qquad\qquad\quad \Longrightarrow \;\; \exists t', w', Q. \;\; \langle s',\, (\lambda x.\, e' v' \;) \rangle \Downarrow \langle t',\, w' \rangle \\
\qquad\qquad\qquad\qquad \wedge \;\; \vdash w \; Q^{\mathsf{cxt}} \; w' \\
\qquad\qquad\qquad\qquad \wedge \;\; (t, t', Q) \in \mathcal{X} \\
\qquad\qquad\qquad\qquad \wedge \;\; R \subseteq Q
\end{array}
\tag{3.9}
$$

For the case of allocation, where $d = \nu x := c.\, d_1$, the proof goes through by the induction hypothesis, and the condition:

$$
\begin{array}{l}
\forall (s, s', R) \in \mathcal{X}. \\
\quad \forall l \notin dom(s), c. \\
\qquad \exists l' \notin dom(s'), Q. \;\; \vdash l \; Q \; l' \\
\qquad\qquad\qquad \wedge \;\; (s[l \mapsto c], s'[l' \mapsto c], Q) \in \mathcal{X} \\
\qquad\qquad\qquad \wedge \;\; R \subseteq Q
\end{array}
\tag{3.10}
$$

For the case of dereferencing, where $d = !d_1$, the proof goes through by condition (3.7), the induction hypothesis, and the condition:

$$
\begin{array}{l}
\forall (s, s', R) \in \mathcal{X}. \\
\quad \forall l, l'. \;\; \vdash l \; R \; l' \\
\qquad\quad \Longrightarrow \;\; \exists Q. \;\; \vdash s(l) \; Q \; s'(l') \\
\qquad\qquad\qquad \wedge \;\; (s, s', Q) \in \mathcal{X} \\
\qquad\qquad\qquad \wedge \;\; R \subseteq Q
\end{array}
\tag{3.11}
$$

Finally, for the case of assignment, where $d = (d_1 := d_2)$, the proof goes

through by condition (3.7), the induction hypothesis, and the condition:

$$\boxed{\begin{array}{l}\forall (s, s', R) \in \mathcal{X}. \\ \quad \forall l, l', v, v'.\ \vdash l\ R\ l' \\ \qquad\qquad \wedge\ \vdash v\ R^{\mathsf{cxt}}\ v' \\ \qquad\qquad\quad \Longrightarrow\ \exists Q.\ (s[l \mapsto v], s'[l' \mapsto v'], Q) \in \mathcal{X} \\ \qquad\qquad\qquad\quad \wedge\ R \subseteq Q\end{array}} \tag{3.12}$$

We reformulate the above proof obligations as a theorem of adequate sets.

Theorem 3.5.1. *A set of annotated relations $\mathcal{X}$ is adequate if and only if for all k and all $(s, s', R) \in \mathcal{X}$, assuming that $IH_{\mathcal{X}}(k-1)$ holds, the following conditions are satisfied:*

1. *R relates values with the same top-level language constructor.*

2. *For all $\vdash (\overline{v})\ R\ (\overline{v'})$ and i, there exists $Q \supseteq R$ such that:*

$$\vdash v_i\ Q^{\mathsf{cxt}}\ v_i' \qquad (s, s', Q) \in \mathcal{X}$$

3. *For all $\vdash \lambda x.e\ R\ \lambda x.e'$, and all $\vdash v\ R^{\mathsf{cxt}}\ v'$, t, and w, such that $\langle s,\ (\lambda x.e\ v)\rangle \Downarrow^k \langle t,\ w\rangle$, there exist t', w', and $Q \supseteq R$ such that:*

$$\langle s',\ (\lambda x.e' v'\ )\rangle \Downarrow \langle t',\ w'\rangle \qquad \vdash w\ Q^{\mathsf{cxt}}\ w' \qquad (t, t', Q) \in \mathcal{X}$$

4. *For all $l \notin dom(s)$ and c, there exist $l' \notin dom(s')$ and $Q \supseteq R$ such that:*

$$\vdash l\ Q\ l' \qquad (s[l \mapsto c], s'[l' \mapsto c], Q) \in \mathcal{X}$$

5. *For all $\vdash l\ R\ l'$, there exists $Q \supseteq R$ such that:*

$$\vdash s(l)\ Q^{\mathsf{cxt}}\ s'(l') \qquad (s, s', Q) \in \mathcal{X}$$

6. *For all $\vdash l\ R\ l'$ and all $\vdash v\ R^{\mathsf{cxt}}\ v'$, there exists $Q \supseteq R$ such that:*

$$(s[l \mapsto v], s'[l' \mapsto v'], Q) \in \mathcal{X}$$

Moreover, the same conditions hold for $\mathcal{X}^{\mathsf{T}}$.

Proof. Forward direction: by showing that the conditions of the theorem are special cases of the proof obligation of the definition of adequate relations (Definition 3.3.8).

Reverse direction: by recapitulating the preceding proof construction scheme. □

The first condition of this theorem is both a weak well-typedness condition and a requirement that relations are the identity on constants. The second condition requires related tuples to contain pairwise related parts. The third condition ensures that related procedures map related arguments to related final values and stores (an argument similar to logical relations [43]. The last three conditions require sets to be closed under store extension, update, and dereferencing.

The conditions of this theorem are simpler to prove than the conditions in the definition of adequacy (Definition 3.3.8). The latter requires sets to be closed under evaluation of all programs containing related values, while the former requires only to be closed under each operation of the language.

Furthermore, the theorem assumes the induction hypothesis for smaller computations and expressions. This can be useful for proving Condition 3, which involves computations, especially in the case of higher-order procedures.

3.6 Examples

Here we use the method described in Section 3.4 to prove several interesting equivalences.[3]

[3]All the examples in this section can also be proved by the Label Transition System recently proposed by Laird [31] and by Sangiorgi et al. [49].

$$\frac{}{([],[],\{(Y_{\text{val}}, Y!)\}) \in \mathcal{X}}$$

$$\frac{(s,s',R) \in \mathcal{X} \qquad \overline{l} \notin dom(s) \qquad \overline{l'} \notin dom(s') \qquad Q = R \uplus \{\overline{(l,l')}\} \qquad \vdash \overline{u}\ Q^{\mathsf{cxt}}\ \overline{u'}}{(s[\overline{l \mapsto u}], s'[\overline{l' \mapsto u'}], Q) \in \mathcal{X}}$$

$$\frac{\begin{array}{c}(s,s',R) \in \mathcal{X} \qquad l' \notin dom(s') \qquad \vdash v\ R^{\mathsf{cxt}}\ v' \\ Q = R \cup \{(\lambda y.((v\ \lambda z.((\Delta_v\ \Delta_v)\ z))\ y),\ \lambda y.((v'\ !l')\ y))\}, \\ (\lambda z.((\Delta_v\ \Delta_v)\ z),\ \lambda y.((v'\ !l')\ y))\end{array}}{(s, s'[l' \mapsto \lambda y.((v'\ !l')\ y)], Q) \in \mathcal{X}}$$

Figure 3.4: Construction of adequate set of annotated relations for proving the equivalence of Y_{val} and $Y!$

3.6.1 Imperative Fixpoint Combinator

Our first example is the equivalence between Church's fixpoint[4] Y_{val}, and Landin's imperative fixpoint combinator, $Y!$.

$$Y_{\text{val}} \stackrel{\text{def}}{=} \lambda f.(\Delta_f\ \Delta_f) \qquad \Delta_f \stackrel{\text{def}}{=} \lambda x.\lambda y.((f\ \lambda z.((x\ x)\ z))\ y)$$
$$Y! \stackrel{\text{def}}{=} \lambda f.\nu x{:=}0.(x := \lambda y.((f\ !x)\ y));\ !x$$

Proof. We need to construct a set $\mathcal{X}$ with the property that for all stores s, there exists R such that

$$(s,s,R) \in \mathcal{X} \qquad \vdash Y_{\text{val}}\ R\ Y! \qquad \forall l \in dom(s).\ \vdash l\ R\ l$$

We then need to show that $\mathcal{X}$ is adequate.

We construct $\mathcal{X}$ by induction; we start by the first two rules of Figure 3.4. The first relates Y_{val} and $Y!$ under the empty stores in the base case. The second rule propagates relations to larger related stores. This rule admits more relations than the ones annotated with identical stores; this is to satisfy

[4]The version of Church's fixpoint combinator in Section 2.7.1 is not equivalent to $Y!$ in the CBV setting; the context $([\cdot]\ \lambda g.\Omega)$ distinguishes them. For this reason we use here a version of Y_{val} that contains an extra η-expansion.

the last three conditions of Theorem 3.5.1. Moreover, $\mathcal{X}$ satisfies the first two conditions of the theorem.

Thus, we only need to verify the third condition of Theorem 3.5.1 for $\mathcal{X}$ and $\mathcal{X}^{\mathsf{T}}$. We show only the case for $\mathcal{X}$; the case for $\mathcal{X}^{\mathsf{T}}$ is analogous.

Let $(s, s', R) \in \mathcal{X}$. The only procedures related by the first two inductive rules of Figure 3.4 are Y_{val} and $Y!$. Let $IH_{\mathcal{X}}(k-1)$ be valid, $\vdash v\ R^{\mathsf{cxt}}\ v'$, and $\langle s,\ (Y_{\text{val}}\ v)\rangle \Downarrow^k \langle t,\ w\rangle$. We need to show that there exist Q, w', and t' such that:

$$\langle s',\ (Y!\ v')\rangle \Downarrow \langle t',\ w'\rangle \qquad \vdash w\ Q^{\mathsf{cxt}}\ w' \qquad (t, t', Q) \in \mathcal{X} \qquad R \subseteq Q$$

By the properties of evaluation we have:

$$\begin{aligned} & \langle s,\ (Y_{\text{val}}\ v)\rangle \Downarrow^k \langle t,\ w\rangle \\ \iff & \langle s,\ (\Delta_v\ \Delta_v)\rangle \Downarrow^{k-1} \langle t,\ w\rangle \\ \iff & \langle s,\ \lambda y.((v\ \lambda z.((\Delta_v\ \Delta_v)\ z))\ y)\rangle \Downarrow^{k-2} \langle t,\ w\rangle \end{aligned}$$

Hence $t = s$ and $w = \lambda y.((v\ \lambda z.((\Delta_v\ \Delta_v)\ z))\ y)$.

Moreover, we have:

$$\begin{aligned} & \langle s',\ (Y!\ v')\rangle \Downarrow \langle t',\ w'\rangle \\ \iff & \langle s',\ \nu x{:=}0.(x := \lambda y.((v'\ !x)\ y));\ !x\rangle \Downarrow \langle t',\ w'\rangle \\ \iff & \langle s'[l'{\mapsto}0],\ (l' := \lambda y.((v'\ !l')\ y));\ !l'\rangle \Downarrow \langle t',\ w'\rangle \qquad (l' \notin dom(s')) \\ \iff & \langle s'[l'{\mapsto}\lambda y.((v'\ !l')\ y)],\ \lambda y.((v'\ !l')\ y)\rangle \Downarrow \langle t',\ w'\rangle \end{aligned}$$

Thus, $t' = s'[l'{\mapsto}\lambda y.((v'\ !l')\ y)]$ and $w' = \lambda y.((v'\ !l')\ y)$. Therefore:

$$\langle s',\ (Y!\ v')\rangle \Downarrow \langle s'[l'{\mapsto}\lambda y.((v'\ !l')\ y)],\ \lambda y.((v'\ !l')\ y)\rangle \qquad l' \notin dom(s')$$

To satisfy the rest of the condition for the two applications we include the third and final inductive rule of Figure 3.4 in the construction of $\mathcal{X}$. This rule relates the final values of the above computations under the appropriate stores. Namely, it relates $\lambda y.((v\ \lambda z.((\Delta_v\ \Delta_v)\ z))\ y)$ to $\lambda y.((v'\ !l')\ y)$, under the stores s and $s'[l'{\mapsto}((v'\ !l')\ y)]$.[5]

[5]The last rule of the construction of $\mathcal{X}$ also relates $\lambda z.((\Delta_v\ \Delta_v)\ z)$ to $\lambda y.((v'\ !l')\ y)$, something that we will use further down in the proof.

We now have to prove the third condition of Theorem 3.5.1 for the procedures related by the final rule of the construction of $\mathcal{X}$.

Let:

$$(s, s', R) \in \mathcal{X} \qquad l' \notin dom(s') \qquad \vdash v\ R^{\mathsf{cxt}}\ v'$$
$$Q = R \cup \{(\lambda y.((v\ \lambda z.((\Delta_v\ \Delta_v)\ z))\ y), \lambda y.((v'\ !l')\ y))\}$$
$$\cup \{(\lambda z.((\Delta_v\ \Delta_v)\ z), \lambda y.((v'\ !l')\ y))\}$$

By construction of $\mathcal{X}$, $(s, s'[l' \mapsto \lambda y.((v'\ !l')\ y)], Q) \in \mathcal{X}$.

Case $\vdash \lambda y.((v\ \lambda z.((\Delta_v\ \Delta_v)\ z))\ y)\ Q\ \lambda y.((v'\ !l')\ y)$

Let $IH_{\mathcal{X}}(k-1)$ be valid, $\vdash u\ Q^{\mathsf{cxt}}\ u'$ and

$$\langle s,\ (\lambda y.((v\ \lambda z.((\Delta_v\ \Delta_v)\ z))\ y)\ u)\rangle \Downarrow^k \langle t,\ w\rangle$$

We need to show that there exist P, w', t', such that

$$\langle s'[l' \mapsto \lambda y.((v'\ !l')\ y)],\ (\lambda y.((v'\ !l')\ y)\ u')\rangle \Downarrow \langle t',\ w'\rangle$$
$$\vdash w\ P^{\mathsf{cxt}}\ w' \qquad (t, t', P) \in \mathcal{X} \qquad Q \subseteq P$$

By the properties of evaluation we have:

$$\langle s,\ (\lambda y.((v\ \lambda z.((\Delta_v\ \Delta_v)\ z))\ y)\ u)\rangle \Downarrow^k \langle t,\ w\rangle$$
$$\iff \langle s,\ ((v\ \lambda z.((\Delta_v\ \Delta_v)\ z))\ u)\rangle \Downarrow^{k-1} \langle t,\ w\rangle$$

and

$$\langle s'[l' \mapsto \lambda y.((v'\ !l')\ y)],\ (\lambda y.((v'\ !l')\ y)\ u')\rangle \Downarrow \langle t',\ w'\rangle$$
$$\iff \langle s'[l' \mapsto \lambda y.((v'\ !l')\ y)],\ ((v'\ !l')\ u')\rangle \Downarrow \langle t',\ w'\rangle$$
$$\iff \langle s'[l' \mapsto \lambda y.((v'\ !l')\ y)],\ ((v'\ \lambda y.((v'\ !l')\ y))\ u')\rangle \Downarrow \langle t',\ w'\rangle$$

Since

$$\vdash v\ Q^{\mathsf{cxt}}\ v' \qquad \text{(by Lemma 3.3.4 (1))}$$
$$\vdash \lambda z.((\Delta_v\ \Delta_v)\ z)\ Q\ \lambda y.((v'\ !l')\ y)$$
$$\vdash u\ Q^{\mathsf{cxt}}\ u'$$

we have:

$$\vdash ((v\ \lambda z.((\Delta_v\ \Delta_v)\ z))\ u)\ Q^{\mathsf{cxt}}\ ((v'\ \lambda y.((v'\ !l')\ y))\ u')$$

Hence, we can use $IH_{\mathcal{X}}(k-1)$ to conclude this case of the proof.

Case $\vdash \lambda z.((\Delta_v\ \Delta_v)\ z)\ Q\ \lambda y.((v'\ !l')\ y)$

Let $IH_{\mathcal{X}}(k-1)$ be valid, $\vdash u\ Q^{\mathsf{cxt}}\ u'$ and

$$\langle s,\ (\lambda z.((\Delta_v\ \Delta_v)\ z)\ u)\rangle \Downarrow^k \langle t,\ w\rangle$$

We need to show that there exist P, w', t', such that

$$\langle s'[l'\mapsto\lambda y.((v'\ !l')\ y)],\ (\lambda y.((v'\ !l')\ y)\ u')\rangle \Downarrow \langle t',\ w'\rangle$$
$$\vdash w\ P^{\mathsf{cxt}}\ w' \qquad (t,t',P)\in\mathcal{X} \qquad Q\subseteq P$$

By the properties of evaluation we have:

$$\begin{aligned}&\langle s,\ (\lambda z.((\Delta_v\ \Delta_v)\ z)\ u)\rangle \Downarrow^k \langle t,\ w\rangle\\ \iff &\langle s,\ ((\Delta_v\ \Delta_v)\ u)\rangle \Downarrow^{k-1} \langle t,\ w\rangle\\ \iff &\langle s,\ ((v\ \lambda z.((\Delta_v\ \Delta_v)\ z))\ u)\rangle \Downarrow^{k-1} \langle t,\ w\rangle\end{aligned}$$

and

$$\begin{aligned}&\langle s'[l'\mapsto\lambda y.((v'\ !l')\ y)],\ (\lambda y.((v'\ !l')\ y)\ u')\rangle \Downarrow \langle t',\ w'\rangle\\ \iff &\langle s'[l'\mapsto\lambda y.((v'\ !l')\ y)],\ ((v'\ \lambda y.((v'\ !l')\ y))\ u')\rangle \Downarrow \langle t',\ w'\rangle\end{aligned}$$

As in the previous case of the proof, we have:

$$\vdash ((v\ \lambda z.((\Delta_v\ \Delta_v)\ z))\ u)\ Q^{\mathsf{cxt}}\ ((v'\ \lambda y.((v'\ !l')\ y))\ u')$$

Hence, by $IH_{\mathcal{X}}(k-1)$, we conclude that $\mathcal{X}$ is adequate. By Theorem 3.3.13 we get $\vdash Y_{\text{val}} \equiv Y!$. □

3.6.2 Meyer and Sieber's Examples

By using the method of Section 3.4 we are able to prove all the equivalences given by Meyer and Sieber for Algol-like languages [35], adapted to the

$$\frac{}{([],[],\{(M,M')\}) \in \mathcal{X}}$$

$$\frac{(s,s',R) \in \mathcal{X} \qquad \overline{l} \notin dom(s) \qquad \overline{l'} \notin dom(s') \qquad Q = R \uplus \{\overline{(l,l')}\} \qquad \vdash \overline{u}\ Q^{\mathsf{cxt}}\ \overline{u'}}{(s[\overline{l \mapsto u}], s'[\overline{l' \mapsto u'}], Q) \in \mathcal{X}}$$

$$\frac{(s,s',R) \in \mathcal{X} \qquad l \notin dom(s) \qquad Q = R \cup \{(\lambda z.(l := !l + 2), \lambda z.0)\} \qquad n \in \mathbb{N}}{(s[l \mapsto 2n], s', Q) \in \mathcal{X}}$$

Figure 3.5: Construction of adequate set of annotated relations for proving the equivalence of Meyer and Sieber's higher-order procedures.

imperative CBV λ-calculus of this section. Here we show the proof of the most intricate of these equivalences.

$$\begin{aligned} M \stackrel{\text{def}}{=} \lambda g.\, \nu x := 0.\ & \\ & \texttt{let}\ f = \lambda z.(x := !x + 2) \\ & \texttt{in}\,((g\ f); \\ & \quad \texttt{if even}(!x)\ \texttt{then 0 else}\ \Omega) \\ M' \stackrel{\text{def}}{=} \lambda g.((g\ \lambda x.0); 0) & \end{aligned}$$

M applies its argument to a procedure that, when called, increments a local location by two. After this application terminates M diverges if this location contains anything other than an even number. The second procedure applies its argument to a dummy procedure and immediately returns. M and M' are contextually equivalent since the only effect that the context can have on the location allocated by M is to arbitrarily increment it by two.

Proof. To prove this equivalence we construct a set of annotated relations, $\mathcal{X}$, by induction. The construction is shown in Figure 3.5.

$\mathcal{X}$ satisfies the property that for all stores s there exists R such that:

$$(s, s, R) \in \mathcal{X} \qquad \vdash M\ R\ M' \qquad \forall l \in dom(s).\ \vdash l\ R\ l$$

We prove that $\mathcal{X}$ is adequate by showing that it satisfies the conditions of Theorem 3.5.1. We show only the proof of the conditions involving $\mathcal{X}$; the proof for $\mathcal{X}^{\mathsf{T}}$ is analogous.

The first two conditions of the theorem are satisfied by the construction of $\mathcal{X}$. Similarly, the last three conditions of the theorem are satisfied due to the second rule in the construction of $\mathcal{X}$. We only need to prove the third condition of the theorem for the procedures related in annotated relations from $\mathcal{X}$.

We initially prove this condition for all $(s, s', R) \in \mathcal{X}$ and $\vdash M\ R\ M'$. We have that $IH_{\mathcal{X}}(k-1)$ is valid, $\vdash v\ R^{\mathsf{cxt}}\ v'$, and $\langle s,\ (M\ v)\rangle \Downarrow^k \langle t,\ w\rangle$. We need to show that there exist Q, t', and w', such that:

$$\langle s',\ (M'\ v')\rangle \Downarrow \langle t',\ w'\rangle \qquad \vdash w\ Q^{\mathsf{cxt}}\ w' \qquad (t, t', Q) \in \mathcal{X} \qquad R \subseteq Q$$

By following the same course as in the proof of the previous section, we have that the above is true if, for all $l \notin dom(s)$, whenever

$$\langle s[l{\mapsto}0],\ (v\ \lambda z.(l := !l + 2))\rangle \Downarrow^{k-1} \langle t,\ w\rangle$$

there exist Q, t', and w', such that:

$$\langle s',\ (v'\ \lambda z.0)\rangle \Downarrow \langle t',\ w'\rangle \qquad \vdash w\ Q^{\mathsf{cxt}}\ w' \qquad (t, t', Q) \in \mathcal{X} \qquad R \subseteq Q$$

But this is true due to the third rule of the construction of $\mathcal{X}$ and the validity of $IH_{\mathcal{X}}(k-1)$.

We now need to prove Condition 3 of Theorem 3.5.1 for all $(s[l{\mapsto}2n], s', Q) \in \mathcal{X}$ and $\vdash \lambda z.(l := !l + 2)\ Q\ \lambda z.0$. We have that $IH_{\mathcal{X}}(k-1)$ is valid, $\vdash v\ R^{\mathsf{cxt}}\ v'$, and $\langle s[l{\mapsto}2n],\ (\lambda z.(l := !l + 2)\ v)\rangle \Downarrow^k \langle t,\ w\rangle$. It must be that $t = s[l{\mapsto}2(n+1)]$ and $w = 0$. Furthermore, $\langle s',\ (\lambda z.0\ v')\rangle \Downarrow \langle s',\ 0\rangle$.

Hence, Condition 3 is satisfied, since $(s[l{\mapsto}2(n+1)], s', Q) \in \mathcal{X}$ and $\vdash 0\ Q^{\mathsf{cxt}}\ 0$.

Therefore, $\mathcal{X}$ is adequate and by Theorem 3.3.13 we get $\vdash M \equiv M'$. $\square$

3.6.3 Two Cell Implementations

Our technique can also deal with the structures underlying object-oriented programs.[6] For example, here are two models of a Cell object.

$$M \stackrel{\text{def}}{=} \lambda x.\, \nu y := 0.\, (set(y), get(y))$$
$$get(y) = \lambda z.\, (y := z)$$
$$set(y) = \lambda z.\, !y$$

$$M' \stackrel{\text{def}}{=} \lambda x.\, \nu y_1 := 0.\, \nu y_2 := 0.\, \nu p := 0.\, (set'(\overline{y}), get'(\overline{y}))$$
$$set'(\overline{y}) \stackrel{\text{def}}{=} \lambda z.\, ((p := !p + 1);\ \texttt{if even}(!p)\ \texttt{then}\ (y_1 := z)\ \texttt{else}\ (y_2 := z))$$
$$get'(\overline{y}) \stackrel{\text{def}}{=} \lambda z.\, \texttt{if even}(!p)\ \texttt{then}\ !y_1\ \texttt{else}\ !y_2$$

When M and M' are applied, they allocate some local store and return a setter and a getter—essentially creating a cell object. M allocates one location, which is used by the setter and getter to store and retrieve a value, respectively. M' allocates three locations. One of these locations is used as a counter, incremented by the setter. When the counter is even the setter stores the value to the first of the other locations and when it is odd to the second. The getter of M' returns the value stored in the appropriate location, based on the value of the counter.

Proof. To prove the equivalence of the two cell implementations we construct the set of annotated relations shown in Figure 3.6. As in the previous examples, the first rule of the construction relates M and M' under the empty stores, and the second rule extends annotated relations under any possible related stores.

The last two rules encode the part of the relations and stores that are due to applications of M and M', and their corresponding getters and setters by the context. Each of the two rules encodes the possible situations in the store: the first is the "world" where the counter in the right-hand side is

[6] We study object-based and class-based languages in more detail in Chapters 5 and 6.

$$\frac{}{([],[],\{(M,M')\}) \in \mathcal{X}}$$

$$\frac{(s,s',R) \in \mathcal{X} \qquad \overline{l} \notin dom(s) \qquad \overline{l'} \notin dom(s') \qquad Q = R \uplus \{\overline{(l,l')}\} \qquad \vdash \overline{u}\ Q^{\mathsf{cxt}}\ \overline{u'}}{(s[\overline{l \mapsto u}], s'[\overline{l' \mapsto u'}], Q) \in \mathcal{X}}$$

$$\frac{\begin{array}{c}(s,s',R) \in \mathcal{X} \qquad l \notin dom(s) \qquad \overline{l'} \notin dom(s') \\ \vdash v\ Q^{\mathsf{cxt}}\ v' \qquad \text{value } u' \qquad n \in I\!N \\ Q = R \cup \{(set(l), set'(\overline{l'})), (get(l), get'(\overline{l'}))\}\end{array}}{(s[l \mapsto v], s'[\boldsymbol{l'_1 \mapsto 2n}, \boldsymbol{l'_2 \mapsto v'}, \boldsymbol{l'_3 \mapsto u'}], Q) \in \mathcal{X}}$$

$$\frac{\begin{array}{c}(s,s',R) \in \mathcal{X} \qquad l \notin dom(s) \qquad \overline{l'} \notin dom(s') \\ \vdash v\ Q^{\mathsf{cxt}}\ v' \qquad \text{value } u' \qquad n \in I\!N \\ Q = R \cup \{(set(l), set'(\overline{l'})), (get(l), get'(\overline{l'}))\}\end{array}}{(s[l \mapsto v], s'[\boldsymbol{l'_1 \mapsto 2n+1}, \boldsymbol{l'_2 \mapsto u'}, \boldsymbol{l'_3 \mapsto v'}], Q) \in \mathcal{X}}$$

Figure 3.6: Construction of adequate set of annotated relations for proving the equivalence of two cell implementations in the imperative CBV λ-calculus.

even and v' is stored in l'_2, and the second is the "world" where the counter is odd and v' is stored in l'_3. This difference is shown in bold typeface in Figure 3.6. Here u' is any value previously stored by the cell.

It is easy to show that $\mathcal{X}$ satisfies the third condition of Theorem 3.5.1 for all related procedures. All of these procedures trivially terminate, and they do so in related stores. The rest of the conditions of the theorem are satisfied by construction. Hence, by Theorem 3.3.13 we get $\vdash M \equiv M'$. $\square$

3.6.4 Diverging Higher-Order Procedures

This example is due to Nina Bohr, Lars Birkedal, and Hongseong Yang [13].[7]

$$
\begin{aligned}
M \stackrel{\text{def}}{=} \lambda g.\, &\nu x := \mathtt{0}.\, \nu y := \mathtt{0}. \\
&((g\ \lambda k.\, \mathtt{if\ zero?}(!x)\ \mathtt{then}\ (y := \mathtt{1})\ \mathtt{else}\ \Omega); \\
&\ \mathtt{if\ zero?}(!y)\ \mathtt{then}\ (x := \mathtt{1})\ \mathtt{else}\ \Omega) \\
M' \stackrel{\text{def}}{=} \lambda g.\, &((g\ \lambda k.\, \Omega);\ \mathtt{0})
\end{aligned}
$$

Here, the argument g supplied to the procedures, after being applied to some argument, will have one of the following behaviors:

1. It diverges (or gets stuck) without calling its argument.

2. It terminates without calling its argument, possibly after storing its argument in the store.

3. It calls its argument.

If g follows strategy 1 or 3, then both sides will diverge. If g follows strategy 2, then M assigns $\mathtt{1}$ to y and returns $\mathtt{0}$. From that point on, $(\lambda k.\, \mathtt{if\ zero?}(!x)\ \mathtt{then}\ (y := \mathtt{1})\ \mathtt{else}\ \Omega)$ is equivalent to $\lambda k.\, \Omega$. Hence, if g stored its argument and the context attempts to apply it after M returns, the two programs will diverge.

To prove $\vdash M \equiv M'$ by the method of Section 3.4 we need to prove the above behavior of g independently. We can do this by following a method similar to the proof of Theorem 2.5.24 in [37].

This equivalence can also be shown by using denotational logical relations [13] and labeled transition systems that encode the interaction between terms and context [31].

Proof. (Sketch.) To prove this equivalence we construct the set of relations,

[7] A proof similar to the one here, using the bisimulation in [49], was proposed by Eijiro Sumii.

$$\frac{}{([],[],\{(M,M')\}) \in \mathcal{X}}$$

$$\frac{(s,s',R) \in \mathcal{X} \qquad \overline{l} \notin dom(s) \qquad \overline{l'} \notin dom(s') \qquad Q = R \uplus \{\overline{(l,l')}\} \qquad \vdash \overline{u}\ Q^{\mathsf{cxt}}\ \overline{u'}}{(s[\overline{l \mapsto u}], s'[\overline{l' \mapsto u'}], Q) \in \mathcal{X}}$$

$$\frac{(s,s',R) \in \mathcal{X} \qquad l_x, l_y \notin dom(s)}{(s[l_x \mapsto 0, l_y \mapsto 0], s', Q) \in \mathcal{X}}$$

$$\frac{(s,s',R) \in \mathcal{X} \qquad l_x, l_y \notin dom(s) \qquad Q = R \cup \{(\lambda k.\,\texttt{if zero?}(!l_x)\ \texttt{then}\ (l_y := 1)\ \texttt{else}\ \Omega, \lambda k.\,\Omega)\}}{(s[l_x \mapsto 1, l_y \mapsto 0], s', Q) \in \mathcal{X}}$$

Figure 3.7: Construction of adequate set of annotated relations for proving the equivalence of two diverging higher-order procedures.

$\mathcal{X}$, shown in Figure 3.7. We show that $\mathcal{X}$ is adequate by showing that it satisfies the conditions of Theorem 3.5.1.

The first two conditions of the theorem are trivially satisfied. Conditions 4-6 are satisfied due to the second rule in the construction of $\mathcal{X}$. We need to prove the third condition of the theorem for all related procedures in $\mathcal{X}$.

Let:

$$(s,s',R) \in \mathcal{X} \qquad \vdash M\ R\ M' \qquad \vdash v\ R^{\mathsf{cxt}}\ v' \qquad IH_{\mathcal{X}}(k-1)$$
$$\langle s,\ (M\ v)\rangle \Downarrow^k \langle t,\ w\rangle$$

From the properties of evaluation we have:

$$\langle s,\ (M\ v)\rangle \Downarrow^k \langle t,\ w\rangle$$
$$\langle s[l_x \mapsto 0, l_y \mapsto 0],\ ((v\ \lambda k.\,\texttt{if zero?}(!l_x)\ \texttt{then}\ (l_y := 1)\ \texttt{else}\ \Omega);$$
$$\texttt{if zero?}(!l_y)\ \texttt{then}\ (l_x := 1)\ \texttt{else}\ \Omega)\rangle \Downarrow^{k-1} \langle t,\ w\rangle$$

Thus, for some t_0 and w_0:

$$\langle s[l_x \mapsto 0, l_y \mapsto 0],\ (v\ \lambda k.\,\texttt{if zero?}(!l_x)\ \texttt{then}\ (l_y := 1)\ \texttt{else}\ \Omega)\rangle \Downarrow^{k-1} \langle t_0,\ w_0\rangle$$

As we discussed above this application can have three possible behaviors:

Case 1: The application diverges (or gets stuck) without applying the argument. This is a contradiction, since we assumed that the application terminates.

Case 2: The application terminates without applying the argument. Since $l_x, l_y \notin locs(v)$, there will be t_1 such that $t_0 = t_1[l_x \mapsto 0, l_y \mapsto 0]$. Furthermore, since the computation does not depend on the argument:

$$\langle s[l_x \mapsto 0, l_y \mapsto 0],\ (v\ \lambda k.\Omega)\rangle \Downarrow^{k-1} \langle t_2[l_x \mapsto 0, l_y \mapsto 0],\ w_2\rangle$$
$$dom(t_1) = dom(t_2)$$
$$\forall l \in dom(t_1).\ \vdash t_1(l)\ S^{\mathsf{cxt}}\ t_2(l)$$

for some t_2, w_2, and $S = \{(\lambda k.\,\texttt{if zero?}(!l_x)\ \texttt{then}\ (l_y := \texttt{1})\ \texttt{else}\ \Omega, \lambda k.\Omega)\}$.

By $IH_{\mathcal{X}}(k-1)$ we also get that there exist t_2', w_2', and Q:

$$\langle s',\ (v'\ \lambda k.\Omega)\rangle \Downarrow \langle t_2',\ w_2'\rangle \qquad R \subseteq Q \qquad (t_2[l_x \mapsto 0, l_y \mapsto 0], t_2', Q) \in \mathcal{X}$$

By construction of $\mathcal{X}$ we have:

$$t_2 = [\overline{l \mapsto v}, \overline{l_x \mapsto i, l_y \mapsto 0}] \qquad t_2' = [\overline{l' \mapsto v'}]$$
$$\vdash \overline{l}\ Q\ \overline{l'} \qquad \vdash \overline{v}\ Q^{\mathsf{cxt}}\ \overline{v'} \qquad \overline{i} \in \{0, 1\}$$

and thus:

$$t_1 = [\overline{l \mapsto u}, \overline{l_x \mapsto i, l_y \mapsto 0}] \qquad \vdash \overline{u}\ (Q \cup S)^{\mathsf{cxt}}\ \overline{v'}$$

By the third rule of the construction of $\mathcal{X}$ we have:

$$(t_2[l_x \mapsto 1, l_y \mapsto 0], t_2', Q \cup S) \in \mathcal{X}$$

and by updating, via the second rule of the construction of $\mathcal{X}$, the contents of all $\vdash \overline{l}\ Q\ \overline{l'}$ with the appropriate values $\vdash \overline{u}\ (Q \cup S)^{\mathsf{cxt}}\ \overline{v'}$:

$$(t_1[l_x \mapsto 1, l_y \mapsto 0], t_2', Q \cup S) \in \mathcal{X}$$

Hence, we established that in this case both the applications will terminate in related stores:

$$\langle s,\ (M\ v)\rangle \Downarrow^k \langle t_1[l_x{\mapsto}1, l_y{\mapsto}0],\ 0\rangle \qquad \langle s',\ (M'\ v')\rangle \Downarrow \langle t_2',\ 0\rangle$$
$$(t_1[l_x{\mapsto}1, l_y{\mapsto}0], t_2', Q \cup S) \in \mathcal{X}$$

Case 3: The application terminates and the argument is applied. In this case there exists t_1, such that $t_0 = t_1[l_x{\mapsto}0, l_y{\mapsto}1]$. Thus, the term $(M\ v)$ will diverge, which is a contradiction.

We now need to prove Condition 3 of Theorem 3.5.1 for the procedures introduced by the forth rule of the construction of $\mathcal{X}$. This is easily done, since they are only related under stores $s[l_x{\mapsto}10]$ and s', therefore they both diverge when applied to any argument.

With a similar argument we prove the conditions involving $\mathcal{X}^{\mathsf{T}}$.

□

CHAPTER 4

The Nu-Calculus

In this chapter we develop our theory of contextual equivalence for the ν-calculus of Pitts and Stark [42, 53], a λ-calculus with first-class names. Effects here are minimal: the language only allows fresh generation of names, which are values and can be tested for equality. No other effect, including divergence, is allowed.

This is a language with features similar to the imperative, CBV λ-calculus of the previous chapter, but more restricted. Thus the development of our theory for each of the two languages is also similar. What makes the study of contextual equivalence in the ν-calculus interesting is that its restricted effects give rise to various subtle and hard-to-prove equivalences. The canonical example of such an equivalence is discussed in Section 4.6.2, where we show how it can be proved using our technique.

4.1 Language and Semantics

The ν-calculus is a simply-typed lambda calculus over base types of names and booleans, extended with a conditional construct and operations for generating and comparing names. The expression `new` generates a fresh name, and $(n_1 = n_2)$ returns `true` when n_1 and n_2 are the same name. We often

TYPE:	T	$::= o$	Boolean
		$\mid \nu$	Name
		$\mid T \to T$	Function
EXPRESSION:	e	$::= x$	Identifier
		$\mid n$	Name
		$\mid$ `true` $\mid$ `false`	Boolean Constants
		$\mid \lambda x{:}T.\, e$	Abstraction
		$\mid e\ e$	Application
		$\mid$ `new`	Fresh Name Generation
		$\mid (e{=}e)$	Name Equality
		$\mid$ `if` e `then` e `else` e	Conditional
VALUE:	u, v, w	$::= n \mid$ `true` $\mid$ `false`	
		$\mid \lambda x{:}T.\, e$	
NAME:	n		
NAMESET:	s, t	$\in \mathscr{F}(\text{NAME})$	

Figure 4.1: Syntactic domains of the ν-calculus

write $\nu x.\, e$ as an abbreviation of the expression $(\lambda x{:}\nu.\, e)$ `new`.[1] We also write $(e{=}e')$, when e and e' have type bool, as syntactic sugar for:

$$(\lambda x{:}o.\, \lambda y{:}o.\, \texttt{if}\ x\ \texttt{then}\ y\ \texttt{else}\ (\texttt{if}\ y\ \texttt{then false else true}))\ e\ e'$$

Names are drown from an infinite (countable) set NAME, of which finite subsets are called *namesets*. We write $s \oplus t$ for the disjoint union of namesets s and t. All syntactic domains of the ν-calculus are shown in Figure 4.1.

The typing judgment $s; E \vdash e : T$ says that the expression e has type T under the nameset s and typing environment E. The typing rules are standard and shown in Figure 4.2.

We write $\lambda\overline{x{:}T_x}.\, e$ for the procedure $\lambda x_1{:}T_{x_1}.\, \ldots \lambda x_n{:}T_{x_n}.\, e$ and $\overline{T_x} \to T$ for the type $T_{x_1} \to \ldots \to T_{x_n} \to T$.

The evaluation judgment $s \vdash e \Downarrow^k (t)\, w$, says that the closed, well-typed expression e, under the nameset s, terminates with the value w producing a

[1] Pitts and Stark take $\nu x.\, e$ as primitive and define `new` as $\nu x.\, x$—the presentations are entirely equivalent.

$\boxed{s; \Gamma \vdash e : T}$

$$\textsc{Typ-Var}\ \frac{x : T \in \Gamma}{s; \Gamma \vdash x : T} \qquad \textsc{Typ-Name}\ \frac{n \in s}{s; \Gamma \vdash n : \nu} \qquad \textsc{Typ-Bool}\ \frac{b \in \{\texttt{true}, \texttt{false}\}}{s; \Gamma \vdash b : o}$$

$$\textsc{Typ-Abs}\ \frac{s; \Gamma, x : T_1 \vdash e : T_2}{s; \Gamma \vdash \lambda x{:}T_1.\, e : T_1 \to T_2} \qquad \textsc{Typ-App}\ \frac{s; \Gamma \vdash e_0 : T_1 \to T_2 \qquad s; \Gamma \vdash e_1 : T_1}{s; \Gamma \vdash e_0\ e_1 : T_2}$$

$$\textsc{Typ-Cond}\ \frac{s; \Gamma \vdash e_0 : o \qquad s; \Gamma \vdash e_1 : T \qquad s; \Gamma \vdash e_2 : T}{s; \Gamma \vdash \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : T}$$

$$\textsc{Typ-New}\ \frac{}{s; E \vdash \texttt{new} : \nu} \qquad \textsc{Typ-Eq}\ \frac{s; E \vdash e_1 : \nu \qquad s; E \vdash e_2 : \nu}{s; E \vdash (e_1 = e_2) : o}$$

Figure 4.2: Typing rules for the ν-calculus

set of fresh names t. The *size* of the evaluation tree is less than k. We write $s \vdash e \Downarrow (t)\, w$ when there exists some k for which $s \vdash e \Downarrow^k (t)\, w$. Figure 4.3 shows the evaluation rules of the ν-calculus.

Evaluation is total and deterministic, modulo fresh name generation. It is also stable under the addition and removal of unused names.

Lemma 4.1.1. *If $s \vdash e \Downarrow (t_1)\, b_1$, $s \vdash e \Downarrow (t_2)\, b_2$, and $\emptyset; \cdot \vdash b_i : o$ $(i = 1, 2)$ then* $b_1 = b_2$.

Lemma 4.1.2. *If $s; \cdot \vdash e : T$ then there exist t and w such that $s \vdash e \Downarrow (t)\, w$.*

Lemma 4.1.3 (Garbage Addition)**.** *If $s \vdash e \Downarrow^k (t)\, w$ and $s \cap s_0 = t \cap s_0 = \emptyset$ then:*

$$s \oplus s_0 \vdash e \Downarrow^k (t)\, w.$$

$\boxed{s \vdash e \Downarrow^k (t)\, w}$

$$\textsc{Eval-Val}\ \frac{k > 0}{s \vdash v \Downarrow^k (\emptyset)\, v} \qquad \textsc{Eval-New}\ \frac{n \notin s \qquad k > 0}{s \vdash \texttt{new} \Downarrow^k (\{n\})\, n}$$

$$\textsc{Eval-Cond}\ \frac{s \vdash e_0 \Downarrow^{k_0} (t_0)\, b \qquad s \oplus t_0 \vdash e_i \Downarrow^k (t)\, w \qquad (i, b) \in \{(1, \texttt{true}), (2, \texttt{false})\}}{s \vdash \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \Downarrow^{1+k_0+k} (t_0 \oplus t)\, w}$$

$$\textsc{Eval-Eq1}\ \frac{s \vdash e_1 \Downarrow^{k_1} (t_1)\, n \qquad s \oplus t_1 \vdash e_2 \Downarrow^{k_2} (t_2)\, n \qquad n \in s}{s \vdash (e_1 = e_2) \Downarrow^{1+k_1+k_2} (t_1 \oplus t_2)\, \texttt{true}}$$

$$\textsc{Eval-Eq2}\ \frac{s \vdash e_1 \Downarrow^{k_1} (t_1)\, n_1 \qquad s \oplus t_1 \vdash e_2 \Downarrow^{k_2} (t_2)\, n_2 \qquad n_1, n_2 \text{ distinct}}{s \vdash (e_1 = e_2) \Downarrow^{1+k_1+k_2} (t_1 \oplus t_2)\, \texttt{false}}$$

$$\textsc{Eval-App}\ \frac{\begin{array}{c} s \vdash e_0 \Downarrow^{k_0} (t_0)\, \lambda x{:}T_1.\, e_2 \qquad s \oplus t_0 \vdash e_1 \Downarrow^{k_1} (t_1)\, w_1 \\ s \oplus t_0 \oplus t_1 \vdash e_2[w_1/x] \Downarrow^{k_2} (t_2)\, w \end{array}}{s \vdash e_0\ e_1 \Downarrow^{1+k_0+k_1+k_2} (t_0 \oplus t_1 \oplus t_2)\, w}$$

Figure 4.3: Operational semantics for the ν-calculus

Lemma 4.1.4 (Garbage Collection)**.** *If* $s \oplus s_0 \vdash e \Downarrow^k (t)\, w$ *and* $s_0 \cap names(e) = \emptyset$ *then:*

$$s \vdash e \Downarrow^k (t)\, w.$$

4.2 Contextual Equivalence

The ν-calculus is strongly normalizing; thus, contextual equivalence (as defined in [53]) observes the resulting values of evaluation at boolean type.

Definition 4.2.1 (Contextual Equivalence $(\equiv)$)**.** $s; \Gamma \vdash e \equiv e' : T$ *if and only*

if for all contexts $s; \cdot \vdash C[\cdot]^T_\Gamma : o$*, and boolean values* b*:*

$$(\exists t.\ s \vdash C[e] \Downarrow (t)\, b) \iff (\exists t.\ s \vdash C[e'] \Downarrow (t)\, b)$$

To reason about contextual equivalence we develop our theory on closed values and extend it to open expressions via the following theorem.

Theorem 4.2.2 (Value Restriction)**.** *For any two expressions* $s; \overline{x : T_x} \vdash e, e' : T$

$$s; \overline{x : T_x} \vdash e \equiv e' : T \quad \iff \quad s; \cdot \vdash \lambda \overline{x{:}T_x}.\, e \equiv \lambda \overline{x{:}T_x}.\, e' : \overline{T_x} \to T$$

Proof. As the proof of Theorem 3.2.2. □

4.3 Derivation of Adequacy

4.3.1 Pre-Adequacy

Our development of the theory for pre-adequacy and adequacy is an adaptation from Chapter 3. The differences here are that we have namesets in the place of stores, relations are typed, and booleans are the observables, since the language is strongly normalizing. Here we write down the definitions and lemmas needed to prove soundness and completeness of adequate sets of annotated relations but omit a lengthy exposition of the proofs. The interested reader can find a Coq script of all omitted proofs accompanying [10].

We consider typed annotated value relations, which we write as:

$$(s, s', R) \in \textsc{Nameset} \times \textsc{Nameset} \times \mathscr{P}(\textsc{Value} \times \textsc{Value} \times \textsc{Type})$$

An annotated relation (s, s', R) is well-formed when:

$$\forall v, v', T.\ (v, v', T) \in R \implies (s; \cdot \vdash v : T) \wedge (s'; \cdot \vdash v' : T)$$

For the rest of this chapter we will consider only well-formed annotated relations.

We reason about sets of such relations:

$$\mathcal{X} \subseteq \textsc{Nameset} \times \textsc{Nameset} \times \mathscr{P}(\textsc{Value} \times \textsc{Value} \times \textsc{Type})$$

The inverse of a set of annotated relations is defined as follows.

Definition 4.3.1. *If $\mathcal{X}$ is a set of annotated relations, the inverse of $\mathcal{X}$, written $\mathcal{X}^{\mathsf{T}}$, is defined as:*

$$(s', s, R^{\mathsf{T}}) \in \mathcal{X}^{\mathsf{T}} \quad \textit{iff} \quad (s, s', R) \in \mathcal{X}$$

We also define the open extension of sets of annotated relations.

Definition 4.3.2 (Open Extension of Sets of Annotated Relations)**.** *If $\mathcal{X}$ is a set of annotated relations, then $\mathcal{X}^\circ$ is the relation such that $s; \overline{x : T_x} \vdash e\ \mathcal{X}^\circ\ e' : T$ if and only if there exists R such that:*

$$(s, s, R) \in \mathcal{X} \qquad s; \cdot \vdash \lambda\overline{x}.e\ R\ \lambda\overline{x}.e' : T \qquad \forall n \in dom(s).\ s; \cdot \vdash n\ R\ n : \nu$$

We close annotated relations under name-free, identical contexts. All names that these contexts can refer to are *public names* related by the core relation.

Definition 4.3.3 (Context Closure)**.** *If R is a binary relation on closed values, then R^{cxt} is the relation defined by*

$$\frac{\emptyset; \overline{x : T_x} \vdash D : T \qquad s, s'; \cdot \vdash \overline{u}\ R\ \overline{u'} : \overline{T_x}}{s, s'; \cdot \vdash D[\overline{u/x}]\ R^{\mathsf{cxt}}\ D[\overline{u'/x}] : T}$$

Using the context closure of annotated relations we give our definition of pre-adequacy for the ν-calculus, which closely resembles the standard definition of contextual equivalence. In fact, we show that the open extension of pre-adequacy coincides with contextual equivalence.

Definition 4.3.4 (Pre-Adequate Annotated Relations)**.** *An annotated relation, (s, s', R), is* pre-adequate *if and only if for all expressions e and e', such that $s, s'; \cdot \vdash e\ R^{\mathsf{cxt}}\ e' : o$, we have:*

$$(\exists t.\ s \vdash e \Downarrow (t)\, b) \iff (\exists t'.\ s' \vdash e' \Downarrow (t')\, b)$$

Definition 4.3.5 (Pre-Adequacy $(\cong)$)**.** $(\cong)$ *is the set of all pre-adequate annotated relations.*

Theorem 4.3.6 (Soundness and Completeness of $(\cong)$)**.** $(\cong)^\circ = (\equiv)$

Proof. As the proof of Theorem 3.3.7. □

4.3.2 Adequacy

Our main technical tool for reasoning about equivalence in the ν-calculus is the definition of adequate sets of annotated relations. This definition permits the use of an induction in the proofs of equivalence.

Definition 4.3.7 (Adequate Sets of Annotated Relations)**.** *A set of annotated relations $\mathcal{X}$ is* adequate *if an only if for all $(s, s', R) \in \mathcal{X}$ we have:*

$$
\begin{aligned}
\forall e, e', t, w.\ \ & s, s'; \cdot \vdash e\ R^{\mathsf{cxt}}\ e' : T \\
\wedge\ & s \vdash e \Downarrow (t)\, w \\
\Longrightarrow\ & \exists t', w', Q.\ \ s' \vdash e' \Downarrow (t')\, w' \\
& \wedge\ (T = o) \implies (w = w') \\
& \wedge\ s \oplus t, s' \oplus t'; \cdot \vdash w\ Q^{\mathsf{cxt}}\ w' : T \\
& \wedge\ (s \oplus t, s' \oplus t', Q) \in \mathcal{X} \\
& \wedge\ R \subseteq Q
\end{aligned}
$$

and similarly for all $(s, s', R) \in \mathcal{X}^{\mathsf{T}}$.

It is easy to see that the union of adequate sets is an adequate set. Thus, the union of all adequate sets is the largest adequate set.

Definition 4.3.8 (Adequacy $(\approx)$)**.** $(\approx)$ *is the largest adequate set of annotated relations.*

We show that adequacy is sound and complete with respect to contextual equivalence by showing that it coincides with pre-adequacy.

Theorem 4.3.9 (Soundness of Adequate Sets)**.** *If $\mathcal{X}$ is adequate then it is included in pre-adequacy.*

Proof. Trivial by the definitions of pre-adequate annotated relations and adequate sets of annotated relations. □

Theorem 4.3.10 (Completeness of Adequate Sets)**.** $(\cong)$ *is adequate.*

Proof. Let $(s, s', R) \in (\cong)$ and $s, s'; \cdot \vdash e\ R^{\mathsf{cxt}}\ e' : T$. We will show that

$$\begin{aligned}
&\forall t, w.\ s \vdash e \Downarrow (t)\, w \\
&\quad \Longrightarrow\ \exists t', w', Q.\ s' \vdash e' \Downarrow (t')\, w' \\
&\qquad\qquad \wedge\ (T = o)\ \Longrightarrow\ (w = w') \\
&\qquad\qquad \wedge\ s{\oplus}t, s'{\oplus}t'; \cdot \vdash w\ Q^{\mathsf{cxt}}\ w' : T \\
&\qquad\qquad \wedge\ (s{\oplus}t, s'{\oplus}t', Q) \in \mathcal{X} \\
&\qquad\qquad \wedge\ R \subseteq Q
\end{aligned}$$

By the definition of pre-adequate annotated relations (Definition 4.3.4) and the determinacy of the semantics, it suffices to show that

$$\begin{aligned}
&\forall t, t', w, w'.\ \langle s,\, e\rangle \Downarrow \langle t,\, w\rangle \\
&\qquad \wedge\ \langle s',\, e'\rangle \Downarrow \langle t',\, w'\rangle \\
&\qquad\quad \Longrightarrow\ \exists Q.\ s{\oplus}t, s'{\oplus}t'; \cdot \vdash w\ Q^{\mathsf{cxt}}\ w' : T \\
&\qquad\qquad \wedge\ (s{\oplus}t, s'{\oplus}t', Q) \in \mathcal{X} \\
&\qquad\qquad \wedge\ R \subseteq Q
\end{aligned}$$

Let $s \vdash e \Downarrow (t)\, w$ and $s' \vdash e' \Downarrow (t')\, w'$; we will show that

$$(s{\oplus}t, s'{\oplus}t', R \cup \{(w, w')\}) \in (\cong)$$

For any $\overline{x, T_x, v, v'}$, y, T_y, b, and d such that $\emptyset; \overline{x : T_x}, y : T_y \vdash d : o$, $s{\oplus}t, s'{\oplus}t'; \cdot \vdash \overline{v}\ R\ \overline{v'} : \overline{T_x}$, we have:

$$\begin{aligned}
&\exists t_1.\ s{\oplus}t \vdash d[\overline{v/x}, w/y] \Downarrow (t_1)\, b \\
\Longleftrightarrow\ &\exists t_1.\ s \vdash \lambda y{:}T.\, d[\overline{v/x}]\ e \Downarrow (t{\oplus}t_1)\, b && \text{(by the properties of evaluation)} \\
\Longleftrightarrow\ &\exists t'_1.\ s' \vdash \lambda y{:}T.\, d[\overline{v'/x}]\ e' \Downarrow (t'{\oplus}t'_1)\, b && ((s, s', R) \in (\cong)) \\
\Longleftrightarrow\ &\exists t'_1.\ s'{\oplus}t' \vdash d[\overline{v'/x}, w'/y] \Downarrow (t'_1)\, b && \text{(by the properties of evaluation)}
\end{aligned}$$

Therefore, by Definitions 2.3.3 and 2.3.5:

$$(t, t', R \cup \{(w, w')\}) \in (\cong)$$

□

Theorem 4.3.11. $(\cong) = (\approx)$.

Proof. By Theorem 4.3.9 we have $(\approx) \subseteq (\cong)$ and by Theorem 4.3.10 we have $(\cong) \subseteq (\approx)$. Thus $(\cong) = (\approx)$. □

From the above we conclude that the open extension of adequacy coincides with the standard definition of contextual equivalence.

Theorem 4.3.12. $(\approx)^\circ = (\equiv)$.

Proof. By Theorems 4.3.6 and 4.3.11. □

4.4 Inductive Proofs of Equivalence

Similarly to the development of our technique in previous chapters, we can show that $s; \overline{x : T_x} \vdash e \equiv e' : T$ by:

1. Finding a set $\mathcal{X}$ such that there exists R with:

$$(s, s, R) \in \mathcal{X}$$
$$s; \cdot \vdash \lambda\overline{x{:}T_x}.\, e\ R\ \lambda\overline{x{:}T_x}.\, e' : \overline{T_x} \to T$$
$$\forall n \in s.\ s; \cdot \vdash n\ R\ n : \nu$$

2. Showing that $\mathcal{X}$ is adequate, and

3. invoking Theorem 4.3.12 to show $s; \overline{x : T_x} \vdash e \equiv e' : T$.

We can prove that a set $\mathcal{X}$ is adequate by induction. The induction hypothesis we use is the following.

Definition 4.4.1.

$$
\begin{aligned}
IH_{\mathcal{X}}(k) \stackrel{\text{def}}{=} \forall (s, s', R) \in \mathcal{X}.\ & \\
\forall e, e', t, w.\ & \ s, s'; \cdot \vdash e\ R^{\mathsf{cxt}}\ e' : T \\
& \wedge\ s \vdash e \Downarrow^{k} (t)\, w \\
& \Longrightarrow\ \exists t', w', Q.\ s' \vdash e' \Downarrow (t')\, w' \\
& \qquad \wedge\ (T = o) \implies (w = w') \\
& \qquad \wedge\ s \oplus t, s' \oplus t'; \cdot \vdash w\ Q^{\mathsf{cxt}}\ w' : T \\
& \qquad \wedge\ (s \oplus t, s' \oplus t', Q) \in \mathcal{X} \\
& \qquad \wedge\ R \subseteq Q
\end{aligned}
$$

The measure of the induction is the size k of the evaluation $s \vdash e \Downarrow^{k} (t)\, w$. Hence, proving a set of annotated relations, $\mathcal{X}$, adequate amounts to proving that for all k, $IH_{\mathcal{X}}(k)$ and $IH_{\mathcal{X}^{\mathsf{T}}}(k)$ hold. For $k = 0$ it is trivial; for $k > 0$ it can be shown by induction:

$$
\boxed{
\begin{array}{l}
\forall k.\ IH_{\mathcal{X}}(k-1) \implies IH_{\mathcal{X}}(k) \\
\forall k.\ IH_{\mathcal{X}^{\mathsf{T}}}(k-1) \implies IH_{\mathcal{X}^{\mathsf{T}}}(k)
\end{array}
}
$$

4.5 Deriving Smaller Proof Obligations for Adequate Sets

By using a *proof construction scheme*, as in Section 3.5, we factor out the common parts of the two inductions at the end of the previous section, and discover necessary and sufficient proof obligations for adequacy. Thus, we arrive at the following adequacy theorem.

Theorem 4.5.1. *A set of annotated relations $\mathcal{X}$ is adequate if and only if for all k and all $(s, s', R) \in \mathcal{X}$, assuming that $IH_{\mathcal{X}}(k-1)$ holds, the following conditions hold:*

1. *For all $s, s'; \cdot \vdash b\ R\ b' : o$ it must be that $b = b'$.*

2. *For all* $s, s'; \cdot \vdash \lambda x{:}T_x.e\ R\ \lambda x{:}T_x.e' : T_x \to T$*, and all* $s, s'; \cdot \vdash v\ R^{\mathsf{cxt}}\ v' : T_x$*,* t*, and* w*, such that* $s \vdash \lambda x{:}T_x.e\ v \Downarrow^k (t)\, w$*, there exist* t'*,* w'*, and* $Q \supseteq R$ *such that:*

$$s' \vdash \lambda x{:}T_x.e'\ v' \Downarrow (t')\, w' \qquad s \oplus t, s' \oplus t'; \cdot \vdash w\ Q^{\mathsf{cxt}}\ w' : T$$
$$(s \oplus t, s' \oplus t', Q) \in \mathcal{X}$$

3. *For all* $n \notin s$ *there exist* $n' \notin s'$ *and* $Q \supseteq R$ *such that:*

$$s \oplus \{n\}, s' \oplus \{n'\}; \cdot \vdash n\ Q\ n' : \nu \qquad (s \oplus \{n\}, s' \oplus \{n'\}, Q) \in \mathcal{X}$$

4. *For all* $s, s'; \cdot \vdash n_1\ R\ n_1' : o$ *and* $s, s'; \cdot \vdash n_2\ R\ n_2' : o$*:*

$$n_1 = n_2 \iff n_1' = n_2'$$

Moreover, the same conditions hold for $\mathcal{X}^{\mathsf{T}}$*.*

Proof. Similar to the proof of Theorem 3.5.1. □

4.6 Examples

Using the preceding theorem, we are able to prove all equivalences in the ν-calculus from [53]. In this section we start with the proof of a straightforward equivalence and then show the proof of the most interesting of these equivalences. The only other method for proving the latter equivalence is by using game semantics [3].

4.6.1 A Simple Example: Local Names

This equivalence demonstrates that the context can not provide names that are local to the terms.

$$M \stackrel{\text{def}}{=} \nu n.\, \lambda x{:}\nu.\, (x = n)$$
$$M' \stackrel{\text{def}}{=} \lambda x{:}\nu.\, \mathtt{false}$$

$$\frac{}{([],[],\{(N,N',o\to\nu\to o)\})\in\mathcal{X}}\ \mathcal{X}\text{-1}$$

$$\frac{(s,s',R)\in\mathcal{X}\qquad n\notin s\qquad n'\notin s'}{(s\oplus\{n\},s'\oplus\{n'\},R\cup\{(n,n',\nu)\})\in\mathcal{X}}\ \mathcal{X}\text{-2}$$

$$\frac{(s,s',R)\in\mathcal{X}\qquad n\notin s}{(s\oplus\{n\},s',R\cup\{(\lambda x{:}\nu.\,(x{=}n),\lambda x{:}\nu.\,\mathtt{false},\nu\to o)\})\in\mathcal{X}}\ \mathcal{X}\text{-3}$$

Figure 4.4: Construction of adequate set of annotated relations for proving the equivalence of a simple equivalence in the ν-calculus.

Proof. From Theorem 4.2.2, it suffices to show that the following two values are equivalent:

$$N \stackrel{\text{def}}{=} \lambda y{:}o.\,\nu n.\,\lambda x{:}\nu.\,(x{=}n)$$
$$N' \stackrel{\text{def}}{=} \lambda y{:}o.\,\lambda x{:}\nu.\,\mathtt{false}$$

To prove $\emptyset;\cdot\vdash N\equiv N' : o\to\nu\to o$ we need to construct an adequate set of annotated relations, $\mathcal{X}$, such that there exists R with:

$$(\emptyset,\emptyset,R)\in\mathcal{X}\qquad \emptyset;\cdot\vdash N\ R\ N' : o\to\nu\to o$$

We start the construction of an adequate $\mathcal{X}$ by the first two rules of Figure 4.4. Rule $\mathcal{X}$-2 fulfills Condition 3 of Theorem 4.5.1. Conditions 1 and 4 are trivially satisfied.

We need to prove Condition 2 of Theorem 4.5.1 for any $(s,s',R)\in\mathcal{X}$ with $s,s';\cdot\vdash N\ R\ N' : o\to\nu\to o$. Let $s,s';\cdot\vdash b\ R^{\mathsf{cxt}}\ b : o$. We have:

$$s\vdash N\ b\Downarrow(\{n\})\,\lambda x{:}\nu.\,(x{=}n)\qquad\qquad n\notin s$$
$$s'\vdash N'\ b\Downarrow(\emptyset)\,\lambda x{:}\nu.\,\mathtt{false}$$

To prove this case we add Rule $\mathcal{X}$-3 of Figure 4.4 in the construction of $\mathcal{X}$. Hence we have:

$$(s\oplus\{n\},s',R\cup\{(\lambda x{:}\nu.\,(x{=}n),\lambda x{:}\nu.\,\mathtt{false},\nu\to o)\})\in\mathcal{X}$$

It remains to prove Condition 2 for any $(s{\oplus}\{n\}, s', R) \in \mathcal{X}$ with $s{\oplus}\{n\}, s'; \cdot \vdash \lambda x{:}\nu.\,(x{=}n)\ R\ \lambda x{:}\nu.\,\texttt{false} : \nu \to o$. Let:

$$s{\oplus}\{n\}, s'; \cdot \vdash n_0\ R^{\mathsf{cxt}}\ n_0' : \nu$$

By the definition of $(\ )^{\mathsf{cxt}}$ we have:

$$s{\oplus}\{n\}, s'; \cdot \vdash n_0\ R\ n_0' : \nu$$

By construction of $\mathcal{X}$ we have that $n \neq n_0$. Hence:

$$\begin{gathered} s{\oplus}\{n\} \vdash (\lambda x{:}\nu.\,(x{=}n))\ n_0 \Downarrow(\emptyset)\,\texttt{false} \\ s' \vdash (\lambda x{:}\nu.\,\texttt{false})\ n_0' \Downarrow(\emptyset)\,\texttt{false} \\ s{\oplus}\{n\}, s'; \cdot \vdash \texttt{false}\ R^{\mathsf{cxt}}\ \texttt{false} : o \\ (s{\oplus}\{n\}, s', R) \in \mathcal{X} \end{gathered}$$

This concludes the proof of adequacy of $\mathcal{X}$, and by Theorem 4.3.12:

$$\emptyset; \cdot \vdash N \equiv N' : \nu \to o$$

□

4.6.2 The 'Hard' Equivalence

Here we show the proof of what is considered the canonical hard-to-prove equivalence of the ν-calculus. This has only been validated with the use of game semantics [3]. Our proof here uses operational semantics and two adequate sets of annotated relations.

$$M \stackrel{\text{def}}{=} \nu n_1.\,\nu n_2.\,U(n_1, n_2) \qquad U(n_1, n_2) \stackrel{\text{def}}{=} \lambda f{:}\nu \to o.\,((f\ n_1){=}(f\ n_2))$$
$$M' \stackrel{\text{def}}{=} \lambda f{:}\nu \to o.\,\texttt{true}$$

The equivalence here means that although the names n_1 and n_2 are revealed to the context (by passing them as arguments to f), they cannot

$$\frac{}{(\emptyset, \emptyset, \{(N, N', o \to (\nu \to o) \to o)\}) \in \mathcal{X}}\ \mathcal{X}\text{-1}$$

$$\frac{(s, s', R) \in \mathcal{X} \qquad n \notin s \qquad n' \notin s'}{(s \oplus \{n\}, s' \oplus \{n'\}, R \cup \{(n, n', \nu)\}) \in \mathcal{X}}\ \mathcal{X}\text{-2}$$

$$\frac{(s, s', R) \in \mathcal{X} \qquad \{n_1, n_2\} \cap s = \emptyset}{(s \oplus \{n_1, n_2\}, s', R \cup \{(U(n_1, n_2), M', (\nu \to o) \to o)\}) \in \mathcal{X}}\ \mathcal{X}\text{-3}$$

$$\frac{(s, s', R) \in \mathcal{X} \qquad s_0 \cap s = 0}{(s \oplus s_0, s', R) \in \mathcal{X}}\ \mathcal{X}\text{-4}$$

Figure 4.5: Construction of the primary adequate set of annotated relations for proving the canonical 'hard' equivalence in the ν-calculus.

be stored between applications of f. Thus the outermost application of $U(n_1, n_2)$ will return `true`. While the body of f is evaluated, though, internal applications of $U(n_1, n_2)$ may return `false`.

Proof. From Theorem 4.2.2, it suffices to show that the following two values are equivalent:

$$N \stackrel{\text{def}}{=} \lambda y{:}o.\, \nu n_1.\, \nu n_2.\, U(n_1, n_2)$$
$$N' \stackrel{\text{def}}{=} \lambda y{:}o.\, \lambda f{:}\nu \to o.\, \texttt{true}$$

To prove $\emptyset; \cdot \vdash N \equiv N' : o \to \nu \to o$ we need to construct an adequate set of annotated relations, $\mathcal{X}$, such that there exists R with:

$$(\emptyset, \emptyset, R) \in \mathcal{X} \qquad \emptyset; \cdot \vdash N\ R\ N' : o \to \nu \to o$$

We start the construction of an adequate $\mathcal{X}$ by the first two rules of Figure 4.5. Rule $\mathcal{X}$-2 fulfills Condition 3 of Theorem 4.5.1. Conditions 1 and 4 are trivially satisfied.

We need to prove Condition 2 of Theorem 4.5.1 for any $(s, s', R) \in \mathcal{X}$ with $s, s'; \cdot \vdash N\ R\ N' : o \to \nu \to o$. Let $s, s'; \cdot \vdash b\ R^{\mathsf{cxt}}\ b : o$. We have:

$$s \vdash N\ b \Downarrow (\{n_1, n_2\})\, U(n_1, n_2) \qquad \{n_1, n_2\} \cap s = \emptyset$$
$$s' \vdash N'\ b \Downarrow (\emptyset)\, M'$$

To prove this case we add Rule $\mathcal{X}$-2 in the construction of $\mathcal{X}$. Hence we have:

$$(s \oplus \{n\}, s', R \cup \{(U(n_1, n_2), M', (\nu \to o) \to o)\}) \in \mathcal{X}$$

It remains to prove Condition 2 for any $(s \oplus \{n_1, n_2\}, s', R) \in \mathcal{X}$ with $s \oplus \{n_1, n_2\}, s'; \cdot \vdash U(n_1, n_2)\ R\ M' : (\nu \to o) \to o$. Let:

$$s \oplus \{n_1, n_2\}, s'; \cdot \vdash u\ R^{\mathsf{cxt}}\ u' : \nu \to o \qquad s \oplus \{n_1, n_2\} \vdash U(n_1, n_2)\ u \Downarrow^k (t)\, w$$

We need to show that there exist Q, t', and w' such that:

$$s' \vdash M'\ u' \Downarrow (t')\, w' \qquad w = w' \qquad (s \oplus \{n_1, n_2\} \oplus t, s' \oplus t', Q) \in \mathcal{X} \qquad R \subseteq Q$$

But we have:

$$s' \vdash M'\ u' \Downarrow (\emptyset)\, \texttt{true}$$

Thus $t' = \emptyset$, $w' = \texttt{true}$, and $Q = R$. We add Rule $\mathcal{X}$-4 to the construction of $\mathcal{X}$ in Figure 4.5 and get:

$$(s \oplus \{n_1, n_2\} \oplus t, s', R) \in \mathcal{X}$$

It only remains to show that the first application evaluates to `true`; i.e.:

$$s \oplus \{n_1, n_2\} \vdash U(n_1, n_2)\ u \Downarrow (t)\, \texttt{true}$$

To prove this we need to reason about the operational behavior of the application. We do by re-using our method.

By the properties of evaluation, it suffices to show that there exist b, t_1, and t_2 such that:

$$s \oplus \{n_1, n_2\} \vdash u\ n_1 \Downarrow (t_1)\, b$$
$$s \oplus \{n_1, n_2\} \oplus t_1 \vdash u\ n_2 \Downarrow (t_2)\, b$$

$$\frac{}{(\emptyset, \emptyset, \{(N, N, o \to (\nu \to o) \to o)\}) \in \mathcal{Y}}\ \mathcal{Y}\text{-1}$$

$$\frac{(s, s, R) \in \mathcal{Y} \qquad n \notin s}{(s \oplus \{n\}, s \oplus \{n\}, R \cup \{(n, n, \nu)\}) \in \mathcal{Y}}\ \mathcal{Y}\text{-2}$$

$$\frac{\begin{array}{c}(s, s, R) \in \mathcal{Y} \qquad \{n_1, n_2\} \cap s = \emptyset \\ Q = R \cup \{(n_1, n_2, \nu), (n_2, n_1, \nu), (U(n_1, n_2), U(n_1, n_2), (\nu \to o) \to o)\}\end{array}}{(s \oplus \{n_1, n_2\}, s \oplus \{n_1, n_2\}, Q) \in \mathcal{Y}}\ \mathcal{Y}\text{-3}$$

$$\frac{(s, s, R) \in \mathcal{Y} \qquad s_0 \cap s = 0}{(s \oplus s_0, s \oplus s_0, R) \in \mathcal{X}}\ \mathcal{Y}\text{-4}$$

Figure 4.6: Construction of the auxiliary adequate set of annotated relations for proving the canonical 'hard' equivalence in the ν-calculus.

or, from Theorem 4.1.3, it suffices to show that there exist b, t_1, and t_2 such that:

$$s \oplus \{n_1, n_2\} \vdash u\ n_1 \Downarrow (t_1)\, b$$

$$s \oplus \{n_1, n_2\} \vdash u\ n_2 \Downarrow (t_2)\, b$$

We show this by constructing an auxiliary adequate set $\mathcal{Y}$ of annotated relations such that there exists a relation P with:

$$s \oplus \{n_1, n_2\}; \cdot \vdash (u\ n_1)\ P^{\mathsf{cxt}}\ (u\ n_2) : o$$

$$(s \oplus \{n_1, n_2\}, s \oplus \{n_1, n_2\}, P) \in \mathcal{Y}$$

The construction of $\mathcal{Y}$ is shown in Figure 4.6.

We establish a correlation between the sets of annotated relations $\mathcal{X}$ and $\mathcal{Y}$ by the following lemma and corollary.

Lemma 4.6.1. *For all $(s, s', R) \in \mathcal{X}$, there exists P such that:*

$$(s, s, P) \in \mathcal{Y}$$

$$\forall v, v'.\ s, s'; \cdot \vdash v\ R\ v' : T \implies s; \cdot \vdash v\ P\ v : T$$

Proof. We proceed by induction on the construction of $\mathcal{X}$.

Case $\mathcal{X}$-1

$$\frac{}{(\emptyset, \emptyset, \{(N, N', o \to (\nu \to o) \to o)\}) \in \mathcal{X}}\ \mathcal{X}\text{-1}$$

This case is trivial because by $\mathcal{Y}$-1

$$(\emptyset, \emptyset, \{(N, N, o \to (\nu \to o) \to o)\}) \in \mathcal{Y}$$

Case $\mathcal{X}$-2

$$\frac{(s, s', R) \in \mathcal{X} \qquad n \notin s \qquad n' \notin s'}{(s \oplus \{n\}, s' \oplus \{n'\}, R \cup \{(n, n', \nu)\}) \in \mathcal{X}}\ \mathcal{X}\text{-2}$$

By the induction hypothesis at $(s, s', R) \in \mathcal{X}$ we get that there exists P such that

$$(s, s, P) \in \mathcal{Y} \tag{4.1}$$

$$\forall v, v'.\ s, s'; \cdot \vdash v\ R\ v' : T \implies s; \cdot \vdash v\ P\ v : T \tag{4.2}$$

By $\mathcal{Y}$-2 and (4.1) we get that

$$(s \oplus \{n\}, s \oplus \{n\}, P \cup \{(n, n, \nu)\}) \in \mathcal{Y}$$

Let $s \oplus \{n\}, s' \oplus \{n'\}; \cdot \vdash v\ (R \cup \{(n, n', \nu)\})\ v' : T$. We have two cases:

1. $s, s'; \cdot \vdash v\ R\ v' : T$. By (4.2) we get $s; \cdot \vdash v\ P\ v : T$, and thus

$$s \oplus \{n\}; \cdot \vdash v\ (P \cup \{(n, n, \nu)\})\ v : T$$

2. $v = n$, $v = n'$, $T = \nu$. It is immediate that

$$s \oplus \{n\}; \cdot \vdash n\ (P \cup \{(n, n, \nu)\})\ n : \nu$$

Case $\mathcal{X}$-3

$$\frac{(s, s', R) \in \mathcal{X} \qquad \{n_1, n_2\} \cap s = \emptyset}{(s \oplus \{n_1, n_2\}, s', R \cup \{(U(n_1, n_2), M', (\nu \to o) \to o)\}) \in \mathcal{X}}\ \mathcal{X}\text{-3}$$

By the induction hypothesis at $(s, s', R) \in \mathcal{X}$ we get that there exists P such that

$$(s, s, P) \in \mathcal{Y} \tag{4.3}$$

$$\forall v, v'.\ s, s'; \cdot \vdash v\ R\ v' : T \implies s; \cdot \vdash v\ P\ v : T \tag{4.4}$$

Let $Q = P \cup \{(n_1, n_2, \nu), (n_2, n_1, \nu), (U(n_1, n_2), U(n_1, n_2), (\nu \to o) \to o)\}$. By $\mathcal{Y}$-3 and (4.3) we get that

$$(s \oplus \{n_1, n_2\}, s \oplus \{n_1, n_2\}, Q) \in \mathcal{Y}$$

Let $s \oplus \{n_1, n_2\}, s'; \cdot \vdash v\ (R \cup \{(U(n_1, n_2), M', (\nu \to o) \to o)\})\ v' : T$. We have two cases:

1. $s, s'; \cdot \vdash v\ R\ v' : T$. By (4.4) we get $s; \cdot \vdash v\ P\ v : T$, and thus

$$s \oplus \{n_1, n_2\}; \cdot \vdash v\ Q\ v : T$$

2. $v = U(n_1, n_2)$, $v' = M'$, $T = (\nu \to o) \to o$. It is immediate that

$$s \oplus \{n_1, n_2\}; \cdot \vdash U(n_1, n_2)\ Q\ U(n_1, n_2) : (\nu \to o) \to o$$

Case $\mathcal{X}$-4

$$\frac{(s, s', R) \in \mathcal{X} \qquad s_0 \cap s = 0}{(s \oplus s_0, s', R) \in \mathcal{X}}\ \mathcal{X}\text{-4}$$

Immediate by the induction hypothesis at $(s, s', R) \in \mathcal{X}$ and $\mathcal{Y}$-4. □

Corollary 4.6.2. *For all $(s, s', R) \in \mathcal{X}$, there exists P such that:*

$$(s, s, P) \in \mathcal{Y}$$

$$s, s'; \cdot \vdash v\ R^{\mathsf{cxt}}\ v' : T \implies s; \cdot \vdash v\ P^{\mathsf{cxt}}\ v : T$$

(Continuing proof from page 84.) From the above corollary and because

$$(s \oplus \{n_1, n_2\}, s', R) \in \mathcal{X} \qquad s \oplus \{n_1, n_2\}, s'; \cdot \vdash (U(n_1, n_2)\ u)\ R^{\mathsf{cxt}}\ (M'\ u') : o$$

we have that there exists P such that:

$$(s \oplus \{n_1, n_2\}, s \oplus \{n_1, n_2\}, P) \in \mathcal{X}$$

$$s \oplus \{n_1, n_2\}; \cdot \vdash (U(n_1, n_2)\ u)\ P^{\mathsf{cxt}}\ (U(n_1, n_2)\ u) : o$$

By the definition of $(\ )^{\mathsf{cxt}}$ we get:

$$s \oplus \{n_1, n_2\}; \cdot \vdash U(n_1, n_2)\ P^{\mathsf{cxt}}\ U(n_1, n_2) : (\nu \to \nu) \to o$$

$$s \oplus \{n_1, n_2\}; \cdot \vdash u\ P^{\mathsf{cxt}}\ u : \nu \to o$$

Because $U(n_1, n_2)$ is related to itself only in rule $\mathcal{X}$-3, we also have that:

$$s \oplus \{n_1, n_2\}; \cdot \vdash n_1\ P\ n_2 : o$$

Therefore:

$$s \oplus \{n_1, n_2\}; \cdot \vdash (u\ n_1)\ P^{\mathsf{cxt}}\ (u\ n_2) : o$$

It remains to show that $\mathcal{Y}$ is adequate by showing that it satisfies the conditions of Theorem 4.5.1.

$\mathcal{Y}$ trivially satisfies Conditions 1 and 4 of Theorem 4.5.1. Condition 3 of the theorem is fulfilled by Rule $\mathcal{Y}$-2. It remains to prove Condition 2 for all related abstractions.

Let $(s, s', R) \in \mathcal{Y}$. It is the case that $\mathcal{Y}$ is the identity modulo the crosswise renaming of some names. Thus $s = s'$ and for some names $\overline{n_1, n_2}$ and values $\overline{v}$ we have that $R = \{\overline{(v, v)}, \overline{(n_1, n_2)}, \overline{(n_2, n_1)}\}$.

Therefore, we consider $(s, s, R) \in \mathcal{Y}$, and prove Condition 2 for the following cases:

Case $s; \cdot \vdash N\ R\ N : o \to (\nu \to o) \to o$

Let $s; \cdot \vdash b\ R^{\mathsf{cxt}}\ b : o$ and $s \vdash N\ b \Downarrow^k (\{n_1, n_2\})\, U(n_1, n_2)$. By Rule $\mathcal{Y}$-3, there exists Q such that:

$$Q = R \cup \{(n_1, n_2, \nu), (n_2, n_1, \nu), (U(n_1, n_2), U(n_1, n_2), (\nu \to o) \to o)\}$$

$$s \vdash N\ b \Downarrow (\{n_1, n_2\})\, U(n_1, n_2)$$

$$s \oplus \{n_1, n_2\}; \cdot \vdash U(n_1, n_2)\ Q^{\mathsf{cxt}}\ U(n_1, n_2) : (\nu \to o) \to o$$

$$(s \oplus \{n_1, n_2\}, s \oplus \{n_1, n_2\}, Q) \in \mathcal{Y}$$

Case $s;\cdot \vdash U(n_1,n_2)\ R\ U(n_1,n_2) : (\nu \to o) \to o$

Let $s;\cdot \vdash v\ R^{\mathsf{cxt}}\ v' : \nu \to o$ and $s \vdash U(n_1,n_2)\ v \Downarrow^k (t)\, b$. By the properties of evaluation we have $t = s_1 \oplus s_2$ and:

$$s \vdash v\ n_1 \Downarrow^{k-1} (s_1)\, b_1 \tag{4.5}$$

$$s \oplus s_1 \vdash v\ n_2 \Downarrow^{k-1} (s_2)\, b_1 \tag{4.6}$$

By Lemma 4.1.4:

$$s \vdash v\ n_2 \Downarrow^{k-1} (s_2)\, b_1 \tag{4.7}$$

Because

$$s;\cdot \vdash (v\ n_1)\ R^{\mathsf{cxt}}\ (v'\ n_2) : o \qquad s;\cdot \vdash (v\ n_2)\ R^{\mathsf{cxt}}\ (v'\ n_1) : o$$

and by $IH_{\mathcal{Y}}(k-1)$, (4.5), and (4.7) we get that there exist Q_1 and Q_2 such that:

$$s \vdash v'\ n_2 \Downarrow (s_1')\, b_1' \qquad s \oplus s_1, s \oplus s_1';\cdot \vdash b_1\ {Q_1}^{\mathsf{cxt}}\ b_1' : o \qquad (s \oplus s_1, s \oplus s_1', Q_1) \in \mathcal{Y}$$

$$s \vdash v'\ n_1 \Downarrow (s_2')\, b_2' \qquad s \oplus s_2, s \oplus s_2';\cdot \vdash b_2\ {Q_2}^{\mathsf{cxt}}\ b_2' : o \qquad (s \oplus s_2, s \oplus s_2', Q_2) \in \mathcal{Y}$$

By the definition of $(\ )^{\mathsf{cxt}}$ we get that $b_1 = b_1'$ and $b_2 = b_2'$. Because each relation in $\mathcal{Y}$ is annotated with identical stores, $s_1 = s_1'$ and $s_2 = s_2'$. Therefore:

$$s \vdash v'\ n_2 \Downarrow (s_1)\, b_1$$

$$s \vdash v'\ n_1 \Downarrow (s_2)\, b_2$$

By (4.6) we get that $s_1 \cap s_2 = s \cap s_2 = \emptyset$. Thus, by Theorem 4.1.3 we get:

$$s \oplus s_2 \vdash v'\ n_2 \Downarrow (s_1)\, b_1$$

and by the properties of evaluation:

$$s \vdash ((v'\ n_1) = (v'\ n_2)) \Downarrow (s_1 \oplus s_2)\, b$$

Furthermore:

$$s \oplus s_1, s \oplus s_1;\cdot \vdash b\ R^{\mathsf{cxt}}\ b : o$$

and by Rule $\mathcal{X}$-4 we get:

$$(s \oplus s_1, s \oplus s_1, R) \in \mathcal{Y}$$

Therefore, $\mathcal{Y}$ is adequate. □

CHAPTER 5

An Imperative Object Calculus

In this chapter we develop our theory of equivalence for Abadi and Cardelli's impς [1], an untyped, imperative object calculus. In contrast to the languages of previous chapters, the values of this calculus do not have significant structure; most of the context is coded as objects in the store. Thus an "up-to context" technique is not useful here. Instead we use an "up-to store" technique to deal with the complex structure of the store.

5.1 Language and Semantics

The syntactic domains and the big-step environment semantics of the language are shown in Figures 5.1 and 5.2, respectively.

In impς, objects are defined by the syntactic construct $\{\!|\overline{l = \varsigma(x)\, b}|\!\}$, where l_i and b_i are the label and the body of the i-th method, respectively. Identifier x is bound to the entire object when b_i is evaluated. During evaluation bindings are kept in environments (called "stacks" in [1]).

Values in impς are objects that map method names to locations in the store; they are denoted by $\{\!|\overline{l = \iota}|\!\}$. Locations range over an infinite, countable set. The store maps locations to method closures, which consist of a method and the appropriate environment, e.g., $\langle \varsigma(x)\, b_i, \rho_i \rangle$.

We write $(a;\, b)$ to mean $\texttt{let}\, x = a\, \texttt{in}\, b$ when x does not appear free in b. We also write $\varsigma()\, b$ when b does not invoke methods on the current object,

EXPRESSION:	$a, b ::= x$	Identifier
	$\mid \{\!\!\{\overline{l = \varsigma(x)\, b}\}\!\!\}$	Object
	$\mid a.l$	Method Invocation
	$\mid a.l \Leftarrow \varsigma(x)\, b$	Method update
	$\mid \texttt{clone}(a)$	Cloning
	$\mid \texttt{let}\, x = a\, \texttt{in}\, b$	Let Expression
LOCATION:	ι	
VALUE:	$v, u ::= \{\!\!\{\overline{l = \iota}\}\!\!\}$	
ENVIRONMENT:	$\rho ::= (\overline{x \mapsto v})$	
STORE:	$s, t ::= [\overline{\iota \mapsto \langle \varsigma(x)\, b, \rho \rangle}]$	

Figure 5.1: Syntactic domains of impς.

and $a[v/x]$ for the capture-avoiding substitution of v for x in a.

We use the operational semantics of impς given in [1], with the addition of an extra condition in the ENV x rule which guarantees that there are no dangling pointers in an environment. Furthermore, there is an implicit α-conversion in the RED SELECT and RED LET rule, so that the identifiers added to the environments are unique.

Judgments of the form $\sigma; \rho \vdash a \Downarrow^k v; \sigma_1$ represent a big-step evaluation, where expression a, under store σ and environment ρ, evaluates to value v and a new store σ_1. The natural number k is an upper bound on the number of derivations contained in the evaluation tree. We use the form $\sigma; \rho \vdash a \Downarrow v; \sigma_1$ to mean that there is a k such that $\sigma; \rho \vdash a \Downarrow^k v; \sigma_1$. This measure corresponds to the length of reductions in a small-step semantics, and serves as a stronger measure for the inductive principle we will define in our theory than the height of the evaluation trees. Moreover, we write $\sigma; \rho \vdash a \Downarrow$ to mean that there exist v, σ_1, such that $\sigma; \rho \vdash a \Downarrow v; \sigma_1$.

Store update and extension are both written as $\sigma[\overline{\iota \mapsto \langle \varsigma(x)\, b, \rho \rangle}]$ and can be distinguished by whether ι is in $dom(\sigma)$ or not. Environment extension is written as $(\rho, \overline{x \mapsto \iota})$. The contents of the location ι of store σ is given by $\sigma(\iota)$. The domain of a store or environment is given by $dom(\sigma)$ or $dom(\rho)$, and the

$\boxed{\sigma \vdash_{\mathsf{wf}} \diamond}$

$$\textsc{Store}\ [] \quad \frac{}{[] \vdash_{\mathsf{wf}} \diamond} \qquad \textsc{Store}\ \iota \quad \frac{\sigma; \rho \vdash_{\mathsf{wf}} \diamond \qquad \iota \notin dom(\sigma) \qquad dom(\rho) \cup \{x\} \vdash b}{\sigma[\iota \mapsto \langle \varsigma(x)\, b, \rho \rangle] \vdash_{\mathsf{wf}} \diamond}$$

$\boxed{\sigma; \rho \vdash_{\mathsf{wf}} \diamond}$

$$\textsc{Env}\ \cdot \quad \frac{\sigma \vdash_{\mathsf{wf}} \diamond}{\sigma; \cdot \vdash_{\mathsf{wf}} \diamond} \qquad \textsc{Env}\ x \quad \frac{\sigma; \rho \vdash_{\mathsf{wf}} \diamond \qquad x \notin dom(\rho) \qquad \{\overline{\iota}\} \subseteq dom(\sigma)}{\sigma; (\rho, x \mapsto \{\!|\overline{l = \iota}|\!\}) \vdash_{\mathsf{wf}} \diamond}$$

$\boxed{\sigma; \rho \vdash a \Downarrow v; \sigma'}$

$$\textsc{Red}\ x \quad \frac{\sigma; (\rho_1, x \mapsto v, \rho_2) \vdash_{\mathsf{wf}} \diamond \qquad k > 0}{\sigma; (\rho_1, x \mapsto v, \rho_2) \vdash x \Downarrow^{k} v; \sigma}$$

$$\textsc{Red Object} \quad \frac{\sigma; \rho \vdash_{\mathsf{wf}} \diamond \qquad \{\overline{\iota}\} \cap dom(\sigma) = \emptyset \qquad k > 0}{\sigma; \rho \vdash \{\!|\overline{l = \varsigma(x)\, b}|\!\} \Downarrow^{k} \{\!|\overline{l = \iota}|\!\}; \sigma[\overline{\iota \mapsto \langle \varsigma(x)\, b, \rho \rangle}]}$$

$$\textsc{Red Select} \quad \frac{\sigma; \rho \vdash a \Downarrow^{k_1} \{\!|\overline{l = \iota}|\!\}; \sigma_1 \qquad l_j \in \{\overline{l}\} \qquad \sigma'(\iota_j) = \langle \varsigma(x)\, b_j, \rho_j \rangle \qquad \sigma_1; (\rho_j, x \mapsto \{\!|\overline{l = \iota}|\!\}) \vdash b_j \Downarrow^{k_2} v; \sigma_2}{\sigma; \rho \vdash a.l_j \Downarrow^{1+k_1+k_2} v; \sigma_2}$$

$$\textsc{Red Update} \quad \frac{\sigma; \rho \vdash a \Downarrow^{k} \{\!|\overline{l = \iota}|\!\}; \sigma_1 \qquad l_j \in \{\overline{l}\}}{\sigma; \rho \vdash a.l_j \Leftarrow \varsigma(x)\, b \Downarrow^{1+k} \{\!|\overline{l = \iota}|\!\}; \sigma_1[\iota_j \mapsto \langle \varsigma(x)\, b, \rho \rangle]}$$

$$\textsc{Red Clone} \quad \frac{\sigma; \rho \vdash a \Downarrow^{k} \{\!|\overline{l = \iota}|\!\}; \sigma_1 \qquad \{\overline{\iota_1}\} \notin dom(\sigma_1)}{\sigma; \rho \vdash \mathtt{clone}(a) \Downarrow^{1+k} \{\!|\overline{l = \iota_1}|\!\}; \sigma_1[\overline{\iota_1 \mapsto \sigma_1(\iota)}]}$$

$$\textsc{Red Let} \quad \frac{\sigma; \rho \vdash a \Downarrow^{k_1} v_1; \sigma_1 \qquad x \notin dom(\rho) \qquad \sigma'; (\rho, x \mapsto v_1) \vdash b \Downarrow^{k_2} v_2; \sigma_2}{\sigma; \rho \vdash \mathtt{let}\ x = a\ \mathtt{in}\ b \Downarrow^{1+k_1+k_2} v_2; \sigma_2}$$

Figure 5.2: Operational semantics of imp$_\varsigma$.

locations of a value v are given by $locs(v)$. We use juxtaposition of stores (or environments) $\sigma_0\sigma$ ($\rho_0\rho$), to denote a store (environment) with two disjoint parts.

Judgments of the form $\sigma \vdash_{\mathsf{wf}} \diamond$ define the *well-formed stores*, while judgments $\sigma; \rho \vdash_{\mathsf{wf}} \diamond$ define the *well-formed environments* under some store. We also call the form $\sigma; \rho \vdash a$ a *configuration*, and define the notion of a *well-formed configuration* as follows:

Definition 5.1.1. *We say that $\sigma; \rho \vdash a$ is a* well-formed configuration*, and we write $\sigma; \rho \vdash_{\mathsf{wf}} a$, if and only if:*

$$\sigma; \rho \vdash_{\mathsf{wf}} \diamond \qquad fv(a) \subseteq dom(\rho) \qquad locs(a) \subseteq dom(\sigma)$$

Evaluation only increases the domain of the store, and when it terminates, the resulting configuration is well-formed.

Lemma 5.1.2. *If $\sigma; \rho \vdash_{\mathsf{wf}} a$ and $\sigma; \rho \vdash a \Downarrow v; \sigma_1$, then $dom(\sigma) \supseteq dom(\sigma_1)$.*

Proof. By straightforward induction on the height of $\sigma; \rho \vdash a \Downarrow v; \sigma_1$. □

Lemma 5.1.3. *If $\sigma; \rho \vdash_{\mathsf{wf}} a$ and $\sigma; \rho \vdash a \Downarrow v; \sigma_1$, then $\sigma; \cdot \vdash_{\mathsf{wf}} v$.*

Proof. By straightforward induction on the height of $\sigma; \rho \vdash a \Downarrow v; \sigma_1$, using Lemma 5.1.2. □

We also prove that evaluation does not depend on extraneous bindings in the environments.

Lemma 5.1.4. *Let $\vdash \sigma < \sigma'$ when*

$$\forall \iota \in dom(\sigma).\ \exists x, b, \rho, \rho_0.\ (\sigma(\iota) = \langle \varsigma(x)\, b, \rho \rangle) \wedge (\sigma'(\iota) = \langle \varsigma(x)\, b, \rho_0\rho \rangle)$$

If $(\sigma; \rho \vdash_{\mathsf{wf}} a)$, $(\sigma'; \rho_0\rho \vdash_{\mathsf{wf}} a)$, and $\vdash \sigma < \sigma'$, then:

$$\begin{aligned} \sigma; \rho \vdash a \Downarrow v; \sigma_1 \implies \exists \sigma_1'.\ & \sigma'; \rho_0\rho \vdash a \Downarrow v; \sigma_1 \\ & \wedge\ \vdash \sigma_1 < \sigma_1' \end{aligned}$$

and vice-versa.

Proof. By induction on $\sigma; \rho \vdash a \Downarrow v; \sigma_1$. □

5.2 Contextual Equivalence

We use the following definition of contextual equivalence for imp_ς.

Definition 5.2.1 (Standard Contextual Equivalence ($\equiv$))**.** $\Phi \vdash a \equiv a'$ *if and only if for all contexts* $C[\cdot]_\Phi$*, stores* σ*, and environments* ρ*, such that* $\sigma; \rho \vdash_{\mathsf{wf}} C[a]$ *and* $\sigma; \rho \vdash_{\mathsf{wf}} C[a']$*, we have:*

$$\sigma; \rho \vdash C[a] \Downarrow \iff \sigma; \rho \vdash C[a'] \Downarrow$$

As for the languages of previous chapters, reasoning directly about contextual equivalence in imp_ς is hard. To address this issue we develop our theory based on equivalence of object values, which we then extend to arbitrary expressions.

For proving the soundness of our theory, we define *evaluation contexts* and a *CIU equivalence* for imp_ς, as in [18].[1]

Definition 5.2.2 (Evaluation Contexts)**.**

$$E \stackrel{\text{def}}{=} [\cdot] \mid E.l \mid E.l \Leftarrow \varsigma(x)\, b \mid \texttt{clone}(E) \mid \texttt{let}\, x = E\, \texttt{in}\, b$$

Definition 5.2.3 (CIU Equivalence ($\sim$))**.** *Two expressions* $\Phi \vdash a, a'$ *are CIU equivalent, and we write* $\Phi \vdash a \sim a'$*, if and only if for all stores* σ*, evaluation contexts* $\vdash E[\cdot]$*, and environments* ρ *with* $\sigma; \rho \vdash_{\mathsf{wf}} E[a]$ *and* $\sigma; \rho \vdash_{\mathsf{wf}} E[a']$*:*

$$\sigma; \rho \vdash E[a] \Downarrow \iff \sigma; \rho \vdash E[a] \Downarrow$$

CIU equivalence coincides with contextual equivalence.

Theorem 5.2.4. $(\sim) = (\equiv)$.

Proof. By showing that CIU equivalence is the largest equivalence relation that is a congruence and respects termination (see [18], Proposition 4.12). □

[1] In [18] evaluation contexts are called reduction contexts and the CIU equivalence is called operational equivalence.

5.3 Derivation of Adequacy

5.3.1 Pre-Adequacy

We consider binary value relations, annotated with a pair of stores:

$$(\sigma, \sigma', R) \in \textsc{Store} \times \textsc{Store} \times \mathscr{P}(\textsc{Value} \times \textsc{Value})$$

An annotated relation (σ, σ', R) is well-formed when for all $\vdash v\ R\ v'$:

$$\sigma; \cdot \vdash_{\mathsf{wf}} v \qquad \sigma'; \cdot \vdash_{\mathsf{wf}} v'$$

For the rest of this chapter we will consider only well-formed annotated relations.

We define identity annotated relations as follows.

Definition 5.3.1 (Identity Relations)**.** (σ, σ', I) *is an* identity annotated relation *if and only if:*

$$dom(\sigma) = dom(\sigma')$$
$$I \subseteq \{(\{\!|\overline{l = \iota}|\!\}, \{\!|\overline{l = \iota}|\!\}) \mid \bar{\iota} \in dom(\sigma)\}$$

Since evaluation of two equivalent expressions can lead to pairs of stores other than the initial one—under which more equivalences may hold—we need to reason about sets of such relations:

$$\mathcal{X} \subseteq \textsc{Store} \times \textsc{Store} \times \mathscr{P}(\textsc{Value} \times \textsc{Value})$$

The inverse of a set of annotated relations is defined as follows.

Definition 5.3.2. *If $\mathcal{X}$ is a set of annotated value relations, the inverse of $\mathcal{X}$, written $\mathcal{X}^{\mathsf{T}}$, is defined as:*

$$(\sigma', \sigma, R^{\mathsf{T}}) \in \mathcal{X}^{\mathsf{T}} \quad \textit{iff} \quad (\sigma, \sigma', R) \in \mathcal{X}$$

Moreover, since object values live in environments we define an *environment closure* (instead of a context closure) of annotated relations.

Definition 5.3.3 (Environment Closure)**.** *If R is a binary relation on closed values, then R^{env} is the relation defined by*

$$\frac{}{\vdash \cdot \; R^{\mathsf{env}} \; \cdot} \qquad \frac{\vdash \rho \; R^{\mathsf{env}} \; \rho' \qquad x \notin dom(\rho) \qquad \vdash v \; R \; v'}{\vdash (\rho, x{\mapsto}v) \; R^{\mathsf{env}} \; (\rho', x{\mapsto}v')}$$

We also define the extension of sets of annotated relations to arbitrary open expressions.

Definition 5.3.4 (Open Extension of Sets of Annotated Relations)**.** *If $\mathcal{X}$ is a set of annotated value relations, then $\mathcal{X}^\circ$ is the relation such that $\{\overline{x}\} \vdash a \; \mathcal{X}^\circ \; a'$ if and only if for all stores σ, locations ι, environments ρ, and relations I with:*

$$\iota \notin dom(\sigma) \qquad \vdash \rho \; I^{\mathsf{env}} \; \rho \qquad \{\overline{x}\} \subseteq dom(\rho)$$

$$(\sigma[\iota{\mapsto}\langle \varsigma() \; a, \rho\rangle], \sigma[\iota{\mapsto}\langle \varsigma() \; a', \rho\rangle], I) \text{ is an identity relation}$$

the following holds:

$$(\sigma[\iota{\mapsto}\langle \varsigma() \; a, \rho\rangle], \sigma[\iota{\mapsto}\langle \varsigma() \; a', \rho\rangle], I) \in \mathcal{X}$$

In imp$_\varsigma$ store locations are not values. Instead, objects values contain references to locations in the store, where their methods are stored. Thus, for this calculus, we define the notion of private and public object values (instead of private and public locations as in the calculi of previous chapters).[2] Public objects are the objects that appear in annotated relations—the ones that do not are private objects and can not be accessed by the context.

We now give the definition of pre-adequate annotated relations.

Definition 5.3.5 (Pre-Adequate Annotated Relations)**.** *An annotated relation, (σ, σ', R), is* pre-adequate *if and only if for all all environments $\vdash \rho \; R^{\mathsf{env}} \; \rho'$, and all expressions a, such that $dom(\rho) \vdash a$, we have:*

$$\sigma; \rho \vdash a \Downarrow \iff \sigma'; \rho' \vdash a \Downarrow$$

[2] We could use the notion of public and private locations by changing the stores to map locations to whole objects (instead of stored methods) and environments to map identifiers to locations (instead of objects). Here we chose to change our definitions to match the calculus, instead of the other way around.

Definition 5.3.6 (Pre-Adequacy $(\cong)$)**.** $(\cong)$ *is the set of all pre-adequate annotated relations.*

We next prove that the open extension of pre-adequacy is contextual equivalence.

Theorem 5.3.7 (Soundness of $(\cong)$)**.** $(\cong)^\circ \subseteq (\equiv)$.

Proof. Let $\overline{x}$ be a non-empty sequence of variables such that $\{\overline{x}\} \vdash a \cong^\circ a'$. We need to show that $\{\overline{x}\} \vdash a \ (\equiv)\ a'$.

From Theorem 5.2.4, it suffices to show that $\{\overline{x}\} \vdash a \ (\sim)\ a'$, which by the definition of $(\sim)$ means that we have to show:

$$\begin{aligned}\forall \sigma, \rho, E.\ \vdash E[\cdot]\\ \wedge\ \sigma; \rho \vdash_{\mathsf{wf}} E[a]\\ \wedge\ \sigma; \rho \vdash_{\mathsf{wf}} E[a']\\ \Longrightarrow\ (\sigma; \rho \vdash E[a]\Downarrow \iff \sigma; \rho \vdash E[a']\Downarrow)\end{aligned}$$

By the properties of evaluation we have:

$$\sigma; \rho \vdash E[a]\Downarrow \iff \sigma[\iota \mapsto \langle \varsigma()\, a, \rho\rangle]; (\rho, z \mapsto \{\!|l = \iota|\!\}) \vdash E[z.l]\Downarrow$$

where $\iota \notin dom(\sigma)$ and $z \notin dom(\rho)$.

Thus, we need to show:

$$\begin{aligned}\forall \sigma, \rho, E, l, \iota, z.\ \vdash E[\cdot]\\ \wedge\ \sigma; \rho \vdash_{\mathsf{wf}} E[a]\\ \wedge\ \sigma; \rho \vdash_{\mathsf{wf}} E[a']\\ \wedge\ \iota \notin dom(\sigma)\\ \wedge\ z \notin dom(\rho)\\ \Longrightarrow\ (\sigma[\iota \mapsto \langle \varsigma()\, a, \rho\rangle]; (\rho, z \mapsto \{\!|l = \iota|\!\}) \vdash E[z.l]\Downarrow\\ \iff\\ \sigma[\iota \mapsto \langle \varsigma()\, a', \rho\rangle]; (\rho, z \mapsto \{\!|l = \iota|\!\}) \vdash E[z.l]\Downarrow)\end{aligned}$$

By the definition of $(\ )^\circ$ and because $\{\overline{x}\} \vdash a\ (\cong)^\circ\ a'$ we have that for all σ, ρ, E, l, ι, z with

$$\vdash E[\cdot] \qquad \sigma;\rho \vdash_{\mathsf{wf}} E[a] \qquad \sigma;\rho \vdash_{\mathsf{wf}} E[a'] \qquad \iota \notin dom(\sigma) \qquad z \notin dom(\rho)$$

there exists I such that

$$\vdash \rho\ I^{\mathsf{env}}\ \rho$$
$$(\sigma[\iota \mapsto \langle \varsigma()\ a, \rho\rangle], \sigma[\iota \mapsto \langle \varsigma()\ a', \rho\rangle], I) \text{ is an identity relation}$$
$$(\sigma[\iota \mapsto \langle \varsigma()\ a, \rho\rangle], \sigma[\iota \mapsto \langle \varsigma()\ a', \rho\rangle], I) \in (\cong)$$

and by Definition 5.3.5 we get

$$\sigma[\iota \mapsto \langle \varsigma()\ a, \rho\rangle];\ (\rho, z \mapsto \{\!|l = \iota|\!\}) \vdash E[z.l] \Downarrow$$
$$\iff$$
$$\sigma[\iota \mapsto \langle \varsigma()\ a', \rho\rangle];\ (\rho, z \mapsto \{\!|l = \iota|\!\}) \vdash E[z.l] \Downarrow$$

□

Theorem 5.3.8 (Completeness of $(\cong)$)**.** $(\equiv) \subseteq (\cong)^\circ$.

Proof. Let $\overline{x}$ be a non-empty sequence of variables and $\{\overline{x}\} \vdash a\ (\equiv)\ a'$. We need to show that $\{\overline{x}\} \vdash a\ (\cong)^\circ\ a'$.

By the definition of $(\ )^\circ$, we need to show that for all σ, ι, ρ, I with:

$$\iota \notin dom(\sigma) \qquad \vdash \rho\ I^{\mathsf{env}}\ \rho \qquad \{\overline{x}\} \subseteq dom(\rho)$$
$$(\sigma[\iota \mapsto \langle \varsigma()\ a, \rho\rangle], \sigma[\iota \mapsto \langle \varsigma()\ a', \rho\rangle], I) \text{ is an identity relation}$$

the following must be true:

$$(\sigma[\iota \mapsto \langle \varsigma()\ a, \rho\rangle], \sigma[\iota \mapsto \langle \varsigma()\ a', \rho\rangle], I) \in (\cong)$$

By Definitions 5.3.5 and 5.3.6, we need to show that for all environments $\vdash \rho_1\ I^{\mathsf{env}}\ \rho_1'$ and expressions b, such that $dom(\rho_1) \vdash b$, we have:

$$\sigma[\iota \mapsto \langle \varsigma()\ a, \rho\rangle];\ \rho_1 \vdash b \Downarrow \iff \sigma[\iota \mapsto \langle \varsigma()\ a, \rho\rangle];\ \rho_1 \vdash b \Downarrow$$

W.l.o.g. we consider $dom(\rho) \cap dom(\rho_1) = \emptyset$. By Lemma 5.1.4, it suffices to show:

$$\sigma[\iota \mapsto \langle \varsigma()\, a, \rho\rho_1 \rangle]; \rho\rho_1 \vdash b \Downarrow \iff \sigma[\iota \mapsto \langle \varsigma()\, a, \rho\rho_1 \rangle]; \rho\rho_1 \vdash b \Downarrow$$

By the properties of evaluation, for some $x \in dom(\rho\rho_1)$, the above is equivalent to:

$$\sigma; \rho\rho_1 \vdash \mathtt{let}\, x = \{\!|l = \varsigma()\, a|\!\}\, \mathtt{in}\, b \Downarrow \iff \sigma; \rho\rho_1 \vdash \mathtt{let}\, x = \{\!|l = \varsigma()\, a'|\!\}\, \mathtt{in}\, b \Downarrow$$

But this is true by the definition of $(\equiv)$, taking $C[\cdot] = \mathtt{let}\, x = \{\!|l = \varsigma()\, [\cdot]|\!\}\, \mathtt{in}\, b$. □

From the above we get that the open extension of pre-adequacy coincides with contextual equivalence.

Theorem 5.3.9. $(\equiv) = (\cong)^\circ$.

Proof. Immediate from Theorems 5.3.7 and 5.3.8. □

5.3.2 Adequacy

The definition of adequate sets of annotated relations will allow us to carry out an induction in proofs of equivalence.

Definition 5.3.10 (Adequate Sets of Annotated Relations)**.** *A set of annotated relations $\mathcal{X}$ is* adequate *if an only if for all $(\sigma, \sigma', R) \in \mathcal{X}$, all environments $\vdash \rho\ R^{\mathsf{env}}\ \rho'$, and all expressions a, such that $dom(\rho) \vdash a$, we have:*

$$\begin{aligned} \forall \sigma_1, w.\ \sigma; \rho \vdash a \Downarrow w; \sigma_1 \\ \implies \exists \sigma_1', w', Q.\ \sigma'; \rho' \vdash a \Downarrow w'; \sigma_1' \\ \wedge\ \vdash w\ Q\ w' \\ \wedge\ (\sigma_1, \sigma_1', Q) \in \mathcal{X} \\ \wedge\ R \subseteq Q \end{aligned}$$

and similarly for all $(\sigma_1, \sigma_1', R) \in \mathcal{X}^{\mathsf{T}}$.

The union of adequate sets is an adequate set. Thus, the union of all adequate sets is the largest adequate set.

Definition 5.3.11 (Adequacy $(\approx)$)**.** $(\approx)$ *is the largest adequate set of annotated value relations.*

Theorem 5.3.12 (Soundness of Adequate Sets)**.** *If $\mathcal{X}$ is adequate then it is in pre-adequacy.*

Proof. Trivial by the definitions of pre-adequate annotated relations and adequate sets of annotated relations. □

Theorem 5.3.13 (Completeness of Adequate Sets)**.** $(\cong)$ *is adequate.*

Proof. Let $(\sigma, \sigma', R) \in (\cong)$, $\vdash \rho\ R^{\mathsf{env}}\ \rho'$, and $dom(\rho) \vdash a$. We will show that

$$\begin{aligned}
\forall \sigma_1, w.\ \ & \sigma; \rho \vdash a \Downarrow w; \sigma_1 \\
& \implies \exists \sigma_1', w', Q.\ \ \sigma'; \rho' \vdash a' \Downarrow w'; \sigma_1' \\
& \qquad\qquad \wedge\ \vdash w\ Q\ w' \\
& \qquad\qquad \wedge\ (\sigma_1, \sigma_1', Q) \in (\cong) \\
& \qquad\qquad \wedge\ R \subseteq Q
\end{aligned}$$

By the definition of $(\cong)$ and the determinacy of the semantics, it suffices to show that

$$\begin{aligned}
\forall \sigma_1, \sigma_1', w, w'.\ \ & \sigma; \rho \vdash a \Downarrow w; \sigma_1 \\
& \qquad \wedge\ \langle \sigma', \rho' \rangle \Downarrow \langle a', w' \rangle \sigma_1' \\
& \qquad\quad \implies \exists Q.\ \vdash w\ Q\ w' \\
& \qquad\qquad\qquad \wedge\ (\sigma_1, \sigma_1', Q) \in (\cong) \\
& \qquad\qquad\qquad \wedge\ R \subseteq Q
\end{aligned}$$

Let $\sigma; \rho \vdash a \Downarrow w; \sigma_1$ and $\sigma'; \rho' \vdash a \Downarrow w'; \sigma_1'$. We will show that

$$(\sigma_1, \sigma_1', R \cup \{(w, w')\}) \in (\cong)$$

For all $\overline{x, v, v'}$, y, and b such that $\{\overline{x}, y\} \vdash b$, and $\vdash \overline{v}\ R\ \overline{v'}$:

$$\begin{aligned}
& \sigma_1; (\overline{x \mapsto v}, y \mapsto w) \vdash b \Downarrow \\
\iff\ & \sigma; (\overline{x \mapsto v}) \vdash \texttt{let}\, y = a\, \texttt{in}\, b \Downarrow && \text{(by properties of evaluation)} \\
\iff\ & \sigma'; (\overline{x \mapsto v'}) \vdash \texttt{let}\, y = a\, \texttt{in}\, b \Downarrow && ((\sigma, \sigma', R) \in (\cong)) \\
\iff\ & \sigma_1'; (\overline{x \mapsto v'}, y \mapsto w') \vdash b \Downarrow && \text{(by properties of evaluation)}
\end{aligned}$$

Therefore, by Definitions 5.3.5 and 5.3.6:

$$(\sigma_1, \sigma_1', R \cup \{(w, w')\}) \in (\cong)$$

□

Theorem 5.3.14. $(\cong) = (\approx)$.

Proof. By Theorem 5.3.12 we have $(\approx) \subseteq (\cong)$ and by Theorem 5.3.13 we have $(\cong) \subseteq (\approx)$. Thus $(\cong) = (\approx)$. □

From the above we conclude that the open extension of adequacy coincides with the standard definition of contextual equivalence.

Theorem 5.3.15. $(\approx)^\circ = (\equiv)$.

Proof. By Theorems 5.3.9 and 5.3.14. □

5.4 Inductive Proofs of Equivalence

We can prove two expressions of $\text{imp}_\varsigma\ \{\overline{x}\} \vdash a, a'$, contextually equivalent by the following proof method:

1. Find a set of annotated relations $\mathcal{X}$, such that for all stores σ, locations ι, environments ρ, and relations I with:

$$\iota \notin dom(\sigma) \qquad \vdash \rho\ I^{\mathsf{env}}\ \rho \qquad \{\overline{x}\} \subseteq dom(\rho)$$

$$(\sigma[\iota \mapsto \langle \varsigma()\, a, \rho \rangle], \sigma[\iota \mapsto \langle \varsigma()\, a', \rho \rangle], I) \text{ is an identity relation}$$

the following holds:

$$(\sigma[\iota \mapsto \langle \varsigma() \, a, \rho \rangle], \sigma[\iota \mapsto \langle \varsigma() \, a', \rho \rangle], I) \in \mathcal{X}$$

2. show that $\mathcal{X}$ is adequate, and

3. invoke Theorem 5.3.15 to show $\{\overline{x}\} \vdash a \equiv a'$.

An induction principle for proving $\mathcal{X}$ adequate is the following.

Definition 5.4.1.

$$\begin{aligned} IH_{\mathcal{X}}(k) \stackrel{\text{def}}{=} \forall (\sigma, \sigma', R) \in \mathcal{X}. \\ \forall \rho, \rho', a, w, \sigma_1. \; & dom(\rho) \vdash a \\ \wedge \; & \vdash \rho \; R^{\mathsf{env}} \; \rho' \\ \wedge \; & \sigma; \rho \vdash a \Downarrow^k w; \sigma_1 \\ \Longrightarrow \; & \exists w', \sigma_1', Q. \; \sigma'; \rho' \vdash a \Downarrow w'; \sigma_1' \\ \wedge \; & \vdash w \; Q \; w' \\ \wedge \; & (\sigma_1, \sigma_1', Q) \in \mathcal{X} \\ \wedge \; & R \subseteq Q \end{aligned}$$

The measure of the induction is the size, k, of the evaluation $\sigma; \rho \vdash a \Downarrow^k w; \sigma_1$. Hence, proving a set of annotated relations, $\mathcal{X}$, adequate amounts to proving that for all k, $IH_{\mathcal{X}}(k)$ and $IH_{\mathcal{X}^\intercal}(k)$ hold. For $k = 0$ it is trivial; for $k > 0$ it can be shown by induction:

$$\boxed{\begin{array}{l} \forall k. \; IH_{\mathcal{X}}(k-1) \implies IH_{\mathcal{X}}(k) \\ \forall k. \; IH_{\mathcal{X}^\intercal}(k-1) \implies IH_{\mathcal{X}^\intercal}(k) \end{array}}$$

5.5 Deriving Smaller Proof Obligations for Adequate Sets

We factor out the common parts of the two inductions at the end of the previous section via a proof construction scheme, as we did in previous chapters. We consider a hypothetical set of annotated relations, $\mathcal{X}$, and attempt

to prove it adequate. We push the proof as much as possible using the induction hypothesis until the places where extra conditions on $\mathcal{X}$ are needed. These conditions are necessary and sufficient proof obligations for adequacy.

Here we consider the forward inductive proof for the case of method invocation only. The other cases and the reverse induction follow a similar reasoning.

For all k we assume $IH_{\mathcal{X}}(k-1)$ holds. Let $(\sigma, \sigma', R) \in \mathcal{X}$, $\vdash \rho\ R^{\mathsf{env}}\ \rho'$, $dom(\rho) \vdash a$, and $\sigma; \rho \vdash a.l_i \Downarrow^k w; \sigma_1$. We need to prove that for some Q, σ_1', w':

$$\sigma'; \rho' \vdash a.l_i \Downarrow w'; \sigma_1' \qquad \vdash w\ Q\ w' \qquad (\sigma_1, \sigma_1', Q) \in \mathcal{X} \qquad R \subseteq Q$$

The RED SELECT evaluation rule for the left-hand side gives:

$$\frac{\begin{array}{c}\sigma; \rho \vdash a \Downarrow^{k_1} \{\!|\overline{l = \iota}|\!\}; \sigma_1 \qquad l_i \in \{\bar{l}\} \qquad \sigma'(\iota_i) = \langle \varsigma(x)\, b_i, \rho_i \rangle \\ \sigma_1; (\rho_i, x \mapsto \{\!|\overline{l = \iota}|\!\}) \vdash b_i \Downarrow^{k_2} v; \sigma_2\end{array}}{\sigma; \rho \vdash a.l_i \Downarrow^k v; \sigma_2} \tag{5.1}$$

with $1 + k_1 + k_2 = k$.

To show that $\sigma'; \rho' \vdash a.l_i \Downarrow v'; \sigma_1'$ we must establish the premises of RED SELECT for the right-hand side as well.

By $IH_{\mathcal{X}}(k-1)$ and the first premise of (5.1) we get that there exist σ_1', $\overline{l', \iota'}$, Q, such that:

$$\sigma'; \rho' \vdash a \Downarrow \{\!|\overline{l' = \iota'}|\!\}; \sigma_1' \quad Q \supseteq R \quad \vdash \{\!|\overline{l = \iota}|\!\}\ Q\ \{\!|\overline{l' = \iota'}|\!\} \quad (\sigma_1, \sigma_1', Q) \in \mathcal{X} \tag{5.2}$$

We also need to show that $l_i \in \{\overline{l'}\}$ and (since this should hold for all $l_i \in \{\bar{l}\}$) that $\{\bar{l}\} \subseteq \{\overline{l'}\}$. This does not follow from the induction hypothesis and therefore we formulate it as a condition on $\mathcal{X}$:

If $(\sigma, \sigma', R) \in \mathcal{X}$ and $\vdash \{\!|\overline{l = \iota}|\!\}\ R\ \{\!|\overline{l' = \iota'}|\!\}$, then $\{\bar{l}\} \subseteq \{\overline{l'}\}$.

By Lemma 5.1.3 and the first formula of (5.2) we get that $\sigma_1'; \cdot \vdash_{\mathsf{wf}} \{\!|\overline{l' = \iota'}|\!\}$, and therefore there exist b_j', ρ_j', such that $\sigma_1'(\iota_j') = \langle \varsigma(x)\, b_j', \rho_j' \rangle$.

Finally, to show that the right-hand side terminates and prove the inductive step, we require the following condition to hold for $\mathcal{X}$:

If $IH_{\mathcal{X}}(k-1)$ holds, $(\sigma, \sigma', R) \in \mathcal{X}$ and $\vdash \{\!|\overline{l = \iota}|\!\}\ R\ \{\!|\overline{l = \iota'}|\!\}$, then for all $l_j \in \{\bar{l}\}$, with $\sigma(\iota_j) = \langle \varsigma(x)\, b, \rho\rangle$ and $\sigma'(\iota'_j) = \langle \varsigma(x)\, b', \rho'\rangle$:

$$\begin{aligned} & \sigma;\, (\rho, x{\mapsto}\{\!|\overline{l = \iota}|\!\}) \vdash b \Downarrow^{k-1} w;\, \sigma_1 \\ \implies\ & \exists w', \sigma'_1, Q.\ \sigma';\, (\rho', x{\mapsto}\{\!|\overline{l = \iota'}|\!\}) \vdash b' \Downarrow w';\, \sigma'_1 \\ & \qquad \wedge\ \vdash w\ Q\ w' \\ & \qquad \wedge\ (\sigma_1, \sigma'_1, Q) \in \mathcal{X} \\ & \qquad \wedge\ Q \supseteq R \end{aligned}$$

With a similar treatment for the rest of the cases we discover all the proof obligations of $\mathcal{X}$, which we formulate as a theorem for adequate sets of annotated relations.

Theorem 5.5.1. *A set of annotated relations $\mathcal{X}$ is adequate if and only if for all k, assuming that $IH_{\mathcal{X}}(k)$ holds, and all $(s, s', R) \in \mathcal{X}$, and $\vdash \{\!|\overline{l = \iota}|\!\}\ R\ \{\!|\overline{l' = \iota'}|\!\}$ the following conditions are satisfied:*

1. (Method names) $\{\bar{l}\} \subseteq \{\overline{l'}\}$.

2. (Method application) *For any $l_j \in \{\bar{l}\}$ with $\sigma(\iota_j) = \langle \varsigma(x)\, b, \rho\rangle$ and $\sigma'(\iota'_j) = \langle \varsigma(x)\, b', \rho'\rangle$, and any w and σ_1 such that $\sigma;\, (\rho, x{\mapsto}\{\!|\overline{l = \iota}|\!\}) \vdash b \Downarrow^{k} w;\, \sigma_1$, there exist w', σ'_1, $Q \supseteq R$ such that:*

$$\sigma';\, (\rho', x{\mapsto}\{\!|\overline{l = \iota'}|\!\}) \vdash b' \Downarrow w';\, \sigma'_1 \qquad \vdash w\ Q\ w' \qquad (\sigma_1, \sigma'_1, Q) \in \mathcal{X}$$

3. (Method update) *For any $l_j \in \{\bar{l}\}$, $\vdash \rho\ R^{\mathsf{env}}\ \rho'$, $x \notin dom(\rho)$, and expression b with $(dom(\rho) \cup \{x\}) \vdash b$, there exists $Q \supseteq R$ such that:*

$$(\sigma[\iota_j{\mapsto}\langle \varsigma(x)\, b, \rho\rangle], \sigma'[\iota'_j{\mapsto}\langle \varsigma(x)\, b, \rho'\rangle], Q) \in \mathcal{X}$$

4. (Object cloning) *For any $\overline{\iota_1} \notin dom(\sigma)$ and $\overline{\iota'_1} \notin dom(\sigma')$, there exists $Q \supseteq R$ such that:*

$$(\sigma[\overline{\iota_1{\mapsto}\sigma(\iota)}], \sigma'[\overline{\iota'_1{\mapsto}\sigma'(\iota')}], Q) \in \mathcal{X}$$

5. (Object creation) *For any* $\overline{\iota_1} \notin dom(\sigma)$, $\overline{\iota'_1} \notin dom(\sigma')$, *labels* $\overline{l_1}$, *environments* $\vdash \rho\ R^{\mathsf{env}}\ \rho'$, $x \notin dom(\rho)$, *and expressions* $\overline{b}$, *with* $(dom(\rho) \cup \{x\}) \vdash \overline{b}$, *there exists* $Q \supseteq R$ *such that:*

$$\vdash \{\!|\overline{l = \iota_1}|\!\}\ Q\ \{\!|\overline{l = \iota'_1}|\!\} \quad (\sigma[\overline{\iota_1 \mapsto \langle \varsigma(x)\, b, \rho \rangle}], \sigma'[\overline{\iota_1 \mapsto \langle \varsigma(x)\, b, \rho' \rangle}], Q) \in \mathcal{X}$$

Proof. For the forward direction we show that the conditions of the theorem are special cases of the proof obligations of Definition 5.3.10. For the reverse we follow the structure of the proof construction scheme. □

The first two conditions of the theorem are the proof obligations derived by the case of method invocation, as explained above. The third condition is produced by the case of method update, and the fourth by for the case of cloning. The final condition is required for the case of object creation and is similar to the condition for store extension in the calculi of previous chapters. The case of `let` follows immediately from the induction hypothesis and generates no proof obligations for the sets.

5.6 Adequate Relations Up to Store

The last three conditions of Theorem 5.5.1 require that adequate sets take into account all possible stores under which related methods may be invoked; that is, they require adequate sets to be closed under all store operations that can possibly be executed by the context.

These stores can be encoded in a set of annotated relations by a standard construction; it suffices to include the following rules in the construction of

a set $\mathcal{X}$:

$$\frac{(\sigma,\sigma',R)\in\mathcal{X} \qquad \vdash \{\!|\overline{l=\iota}|\!\}\ R\ \{\!|\overline{l=\iota'}|\!\} \qquad l_j\in\{\bar{l}\} \qquad \vdash \rho\ R^{\mathsf{env}}\ \rho' \qquad x\notin dom(\rho) \qquad dom(\rho)\cup\{x\}\vdash b}{(\sigma[\iota_j{\mapsto}\langle\varsigma(x)\,b,\rho\rangle],\sigma'[\iota'_j{\mapsto}\langle\varsigma(x)\,b,\rho'\rangle],R)\in\mathcal{X}}$$

$$\frac{(\sigma,\sigma',R)\in\mathcal{X} \qquad \vdash \{\!|\overline{l=\iota}|\!\}\ R\ \{\!|\overline{l=\iota'}|\!\} \qquad \overline{\sigma(\iota)=\langle\varsigma(x)\,b,\rho\rangle} \qquad \overline{\sigma'(\iota')=\langle\varsigma(x)\,b',\rho'\rangle} \qquad \overline{\iota_1}\notin dom(\sigma) \qquad \overline{\iota'_1}\notin dom(\sigma')}{(\sigma[\overline{\iota_1{\mapsto}\langle\varsigma(x)\,b,\rho\rangle}],\sigma'[\overline{\iota'_1{\mapsto}\langle\varsigma(x)\,b,\rho'\rangle}],R\cup\{(\{\!|\overline{l=\iota_1}|\!\},\{\!|\overline{l=\iota_1}|\!\})\})\in\mathcal{X}}$$

$$\frac{(\sigma,\sigma',R)\in\mathcal{X} \qquad \vdash \rho\ R^{\mathsf{env}}\ \rho' \qquad \overline{\iota_1}\notin dom(\sigma) \qquad \overline{\iota'_1}\notin dom(\sigma') \qquad x\notin dom(\rho) \qquad dom(\rho)\cup\{x\}\vdash \overline{b}}{(\sigma[\overline{\iota_1{\mapsto}\langle\varsigma(x)\,b,\rho\rangle}],\sigma'[\overline{\iota'_1{\mapsto}\langle\varsigma(x)\,b,\rho'\rangle}],R\cup\{(\{\!|\overline{l=\iota_1}|\!\},\{\!|\overline{l=\iota_1}|\!\})\})\in\mathcal{X}}$$

The above rules introduce in $\mathcal{X}$ new annotated relations, together with the obligation of proving Condition 2 of Theorem 5.5.1 for the newly related methods. These proof obligations, though, can immediately be discharged by the induction hypothesis due to the fact that they have the same bodies and related stored environments.[3]

In this section we use an *up-to store* technique to remove the need of adding the above three rules in the construction of adequate sets of annotated relations, and also remove altogether the three conditions involving store operations from the requirements of adequate relations.

To do this, we will allow the construction of *partial sets of annotated relations* that can be combined into complete sets.

5.6.1 Partial Annotated Relations

We start by defining a set of *abstract values*, $\mathcal{A}$, disjoint from all other syntactic domains of imp$_\varsigma$; χ and ψ range over abstract values. *Abstract envi-*

[3] In the languages of previous chapters, closing the relations under the context's store operations (extension, update) did not introduce much complication in constructing the sets of annotated relations. More importantly, because of the R^{cxt}, it did not introduce the obligation to prove more values equivalent.

ronments are finite maps from identifiers to the union of values and abstract values, and *abstract stores* are finite maps from locations to stored methods containing abstract environments; $\hat{\rho}$ and $\hat{\sigma}$ range over abstract environments and abstract stores, respectively.

A *partial annotated relation* (PAR), $(\hat{\sigma}, \hat{\sigma}', R)_{\overline{\chi}}$, is a binary relation on object values, annotated with two abstract stores $\hat{\sigma}$ and $\hat{\sigma}'$. The abstract values contained in environments stored in $\hat{\sigma}$ and $\hat{\sigma}'$ are in $\{\overline{\chi}\}$.

A PAR relates objects stored in two partial stores. The abstract values in these stores correspond to pointers to related objects known to the context, possibly stored outside of the domain of the annotating stores. These pointers become concrete when we combine two PARs.

Definition 5.6.1 (Combination of PARs)**.** *Let* $(\hat{\sigma}_1, \hat{\sigma}'_1, R_1)_{\overline{\chi_1}}$ *and* $(\hat{\sigma}_2, \hat{\sigma}'_2, R_2)_{\overline{\chi_2}}$ *be PARs; then:*

$$
\begin{aligned}
&(\hat{\sigma}_1, \hat{\sigma}'_1, R_1)_{\overline{\chi_1}} \bowtie (\hat{\sigma}_2, \hat{\sigma}'_2, R_2)_{\overline{\chi_2}} \stackrel{\text{def}}{=} \\
&\quad \{(\sigma_1\sigma_2,\ \sigma'_1\sigma'_2,\ R_1 \cup R_2) \mid dom(\hat{\sigma}_1) \cap dom(\hat{\sigma}_2) = \emptyset \\
&\qquad\qquad dom(\hat{\sigma}'_1) \cap dom(\hat{\sigma}'_2) = \emptyset \\
&\qquad\qquad \vdash \overline{v}, \overline{u}\ (R_1 \cup R_2)\ \overline{v'}, \overline{u'} \\
&\qquad\qquad \sigma_1 = \hat{\sigma}_1[\overline{v/\chi_1}] \quad \sigma'_1 = \hat{\sigma}'_1[\overline{v'/\chi_1}] \\
&\qquad\qquad \sigma_2 = \hat{\sigma}_2[\overline{u/\chi_2}] \quad \sigma'_2 = \hat{\sigma}'_2[\overline{u'/\chi_2}]\}
\end{aligned}
$$

Here we use the notation $\hat{\sigma}[\overline{v/\chi}]$ to denote the substitution of the values $\overline{v}$ for the abstract values $\overline{\chi}$ in the environments stored in σ.

Combining two PARs produces a set of annotated relations with stores that are the concatenation of the stores in the two PARs and a binary relation that is the union of the binary relations of the PARs. All abstract values in the stores of the PARs are substituted with objects related by the resulting annotated relation; i.e. objects known to the context. If two PARs contain stores with overlapping domains, then their combination will be the empty set.

Combination of PARs is extended to sets of PARs as follows.

Definition 5.6.2 (Combination of sets of PARs)**.** *Let $\hat{\mathcal{X}}$ and $\hat{\mathcal{Y}}$ be sets of PARs; then:*

$$\begin{aligned}\hat{\mathcal{X}} \bowtie \hat{\mathcal{Y}} \stackrel{\text{def}}{=} \{(\sigma, \sigma', R) \mid\ & (\sigma, \sigma', R) \in (\hat{\sigma}_1, \hat{\sigma}'_1, R_1)_{\overline{\chi_1}} \bowtie (\hat{\sigma}_2, \hat{\sigma}'_2, R_2)_{\overline{\chi_2}} \\ & (\hat{\sigma}_1, \hat{\sigma}'_1, R_1)_{\overline{\chi_1}} \in \hat{\mathcal{X}} \\ & (\hat{\sigma}_2, \hat{\sigma}'_2, R_2)_{\overline{\chi_2}} \in \hat{\mathcal{Y}}\}\end{aligned}$$

To illustrate the use of PARs for constructing adequate sets of abstract relations, consider the following object implementing a single-location cell:

$$\begin{aligned}M \stackrel{\text{def}}{=}\ & \texttt{let}\ o = \{\!|y{=}\varsigma(s)\ s|\!\} \\ & \texttt{in}\ \{\!|arg{=}\varsigma(s)\ s, \\ & \qquad set{=}\varsigma(s)\ \texttt{let}\ z = s.arg \\ & \qquad\qquad \texttt{in}\ o.y \Leftarrow \varsigma()\ z, \\ & \qquad get{=}\varsigma()\ o.y|\!\}\end{aligned}$$

When evaluated, the above `let`-expression produces two objects: a local object with a single method y, and an object with three methods, arg, set, and get, that is returned to the context. By invoking the set method, the context can assign a new value to the cell, which is stored in method y of the private object. Passing the argument to the set method is encoded by updating the arg method. The current value stored in the cell can be retrieved by invoking the get method.

The part of the store involving the above implementation contains the methods of both the private object and the cell object returned to the context. Furthermore, the environments stored in arg and y can potentially contain the cell object itself or any other objects known to the context with methods residing in other parts of the store.

Let us now consider for simplicity the proof of $\vdash M \equiv M$. According to Section 5.4, we would have to define an appropriate adequate set of annotated relations. Instead of defining this set directly, it is easier to define it as the combination of two sets of PARs: one containing relations annotated with abstract stores containing only the locations of the cell object,

and another containing relations annotated with abstract stores containing the locations of all other objects in the context. In each PAR, the places where objects known to the context may appear (e.g., inside the environments of the stored methods for arg and y) are encoded as abstract values, and "linked" to actual values when two PARs are combined. The set of PARs that corresponds to a cell object is the following.

$$\begin{aligned}\hat{\mathcal{X}} \stackrel{\text{def}}{=} \{(\hat{\sigma}, \hat{\sigma}', R)_{\overline{\chi}} \mid \exists \iota_y, \iota_{arg}, \iota_{set}, \iota_{get}, \iota'_y, \iota'_{arg}, \iota'_{set}, \iota'_{get}, a, \overline{x}.\\ \hat{\sigma} = [\ \iota_y \mapsto \langle \varsigma(s)\, x_1, (x_1 \mapsto \chi_1)\rangle,\\ \iota_{arg} \mapsto \langle \varsigma(s)\, a, (\overline{x \mapsto \chi})\rangle,\\ \iota_{set} \mapsto \langle \varsigma(s)\, set, (o \mapsto \{\!|y = \iota_y|\!\})\rangle,\\ \iota_{get} \mapsto \langle \varsigma(s)\, get, (o \mapsto \{\!|y = \iota_y|\!\})\rangle]\\ \hat{\sigma}' = [\ \iota'_y \mapsto \langle \varsigma(s)\, x_1, (x_1 \mapsto \chi_1)\rangle,\\ \iota'_{arg} \mapsto \langle \varsigma(s)\, a, (\overline{x \mapsto \chi})\rangle,\\ \iota'_{set} \mapsto \langle \varsigma(s)\, set, (o \mapsto \{\!|y = \iota'_y|\!\})\rangle,\\ \iota'_{get} \mapsto \langle \varsigma(s)\, get, (o \mapsto \{\!|y = \iota'_y|\!\})\rangle]\\ R = \{(\ \{\!|arg = \iota_{arg}, set = \iota_{set}, get = \iota_{get}|\!\},\\ \{\!|arg = \iota'_{arg}, set = \iota'_{set}, get = \iota'_{get}|\!\})\}\\ \{\overline{x}\} \vdash a\}\end{aligned}$$

At the end of this section we give the definition of the set $\hat{\mathcal{I}}$ of PARs corresponding to the objects created by the context. The set $\hat{\mathcal{X}} \bowtie \hat{\mathcal{I}}$ gives us the set of all combinations of PARs in $\hat{\mathcal{X}}$ and $\hat{\mathcal{I}}$, where all $\overline{\chi}$ in each combination are replaced by objects related by the combination.

The combination of two PARs is symmetric and $(\,)^{\mathsf{T}}$ distributes over $\bowtie$.

Lemma 5.6.3.

1. $(\hat{\sigma}_1, \hat{\sigma}'_1, R_1)_{\overline{\chi_1}} \bowtie (\hat{\sigma}_2, \hat{\sigma}'_2, R_2)_{\overline{\chi_2}} = (\hat{\sigma}_2, \hat{\sigma}'_2, R_2)_{\overline{\chi_2}} \bowtie (\hat{\sigma}_1, \hat{\sigma}'_1, R_1)_{\overline{\chi_1}}$
2. $((\hat{\sigma}_1, \hat{\sigma}'_1, R_1)_{\overline{\chi_1}} \bowtie (\hat{\sigma}_2, \hat{\sigma}'_2, R_2)_{\overline{\chi_2}})^{\mathsf{T}} = (\hat{\sigma}'_1, \hat{\sigma}_1, R_1^{\mathsf{T}})_{\overline{\chi_1}} \bowtie (\hat{\sigma}'_2, \hat{\sigma}_2, R_2^{\mathsf{T}})_{\overline{\chi_2}}$

We also define the *extension* of a PAR by another PAR that can be derived by the former using cloning and update operations.

Definition 5.6.4 (PAR Extension)**.** *The PAR* $(\hat{\sigma}_1, \hat{\sigma}'_1, R_1)_{\overline{\chi_1}}$ *is an extension of* $(\hat{\sigma}, \hat{\sigma}', R)_{\overline{\chi}}$*, written* $(\hat{\sigma}, \hat{\sigma}', R)_{\overline{\chi}} \sqsubseteq (\hat{\sigma}_1, \hat{\sigma}'_1, R_1)_{\overline{\chi_1}}$*, if and only if it satisfies the following rules.*

$\sqsubseteq$-BASE

$$\frac{}{(\hat{\sigma}, \hat{\sigma}', R)_{\overline{\chi}} \sqsubseteq (\hat{\sigma}, \hat{\sigma}', R)_{\overline{\chi}}}$$

$\sqsubseteq$-UPDATE

$$\frac{(\hat{\sigma}_0, \hat{\sigma}'_0, R_0)_{\overline{\chi_0}} \sqsubseteq (\hat{\sigma}, \hat{\sigma}', R)_{\overline{\chi}} \qquad \vdash \{\!|\overline{l = \iota}|\!\}\ R\ \{\!|\overline{l = \iota'}|\!\} \qquad l_j \in \{\bar{l}\} \qquad \hat{\rho} = \overline{x \mapsto \psi} \qquad x \notin dom(\hat{\rho}) \qquad dom(\hat{\rho}) \cup x \vdash b}{(\hat{\sigma}_0, \hat{\sigma}'_0, R_0)_{\overline{\chi_0}} \sqsubseteq (\hat{\sigma}[\iota_j \mapsto \langle \varsigma(x)\, b, \hat{\rho}\rangle], \hat{\sigma}'[\iota'_j \mapsto \langle \varsigma(x)\, b, \hat{\rho}\rangle], R)_{\overline{\chi}\overline{\psi}}}$$

$\sqsubseteq$-CLONE

$$\frac{(\hat{\sigma}_0, \hat{\sigma}'_0, R_0)_{\overline{\chi_0}} \sqsubseteq (\hat{\sigma}, \hat{\sigma}', R)_{\overline{\chi}} \qquad \vdash \{\!|\overline{l = \iota}|\!\}\ R\ \{\!|\overline{l = \iota'}|\!\} \qquad \hat{\sigma}(\bar{\iota}) = \overline{\hat{m}} \qquad \hat{\sigma}'(\overline{\iota'}) = \overline{\hat{m}'} \qquad \overline{\iota_1} \notin dom(\hat{\sigma}) \qquad \overline{\iota'_1} \notin dom(\hat{\sigma}')}{(\hat{\sigma}_0, \hat{\sigma}'_0, R_0)_{\overline{\chi_0}} \sqsubseteq (\hat{\sigma}[\overline{\iota_1 \mapsto \hat{m}}], \hat{\sigma}'[\overline{\iota'_1 \mapsto \hat{m}'}], R \cup \{(\overline{\{\!|l = \iota_1|\!\}}, \overline{\{\!|l = \iota'_1|\!\}})\})_{\overline{\chi}} \in (\hat{\mathcal{X}}^{\sqsubseteq})}$$

Judgment $\sqsubseteq$-BASE admits the identity extension. Judgment $\sqsubseteq$-UPDATE updates related methods of related objects with a pair of methods that have the same body and abstract environment. The new environment has bindings to abstract values only; this is because after combining with another PAR, these abstract values will be replaced with arbitrary related objects from the resulting annotated relation. The judgment $\sqsubseteq$-CLONE clones any related object and extends the stores and the relation accordingly.

We reason about sets of PARs. We write $(\hat{\mathcal{X}}^{\sqsubseteq})$ for the set containing all extensions of PARs in $\hat{\mathcal{X}}$. The cartesian combination of two sets of PARs, $\hat{\mathcal{X}}$ and $\hat{\mathcal{Y}}$, is the set of annotated relations $\hat{\mathcal{X}} \bowtie \hat{\mathcal{Y}}$.

We define $\hat{\mathcal{I}}$, the set of all PARs relating objects with identical stored methods. $\hat{\mathcal{I}}$ relates objects and partial stores that might be created by the context.

Definition 5.6.5. *The set of PARs* $\hat{\mathcal{I}}$ *is defined inductively by the following*

rules.

$\hat{\mathcal{I}}$-EMPTY

$$\frac{}{([],[],\emptyset) \in \hat{\mathcal{I}}}$$

$\hat{\mathcal{I}}$-EXTEND

$$\frac{(\hat{\sigma},\hat{\sigma}',I)_{\overline{\chi}} \in \hat{\mathcal{I}} \qquad \overline{\iota} \notin \hat{\sigma} \qquad \overline{\iota'} \notin \hat{\sigma}' \qquad \hat{\rho} = \overline{x \mapsto \psi} \qquad y \notin \{\overline{x}\} \qquad \{\overline{x}, y\} \vdash b}{(\hat{\sigma}[\overline{\iota \mapsto \langle \varsigma(y)\, b, \hat{\rho}\rangle}], \hat{\sigma}'[\overline{\iota' \mapsto \langle \varsigma(y)\, b, \hat{\rho}\rangle}], I \cup \{(\{\!|\overline{l = \iota}|\!\}, \{\!|\overline{l = \iota'}|\!\})\})_{\overline{\chi}\overline{\psi}} \in \hat{\mathcal{I}}}$$

The environments in stored methods in $\hat{\mathcal{I}}$ bind variables to abstract values only, as in the judgment $\sqsubseteq$-UPDATE of Definition 5.6.4. When any of the PARs in $\hat{\mathcal{I}}$ is combined with another PAR, these abstract values will be replaced by objects related in the resulting annotated relation (i.e. objects known by the context).

5.6.2 Deriving Conditions For Adequate Sets of PARs

If $\hat{\mathcal{X}}$ is a set of PARs, then $(\hat{\mathcal{X}}^{\sqsubseteq}) \bowtie \hat{\mathcal{I}}$ is a set of annotated relations that satisfies the last three conditions of Theorem 5.5.1. Although we can prove this fact directly, we choose to use again a proof construction scheme and find all necessary and sufficient conditions for $(\hat{\mathcal{X}}^{\sqsubseteq}) \bowtie \hat{\mathcal{I}}$ to be adequate.

We want that for all k, $IH_{(\hat{\mathcal{X}}^{\sqsubseteq}) \bowtie \hat{\mathcal{I}}}(k)$ holds.

For all k we assume $IH_{(\hat{\mathcal{X}}^{\sqsubseteq}) \bowtie \hat{\mathcal{I}}}(k-1)$ holds. Let

$$(\hat{\sigma},\hat{\sigma}',R)_{\overline{\chi}} \in (\hat{\mathcal{X}}^{\sqsubseteq}) \qquad (\hat{\sigma}_0,\hat{\sigma}'_0,I)_{\overline{\psi}} \in \hat{\mathcal{I}}$$

$$(\sigma\sigma_0,\ \sigma'\sigma'_0,\ R \cup I) \in (\hat{\sigma},\hat{\sigma}',R)_{\overline{\chi}} \bowtie (\hat{\sigma}_0,\hat{\sigma}'_0,I)_{\overline{\psi}}$$

$$\vdash \rho\ (R \cup I)^{\mathsf{env}}\ \rho'$$

$$dom(\rho) \vdash a$$

$$\sigma\sigma_0; \rho \vdash a \Downarrow^k w; \sigma_1$$

We need to show that for some Q, σ'_1, w':

$$\sigma'\sigma'_0; \rho' \vdash a \Downarrow w'; \sigma'_1 \qquad \vdash w\ Q\ w' \qquad (\sigma_1,\sigma'_1,Q) \in (\hat{\mathcal{X}}^{\sqsubseteq}) \bowtie \hat{\mathcal{I}} \qquad R \cup I \subseteq Q$$

By the definition of $(\hat{\mathcal{X}}^{\sqsubseteq}) \bowtie \hat{\mathcal{I}}$, we need to show that for some Q, J, σ_{10}, σ_{11}, σ'_{10}, σ'_{11}, $\hat{\sigma}_{10}$, $\hat{\sigma}_{11}$, $\hat{\sigma}'_{10}$, $\hat{\sigma}'_{11}$, $\overline{\chi_1}$, $\overline{\psi_1}$, w':

$$\begin{gathered}\sigma_1 = \sigma_{11}\sigma_{10} \qquad \sigma'\sigma'_0; \rho' \vdash a \Downarrow w'; \sigma'_{11}\sigma'_{10} \qquad \vdash w\ (Q \cup J)\ w' \\ (\sigma_{11}\sigma_{10},\ \sigma'_{11}\sigma'_{10},\ Q \cup J) \in (\hat{\sigma}_{11}, \hat{\sigma}'_{11}, Q)_{\overline{\chi_1}} \bowtie (\hat{\sigma}_{10}, \hat{\sigma}'_{10}, J)_{\overline{\psi_1}} \\ (\hat{\sigma}_{11}, \hat{\sigma}'_{11}, Q)_{\overline{\chi_1}} \in (\hat{\mathcal{X}}^{\sqsubseteq}) \qquad (\hat{\sigma}_{10}, \hat{\sigma}'_{10}, J)_{\overline{\psi_1}} \in \hat{\mathcal{I}} \\ R \cup I \subseteq Q \cup J\end{gathered} \tag{5.3}$$

Even though the proof obligation seems more complex than the one in Section 5.5, it makes most of the proof to go through without any extra conditions on $\hat{\mathcal{X}}$. In fact, the only case that leaves such conditions to be proved is the case of method invocation. The rest of the cases are all taken care of by the induction hypothesis and our up-to store construction, and do not produce any more conditions on $\hat{\mathcal{X}}$. We show the case of method update as an example.

Let $a = a_0.l \Leftarrow \varsigma(x)\, b$. We need to prove (5.3). The RED UPDATE evaluation rule for the left-hand side gives:

$$\frac{\sigma\sigma_0; \rho \vdash a_0 \Downarrow^{k-1} \{\!|\overline{l = \iota}|\!\}; \sigma_2 \qquad l_j \in \{\bar{l}\}}{\sigma\sigma_0; \rho \vdash a_0.l_j \Leftarrow \varsigma(x)\, b \Downarrow^{k} \{\!|\overline{l = \iota}|\!\}; \sigma_2[\iota_j \mapsto \langle \varsigma(x)\, b, \rho \rangle]} \tag{5.4}$$

To show that $\sigma'\sigma'_0; \rho' \vdash a \Downarrow w'; \sigma'_{11}\sigma'_{10}$ we must establish the premises of RED UPDATE for the right-hand side as well.

By $IH_{(\hat{\mathcal{X}}^{\sqsubseteq}) \bowtie \hat{\mathcal{I}}}(k-1)$ and the first premise of (5.4) we get that there exist Q, J, σ_{20}, σ_{21}, σ'_{20}, σ'_{21}, $\hat{\sigma}_{20}$, $\hat{\sigma}_{21}$, $\hat{\sigma}'_{20}$, $\hat{\sigma}'_{21}$, $\overline{\chi_2}$, $\overline{\psi_2}$, w', such that:

$$\begin{gathered}\sigma_2 = \sigma_{21}\sigma_{20} \qquad \sigma'\sigma'_0; \rho' \vdash a_0 \Downarrow w'; \sigma'_{21}\sigma'_{20} \qquad \vdash \{\!|\overline{l = \iota}|\!\}\ (Q \cup J)\ w' \\ (\sigma_{21}\sigma_{20},\ \sigma'_{21}\sigma'_{20},\ Q \cup J) \in (\hat{\sigma}_{21}, \hat{\sigma}'_{21}, Q)_{\overline{\chi_2}} \bowtie (\hat{\sigma}_{20}, \hat{\sigma}'_{20}, J)_{\overline{\psi_2}} \\ (\hat{\sigma}_{21}, \hat{\sigma}'_{21}, Q)_{\overline{\chi_2}} \in (\hat{\mathcal{X}}^{\sqsubseteq}) \qquad (\hat{\sigma}_{20}, \hat{\sigma}'_{20}, J)_{\overline{\psi_2}} \in \hat{\mathcal{I}} \\ R \cup I \subseteq Q \cup J\end{gathered}$$

It must be that for some $\overline{l', \iota'}$, $w = \{\!|\overline{l' = \iota'}|\!\}$, and by the condition derived from the case of method invocation, $\{\bar{l}\} \subseteq \{\overline{l'}\}$. Hence:

$$\sigma'\sigma'_0; \rho' \vdash a_0.l \Leftarrow \varsigma(x)\, b \Downarrow \{\!|\overline{l' = \iota'}|\!\}; (\sigma'_{21}\sigma'_{20})[\iota'_j \mapsto \langle \varsigma(x)\, b, \rho' \rangle]$$

By the definition of $\bowtie$ we know that either $\vdash \{\!|\overline{l = \iota}|\!\}\ Q\ \{\!|\overline{l' = \iota'}|\!\}$ or $\vdash \{\!|\overline{l = \iota}|\!\}\ J\ \{\!|\overline{l' = \iota'}|\!\}$.

If $\vdash \{\!|\overline{l = \iota}|\!\}\ J\ \{\!|\overline{l' = \iota'}|\!\}$ then by the definition of $\hat{\mathcal{I}}$ we have that:

$$(\hat{\sigma}_{20}[\iota_j \mapsto \langle \varsigma(x)\, b, (\overline{y \mapsto \psi_2})\rangle], \hat{\sigma}'_{20}[\iota'_j \mapsto \langle \varsigma(x)\, b, (\overline{y \mapsto \psi_2})\rangle], J)_{\overline{\psi_2}} \in \hat{\mathcal{I}}$$

Moreover, since $\vdash \rho\ (R \cup I)^{\mathsf{env}}\ \rho'$ and $R \cup I \subseteq Q \cup J$, we have that $\vdash \rho\ (Q \cup J)^{\mathsf{env}}\ \rho'$ and thus:

$$\begin{aligned}&(\sigma_{21}\sigma_{20}[\iota_j \mapsto \langle \varsigma(x)\, b, \rho\rangle],\ \sigma'_{21}\sigma'_{20}[\iota'_j \mapsto \langle \varsigma(x)\, b, \rho'\rangle],\ Q \cup J)\\ &\in (\hat{\sigma}_{21}, \hat{\sigma}'_{21}, Q)_{\overline{\chi_2}} \bowtie (\hat{\sigma}_{20}[\iota_j \mapsto \langle \varsigma(x)\, b, (\overline{y \mapsto \psi_2})\rangle], \hat{\sigma}'_{20}[\iota'_j \mapsto \langle \varsigma(x)\, b, (\overline{y \mapsto \psi_2})\rangle], J)_{\overline{\psi_2}}\end{aligned}$$

If $\vdash \{\!|\overline{l = \iota}|\!\}\ Q\ \{\!|\overline{l' = \iota'}|\!\}$ then with similar reasoning and using the definition of $(\hat{\mathcal{X}}^{\sqsubseteq})$ we conclude that all proof obligations are discharged and the case of method update is completed.

We similarly discharge the proof obligations for all cases of the proof construction scheme, with the exception of method invocation. For this case the proof follows the structure of the corresponding case explained in Section 5.5, but taking cases on whether methods are related in $\hat{\mathcal{X}}$, $(\hat{\mathcal{X}}^{\sqsubseteq})$, or $\hat{\mathcal{I}}$.

When the methods are related in $\hat{\mathcal{I}}$, the proof goes through by the induction hypothesis.

Next, we examine methods related in $(\hat{\mathcal{X}}^{\sqsubseteq})$ by an inner induction on $\sqsubseteq$. When the related methods are introduced by the $\sqsubseteq$-UPDATE judgment, then the outer induction hypothesis, $IH_{(\hat{\mathcal{X}}^{\sqsubseteq}) \bowtie \hat{\mathcal{I}}}(k-1)$, is sufficient for proving this case. When the related methods are introduced by the $\sqsubseteq$-CLONE judgment we argue that since the objects and methods being cloned satisfy the conditions of the proof (due to the inner induction hypothesis), then their cloned versions satisfy them, too.[4]

The only case left is for when methods and objects are related in the core $\hat{\mathcal{X}}$. This is the case that generates proof obligations for $\hat{\mathcal{X}}$, which we show in the following theorem.

[4]We omit a lemma stating that a cloned object shares the same behavior as the original object.

Theorem 5.6.6. *If $\hat{\mathcal{X}}$ is a set of partial annotated relations, then the set $(\hat{\mathcal{X}}^{\sqsubseteq}) \bowtie \hat{\mathcal{I}}$ is adequate if and only if, when $IH_{(\hat{\mathcal{X}}^{\sqsubseteq}) \bowtie \hat{\mathcal{I}}}(k)$ holds and:*

$$(\hat{\sigma}, \hat{\sigma}', R)_{\overline{\chi}} \in \hat{\mathcal{X}} \qquad (\hat{\sigma}, \hat{\sigma}', R)_{\overline{\chi}} \sqsubseteq (\hat{\sigma}_1, \hat{\sigma}'_1, R_1)_{\overline{\chi_1}} \qquad (\hat{\sigma}_0, \hat{\sigma}'_0, I)_{\overline{\psi}} \in \hat{\mathcal{I}}$$
$$(\sigma_1\sigma_0,\ \sigma'_1\sigma'_0,\ R_1 \cup I) \in (\hat{\sigma}_1, \hat{\sigma}'_1, R_1)_{\overline{\chi_1}} \bowtie (\hat{\sigma}_0, \hat{\sigma}'_0, I)_{\overline{\psi}}$$
$$\vdash \{\!|\overline{l = \iota}|\!\}\ R\ \{\!|\overline{l' = \iota'}|\!\}$$

the following conditions are satisfied:

1. $\{\bar{l}\} \subseteq \{\overline{l'}\}$.

2. *For any $l_j \in \{\bar{l}\}$ with:*

$$\sigma_1(\iota_j) = (\hat{\sigma}[\overline{v/\chi}])(\iota_j) \ = \langle \varsigma(x)\, b, \rho \rangle$$
$$\sigma'_1(\iota'_j) = (\hat{\sigma}'[\overline{v'/\chi}])(\iota'_j) = \langle \varsigma(x)\, b', \rho' \rangle$$
$$\vdash \overline{v}\ (R_1 \cup I)\ \overline{v'}$$

   and any w and σ_2 such that $\sigma_1\sigma_0; (\rho, x{\mapsto}\{\!|\overline{l = \iota}|\!\}) \vdash b \Downarrow^k w; \sigma_2$, there exist Q, J, σ'_2, $\hat{\sigma}_{20}$, $\hat{\sigma}'_{20}$, $\hat{\sigma}_{21}$, $\hat{\sigma}'_{21}$, $\overline{\chi_2}$, $\overline{\psi_2}$, w' such that:

$$\sigma'_1\sigma'_0; \rho' \vdash b' \Downarrow w'; \sigma'_2 \qquad \vdash\ w\ (Q \cup J)\ w'$$
$$(\sigma_2,\ \sigma'_2,\ Q \cup J) \in (\hat{\sigma}_{21}, \hat{\sigma}'_{21}, Q)_{\overline{\chi_2}} \bowtie (\hat{\sigma}_{20}, \hat{\sigma}'_{20}, J)_{\overline{\psi_2}}$$
$$(\hat{\sigma}_{21}, \hat{\sigma}'_{21}, Q)_{\overline{\chi_2}} \in (\hat{\mathcal{X}}^{\sqsubseteq}) \qquad (\hat{\sigma}_{20}, \hat{\sigma}'_{20}, J)_{\overline{\psi_2}} \in \hat{\mathcal{I}}$$
$$R \cup I \subseteq Q \cup J$$

   Moreover, the same holds for $\hat{\mathcal{X}}^{\mathsf{T}}$.

5.7 Example

We use Theorem 5.6.6 to prove the equivalence of two implementations of a cell.

For this example we consider the extension of imp_ς in the usual way with integers, arithmetic operators, and a conditional statement. Moreover, we

encode methods containing λ-abstractions as follows:

$$\{\!|\cdots, f = \varsigma(s)\,\lambda y.\,a, \cdots|\!\} \stackrel{\text{def}}{=} \{\!|arg = \uparrow, \cdots, f = \varsigma(s)\,a[s.arg/y], \cdots|\!\}$$

where $\uparrow = \varsigma(s)\,s.arg$.

The context passes an argument to the body of method f by updating the arg label and then invoking f. Because the arg label may be updated with an arbitrary method, every argument is potentially arbitrarily complicated and the induction hypothesis is crucial for reasoning about arguments.

We prove equivalent the following two expressions in impς.

$$\begin{aligned} M &\stackrel{\text{def}}{=} \texttt{let}\; o = \{\!|y = \varsigma()\,0|\!\} \\ &\qquad \texttt{in}\; \{\!|arg = \uparrow, set = set_M, get = get_M|\!\} \\ M' &\stackrel{\text{def}}{=} \texttt{let}\; o = \{\!|y_1 = \varsigma()\,0, y_2 = \varsigma()\,0, c = \varsigma()\,0|\!\} \\ &\qquad \texttt{in}\; \{\!|arg = \uparrow, set = set_{M'}, get = get_{M'}|\!\} \end{aligned}$$

where:

$$\begin{aligned} set_M &\stackrel{\text{def}}{=} \varsigma(s)\,\texttt{let}\; z = s.arg \\ &\qquad \texttt{in}\; o.y \Leftarrow \varsigma()\,z \end{aligned} \qquad\qquad get_M \stackrel{\text{def}}{=} \varsigma()\,o.y$$

$$\begin{aligned} set_{M'} &\stackrel{\text{def}}{=} \varsigma(s)\,\texttt{let}\; n = o.c + 1 \\ &\qquad\quad z = s.arg \\ &\qquad \texttt{in}\; (o.c \Leftarrow \varsigma()\,n; \\ &\qquad\quad o.y_1 \Leftarrow \varsigma()\,z; \\ &\qquad\quad o.y_2 \Leftarrow \varsigma()\,z) \end{aligned} \qquad\qquad \begin{aligned} get_{M'} &\stackrel{\text{def}}{=} \varsigma()\,\texttt{let}\; x = \texttt{even?}(o.c) \\ &\qquad \texttt{in}\; \texttt{if}\; x\; \texttt{then}\; o.y_1 \\ &\qquad\quad \texttt{else}\; o.y_2 \end{aligned}$$

Expression M encodes a cell object with a get and a set method for storing and retrieving a value. The value is stored in method y of a private object bound to o. M' also encodes a cell object; it stores, though, the value passed to its set method in two methods of its private object. It also keeps a counter, incremented each time the set is invoked. The get method returns the stored value from one of the methods it is stored in, depending on the parity of the counter.

Proof. According to Section 5.4, to prove M and M' contextually equivalent we need to find a set of annotated relations $\mathcal{X}$, such that for all stores σ, locations ι, environments ρ, and relations I with:

$$\iota \notin dom(\sigma) \qquad \vdash \rho\ I^{\mathsf{env}}\ \rho \qquad \{\overline{x}\} \subseteq dom(\rho)$$
$$(\sigma[\iota{\mapsto}\langle\varsigma()\ a, \rho\rangle], \sigma[\iota{\mapsto}\langle\varsigma()\ a', \rho\rangle], I) \text{ is an identity relation}$$

the following must hold:

$$(\sigma[\iota{\mapsto}\langle\varsigma()\ a, \rho\rangle], \sigma[\iota{\mapsto}\langle\varsigma()\ a', \rho\rangle], I) \in \mathcal{X}$$

Furthermore, we need to show that $\mathcal{X}$ is adequate.

We construct a set of partial relations $\hat{\mathcal{X}}$ and prove that $(\hat{\mathcal{X}}^{\sqsubseteq}) \bowtie \hat{\mathcal{I}}$ satisfies the above requirements. In particular, we construct the set $\hat{\mathcal{X}}$, shown in Figure 5.3. The first rule of the construction relates arbitrary one-method objects with the terms M and M' as the bodies of their methods, as required by the method of Section 5.4. The combination of this PAR with PARs from $\hat{\mathcal{I}}$ will include all possible identity relations on extensions of these one-method stores. The second rule describes the part of the store involving two related cell objects, created by the evaluation of M and M' in an arbitrary context. The objects v and v' are the two related cell objects returned to the context, and u and u' are their private objects, respectively. The abstract stores $\hat{\sigma}_1$, $\hat{\sigma}'_1$, $\hat{\sigma}_2$, and $\hat{\sigma}'_2$ contain the stored methods of v, v', u, and u', respectively. The abstract environment $(\overline{x{\mapsto}\psi_1})$ represents the related environments under which the cell objects where created, and thus it ends up as part of the stored environments of their methods.

It suffices to show that $(\hat{\mathcal{X}}^{\sqsubseteq}) \bowtie \hat{\mathcal{I}}$ is adequate.

By Theorem 5.6.6, we assume that $IH_{(\hat{\mathcal{X}}^{\sqsubseteq}) \bowtie \hat{\mathcal{I}}}(k)$ holds and:

$$(\hat{\sigma}, \hat{\sigma}', R)_{\overline{\chi}} \in \hat{\mathcal{X}} \qquad (\hat{\sigma}, \hat{\sigma}', R)_{\overline{\chi}} \sqsubseteq (\hat{\sigma}_1, \hat{\sigma}'_1, R_1)_{\overline{\chi_1}} \qquad (\hat{\sigma}_0, \hat{\sigma}'_0, I)_{\overline{\psi}} \in \hat{\mathcal{I}}$$
$$(\sigma_1\sigma_0,\ \sigma'_1\sigma'_0,\ R_1 \cup I) \in (\hat{\sigma}_1, \hat{\sigma}'_1, R_1)_{\overline{\chi_1}} \bowtie (\hat{\sigma}_0, \hat{\sigma}'_0, I)_{\overline{\psi}}$$
$$\vdash \{\!|\overline{l = \iota}|\!\}\ R\ \{\!|\overline{l' = \iota'}|\!\}$$

$$\frac{\text{locations } \iota, \iota'}{([\iota \mapsto \langle \varsigma() \, M, (\overline{x \mapsto \chi}) \rangle], [\iota' \mapsto \langle \varsigma() \, M', (\overline{x \mapsto \chi}) \rangle], \{(\{\!|l = \iota|\!\}, \{\!|l = \iota'|\!\})\})_{\overline{\chi}} \in \hat{\mathcal{X}}}$$

$$\frac{\begin{array}{c}(\hat{\sigma}, \hat{\sigma}', R)_{\overline{\chi}} \in \hat{\mathcal{X}} \\ \iota_y, \iota_{arg}, \iota_{set}, \iota_{get} \notin dom(\hat{\sigma}) \qquad \iota'_{y_1}, \iota'_{y_2}, \iota'_c, \iota'_{arg}, \iota'_{set}, \iota'_{get} \notin dom(\hat{\sigma}') \qquad i \in I\!N \\ v = \{\!|arg = \iota_{arg}, set = \iota_{set}, get = \iota_{get}|\!\} \qquad u = \{\!|y = \iota_y|\!\} \\ v' = \{\!|arg = \iota'_{arg}, set = \iota'_{set}, get = \iota'_{get}|\!\} \qquad u' = \{\!|y_1 = \iota'_{y_1}, y_2 = \iota'_{y_2}, c = \iota'_c|\!\} \\ \hat{\sigma}_1 = [\iota_{arg} \mapsto \langle \uparrow, \hat{\rho}_1 \rangle, \iota_{set} \mapsto \langle set_M, \hat{\rho}_1 \rangle, \iota_{get} \mapsto \langle get_M, \hat{\rho}_1 \rangle] \qquad \hat{\rho}_1 = ((\overline{x \mapsto \psi_1}, o \mapsto u)) \\ \hat{\sigma}'_1 = [\iota_{arg} \mapsto \langle \uparrow, \hat{\rho}'_1 \rangle, \iota_{set} \mapsto \langle set_{M'}, \hat{\rho}'_1 \rangle, \iota_{get} \mapsto \langle get_{M'}, \hat{\rho}'_1 \rangle] \qquad \hat{\rho}'_1 = ((\overline{x \mapsto \psi_1}, o \mapsto u')) \\ \hat{\sigma}_2 = [\iota_y \mapsto \langle \varsigma() \, z, \hat{\rho}_2 \rangle] \qquad \hat{\rho}_2 = ((\hat{\rho}_1, z \mapsto \psi)) \\ \hat{\sigma}'_2 = [\iota'_{y_1} \mapsto \langle \varsigma() \, z, \hat{\rho}'_2 \rangle, \iota'_{y_2} \mapsto \langle \varsigma() \, z, \hat{\rho}'_2 \rangle, \iota'_c \mapsto \langle \varsigma() \, n, \hat{\rho}'_1 \rangle] \qquad \hat{\rho}'_2 = ((\hat{\rho}'_1, z \mapsto \psi, n \mapsto i))\end{array}}{(\hat{\sigma}\hat{\sigma}_1\hat{\sigma}_2, \ \hat{\sigma}'\hat{\sigma}'_1\hat{\sigma}'_2, \ R \cup \{(v, v')\})_{\overline{\chi}\overline{\psi_1}\psi} \in \hat{\mathcal{X}}}$$

Figure 5.3: Construction of an up-to store adequate set of partial annotated relations for proving the equivalence of two implementations of a cell object.

We need to prove the two conditions of the theorem. The first is true by construction of $\hat{\mathcal{X}}$. To prove the second condition we consider the cases for $\vdash \{\!|\overline{l = \iota}|\!\} \ R \ \{\!|\overline{l' = \iota'}|\!\}$:

Case $\vdash \{\!|l = \iota|\!\} \ R \ \{\!|l = \iota'|\!\}$, with $\sigma_1(\iota) = (\hat{\sigma}[\overline{v/\chi}])(\iota) = \langle \varsigma() \, M, \rho \rangle$, $\sigma'_1(\iota') = \hat{\sigma}'[\overline{v'/\chi}](\iota') = \langle \varsigma() \, M', \rho' \rangle$, and $\vdash \rho \ (R_1 \cup I)^{\mathsf{env}} \ \rho'$. Let:

$$\sigma_1\sigma_0; x \mapsto \{\!|l = \iota|\!\} \vdash M \Downarrow^k w; \sigma_2$$

It is the case that for some $\iota_{arg}, \iota_{set}, \iota_{get}, \iota_y \notin dom(\sigma_1\sigma_0)$:

$$\begin{aligned} w &= \{\!|arg = \iota_{arg}, set = \iota_{set}, get = \iota_{get}|\!\} \\ \rho_1 &= (\rho, o \mapsto \{\!|y = \iota_y|\!\}) \\ \sigma_2 &= \sigma_{21}\sigma_0 \\ \sigma_{21} &= \sigma_1[\iota_{arg} \mapsto \langle \uparrow, \rho_1 \rangle, \iota_{set} \mapsto \langle \varsigma() \, set_M, \rho_1 \rangle, \iota_{get} \mapsto \langle \varsigma() \, get_M, \rho_1 \rangle, \\ &\qquad \iota_y \mapsto \langle \varsigma() \, z, ((\rho_1, z \mapsto 0)) \rangle] \end{aligned}$$

Furthermore there exist w', σ'_2, $\iota'_{arg}, \iota'_{set}, \iota'_{get}, \iota'_{y_1}, \iota'_{y_2}, \iota'_c \notin dom(\sigma_1\sigma_0)$, such

that:

$$\sigma_1'\sigma_0'; x{\mapsto}\{\!|l = \iota'|\!\} \vdash M' \Downarrow w'; \sigma_2'$$

$$\begin{aligned}
w' &= \{\!|arg = \iota'_{arg}, set = \iota'_{set}, get = \iota'_{get}|\!\}\\
\rho_1' &= (\rho, o{\mapsto}\{\!|y_1 = \iota'_{y_1}, y_2 = \iota'_{y_2}, y_c = \iota'_c|\!\})\\
\sigma_2' &= \sigma_{21}'\sigma_0'\\
\sigma_{21}' &= \sigma_1'[\iota'_{arg}{\mapsto}\langle \varsigma(\uparrow)\, \rho_1', \rangle,\ \iota'_{set}{\mapsto}\langle \varsigma()\, set_{M'}, \rho_1'\rangle,\ \iota'_{get}{\mapsto}\langle \varsigma()\, get_{M'}, \rho_1'\rangle,\\
&\qquad \iota'_{y_1}{\mapsto}\langle \varsigma()\, z, ((\rho_1', z{\mapsto}0, n{\mapsto}0))\rangle,\ \iota'_{y_2}{\mapsto}\langle \varsigma()\, z, ((\rho_1', z{\mapsto}0, n{\mapsto}0))\rangle,\\
&\qquad \iota'_c{\mapsto}\langle \varsigma()\, n, ((\rho_1', z{\mapsto}0, n{\mapsto}0))\rangle]
\end{aligned}$$

Moreover, by the definitions of $\hat{\mathcal{X}}$ and $(\hat{\mathcal{X}}^{\sqsubseteq})$, there exist some $\overline{\chi_2}$ and the appropriate $\hat{\sigma}_{21}$ and $\hat{\sigma}'_{21}$:

$$(\sigma_2,\ \sigma_2',\ R_1 \cup \{(w, w')\} \cup I) \in (\hat{\sigma}_{21}, \hat{\sigma}'_{21}, R_1 \cup \{(w, w')\})_{\overline{\chi\chi_2}} \bowtie (\hat{\sigma}_0, \hat{\sigma}'_0, I)_{\overline{\psi}}$$

$$(\hat{\sigma}_{21}, \hat{\sigma}'_{21}, R_1 \cup \{(w, w')\})_{\overline{\chi\chi_2}} \in (\hat{\mathcal{X}}^{\sqsubseteq}) \qquad (\hat{\sigma}_0, \hat{\sigma}'_0, I)_{\overline{\psi}} \in \hat{\mathcal{I}}$$

Case $\vdash \{\!|arg = \iota_{arg}, set = \iota_{set}, get = \iota_{get}|\!\}\ R\ \{\!|arg = \iota'_{arg}, set = \iota'_{set}, get = \iota'_{get}|\!\}$

Here we will consider only the case of invocation of the set method. The case for get and arg are similar.

$$\begin{aligned}
\sigma_1(\iota_{set}) &= (\hat{\sigma}[\overline{v/\chi}])(\iota_{set}) &&= \langle \varsigma(s)\, set_M, ((\rho, o{\mapsto}u))\rangle\\
\sigma_1'(\iota'_{set}) &= (\hat{\sigma}'[\overline{v'/\chi}])(\iota'_{set}) &&= \langle \varsigma(s)\, set_{M'}, ((\rho', o{\mapsto}u'))\rangle\\
\sigma_1(\iota_{arg}) &= \langle \varsigma(x)\, b, \rho_b\rangle\\
\sigma_1'(\iota'_{arg}) &= \langle \varsigma(x)\, b, \rho_b'\rangle\\
\sigma_1(\iota_y) &= (\hat{\sigma}[\overline{v/\chi}])(\iota_y) &&= m\\
\sigma_1'(\iota'_{y_1}) &= (\hat{\sigma}'[\overline{v'/\chi}])(\iota'_{y_1}) &&= m'\\
\sigma_1'(\iota'_{y_2}) &= (\hat{\sigma}'[\overline{v'/\chi}])(\iota'_{y_2}) &&= m'\\
\sigma_1'(\iota'_c) &= (\hat{\sigma}'[\overline{v'/\chi}])(\iota'_c) &&= i
\end{aligned}$$

$$u = \{\!|y = \iota'_y|\!\} \qquad u' = \{\!|y_1 = \iota'_{y_1}, y_2 = \iota'_{y_2}, y_c = \iota'_c|\!\}$$

$$\vdash \overline{v}\ (R_1 \cup I)\ \overline{v'} \qquad \vdash \overline{\rho}\ (R_1 \cup I)^{\mathsf{env}}\ \overline{\rho'} \qquad \vdash\ \rho_b\ (R_1 \cup I)^{\mathsf{env}}\ \rho_b'$$

Let b_{set} and b'_{set} be the bodies of set_M and $set_{M'}$, respectively, and $\vdash \rho_1\ (R_1 \cup I)\ \rho'_1$. We have:

$$\sigma_1\sigma_0; (\rho_1, s{\mapsto}u) \vdash b_{set} \Downarrow^k u; \sigma_2$$

By the properties of evaluation, there exist v and σ_3 such that:

$$\sigma_1\sigma_0; (\rho_1, s{\mapsto}u) \vdash s.arg \Downarrow^{k-1} v; \sigma_3$$
$$\sigma_2 = \sigma_3[\iota_y{\mapsto}\langle\varsigma()\ z, ((\rho_1, s{\mapsto}u, z{\mapsto}v))\rangle]$$

By $IH_{(\hat{\mathcal{X}}^{\sqsubseteq})\bowtie\hat{\mathcal{I}}}(k-1)$, there exist Q, J, σ'_3, $\hat{\sigma}_{30}$, $\hat{\sigma}'_{30}$, $\hat{\sigma}_{31}$, $\hat{\sigma}'_{31}$, $\overline{\chi_3}$, $\overline{\psi_3}$, v' such that:

$$\sigma'_1\sigma'_0; (\rho'_1, s{\mapsto}u') \vdash s.arg \Downarrow v'; \sigma'_3 \qquad \vdash v\ (Q \cup J)\ v'$$
$$(\sigma_3, \sigma'_3, Q \cup J) \in (\hat{\sigma}_{31}, \hat{\sigma}'_{31}, Q)_{\overline{\chi_3}} \bowtie (\hat{\sigma}_{30}, \hat{\sigma}'_{30}, J)_{\overline{\psi_3}}$$
$$(\hat{\sigma}_{31}, \hat{\sigma}'_{31}, Q)_{\overline{\chi_3}} \in (\hat{\mathcal{X}}^{\sqsubseteq}) \qquad (\hat{\sigma}_{30}, \hat{\sigma}'_{30}, J)_{\overline{\psi_2}} \in \hat{\mathcal{I}}$$
$$R \cup I \subseteq Q \cup J$$

Therefore:

$$\sigma'_1\sigma'_0; (\rho'_1, s{\mapsto}u') \vdash b'_{set} \Downarrow u'; \sigma'_2$$

where:

$$\begin{aligned}\sigma'_2 = \sigma'_3[&\iota'_{y_1}{\mapsto}\langle\varsigma()\ z, ((\rho'_1, s{\mapsto}u', z{\mapsto}v', n{\mapsto}i+1))\rangle,\\ &\iota'_{y_2}{\mapsto}\langle\varsigma()\ z, ((\rho'_1, s{\mapsto}u', z{\mapsto}v', n{\mapsto}i+1))\rangle,\\ &\iota'_c{\mapsto}\langle\varsigma()\ n, ((\rho'_1, s{\mapsto}u', z{\mapsto}v', n{\mapsto}i+1))\rangle]\end{aligned}$$

Moreover, by the definitions of $\hat{\mathcal{X}}$ and $(\hat{\mathcal{X}}^{\sqsubseteq})$, there exist the appropriate $\hat{\sigma}_{21}$ and $\hat{\sigma}'_{21}$ such that:

$$(\sigma_2, \sigma'_2, Q \cup J) \in (\hat{\sigma}_{21}, \hat{\sigma}'_{21}, Q)_{\overline{\chi_3}} \bowtie (\hat{\sigma}_{30}, \hat{\sigma}'_{30}, J)_{\overline{\psi_3}}$$
$$(\hat{\sigma}_{21}, \hat{\sigma}'_{21}, Q)_{\overline{\chi_3}} \in (\hat{\mathcal{X}}^{\sqsubseteq})$$

Hence, by Theorem 5.6.6, $\hat{\mathcal{X}}$ is adequate, therefore $\vdash M \equiv M'$. $\square$

CHAPTER 6

A Java-Like Language

In this chapter we develop our theory for a Java-like language. We do this for consecutive extensions of a base class-based language ($\mathcal{J}_1$), adding first inheritance ($\mathcal{J}_2$) and then downcasting ($\mathcal{J}_3$). We show that these extensions only incrementally affect the derived proof method of equivalence by adding more closure conditions. We also show that interesting equivalences in $\mathcal{J}_2$, involving inheritance, cease to hold with the addition of downcasting.

The operational semantics in this chapter is a small-step semantics, which illustrates that our reasoning technique is independent of the style of semantics. Furthermore, we focus on class equivalence, rather than expression equivalence.

6.1 $\mathcal{J}_1$: A Basic Class-Based Language

We start by defining a small class-based language which we call $\mathcal{J}_1$. This language is a subset of Java containing class definitions, imperative private and public fields, private and public methods, ground types, constants, conditional, and a let expression. However, it does not allow classes to inherit behavior from other classes or to override methods. The syntax of $\mathcal{J}_1$ is shown in Figure 6.1.

The main difference between $\mathcal{J}_1$ and other imperative Java calculi, such as Middleweight Java [12] and Classic Java [17], is that $\mathcal{J}_1$ does not have

CONFIGURATION:	$pconf$	$\in \mathcal{CT} \times$ STORE $\times$ EXPRESSION	
CLASS TABLE:	$\mathcal{CT}$	$\in \mathscr{P}$(CLASS DEF.)	
CLASS DEFN.:	$\mathcal{C}$	$::= \texttt{class}\ C\{\overline{mod\, t\ f}; \mathcal{K}\, \overline{\mathcal{M}}\}$	
CONSTRUCTOR:	$\mathcal{K}$	$::= C()\{\overline{\texttt{this}.f := c}\}$	
METHOD:	$\mathcal{M}$	$::= mod\, t\, m(\overline{t\ x})\{e\}$	
TYPE:	t	$::= \texttt{void} \mid \texttt{int} \mid \texttt{bool} \mid C$	
MODIFIER:	mod	$::= \texttt{public} \mid \texttt{private}$	
EXPRESSION:	e, d	$::= v$	Value
		$\mid\ x$	Identifier
		$\mid\ \texttt{new}\ C$	Instantiation
		$\mid\ e.m(\overline{e})$	Invocation
		$\mid\ e.f$	Field Lookup
		$\mid\ e.f := e$	Field Update
		$\mid\ \texttt{op}(\overline{e}) \mid \texttt{eq}(e, e)$	Operators
		$\mid\ \texttt{let}\ x = e\ \texttt{in}\ e$	Let Binding
		$\mid\ \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e$	Conditional
		$\mid\ \varepsilon$	Error
VALUE:	v, u, w	$::= c \mid l$	
CONSTANT:	c	$::= \texttt{unit} \mid \texttt{null} \mid \texttt{true} \mid \texttt{false}$	
		$\mid\ 0 \mid \pm 1 \mid \pm 2 \mid \ldots$	
LOCATION:	l, k		
STORE:	s	$\in$ LOCATION $\xrightarrow{\text{fin}}$ OBJECT	
OBJECT:	o	$::= \texttt{obj}\, C\, \{\overline{f = v}\}$	
ERROR:	ε	$::= \texttt{nerr}$	Null Error

Figure 6.1: Syntactic domains of of $\mathcal{J}_1$

inheritance and casting. We later extend $\mathcal{J}_1$ with inheritance to create the language $\mathcal{J}_2$, and then we add a casting operator to create $\mathcal{J}_3$. $\mathcal{J}_3$ has the same constructs as Middleweight Java, with the addition of constants and access modifiers. However $\mathcal{J}_3$ does not have the explicit interfaces and mixins of Classic Java.

The values of the language are constants or locations of objects in the

$\mathcal{CT}.C$.super	Returns the name of the immediate super-class of C.
$\mathcal{CT}.C$.constr	Returns the constructor definition of class C.
$\mathcal{CT}.C$.fields	Returns a sequence of the field definitions of class C and its super-classes.
$\mathcal{CT}.C$.methods	Returns a sequence of the method definitions that can be invoked on an instance of class C.

Figure 6.2: Meta-functions for static lookup in class table definitions.

store. Stored objects are structures that contain the name of the class which they instantiate, and a binding for each field of the class to a value of the appropriate type. A program can test for pointer equivalence of objects using the operator `eq`.

The types of $\mathcal{J}_1$ are class types, as well as the ground types `void`, `int`, and `bool`. There is a `unit` constant of type `void`, `true` and `false` of type `bool`, and the integers of type `int`. The constant `null` has any class type in the class-table. $\mathcal{J}_1$ has also a null error (`nerr`), used when a program tries to perform an operation on a `null` value.

Class definitions declare the name of a class, its fields and methods, and its constructor. The `public` and `private` access modifiers in the definitions of fields and methods specify the scope of these names. Public methods and fields are visible to all classes, while private methods and fields are visible only from within the same class. Class tables are sets of class definitions. Well-formed class tables contain class definitions with distinct names.

We will use meta-operations and the dot notation to perform static lookup on class tables and classes. For example $\mathcal{CT}.C$ returns the definition of the class named C from the class table $\mathcal{CT}$, and $\mathcal{CT}.C$.fields returns a sequence of all the field definitions in C. A complete table of these meta-operations

and a description of their functionality is shown in Figure 6.2.

Stores are finite partial functions from locations to objects. A *program configuration*, written $\mathcal{CT} \rhd_C s, e$, is a tuple composed of a class table $\mathcal{CT}$, a store s, a closed expression e, and the class name C in which the expression should be typable. The last is needed to type-check method and field access, and is left implicit when type-checking is not important (e.g., in the operational semantics). An *initial configuration* is a configuration with an empty store ($[]$), and whose expression is typable in the `Object` class.

The typing rules of $\mathcal{J}_1$ are shown in Figure 6.3. The typing judgment for expressions has the form $\mathcal{CT}; \Sigma; \Gamma \vdash_C e : t$. Γ is the type environment, Σ is a store typing, and C is the type of `this`. We write Σ_s for the store typing that has the same domain as s and the type of each location is the class name of the object stored in that location. In these rules we use the meta-functions *accessible* and *wfClassHierarchy*. The former returns the set of fields and methods of a class visible by another class. The latter is a predicate that checks the well-formedness of class tables.

The typing judgments for method, class, and class-table definitions are $\mathcal{CT} \vdash_C \mathcal{M} : \mathsf{OK}$, $\mathcal{CT} \vdash C : \mathsf{OK}$, and $\mathcal{CT} : \mathsf{OK}$, respectively. $(\mathcal{CT} \rhd_C s, e) : \mathsf{OK}$ is the typing judgment for program configurations.

In Figure 6.4 we give a small-step semantics for $\mathcal{J}_1$. A small-step $\mathcal{CT} \rhd s, e \rightarrow s_1, e_1$ describes a transition from the program configuration $\mathcal{CT} \rhd s, e$ to the configuration $\mathcal{CT} \rhd s_1, e_1$. We also define $\rightarrow^*$ to be the reflexive and transitive closure of $\rightarrow$, and $\rightarrow^{<k}$ the reflexive and up to $k-1$ steps transitive closure of $\rightarrow$. Moreover, we write $\mathcal{CT} \rhd s, e \downarrow$ iff there exists s_1, w, such that $\mathcal{CT} \rhd s, e \rightarrow^* s_1, w$.

We use calligraphic font for the meta-identifiers that denote class table, class, constructor, or method definitions. We also use the notation $\overline{f = v} \backslash f_i = u$ to denote the sequence $f_1 = v_1, \cdots, f_{i-1} = v_{i-1}, f_i = u, f_{i+1} = v_{i+1}, \cdots$.

$\boxed{\mathcal{CT}; \Sigma; \Gamma \vdash_D e : t}$

$$\frac{D \in \mathcal{CT}.\mathsf{cnames}}{\mathcal{CT}; \Sigma; \Gamma \vdash_D \mathtt{this} : D} \qquad \frac{}{\mathcal{CT}; \Sigma; \Gamma \vdash_D \mathtt{unit} : \mathtt{void}} \qquad \frac{c \in \{\mathtt{true}, \mathtt{false}\}}{\mathcal{CT}; \Sigma; \Gamma \vdash_D \overline{c : \mathtt{bool}}}$$

$$\frac{c \in \{0, \pm 1, \pm 2, \dots\}}{\mathcal{CT}; \Sigma; \Gamma \vdash_D c : \mathtt{int}} \qquad \frac{C \in \mathcal{CT}.\mathsf{cnames}}{\mathcal{CT}; \Sigma; \Gamma \vdash_D \mathtt{null} : C} \qquad \frac{x : t \in \Gamma}{\mathcal{CT}; \Sigma; \Gamma \vdash_D x : t}$$

$$\frac{C \in \mathcal{CT}.\mathsf{cnames} \quad \Sigma(l) = C}{\mathcal{CT}; \Sigma; \Gamma \vdash_D l : C} \qquad \frac{C \in \mathcal{CT}.\mathsf{cnames}}{\mathcal{CT}; \Sigma; \Gamma \vdash_D \mathtt{new}\ C : C} \qquad \frac{\mathcal{CT}; \Sigma; \Gamma \vdash_D \overline{e : t_0} \quad \mathtt{op}.\mathit{type} = \overline{t_0} \to t}{\mathcal{CT}; \Sigma; \Gamma \vdash_D \mathtt{op}(\overline{e}) : t}$$

$$\frac{\mathcal{CT}; \Sigma; \Gamma \vdash_D e_1 : t_1 \quad \mathcal{CT}; \Sigma; \Gamma, x : t_1 \vdash_D e : t}{\mathcal{CT}; \Sigma; \Gamma \vdash_D \mathtt{let}\ x = e_1\ \mathtt{in}\ e : t} \qquad \frac{\mathcal{CT}; \Sigma; \Gamma \vdash_D e : C \quad \mathit{mod}\, t\ f \in \mathcal{CT}.C.\mathsf{fields} \quad f \in \mathit{accessible}(\mathcal{CT}, C, D)}{\mathcal{CT}; \Sigma; \Gamma \vdash_D e.f : t}$$

$$\frac{\mathcal{CT}; \Sigma; \Gamma \vdash_D e : C \quad \mathit{mod}\, t\ f \in \mathcal{CT}.C.\mathsf{fields} \quad f \in \mathit{accessible}(\mathcal{CT}, C, D) \quad \mathcal{CT}; \Sigma; \Gamma \vdash_D e_1 : t}{\mathcal{CT}; \Sigma; \Gamma \vdash_D e.f := e_1 : \mathtt{void}} \qquad \frac{\mathcal{CT}; \Sigma; \Gamma \vdash_D e : C \quad \mathit{mod}\, \overline{t_0} \to t\ m \in \mathcal{CT}.C.\mathsf{methods} \quad m \in \mathit{accessible}(\mathcal{CT}, C, D) \quad \mathcal{CT}; \Sigma; \Gamma \vdash_D \overline{e_0 : t_0}}{\mathcal{CT}; \Sigma; \Gamma \vdash_D e.m(\overline{e_0}) : t}$$

$$\frac{\mathcal{CT}; \Sigma; \Gamma \vdash_D e_1 : \mathtt{bool} \quad \mathcal{CT}; \Sigma; \Gamma \vdash_D e_2 : t \quad \mathcal{CT}; \Sigma; \Gamma \vdash_D e_3 : t}{\mathcal{CT}; \Sigma; \Gamma \vdash_D \mathtt{if}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ e_3 : t}$$

$\boxed{\mathcal{CT} \vdash_C \mathcal{M} : \mathsf{OK}}$

$$\frac{\mathcal{CT}; \emptyset; \overline{x : t_1} \vdash_C e : t}{\mathcal{CT} \vdash_C \mathit{mod}\, t\ m(\overline{t_1\ x})\{e\} : \mathsf{OK}}$$

$\boxed{\mathcal{CT} \vdash \mathcal{C} : \mathsf{OK}}$

$$\frac{\mathcal{K} = C()\{\overline{\mathtt{this}.f := c}\} \quad \mathcal{CT}; \emptyset; \cdot \vdash_C \overline{c : t} \quad \mathcal{CT} \vdash_C \overline{\mathcal{M}} : \mathsf{OK}}{\mathcal{CT} \vdash \mathtt{class}\ C\{\overline{t\ f}; \mathcal{K}\, \overline{\mathcal{M}}\} : \mathsf{OK}}$$

$\boxed{\mathcal{CT} : \mathsf{OK}}$

$$\frac{\{\overline{\mathcal{C}}\} \vdash \overline{\mathcal{C}} : \mathsf{OK} \quad \mathit{wfClassHierarchy}(\{\overline{\mathcal{C}}\})}{\{\overline{\mathcal{C}}\} : \mathsf{OK}}$$

$\boxed{(\mathcal{CT} \rhd_C s, e) : \mathsf{OK}}$

$$\frac{\mathcal{CT} : \mathsf{OK} \quad \mathcal{CT}; \Sigma_s; \cdot \vdash_C e : t}{(\mathcal{CT} \rhd_C s, e) : \mathsf{OK}}$$

Figure 6.3: Typing rules of $\mathcal{J}_1$

$\boxed{\mathcal{CT} \rhd s, e \to s_1, e_1}$

$$\frac{\mathcal{CT}.C.m = mod\, t\, m(\overline{t_x\, x})\{e\}}{\mathcal{CT} \rhd s, l.m(\overline{v}) \to s, e[\overline{v/x}, l/\texttt{this}]} \qquad \frac{s(l) = \texttt{obj}\, C\, \{\overline{f = v}\}}{\mathcal{CT} \rhd s, l.f_i \to s, v_i}$$

$$\frac{}{\mathcal{CT} \rhd s, \texttt{null}.m(\overline{v}) \to s, \texttt{nerr}} \qquad \frac{}{\mathcal{CT} \rhd s, \texttt{null}.f_i \to s, \texttt{nerr}}$$

$$\frac{s(l) = \texttt{obj}\, C\, \{\overline{f = v}\}}{\mathcal{CT} \rhd s, l.f_i := u \to s[l \mapsto \texttt{obj}\, C\, \{\overline{f = v} \backslash f_i = u\}], \texttt{unit}}$$

$$\frac{}{\mathcal{CT} \rhd s, \texttt{null}.f_i := v \to s, \texttt{nerr}}$$

$$\frac{\begin{array}{c}\mathcal{CT}.C.\mathsf{constr} = C()\{\overline{\texttt{this}.f := c}\} \\ l \notin dom(s)\end{array}}{\mathcal{CT} \rhd s, \texttt{new}\, C \to s[l \mapsto \texttt{obj}\, C\, \{\overline{f = c}\}], l}$$

$$\frac{}{\mathcal{CT} \rhd s, \texttt{let}\, x = v\, \texttt{in}\, e \to s, e[v/x]}$$

$$\frac{(c, i) \in \{(\texttt{true}, 1), (\texttt{false}, 2)\}}{\mathcal{CT} \rhd s, \texttt{if}\ c\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \to s, e_i}$$

Evaluation Contexts

$$\begin{aligned} K &::= \kappa \mid K[\kappa] \\ \kappa &::= [\cdot] \mid [\cdot].f \mid [\cdot].m(\overline{e}) \mid v.m(\overline{v}, [\cdot], \overline{e}) \mid [\cdot].f := e \mid v.f := [\cdot] \\ &\quad \mid \texttt{let}\, x = [\cdot]\, \texttt{in}\, e \mid \texttt{if}\ [\cdot]\ \texttt{then}\ e\ \texttt{else}\ e \mid \texttt{op}(\overline{v}, [\cdot], \overline{e}) \end{aligned}$$

$\boxed{\mathcal{CT} \rhd s, K[e] \to s_1, K_1[e_1]}$

$$\mathcal{CT} \rhd s, e \to s_1, e_1 \mathcal{CT} \rhd s, K[e] \to s_1, K[e_1] \qquad \mathcal{CT} \rhd s, \kappa[\varepsilon] \to s, \varepsilon$$

Figure 6.4: Small-step Operational Semantics of $\mathcal{J}_1$

6.2 Class Equivalence

We want to study equivalence of class definitions in $\mathcal{J}_1$ and its extensions. To do this we need to define class-table contexts, relations on classes, and their extension to class tables.

Definition 6.2.1 (Class-Table Contexts)**.** *A* class-table context $\mathcal{CT}[\ ]$ *is a set of class definitions. Placing a class definition (or a set of class definitions) in the hole of a class-table context corresponds to set union:*

$$\mathcal{CT}[\overline{\mathcal{C}}] \stackrel{\text{def}}{=} \mathcal{CT} \cup \{\overline{\mathcal{C}}\}$$

Definition 6.2.2 (Class Relations)**.** G *is a* relation on classes *if and only if it is a set of pairs of class definitions such that for all* $(C, C') \in G$*, and all class-table contexts* $\mathcal{CT}[\ ]$*:*

$$\mathcal{CT}[\mathcal{C}] : \mathsf{OK} \iff \mathcal{CT}[\mathcal{C}'] : \mathsf{OK}$$

The above definition requires that we relate only class definitions that are interchangeable at compile time; in other words, replacing one with the other in a class-table context doesn't affect the typing judgment of the program. In practice this means that the related classes have the same name and the same public interface.

Definition 6.2.3 (G-Related Class Tables)**.** *If* G *is a relation on classes, then the following is its extension to class tables:*

$$G^{\mathsf{ct}} \stackrel{\text{def}}{=} \{(\mathcal{CT}[\overline{\mathcal{C}}], \mathcal{CT}[\overline{\mathcal{C}'}]) \mid \overline{(\mathcal{C}, \mathcal{C}')} \in G, \\ \mathcal{CT}[\overline{\mathcal{C}}] : \mathsf{OK}\}$$

We give the following definition of contextual equivalence for classes.

Definition 6.2.4 (Class Equivalence ($\equiv$))**.** $(\equiv)$ *is the largest relation on classes such that for all class tables* $\vdash \mathcal{CT}\ (\equiv)^{\mathsf{ct}}\ \mathcal{CT}'$*, expressions* e*, and types* t*, with* $\mathcal{CT}; \emptyset; \cdot \vdash_{\texttt{Object}} e : t$*, we have:*

$$\mathcal{CT} \rhd [], e\downarrow \iff \mathcal{CT}' \rhd [], e\downarrow$$

6.3 Derivation of Adequacy

6.3.1 Pre-Adequacy

Classes are static entities. Instead of reasoning directly about relations on classes we will reason about relations on references to objects, which are dynamic entities. We consider typed relations between references to objects, annotated with a pair of stores and a pair of class tables.

$$(\mathcal{CT}, \mathcal{CT}', s, s', R) \in (\textsc{Class Table}) \times (\textsc{Class Table}) \times \textsc{Store} \times \textsc{Store} \times \mathscr{P}(\textsc{Location} \times \textsc{Location} \times \textsc{Type})$$

Relating two object references under a type t encodes the intention that the references induce the same behavior when used as references of type t.

We write $(\mathcal{CT}, \Sigma), (\mathcal{CT}', \Sigma'); \Gamma \vdash e\ R\ e' : t$ when:

$$(e, e') \in R \qquad \mathcal{CT}; \Sigma; \Gamma \vdash_{\texttt{Object}} e : t \qquad \mathcal{CT}; \Sigma; \Gamma \vdash_{\texttt{Object}} e' : t$$

and $(\mathcal{CT}, \Sigma), (\mathcal{CT}', \Sigma'); \Gamma \vdash \overline{e}\ R\ \overline{e'} : \overline{t}$ when for some k:

$$(\mathcal{CT}, \Sigma), (\mathcal{CT}', \Sigma'); \Gamma \vdash e_1\ R\ e'_1 : t_1 \quad \ldots \quad (\mathcal{CT}, \Sigma), (\mathcal{CT}', \Sigma'); \Gamma \vdash e_k\ R\ e'_k : t_k$$

An annotated relation $(\mathcal{CT}, \mathcal{CT}', s, s', R)$ is well-formed when:

$$\forall l, l', t.\ (l, l', t) \in R \implies (\mathcal{CT}; \Sigma_s; \cdot \vdash_{\texttt{Object}} l : t) \wedge (\mathcal{CT}'; \Sigma_{s'}; \cdot \vdash_{\texttt{Object}} l' : t)$$

For the rest of this chapter we consider only well-formed annotated relations. Furthermore, we consider only annotated relations containing G-related class tables, for some class relation G, according to Definition 6.2.3. Therefore, both class tables have the same class hierarchy. [1]

We reason about sets of such relations:

$$\mathcal{X} \subseteq (\textsc{Class Table}) \times (\textsc{Class Table}) \times \textsc{Store} \times \textsc{Store} \times \mathscr{P}(\textsc{Location} \times \textsc{Location} \times \textsc{Type})$$

[1] In $\mathcal{J}_1$ this means that the class tables annotating a relation contain the same sets of class names, but in $\mathcal{J}_2$ and $\mathcal{J}_3$ it means that they have the same tree structure.

The inverse of an annotated relation is defined as follows.

Definition 6.3.1. *If $\mathcal{X}$ is a set of annotated relations, the inverse of $\mathcal{X}$, written $\mathcal{X}^{\mathsf{T}}$, is defined as:*

$$(\mathcal{CT}', \mathcal{CT}, s', s, R^{\mathsf{T}}) \in \mathcal{X}^{\mathsf{T}} \quad \textit{iff} \quad (\mathcal{CT}, \mathcal{CT}', s, s', R) \in \mathcal{X}$$

For the definition of pre-adequacy and adequacy we also need to define R-related values, answers, and expressions.

Definition 6.3.2. *If $(\mathcal{CT}, \mathcal{CT}', s, s', R)$ is an annotated relation, then we define the following typed relations:*

- *Related values:*

$$R^{\mathsf{val}} \stackrel{\text{def}}{=} R \cup \{(c, c, t) \mid \mathcal{CT}; \Sigma_s; \cdot \vdash_{\texttt{Object}} c : t, \\ \mathcal{CT}'; \Sigma_s; \cdot \vdash_{\texttt{Object}} c : t\}$$

- *Related answers:*

$$R^{\mathsf{ans}} \stackrel{\text{def}}{=} R^{\mathsf{val}} \cup \{(\texttt{nerr}, \texttt{nerr}, t) \mid t \in \textsc{Types}\}$$

- *Related expressions:*

$$R^{\mathsf{exp}} \stackrel{\text{def}}{=} \{(e[\overline{l/x}], e[\overline{l'/x}], t_e) \mid \mathcal{CT}; \emptyset; \overline{x : t} \vdash_{\texttt{Object}} e : t_e, \\ (\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash \bar{l}\ R\ \bar{l'} : \bar{t}, \}$$

The above definition describes when terms are considered to be related. Values are related when they are R-related locations at some type, or identical constants. Final answers are related when they are R-related values or identical errors. Two expressions are related under some type when they are constructed by the same expression context with related values in its holes (which boils down to just related locations since identical constants are subsumed by the identical contexts e).

We consider only related expressions that are typable in the `Object` class. We do this to avoid the situation where related expressions access private

fields and methods of classes with different implementations. Furthermore, the definition implies that $locs(e) = \emptyset$, forcing all R-related expressions to use only R-related references.

We now give the definition of pre-adequate annotated relations. The related program configurations derived from adequate annotated relations have the same operational behavior.

Definition 6.3.3 (Pre-Adequate Annotated Relations)**.** *An annotated relation, $(\mathcal{CT}, \mathcal{CT}', s, s', R)$, is pre-adequate if and only if for all expressions $(\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash e\ R^{\mathsf{exp}}\ e' : t$,*

$$\mathcal{CT} \rhd s, e\downarrow \iff \mathcal{CT}' \rhd s', e'\downarrow$$

Definition 6.3.4 (Pre-Adequacy $(\cong)$)**.** $(\cong)$ *is the set of all pre-adequate annotated relations.*

Pre-adequacy is sound and complete with respect to class equivalence in the following way.

Theorem 6.3.5 (Soundness and Completeness of $(\cong)$)**.**

$$(\mathcal{CT}, \mathcal{CT}', [], [], \emptyset) \in (\cong) \iff \vdash \mathcal{CT}\ (\equiv)^{\mathsf{ct}}\ \mathcal{CT}'$$

Proof. Forward direction: Let $(\mathcal{CT}, \mathcal{CT}', [], [], \emptyset) \in (\cong)$. For all expressions e and types t such that $\mathcal{CT}; \emptyset; \cdot \vdash_{\texttt{Object}} e : t$, we have:

$$(\mathcal{CT}, \emptyset), (\mathcal{CT}', \emptyset); \cdot \vdash e\ (\emptyset)^{\mathsf{exp}}\ e : t$$

Thus, by Definition 6.3.3:

$$\mathcal{CT} \rhd [], e\downarrow \iff \mathcal{CT}' \rhd [], e\downarrow$$

and, by the definition of $(\equiv)$, we have $\vdash \mathcal{CT}\ (\equiv)^{\mathsf{ct}}\ \mathcal{CT}'$.

Reverse direction: Let $\vdash \mathcal{CT}\ (\equiv)^{\mathsf{ct}}\ \mathcal{CT}'$. We will show that $(\mathcal{CT}, \mathcal{CT}', [], [], \emptyset)$ is pre-adequate. By Definition 6.3.3, it suffices to show that for all expressions $(\mathcal{CT}, \emptyset), (\mathcal{CT}', \emptyset); \cdot \vdash e\ (\emptyset)^{\mathsf{exp}}\ e' : t$

$$\mathcal{CT} \rhd [], e\downarrow \iff \mathcal{CT}' \rhd [], e'\downarrow$$

By the definition of $(\emptyset)^{\mathsf{exp}}$ we have that $e = e'$. Thus we need to show that

$$\mathcal{CT} \rhd [], e\downarrow \iff \mathcal{CT}' \rhd [], e\downarrow$$

This is immediate by Definition 6.2.4 and because $\vdash \mathcal{CT}\ (\equiv)^{\mathsf{ct}}\ \mathcal{CT}'$. □

6.3.2 Adequacy

From the definition of pre-adequate annotated relations we derive the definition of adequate sets of annotated relations by relating the final stores and answers of computations. The latter is a definition that can facilitate an inductive proof of equivalence.

Definition 6.3.6 (Adequate Sets of Annotated Relations)**.** *A set of annotated relations, $\mathcal{X}$, is adequate if and only if for all $(\mathcal{CT}, \mathcal{CT}', s, s', R) \in \mathcal{X}$:*

$$\begin{aligned}
\forall e, e', t, s_1, w.\ & (\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash e\ R^{\mathsf{exp}}\ e' : t \\
& \wedge\ \mathcal{CT} \rhd s, e \rightarrow^* s_1, w \\
& \quad \Longrightarrow\ \exists s_1', w', R_1.\ \mathcal{CT}' \rhd s', e' \rightarrow^* s_1', w' \\
& \qquad \wedge\ (\mathcal{CT}, \Sigma_{s_1}), (\mathcal{CT}', \Sigma_{s_1'}); \cdot \vdash w\ {R_1}^{\mathsf{ans}}\ w' : t \\
& \qquad \wedge\ (\mathcal{CT}, \mathcal{CT}', s_1, s_1', R_1) \in \mathcal{X} \\
& \qquad \wedge\ R \subseteq R_1
\end{aligned}$$

and the same holds for $\mathcal{X}^{\mathsf{T}}$.

It is easy to see that the union of adequate sets of annotated relations is adequate. Thus, the union of all adequate sets is the largest adequate set.

Definition 6.3.7 (Adequacy $(\approx)$)**.** $(\approx)$ *is the largest adequate set of annotated relations.*

Adequate sets of annotated relations are sound and complete with respect to pre-adequacy, and class equivalence.

Theorem 6.3.8 (Soundness of Adequate Sets)**.** *If $\mathcal{X}$ is adequate then it is also pre-adequate.*

Proof. Trivial by the definitions of $(\approx)$ and $(\cong)$. □

Theorem 6.3.9 (Completeness of $(\approx)$)**.** $(\cong)$ *is adequate.*

Proof. We will show that for all $(\mathcal{CT}, \mathcal{CT}', s, s', R) \in (\cong)$:

$$\begin{aligned}\forall e, e', t, s_1, w.\ & (\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash e\ R^{\mathsf{exp}}\ e' : t \\ & \wedge\ \mathcal{CT} \rhd s, e \to^* s_1, w \\ & \quad \Longrightarrow\ \exists s_1', w', R_1.\ \mathcal{CT}' \rhd s', e' \to^* s_1', w' \\ & \qquad\qquad \wedge\ (\mathcal{CT}, \Sigma_{s_1}), (\mathcal{CT}', \Sigma_{s_1'}); \cdot \vdash w\ {R_1}^{\mathsf{ans}}\ w' : t \\ & \qquad\qquad \wedge\ (\mathcal{CT}, \mathcal{CT}', s_1, s_1', R_1) \in (\cong) \\ & \qquad\qquad \wedge\ R \subseteq R_1\end{aligned}$$

and the same holds for $(\cong)^{\mathsf{T}}$.

Let $(\mathcal{CT}, \mathcal{CT}', s, s', R) \in (\cong)$ and $(\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash e\ R^{\mathsf{exp}}\ e' : t$. By the definition of $(\cong)$ the two configurations $\mathcal{CT} \rhd s, e$ and $\mathcal{CT}' \rhd s', e'$ co-terminate. Thus, it suffices to show that if

$$\mathcal{CT} \rhd s, e \to^* s_1, w \qquad \mathcal{CT}' \rhd s', e' \to^* s_1', w'$$

then

$$\begin{aligned}\exists R_1.\ & (\mathcal{CT}, \Sigma_{s_1}), (\mathcal{CT}', \Sigma_{s_1'}); \cdot \vdash w\ {R_1}^{\mathsf{ans}}\ w' : t \\ & \wedge\ (\mathcal{CT}, \mathcal{CT}', s_1, s_1', R_1) \in (\cong) \\ & \wedge\ R \subseteq R_1\end{aligned}$$

We will show that

$$(\mathcal{CT}, \mathcal{CT}', s_1, s_1', R \cup \{(w, w')\}) \in (\cong)$$

For all $\overline{x, t_x, l, l'}$, y, e_1, and t_1 such that $(\mathcal{CT}, \Sigma_{s_1}), (\mathcal{CT}', \Sigma_{s_1'}); \cdot \vdash \overline{l}\ R\ \overline{l'} : \overline{t_x}$, and $\mathcal{CT}; \emptyset; \overline{x : t_x} \vdash_{\texttt{Object}} e_1 : t_1$

$$\begin{aligned}& \mathcal{CT} \rhd s_1, e_1[\overline{l/x}, w/y] \downarrow \\ \iff\ & \mathcal{CT} \rhd s, \texttt{let}\ y = e\ \texttt{in}\ e_1[\overline{l/x}] \downarrow && \text{(by properties of reduction)} \\ \iff\ & \mathcal{CT}' \rhd s', \texttt{let}\ y = e'\ \texttt{in}\ e_1[\overline{l'/x}] \downarrow && ((\mathcal{CT}, \mathcal{CT}', s, s', R) \in (\cong)) \\ \iff\ & \mathcal{CT}' \rhd s_1', e_1[\overline{l'/x}, w'/y] \downarrow && \text{(by properties of reduction)}\end{aligned}$$

Therefore, by Definitions 6.3.3 and 6.3.4:

$$(\mathcal{CT}, \mathcal{CT}', s_1, s_1', R \cup \{(w, w')\}) \in (\cong)$$

Similarly we prove the condition for $(\cong)^{\mathsf{T}}$. □

Theorem 6.3.10. $(\cong) = (\approx)$

Proof. By Theorem 6.3.8 we have $(\approx) \subseteq (\cong)$ and by Theorem 6.3.9 we have $(\cong) \subseteq (\approx)$. Thus $(\cong) = (\approx)$. □

From the above theorem we show that adequacy is sound and complete with respect to class equivalence.

Theorem 6.3.11.

$$(\mathcal{CT}, \mathcal{CT}', [], [], \emptyset) \in (\approx) \iff (\mathcal{CT}, \mathcal{CT}') \in (\equiv)^{\mathsf{ct}}$$

Proof. By Theorems 6.3.10 and 6.3.5. □

6.4 Inductive Proofs of Equivalence

To prove equivalent two classes, or sets of classes, related by a class relation G we use the following proof method:

1. Find a set of annotated relations $\mathcal{X}$ such that for all class tables $\vdash \mathcal{CT}\ G^{\mathsf{ct}}\ \mathcal{CT}'$ we have $(\mathcal{CT}, \mathcal{CT}', [], [], \emptyset) \in \mathcal{X}$,

2. show that $\mathcal{X}$ is adequate,

3. invoke Theorem 6.3.11 and the definition of $(\equiv)$ to show that $G \subseteq (\equiv)$.

An in previous chapters, $\mathcal{X}$ can be shown adequate by two inductions; one for proving the condition involving $\mathcal{X}$ and one for proving the condition involving $\mathcal{X}^{\mathsf{T}}$. The induction hypothesis in both cases is the following.

Definition 6.4.1.

$$
\begin{aligned}
IH_{\mathcal{X}}(k) \stackrel{\text{def}}{=} \forall \mathcal{CT}, \mathcal{CT}', s, s', e, e', t, s_1, w.& \\
& (\mathcal{CT}, \mathcal{CT}', s, s', R) \in \mathcal{X} \\
\wedge\ & (\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash e\ R^{\mathsf{exp}}\ e' : t \\
\wedge\ & \mathcal{CT} \rhd s, e \rightarrow^{<k} s_1, w \\
& \implies \exists s_1', w', R_1.\ \mathcal{CT}' \rhd s', e' \rightarrow^{*} s_1', w' \\
& \qquad \wedge\ (\mathcal{CT}, \Sigma_{s_1}), (\mathcal{CT}', \Sigma_{s_1'}); \cdot \vdash w\ R_1{}^{\mathsf{ans}}\ w' : t \\
& \qquad \wedge\ (\mathcal{CT}, \mathcal{CT}', s_1, s_1', R_1) \in \mathcal{X} \\
& \qquad \wedge\ R \subseteq R_1
\end{aligned}
$$

The measure of the induction is the maximum number of steps, k, that the left-hand side takes until it reaches an answer (value or error).

Hence, proving $\mathcal{X}$ adequate amounts to proving that for all k, $IH_{\mathcal{X}}(k)$ and $IH_{\mathcal{X}^{\mathsf{T}}}(k)$ hold. For $k = 0$ it is trivial; for $k > 0$ it can be shown by induction:

$$
\boxed{
\begin{array}{l}
\forall k.\ IH_{\mathcal{X}}(k-1) \implies IH_{\mathcal{X}}(k) \\
\forall k.\ IH_{\mathcal{X}^{\mathsf{T}}}(k-1) \implies IH_{\mathcal{X}^{\mathsf{T}}}(k)
\end{array}
}
$$

6.5 Deriving Smaller Proof Obligations for Adequate Sets

We use a *proof construction scheme*—a hypothetical proof of the inductions of the previous section for arbitrary $\mathcal{X}$—to find smaller, necessary and sufficient, conditions for $\mathcal{X}$ to be adequate. This process follows the same structure as the proof construction schemes in previous chapters. These conditions form the following theorem for adequate sets of annotated relations.

Theorem 6.5.1. *A set of annotated relations, $\mathcal{X}$, is adequate if and only if for all k and $(\mathcal{CT}, \mathcal{CT}', s, s', R) \in \mathcal{X}$, assuming $IH_{\mathcal{X}}(k-1)$ holds, the following conditions are satisfied:*

1. (Enough instances.) *For all class names $C \in \mathcal{CT}.\mathit{cnames}$ with $\mathcal{CT}.C.\mathit{fields} = \overline{mod\ t\ f = c}$, and all locations $l \notin dom(s)$, it must be that $\mathcal{CT}'.C.\mathit{fields} = \overline{mod'\ t'\ f' = c'}$, and there exist $l' \notin dom(s')$ and $R_1 \supseteq R$ such that:*

$$(\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash l\ R_1\ l' : C$$
$$(\mathcal{CT}, \mathcal{CT}', s[l \mapsto \mathtt{obj}\, C\, \{\overline{f = c}\}], s'[l' \mapsto \mathtt{obj}\, C\, \{\overline{f' = c'}\}], R_1) \in \mathcal{X}$$

2. (Related `public` fields.) *For all locations l, l', and fields f with*

$$(\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash l\ R\ l' : C$$
$$s(l) = \mathtt{obj}\, D\, \{f = v, \overline{g = u}\} \qquad s'(l') = \mathtt{obj}\, D\, \{f = v', \overline{g' = u'}\}$$
$$\mathtt{public}\, t\ f \in \mathcal{CT}.C.\mathit{fields}$$

the following must be true:

   a) $(\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash v\ R^{\mathsf{val}}\ v' : t$

   b) *for all $(\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash v_1\ R^{\mathsf{val}}\ v_1' : t$, there exists $R_1 \supseteq R$ such that:*

$$(\mathcal{CT}, \mathcal{CT}', s[l \mapsto \mathtt{obj}\, C\, \{f = v_1, \overline{g = u}\}],$$
$$\qquad s'[l' \mapsto \mathtt{obj}\, C\, \{f = v_1', \overline{g' = u'}\}], R_1) \in \mathcal{X}$$

3. (Related `public` methods.) *For all locations l, l', class names C, and methods m with:*

$$(\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash l\ R\ l' : C \qquad \mathcal{CT}.C \neq \mathcal{CT}'.C$$
$$\mathcal{CT}.C.m = \mathtt{public}\, t_m\ m(\overline{t_x\ x})\{e\} \qquad \mathcal{CT}'.C.m = \mathtt{public}\, t_m\ m(\overline{t_x\ x})\{e'\}$$

and all values $(\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash \overline{v}\ R^{\mathsf{val}}\ \overline{v'} : \overline{t_x}$, if:

$$\mathcal{CT} \rhd s, l.m(\overline{v}) \rightarrow^{<k} s_1, w$$

then there exist s_1', w', and $R_1 \supseteq R$ such that:

$$\mathcal{CT}' \rhd s', l'.m(\overline{v'}) \rightarrow^{*} s_1', w' \qquad (\mathcal{CT}, \Sigma_{s_1}), (\mathcal{CT}', \Sigma_{s_1'}); \cdot \vdash w\ R_1\ w' : t_m$$
$$(\mathcal{CT}, \mathcal{CT}', s_1, s_1', R_1) \in \mathcal{X}$$

Furthermore, the same conditions hold for $\mathcal{X}^{\mathsf{T}}$.

Proof. Forward direction: by showing that the conditions of the theorem are special cases of the proof obligation of the definition of adequate relations (Definition 6.3.6).

Reverse direction: by recapitulating the aforementioned proof construction scheme. □

The first condition of Theorem 6.5.1 requires that adequate sets be closed under instantiation of new objects. The second condition requires that the set be closed under `public` method lookup and update. The last condition of the theorem requires that the sets be closed under invocation of `public` methods on related arguments—this condition is applicable only to non-identical class definitions ($\mathcal{CT}.C \neq \mathcal{CT}'.C$), since invocation of methods of identical classes is handled by the induction hypothesis. There are no conditions on `private` fields or methods because they are not accessible from the context.

6.6 $\mathcal{J}_2$: An extension of $\mathcal{J}_1$ With Inheritance

We now extend $\mathcal{J}_1$ by adding class inheritance. We also add the `protected` access modifier for fields and methods. Protected fields and methods are accessible from the same class and its subclasses. We do not yet add a cast operation to the language, allowing, thus, only implicit (and type-safe) upcasting of objects. We assume there is no shadowing of fields, something that can be accomplished automatically by adding the name of the class as part of the name of each field. The differences in the syntax, typing, and operational semantics of $\mathcal{J}_1$ and $\mathcal{J}_2$ are shown in Figure 6.5.

To encode inheritance we have added the rule of subsumption and the subtyping judgments imposed by the class hierarchy and the reflexive and transitive property. The rest of the typing rules of the form $\mathcal{CT}; \Sigma; \Gamma \vdash_C e : t$

CLASS DEFN.: $\mathcal{C}, \mathcal{D} ::= \texttt{class}\ C\ \texttt{extends}\ D\{\overline{mod\ t\ f}; \mathcal{K}\ \overline{\mathcal{M}}\}$
MODIFIERS: $mod ::= \cdots \mid \texttt{protected}$

$\boxed{\mathcal{CT} \vdash t_1 <: t_2}$

$$\frac{}{\mathcal{CT} \vdash t <: t} \qquad \frac{\mathcal{CT} \vdash t_1 <: t_2 \qquad \mathcal{CT} \vdash t_2 <: t_3}{\mathcal{CT} \vdash t_1 <: t_3} \qquad \frac{\mathcal{CT}.C.\mathsf{super} = D}{\mathcal{CT} \vdash C <: D}$$

$\boxed{\mathcal{CT}; \Sigma; \Gamma \vdash_C e : t}$

$$\frac{\mathcal{CT}; \Sigma; \Gamma \vdash_C e : t_1 \quad \mathcal{CT} \vdash t_1 <: t}{\mathcal{CT}; \Sigma; \Gamma \vdash_C e : t}$$

$\boxed{\mathcal{CT} \vdash_C \mathcal{M} : \mathsf{OK}}$

$$\frac{\mathcal{CT}; \emptyset; \overline{x : t_1} \vdash_C e : t \quad \mathcal{CT}.C.\mathsf{super} = D \quad \mathit{overridable}(\mathcal{CT}, mod\ \overline{t_1} \to t\ m, D)}{\mathcal{CT} \vdash_C mod\ t\ m(\overline{t_1\ x})\{e\} : \mathsf{OK}}$$

$\boxed{\mathcal{CT} \rhd s, e \to s_1, e_1}$

$$\frac{l \notin dom(s)}{\mathcal{CT} \rhd s, \texttt{new Object} \to s[l \mapsto \texttt{obj Object}\{\}], l}$$

$$\frac{\texttt{class}\ C\ \texttt{extends}\ D\{\cdots C()\{\overline{\texttt{this}.f := c}\}\cdots\} \in \mathcal{CT}}{\mathcal{CT} \rhd s, \texttt{new}\ C \to s, (\texttt{new}\ D; \overline{\texttt{this}.f := c})_C}$$

$$\frac{}{\mathcal{CT} \rhd s[l \mapsto \texttt{obj}\ D\{\overline{f_D = c_D}\}], (l; \overline{\texttt{this}.f := c})_C \to s[l \mapsto \texttt{obj}\ C\{\overline{f_D = c_D}, \overline{f = c}\}], l}$$

Evaluation Contexts

$$\kappa ::= \cdots \mid ([\cdot]; \overline{\texttt{this}.f := c})_C$$

Figure 6.5: Syntax, typing, and operational semantics of $\mathcal{J}_2$ (differences from $\mathcal{J}_1$)

have only trivial changes, assuming that the implementation of the meta-function *accessible* handles `protected` fields and methods in the right way. Furthermore we have changed the rule for type-checking method definitions to check that subclasses override only `public` and `protected` methods. A

valid class hierarchy is now a tree with an empty class `Object` as its root. In the typing judgments of $\mathcal{J}_2$, the meta-function *wfClassHierarchy* is true exactly when these conditions hold for a set of classes.

The operational semantics of $\mathcal{J}_2$ are mostly the same as that of $\mathcal{J}_1$, with the exception of the steps that involve object instantiation. In $\mathcal{J}_2$, when a new object is created, the fields from all the super-classes must be initialized.

6.6.1 Adequacy in $\mathcal{J}_2$

In this section we study adequacy for $\mathcal{J}_2$. The definition of class equivalence (Definition 6.2.4) still holds for $\mathcal{J}_2$. It conceals, though, the fact that the context seems to have more distinguishing power: it can extend related classes and use these extensions to distinguish related classes. We apply our method for deriving conditions for adequacy to study this extra distinguishing ability of the context. This reveals the effect of inheritance on the equivalence of classes.

The outline of our technique is the same as before. We reason about the same sets of tuples, $\mathcal{X}$, and we use the same definitions for $(\ )^{\mathsf{val}}$, $(\ )^{\mathsf{ans}}$, and $(\ )^{\mathsf{exp}}$ (Definition 6.3.2). The latter sets, though, are larger than before. Two related object references may be related under the class they instantiate, or any of its super-classes. This is where inheritance appears in our setup.

The definitions of pre-adequate annotated relations (Definition 6.3.3) and adequate sets of annotated relations (Definition 6.3.6) remain unchanged for $\mathcal{J}_2$, and, as a consequence, the induction hypothesis of the corresponding proof construction scheme is the same. When we re-unwind this proof, though, we discover additional proof obligations for adequate sets. Theorem 6.5.1 for $\mathcal{J}_2$ becomes:

Theorem 6.6.1. *A set of annotated relations, $\mathcal{X}$, is adequate if and only if for all k and $(\mathcal{CT}, \mathcal{CT}', s, s', R) \in \mathcal{X}$, assuming $IH_{\mathcal{X}}(k-1)$ holds, both the following conditions and the conditions of Theorem 6.5.1 are satisfied:*

1. (Related upcasts.) *For all locations l, l' and types t, t_1 with*

$$(\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash l \; R \; l' : t \qquad \mathcal{CT} \vdash t <: t_1$$

there exists $R_1 \supseteq R$ such that:

$$(\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash l \; R_1 \; l' : t_1 \qquad (\mathcal{CT}, \mathcal{CT}', s, s', R_1) \in \mathcal{X}$$

2. (Related `protected` fields.) *For all locations l, l', and fields f with*

$$(\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash l \; R \; l' : C$$
$$s(l) = \texttt{obj}\, D\, \{f = v, \overline{g = u}\} \qquad s'(l') = \texttt{obj}\, D\, \{f = v', \overline{g' = u'}\}$$
$$\texttt{protected} t \; f \in \mathcal{CT}.C.\mathit{fields}$$

the following must be true:

   a) $(\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash v \; R^{\mathsf{val}} \; v' : t$

   b) *for all* $(\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash v_1 \; R^{\mathsf{val}} \; v_1' : t$, *there exists* $R_1 \supseteq R$ *such that:*

$$(\mathcal{CT}, \mathcal{CT}', s[l \mapsto \texttt{obj}\, C\, \{f = v_1, \overline{g = u}\}], \\ s'[l' \mapsto \texttt{obj}\, C\, \{f = v_1', \overline{g' = u'}\}], R_1) \in \mathcal{X}$$

3. (Related `protected` methods.) *For all locations l, l', class names C, and methods m with:*

$$(\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash l \; R \; l' : C \qquad \mathcal{CT}.C \neq \mathcal{CT}'.C$$
$$\mathcal{CT}.C.m = \texttt{protected} t_m \; m(\overline{t_x \; x})\{e\}$$
$$\mathcal{CT}'.C.m = \texttt{protected} t_m \; m(\overline{t_x \; x})\{e'\}$$

and all values $(\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash \overline{v} \; R^{\mathsf{val}} \; \overline{v'} : \overline{t_x}$, if:

$$\mathcal{CT} \rhd s, l.m(\overline{v}) \rightarrow^{<k} s_1, w$$

then there exist s_1', w', and $R_1 \supseteq R$ such that:

$$\mathcal{CT}' \rhd s', l'.m(\overline{v'}) \rightarrow^{*} s_1', w' \qquad (\mathcal{CT}, \Sigma_{s_1}), (\mathcal{CT}', \Sigma_{s_1'}); \cdot \vdash w \; R_1 \; w' : t_m$$
$$(\mathcal{CT}, \mathcal{CT}', s_1, s_1', R_1) \in \mathcal{X}$$

The first of the above conditions is the most significant one. Two object references need to be related not only under a particular class type, but also under all of its super-classes. This has the following consequences when using an adequate set to show two classes $\mathcal{C}$ and $\mathcal{C}'$ equivalent:

1. Objects of $\mathcal{C}$ and $\mathcal{C}'$ must be shown equivalent under the interface of their super-classes. This is usually easy to show.

2. Objects of any classes that are defined by the context and extend $\mathcal{C}$ and $\mathcal{C}'$ need to be shown equivalent at the type of $\mathcal{C}$ and $\mathcal{C}'$. This implies that when we test the conditions for method invocation, there are related objects at the type of $\mathcal{C}$ and $\mathcal{C}'$ that may have overridden methods.

The other two conditions treat `protected` fields and methods as `public`, since they can be exposed to the context through the methods of subclasses.

Note that the condition for `public` method invocation of Theorem 6.5.1 needs not be changed for $\mathcal{J}_2$, where method override is permitted. We still need to test just the methods of non-identical classes. As mentioned earlier, when checking this condition (and the analogous condition for `protected` methods of Theorem 6.6.1) we need to take into account that some of the surrounding (`public` and `protected`) methods may be overridden, but the overrides will be identical textually and—because of the induction hypothesis—behaviorally. Thus, inheritance does not break local reasoning when using our proof technique.

Adding inheritance to $\mathcal{J}_1$ is not a conservative extension. Inheritance, in combination with method override, enhances the distinguishing power of

the context. For example consider the following classes:

```
C = class C {
      C() {}
      public bool m1()
        {true}
      public bool m2()
        {this.m1()} }
```

```
C' = class C {
       C() {}
       public bool m1()
         {true}
       public bool m2()
         {true} }
```

These are equivalent in $\mathcal{J}_1$, but not in $\mathcal{J}_2$. A context containing the following subclass of `C` can distinguish the two implementations:

```
class D extends C {
  D() {}
  public bool m1() {false}}
```

Thus, extending the language with inheritance gives more distinguishing power to the context. By overriding methods the context can break invariants that hold for `public` and `protected` methods and are needed by other methods. The only way for a class to maintain these invariants is to hide such methods from the context by declaring them `private`.

6.7 $\mathcal{J}_3$: Adding Downcasting to $\mathcal{J}_2$

Our last extension adds a cast expression to $\mathcal{J}_2$, giving us the language $\mathcal{J}_3$. The cast operator allows both explicit upcasting and downcasting. The latter is a non-type-safe operation which introduces one more error to the language, `cerr`. The extra syntax and semantics for the cast operator are shown in Figure 6.6.

6.7.1 Adequacy in $\mathcal{J}_3$

The addition of a casting operation has minimal effect on the technical machinery we have set up so far. The first thing we need to do is to add the new error to the set of answers of type t.

$$\begin{array}{rrll} \textsc{Expressions}: & e ::= \cdots \mid (C)e & \text{Cast Expression} \\ \textsc{Errors}: & \varepsilon ::= \cdots \mid \texttt{cerr} & \text{Cast Error} \end{array}$$

$\boxed{\mathcal{CT}; \Sigma; \Gamma \vdash_C e : t}$

$$\frac{\mathcal{CT}; \Sigma; \Gamma \vdash_E e : D \quad \mathcal{CT} \vdash D <: C}{\mathcal{CT}; \Sigma; \Gamma \vdash_E (C)e : C} \qquad \frac{\mathcal{CT}; \Sigma; \Gamma \vdash_E e : D \quad \mathcal{CT} \vdash C <: D \quad C \neq D}{\mathcal{CT}; \Sigma; \Gamma \vdash_E (C)e : C}$$

$$\frac{\mathcal{CT}; \Sigma; \Gamma \vdash_E e : D \quad \mathcal{CT} \vdash C \not<: D \quad \mathcal{CT} \vdash D \not<: C \quad \text{stupid-cast warning}}{\mathcal{CT}; \Sigma; \Gamma \vdash_E (C)e : C}$$

$\boxed{\mathcal{CT} \rhd s, e \to s_1, e_1}$

$$\frac{s(l) = \texttt{obj}\, C\, \{\overline{f = l}\} \quad \mathcal{CT} \vdash C <: D}{\mathcal{CT} \rhd s, (D)l \to s, l} \qquad \frac{s(l) = \texttt{obj}\, C\, \{\overline{f = l}\} \quad \mathcal{CT} \vdash C \not<: D}{\mathcal{CT} \rhd s, (D)l \to s, \texttt{cerr}}$$

$$\frac{}{\mathcal{CT} \rhd s, (D)\texttt{null} \to s, \texttt{null}}$$

Evaluation Contexts

$$\kappa ::= \cdots \mid (C)[\cdot]$$

Figure 6.6: Syntax, Typing, and Operational Semantics of $\mathcal{J}_3$ (differences with $\mathcal{J}_2$)

Definition 6.7.1. *If $(\mathcal{CT}, \mathcal{CT}', s, s', R)$ is an annotated relation, then we define the following typed relation on answers:*

$$R^{\mathsf{ans}} \stackrel{\text{def}}{=} R^{\mathsf{val}} \cup \{(\varepsilon, \varepsilon, t) \mid t \in \textsc{Type}, \varepsilon \in \{\texttt{nerr}, \texttt{cerr}\}\}$$

The rest of the definitions remain the same as before.

The second and last thing we need to do is to consider the new cast expression and the corresponding typing rules in the proof construction scheme for $\mathcal{J}_3$. Explicit upcasting produces—as expected—the same con-

dition as condition 1 of Theorem 6.6.1. Explicit downcasting produces one extra condition and "stupid" casting does not produce any new conditions. Thus the adequacy theorem for $\mathcal{J}_3$ becomes:

Theorem 6.7.2. *A set of annotated relations, $\mathcal{X}$, is adequate if and only if for all k and $(\mathcal{CT}, \mathcal{CT}', s, s', R) \in \mathcal{X}$, assuming $IH_{\mathcal{X}}(k-1)$ holds, both the following condition and the conditions of Theorems 6.5.1 and 6.6.1 are satisfied:*

1. (Related downcasts.) *For all locations l, l' and types t, t_1 with*

$$(\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash l \; R \; l' : t \qquad \mathcal{CT} \vdash t_1 <: t$$

there exists $R_1 \supseteq R$ such that:

$$(\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash l \; R_1 \; l' : t_1 \qquad (\mathcal{CT}, \mathcal{CT}', s, s', R_1) \in \mathcal{X}$$

Equivalences that hold in $\mathcal{J}_2$ break in $\mathcal{J}_3$ because of downcasting. This is due to the extra condition for $\mathcal{J}_3$ above: it is not sufficient that two objects instantiating class C be equivalent at type C, or even at the types of all super-classes of C. They must also be equivalent at the types of all subclasses of C, since they may be downcasted to these classes by the context. Hence, objects instantiating different classes may be equivalent in $\mathcal{J}_2$ at the type of a common super-class, but in $\mathcal{J}_3$ they can be distinguished by the context using downcasting. The example in Section 6.8.3 demonstrates this situation.

6.8 Examples

6.8.1 The Cell Example in $\mathcal{J}_1$

Here we give two implementations of a `Cell` class that store objects of some class `A` that is provided by the context. The first implementation of `Cell` is the usual one, while the other uses two private fields to keep the stored object, and a counter to decide which one to return when the `get` method is

invoked.

```
C ≝ class Cell {
      private A c;
      Cell() {this.c := null}
      public void set(A o) {this.c := o}
      public A get() {this.c}
    }

C' ≝ class Cell {
      private A c1, c2;
      private int cnt;
      Cell() {
        this.c1 := null; this.c2 := null; this.cnt := 0 }
      public void set(A o) {
        this.c1 := o; this.c2 := o }
      public A get() {
        this.cnt := this.cnt + 1;
        if even(this.cnt) then this.c1 else this.c2 }
    }
```

Following the proof method described in Section 6.4, to prove the above two class implementations equivalent we construct the class relation $G = \{(\mathcal{C}, \mathcal{C}')\}$ and the set $\mathcal{X}$ of annotated relations, shown in Figure 6.7. It suffices to show that $\mathcal{X}$ is adequate.

$\mathcal{X}$ validates the conditions of Theorem 6.5.1. The first condition of the theorem is satisfied by the $\mathcal{X}$-EXTEND rule. The second condition of the theorem is satisfied by the $\mathcal{X}$-UPDATE rule. It remains to show that the final condition of Theorem 6.5.1 is satisfied.

To prove this condition we consider the non-identical related classes, which in this case are the `Cell` classes.

$\mathcal{X}$-BASE
$$\frac{\vdash \mathcal{CT}\ G^{\mathsf{ct}}\ \mathcal{CT}'}{(\mathcal{CT}, \mathcal{CT}', [], [], \emptyset) \in \mathcal{X}}$$

$\mathcal{X}$-EXTEND
$$\frac{\begin{array}{c}(\mathcal{CT}, \mathcal{CT}', s, s', R) \in \mathcal{X} \qquad C \in \mathcal{CT}.\mathsf{cnames} \\ \mathcal{CT}.C.\mathsf{fields} = \overline{mod\, t\ f = c} \qquad \mathcal{CT}'.C.\mathsf{fields} = \overline{mod'\, t'\ f' = c'} \\ l \notin dom(s) \qquad l' \notin dom(s')\end{array}}{(\mathcal{CT}, \mathcal{CT}', s[l \mapsto \mathtt{obj}\, C\, \{\overline{f = c}\}], s'[l' \mapsto \mathtt{obj}\, C\, \{\overline{f' = c'}\}], R \cup \{(l, l', C)\}) \in \mathcal{X}}$$

$\mathcal{X}$-UPDATE
$$\frac{\begin{array}{c}(\mathcal{CT}, \mathcal{CT}', s, s', R) \in \mathcal{X} \qquad (\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash l\ R\ l' : C \\ s(l) = \mathtt{obj}\, C\, \{\overline{f = u}\} \qquad s'(l') = \mathtt{obj}\, C\, \{\overline{f' = u'}\} \\ \mathtt{public}\, t\ f_i \in \mathcal{CT}.C.\mathsf{fields} \qquad (\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash v\ R^{\mathsf{val}}\ v' : t\end{array}}{(\mathcal{CT}, \mathcal{CT}', s[l \mapsto \mathtt{obj}\, C\, \{\overline{f = u} \backslash f_i = v\}], s'[l' \mapsto \mathtt{obj}\, C\, \{\overline{f' = u'} \backslash f_i = v'\}], R) \in \mathcal{X}}$$

$\mathcal{X}$-CELLS
$$\frac{\begin{array}{c}(\mathcal{CT}, \mathcal{CT}', s, s', R) \in \mathcal{X} \qquad (\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash l\ R\ l' : \mathtt{Cell} \\ s(l) = \mathtt{obj}\, \mathtt{Cell}\, \{\mathtt{c} = u\} \qquad s'(l') = \mathtt{obj}\, \mathtt{Cell}\, \{\mathtt{c1} = u', \mathtt{c2} = u', \mathtt{cnt} = n\} \\ (\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash v\ R^{\mathsf{val}}\ v' : \mathtt{A}\end{array}}{\begin{array}{l}(\mathcal{CT}, \mathcal{CT}',\ s[l \mapsto \mathtt{obj}\, \mathtt{Cell}\, \{\mathtt{c} = v\}], \\ \qquad s'[l' \mapsto \mathtt{obj}\, \mathtt{Cell}\, \{\mathtt{c1} = v', \mathtt{c2} = v', \mathtt{cnt} = n + 1\}], R) \in \mathcal{X}\end{array}}$$

Figure 6.7: Construction of adequate set of annotated relations for proving the equivalence of two cell implementations in $\mathcal{J}_1$.

For some k let $IH_{\mathcal{X}}(k-1)$ holds and:

$$(\mathcal{CT}, \mathcal{CT}', s, s', R) \in \mathcal{X}$$

$$(\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash l\ R\ l' : \mathtt{Cell}$$

$$s(l) = \mathtt{obj}\, \mathtt{Cell}\, \{\mathtt{c} = u\}$$

$$s'(l') = \mathtt{obj}\, \mathtt{Cell}\, \{\mathtt{c1} = u', \mathtt{c2} = u', \mathtt{cnt} = n\}$$

We consider the case of the invocation of `set`. We have:

$$(\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash v\ R^{\mathsf{val}}\ v' : \mathtt{A}$$

If $v = v' = \mathtt{null}$ then both sides raise the error `nerr`. Otherwise we have:

$$\mathcal{CT} \rhd s, l.\mathtt{set}(v) \rightarrow^{<k} s[l \mapsto \mathtt{obj}\, \mathtt{Cell}\, \{\mathtt{c} = v\}], \mathtt{unit}$$

By the definition of $\mathcal{C}$ and the $\mathcal{X}$-CELLS rule we get:

$$\mathcal{CT}' \rhd s', l'.\mathtt{set}(v') \rightarrow^* s'[l' \mapsto \mathtt{obj}\,\mathtt{Cell}\,\{\mathtt{c1} = v', \mathtt{c2} = v', \mathtt{cnt} = n+1\}], \mathtt{unit}$$

$$(\mathcal{CT}, \mathcal{CT}',\ s[l \mapsto \mathtt{obj}\,\mathtt{Cell}\,\{\mathtt{c} = v\}],$$
$$s'[l' \mapsto \mathtt{obj}\,\mathtt{Cell}\,\{\mathtt{c1} = v', \mathtt{c2} = v', \mathtt{cnt} = n+1\}], R) \in \mathcal{X}$$

Thus Condition 3 of Theorem 6.5.1 is satisfied for the case of `set`.

To show that it is also satisfied for the case of the invocation of `set` we consider:

$$\mathcal{CT} \rhd s, l.\mathtt{get}() \rightarrow^{<k} s, u$$

We also have:

$$\mathcal{CT}' \rhd s', l'.\mathtt{get}() \rightarrow^* s', u'$$

It suffices to show that $(\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash u\ R^{\mathsf{val}}\ u' : \mathtt{A}$, which is trivial by an induction on the construction of $\mathcal{X}$.

Similarly we show that the same conditions are satisfied by $\mathcal{X}^{\mathsf{T}}$. Thus, $\mathcal{X}$ is adequate and by Theorem 6.3.11 and the definition of $(\equiv)$ we have that $G \subseteq (\equiv)$.

6.8.2 The Cell Example in $\mathcal{J}_2$

We adapt the two implementations of a Cell from the previous section to the language $\mathcal{J}_2$, and we show that the equivalence still holds. The code of the two implementations remains the same, except that the two `Cell` classes extend `Object`. In $\mathcal{J}_2$ the context can store any subclass of `A` in a cell, and, more importantly, it can arbitrarily extend `Cell`.

To prove this equivalence in $\mathcal{J}_2$ we construct again the relation $G = \{(\mathcal{C}, \mathcal{C}')\}$ and the set of annotated relations, $\mathcal{X}$, shown in Figure 6.8, and we show that $\mathcal{X}$ is adequate.

The construction of $\mathcal{X}$ here is largely the same as in the previous section. The differences are shown in bold face in Figure 6.8. The $\mathcal{X}$-EXTEND rule

$\mathcal{X}$-BASE
$$\frac{\vdash \mathcal{CT}\ G^{\mathsf{ct}}\ \mathcal{CT}'}{(\mathcal{CT}, \mathcal{CT}', [], [], \emptyset) \in \mathcal{X}}$$

$\mathcal{X}$-EXTEND
$$\frac{\begin{array}{c}(\mathcal{CT}, \mathcal{CT}', s, s', R) \in \mathcal{X} \qquad C \in \mathcal{CT}.\mathsf{cnames} \qquad \boldsymbol{\mathcal{CT} \vdash C <: D} \\ \mathcal{CT}.C.\mathsf{fields} = \overline{mod\, t\ f = c} \qquad \mathcal{CT}'.C.\mathsf{fields} = \overline{mod'\, t'\ f' = c'} \\ l \notin dom(s) \qquad l' \notin dom(s')\end{array}}{(\mathcal{CT}, \mathcal{CT}', s[l \mapsto \mathtt{obj}\, C\, \{\overline{f = c}\}], s'[l' \mapsto \mathtt{obj}\, C\, \{\overline{f' = c'}\}], R \cup \{(l, l', \boldsymbol{D})\}) \in \mathcal{X}}$$

$\mathcal{X}$-UPDATE
$$\frac{\begin{array}{c}(\mathcal{CT}, \mathcal{CT}', s, s', R) \in \mathcal{X} \qquad (\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash l\ R\ l' : \boldsymbol{D} \\ s(l) = \mathtt{obj}\, C\, \{\overline{f = u}\} \qquad s'(l') = \mathtt{obj}\, C\, \{\overline{f' = u'}\} \\ \boldsymbol{mod}\, t\ f_i \in \mathcal{CT}.\boldsymbol{D}.\mathsf{fields} \qquad (\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash v\ R^{\mathsf{val}}\ v' : t \\ \boldsymbol{mod} \in \{\mathtt{public}, \mathtt{protected}\}\end{array}}{(\mathcal{CT}, \mathcal{CT}', s[l \mapsto \mathtt{obj}\, C\, \{\overline{f = u} \backslash f_i = v\}], s'[l' \mapsto \mathtt{obj}\, C\, \{\overline{f' = u'} \backslash f_i = v'\}], R) \in \mathcal{X}}$$

$\mathcal{X}$-CELLS
$$\frac{\begin{array}{c}(\mathcal{CT}, \mathcal{CT}', s, s', R) \in \mathcal{X} \qquad (\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash l\ R\ l' : \mathtt{Cell} \\ s(l) = \mathtt{obj}\, \boldsymbol{C}\, \{\overline{\boldsymbol{f_0 = u_0}}, \mathtt{c} = u\} \\ s'(l') = \mathtt{obj}\, \boldsymbol{C}\, \{\overline{\boldsymbol{f_0 = u'_0}}, \mathtt{c1} = u', \mathtt{c2} = u', \mathtt{cnt} = n\} \\ (\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash v\ R^{\mathsf{val}}\ v' : \mathtt{A}\end{array}}{\begin{array}{l}(\mathcal{CT}, \mathcal{CT}', s[l \mapsto \mathtt{obj}\, \boldsymbol{C}\, \{\overline{\boldsymbol{f_0 = u_0}}, \mathtt{c} = v\}], \\ \qquad s'[l' \mapsto \mathtt{obj}\, \boldsymbol{C}\, \{\overline{\boldsymbol{f_0 = u'_0}}, \mathtt{c1} = v', \mathtt{c2} = v', \mathtt{cnt} = n + 1\}], R) \in \mathcal{X}\end{array}}$$

Figure 6.8: Construction of adequate set of annotated relations for proving the equivalence of two cell implementations in $\mathcal{J}_2$.

now relates references to freshly instantiated objects under the type of any super-class of these objects. The $\mathcal{X}$-UPDATE rule updates only the public fields visible at the type that references are related. Lastly, the $\mathcal{X}$-CELLS rule takes into account that at type `Cell` there may be related objects of some subtype of `Cell`.

The proof that $\mathcal{X}$ is adequate is similar to the one of the previous section. The most notable difference is that when we test the conditions for method invocations, we need to take into account that these methods may have been overridden. Such methods, though, would be identical and, by the induction

hypothesis, will have the related behavior.

6.8.3 A Read-Write Cell Example in $\mathcal{J}_2$

This example, due to Andrew Kennedy, demonstrates that with the addition of inheritance, there are more equivalences that hold between objects. Here, objects of different classes are equivalent at the type of a common superclass.

Consider the classes:

$\mathcal{RO} \stackrel{\text{def}}{=}$
```
class RO extends Object {
    protected int val;
    RO() {this.val := 0}
    public int get() {this.val} }
```

$\mathcal{RW} \stackrel{\text{def}}{=}$
```
class RW extends RO {
    RW() {}
    public void set(int i) {this.val := i} }
```

$\mathcal{CB} \stackrel{\text{def}}{=}$
```
class Callback extends Object {
    Callback() {}
    public void act(RO arg){this.act(arg)} }
```

Under these classes, the following two classes are equivalent in $\mathcal{J}_2$:

$\mathcal{C} \stackrel{\text{def}}{=}$
```
class Proxy extends Object {
    Proxy() {}
    public int foo(Callback cb) {
      let rw := new RW()
      in rw.set(17); cb.act(rw); rw.get() } }
```

$\mathcal{C}' \stackrel{\text{def}}{=}$
```
class Proxy extends Object {
    Proxy() {}
    public int foo(Callback cb) {
      let rw = new RW()
      in rw.set(17); cb.act(rw); 17 } }
```

The two implementations of the class `Proxy` instantiate an `RW` object, store the number `17` in it, and pass it to the context by invoking the method `act` on the argument `cb`. This call upcasts the `RW` object to an `RO` object, and thus (in the absence of downcasting) the context cannot access the `set` method defined in the `RW` class. Hence, when the call to `act` returns, both $\mathcal{C}$, that returns the current value stored in the `RW` object, and $\mathcal{C}'$, that returns the number `17`, will return the same value. That is to say that, due to the upcasting, the context can not change the value in the `RW` object.

The invocation of method `act` will return only if `cb` is bound to an instance of a subclass of `Callback`—defined in the context—in which `act` is overriden. Our proof method deals with this because, as required by Condition 1 of Theorem 6.6.1, in adequate sets of annotated relations instances of all classes (including subclasses of `Callback`) need to be related at the type of any of their super-classes (including `Callback`).

This equivalence does not hold in $\mathcal{J}_3$, where the context can downcast the two `RW` objects and access their `set` method.

To prove this equivalence we construct the class relation

$$G \stackrel{\text{def}}{=} \{(\mathcal{RW}, \mathcal{RW}), (\mathcal{RO}, \mathcal{RO}), (\mathcal{CB}, \mathcal{CB}), (\mathcal{C}, \mathcal{C})\}$$

and the set of annotated relations shown in Figure 6.9.

According to the method of Section 6.4, to prove the equivalence between $\mathcal{C}$ and $\mathcal{C}'$ it suffices to show that $\mathcal{X}$ is adequate. The rules $\mathcal{X}$-EXTEND and $\mathcal{X}$-UPDATE satisfy the first two conditions of Theorem 6.5.1, as well as the first two conditions of Theorem 6.6.1. The last condition of Theorem 6.6.1 is trivially satisfied.

We need to prove the condition of Theorem 6.6.1 involving the invocation of `public` methods on objects related at the type of non-identical classes. Hence, we only need to test the condition for the invocation of the method `foo`.

$$\mathcal{X}\text{-BASE}\quad \frac{\vdash \mathcal{CT}\ G^{\mathsf{ct}}\ \mathcal{CT}'}{(\mathcal{CT}, \mathcal{CT}', [], [], \emptyset) \in \mathcal{X}}$$

$$\mathcal{X}\text{-EXTEND}\quad \frac{\begin{array}{c}(\mathcal{CT}, \mathcal{CT}', s, s', R) \in \mathcal{X} \qquad C \in \mathcal{CT}.\mathsf{cnames} \qquad \mathcal{CT} \vdash C <: D \\ \mathcal{CT}.C.\mathsf{fields} = \overline{mod\, t\ f = c} \qquad \mathcal{CT}'.C.\mathsf{fields} = \overline{mod'\, t'\ f' = c'} \\ l \notin dom(s) \qquad l' \notin dom(s')\end{array}}{(\mathcal{CT}, \mathcal{CT}', s[l \mapsto \mathtt{obj}\, C\, \{\overline{f = c}\}], s'[l' \mapsto \mathtt{obj}\, C\, \{\overline{f' = c'}\}], R \cup \{(l, l', D)\}) \in \mathcal{X}}$$

$$\mathcal{X}\text{-UPDATE}\quad \frac{\begin{array}{c}(\mathcal{CT}, \mathcal{CT}', s, s', R) \in \mathcal{X} \qquad (\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash l\ R\ l' : D \\ s(l) = \mathtt{obj}\, C\, \{\overline{f = u}\} \qquad s'(l') = \mathtt{obj}\, C\, \{\overline{f' = u'}\} \\ mod\, t\ f_i \in \mathcal{CT}.D.\mathsf{fields} \qquad (\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash v\ R^{\mathsf{val}}\ v' : t \\ mod \in \{\mathtt{public}, \mathtt{protected}\}\end{array}}{(\mathcal{CT}, \mathcal{CT}', s[l \mapsto \mathtt{obj}\, C\, \{\overline{f = u} \backslash f_i = v\}], s'[l' \mapsto \mathtt{obj}\, C\, \{\overline{f' = u'} \backslash f_i = v'\}], R) \in \mathcal{X}}$$

$$\mathcal{X}\text{-CELLS}\quad \frac{\begin{array}{c}(\mathcal{CT}, \mathcal{CT}', s, s', R) \in \mathcal{X} \\ l \notin dom(s) \qquad l' \notin dom(s') \qquad \mathcal{CT} \vdash \mathtt{RO} <: D\end{array}}{\begin{array}{l}(\mathcal{CT}, \mathcal{CT}',\ s[l \mapsto \mathtt{obj}\, \mathtt{RW}\, \{\mathtt{val} = 17\}], \\ \qquad s'[l' \mapsto \mathtt{obj}\, \mathtt{RW}\, \{\mathtt{val} = 17\}], R \cup \{(l, l', D)\}) \in \mathcal{X}\end{array}}$$

Figure 6.9: Construction of adequate set of annotated relations for proving the equivalence between two proxy classes in $\mathcal{J}_2$.

For some k let $IH_{\mathcal{X}}(k-1)$ holds and:

$$(\mathcal{CT}, \mathcal{CT}', s, s', R) \in \mathcal{X}$$

$$(\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash l\ R\ l' : \mathtt{Proxy}$$

$$(\mathcal{CT}, \Sigma_s), (\mathcal{CT}', \Sigma_{s'}); \cdot \vdash v\ R^{\mathsf{val}}\ v' : \mathtt{Callback}$$

$$\mathcal{CT} \rhd s, l.\mathtt{foo}(v) \rightarrow^{<k} s_1, w$$

If $v = v' = \mathtt{null}$ then both sides will raise the error `nerr`. Otherwise, by the operational semantics, we have that for some $l \notin dom(s)$ and s_1:

$$\mathcal{CT} \rhd s[l \mapsto \mathtt{obj}\, \mathtt{RW}\, \{\mathtt{val} = 17\}], v.\mathtt{act}(l) \rightarrow^{<k-1} s_1, \mathtt{unit}$$

Let $s_0 = s[l \mapsto \texttt{obj}\,\texttt{RW}\,\{\texttt{val} = 17\}]$, $s'_0 = s'[l' \mapsto \texttt{obj}\,\texttt{RW}\,\{\texttt{val} = 17\}]$, and $R_0 = R \cup \{(l, l', \texttt{RO})\}$. By the $\mathcal{X}$-CELLS rule we get:

$$(\mathcal{CT}, \mathcal{CT}', s_0, s'_0, R_0) \in \mathcal{X}$$

By $IH_{\mathcal{X}}(k-1)$ and because

$$(\mathcal{CT}, \Sigma_{s_0}), (\mathcal{CT}', \Sigma_{s_0}); \cdot \vdash v.\texttt{act}(l)\ {R_0}^{\mathsf{exp}}\ v'.\texttt{act}(l') : \texttt{Callback}$$

there exist $R_1 \supseteq R_0$ and s'_1 such that:

$$\mathcal{CT}' \rhd s'_0, v'.\texttt{act}(l') \rightarrow^* s'_1, \texttt{unit} \qquad (\mathcal{CT}, \mathcal{CT}', s_1, s'_1, R_1) \in \mathcal{X}$$

Furthermore, by an induction on the construction of $\mathcal{X}$ we get that $s_1(l) = \texttt{obj}\,\texttt{RW}\,\{\texttt{val} = 17\}$ and $s'_1(l') = \texttt{obj}\,\texttt{RW}\,\{\texttt{val} = 17\}$. Therefore $w = 17$ and $\mathcal{CT}' \rhd s', l'.\texttt{foo}(v') \rightarrow^* s'_1, 17$, which concludes the proof of this case.

Similarly we show that the same conditions are satisfied by $\mathcal{X}^{\mathsf{T}}$. Thus, $\mathcal{X}$ is adequate and by Theorem 6.3.11 and the definition of $(\equiv)$ we have that $G \subseteq (\equiv)$.

CHAPTER 7

Related Work and Future Directions

7.1 Related Work

Reasoning about contextual equivalence has a long history. As a result, a number of methods have been developed for proving terms contextually equivalent. Here we focus on methods for imperative and higher-order languages.

Bisimulations Bisimulation, due to Hennessy and Milner [20, 21], originated as a technique to characterize the behavior of non-deterministic systems. Abramsky [2] adapted this idea to deterministic languages, creating what is known as *applicative bisimulations*, and used it to reason about an untyped lazy lambda-calculus. Using domain-theoretic methods he proved that applicative bisimulation is a congruence for that language, which later was shown by a direct operational proof by Howe [24]. Gordon and Rees [19] applied applicative bisimilarity to one of the stateless, typed, object calculi of Abadi and Cardelli [1] and proved that it coincides with contextual equivalence—this was the first study of contextual equivalence for an object calculus.

Applicative bisimulation has also been used in continuation-passing style languages to prove contextual equivalence. For example, Tiuryn and Wand [60] presented a continuation-passing model of an untyped lambda-calculus

with input and output, and proved that applicative approximation coincides with contextual approximation. Wand and Sullivan [61] gave a denotational semantics to a recursively-typed higher-order language with side effects by translation to a CPS calculus, and used the technique to prove the correctness of assignment elimination, an important step in the compilation of Scheme.

Our technique does not require conversion to CPS and can prove all equivalences provable by applicative bisimulation. Furthermore, our use of the induction hypothesis in our theorem of adequacy greatly simplifies reasoning about many higher-order terms.

Sumii and Pierce [55, 56] simplified the use of bisimulations in sequential languages. They replaced a single bisimulation with a set of partial bisimulations, each corresponding to a "world", representing the conditions of knowledge, or the state, in which it holds. Similar ideas have previously been used in process calculi (e.g., [15, 40]) and suggested for imperative languages [32]. Their method, as presented, has limited applicability when dealing with some equivalences of higher-order procedures [35]. In [56], though, they discuss a variation of their method that better deals with higher-order procedures, and is similar to the ones derived by the technique described in this dissertation. They dismiss it, though, because it is hard to show closure under unions for the bisimulations. We overcome this problem by proving completeness of the derived methods without using closure under unions.

Our technique (also presented in parts in [29, 28, 30, 10]) continues the work of Sumii and Pierce. By using their Kripke-style relations we were able to reason about imperative languages where parts of the store are hidden from the context. We improved on their method by using an up-to context technique [45, 46, 51] to account for large parts of the relations, and in Chapter 5, where the benefit is significant, an up-to store technique to account for parts of the related stores. We also introduced an induction

hypothesis in our definitions that can be used to immediately reason about smaller related computations, including computations that "leak" from the context into the procedures. Such computations are the applications of higher-order arguments in a functional language (Chapters 2, 3, and 4), or invocations of callbacks in an object-oriented language (Chapters 5 and 6). Using our method we were able to prove equivalences of imperative, higher-order procedures [35] where previous operational techniques had limited applicability [11, 41, 56]. Furthermore, our reasoning method can be derived in a systematic way for a variety of imperative and higher-order sequential languages.

Recently, Sangiorgi, Kobayashi, and Sumii [49, 50] presented an alternative technique to our adequate sets of relations. It starts by defining a bare-bones bisimulation à la Sumii and Pierce on a small-step semantics and then builds on top a number of up-to techniques. The resulting bisimulation-up-to is theoretically equivalent to our adequacy and both have been used to prove the same equivalences in the languages for which they have been defined. The benefit of the technique of Sangiorgi et al. is that it uses an "up-to reduction" that effectively replaces our induction hypothesis in the definition of adequate sets of relations, and thus it can be used more straightforwardly in higher-order concurrent languages. On the other hand, as shown in this dissertation, our technique is more systematically adaptable to higher-order sequential languages.

Also recently, Støvring and Lassen [54] presented an *eager normal form bisimulation* for a language with control operators and general references. This is an extension of previous work from Lassen [33], where, roughly, two expressions are bisimilar if they have bisimilar normal forms. They make use of possible-worlds relations, similar to the ones we use here. Their bisimulation is sound and complete for the language with control operators and state, but incomplete when one of these two features is removed.

Up-to techniques for bisimulations Up-to techniques were originally introduced in concurrency [47, 45, 51]. Their purpose was to reduce the size of bisimulations by closing them automatically up to context, β-equivalence, injective substitution, bisimilarity etc., hence making their construction easier. Our inductive principle of contextual equivalence naturally encodes an up-to context technique. We start by considering expressions related in R^{cxt}, which is a soundness requirement for proving contextual equivalence, and require that the final values of their evaluation are related in R^{cxt}, instead of R. Moreover, in Section 5.6, we introduce a notion of an "up-to store" technique.

As mentioned above, Sangiorgi et al. [49, 50] gave a more layered account of up-to techniques for reasoning principles like the ones presented here in the tradition of the π-calculus.

Operational Logical Relations Another method for proving contextual equivalence is to use operational logical relations. This method relies on induction on types and the operational semantics of the language to construct a concrete relation of equivalent terms. Pitts and Stark [41] gave an operational logical relation for a simply-typed functional language with integer locations that is sound and complete with respect to contextual equivalence. The relation is parameterized over store relations, enabling reasoning implicitly about stores with desired local properties. Their method is based on a $\top\top$-closure of the relation that makes it difficult to use directly in a proof of equivalence. Instead, they give a proof method called "the principle of local invariants", which can be used to prove that two expressions are included in the logical relation. This method is shown to be sound but not complete. Our method can be used for all their examples, including ones for which their method does not apply, such as the example of Section 3.6.2.

Pitts and Stark [42] also gave an operational logical relation for a fragment of Standard ML with names, and Stark [53] gave a similar logical

relation for the ν-calculus. These logical relations were complete for first-order types but incomplete at higher types, and could not be used to prove the "hard" equivalence of Section 4.6.2.

Simple induction on types is not possible in the presence of recursive types. Appel and McAllester [8] address this issue by a step-indexed logical relation for a language with recursive types. Umut, Ahmed, and Blume [7] use step-indexed logical relations to show that self-adjusting computations are contextually equivalent to normal computations for a higher-order, imperative language. Step-indexed logical relations share with our technique that reduction steps appear in the definition; a more exact correlation of the two techniques, though, remains to be studied.

Label Transition Systems Trace semantics, formulated in terms of a Labeled Transition Systems (LTS) is another useful technique for reasoning about contextual equivalence. This technique shares common intuition with game semantics, since the focus of both is to encode the interaction of terms with the context. In an LTS two terms are indistinguishable by the context when they have the same set of traces. The difficulty of these methods is that, depending on the language, one has to choose the observable actions of the LTS carefully so that as many terms as possible have the same traces, while maintaining soundness. In contrast, our technique lets us derive sound and complete reasoning methods systematically. Carrying out a proof of equivalence using an LTS usually requires formulating an induction on the lengths of traces, similar to the induction of our proof construction scheme, or to define a suitable bisimulation, similar to our adequacy.

Jeffrey and Rathke have used this method for reasoning about a class-based [27], and a concurrent object language [26]. In [27] they study a Java-like language for which they define a semantic trace equivalence and show that it is sound and complete with respect to testing equivalence. Their language is quite different from the one we considered in Chapter 6, since it

allows only a restricted form of interaction between a class and its context. It is not clear how their technique can be applied to languages with more complex interactions, such as via class inheritance or higher-order methods.

Jagadeesan et al. [25] have developed a bisimulation proof technique in an LTS for an aspect calculus. Their bisimulation principle is based on Sangiorgi's open bisimulations [48] and replaces accounting for infinite number of contexts with explicit bookkeeping mechanisms of the knowledge exposed to the context. This method fails to prove some equivalences from [35] because it makes reading from a reference an observable effect.

Recently, Laird [31] gave a sound and complete method for general references using trace semantics of a LTS, and sketched a correspondence between the LTS and game semantics. We believe his method is capable of proving all examples of Chapter 3.

CIU Theorems Theorems of closed instantiations of uses (or CIU theorems) attempt to restrict the set of contexts that must be considered in a direct proof. In this approach, one defines a set of contexts (usually the evaluation contexts) such that if two expressions are equivalent in these contexts, then they are equivalent in every context. Mason and Talcott [34] provide such a theorem for a language similar to the CBV λ-calculus we considered in Chapter 3. Similar results were obtained by Felleisen [16].

Gordon, Hankin and Lassen [18] gave an operational equivalence for the same calculus that we consider in Chapter 5, and they showed that it coincides with contextual equivalence. This equivalence is a CIU theorem for the language. Their relation does not provide a technique for proving difficult equivalences. For example, proving the cell example of Section 5.7 with their CIU theorem would require an induction over all reduction contexts. Such an induction is not obvious because the stores and the environments of the two sides may change in a different way in some of the cases. Attempting this proof would be at least as difficult as the induction discussed

in Section 5.5. On the other hand, using our bisimulations we can prove the equivalence for all of their examples, as well as the cell example.

Denotational Methods In the denotational realm, the usual approach to prove two expressions contextually equivalent is to assign them the same denotation [62, 36]. There is a long line of work in the literature that attempts to find satisfactory stateful models for imperative languages, with or without higher-order features (see [59] for an overview). The advantage of such models is that they are compositional and thus do not require explicit reasoning about the context. Constructing good stateful models for imperative languages is hard, though, and highly dependent on the language. For example, the two cell implementations we proved equivalent in Sections 3.6.3, 5.7, and 6.8.1 have sufficiently different store behavior that usual stateful models would assign different denotations to each one. Moreover, stateful models of imperative languages are almost never fully abstract without the addition of artificial features in the language. Sieber [52], for instance, constructs a fully-abstract semantics for the second-order subset of an Algol-like language by adding a "snap-back" mechanism to the language that, after a procedure returns, restores the entire store to the state before the call to the procedure.

Stateless denotational models of imperative languages based on game theory by Abramsky et al. [4, 5, 6] capture the entire behavior of terms as the strategy of a game between the term and the context. If two procedures have the same strategy then their interaction with the context is indistinguishable. These models achieve full abstraction by making use of a "quotient" relation to identify equivalent strategies (e.g., strategies that read a location in the context any number of times). Furthermore, using games in the topos of FM-sets, Abramsky et al. [3] gave the first fully abstract model of the ν-calculus and validated the "hard" equivalence that we proved in Section 4.6.2.

Another denotational approach for proving contextual equivalence is by using a logical relation on the denotations of terms. Benton and Lepechey [11] develop a logical relation on FM-cpos to reason about a language with dynamic allocation of integer references. Their method, inspired by Pitts and Stark's [41], is parameterized over state relations, allowing a form of local reasoning. Recently, Bohr and Birkedal [14, 13] extended this method to reason about contextual equivalence for a language with higher-order store, like the one we considered in Chapter 3, and proved all equivalences of that chapter. Unlike ours, though, their technique is not complete.

Banerjee and Naumann in [9] present a method for reasoning about whole-program equivalence in a subset of Java similar to $\mathcal{J}_3$. Their technique is based on a denotational model in which they build a relation on denotations. They use a notion of *confinement* to restrict certain pointers to the heap, and show that if two class tables are confined and their denotations are related, then these class tables are equivalent. It is not clear if this technique is complete. This is because the authors do not show that there is a simulation relation for any two equivalent and confined class tables, but also because the technique seems helpful only for reasoning about confined class tables. Furthermore it is not obvious how this technique can be extended to contextual equivalence of classes, a stronger property than whole program equivalence.

7.2 Future Directions

We developed a technique for deriving sound and complete reasoning methods for proving contextual equivalence in a variety of sequential languages. The advantage of the reasoning methods is that they deal with imperative as well as higher-order features, and can successfully be used to prove many interesting equivalences.

There are several future directions worth exploring:

- The first is to study the possible benefits that may come from using an assertion logic to specify bisimulations. Such a logic should most probably have a notion of separating conjunction [44]. Using such a logic would simplify writing down concrete bisimulations for proving equivalences. It may also ease the burden of accounting for parts of the store that are irrelevant to a specific equivalence, similar to the up-to store technique of Section 5.6. Furthermore it is worth investigating if equivalence proofs can become more syntactic, by defining a set of appropriate deduction rules.

- Then, it would be of interest to study equivalences that are weaker than contextual equivalence and verify the applicability of the technique described in this dissertation to the derivation of reasoning principles for these equivalences.

- Another interesting direction worth exploring is the applicability of our technique to concurrent languages, especially higher-order concurrent calculi.

- Moreover, the mechanical verification and partial automation of proving equivalence using the method we described would be beneficial. Preliminary work appears in [10].

Bibliography

[1] M. Abadi and L. Cardelli. *A Theory of Objects*. Springer-Verlag, Berlin, Heidelberg, and New York, 1996.

[2] S. Abramsky. The lazy lambda calculus. In D. A. Turner, editor, *Research Topics in Functional Programming*, pages 65–116. Addison-Wesley, Boston, MA, USA, 1990.

[3] S. Abramsky, D. R. Ghica, A. S. Murawski, C.-H. L. Ong, and Ian D. B. Stark. Nominal games and full abstraction for the nu-calculus. In *Proc. 19th Annual IEEE Symposium on Logic in Computer Science (LICS 2004)*, pages 150–159, Washington, DC, USA, 2004. IEEE Computer Society.

[4] S. Abramsky, K. Honda, and G. McCusker. A fully abstract game semantics for general references. In *Proc. 13th Annual IEEE Symposium on Logic in Computer Science (LICS 1998)*, page 334, Washington, DC, USA, 1998. IEEE Computer Society.

[5] S. Abramsky and G. McCusker. Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In Peter W. O'Hearn and R. D. Tennent, editors, *Algol-like languages*, volume 2 of *Progress in Theoretical Computer Science*, chapter 20, pages 297–329. Birkhauser, Boston, MA, USA, 1997.

[6] S. Abramsky and G. McCusker. Full abstraction for Idealized Algol with passive expressions. *Theoretical Computer Science*, 227(1):3–42, 1999.

[7] U. A. Acar, A. Ahmed, and M. Blume. Imperative self-adjusting computation. In *Proc. 35th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL 2008)*, pages 309–322, New York, NY, USA, 2008. ACM.

[8] A. W. Appel and D. McAllester. An indexed model of recursive types for foundational proof-carrying code. *ACM Transactions on programming languages and systems (TOPLAS 2001)*, 23(5):657–683, 2001.

[9] A. Banerjee and D. A. Naumann. Ownership confinement ensures representation independence for object-oriented programs. *Journal of the ACM*, 52(6):894–960, 2005.

[10] N. Benton and V. Koutavas. A mechanized bisimulation for the nu-calculus. Technical Report MSR-TR-2008-129, Microsoft Research, September 2008.

[11] N. Benton and B. Leperchey. Relational reasoning in a nominal semantics for storage. In *Proc. 7th International Conference on Typed Lambda Calculi and Applications (TLCA 2005)*, volume 3461 of *Lecture Notes in Computer Science*, pages 86–101, Berlin, Heidelberg, and New York, 2005. Springer.

[12] G. Bierman, M. Parkinson, and A. Pitts. MJ: An imperative core calculus for Java and Java with effects. Technical Report 563, Cambridge University Computer Laboratory, April 2003.

[13] N. Bohr. *Advances in Reasoning Principles for Contextual Equivalence and Termination*. PhD thesis, The IT University of Copenhagen, Copenhagen, DK, October 2007.

[14] N. Bohr and L. Birkedal. Relational reasoning for recursive types and references. In N. Kobayashi, editor, *Proc. 4th Asian Symposium on Programming Languages and Systems (APLAS 2006)*, volume 4279 of *Lec-*

ture Notes in Computer Science, pages 79–96, Berlin, Heidelberg, and New York, 2006. Springer.

[15] Y. Deng and D. Sangiorgi. Towards an algebraic theory of typed mobile processes. In *Proc. 31st International Colloquium on Automata, Languages and Programming (ICALP 2004)*, volume 3142 of *Lecture Notes in Computer Science*, pages 445–456, Berlin, Heidelberg, and New York, 2004. Springer.

[16] M. Felleisen. *The Calculi of Lambda-v-cs Conversion: A Syntactic Theory of Control and State in Imperative Higher-Order Programming Languages*. PhD thesis, Indiana University, 1987.

[17] M. Flatt, S. Krishnamurthi, and M. Felleisen. A programmer's reduction semantics for classes and mixins. *Lecture Notes in Computer Science*, 1523:241–269, 1999.

[18] A. D. Gordon, P. D. Hankin, and S. B. Lassen. Compilation and equivalence of imperative objects. *Journal of Functional Programming*, 9(04):373–426, 2000.

[19] A. D. Gordon and G. D. Rees. Bisimilarity for a first-order calculus of objects with subtyping. In *Proc. 23rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL 1996)*, pages 386–395, New York, NY, USA, 1996. ACM.

[20] M. Hennessy and R. Milner. On observing nondeterminism and concurrency. In J. W. de Bakker and Jan van Leeuwen, editors, *Proc. 7th International Colloquium Automata, Languages and Programming (ICALP 1980)*, volume 85 of *Lecture Notes in Computer Science*, pages 299–309, Berlin, Heidelberg, and New York, 1980. Springer.

[21] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32:137–161, 1985.

[22] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 583, 1969.

[23] K. Honda, N. Yoshida, and M. Berger. An observationally complete program logic for imperative higher-order frame rules. In *Proc. 20th IEEE Symposium on Logic in Computer Science (LICS 2005)*, pages 270–279, Chicago, IL, June 2005. IEEE Computer Society.

[24] D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, February 1996.

[25] R. Jagadeesan, C. Pitcher, and J. Riely. Open bisimulation for aspects. In *Proc. 6th international conference on Aspect-oriented software development (AOSD 07)*, pages 107–120, New York, NY, USA, 2007. ACM.

[26] A. Jeffrey and J. Rathke. A fully abstract may testing semantics for concurrent objects. *Theoretical Computer Science*, 338(1–3):17–63, 2005.

[27] A. Jeffrey and J. Rathke. Java jr.: Fully abstract trace semantics for a core Java language. In *Proc. 14th European Symposium on Programming (ESOP 2005), Programming Languages and Systems*, volume 3444 of *Lecture Notes in Computer Science*, pages 423–438. Springer, 2005.

[28] V. Koutavas and M. Wand. Bisimulations for untyped imperative objects. In P. Sestoft, editor, *Proc. 15th European Symposium on Programming (ESOP 2006), Programming Languages and Systems*, volume 3924 of *Lecture Notes in Computer Science*, pages 146–161, Berlin, Heidelberg, and New York, 2006. Springer-Verlag.

[29] V. Koutavas and M. Wand. Small bisimulations for reasoning about higher-order imperative programs. In *Proc. 33rd ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages (POPL 2006)*, pages 141–152, New York, NY, USA, January 2006. ACM Press.

[30] V. Koutavas and M. Wand. Reasoning about class behavior. Appeared in FOOL/WOOD 2007 Workshop, January 2007.

[31] J. Laird. A fully abstract trace semantics for general references. In L. Arge, C. Cachin, T. Jurdzinski, and A. Tarlecki, editors, *Proc. 34st International Colloquium on Automata, Languages and Programming (ICALP 2007)*, volume 4596 of *Lecture Notes in Computer Science*, pages 667–679, Berlin, Heidelberg, and New York, 2007. Springer.

[32] S. B. Lassen. Bisimulation up to context for imperative lambda calculus. Part of a presentation "Bisimulation up to Context for Sequential Higher-Order Languages" at The Semantic Challenge of Object-Oriented Programming, Dagstuhl Seminar 98261. Schloss Dagstuhl, Wadern, Germany. June 28 - July 3, 1998.

[33] S. B. Lassen. Eager normal form bisimulation. In *Proc. 20th Annual IEEE Symposium on Logic in Computer Science (LICS 2005)*, pages 345–354, Washington, DC, USA, 2005. IEEE Computer Society.

[34] I. A. Mason and C. L. Talcott. Equivalence in functional languages with effects. *Journal of Functional Programming*, 1:287–327, 1991.

[35] A. R. Meyer and K. Sieber. Towards fully abstract semantics for local variables. In *Proc. 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 1988)*, pages 191–203, New York, NY, USA, 1988. ACM.

[36] R. Milne and C. Strachey. *A Theory of Programming Language Semantics*. Chapman and Hall, London, UK, 1976. Also Wiley, New York.

[37] J. C. Mitchell. *Foundations of Programming Languages*. MIT Press, Cambridge, MA, USA, 1996.

[38] J. H. Morris, Jr. *Lambda Calculus Models of Programming Languages*. PhD thesis, MIT, Cambridge, MA, USA, 1968.

[39] A. Nanevski, G. Morrisett, and L. Birkedal. Polymorphism and separation in hoare type theory. In J. H. Reppy and J. L. Lawall, editors, *Proc. 11th ACM SIGPLAN International Conference on Functional Programming (ICFP 2006)*, pages 62–73, Portland, OR, USA, September 2006. ACM press.

[40] B. C. Pierce and D. Sangiorgi. Behavioral equivalence in the polymorphic pi-calculus. In *Proc. 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL 1997)*, pages 242–255, New York, NY, USA, 1997. ACM Press.

[41] A. Pitts and I. Stark. Operational reasoning for functions with local state. In A. Gordon and A. Pitts, editors, *Higher Order Operational Techniques in Semantics*, pages 227–273. Publications of the Newton Institute, Cambridge University Press, 1998.

[42] A. M. Pitts and I. D. B. Stark. Observable properties of higher order functions that create local names, or: What's new? In *Proc. 18th International Symposium on Mathematical Foundations of Computer Science (MFCS 1993)*, volume 711 of *Lecture Notes in Computer Science*, pages 122–141, Berlin, Heidelberg, and New York, 1993. Springer-Verlag.

[43] G. D. Plotkin. Lcf considered as a programming language. *Theoretical Computer Science*, 5(3):225–255, 1977.

[44] J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proc. 17th IEEE Symposium on Logic in Computer Science (LICS 2002)*, pages 55–74, Washington, DC, USA, 2002. IEEE Computer Society.

[45] D. Sangiorgi. Locality and non-interleaving semantics in calculi for mobile processes. *Theoretical Computer Science*, 155:39–83, 1996.

[46] D. Sangiorgi. On the bisimulation proof method. *Mathematical Structures in Computer Science*, 8:447–479, 1998.

[47] D. Sangiorgi. On the bisimulation proof method. *Mathematical Structures in Computer Science*, 8(5):447–479, 1998.

[48] D. Sangiorgi. The bisimulation proof method: Enhancements and open problems. In R. Gorrieri and H. Wehrheim, editors, *Proc. 8th IFIP International Conference Formal Methods for Open Object-Based Distributed Systems*, volume 4037 of *Lecture Notes in Computer Science*, pages 18–19, Berlin, Heidelberg, and New York, 2006. Springer.

[49] D. Sangiorgi, N. Kobayashi, and E. Sumii. Environmental bisimulations for higher-order languages. In *Proc. 22th Annual IEEE Symposium on Logic in Computer Science (LICS 2007)*, pages 293–302. IEEE Computer Society, 2007.

[50] D. Sangiorgi, N. Kobayashi, and E. Sumii. Logical bisimulations and functional languages. In *Proc. International Symposium on Fundamentals of Software Engineering (FSEN 2007)*, volume 4767 of *Lecture Notes in Computer Science*, pages 364–379, Berlin, Heidelberg, and New York, 2007. Springer.

[51] D. Sangiorgi and R. Milner. The problem of "Weak Bisimulation up to". In W.R. Cleveland, editor, *Proc. 3rd International Conference on Concurrency Theory (CONCUR 1992)*, volume 630 of *Lecture Notes in Computer Science*, pages 32–46, Berlin, Heidelberg, and New York, 1992. Springer.

[52] K. Sieber. Full abstraction for the second order subset of an Algol-like language. *Theoretical Computer Science*, 168(1):155–212, 1996.

[53] I. Stark. *Names and Higher-Order Functions*. PhD thesis, University of Cambridge, Cambridge, UK, December 1994. Also published as Technical Report 363, University of Cambridge Computer Laboratory.

[54] K. Støvring and S. B. Lassen. A complete, co-inductive syntactic theory of sequential control and state. In *Proc. 34th ACM SIGPLAN-SIGACT symposium on principles of programming languages (POPL 2007)*, pages 161–172, New York, NY, USA, 2007. ACM.

[55] E. Sumii and B. C. Pierce. A bisimulation for dynamic sealing. In *Proc. 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL 2004)*, pages 161–172, New York, NY, USA, 2004. ACM.

[56] E. Sumii and B. C. Pierce. A bisimulation for type abstraction and recursion. In *Proc. 32nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL 2005)*, pages 63–74, New York, NY, USA, 2005. ACM.

[57] E. Sumii and B. C. Pierce. A bisimulation for dynamic sealing. *Theoretical Computer Science*, 375(1–3):169–192, May 2007.

[58] E. Sumii and B. C. Pierce. A bisimulation for type abstraction and recursion. *Journal of the ACM*, 54(5):1–43, 2007.

[59] R. D. Tennent and D. R. Ghica. Abstract models of storage. *Higher Order and Symbolic Computation*, 13(1–2):119–129, 2000.

[60] J. Tiuryn and M. Wand. Untyped lambda-calculus with input-output. In H. Kirchner, editor, *Proc. 21st International Colloquium on Trees in Algebra and Programming (CAAP 1996)*, volume 1059 of *Lecture Notes in Computer Science*, pages 317–329, Berlin, Heidelberg, and New York, April 1996. Springer.

[61] M. Wand and G. T. Sullivan. Denotational semantics using an operationally-based term model. In *Proc. 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL 1997)*, pages 386–399, New York, NY, USA, 1997. ACM.

[62] G. Winskel. *The Formal Semantics of Programming Languages*. MIT Press, Cambridge, MA, 1993.