A Bisimulation for Type Abstraction and Recursion

EIJIRO SUMII
Tohoku University
and
BENJAMIN C. PIERCE
University of Pennsylvania

We present a bisimulation method for proving the contextual equivalence of packages in λ-calculus with full existential and recursive types. Unlike traditional logical relations (either semantic or syntactic), our development is "elementary," using only sets and relations and avoiding advanced machinery such as domain theory, admissibility, and $\top\top$-closure. Unlike other bisimulations, ours is complete even for existential types. The key idea is to consider *sets* of relations—instead of just relations—as bisimulations.

Categories and Subject Descriptors: D.3.3 [**Programming Languages**]: Language Constructs and Features—*Abstract data types*; F.3.3 [**Logics and Meanings of Programs**]: Studies of Program Constructs—*Type structure*

General Terms: Theory, Languages

Additional Key Words and Phrases: Lambda-Calculus, Contextual Equivalence, Bisimulations, Logical Relations, Existential Types, Recursive Types

1. INTRODUCTION

Proving the equivalence of computer programs is important not only for verifying the correctness of program transformations such as compiler optimizations, but also for showing the compatibility (substitutability) of program modules. Consider two modules M and M' implementing the same interface I; if these different implementations are equivalent under this common interface, then they are indeed compatible, correctly hiding their differences from outside view.

Contextual equivalence is a natural definition of program equivalence: two programs are called contextually equivalent if they exhibit the same observable behavior when put in any legitimate context of the language. However, direct proofs of contextual equivalence are typically infeasible, because its definition involves a universal quantification over an infinite number of contexts (and naive approaches such as structural induction on the syntax of contexts do not work). This has led to a search for alternative methods for proving contextual equivalence, whose fruits can be grouped into two categories: *logical relations* and *bisimulations.*

An extended abstract of this article has appeared in *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 63–74, 2005.

Logical relations. Logical relations were first developed for denotational semantics of typed λ-calculi (see, e.g., [Mitchell 1996, Chapter 8] for details) and can also be adapted [Pitts 2000; 2005] to their term models; this adaptation is sometimes called syntactic logical relations [Crary and Harper 2007]. Logical relations are relations on terms defined by induction on their types: for instance, two pairs are related when their elements are pairwise related; two tagged terms $\texttt{in}_i(M)$ and $\texttt{in}_j(N)$ of a sum type are related when the tags i and j are equal and the contents M and N are also related; and, crucially, two functions are related when they map related arguments to related results. The soundness of logical relations is proved via the Fundamental Property (or Basic Lemma), which states that any well-typed term is related to itself.

Logical relations are pleasantly straightforward, as long as we stick to the simply typed λ-calculus (or even the polymorphic λ-calculus) without recursion. However, their extension with recursion is challenging. Recursive functions cause a problem in the proof of the fundamental property that must be addressed by introducing additional "unwinding properties" [Pitts 2000; 2005; Birkedal and Harper 1999; Crary and Harper 2007]. Recursive *types* are even more difficult (in particular with negative occurrences): since logical relations are defined by induction on types, recursive types (with negative occurrences) require topological properties in the *definition* of logical relations [Birkedal and Harper 1999; Crary and Harper 2007]. These difficulties are not confined to meta-theorems, but are visible to the users of logical relations: in order to prove contextual equivalence using logical relations, one often has to prove the admissibility, compute the limit, or calculate the $\top\top$-closure of particular relations.[1]

Bisimulations. Bisimulations were originally developed for process calculi [Milner 1980; 1989; 1999] and state transition systems in general. Abramsky [1990] adapted bisimulations to untyped λ-calculus and called them *applicative bisimulations.* Briefly, two functions $\lambda x.\, M$ and $\lambda x.\, M'$ are bisimilar when $(\lambda x.\, M)N \Downarrow \iff (\lambda x.\, M')N \Downarrow$ for any N and the results are also bisimilar if these evaluations converge. Gordon [1995a; 1995b] and Gordon and Rees [1996; 1995] extended applicative bisimulations to calculi with objects, subtyping, universal polymorphism, and recursive types. Sangiorgi [1992] defined *context bisimulation*, which is a variant of applicative bisimulation for higher-order π-calculus.

Unlike logical relations, bisimulations have no difficulty with recursion (or even concurrency). However, existing bisimulation methods for typed λ-calculi are too weak in the presence of existential polymorphism; that is, they are not able to prove interesting equivalence properties of existential packages. For instance, consider the two packages

$$\begin{aligned} M &= \texttt{pack int}, \langle 1, \lambda x\!:\!\texttt{int}.\, x \overset{\texttt{int}}{=} 0\rangle \;\texttt{as}\; \tau \\ M' &= \texttt{pack bool}, \langle \texttt{true}, \lambda x\!:\!\texttt{bool}.\, \neg x\rangle \;\texttt{as}\; \tau \end{aligned}$$

where $\tau = \exists\alpha.\, \alpha \times (\alpha \rightarrow \texttt{bool})$. Existing bisimulation methods cannot prove the contextual equivalence $\vdash M \equiv M' : \tau$ of these simple packages, because they do not

[1]Recently, Ahmed [2006] proposed syntactic logical relations based on evaluation step indices. See Section 7 for its relationship to our bisimulations.

capture the fact that the only values of type α are 1 in the "left-hand world" and `true` in the right-hand world. The same observation applies to context bisimulation.

The only exceptions to the problem above are bisimulations for polymorphic π-calculi [Pierce and Sangiorgi 2000; Berger et al. 2003]. However, π-calculus is name-based and low-level. As a result, it is rather difficult to encode polymorphic λ-calculus into polymorphic π-calculus while preserving equivalence (though there are some results [Berger et al. 2003] for the case without recursion), so it is at least as difficult to use π-calculus for reasoning about abstraction in λ-calculus or similar languages with (in particular higher-order) functions and recursion. In addition to the problem of encoding, existing bisimulations for polymorphic π-calculi are incomplete [Pierce and Sangiorgi 2000] and complex [Berger et al. 2003].

Encoding existential polymorphism in terms of universal polymorphism does not help either. Consider the following encodings of M and M'

$$\begin{aligned} N &= \lambda f : \sigma.\, f[\mathtt{int}]\langle 1, \lambda x : \mathtt{int}.\, x \overset{\mathtt{int}}{=} 0\rangle \\ N' &= \lambda f : \sigma.\, f[\mathtt{bool}]\langle \mathtt{true}, \lambda x : \mathtt{bool}.\, \neg x\rangle \end{aligned}$$

where $\sigma = \forall\alpha.\, \alpha \times (\alpha \to \mathtt{bool}) \to \mathtt{ans}$ and $\mathtt{ans}$ is some answer type. In order to establish the bisimulation between N and N', one has at least to prove

$$\begin{aligned} & f[\mathtt{int}]\langle 1, \lambda x : \mathtt{int}.\, x \overset{\mathtt{int}}{=} 0\rangle \Downarrow \\ \Longleftrightarrow\;\; & f[\mathtt{bool}]\langle \mathtt{true}, \lambda x : \mathtt{bool}.\, \neg x\rangle \Downarrow \end{aligned}$$

for any observer function f of type σ, which is almost the same as the *definition* of $\vdash M \equiv M' : \tau$.

Our solution. We address these problems—and thereby obtain a sound and complete bisimulation for existential types (as well as universal and recursive types)—by adapting key ideas from our previous work [Sumii and Pierce 2004] on bisimulation for *sealing* [Morris 1973a; 1973b], a dynamic form of data abstraction. The crucial insight is that we should define bisimulations as *sets* of relations—rather than just relations—annotated with type information.

For instance, a bisimulation X showing the contextual equivalence of M and M' above can be defined (roughly) as

$$X = \{(\emptyset, \mathcal{R}_0), (\Delta, \mathcal{R}_1), (\Delta, \mathcal{R}_2), (\Delta, \mathcal{R}_3)\}$$

where

$$\begin{aligned} \mathcal{R}_0 &= \{(M, M', \tau)\} \\ \mathcal{R}_1 &= \mathcal{R}_0 \cup \{(\langle 1, \lambda x : \mathtt{int}.\, x \overset{\mathtt{int}}{=} 0\rangle, \\ &\qquad\quad \langle \mathtt{true}, \lambda x : \mathtt{bool}.\, \neg x\rangle, \\ &\qquad\quad \alpha \times (\alpha \to \mathtt{bool}))\} \\ \mathcal{R}_2 &= \mathcal{R}_1 \cup \{(1, \mathtt{true}, \alpha), \\ &\qquad\quad (\lambda x : \mathtt{int}.\, x \overset{\mathtt{int}}{=} 0, \\ &\qquad\quad \lambda x : \mathtt{bool}.\, \neg x, \\ &\qquad\quad \alpha \to \mathtt{bool})\} \\ \mathcal{R}_3 &= \mathcal{R}_2 \cup \{(\mathtt{false}, \mathtt{false}, \mathtt{bool})\} \end{aligned}$$

$$\Delta = \{(\alpha, \mathtt{int}, \mathtt{bool})\}.$$

Because we are ultimately interested in the equivalence of M and M', we begin by including $(\emptyset, \mathcal{R}_0)$ in X. (The role of the first element $\emptyset$ of this pair will be explained in a moment.) Next, since a context can open those packages and examine their contents, we add $(\Delta, \mathcal{R}_1)$ to X, where Δ is a *concretion environment* mapping the abstract type α to its respective concrete types in the left-hand side and the right-hand side. Then, since the contents of the packages are pairs, a context can examine their elements, so we add $(\Delta, \mathcal{R}_2)$ to X. Last, since the second elements of the pairs are functions of type $\alpha \to \mathtt{bool}$, a context can apply them to any arguments of type α; the only such arguments are, in fact, 1 in the left-hand world and `true` in the right-hand world, so we add $(\Delta, \mathcal{R}_3)$ to X. Since the results of these applications are equal as booleans, there is nothing else that a context can do to distinguish the values in R_3.

Conceptually, each $\mathcal{R}$ occurring in a pair $(\Delta, \mathcal{R}) \in X$ represents the *knowledge* of a context at some point in time, which increases via new observations by the context. In order to prove contextual equivalence, it suffices to find a bisimulation X that is closed under this increase of contexts' knowledge. (Thus, in fact, not only X but also the singleton set $\{(\Delta, \mathcal{R}_3)\}$ is a bisimulation in our definition.)

Why do we consider a bisimulation X to be a set of $\mathcal{R}$s (with corresponding Δs) instead of taking their union in the first place? Because the latter does not exist in general! In other words, the union of two "valid" $\mathcal{R}$s is not always a valid $\mathcal{R}$. For instance, consider the union of $\mathcal{R}_3$ and its inverse $\mathcal{R}_3^{-1} = \{(V', V, \tau) \mid (V, V', \tau) \in \mathcal{R}_3\}$. Although each of them makes perfect sense by itself, taking their union is nonsensical because it confuses two different worlds (which, in fact, is not even type-safe). This observation is absolutely fundamental in the presence of type abstraction (or other forms of information hiding such as sealing), and it forms the basis of many technicalities in the present work (as in our previous work [Sumii and Pierce 2004]). By considering a set of relations instead of taking their union, it becomes straightforward to define bisimilarity to be the largest bisimulation and thereby apply standard co-inductive arguments (to prove the completeness of bisimilarity, for instance). In addition, this also gives a natural account of the *generativity* of existential types, i.e., of the fact that opening the same package twice gives incompatible contents.

This decision does not incur any significant difficulty for users of our bisimulation: we devise a trick—explained below, in the definition of bisimulation for packages—that keeps the set of relations finite in many cases; even where this trick does not apply, it is not very difficult to define the infinite set of relations (e.g., by set comprehension or by induction) and check it against our definition of bisimulation (as we will do in Section 5.3 for generative functors, or as we did in previous work [Sumii and Pierce 2004, Examples 4.7 and 4.8] for security protocols).

Contributions. This is an "elementary" (i.e., not using domain theory or its variants) characterization and proof method of contextual equivalence in a language with full existential and recursive types. As discussed above, previous results in this area were (1) limited to recursive types with no negative occurrence, (2) incomplete for existential types, and/or (3) apt to be technically involved in general.

Many of the ideas used here are drawn from our previous work [Sumii and Pierce

2004] on a sound and complete bisimulation for untyped λ-calculus with *dynamic sealing* (also known as *perfect encryption*). This form of information hiding is very different from static type abstraction. Given the difference, it is surprising (and interesting) in itself to find that similar ideas can be adapted to both settings. Furthermore, the language in the present paper is typed (unlike in our previous work), requiring many refinements to take type information into account throughout the technical development. In general, typed equivalence is much coarser than untyped equivalence—in particular with polymorphism—because not only terms but also contexts have to respect types. Accordingly, our bisimulation keeps careful track of the mapping of abstract type variables to concrete types, substituting the former with the latter if and only if appropriate.

Overview. The rest of this paper is structured as follows. Section 2 presents our language and its contextual equivalence, generalized in a non-trivial way for open types as required by the technicalities which follow. Section 3 defines our bisimulation. Section 4 proves soundness and completeness of the bisimulation with respect to the generalized contextual equivalence and Section 5 gives examples to illustrate its uses. Section 6 generalizes these results, which have been restricted to closed values for simplicity, to non-values and open terms. Section 7 discusses an extension of our bisimulation concerning higher-order functions and an "up-to context" technique. Section 8 discusses related work, and Section 9 concludes with future work.

Throughout the paper, we use overbars as shorthands for sequences—e.g., we write $\overline{x}$, $[\overline{V}/\overline{x}]$, $(\overline{\alpha}, \overline{\sigma}, \overline{\sigma'})$ and $\overline{x}:\overline{\tau}$ instead of $x_1, \ldots, x_n$, $[V_1, \ldots, V_n/x_1, \ldots, x_n]$, $(\alpha_1, \sigma_1, \sigma'_1), \ldots, (\alpha_n, \sigma_n, \sigma'_n)$ and $x_1:\tau_1, \ldots, x_n:\tau_n$ where $n \geq 0$.

2. GENERALIZED CONTEXTUAL EQUIVALENCE

Our language is call-by-value λ-calculus with polymorphic and recursive types. (We conjecture that it would also be straightforward to adapt our method to a call-by-name setting.) Its syntax, semantics, and typing rules are given in Figures 1, 2, and 3. We include recursive functions $\texttt{fix}\ f(x:\tau):\sigma = M$ as a primitive for the sake of exposition; alternatively, they can be implemented in terms of a fixed-point operator, which is typable using recursive types. We adopt the standard notion of variable binding with implicit α-conversion and write $\lambda x:\tau.\, M$ for $\texttt{fix}\ f(x:\tau):\sigma = M$ when f is not free in M. We will write $\texttt{let}\ x:\tau = M\ \texttt{in}\ N$ for $(\lambda x:\tau.\, N)M$. We sometimes omit type annotations—as in $\lambda x.\, M$ and $\texttt{let}\ x = M\ \texttt{in}\ N$—when they are obvious from the context. The semantics is defined by the evaluation $M \Downarrow V$ of term M to value V. We choose not to evaluate under Λ (unlike [Pitts 2005], for example), though this choice is not essential for our developments. As is often the case in polymorphic λ-calculus, a type environment Γ is the union of (1) a finite set of type variables and (2) a finite map from variables to types. We write Γ, α for $\Gamma \cup \{\alpha\}$ when $\alpha \notin \Gamma$, and $\Gamma, x : \tau$ for $\Gamma \cup \{(x, \tau)\}$ when $x \notin dom(\Gamma)$, where $dom(\Gamma)$ denotes $\{\overline{x}\}$ for $\Gamma = \{\overline{\alpha}, (\overline{x}, \overline{\tau})\}$.

For simplicity, we consider the equivalence of closed values only. (This restriction entails no loss of generality: see Section 6.) However, in order to formalize the soundness and completeness of our bisimulation with respect to contextual equivalence, it helps to extend the definition of contextual equivalence to values

$M, N, C, D ::=$		term
	x	variable
	$\texttt{fix}\ f(x\!:\!\tau)\!:\!\sigma = M$	recursive function
	MN	application
	$\Lambda\alpha.\, M$	type function
	$M[\tau]$	type application
	$\texttt{pack}\ \tau, M\ \texttt{as}\ \exists\alpha.\,\sigma$	packing
	$\texttt{open}\ M\ \texttt{as}\ \alpha, x\ \texttt{in}\ N$	opening
	$\langle M_1, \ldots, M_n\rangle$	tupling
	$\#_i(M)$	projection
	$\texttt{in}_i(M)$	injection
	$\texttt{case}\ M\ \texttt{of}\ \texttt{in}_1(x_1) \Rightarrow M_1 \parallel \ldots \parallel \texttt{in}_n(x_n) \Rightarrow M_n$	case branch
	$\texttt{fold}(M)$	folding
	$\texttt{unfold}(M)$	unfolding
$U, V, W ::=$		value
	$\texttt{fix}\ f(x\!:\!\tau)\!:\!\sigma = M$	recursive function
	$\Lambda\alpha.\, M$	type function
	$\texttt{pack}\ \tau, V\ \texttt{as}\ \exists\alpha.\,\sigma$	package
	$\langle V_1, \ldots, V_n\rangle$	tuple
	$\texttt{in}_i(V)$	injected value
	$\texttt{fold}(V)$	folded value
$\pi, \rho, \sigma, \tau ::=$		type
	α	type variable
	$\tau \to \sigma$	function type
	$\forall\alpha.\,\tau$	universal type
	$\exists\alpha.\,\tau$	existential type
	$\tau_1 \times \ldots \times \tau_n$	product type
	$\tau_1 + \ldots + \tau_n$	sum type
	$\mu\alpha.\,\tau$	recursive type

Fig. 1. Syntax

of open *types*. For instance, we will have to consider whether $\lambda x\!:\!\texttt{int}.\,x$ is contextually equivalent to $\lambda x\!:\!\texttt{int}.\,x - 1$ at type $\alpha \to \texttt{int}$, where the implementation of abstract type α is $\texttt{int}$ in fact. But this clearly depends on what values of type α (or, more generally, what values involving type α) exist in the context: for instance, if the only values of type α are 2 in the left-hand world and 3 in the right-hand world, then the equivalence does hold; however, if some integers i on the left and j on the right have type α where $i \neq j - 1$, then it does not hold. In order to capture at once all such values in the context involving type α, we consider the equivalence of *multiple* pairs of values—annotated with their types—such as $\{(2, 3, \alpha), ((\lambda x\!:\!\texttt{int}.\,x), (\lambda x\!:\!\texttt{int}.\,x - 1), \alpha \to \texttt{int})\}$ and $\{(i, j, \alpha), ((\lambda x\!:\!\texttt{int}.\,x), (\lambda x\!:\!\texttt{int}.\,x - 1), \alpha \to \texttt{int})\}$; the former should be included in the equivalence while the latter should not, provided that $i \neq j - 1$. For this reason, we define a generalized form of contextual equivalence as follows.

DEFINITION 2.1. *A* concretion environment Δ *is a finite set of triples of the form* $(\alpha, \sigma, \sigma')$ *with* σ, σ' *closed types and* $(\alpha, \tau, \tau') \in \Delta \wedge (\alpha, \sigma, \sigma') \in \Delta \Rightarrow \tau = \sigma \wedge \tau' = \sigma'$.

The intuition is that, under Δ, abstract type α is implemented by concrete type σ in the left-hand side and by another concrete type σ' in the right-hand side (of an equivalence). For instance, in the example in Section 1, the concrete imple-

$$\frac{}{(\texttt{fix}\ f(x\!:\!\tau)\!:\!\sigma = M) \Downarrow (\texttt{fix}\ f(x\!:\!\tau)\!:\!\sigma = M)}\text{(E-Fix)}$$

$$\frac{M \Downarrow (\texttt{fix}\ f(x\!:\!\tau)\!:\!\sigma = M_0) \quad N \Downarrow V \quad [V/x][(\texttt{fix}\ f(x\!:\!\tau)\!:\!\sigma = M_0)/f]M \Downarrow W}{MN \Downarrow W}\text{(E-App)}$$

$$\frac{}{\Lambda\alpha.\,M \Downarrow \Lambda\alpha.\,M}\text{(E-TAbs)} \qquad \frac{M \Downarrow \Lambda\alpha.\,N \quad [\tau/\alpha]N \Downarrow V}{M[\tau] \Downarrow V}\text{(E-TApp)}$$

$$\frac{M \Downarrow V}{\texttt{pack}\ \tau, M\ \texttt{as}\ \exists\alpha.\,\sigma \Downarrow \texttt{pack}\ \tau, V\ \texttt{as}\ \exists\alpha.\,\sigma}\text{(E-Pack)}$$

$$\frac{M \Downarrow \texttt{pack}\ \sigma, V\ \texttt{as}\ \exists\alpha.\,\tau \quad [V/x][\sigma/\alpha]N \Downarrow W}{\texttt{open}\ M\ \texttt{as}\ \alpha, x\ \texttt{in}\ N \Downarrow W}\text{(E-Open)}$$

$$\frac{M_1 \Downarrow V_1 \quad \dots \quad M_n \Downarrow V_n}{\langle M_1, \dots, M_n\rangle \Downarrow \langle V_1, \dots, V_n\rangle}\text{(E-Tuple)} \qquad \frac{M \Downarrow \langle V_1, \dots, V_i, \dots, V_n\rangle}{\#_i(M) \Downarrow V_i}\text{(E-Proj)}$$

$$\frac{M \Downarrow V}{\texttt{in}_i(M) \Downarrow \texttt{in}_i(V)}\text{(E-Inj)}$$

$$\frac{M \Downarrow \texttt{in}_i(V) \quad [V/x_i]M_i \Downarrow W}{\texttt{case}\ M\ \texttt{of}\ \texttt{in}_1(x_1) \Rightarrow M_1 \parallel \dots \parallel \texttt{in}_i(x_i) \Rightarrow M_i \parallel \dots \parallel \texttt{in}_n(x_n) \Rightarrow M_n \Downarrow W}\text{(E-Case)}$$

$$\frac{M \Downarrow V}{\texttt{fold}(M) \Downarrow \texttt{fold}(V)}\text{(E-Fold)} \qquad \frac{M \Downarrow \texttt{fold}(V)}{\texttt{unfold}(M) \Downarrow V}\text{(E-Unfold)}$$

Fig. 2. Semantics

mentations of abstract type α were `int` in the left-hand world and `bool` in the right-hand world, so Δ was $\{(\alpha, \texttt{int}, \texttt{bool})\}$. We write $dom(\Delta)$ for $\{\alpha_1, \dots, \alpha_n\}$ when $\Delta = \{(\alpha_1, \sigma_1, \sigma'_1), \dots, (\alpha_n, \sigma_n, \sigma'_n)\}$ and write $\Delta_1 \uplus \Delta_2$ for $\Delta_1 \cup \Delta_2$ when $dom(\Delta_1) \cap dom(\Delta_2) = \emptyset$.

Definition 2.2. *A* typed value relation $\mathcal{R}$ *is a (either finite or infinite) set of triples of the form* (V, V', τ).

The intuition is that $\mathcal{R}$ relates value V in the left-hand side and value V' in the right-hand side at type τ.

Definition 2.3. *Let* $\Delta = \{(\alpha_1, \sigma_1, \sigma'_1), \dots, (\alpha_m, \sigma_m, \sigma'_m)\}$. *We write* $\Delta \vdash \mathcal{R}$ *if, for any* $(V, V', \tau) \in \mathcal{R}$, *we have* $\vdash V : [\overline{\sigma}/\overline{\alpha}]\tau$ *and* $\vdash V' : [\overline{\sigma}'/\overline{\alpha}]\tau$.

Definition 2.4 Typed Value Relation in Context. *We write* $(\Delta, \mathcal{R})^\circ$ *for the relation*

$$\begin{aligned}\{([\overline{U}/\overline{y}][\overline{\sigma}/\overline{\alpha}]D,\ [\overline{U}'/\overline{y}][\overline{\sigma}'/\overline{\alpha}]D,\ \tau) \mid \\ \Delta = \{(\alpha_1, \sigma_1, \sigma'_1), \dots, (\alpha_m, \sigma_m, \sigma'_m)\}, \\ (U_1, U'_1, \rho_1), \dots, (U_n, U'_n, \rho_n) \in \mathcal{R}, \\ \alpha_1, \dots, \alpha_m, y_1 : \rho_1, \dots, y_n : \rho_n \vdash D : \tau\}.\end{aligned}$$

Intuitively, this relation represents contexts D into which values $\overline{U}$ and $\overline{U}'$, which are related by $\mathcal{R}$, have been put.

$$\frac{(x,\tau) \in \Gamma}{\Gamma \vdash x : \tau}\text{(T-Var)}$$

$$\frac{FTV(\tau) \subseteq \Gamma \quad \Gamma, f : \tau \to \sigma, x : \tau \vdash M : \sigma}{\Gamma \vdash (\texttt{fix}\ f(x:\tau):\sigma = M) : \tau \to \sigma}\text{(T-Fix)} \qquad \frac{\Gamma \vdash M : \tau \to \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash MN : \sigma}\text{(T-App)}$$

$$\frac{\Gamma, \alpha \vdash M : \tau}{\Gamma \vdash \Lambda\alpha.\, M : \forall\alpha.\, \tau}\text{(T-TAbs)} \qquad \frac{\Gamma \vdash M : \forall\alpha.\, \sigma \quad FTV(\tau) \subseteq \Gamma}{\Gamma \vdash M[\tau] : [\tau/\alpha]\sigma}\text{(T-TApp)}$$

$$\frac{FTV(\tau) \subseteq \Gamma \quad \Gamma \vdash M : [\tau/\alpha]\sigma}{\Gamma \vdash \texttt{pack}\ \tau, M\ \texttt{as}\ \exists\alpha.\, \sigma : \exists\alpha.\, \sigma}\text{(T-Pack)}$$

$$\frac{\Gamma \vdash M : \exists\alpha.\, \tau \quad \Gamma, \alpha, x : \tau \vdash N : \sigma \quad \alpha \notin FTV(\sigma)}{\Gamma \vdash \texttt{open}\ M\ \texttt{as}\ \alpha, x\ \texttt{in}\ N : \sigma}\text{(T-Open)}$$

$$\frac{\Gamma \vdash M_1 : \tau_1 \quad \ldots \quad \Gamma \vdash M_n : \tau_n}{\Gamma \vdash \langle M_1, \ldots, M_n\rangle : \tau_1 \times \ldots \times \tau_n}\text{(T-Tuple)} \qquad \frac{\Gamma \vdash M : \tau_1 \times \ldots \times \tau_i \times \ldots \times \tau_n}{\Gamma \vdash \#_i(M) : \tau_i}\text{(T-Proj)}$$

$$\frac{\Gamma \vdash M : \tau_i \quad FTV(\tau_1) \subseteq \Gamma \quad \ldots \quad FTV(\tau_n) \subseteq \Gamma}{\Gamma \vdash \texttt{in}_i(M) : \tau_1 + \ldots + \tau_i + \ldots + \tau_n}\text{(T-Inj)}$$

$$\frac{\Gamma \vdash M : \tau_1 + \ldots + \tau_n \quad \Gamma, x_1 : \tau_1 \vdash M_1 : \tau \quad \ldots \quad \Gamma, x_n : \tau_n \vdash M_n : \tau}{\Gamma \vdash \texttt{case}\ M\ \texttt{of}\ \texttt{in}_1(x_1) \Rightarrow M_1 \parallel \ldots \parallel \texttt{in}_n(x_n) \Rightarrow M_n : \tau}\text{(T-Case)}$$

$$\frac{\Gamma \vdash M : [\mu\alpha.\, \tau/\alpha]\tau}{\Gamma \vdash \texttt{fold}(M) : \mu\alpha.\, \tau}\text{(T-Fold)} \qquad \frac{\Gamma \vdash M : \mu\alpha.\, \tau}{\Gamma \vdash \texttt{unfold}(M) : [\mu\alpha.\, \tau/\alpha]\tau}\text{(T-Unfold)}$$

Fig. 3. Typing Rules

Definition 2.5. Generalized contextual equivalence *is the set* $\equiv$ *of all pairs* $(\Delta, \mathcal{R})$ *such that:*

A. $\Delta \vdash \mathcal{R}$.

B. For any $(M, M', \tau) \in (\Delta, \mathcal{R})^\circ$, *we have* $M \Downarrow \iff M' \Downarrow$.

Note that the standard contextual equivalence—between two closed values of a closed type—is subsumed by the case where each Δ is empty and each $\mathcal{R}$ is a singleton. Conversely, the standard contextual equivalence is implied by the generalized one in the following sense: if $(V, V', \tau) \in \mathcal{R}$ for some $(\Delta, \mathcal{R}) \in \equiv$ where V, V', and τ are closed, then it is immediate by definition that $K[V] \Downarrow \iff K[V'] \Downarrow$ for any context K with a hole $[\,]$ for terms of type τ. More generally, if $(\Delta, \mathcal{R}) \in \equiv$ for $\Delta = \{(\overline{\alpha}, \overline{\sigma}, \overline{\sigma}')\}$ and $\mathcal{R} = \{(\overline{V}, \overline{V}', \overline{\tau})\}$, then

$$\texttt{pack}\ \overline{\sigma}, (V_1, \ldots, V_n)\ \texttt{as}\ \exists\overline{\alpha}.\, \tau_1 \times \ldots \times \tau_n$$

and

$$\texttt{pack}\ \overline{\sigma}', (V_1', \ldots, V_n')\ \texttt{as}\ \exists\overline{\alpha}.\, \tau_1 \times \ldots \times \tau_n$$

are contextually equivalent in the standard sense. (A formal proof of this can be given by observing that the packages above are bisimilar as defined in Section 3.) See also Section 6 for discussions on non-values and open terms.

We write

$$\Delta \vdash V_1, V_2, \ldots \equiv V'_1, V'_2, \ldots : \tau_1, \tau_2, \ldots$$

for

$$(\Delta, \{(V_1, V'_1, \tau_1), (V_2, V'_2, \tau_2), \ldots\}) \in \equiv.$$

We also write $\Delta \vdash V \equiv_{\mathcal{R}} V' : \tau$ for $(V, V', \tau) \in \mathcal{R}$ with $(\Delta, \mathcal{R}) \in \equiv$. Intuitively, this can be read, "values V and V' have type τ under concretion environment Δ and are contextually equivalent under knowledge $\mathcal{R}$."

The following properties follow immediately from the definition above.

Corollary 2.6 Reflexivity. *If* $\vdash V_1 : [\overline{\sigma}/\overline{\alpha}]\tau_1$, $\vdash V_2 : [\overline{\sigma}/\overline{\alpha}]\tau_2$, $\ldots$, *then*

$$\{(\overline{\alpha}, \overline{\sigma}, \overline{\sigma})\} \vdash V_1, V_2, \ldots \equiv V_1, V_2, \ldots : \tau_1, \tau_2, \ldots.$$

Corollary 2.7 Symmetry. *If*

$$\{(\overline{\alpha}, \overline{\sigma}, \overline{\sigma}')\} \vdash V_1, V_2, \ldots \equiv V'_1, V'_2, \ldots : \tau_1, \tau_2, \ldots$$

then

$$\{(\overline{\alpha}, \overline{\sigma}', \overline{\sigma})\} \vdash V'_1, V'_2, \ldots \equiv V_1, V_2, \ldots : \tau_1, \tau_2, \ldots.$$

Corollary 2.8 Transitivity. *If*

$$\{(\overline{\alpha}, \overline{\sigma}, \overline{\sigma}')\} \vdash V_1, V_2, \ldots \equiv V'_1, V'_2, \ldots : \tau_1, \tau_2, \ldots$$

and

$$\{(\overline{\alpha}, \overline{\sigma}', \overline{\sigma}'')\} \vdash V'_1, V'_2, \ldots \equiv V''_1, V''_2, \ldots : \tau_1, \tau_2, \ldots$$

then

$$\{(\overline{\alpha}, \overline{\sigma}, \overline{\sigma}'')\} \vdash V_1, V_2, \ldots \equiv V''_1, V''_2, \ldots : \tau_1, \tau_2, \ldots.$$

Example 2.9. *Suppose that our language is extended in the obvious way with integers and booleans (these are, of course, definable in the language we have already given, but we prefer not to clutter examples with encodings), and let* $\Delta = \{(\alpha, \mathtt{int}, \mathtt{int})\}$. *Then we shall have:*

$$\begin{aligned} \Delta \vdash\ & 2,\ (\lambda x : \mathtt{int}.\, x) \\ \equiv\ & 3,\ (\lambda x : \mathtt{int}.\, x - 1) \\ :\ & \alpha,\ (\alpha \to \mathtt{int}) \end{aligned}$$

More generally,

$$\begin{aligned} \Delta \vdash\ & i,\ (\lambda x : \mathtt{int}.\, x) \\ \equiv\ & j,\ (\lambda x : \mathtt{int}.\, x - 1) \\ :\ & \alpha,\ (\alpha \to \mathtt{int}) \end{aligned}$$

if and only if $i = j - 1$. *Formal proofs of these equivalences can be given by using the bisimulation method presented in the following sections. Intuitively, these equivalences should hold because the only way to use the values of type* α *is to apply the functions of type* $\alpha \to \mathtt{int}$.

EXAMPLE 2.10. *Let* $\Delta = \{(\alpha, \texttt{int}, \texttt{bool})\}$*. We have*

$$\begin{array}{rl}\Delta \vdash & 1,\ (\lambda x\!:\!\texttt{int}.\, x \overset{\texttt{int}}{=} 0) \\ \equiv & \texttt{true},\ (\lambda x\!:\!\texttt{bool}.\, \neg x) \\ : & \alpha,\ (\alpha \to \texttt{bool})\end{array}$$

$$\begin{array}{rl}\Delta \vdash & 1,\ (\lambda x\!:\!\texttt{int}.\, x \overset{\texttt{int}}{=} 0) \\ \equiv & \texttt{false},\ (\lambda x\!:\!\texttt{bool}.\, x) \\ : & \alpha,\ (\alpha \to \texttt{bool})\end{array}$$

but

$$\begin{array}{rl}\Delta \vdash & 1,\ (\lambda x\!:\!\texttt{int}.\, x \overset{\texttt{int}}{=} 0),\ 1,\ (\lambda x\!:\!\texttt{int}.\, x \overset{\texttt{int}}{=} 0) \\ \not\equiv & \texttt{true},\ (\lambda x\!:\!\texttt{bool}.\, \neg x),\ \texttt{false},\ (\lambda x\!:\!\texttt{bool}.\, x) \\ : & \alpha,\ (\alpha \to \texttt{bool}),\ \alpha,\ (\alpha \to \texttt{bool}).\end{array}$$

Again, the first two equivalences can be proved by means of bisimulations, defined in the next section. The last inequivalence can be proved by taking $D = \texttt{if}\ y_4 y_1\ \texttt{then}\ \bot\ \texttt{else}\ 0$ *(or, alternatively,* $D = \texttt{if}\ y_2 y_3\ \texttt{then}\ \bot\ \texttt{else}\ 0$*) in Definition 2.4 (and 2.5), where* $\bot$ *is any divergent term, e.g.,* $(\texttt{fix}\ f(x\!:\!\texttt{int})\!:\!\texttt{int} = f\,x)\,0$.

The last example shows that, even if $(\Delta, \mathcal{R}_1) \in\ \equiv$ and $(\Delta, \mathcal{R}_2) \in\ \equiv$, the union $(\Delta, \mathcal{R}_1 \cup \mathcal{R}_2)$ does not always belong to $\equiv$. In other words, one should not confuse two different implementations of an abstract type, even if each of them is correct in itself.

3. BISIMULATION

Contextual equivalence is difficult to prove directly, because it involves a universal quantification over arbitrary contexts. Fortunately, we can avoid considering all contexts by observing that only a few "primitive" operations can actually be performed on the values that contexts have access to: for instance, if a context is comparing a pair $\langle v, w\rangle$ with another pair $\langle v', w'\rangle$, all it can do is to project the first elements v and v' or the second elements w and w' (and add them to its knowledge for later use). Similarly, in order to compare functions $\lambda x.\, M$ and $\lambda x.\, M'$, a context has to apply them to some arguments it can make up from its knowledge. Intuitively, our bisimulations are sets of relations representing such contextual knowledge, closed under increase of knowledge via primitive operations like projection and application.

Based on the ideas above, our bisimulation is defined as follows. More detailed technical intuitions will be given after the definition.

DEFINITION 3.1 BISIMULATION. *A* bisimulation *is a set* X *of pairs* $(\Delta, \mathcal{R})$ *such that:*

(*1*) $\Delta \vdash \mathcal{R}$.

(*2*) *For each*

$$(\texttt{fix}\ f(x\!:\!\pi)\!:\!\rho = M,\ \texttt{fix}\ f(x\!:\!\pi')\!:\!\rho' = M',\ \tau \to \sigma) \in \mathcal{R}$$

and for any $(V, V', \tau) \in (\Delta, \mathcal{R})^\circ$, *we have*

$$\begin{array}{rl} & (\mathtt{fix}\ f(x : \pi) : \rho = M)V \Downarrow \\ \iff & (\mathtt{fix}\ f(x : \pi') : \rho' = M')V' \Downarrow. \end{array}$$

Furthermore, if $(\mathtt{fix}\ f(x : \pi) : \rho = M)V \Downarrow W$ *and* $(\mathtt{fix}\ f(x : \pi') : \rho' = M')V' \Downarrow W'$, *then*

$$(\Delta, \mathcal{R} \cup \{(W, W', \sigma)\}) \in X.$$

(*3*) *Let* $\Delta = \{(\alpha_1, \sigma_1, \sigma'_1), \dots, (\alpha_m, \sigma_m, \sigma'_m)\}$. *For each*

$$(\Lambda\alpha.\, M, \Lambda\alpha.\, M', \forall\alpha.\, \tau) \in \mathcal{R}$$

and for any ρ *with* $FTV(\rho) \subseteq dom(\Delta)$, *we have*

$$(\Lambda\alpha.\, M)[[\overline{\sigma}/\overline{\alpha}]\rho] \Downarrow \iff (\Lambda\alpha.\, M')[[\overline{\sigma}'/\overline{\alpha}]\rho] \Downarrow.$$

Furthermore, if $(\Lambda\alpha.\, M)[[\overline{\sigma}/\overline{\alpha}]\rho] \Downarrow W$ *and* $(\Lambda\alpha.\, M')[[\overline{\sigma}'/\overline{\alpha}]\rho] \Downarrow W'$, *then*

$$(\Delta, \mathcal{R} \cup \{(W, W', [\rho/\alpha]\tau)\}) \in X.$$

(*4*) *For each*

$$(\mathtt{pack}\ \sigma, V\ \mathtt{as}\ \exists\alpha.\, \tau,\ \mathtt{pack}\ \sigma', V'\ \mathtt{as}\ \exists\alpha.\, \tau',\ \exists\alpha.\, \tau'') \in \mathcal{R},$$

we have either

$$(\Delta \uplus \{(\alpha, \sigma, \sigma')\}, \mathcal{R} \cup \{(V, V', \tau'')\}) \in X,$$

or else $(\beta, \sigma, \sigma') \in \Delta$ *and* $(V, V', [\beta/\alpha]\tau'') \in \mathcal{R}$ *for some* β.

(*5*) *For each* $(\langle V_1, \dots, V_n\rangle, \langle V'_1, \dots, V'_n\rangle, \tau_1 \times \dots \times \tau_n) \in \mathcal{R}$ *and for any* $1 \le i \le n$, *we have* $(\Delta, \mathcal{R} \cup \{(V_i, V'_i, \tau_i)\}) \in X$.

(*6*) *For each* $(\mathtt{in}_i(V), \mathtt{in}_j(V'), \tau_1 + \dots + \tau_n) \in \mathcal{R}$, *we have* $i = j$ *and* $(\Delta, \mathcal{R} \cup \{(V, V', \tau_i)\}) \in X$.

(*7*) *For each* $(\mathtt{fold}(V), \mathtt{fold}(V'), \mu\alpha.\, \tau) \in \mathcal{R}$, *we have* $(\Delta, \mathcal{R} \cup \{(V, V', [\mu\alpha.\, \tau/\alpha]\tau)\}) \in X$.

As usual, *bisimilarity*, written $\sim$, is the largest bisimulation (which is the union of all bisimulations); it exists because the union of any bisimulations is again a bisimulation.

We write

$$\Delta \vdash V_1, \dots, V_n\ X\ V'_1, \dots V'_n : \tau_1, \dots, \tau_n$$

for

$$(\Delta, \{(V_1, V'_1, \tau_1), \dots, (V_n, V'_n, \tau_n)\}) \in X.$$

We also write $\Delta \vdash V\ X_{\mathcal{R}}\ V' : \tau$ for $(V, V', \tau) \in \mathcal{R}$ with $(\Delta, \mathcal{R}) \in X$. Intuitively, it can be read: values V and V' of type τ with concretion environment Δ are bisimilar under knowledge $\mathcal{R}$. (This may be reminiscent of Kripke logical relations [Mitchell 1996, page 590], but our bisimulation is not monotone about $\mathcal{R}$ as discussed in the introduction.)

We now elaborate the intuitions behind the definition of bisimulation. Condition 1 ensures that bisimilar values V and V' are well typed under the concretion environment Δ. The other conditions require that the bisimulation is closed under the increase of a context's knowledge via primitive operations, as explained below.

Condition 2 deals with the case where a context applies two functions it knows ($\texttt{fix}\ f(x\!:\!\pi)\!:\!\rho = M$ and $\texttt{fix}\ f(x\!:\!\pi')\!:\!\rho' = M'$) to some arguments V and V'. To make up these arguments, the context can make use of any values it already knows ($\overline{U}$ and $\overline{U}'$ in Definition 2.4) and assemble them using a term D with free variables $\overline{y}$, where the abstract types $\overline{\alpha}$ (in Definition 2.4) are kept abstract.

The crucial observation here is that it suffices to consider value arguments only, i.e., only the cases where the assembled terms $[\overline{U}/\overline{y}][\overline{\sigma}/\overline{\alpha}]D$ and $[\overline{U}'/\overline{y}][\overline{\sigma}'/\overline{\alpha}]D'$ are values. This simplification is essential for proving the bisimilarity of functions. Intuitively, it can be understood via the fact that any *terms* of the form $[\overline{U}/\overline{y}][\overline{\sigma}/\overline{\alpha}]D$ and $[\overline{U}'/\overline{y}][\overline{\sigma}'/\overline{\alpha}]D$ evaluate to *values* of the same form, as proved in Lemma 4.3 below.

Then, to avoid exhibiting an observable difference in behaviors, the function applications should either both diverge or else both converge; in the latter case, the resulting values become part of the context's knowledge and can be used for further experiments.[2]

Condition 3 is similar to Condition 2, but for type application rather than term application.

Condition 4 is for packages defining an abstract type α. Essentially, a context can open the two packages and examine their contents only abstractly, as expressed in the first half of this condition. However, if the context happens to know another abstract type β whose implementations coincide with α's, there is no need for us to consider them twice. The second half of the condition expresses this simplification. It is not so crucial as the previous simplification in Condition 2, but it is useful for proving the bisimulation of packages, keeping X finite in many cases despite the generativity of `open`, as we mentioned in the introduction.

Conditions 5, 6, and 7 are for tuples, injected values, and folded values, respectively. They capture the straightforward increase of the context's knowledge via projection, case branch, or unfolding.

4. SOUNDNESS AND COMPLETENESS

We prove that bisimilarity coincides with contextual equivalence (in the generalized form presented in Section 2). That is, two values can be proved to be bisimilar if and only if they are contextually equivalent.

First, we prove the "if" part, i.e., that contextual equivalence is included in bisimilarity. This direction is easier because our bisimulation is defined co-inductively: it suffices simply to prove that contextual equivalence is a bisimulation.

LEMMA 4.1 COMPLETENESS OF BISIMULATION. *$\equiv \subseteq \sim$.*

PROOF. By checking that $\equiv$ satisfies each condition of bisimulation. Details can

[2]Another technical point may deserve mentioning here: instead of $(\Delta, \mathcal{R} \cup \{(W, W', \sigma)\}) \in X$, we could require $(W, W', \sigma) \in \mathcal{R}$ to reduce the number of $\mathcal{R}$s required to be in X by "predicting" the increase of contexts' knowledge *a priori*. We rejected this alternative for the sake of uniformity with Condition 4, which anyway requires the concretion environment Δ to be extended. This decision does not make it difficult to construct a bisimulation, as we will see soon in the examples.

be found in Appendix A. □

Next, we show that bisimilarity is included in contextual equivalence. To do so, we need to consider the question: When we put bisimilar values into a context and evaluate them, what changes? The answer is: Nothing! I.e., evaluating a pair of expressions, each consisting of some set of bisimilar values placed in some context, results again in a pair of expressions that can be described by some set of bisimilar values placed in some context. Furthermore, this evaluation converges in the left-hand side if and only if it converges in the right-hand side. Since the proof of the latter property requires the former property, we formalize the observations above in the following order.

Definition 4.2 Bisimilarity in Context. *We write* $\Delta \vdash N \sim^{\circ}_{\mathcal{R}} N' : \tau$ *if* $(N, N', \tau) \in (\Delta, \mathcal{R})^{\circ}$ *and* $(\Delta, \mathcal{R}) \in \sim$.

The intuition is that $\sim^{\circ}$ relates bisimilar values put in contexts.

Lemma 4.3 Fundamental Property, Part I. *Suppose* $\Delta_0 \vdash N \sim^{\circ}_{\mathcal{R}_0} N' : \tau$. *If* $N \Downarrow W$ *and* $N' \Downarrow W'$, *then* $\Delta \vdash W \sim^{\circ}_{\mathcal{R}} W' : \tau$ *for some* $\Delta \supseteq \Delta_0$ *and* $\mathcal{R} \supseteq \mathcal{R}_0$.

Proof. By induction on the derivation of $N \Downarrow W$. Details are found in Appendix B. □

Lemma 4.4 Fundamental Property, Part II. *If* $\Delta_0 \vdash N \sim^{\circ}_{\mathcal{R}_0} N' : \tau$ *then* $N \Downarrow \iff N' \Downarrow$.

Proof. By induction on the derivation of $N \Downarrow$ together with Lemma 4.3. Details are in Appendix C. □

Corollary 4.5 Soundness of Bisimilarity. $\sim \subseteq \equiv$.

Proof. By the definitions of $\equiv$ and $\sim^{\circ}$ together with Lemma 4.4. □

Combining soundness and completeness, we obtain the main theorem about our bisimulation: that bisimilarity coincides with contextual equivalence.

Theorem 4.6. $\sim = \equiv$.

Proof. By Corollary 4.5 and Lemma 4.1. □

Note that these proofs are much simpler than soundness proofs of applicative bisimulations in previous work [Howe 1996; Gordon 1995a; Gordon and Rees 1996; Gordon 1995b; Gordon and Rees 1995; Abramsky 1990] thanks to the generalized condition on functions (Condition 2), which is anyway required in the presence of existential polymorphism (i.e., we cannot just apply bisimilar functions to identical arguments).

5. EXAMPLES

We develop several examples illustrating concrete applications of the bisimulation method. The first three examples involve existential packages, whose equivalence cannot be proved by other bisimulations for λ-calculi. The fourth example involves recursive types with negative occurrences, for which traditional logical relations have difficulties. The last example uses higher-order functions (continuation passing style) and universal types in place of existential packages. Our bisimulation technique yields a straightforward proof of equivalence for each of the examples.

5.1 Warm-Up

Consider the following simple packages

$$\begin{aligned} U &= \texttt{pack int}, \langle 1, \lambda x\!:\!\texttt{int}.\, x \overset{\texttt{int}}{=} 0\rangle \texttt{ as } \tau \\ U' &= \texttt{pack bool}, \langle \texttt{true}, \lambda x\!:\!\texttt{bool}.\, \neg x\rangle \texttt{ as } \tau \end{aligned}$$

where $\tau = \exists\alpha.\, \alpha \times (\alpha \to \texttt{bool})$. We aim to prove that U and U' are contextually equivalent at type τ. To this end, let

$$X = \{(\emptyset, \mathcal{R}_0), (\Delta, \mathcal{R}_1), (\Delta, \mathcal{R}_2), (\Delta, \mathcal{R}_3), (\Delta, \mathcal{R}_4), (\Delta, \mathcal{R}_5)\}$$

where

$$\begin{aligned} \Delta &= (\alpha, \texttt{int}, \texttt{bool}) \\ \mathcal{R}_0 &= \{(U, U', \tau)\} \\ \mathcal{R}_1 &= \mathcal{R}_0 \cup \{(\langle 1, \lambda x\!:\!\texttt{int}.\, x \overset{\texttt{int}}{=} 0\rangle, \\ &\qquad\qquad \langle \texttt{true}, \lambda x\!:\!\texttt{bool}.\, \neg x\rangle, \\ &\qquad\qquad \alpha \times (\alpha \to \texttt{bool}))\} \\ \mathcal{R}_2 &= \mathcal{R}_1 \cup \{(1, \texttt{true}, \alpha)\} \\ \mathcal{R}_3 &= \mathcal{R}_1 \cup \{(\lambda x\!:\!\texttt{int}.\, x \overset{\texttt{int}}{=} 0, \\ &\qquad\qquad \lambda x\!:\!\texttt{bool}.\, \neg x, \\ &\qquad\qquad \alpha \to \texttt{bool})\} \\ \mathcal{R}_4 &= \mathcal{R}_2 \cup \mathcal{R}_3 \\ \mathcal{R}_5 &= \mathcal{R}_4 \cup \{(\texttt{false}, \texttt{false}, \texttt{bool})\}. \end{aligned}$$

Then, X is a bisimulation. To prove it, we must check each condition in Definition 3.1 for every $(\Delta, \mathcal{R}) \in X$. Most of the checks are trivial, except the following cases:

—Condition 4 on $(U, U', \tau) \in \mathcal{R}_i$ for $i \geq 1$, where the second half of the condition holds.

—Condition 2 on

$$(\lambda x\!:\!\texttt{int}.\, x \overset{\texttt{int}}{=} 0,\ \lambda x\!:\!\texttt{bool}.\, \neg x,\ \alpha \to \texttt{bool}) \in \mathcal{R}_i$$

for $i \geq 3$. Since V and V' are values with $(V, V', \alpha) \in (\Delta, \mathcal{R}_i)^\circ$, the D in Definition 2.4 is either a value or a variable. However, if D is a value, it can never satisfy the assumption $\alpha, y_1\!:\!\rho_1, \ldots, y_n\!:\!\rho_n \vdash D : \alpha$ (easy case analysis on the syntax of D). Thus, D must be a variable. Without loss of generality, let $D = y_1$. Then, by inversion of (T-Var), $\rho_1 = \alpha$. Since $(U_1, U_1', \rho_1) \in \mathcal{R}_i$, we have $U_1 = 1$ and $U_1' = \texttt{true}$. Thus, $V = 1$ and $V' = \texttt{true}$, from which the rest of this condition is obvious.

Alternatively, in this particular example, we can just take $X = \{(\Delta, \mathcal{R}_5)\}$ in the first place and prove it to be a bisimulation by the same arguments as above. Since $(U, U', \tau) \in \mathcal{R}_5$, this still suffices for showing the contextual equivalence of U and U', thanks to the soundness of bisimilarity (Corollary 4.5) and the generalized definition of contextual equivalence (Definition 2.5). Simplifications like this (i.e., taking the maximum union in the first place) are possible when the example does not involve generativity (cf. Section 5.3).

5.2 Complex Numbers

Suppose now that we have real numbers and operations in the language. Then the following two implementations U and U' of complex numbers should be contextually equivalent at type $\exists\alpha.\,\tau$.

$$\begin{aligned}
U &= \texttt{pack}\ (\texttt{real}\times\texttt{real}), \langle \mathit{id}, \mathit{id}, \mathit{cmul}\rangle\ \texttt{as}\ \exists\alpha.\,\tau\\
U' &= \texttt{pack}\ (\texttt{real}\times\texttt{real}), \langle \mathit{ctop}, \mathit{ptoc}, \mathit{pmul}\rangle\ \texttt{as}\ \exists\alpha.\,\tau\\
\tau &= (\texttt{real}\times\texttt{real}\to\alpha)\times(\alpha\to\texttt{real}\times\texttt{real})\times(\alpha\to\alpha\to\alpha)
\end{aligned}$$

$$\begin{aligned}
\mathit{id} &= \lambda c{:}\texttt{real}\times\texttt{real}.\,c\\
\mathit{cmul} &= \lambda c_1{:}\texttt{real}\times\texttt{real}.\,\lambda c_2{:}\texttt{real}\times\texttt{real}.\\
&\qquad \langle \#_1(c_1)\times\#_1(c_2)-\#_2(c_1)\times\#_2(c_2),\\
&\qquad\ \ \#_2(c_1)\times\#_1(c_2)+\#_1(c_1)\times\#_2(c_2)\rangle
\end{aligned}$$

$$\begin{aligned}
\mathit{ctop} &= \lambda c{:}\texttt{real}\times\texttt{real}.\\
&\qquad \langle \sqrt{(\#_1(c))^2+(\#_2(c))^2},\ \operatorname{atan2}(\#_2(c),\#_1(c))\rangle\\
\mathit{ptoc} &= \lambda p{:}\texttt{real}\times\texttt{real}.\\
&\qquad \langle \#_1(p)\times\cos(\#_2(p)),\ \#_1(p)\times\sin(\#_2(p))\rangle\\
\mathit{pmul} &= \lambda p_1{:}\texttt{real}\times\texttt{real}.\,\lambda p_2{:}\texttt{real}\times\texttt{real}.\\
&\qquad \langle \#_1(p_1)\times\#_1(p_2),\ \#_2(p_1)+\#_2(p_2)\rangle
\end{aligned}$$

The first functions in these packages make a complex number from its real and imaginary parts, and the second functions perform the converse conversion. The third functions multiply complex numbers.

To prove the contextual equivalence of U and U', consider $X=\{(\Delta,\mathcal{R})\}$ where

$$\begin{aligned}
\Delta &= \{(\alpha,\texttt{real}\times\texttt{real},\texttt{real}\times\texttt{real})\}\\
\mathcal{R} &= \{(U,U',\exists\alpha.\,\tau),\\
&\qquad (\langle \mathit{id},\mathit{id},\mathit{cmul}\rangle,\langle \mathit{ctop},\mathit{ptoc},\mathit{pmul}\rangle,\tau),\\
&\qquad (\mathit{id},\mathit{ctop},\texttt{real}\times\texttt{real}\to\alpha),\\
&\qquad (\mathit{id},\mathit{ptoc},\alpha\to\texttt{real}\times\texttt{real}),\\
&\qquad (\mathit{cmul},\mathit{pmul},\alpha\to\alpha\to\alpha)\}\\
&\quad \cup\ \{(v,w,\alpha)\mid w=\langle r,t\rangle,\\
&\qquad\qquad \langle r\times\cos(t), r\times\sin(t)\rangle \Downarrow v,\\
&\qquad\qquad r\ge 0\}\\
&\quad \cup\ \{(c,c,\texttt{real}\times\texttt{real})\mid\ \vdash c:\texttt{real}\times\texttt{real}\}\\
&\quad \cup\ \{(r,r,\texttt{real})\mid\ \vdash r:\texttt{real}\}.
\end{aligned}$$

Then X is a bisimulation, as can be checked in the same manner as in the previous example.

5.3 Functions Generating Packages

The following functions U and U' *generate* packages. (I.e., they behave a bit like *functors* in ML-style module systems.)

$$\begin{array}{rcl} U & = & \lambda y\!:\!\texttt{real}.\,M \\ U' & = & \lambda y\!:\!\texttt{real}.\,M' \\ M & = & \texttt{pack real}, \langle y, \lambda x\!:\!\texttt{real}.\,x\rangle \texttt{ as } \tau \\ M' & = & \texttt{pack real}, \langle y+1.0, \lambda x\!:\!\texttt{real}.\,x-1.0\rangle \texttt{ as } \tau \\ \tau & = & \exists\alpha.\,\alpha\times(\alpha\rightarrow\texttt{real}) \end{array}$$

To prove that U is contextually equivalent to U' at type $\texttt{real}\rightarrow\tau$, it suffices to consider the following infinite bisimulation.

$$\begin{array}{rcl} X & = & \{(\Delta,\mathcal{R}) \mid \\ & & \quad \vdash r_i : \texttt{real} \text{ for } i=0,1,2,...,n, \\ & & \quad \Delta = \{(\beta_i,\texttt{real},\texttt{real}) \mid i=0,1,2,...,n\}, \\ & & \quad \mathcal{R} \subseteq \cup\{\mathcal{R}(i,r_i) \mid i=0,1,2,...,n\}\} \\ \mathcal{R}(i,r) & = & \{(U,U',\texttt{real}\rightarrow\tau), \\ & & \quad ([r/y]M,[r/y]M',\tau), \\ & & \quad (\langle r,\lambda x\!:\!\texttt{real}.\,x\rangle, \langle r+1.0,\lambda x\!:\!\texttt{real}.\,x-1.0\rangle, \beta_i\times(\beta_i\rightarrow\texttt{real})), \\ & & \quad (r, r+1.0, \beta_i), \\ & & \quad (\lambda x\!:\!\texttt{real}.\,x, \lambda x\!:\!\texttt{real}.\,x-1.0, \beta_i\rightarrow\texttt{real}), \\ & & \quad (r,r,\texttt{real})\} \end{array}$$

The generativity of U and U' is given a simple account by having a different abstract type β_i for each instantiation of U and U' with $y=r_i$. Unlike the example in Section 5.1, we cannot take the union of all $\mathcal{R}(i,r)$, because it would require an uncountably infinite number of type variables in Δ.

The inclusion of all $\mathcal{R} \subseteq \cup\{\mathcal{R}(i,r_i) \mid \ldots\}$ in the definition of X simplifies the definition of this bisimulation; although it admits some $\mathcal{R}$s that are not strictly relevant to the proof (such as those with only the elements of tuples, but without the tuples themselves), they are not a problem since they do not violate any of the conditions of bisimulation. In other words, to prove the contextual equivalence of two values, one has only to find *some* bisimulation including the values rather than the minimal one.

5.4 Recursive Types with Negative Occurrence

Consider the packages C and C' implementing counter objects as follows: each counter is implemented as a pair of its state part (of abstract type `st`) and its method part; the latter is implemented as a function that takes a state and returns the tuple of methods[3]; in this example, there are two methods in the tuple: one

[3]This implementation can be viewed as a variant of the so-called recursive existential encoding of objects (see [Bruce et al. 1999] for details), but our goal here is to illustrate the power of our bisimulation with existential recursive types, rather than to discuss the object encoding itself.

returns a new counter object with the state incremented (or, in the second implementation, decremented) by 1, while the other tells whether another counter object has been incremented (or decremented) the same number of times as the present one.

$$
\begin{aligned}
\tau &= \exists \mathtt{st}.\, \sigma \\
\sigma &= \mu \mathtt{self}.\, \mathtt{st} \times (\mathtt{st} \to \rho) \\
\rho &= \mathtt{self} \times (\mathtt{self} \to \mathtt{bool})
\end{aligned}
$$

$$
\begin{aligned}
C &= \mathtt{pack}\ \mathtt{int}, \mathtt{fold}(\langle 0, M\rangle)\ \mathtt{as}\ \tau \\
C' &= \mathtt{pack}\ \mathtt{int}, \mathtt{fold}(\langle 0, M'\rangle)\ \mathtt{as}\ \tau
\end{aligned}
$$

$$
\begin{aligned}
M &= \mathtt{fix}\ f(s : \mathtt{int}) : [\mathtt{int}/\mathtt{st}][\sigma/\mathtt{self}]\rho = \\
&\qquad \langle \mathtt{fold}(\langle s+1, f\rangle), \\
&\qquad\ \ \lambda c : [\mathtt{int}/\mathtt{st}]\sigma.\, (s \overset{\mathtt{int}}{=} \#_1(\mathtt{unfold}(c)))\rangle \\
M' &= \mathtt{fix}\ f(s : \mathtt{int}) : [\mathtt{int}/\mathtt{st}][\sigma/\mathtt{self}]\rho = \\
&\qquad \langle \mathtt{fold}(\langle s-1, f\rangle), \\
&\qquad\ \ \lambda c : [\mathtt{int}/\mathtt{st}]\sigma.\, (s \overset{\mathtt{int}}{=} \#_1(\mathtt{unfold}(c)))\rangle
\end{aligned}
$$

Let us prove the contextual equivalence of C and C' at type τ. To do so, we consider the bisimulation $X = \{(\Delta, \mathcal{R})\}$ where:

$$
\begin{aligned}
\Delta &= \{(\mathtt{st}, \mathtt{int}, \mathtt{int})\} \\
\mathcal{R} &= \{(C, C', \tau), \\
&\quad (\mathtt{fold}(\langle n, M\rangle), \mathtt{fold}(\langle -n, M'\rangle), \sigma), \\
&\quad (\langle n, M\rangle, \langle -n, M'\rangle, \mathtt{st} \times (\mathtt{st} \to [\sigma/\mathtt{self}]\rho)), \\
&\quad (n, -n, \mathtt{st}), \\
&\quad (M, M', \mathtt{st} \to [\sigma/\mathtt{self}]\rho), \\
&\quad (\langle \mathtt{fold}(\langle n+1, M\rangle), \\
&\qquad \lambda c : [\mathtt{int}/\mathtt{st}]\sigma.\, (n \overset{\mathtt{int}}{=} \#_1(\mathtt{unfold}(c)))\rangle, \\
&\quad\ \langle \mathtt{fold}(\langle -n-1, M'\rangle), \\
&\qquad \lambda c : [\mathtt{int}/\mathtt{st}]\sigma.\, (-n \overset{\mathtt{int}}{=} \#_1(\mathtt{unfold}(c)))\rangle, \\
&\quad\ \sigma \times (\sigma \to \mathtt{bool})), \\
&\quad (\lambda c : [\mathtt{int}/\mathtt{st}]\sigma.\, (n \overset{\mathtt{int}}{=} \#_1(\mathtt{unfold}(c))), \\
&\quad\ \lambda c : [\mathtt{int}/\mathtt{st}]\sigma.\, (-n \overset{\mathtt{int}}{=} \#_1(\mathtt{unfold}(c))), \\
&\quad\ \sigma \to \mathtt{bool}), \\
&\quad (\mathtt{true}, \mathtt{true}, \mathtt{bool}), \\
&\quad (\mathtt{false}, \mathtt{false}, \mathtt{bool}) \mid \\
&\quad n = 0, 1, 2, \ldots\}
\end{aligned}
$$

It can indeed be shown to be a bisimulation just as the bisimulations in previous examples. That is, our bisimulation incurs no difficulty for recursive functions or recursive types (with negative occurrence).

5.5 Dual of Existential Packages

The following higher-order functions represent the CPS-converted version of the example in Section 5.1.

$$\begin{aligned} U &= \lambda k:\sigma.\, k[\mathtt{int}]\langle 1, \lambda x:\mathtt{int}.\, x \overset{\mathtt{int}}{=} 0\rangle \\ U' &= \lambda k:\sigma.\, k[\mathtt{bool}]\langle \mathtt{true}, \lambda x:\mathtt{bool}.\, \neg x\rangle \\ \sigma &= \forall\alpha.\, \alpha \times (\alpha \to \mathtt{bool}) \to \mathtt{unit} \end{aligned}$$

It is straightforward to prove the contextual equivalence of U and U' at type $\sigma \to \mathtt{unit}$, i.e.,

$$[U/x]C \Downarrow \iff [U'/x]C \Downarrow$$

for any $x : \sigma \to \mathtt{unit} \vdash C : \tau$. Since

$$\begin{aligned} [U/x]C &= [1, (\lambda x:\mathtt{int}.\, x \overset{\mathtt{int}}{=} 0)/y, z][\mathtt{int}/\beta]D_0 \\ [U'/x]C &= [\mathtt{true}, (\lambda x:\mathtt{bool}.\, \neg x)/y, z][\mathtt{bool}/\beta]D_0 \end{aligned}$$

for $D_0 = [(\lambda k:\sigma.\, k[\beta]\langle y, z\rangle)/x]C$, it suffices to prove

$$\begin{aligned} & [1, (\lambda x:\mathtt{int}.\, x \overset{\mathtt{int}}{=} 0)/y, z][\mathtt{int}/\beta]D \Downarrow \\ \iff & [\mathtt{true}, (\lambda x:\mathtt{bool}.\, \neg x)/y, z][\mathtt{bool}/\beta]D \Downarrow \end{aligned}$$

for every $\beta, y:\beta, z:\beta \to \mathtt{bool} \vdash D : \tau$. (Note that D_0 has the same typing as D thanks to the standard substitution lemma.) However, this follows immediately from the bisimulation $\{(\Delta, \mathcal{R})\}$ where

$$\begin{aligned} \Delta &= \{(\beta, \mathtt{int}, \mathtt{bool})\} \\ \mathcal{R} &= \{(1, \mathtt{true}, \beta), \\ &\qquad (\lambda x:\mathtt{int}.\, x \overset{\mathtt{int}}{=} 0,\ \lambda x:\mathtt{bool}.\, \neg x,\ \beta \to \mathtt{bool}), \\ &\qquad (\mathtt{false}, \mathtt{false}, \mathtt{bool})\} \end{aligned}$$

along with the soundness of bisimilarity in the previous section.

6. NON-VALUES AND OPEN TERMS

So far, we have restricted ourselves to the equivalence of closed values for the sake of simplicity. In this section, we show how our method can be used for proving the standard contextual equivalence of non-values and open terms as well. (For other studies on equivalences of open terms, see [Pitts 2000; 2005] for instance.)

A *context* K in the standard sense is a term with some subterm replaced by a *hole* $[\,]$. Unlike terms, (standard) contexts are *not* identified up to α-conversion of bound variables.

$$\begin{aligned} K ::= {} & [\,] \mid \mathtt{fix}\ f(x:\tau):\sigma = K \mid KN \mid MK \mid \Lambda\alpha.\, K \mid K[\tau] \mid \mathtt{pack}\ \tau, K\ \mathtt{as}\ \exists\alpha.\,\sigma \mid \\ & \mathtt{open}\ K\ \mathtt{as}\ \alpha, x\ \mathtt{in}\ N \mid \mathtt{open}\ M\ \mathtt{as}\ \alpha, x\ \mathtt{in}\ K \mid \langle M_1, \ldots, K, \ldots, M_n\rangle \mid \end{aligned}$$

$$\#_i(K) \mid \mathtt{in}_i(K) \mid \mathtt{case}\ K\ \mathtt{of}\ \mathtt{in}_1(x_1) \Rightarrow M_1 \parallel \ldots \parallel \mathtt{in}_n(x_n) \Rightarrow M_n \mid$$
$$\mathtt{case}\ M\ \mathtt{of}\ \mathtt{in}_1(x_1) \Rightarrow M_1 \parallel \ldots \parallel \mathtt{in}_i(x_i) \Rightarrow K \parallel \ldots \parallel \mathtt{in}_n(x_n) \Rightarrow M_n \mid$$
$$\mathtt{fold}(K) \mid \mathtt{unfold}(K)$$

We write $K[M]$ for the term obtained by substituting the hole in K with M (which does not apply α-conversion and may capture free variables). Then, the standard contextual equivalence

$$\alpha_1, \ldots, \alpha_m, x_1 : \tau_1, \ldots, x_n : \tau_n \ \vdash\ M \overset{\text{std}}{\equiv} M' \ :\ \tau$$

for well-typed terms $\overline{\alpha}, \overline{x} : \overline{\tau} \ \vdash\ M \ :\ \tau$ and $\overline{\alpha}, \overline{x} : \overline{\tau} \ \vdash\ M' \ :\ \tau$ can be defined as: $K[M] \Downarrow \iff K[M'] \Downarrow$ for every context K with $\vdash K[M] : \mathtt{unit}$ and $\vdash K[M'] : \mathtt{unit}$, where $\mathtt{unit}$ is the nullary product type. (In fact, any closed, non-empty type works in place of $\mathtt{unit}$.)

We will show that the standard contextual equivalence above holds if and only if the closed values $V = \Lambda\overline{\alpha}.\, \lambda\overline{x} : \overline{\tau}.\, M$ and $V' = \Lambda\overline{\alpha}.\, \lambda\overline{x} : \overline{\tau}.\, M'$ are bisimilar, i.e.,

$$\begin{aligned} \emptyset \ \vdash\ & \Lambda\alpha_1.\ \ldots\ \Lambda\alpha_m.\, \lambda x_1 : \tau_1.\ \ldots\ \lambda x_n : \tau_n.\, M \\ \sim\ & \Lambda\alpha_1.\ \ldots\ \Lambda\alpha_m.\, \lambda x_1 : \tau_1.\ \ldots\ \lambda x_n : \tau_n.\, M' \\ :\ & \forall\alpha_1.\ \ldots\ \forall\alpha_m.\, \tau_1 \to \ldots \to \tau_n \to \tau. \end{aligned}$$

(If M and M' have no free term/type variables at all, it suffices just to take $V = \Lambda\alpha.\, M$ and $V' = \Lambda\alpha.\, M'$ for any type variable α.) The "only if" direction is obvious from the definitions of contextual equivalences—both the standard one above and the generalized one in Section 2—and from the completeness of bisimulation. To prove the "if" direction, suppose $\emptyset \vdash V \sim V' : \forall\overline{\alpha}.\, \overline{\tau} \to \tau$. By the soundness of bisimulation, we have $\emptyset \vdash V \equiv V' : \forall\overline{\alpha}.\, \overline{\tau} \to \tau$. Given any K with $\vdash K[M] : \mathtt{unit}$ and $\vdash K[M'] : \mathtt{unit}$, take $C = K[z[\alpha_1] \ldots [\alpha_m] x_1 \ldots x_n]$ for fresh z. Then, it suffices to prove $K[M] \Downarrow \iff [V/z]C \Downarrow$ and $K[M'] \Downarrow \iff [V'/z]C \Downarrow$.

To this end, we prove the more general lemma below in order for induction to work. The intuition is that a term M and its β-expanded version $(\Lambda\overline{\alpha}.\, \lambda\overline{x} : \overline{\tau}.\, M)[\overline{\alpha}]\overline{x}$ should behave equivalently under any context. Since the free type/term variables $\overline{\alpha}$ and $\overline{x}$ are to be substituted by some types/values during evaluation under a context, this "β-expansion" relation needs to be generalized to allow nesting. Thus, we define:

Definition 6.1 β-Expansion. $\Gamma \vdash M \preceq M' : \tau$ *is the smallest (typed and substitution-closed) congruence relation between λ-terms that satisfies:*

$$\frac{\Gamma \vdash M \preceq M' : \rho \quad \{\overline{\alpha}\} \subseteq \mathit{dom}(\Gamma) \quad \Gamma \vdash \overline{x} : \overline{\tau}}{\Gamma \vdash M \preceq (\Lambda\overline{\alpha}.\, \lambda\overline{x} : \overline{\tau}.\, M')[\overline{\alpha}]\overline{x} : \rho}\ \textit{(B-Exp)}$$

Then, we can prove:

Lemma 6.2. *For any*

$$\alpha_1, \ldots, \alpha_m, x_1 : \tau_1, \ldots, x_n : \tau_n \ \vdash\ M \preceq M' \ :\ \tau,$$

for any closed $\sigma_1, \ldots, \sigma_m$, and for any $(\vdash V_1 \preceq V_1' : [\overline{\sigma}/\overline{\alpha}]\tau_1) \wedge \ldots \wedge (\vdash V_n \preceq V_n' : [\overline{\sigma}/\overline{\alpha}]\tau_n)$, we have

$$[\overline{V}/\overline{x}][\overline{\sigma}/\overline{\alpha}]M \Downarrow \iff [\overline{V}'/\overline{x}][\overline{\sigma}/\overline{\alpha}]M' \Downarrow.$$

Furthermore, if $[\overline{V}/\overline{x}][\overline{\sigma}/\overline{\alpha}]M \Downarrow W$ *and* $[\overline{V}'/\overline{x}][\overline{\sigma}/\overline{\alpha}]M' \Downarrow W'$*, then* $\vdash W \preceq W' : [\overline{\sigma}/\overline{\alpha}]\tau$.

PROOF. By induction on the derivations of $\overline{\alpha}, \overline{x} : \overline{\tau} \vdash M \preceq M' : \tau$ and $[\overline{V}/\overline{x}][\overline{\sigma}/\overline{\alpha}]M \Downarrow W$ (or $[\overline{V}'/\overline{x}][\overline{\sigma}/\overline{\alpha}]M' \Downarrow W'$). □

LEMMA 6.3. *For any* $\Gamma \vdash M : \tau$,

(*1*) $\Gamma \vdash M \preceq M : \tau$,

(*2*) $\Gamma \vdash M \preceq (\Lambda\overline{\alpha}.\, \lambda\overline{x}.\, M)[\overline{\alpha}]\overline{x} : \tau$, *and*

(*3*) $\vdash K[M] \preceq K[(\Lambda\overline{\alpha}.\, \lambda\overline{x}.\, M)[\overline{\alpha}]\overline{x}] : \mathtt{unit}$ *for any* K *with* $\vdash K[M] : \mathtt{unit}$.

PROOF. Immediate from Definition 6.1. □

THEOREM 6.4. *For any* $\overline{\alpha}, \overline{x} : \overline{\tau} \vdash M : \tau$ *and* $\overline{\alpha}, \overline{x} : \overline{\tau} \vdash M' : \tau$*, if* $\vdash \Lambda\overline{\alpha}.\, \lambda\overline{x} : \overline{\tau}.\, M \sim \Lambda\overline{\alpha}.\, \lambda\overline{x} : \overline{\tau}.\, M' : \forall\overline{\alpha}.\, \overline{\tau} \to \tau$*, then* $\overline{\alpha}, \overline{x} : \overline{\tau} \vdash M \overset{\text{std}}{\equiv} M' : \tau$.

PROOF. By the soundness of bisimulation, we have $[(\Lambda\overline{\alpha}.\, \lambda\overline{x} : \overline{\tau}.\, M)/z]C \Downarrow \iff [(\Lambda\overline{\alpha}.\, \lambda\overline{x} : \overline{\tau}.\, M')/z]C$ for any well-typed C. Thus, given K, take $C = K[z[\overline{\alpha}]\overline{x}]$ and we get $K[(\Lambda\overline{\alpha}.\, \lambda\overline{x} : \overline{\tau}.\, M)[\overline{\alpha}]\overline{x}] \Downarrow \iff K[(\Lambda\overline{\alpha}.\, \lambda\overline{x} : \overline{\tau}.\, M')[\overline{\alpha}]\overline{x}] \Downarrow$. Meanwhile, by Lemma 6.3 (3), we have $\vdash K[M] \preceq K[(\Lambda\overline{\alpha}.\, \lambda\overline{x} : \overline{\tau}.\, M)[\overline{\alpha}]\overline{x}] : \mathtt{unit}$ and $\vdash K[M'] \preceq K[(\Lambda\overline{\alpha}.\, \lambda\overline{x} : \overline{\tau}.\, M')[\overline{\alpha}]\overline{x}] : \mathtt{unit}$. By Lemma 6.2, we obtain $K[M] \Downarrow \iff K[(\Lambda\overline{\alpha}.\, \lambda\overline{x} : \overline{\tau}.\, M)[\overline{\alpha}]\overline{x}] \Downarrow$ and $K[M'] \Downarrow \iff K[(\Lambda\overline{\alpha}.\, \lambda\overline{x} : \overline{\tau}.\, M')[\overline{\alpha}]\overline{x}] \Downarrow$. Hence $K[M] \Downarrow \iff K[M'] \Downarrow$. □

EXAMPLE 6.5. *We have* $x : \mathtt{int} \vdash x + 1 \overset{\text{std}}{\equiv} 1 + x : \mathtt{int}$*. That is,* $x + 1$ *and* $1 + x$ *are contextually equivalent (in the standard sense above) at type* $\mathtt{int}$ *provided that* x *has type* $\mathtt{int}$*. To show this, it suffices to prove* $\emptyset \vdash \lambda x : \mathtt{int}.\, x + 1 \sim \lambda x : \mathtt{int}.\, 1 + x : \mathtt{int} \to \mathtt{int}$*, which is trivial.*

EXAMPLE 6.6. *The packages*

$$\begin{aligned} M &= \mathtt{pack}\ \mathtt{real}, \langle y, \lambda x : \mathtt{real}.\, x\rangle\ \mathtt{as}\ \tau \\ M' &= \mathtt{pack}\ \mathtt{real}, \langle y + 1.0, \lambda x : \mathtt{real}.\, x - 1.0\rangle\ \mathtt{as}\ \tau \end{aligned}$$

are contextually equivalent (again in the standard sense above) at type

$$\tau = \exists\alpha.\, \alpha \times (\alpha \to \mathtt{real})$$

provided that y *has type* $\mathtt{real}$*. This follows from the bisimilarity of* $\lambda y : \mathtt{real}.\, M$ *and* $\lambda y : \mathtt{real}.\, M'$*, shown in Section 5.3.*

Although these examples are trivial, other examples of open terms can be treated in the same way, i.e., by means of closing abstractions.

7. EXTENSION FOR HIGHER-ORDER FUNCTIONS AND "UP-TO CONTEXT"

Although the contextual equivalence proof of higher-order functions in Section 5.5 was particularly simple, the trick used there (i.e., factoring out the common part $\lambda k : \sigma.\, k[\beta]\langle y, z\rangle$ of the two terms U, U' into context D) does not apply in general. For example, consider the following implementations of integer multisets with a higher-order function to compute a weighed sum of all elements. (We assume standard definitions of lists and binary trees.)

$$\begin{aligned}
\texttt{IntSet} &= \texttt{pack intList}, (\texttt{Nil}, \texttt{add}, \texttt{weigh}) \texttt{ as } \exists\alpha.\,\tau \\
\texttt{IntSet}' &= \texttt{pack intTree}, (\texttt{Leaf}, \texttt{add}', \texttt{weigh}') \texttt{ as } \exists\alpha.\,\tau \\
\tau &= \alpha \times (\texttt{int} \to \alpha \to \alpha) \times ((\texttt{int} \to \texttt{real}) \to \alpha \to \texttt{real}) \\
\texttt{add} &= \lambda i\,{:}\,\texttt{int}.\,\lambda s\,{:}\,\texttt{intList}.\,\texttt{Cons}(i, s) \\
\texttt{add}' &= \lambda i\,{:}\,\texttt{int}.\,\texttt{fix } f(s\,{:}\,\texttt{intTree})\,{:}\,\texttt{intTree} = \\
&\qquad \texttt{case } s \texttt{ of Leaf} \Rightarrow \texttt{Node}(i, \texttt{Leaf}, \texttt{Leaf}) \\
&\qquad [\!]\ \texttt{Node}(j, s_1, s_2) \Rightarrow \texttt{if } i < j \texttt{ then Node}(j, f s_1, s_2) \\
&\qquad\qquad\qquad\qquad\qquad\qquad \texttt{else Node}(j, s_1, f s_2) \\
\texttt{weigh} &= \lambda g\,{:}\,\texttt{int} \to \texttt{real}.\,\texttt{fix } f(s\,{:}\,\texttt{intList})\,{:}\,\texttt{real} = \\
&\qquad \texttt{case } s \texttt{ of Nil} \Rightarrow 0\ [\!]\ \texttt{Cons}(j, s_0) \Rightarrow gj + f s_0 \\
\texttt{weigh}' &= \lambda g\,{:}\,\texttt{int} \to \texttt{real}.\,\texttt{fix } f(s\,{:}\,\texttt{intTree})\,{:}\,\texttt{real} = \\
&\qquad \texttt{case } s \texttt{ of Leaf} \Rightarrow 0\ [\!]\ \texttt{Node}(j, s_1, s_2) \Rightarrow gj + f s_1 + f s_2
\end{aligned}$$

Unlike the previous example, these implementations have no syntactic similarity, which disables the simple proof. Instead, we have to put the whole packages into the bisimulation along with their elements. Then, by Condition 2 of bisimulation, we need at least to prove $\texttt{weigh}\ V\ W \Downarrow \iff \texttt{weigh}'\ V'\ W' \Downarrow$ for a certain class of V, W, V', and W'. In particular, V and V' can be of the forms $\lambda z\,{:}\,\texttt{int}.\,[\texttt{IntSet}/y]D$ and $\lambda z\,{:}\,\texttt{int}.\,[\texttt{IntSet}'/y]D$ for any D of appropriate type. Thus, because of the function application gj in $\texttt{weigh}$ and $\texttt{weigh}'$, we must prove

$$[\texttt{IntSet}, j/y, z]D \Downarrow \iff [\texttt{IntSet}', j/y, z]D \Downarrow$$

for every D (and j). We are stuck, however, since this subsumes the definition of $\texttt{IntSet} \equiv \texttt{IntSet}'$ and is *harder* to prove!

Resolving this problem requires weakening Condition 2. Recall that the fundamental properties of our bisimulations—that evaluation preserves $\sim^\circ$ and that $\sim^\circ$ respects convergence—were proved by structural induction on the evaluation derivations. Specifically, Condition 2 was used in sub-case $M_4 = x_{1_i}$ of case E-App in the proof of Lemma 4.3 (and 4.4). By further exploiting the induction hypothesis in this proof, we obtain the following weaker yet sound condition. (Difference from Condition 2 is shown in bold.)

(*2′*) Take any

$$(\texttt{fix } f(x\,{:}\,\pi)\,{:}\,\rho = M,\ \texttt{fix } f(x\,{:}\,\pi')\,{:}\,\rho' = M',\ \tau \to \sigma) \in \mathcal{R}$$

and any $(V, V', \tau) \in (\Delta, \mathcal{R})^\circ$. **Assume that, for any**

$$\begin{aligned}
N &\sqsubset (\texttt{fix } f(x\,{:}\,\pi)\,{:}\,\rho = M)V \\
N' &\sqsubset (\texttt{fix } f(x\,{:}\,\pi')\,{:}\,\rho' = M')V'
\end{aligned}$$

with $(N, N', \sigma) \in (\Delta_0, \mathcal{R}_0)^\circ$ **and** $(\Delta_0, \mathcal{R}_0) \in X$**, we have** $N \Downarrow \iff N' \Downarrow$**. Assume furthermore that, if** $N \Downarrow U$ **and** $N' \Downarrow U'$**, then** $(U, U', \sigma) \in (\Delta_1, \mathcal{R}_1)^\circ$ **for some** $\Delta_1 \supseteq \Delta_0$ **and** $\mathcal{R}_1 \supseteq \mathcal{R}_0$ **with** $(\Delta_1, \mathcal{R}_1) \in X$**.** Then, we have

$$\begin{aligned} & (\texttt{fix}\ f(x:\pi):\rho = M)V \Downarrow \\ \Longleftrightarrow\ & (\texttt{fix}\ f(x:\pi'):\rho' = M')V' \Downarrow. \end{aligned}$$

Furthermore, if $(\texttt{fix}\ f(x:\pi):\rho = M)V \Downarrow W$ and $(\texttt{fix}\ f(x:\pi'):\rho' = M')V \Downarrow W'$, then $(W, W', \sigma) \in (\Delta_2, \mathcal{R}_2)^\circ$ **for some $\Delta_2 \supseteq \Delta$ and $\mathcal{R}_2 \supseteq \mathcal{R}$ with $(\Delta_2, \mathcal{R}_2) \in X$.**

Here, $N_1 \sqsubset N_2$ means that, if $N_2 \Downarrow$, then $N_1 \Downarrow$ and the evaluation derivation for N_2 is strictly taller than that for N_1. (This is reminiscent of *indexed models* and *indexed logical relations* [Appel and McAllester 2001; Ahmed et al. 2003; Ahmed 2006]. However, we consider the height of derivation only in the case of higher-order functions, while they carry indices throughout the definitions—in particular for recursive types—and proofs.)

This generalization seems quite powerful: for instance, it allows us to conclude that gj in `weigh` and `weigh`$'$ gives the same result when g is substituted by V or V'. Although the condition above has X in a negative position ($(\Delta_1, \mathcal{R}_1) \in X$) and breaks the monotonicity property (the union of two bisimulations may no longer be a bisimulation), we still have soundness[4] and completeness ($\equiv$ is still a bisimulation because Condition 2$'$ is *weaker* than Condition 2). In fact, thanks to these soundness and completeness properties, $\equiv$ *is* the largest bisimulation.

Coincidentally, the last part of Condition 2$'$—that is, $(W, W', \sigma) \in (\Delta_2, \mathcal{R}_2)^\circ$ with $(\Delta_2, \mathcal{R}_2) \in X$, in place of $(\Delta, \mathcal{R} \cup \{(W, W', \sigma)\}) \in X$—also simplifies some bisimulation proofs for first-order functions. Intuitively, this is an "up-to context" technique: instead of $(W, W', \sigma) \in \mathcal{R}_2$ for some $(\Delta_2, \mathcal{R}_2) \in X$, one only needs to prove $(W, W', \sigma) \in (\Delta_2, \mathcal{R}_2)^\circ$, which reduces the size of X as in the following example:

Example 7.1. *Let $\tau = \mu\alpha.\,\texttt{unit} + \texttt{int} \times \alpha$. Then the function*

$$\begin{aligned} M \ = \ & \texttt{fix}\ f(x:\tau):\tau = \\ & \quad \texttt{case}\ \texttt{unfold}(x)\ \texttt{of}\ \texttt{in}_1(\_) \Rightarrow \texttt{fold}(\texttt{in}_1(\langle\rangle)) \\ & \quad [\![\ \texttt{in}_2(y) \Rightarrow \texttt{fold}(\texttt{in}_2(\#_1(y), f(\#_2(y)))) \end{aligned}$$

is bisimilar to the identity $M' = \lambda x:\tau.\,x$ at type $\tau \to \tau$. To prove this, take $X = \{(\emptyset, \mathcal{R})\}$ with $\mathcal{R} = \{(M, M', \tau \to \tau)\}$. To prove Condition 2$'$ (other conditions are trivial), let $(V, V', \tau) \in (\emptyset, \mathcal{R})^\circ$, i.e., $V = [M/z]D$ and $V' = [M'/z]D$ for $z:\tau \to \tau \vdash D : \tau$. By easy induction on the syntax of D, we have that D has no free variable, so $V = V' = D$ with $\vdash D : \tau$ in fact. Again by straightforward induction on the syntax of V, we have $MV \Downarrow V$ for any $\vdash V : \tau$. This concludes the proof since $(V, V, \tau) \in (\emptyset, \mathcal{R})^\circ$ by Definition 2.4.

If the up-to context technique were not available, we would have to include in $\mathcal{R}$ the identity relation at type τ, because of the original requirement in Condition 2 that

[4]The proofs of Lemma 4.3 and 4.4 remain valid just by replacing the phrase "by induction on the derivation" with "by induction on the height of the derivation." In fact, as stated above, Condition 2$'$ is *derived* by analyzing these proofs, so that it is sound by construction. See also Koutavas and Wand [2006b] for a more detailed presentation of this derivation.

$(V,V,\tau) \in R$. Worse, we would also have to include identities at $\mathtt{unit} + \mathtt{int} \times \tau$, $\mathtt{int} \times \tau$, $\mathtt{int}$ and $\mathtt{unit}$ because of other conditions of bisimulation for values of recursive, product, sum, and primitive types. "Up-to context" liberates us from these (easy but boring) burdens.

The following examples are similar to Example 7.1 in essence, but more non-trivial.

Example 7.2. *Let* $\tau = \mu\alpha.\,\mathtt{unit} \to \mathtt{int} \times \alpha$ *and*

$$\begin{aligned}
\mathtt{ones} &= \mathtt{fold}(\mathtt{fix}\ f(\_ : \mathtt{unit}) : \mathtt{int} \times \tau = \langle 1, \mathtt{fold}(f)\rangle)\\
\mathtt{twos} &= \mathtt{fold}(\mathtt{fix}\ f(\_ : \mathtt{unit}) : \mathtt{int} \times \tau = \langle 2, \mathtt{fold}(f)\rangle)\\
\mathtt{succ} &= \mathtt{fix}\ f(s : \tau) : \tau = \\
&\qquad \mathtt{let}\ c : \mathtt{int} \times \tau = \mathtt{unfold}(s)\langle\rangle\ \mathtt{in}\\
&\qquad\quad \mathtt{fold}(\lambda\_ : \mathtt{unit}.\,\langle 1 + \#_1(c),\ f(\#_2(c))\rangle).
\end{aligned}$$

Then $\mathtt{twos}$ *is bisimilar to* $\mathtt{twos}'$ *for* $\mathtt{succ}\ \mathtt{ones} \Downarrow \mathtt{twos}'$. *To prove this, take* $X = \{(\emptyset, \mathcal{R})\}$ *for* $\mathcal{R} = \{(\mathtt{twos}, \mathtt{twos}', \tau), (M, M', \mathtt{unit} \to \mathtt{int} \times \tau)\}$ *with*

$$\begin{aligned}
M &= \mathtt{fix}\ f(\_ : \mathtt{unit}) : \mathtt{int} \times \tau = \langle 2, \mathtt{fold}(f)\rangle\\
M' &= \lambda\_ : \mathtt{unit}.\,\langle 1 + \#_1(\langle 1, \mathtt{ones}\rangle),\ \mathtt{succ}(\#_2(\langle 1, \mathtt{ones}\rangle))\rangle.
\end{aligned}$$

Condition 7 (bisimulation for values of recursive types) is immediate from the definition of $\mathcal{R}$. *For Condition 2′, we apply M and M' to* $\langle\rangle$, *obtaining* $\langle 2, \mathtt{twos}\rangle$ *and* $\langle 2, \mathtt{twos}'\rangle$, *respectively. These values are again in* $(\emptyset, \mathcal{R})^\circ$, *which concludes the proof.*

Example 7.3. *Let* $\tau = \mu\alpha.\,\mathtt{unit} + (\alpha \to \alpha)$ *and*

$$\begin{aligned}
M = {} & \mathtt{fix}\ f(x : \tau) : \tau = \\
& \quad \mathtt{case}\ \mathtt{unfold}(x)\ \mathtt{of}\ \mathtt{in}_1(\_) \Rightarrow \mathtt{fold}(\mathtt{in}_1(\langle\rangle))\\
& \quad\ \ [\!]\ \mathtt{in}_2(g) \Rightarrow \mathtt{fold}(\mathtt{in}_2(\lambda y : \tau.\, f(g(fy)))).
\end{aligned}$$

We prove that M *is bisimilar to an identify function* $M' = \lambda x : \tau.\, x$. *Let* $X = \{(\emptyset, \mathcal{R})\}$, *where* $\mathcal{R}$ *is the smallest set such that* $(M, M', \tau \to \tau) \in \mathcal{R}$ *and* $(\lambda y : \tau.\, M(U(My)), U', \tau \to \tau) \in \mathcal{R}$ *for any* $(U, U', \tau \to \tau) \in (\emptyset, \mathcal{R})^\circ$.

First, by Condition 2′, we must apply M *and* M' *to* V *and* V', *respectively, for any* $(V, V', \tau) \in (\emptyset, \mathcal{R})^\circ$. *By an easy case analysis as in Example 7.1, we have either* $V = V' = \mathtt{fold}(\mathtt{in}_1(\langle\rangle))$, *or else* $V = \mathtt{fold}(\mathtt{in}_2(U))$ *and* $V' = \mathtt{fold}(\mathtt{in}_2(U'))$ *for* $(U, U', \tau \to \tau) \in (\emptyset, \mathcal{R})^\circ$. *The former case is trivial. In the latter case,* MV *and* $M'V'$ *evaluate to* $\mathtt{fold}(\mathtt{in}_2(\lambda y : \tau.\, M(U(My))))$ *and* $\mathtt{fold}(\mathtt{in}_2(U'))$, *respectively. We are done since these values are already in* $(\emptyset, \mathcal{R})^\circ$.

Second, again by Condition 2′, we must apply $\lambda y : \tau.\, M(U(My))$ *and* U' *to* V *and* V', *respectively, for any* $(U, U', \tau \to \tau), (V, V', \tau) \in (\emptyset, \mathcal{R})^\circ$. *Again, the non-trivial case is when* $V = \mathtt{fold}(\mathtt{in}_2(W))$ *and* $V' = \mathtt{fold}(\mathtt{in}_2(W'))$ *for* $(W, W', \tau \to \tau) \in (\emptyset, \mathcal{R})^\circ$. *Then,* $MV \Downarrow \mathtt{fold}(\mathtt{in}_2(\lambda y : \tau.\, M(W(My))))$. *Let this value be* V''. *Let furthermore* $U = \lambda x : \tau.\, N$ *and* $U' = \lambda x : \tau.\, N'$. *Then,* $(V'', V', \tau), (N, N', \tau) \in (\emptyset, \mathcal{R})^\circ$, *so* $[V''/x]N$ *and* $[V'/x]N'$ *are also in* $(\emptyset, \mathcal{R})^\circ$. *Thus, by the assumption of Condition 2′, we have* $[V''/x]N \Downarrow V_0$ *and* $[V'/x]N' \Downarrow V_0'$ *with* $(V_0, V_0', \tau) \in (\emptyset, \mathcal{R})^\circ$ *(or both evaluations diverge). The rest of the proof—that* MV_0 *and* V_0' *evaluate to values in* $(\emptyset, \mathcal{R})^\circ$*—is the same as the first case above.*

8. RELATED WORK

Variants of our bisimulation. Recently, Koutavas and Wand adapted our bisimulation to λ-calculus with ML-like references and store [Koutavas and Wand 2006b]. In addition, they gave a clearer account of our condition for higher-order functions in Section 7. They furthermore applied this approach to an untyped imperative object calculus [Koutavas and Wand 2006a] and a Java-like language with private members [Koutavas and Wand 2007]. These results give a new solution to the classical problem of proving contextual equivalence in such languages [Meyer and Sieber 1988; Pitts and Stark 1993; 1998]. They also showed that our bisimulation can be adapted for languages with small-step semantics [Koutavas and Wand 2007].

Sangiorgi, Kobayashi and Sumii [2007] developed various up-to techniques for our style of bisimulations (e.g., bisimulations that apply bisimilar functions to arguments of the forms $C[\overline{V}]$ and $C[\overline{V}']$ for bisimilar $\overline{V}$ and $\overline{V}'$), which simplifies contextual equivalence proofs of higher-order functions as well. They applied those techniques to pure call-by-name λ-calculus, imperative call-by-value λ-calculus, and pure higher-order π-calculus. Although their languages are untyped, it would be straightforward to combine their techniques into typed languages like ours, as their developments do not depend on the type system (or lack of it).

Semantic logical relations. Originally, logical relations were devised in denotational semantics for relating models of λ-calculus. Although they are indeed useful for this purpose (e.g., relating CPS semantics and direct-style semantics), they are not as useful for proving contextual equivalence or abstraction properties, for the following reasons. First, denotational semantics tend to require more complex mathematics (such as CPOs and categories) than operational semantics. Second, it is hard—though not impossible [Hughes 1997]—to define a model of polymorphic λ-calculus that preserves equivalence.

Logical relations for polymorphic λ-calculus are also useful for proving *parametricity* properties [Wadler 1989], e.g., that all functions of type $\forall\alpha.\,\alpha \to \alpha$ behave like the polymorphic identity function (or diverge, if there is recursion in the language). Our bisimulation can also derive such properties for existential types. For example, any package $p = \texttt{pack}\ \sigma, v\ \texttt{as}\ \tau$ of type $\tau = \exists\alpha.\,\alpha$ can be proved contextually equivalent to $p' = \texttt{pack unit}, \langle\rangle\ \texttt{as}\ \tau$, by taking $X = \{(\emptyset, \mathcal{R}_1), (\Delta, \mathcal{R}_2)\}$ where $\mathcal{R}_1 = \{(p, p', \tau)\}$, $\Delta = \{(\alpha, \sigma, \texttt{unit})\}$ and $\mathcal{R}_2 = \mathcal{R}_1 \cup \{(v, \langle\rangle, \alpha)\}$. However, for universal types, we would have to change Condition 3 to something like

(*3'*) Let $\Delta = \{(\alpha_1, \sigma_1, \sigma'_1), \ldots, (\alpha_m, \sigma_m, \sigma'_m)\}$. For each

$$(\Lambda\alpha.\,M, \Lambda\alpha.\,M', \forall\alpha.\,\tau) \in \mathcal{R}$$

and for any ρ **and** ρ' with $FTV(\rho) \subseteq dom(\Delta)$ **and** $FTV(\rho') \subseteq dom(\Delta)$, we have

$$(\Lambda\alpha.\,M)[[\overline{\sigma}/\overline{\alpha}]\rho] \Downarrow \iff (\Lambda\alpha.\,M')[[\overline{\sigma}'/\overline{\alpha}]\rho'] \Downarrow.$$

Furthermore, if $(\Lambda\alpha.\,M)[[\overline{\sigma}/\overline{\alpha}]\rho] \Downarrow W$ and $(\Lambda\alpha.\,M')[[\overline{\sigma}'/\overline{\alpha}]\rho'] \Downarrow W'$, then

$$(\Delta \cup \{(\alpha, [\overline{\sigma}/\overline{\alpha}]\rho, [\overline{\sigma}'/\overline{\alpha}]\rho')\}, \mathcal{R} \cup \mathcal{R}_1 \cup \{(W, W', \tau)\}) \in X$$

for any $\mathcal{R}_1 \subseteq \{([\overline{\sigma}/\overline{\alpha}]W_1, [\overline{\sigma}'/\overline{\alpha}]W'_1, \alpha) \mid \overline{\alpha} \vdash W_1 : \rho \wedge \overline{\alpha} \vdash W'_1 : \rho'\}$.

in order to keep track of all the values of type α, as in logical relations for universal types. We conjecture that soundness (and completeness) of such a condition would be just as difficult to establish as those for logical relations. In particular, reflexivity would be a main challenge (like the fundamental theorem for logical relations).

Syntactic logical relations. Pitts [2000] proposed *syntactic logical relations*, which use only the term model of polymorphic λ-calculus to prove contextual equivalence. He introduced the notion of $\top\top$-closure—based on a Galois connection between terms and contexts (for a term relation r, stack relation $r^\top$ denotes the set of pairs of stacks that behave equivalently for all the pairs of terms in r, and for a stack relation s, term relation $s^\top$ denotes the set of pairs of terms that behave equivalently for all the pairs of stacks in s)—in order to treat recursive functions without using denotational semantics. He proved that his syntactic logical relations are complete with respect to contextual equivalence in call-by-name polymorphic λ-calculus with recursive functions and universal types (and lists).

Pitts [2005] also proposed syntactic logical relations for call-by-value λ-calculus with recursive functions, universal types, and existential types. Although he showed (by a counter-example) that his proof principles based on logical relations are incomplete [Pitts 2005, Remark 7.7.4] and attributed the incompleteness to the presence of divergence, we have shown that a similar counter-example can be given without using divergence [Pitts 2005, Remark 7.7.7]. Recently, Derek Dreyer [personal communication, February 2006] suggested yet another counter-example

$$\begin{aligned}&\vdash (\texttt{pack bool}, \lambda f.\, f\,\texttt{true} \wedge \neg(f\,\texttt{false})\ \texttt{as}\ \tau)\\ &\quad \overset{?}{\equiv} (\texttt{pack int}, \lambda f.\, f\,1 \overset{\texttt{int}}{=} 1 \wedge f\,2 \overset{\texttt{int}}{=} 2 \wedge f\,3 \overset{\texttt{int}}{=} 3\ \texttt{as}\ \tau) : \tau\end{aligned}$$

where $\tau = \exists\alpha.\,(\alpha \to \alpha) \to \texttt{bool}$. The contextual equivalences in these examples are hard to prove, either with logical relations or with our bisimulations. Pitts gave a "brute force" proof of contextual equivalence for the first example [Pitts 2005, page 283, line 23], while no formal proof has been published for the other examples. (Although our bisimulations are complete in the sense that the bisimilarity coincides with generalized contextual equivalence, this does not mean an automatic proof for every instance of contextual equivalence, which is an undecidable problem.)

Birkedal and Harper [1999] and Crary and Harper [2007] extended syntactic logical relations with recursive *types* by proving certain unwinding properties. The latter work treated existential types via encoding into universal types (though this treatment is still incomplete and cannot prove the examples above [Crary and Harper 2007, Section 6.4]).

Melliès and Vouillon [2005] developed an operational model of λ-calculus with (semantic versions of) universal, existential, and recursive types, using a form of $\top\top$-closure (*biorthogonality*) and interpreting types as the limits of converging sequences of partial types (*interval types*). They also outlined relational versions of the model, which characterizes contextual equivalence of (syntactically) typed λ-terms. They leave state and concurrency to future work.

Applicative bisimulations. Abramsky [1990] proposed *applicative bisimulations* for proving contextual equivalence of untyped λ-terms. Gordon [1995a; 1995b] and Gordon and Rees [1996; 1995] adapted applicative bisimulations to calculi with objects, subtyping, universal polymorphism, and recursive types. As discussed in Section 1, however, these results do not apply to type abstraction using existential types. We solved this issue by considering sets of relations as bisimulations.

As a byproduct, it has become much *easier* to prove the soundness of our bisimulation (than the standard method for proving the soundness of applicative bisimulations [Howe 1996]). Technically, this simplification is due to the generalization for functions (Condition 2 in Definition 3.1), where we allow different arguments $C[\overline{V}]$ and $C[\overline{V}']$ while applicative bisimulation requires them to be the same.

Bisimulations for polymorphic π-calculi. Pierce and Sangiorgi [2000] developed a bisimulation proof technique for polymorphic π-calculus, using a separate environment for representing contexts' knowledge. In a sense, our bisimulation unifies the environmental knowledge with the bisimulation itself by generalizing the latter as a set of relations. Because of the imperative nature of π-calculus, their bisimulation is far from complete—in particular, aliasing of names is problematic.

Berger et al. [2003] defined two equivalence proof methods for linear π-calculi, one based on the syntactic logical relations of Pitts [2005; 2000] and the other based on the bisimulations of Pierce and Sangiorgi [2000]. Their main goal is to give a generic account for various features such as functions, state and concurrency by encoding them into appropriate versions of linear π-calculi. They proved soundness and completeness of their logical relations for one of the linear π-calculi, which directly corresponds to polymorphic λ-calculus (without recursion). They also proved full abstraction of the call-by-value and call-by-name encodings of the polymorphic λ-calculus to this version of linear π-calculus. However, for the other settings (e.g., with recursive functions or types), full abstraction of encodings and completeness of their logical relations are unclear. Completeness of their bisimulations is not discussed either. In addition, their developments are much heavier than ours for the purpose of just proving the equivalence of typed λ-terms.

Bisimulations for cryptographic calculi. Various bisimulations [Abadi and Gordon 1998; Abadi and Fournet 2001; Boreale et al. 2002; Borgström and Nestmann 2002] have been proposed for extensions of π-calculus with cryptographic primitives [Abadi and Gordon 1999; Abadi and Fournet 2001]. Their main idea is similar to Pierce and Sangiorgi's: using a separate environment to represent attackers' knowledge. In previous work [Sumii and Pierce 2004], we have applied our idea of using sets of relations as bisimulations to an extension of λ-calculus with perfect encryption (also known as dynamic sealing) and obtained a sound and complete characterization of contextual equivalence in this setting. Although this extension was untyped, it is straightforward to combine the present work with the previous one and obtain a bisimulation for typed λ-calculus with perfect encryption. The fact that our idea applies to such apparently different forms of information hiding as encryption and type abstraction might suggest that it is successful in capturing the essence of "information hiding" in programming languages and computation models.

9. CONCLUSION

We have presented a (characterization and) proof method for contextual equivalence of existential packages in λ-calculus with full recursive types, based on bisimulations generalized as sets of relations.

Although full automation is impossible because equivalence of λ-terms (with recursion) is undecidable, some mechanical support would be useful.

Another direction of future work is to extend the calculus with more complex features such as state (cf. [Pitts and Stark 1998; Bierman et al. 2000; Koutavas and Wand 2006b]). For example, it would be possible to treat state by passing around the state throughout the evaluation of terms and their bisimulation. More ambitiously, one could imagine generalizing this state-passing approach to more general "monadic bisimulation" by formalizing effects via monads [Moggi 1991].

Yet another possibility is to adopt our idea of "sets of relations as bisimulations" to other higher-order calculi with information hiding—such as higher-order π-calculus [Sangiorgi 1992], where restriction hides names and complicates equivalence—and compare the outcome with context bisimulation. Sangiorgi et al. [2007] gives a first result in this course.

Finally, as suggested in the previous section, the idea of considering sets of relations as bisimulations may be useful for other forms of information hiding such as secrecy typing [Heintze and Riecke 1998]. It would be interesting to see whether such an adaptation is indeed possible and, furthermore, to consider if these variations can be generalized into a unified theory of information hiding.

APPENDIX

A. PROOF OF LEMMA 4.1

The lemma to prove was: $\equiv \subseteq \sim$.

Since $\sim$ is the greatest bisimulation, it suffices to check that $\equiv$ is a bisimulation by checking each condition of bisimulation. Take any $(\Delta, \mathcal{R}) \in \equiv$ with $\Delta = \{(\overline{\alpha}, \overline{\sigma}, \overline{\sigma}')\}$. Then, from the definition of $\equiv$, we have:

A$_0$. $\vdash V_0 : [\overline{\sigma}/\overline{\alpha}]\tau_0$ and $\vdash V_0' : [\overline{\sigma}'/\overline{\alpha}]\tau_0$ for any $(V_0, V_0', \tau_0) \in \mathcal{R}$, and

B$_0$. $[\overline{V}_0/\overline{x}_0][\overline{\sigma}/\overline{\alpha}]C_0 \Downarrow \iff [\overline{V}'_0/\overline{x}_0][\overline{\sigma}'/\overline{\alpha}]C_0 \Downarrow$ for any $(\overline{V}_0, \overline{V}'_0, \overline{\tau}_0) \in \mathcal{R}$ and for any $\overline{\alpha}, \overline{x}_0 : \overline{\tau}_0 \vdash C_0 : \tau_0$.

We now check the conditions in the definition of bisimulation.

Condition 1: Immediate, since it is just the same as A$_0$.

Condition 2: Suppose that

$$((\texttt{fix}\ f(x : \pi) : \rho = M), (\texttt{fix}\ f(x : \pi') : \rho' = M'), \tau \rightarrow \sigma) \ \in\ \mathcal{R}$$

and that $V = [\overline{U}/\overline{y}][\overline{\sigma}/\overline{\alpha}]D$ and $V' = [\overline{U}'/\overline{y}][\overline{\sigma}'/\overline{\alpha}]D$ with $(\overline{U}, \overline{U}', \overline{\rho}) \in \mathcal{R}$ and $\overline{\alpha}, \overline{y} : \overline{\rho} \vdash D : \tau$. Then,

$$(\texttt{fix}\ f(x : \pi) : \rho = M)V \Downarrow \iff (\texttt{fix}\ f(x : \pi') : \rho' = M')V' \Downarrow$$

follows from B$_0$ by taking (for fresh g):

$$\begin{aligned} C_0 &= gD \\ \overline{x}_0 &= g, \overline{y} \end{aligned}$$

$$\begin{aligned}\overline{\tau}_0 &= \tau \to \sigma, \overline{\rho}\\ \overline{V}_0 &= (\texttt{fix}\ f(x{:}\pi){:}\rho = M), \overline{U}\\ \overline{V}'_0 &= (\texttt{fix}\ f(x{:}\pi'){:}\rho' = M'), \overline{U}'\end{aligned}$$

Suppose furthermore that $(\texttt{fix}\ f(x{:}\pi){:}\rho = M)V \Downarrow W$ and $(\texttt{fix}\ f(x{:}\pi'){:}\rho' = M')V' \Downarrow W'$. Then

$$(\Delta, \mathcal{R} \cup \{(W, W', \sigma)\}) \in \equiv$$

will follow from the definition of $\equiv$ if we can prove:

A$_2$. $\vdash V_2 : [\overline{\sigma}/\overline{\alpha}]\tau_2$ and $\vdash V'_2 : [\overline{\sigma}'/\overline{\alpha}]\tau_2$ for any $(V_2, V'_2, \tau_2) \in \mathcal{R} \cup \{(W, W', \sigma)\}$, and

B$_2$. $[\overline{V}_2/\overline{x}_2][\overline{\sigma}/\overline{\alpha}]C_2 \Downarrow \iff [\overline{V}'_2/\overline{x}_2][\overline{\sigma}'/\alpha]C_2 \Downarrow$ for any $(\overline{V}_2, \overline{V}'_2, \overline{\tau}_2) \in \mathcal{R} \cup \{(W, W', \sigma)\}$ and for any $\overline{\alpha}, \overline{x}_2 : \overline{\tau}_2 \vdash C_2 : \tau_2$.

But A$_2$ follows from A$_0$ in the case where (V_2, V'_2, τ_2) is drawn from $\mathcal{R}$ and from type preservation in the case where $(V_2, V'_2, \tau_2) = (W, W', \sigma)$. B$_2$ holds as follows: without loss of generality, let $(V_{2_1}, V'_{2_1}, \tau_{2_1}) = (W, W', \sigma)$ and $(V_{2_i}, V'_{2_i}, \tau_{2_i}) \in \mathcal{R}$ for $2 \le i \le n$; then, it suffices to take in B$_0$ (for fresh g)

$$\begin{aligned}C_0 &= \texttt{let}\ x_{2_1} = gD\ \texttt{in}\ C_2\\ \overline{x}_0 &= g, \overline{y}, x_{2_2}, \dots, x_{2_n}\\ \overline{\tau}_0 &= \tau \to \sigma, \overline{\rho}, \tau_{2_2}, \dots, \tau_{2_n}\\ \overline{V}_0 &= (\texttt{fix}\ f(x{:}\pi){:}\rho = M), \overline{U}, V_{2_2}, \dots, V_{2_n}\\ \overline{V}'_0 &= (\texttt{fix}\ f(x{:}\pi'){:}\rho' = M'), \overline{U}', V'_{2_2}, \dots, V'_{2_n}\end{aligned}$$

so that the evaluations of $[\overline{V}_0/\overline{x}_0][\overline{\sigma}/\overline{\alpha}]C_0$ and $[\overline{V}'_0/\overline{x}_0][\overline{\sigma}'/\overline{\alpha}]C_0$ amount to the evaluations of $[\overline{V}_2/\overline{x}_2][\overline{\sigma}/\overline{\alpha}]C_2$ and $[\overline{V}'_2/\overline{x}_2][\overline{\sigma}'/\overline{\alpha}]C_2$ as below.

$$\frac{\begin{array}{c}[(\texttt{fix}\ f(x{:}\pi){:}\rho = M), \overline{U}/g, \overline{y}][\overline{\sigma}/\overline{\alpha}](gD) \Downarrow W\\ [W, V_{2_2}, \dots, V_{2_n}/x_{2_1}, x_{2_2}, \dots, x_{2_n}][\overline{\sigma}/\overline{\alpha}]C_2 \Downarrow\end{array}}{[(\texttt{fix}\ f(x{:}\pi){:}\rho = M), \overline{U}, V_{2_2}, \dots, V_{2_n}/g, \overline{y}, x_{2_2}, \dots, x_{2_n}][\overline{\sigma}/\overline{\alpha}](\texttt{let}\ x_{2_1} = gD\ \texttt{in}\ C_2) \Downarrow}$$

$$\frac{\begin{array}{c}[(\texttt{fix}\ f(x{:}\pi'){:}\rho' = M'), \overline{U}'/g, \overline{y}][\overline{\sigma}'/\overline{\alpha}](gD) \Downarrow W'\\ [W', V'_{2_2}, \dots, V'_{2_n}/x_{2_1}, x_{2_2}, \dots, x_{2_n}][\overline{\sigma}'/\overline{\alpha}]C_2 \Downarrow\end{array}}{[(\texttt{fix}\ f(x{:}\pi'){:}\rho' = M'), \overline{U}', V'_{2_2}, \dots, V'_{2_n}/g, \overline{y}, x_{2_2}, \dots, x_{2_n}][\overline{\sigma}'/\overline{\alpha}](\texttt{let}\ x_{2_1} = gD\ \texttt{in}\ C_2) \Downarrow}$$

Condition 3: Suppose that

$$(\Lambda\alpha.\, M, \Lambda\alpha.\, M', \forall\alpha.\, \tau) \in \mathcal{R}$$

and that $FTV(\rho) \subseteq dom(\Delta)$. Then

$$(\Lambda\alpha.\, M)[[\overline{\sigma}/\overline{\alpha}]\rho] \Downarrow \iff (\Lambda\alpha.\, M')[[\overline{\sigma}'/\overline{\alpha}]\rho] \Downarrow$$

follows from B$_0$ by taking (for fresh g):

$$\begin{aligned}C_0 &= g[\rho]\\ \overline{x}_0 &= g\\ \overline{\tau}_0 &= \forall\alpha.\, \tau\end{aligned}$$

$$\begin{aligned}\overline{V}_0 &= \Lambda\alpha.\, M\\ \overline{V}'_0 &= \Lambda\alpha.\, M'\end{aligned}$$

Suppose furthermore that $(\Lambda\alpha.\, M)[[\overline{\sigma}/\overline{\alpha}]\rho] \Downarrow W$ and $(\Lambda\alpha.\, M')[[\overline{\sigma}'/\overline{\alpha}]\rho] \Downarrow W'$. Then

$$(\Delta, \mathcal{R} \cup \{(W, W', [\rho/\alpha]\tau)\}) \in \equiv$$

will follow from the definition of $\equiv$ if we can prove:

A$_3$. $\vdash V_3 : [\overline{\sigma}/\overline{\alpha}]\tau_3$ and $\vdash V'_3 : [\overline{\sigma}'/\overline{\alpha}]\tau_3$ for any $(V_3, V'_3, \tau_3) \in \mathcal{R} \cup \{(W, W', [\rho/\alpha]\tau)\}$, and

B$_3$. $[\overline{V}_3/\overline{x}_3][\overline{\sigma}/\overline{\alpha}]C_3 \Downarrow \iff [\overline{V}'_3/\overline{x}_3][\overline{\sigma}'/\overline{\alpha}]C_3 \Downarrow$ for any $(\overline{V}_3, \overline{V}'_3, \overline{\tau}_3) \in \mathcal{R} \cup \{(W, W', [\rho/\alpha]\tau)\}$ and for any $\overline{\alpha}, \overline{x}_3 : \overline{\tau}_3 \vdash C_3 : \tau_3$.

But A$_3$ follows from A$_0$ in the case where (V_3, V'_3, τ_3) is drawn from $\mathcal{R}$ and from type preservation in the case where $(V_3, V'_3, \tau_3) = (W, W', [\rho/\alpha]\tau)$. B$_3$ holds as follows: without loss of generality, let $(V_{3_1}, V'_{3_1}, \tau_{3_1}) = (W, W', [\rho/\alpha]\tau)$ and $(V_{3_i}, V'_{3_i}, \tau_{3_i}) \in \mathcal{R}$ for $2 \leq i \leq n$; then, it suffices to take in B$_0$ (for fresh g)

$$\begin{aligned}C_0 &= \texttt{let } x_{3_1} = g[\rho] \texttt{ in } C_3\\ \overline{x}_0 &= g, x_{3_2}, \ldots, x_{3_n}\\ \overline{\tau}_0 &= \forall\alpha.\, \tau, \tau_{3_2}, \ldots, \tau_{3_n}\\ \overline{V}_0 &= \Lambda\alpha.\, M, V_{3_2}, \ldots, V_{3_n}\\ \overline{V}'_0 &= \Lambda\alpha.\, M', V'_{3_2}, \ldots, V'_{3_n}\end{aligned}$$

so that the evaluations of $[\overline{V}_0/\overline{x}_0][\overline{\sigma}/\overline{\alpha}]C_0$ and $[\overline{V}'_0/\overline{x}_0][\overline{\sigma}'/\overline{\alpha}]C_0$ amount to the evaluations of $[\overline{V}_3/\overline{x}_3][\overline{\sigma}/\overline{\alpha}]C_3$ and $[\overline{V}'_3/\overline{x}_3][\overline{\sigma}'/\overline{\alpha}]C_3$ as below.

$$\frac{\begin{array}{c}[\Lambda\alpha.\, M/g][\overline{\sigma}/\overline{\alpha}](g[\rho]) \Downarrow W\\ [W, V_{3_2}, \ldots, V_{3_n}/x_{3_1}, x_{3_2}, \ldots, x_{3_n}][\overline{\sigma}/\overline{\alpha}]C_3 \Downarrow\end{array}}{[\Lambda\alpha.\, M, V_{3_2}, \ldots, V_{3_n}/g, x_{3_2}, \ldots, x_{3_n}][\overline{\sigma}/\overline{\alpha}](\texttt{let } x_{3_1} = g[\rho] \texttt{ in } C_3) \Downarrow}$$

$$\frac{\begin{array}{c}[\Lambda\alpha.\, M'/g][\overline{\sigma}'/\overline{\alpha}](g[\rho]) \Downarrow W'\\ [W', V'_{3_2}, \ldots, V'_{3_n}/x_{3_1}, x_{3_2}, \ldots, x_{3_n}][\overline{\sigma}'/\overline{\alpha}]C_3 \Downarrow\end{array}}{[\Lambda\alpha.\, M', V'_{3_2}, \ldots, V'_{3_n}/g, x_{3_2}, \ldots, x_{3_n}][\overline{\sigma}'/\overline{\alpha}](\texttt{let } x_{3_1} = g[\rho] \texttt{ in } C_3) \Downarrow}$$

Condition 4: Suppose that

$$((\texttt{pack } \sigma, V \texttt{ as } \exists\alpha.\, \tau), (\texttt{pack } \sigma', V' \texttt{ as } \exists\alpha.\, \tau'), \exists\alpha.\, \tau'') \in \mathcal{R}.$$

Then,

$$(\Delta \uplus \{(\alpha, \sigma, \sigma')\}, \mathcal{R} \cup \{(V, V', \tau'')\}) \in \equiv$$

follows from the definition of $\equiv$ if we prove:

A$_4$. $\vdash V_4 : [\overline{\sigma}, \sigma/\overline{\alpha}, \alpha]\tau_4$ and $\vdash V'_4 : [\overline{\sigma}', \sigma'/\overline{\alpha}, \alpha]\tau_4$ for any $(V_4, V'_4, \tau_4) \in \mathcal{R} \cup \{(V, V', \tau'')\}$, and

B$_4$. $[\overline{V}_4/\overline{x}_4][\overline{\sigma}, \sigma/\overline{\alpha}, \alpha]C_4 \Downarrow \iff [\overline{V}'_4/\overline{x}_4][\overline{\sigma}', \sigma/\overline{\alpha}, \alpha]C_4 \Downarrow$ for any $(\overline{V}_4, \overline{V}'_4, \overline{\tau}_4) \in \mathcal{R} \cup \{(V, V', \tau'')\}$ and for any $\overline{\alpha}, \alpha, \overline{x}_4 : \overline{\tau}_4 \vdash C_4 : \tau_4$.

But A_4 follows from A_0 in the case where (V_4, V_4', τ_4) is drawn from $\mathcal{R}$ and, in the case where $(V_4, V_4', \tau_4) = (V, V', \tau'')$, by inversion of (T-Pack) with

$$\vdash \texttt{pack}\ \sigma, V\ \texttt{as}\ \exists\alpha.\tau\ :\ [\overline{\sigma}/\overline{\alpha}](\exists\alpha.\tau'')$$

and

$$\vdash \texttt{pack}\ \sigma', V'\ \texttt{as}\ \exists\alpha.\tau'\ :\ [\overline{\sigma}'/\overline{\alpha}](\exists\alpha.\tau''),$$

which follow from A_0 with

$$((\texttt{pack}\ \sigma, V\ \texttt{as}\ \exists\alpha.\tau), (\texttt{pack}\ \sigma', V'\ \texttt{as}\ \exists\alpha.\tau'), \exists\alpha.\tau'')\ \in\ \mathcal{R}.$$

B_4 holds as follows: without loss of generality, let $(V_{4_1}, V_{4_1}', \tau_{4_1}) = (V, V', \tau'')$ and $(V_{4_i}, V_{4_i}', \tau_{4_i}) \in \mathcal{R}$ for $2 \le i \le n$; then, it suffices to take in B_0 (for fresh p)

$$\begin{aligned}
C_0 &= \texttt{open}\ p\ \texttt{as}\ \alpha, x_{4_1}\ \texttt{in}\ C_4 \\
\overline{x}_0 &= p, x_{4_2}, \ldots, x_{4_n} \\
\overline{\tau}_0 &= \exists\alpha.\tau'', \tau_{4_2}, \ldots, \tau_{4_n} \\
\overline{V}_0 &= \texttt{pack}\ \sigma, V\ \texttt{as}\ \exists\alpha.\tau, V_{4_2}, \ldots, V_{4_n} \\
\overline{V}_0' &= \texttt{pack}\ \sigma', V'\ \texttt{as}\ \exists\alpha.\tau', V_{4_2}', \ldots, V_{4_n}'
\end{aligned}$$

so that the evaluations of $[\overline{V}_0/\overline{x}_0][\overline{\sigma}/\overline{\alpha}]C_0$ and $[\overline{V}_0'/\overline{x}_0][\overline{\sigma}'/\overline{\alpha}]C_0$ amount to the evaluations of $[\overline{V}_4/\overline{x}_4][\overline{\sigma}, \sigma/\overline{\alpha}, \alpha]C_4$ and $[\overline{V}_4'/\overline{x}_4][\overline{\sigma}', \sigma'/\overline{\alpha}, \alpha]C_4$ as below.

$$\frac{[V, V_{4_2}, \ldots, V_{4_n}/x_{4_1}, x_{4_2}, \ldots, x_{4_n}][\overline{\sigma}, \sigma/\overline{\alpha}, \alpha]C_4 \Downarrow}{[\texttt{pack}\ \sigma, V\ \texttt{as}\ \exists\alpha.\tau, V_{4_2}, \ldots, V_{4_n}/p, x_{4_2}, \ldots, x_{4_n}][\overline{\sigma}/\overline{\alpha}](\texttt{open}\ p\ \texttt{as}\ \alpha, x_{4_1}\ \texttt{in}\ C_4) \Downarrow}$$

$$\frac{[V', V_{4_2}', \ldots, V_{4_n}'/x_{4_1}, x_{4_2}, \ldots, x_{4_n}][\overline{\sigma}', \sigma'/\overline{\alpha}, \alpha]C_4 \Downarrow}{[\texttt{pack}\ \sigma', V'\ \texttt{as}\ \exists\alpha.\tau, V_{4_2}', \ldots, V_{4_n}'/p, x_{4_2}, \ldots, x_{4_n}][\overline{\sigma}'/\overline{\alpha}](\texttt{open}\ p\ \texttt{as}\ \alpha, x_{4_1}\ \texttt{in}\ C_4) \Downarrow}$$

Proofs of the other conditions are similar. □

B. PROOF OF LEMMA 4.3

The lemma to prove was: Suppose $\Delta_0 \vdash N \sim^{\circ}_{\mathcal{R}_0} N' : \tau$. If $N \Downarrow W$ and $N' \Downarrow W'$, then $\Delta \vdash W \sim^{\circ}_{\mathcal{R}} W' : \tau$ for some $\Delta \supseteq \Delta_0$ and $\mathcal{R} \supseteq \mathcal{R}_0$.

The proof is by induction on the derivation of $N \Downarrow W$.

By the definition of $\sim^{\circ}$, we have

$$N\ =\ [\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0]M_0$$

and

$$N'\ =\ [\overline{V}_0'/\overline{x}_0][\overline{\sigma}_0'/\overline{\alpha}_0]M_0$$

for some

$$\Delta_0\ \vdash\ \overline{V}_0\ \sim_{\mathcal{R}_0}\ \overline{V}_0'\ :\ \overline{\tau}_0$$

and

$$\overline{\alpha}_0, \overline{x}_0 : \overline{\tau}_0\ \vdash\ M_0\ :\ \tau$$

with $\Delta_0 = \{(\overline{\alpha}_0, \overline{\sigma}_0, \overline{\sigma}_0')\}$. If N is a value, then N' is also a value (easy case analysis on the syntax of M_0) and the result is immediate, because every value evaluates only to itself. We consider the remaining possibilities—where M_0 is neither a value nor a variable—in detail; there is one case for each of the non-value evaluation rules.

Case (E-Open). Then M_0 has the form

$$M_0 \;=\; \texttt{open}\ M_1\ \texttt{as}\ \alpha, y\ \texttt{in}\ M_2$$

and the given evaluation derivations have the forms:

$$\frac{\begin{array}{c}[\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0]M_1 \Downarrow \texttt{pack}\ \rho_1, W_1\ \texttt{as}\ \exists\alpha.\,\rho_2 \\ [W_1/y][\rho_1/\alpha][\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0]M_2 \Downarrow W\end{array}}{[\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0](\texttt{open}\ M_1\ \texttt{as}\ \alpha, y\ \texttt{in}\ M_2) \Downarrow W}$$

$$\frac{\begin{array}{c}[\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}_0]M_1 \Downarrow \texttt{pack}\ \rho'_1, W'_1\ \texttt{as}\ \exists\alpha.\,\rho'_2 \\ [W'_1/y][\rho'_1/\alpha][\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}_0]M_2 \Downarrow W'\end{array}}{[\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}_0](\texttt{open}\ M_1\ \texttt{as}\ \alpha, y\ \texttt{in}\ M_2) \Downarrow W'}$$

By inversion of (T-Open), we have

$$\overline{\alpha}_0, \overline{x}_0 : \overline{\tau}_0 \;\vdash\; M_1 \;:\; \exists\alpha.\,\rho''_2$$

and

$$\overline{\alpha}_0, \overline{x}_0 : \overline{\tau}_0, \alpha, y : \rho''_2 \;\vdash\; M_2 \;:\; \tau$$

with $\alpha \notin FTV(\tau)$. Thus, by the definition of $\sim^\circ$, we have

$$\Delta_0 \;\vdash\; [\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}]M_1 \;\sim^\circ_{\mathcal{R}_0}\; [\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}]M_1 \;:\; \exists\alpha.\,\rho''_2.$$

Therefore, by the induction hypothesis, we have

$$\Delta_1 \;\vdash\; \texttt{pack}\ \rho_1, W_1\ \texttt{as}\ \exists\alpha.\,\rho_2 \;\sim^\circ_{\mathcal{R}_1}\; \texttt{pack}\ \rho'_1, W'_1\ \texttt{as}\ \exists\alpha.\,\rho'_2 \;:\; \exists\alpha.\,\rho''_2$$

for some $\Delta_1 \supseteq \Delta_0$ and $\mathcal{R}_1 \supseteq \mathcal{R}_0$. Then, by the definition of $\sim^\circ$, we have

$$\texttt{pack}\ \rho_1, W_1\ \texttt{as}\ \exists\alpha.\,\rho_2 \;=\; [\overline{V}_1/\overline{x}_1][\overline{\sigma}_1/\overline{\alpha}_1]M_3$$

and

$$\texttt{pack}\ \rho'_1, W'_1\ \texttt{as}\ \exists\alpha.\,\rho'_2 \;=\; [\overline{V}'_1/\overline{x}_1][\overline{\sigma}'_1/\overline{\alpha}_1]M_3$$

for some

$$\Delta_1 \;\vdash\; \overline{V}_1 \;\sim_{\mathcal{R}_1}\; \overline{V}'_1 \;:\; \overline{\tau}_1$$

and

$$\overline{\alpha}_1, \overline{x}_1 : \overline{\tau}_1 \;\vdash\; M_3 \;:\; \exists\alpha.\,\rho''_2$$

with $\Delta_1 = \{(\overline{\alpha}_1, \overline{\sigma}_1, \overline{\sigma}'_1)\}$.

Sub-case $M_3 = (\texttt{pack}\ \rho''_1, M_4\ \texttt{as}\ \exists\alpha.\,\rho''_2)$. Then

$$\begin{aligned} W_1 \;&=\; [\overline{V}_1/\overline{x}_1][\overline{\sigma}_1/\overline{\alpha}_1]M_4 \\ W'_1 \;&=\; [\overline{V}'_1/\overline{x}_1][\overline{\sigma}'_1/\alpha_1]M_4 \end{aligned}$$

and

$$\begin{aligned} \rho_1 \;&=\; [\overline{\sigma}_1/\overline{\alpha}_1]\rho''_1 \\ \rho'_1 \;&=\; [\overline{\sigma}'_1/\overline{\alpha}_1]\rho''_1. \end{aligned}$$

Since we have

$$\overline{\alpha}_1, \overline{x}_1 : \overline{\tau}_1 \vdash M_4 : [\rho_1''/\alpha]\rho_2''$$

by inversion of (T-Pack), we have

$$\overline{\alpha}_1, \overline{x}_0 : \overline{\tau}_0, \overline{x}_1 : \overline{\tau}_1 \vdash [M_4/y][\rho_1''/\alpha]M_2 : \tau$$

by weakening and the substitution lemmas for types and terms. Therefore, by the definition of $\sim^\circ$, we have

$$\Delta_1 \vdash [\overline{V}_0, \overline{V}_1/\overline{x}_0, \overline{x}_1][\overline{\sigma}_1/\overline{\alpha}_1][M_4/y][\rho_1''/\alpha]M_2 \sim^\circ_{\mathcal{R}_1} [\overline{V}'_0, \overline{V}_1/\overline{x}_0, \overline{x}_1][\overline{\sigma}'_1/\overline{\alpha}_1][M_4/y][\rho_1''/\alpha]M_2 : \tau.$$

Since

$$\begin{aligned}
& [\overline{V}_0, \overline{V}_1/\overline{x}_0, \overline{x}_1][\overline{\sigma}_1/\overline{\alpha}_1][M_4/y][\rho_1''/\alpha]M_2 \\
= \ & [(([\overline{V}_1/\overline{x}_1][\overline{\sigma}_1/\overline{\alpha}_1]M_4)/y][(([\overline{\sigma}_1/\overline{\alpha}_1]\rho_1'')/\alpha][\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0]M_2 \\
= \ & [W_1/y][\rho_1/\alpha][\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0]M_2
\end{aligned}$$

and

$$\begin{aligned}
& [\overline{V}'_0, \overline{V}'_1/\overline{x}_0, \overline{x}_1][\overline{\sigma}'_1/\overline{\alpha}_1][M_4/y][\rho_1''/\alpha]M_2 \\
= \ & [(([\overline{V}'_1/\overline{x}_1][\overline{\sigma}'_1/\overline{\alpha}_1]M_4)/y][(([\overline{\sigma}'_1/\overline{\alpha}_1]\rho_1'')/\alpha][\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}_0]M_2 \\
= \ & [W'_1/y][\rho'_1/\alpha][\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}_0]M_2,
\end{aligned}$$

we have

$$\Delta_2 \vdash W \sim^\circ_{\mathcal{R}_2} W' : \tau$$

for some $\Delta_2 \supseteq \Delta_1$ and $\mathcal{R}_2 \supseteq \mathcal{R}_1$ by the induction hypothesis.

Sub-case $M_3 = x_{1_i}$**.** Then

$$\begin{aligned}
V_{1_i} &= \texttt{pack } \rho_1, W_1 \texttt{ as } \exists\alpha.\, \rho_2 \\
V'_{1_i} &= \texttt{pack } \rho'_1, W'_1 \texttt{ as } \exists\alpha.\, \rho'_2.
\end{aligned}$$

Since

$$\Delta_1 \vdash V_{1_i} \sim_{\mathcal{R}_1} V'_{1_i} : \tau_{1_i},$$

we have the following two possibilities by Condition 4 of bisimulation.

Sub-sub-case $(\beta, \rho_1, \rho'_1) \in \Delta_1$ **and** $(W_1, W'_1, [\beta/\alpha]\rho_2'') \in \mathcal{R}_1$**.** Since we have

$$\overline{\alpha}_0, \overline{x}_0 : \overline{\tau}_0, \alpha, y : \rho_2'' \vdash M_2 : \tau$$

with $\alpha \notin FTV(\tau)$, we have

$$\overline{\alpha}_0, \overline{x}_0 : \overline{\tau}_0, \beta, y : [\beta/\alpha]\rho_2'' \vdash [\beta/\alpha]M_2 : \tau.$$

Then, we have

$$\Delta_1 \vdash [\overline{V}_0, W_1/\overline{x}_0, y][\overline{\sigma}_1/\overline{\alpha}_1][\beta/\alpha]M_2 \sim^\circ_{\mathcal{R}_1} [\overline{V}'_0, W'_1/\overline{x}_0, y][\overline{\sigma}'_1/\overline{\alpha}_1][\beta/\alpha]M_2 : \tau$$

by the definition of $\sim^\circ$. Since we have $\beta = \alpha_{1_i}$ with $\rho_1 = \sigma_{1_i}$ and $\rho'_1 = \sigma'_{1_i}$ for some i, we have

$$[\overline{V}_0, W_1/\overline{x}_0, y][\overline{\sigma}_1/\overline{\alpha}_1][\beta/\alpha]M_2 = [W_1/y][\rho_1/\alpha][\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0]M_2$$

and

$$[\overline{V}'_0, W'_1/\overline{x}_0, y][\overline{\sigma}'_1/\overline{\alpha}_1][\beta/\alpha]M_2 \;=\; [W'_1/y][\rho'_1/\alpha][\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}_0]M_2,$$

so we have

$$\Delta_2 \;\vdash\; W \;\sim^{\circ}_{\mathcal{R}_2} W' \;:\; \tau$$

for some $\Delta_2 \supseteq \Delta_1$ and $\mathcal{R}_2 \supseteq \mathcal{R}_1$ by the induction hypothesis.

Sub-sub-case $(\Delta_1 \uplus \{(\alpha, \rho_1, \rho'_1)\}, \mathcal{R}_1 \cup \{(W_1, W'_1, \rho''_2)\}) \in \sim$**.** Then we have

$$\Delta_2 \;\vdash\; [\overline{V}_0, W_1/\overline{x}_0, y][\overline{\sigma}_0, \rho_1/\overline{\alpha}_0, \alpha]M_2 \;\sim^{\circ}_{\mathcal{R}_2}\; [\overline{V}'_0, W'_1/\overline{x}_0, y][\overline{\sigma}'_0, \rho'_1/\overline{\alpha}_0, \alpha]M_2 \;:\; \tau$$

for $\Delta_2 = \Delta_1 \cup \{(\alpha, \rho_1, \rho'_1)\}$ and $\mathcal{R}_2 = \mathcal{R}_1 \cup \{(W_1, W'_1, \rho''_2)\}$ by the definition of $\sim^{\circ}$. Since we have

$$[\overline{V}_0, W_1/\overline{x}_0, y][\overline{\sigma}_0, \rho_1/\overline{\alpha}_0, \alpha]M_2 \;=\; [W_1/y][\rho_1/\alpha][\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0]M_2$$

and

$$[\overline{V}'_0, W'_1/\overline{x}_0, y][\overline{\sigma}'_0, \rho'_1/\overline{\alpha}_0, \alpha]M_2 \;=\; [W'_1/y][\rho'_1/\alpha][\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}_0]M_2,$$

we have

$$\Delta_3 \;\vdash\; W \;\sim^{\circ}_{\mathcal{R}_3}\; W' \;:\; \tau$$

for some $\Delta_3 \supseteq \Delta_2$ and $\mathcal{R}_3 \supseteq \mathcal{R}_2$ by the induction hypothesis.

Case (E-Pack). Then M_0 has the form

$$M_0 \;=\; \texttt{pack}\ \rho_1, M_1\ \texttt{as}\ \exists\alpha.\,\rho_2$$

and the given evaluation derivations have the forms

$$\frac{[\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0]M_1 \Downarrow W_1}{[\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0](\texttt{pack}\ \rho_1, M_1\ \texttt{as}\ \exists\alpha.\,\rho_2) \Downarrow \texttt{pack}\ [\overline{\sigma}_0/\overline{\alpha}_0]\rho_1, W_1\ \texttt{as}\ [\overline{\sigma}_0/\overline{\alpha}_0](\exists\alpha.\,\rho_2)}$$

$$\frac{[\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}_0]M_1 \Downarrow W'_1}{[\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}_0](\texttt{pack}\ \rho_1, M_1\ \texttt{as}\ \exists\alpha.\,\rho_2) \Downarrow \texttt{pack}\ [\overline{\sigma}'_0/\overline{\alpha}_0]\rho_1, W'_1\ \texttt{as}\ [\overline{\sigma}'_0/\overline{\alpha}_0](\exists\alpha.\,\rho_2)}$$

where

$$W \;=\; \texttt{pack}\ [\overline{\sigma}_0/\overline{\alpha}_0]\rho_1, W_1\ \texttt{as}\ [\overline{\sigma}_0/\overline{\alpha}_0](\exists\alpha.\,\rho_2)$$

and

$$W' \;=\; \texttt{pack}\ [\overline{\sigma}'_0/\overline{\alpha}_0]\rho_1, W'_1\ \texttt{as}\ [\overline{\sigma}'_0/\overline{\alpha}_0](\exists\alpha.\,\rho_2).$$

By inversion of (T-Pack), we have

$$\overline{\alpha}_0, \overline{x}_0 : \overline{\tau}_0 \;\vdash\; M_1 \;:\; [\rho_1/\alpha]\rho_2$$

with $\tau = \exists\alpha.\,\rho_2$. Thus, by the definition of $\sim^{\circ}$, we have

$$\Delta_0 \;\vdash\; [\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0]M_1 \;\sim^{\circ}_{\mathcal{R}_0}\; [\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}_0]M_1 \;:\; [\rho_1/\alpha]\rho_2.$$

Then, by the induction hypothesis, we have

$$\Delta_1 \;\vdash\; W_1 \;\sim^{\circ}_{\mathcal{R}_1}\; W'_1 \;:\; [\rho_1/\alpha]\rho_2$$

for some $\Delta_1 \supseteq \Delta_0$ and $\mathcal{R}_1 \supseteq \mathcal{R}_0$. Therefore, by the definition of $\sim^\circ$, we have

$$\begin{aligned} W_1 &= [\overline{V}_1/\overline{x}_1][\overline{\sigma}_1/\overline{\alpha}_1]M_2 \\ W_1' &= [\overline{V}_1'/\overline{x}_1][\overline{\sigma}_1'/\overline{\alpha}_1]M_2 \end{aligned}$$

for some

$$\Delta_1 \vdash \overline{V}_1 \sim_{\mathcal{R}_1} \overline{V}_1' : \overline{\tau}_1$$

and

$$\overline{\alpha}_1, \overline{x}_1 : \overline{\tau}_1 \vdash M_2 : [\rho_1/\alpha]\rho_2$$

with $\Delta_1 = \{(\overline{\alpha}_1, \overline{\sigma}_1, \overline{\sigma}_1')\}$. Thus, by (T-Pack), we have

$$\overline{\alpha}_1, \overline{x}_1 : \overline{\tau}_1 \vdash M_3 : \exists\alpha.\,\rho_2$$

for

$$M_3 = \texttt{pack}\ \rho_1, M_2\ \texttt{as}\ \exists\alpha.\,\rho_2.$$

Then, by the definition of $\sim^\circ$, we have

$$\Delta_1 \vdash [\overline{V}_1/\overline{x}_1][\overline{\sigma}_1/\overline{\alpha}_1]M_3 \sim^\circ_{\mathcal{R}_1} [\overline{V}_1'/\overline{x}_1][\overline{\sigma}_1'/\overline{\alpha}_1]M_3 : \exists\alpha.\,\rho_2,$$

i.e.,

$$\Delta_1 \vdash W \sim^\circ_{\mathcal{R}_1} W' : \tau.$$

Case (E-TApp). Then M_0 has the form

$$M_0 = M_1[\rho_1]$$

and the given evaluation derivations have the forms:

$$\frac{\begin{array}{c}[\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0]M_1 \Downarrow \Lambda\alpha.\,M_2 \\ [([\overline{\sigma}_0/\overline{\alpha}_0]\rho_1)/\alpha]M_2 \Downarrow W\end{array}}{[\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0](M_1[\rho_1]) \Downarrow W}$$

$$\frac{\begin{array}{c}[\overline{V}_0'/\overline{x}_0][\overline{\sigma}_0'/\overline{\alpha}_0]M_1 \Downarrow \Lambda\alpha.\,M_2' \\ [([\overline{\sigma}_0'/\overline{\alpha}_0]\rho_1)/\alpha]M_2' \Downarrow W'\end{array}}{[\overline{V}_0'/\overline{x}_0][\overline{\sigma}_0'/\overline{\alpha}_0](M_1[\rho_1]) \Downarrow W'}$$

By inversion of (T-TApp), we have

$$\overline{\alpha}_0, \overline{x}_0 : \overline{\tau}_0 \vdash M_1 : \forall\alpha.\,\rho_2$$

with $\tau = [\rho_1/\alpha]\rho_2$. Thus, by the definition of $\sim^\circ$, we have

$$\Delta_0 \vdash [\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0]M_1 \sim^\circ_{\mathcal{R}_0} [\overline{V}_0'/\overline{x}_0][\overline{\sigma}_0'/\overline{\alpha}_0]M_1 : \forall\alpha.\,\rho_2.$$

Then, by the induction hypothesis, we have

$$\Delta_1 \vdash \Lambda\alpha.\,M_2 \sim^\circ_{\mathcal{R}_1} \Lambda\alpha.\,M_2' : \forall\alpha.\,\rho_2$$

for some $\Delta_1 \supseteq \Delta_0$ and $\mathcal{R}_1 \supseteq \mathcal{R}_0$. Therefore, by the definition of $\sim^\circ$, we have

$$\begin{aligned}\Lambda\alpha.\,M_2 &= [\overline{V}_1/\overline{x}_1][\overline{\sigma}_1/\overline{\alpha}_1]M_3\\ \Lambda\alpha.\,M_2' &= [\overline{V}_1'/\overline{x}_1][\overline{\sigma}_1'/\overline{\alpha}_1]M_3\end{aligned}$$

for some

$$\Delta_1 \vdash \overline{V}_1 \sim_{\mathcal{R}_1} \overline{V}_1' : \overline{\tau}_1$$

and

$$\overline{\alpha}_1, \overline{x}_1 : \overline{\tau}_1 \vdash M_3 : \forall\alpha.\,\rho_2$$

with $\Delta_1 = \{(\overline{\alpha}_1, \overline{\sigma}_1, \overline{\sigma}_1')\}$.

Sub-case $M_3 = \Lambda\alpha.\,M_2''$. Then, by inversion of (T-TAbs), we have

$$\overline{\alpha}_1, \overline{x}_1 : \overline{\tau}_1, \alpha \vdash M_2'' : \rho_2.$$

By the substitution lemma for types, we have

$$\overline{\alpha}_1, \overline{x}_1 : \overline{\tau}_1 \vdash [\rho_1/\alpha]M_2'' : [\rho_1/\alpha]\rho_2.$$

By the definition of $\sim^\circ$, we have

$$\Delta_1 \vdash [\overline{V}_1/\overline{x}_1][\overline{\sigma}_1/\overline{\alpha}_1][\rho_1/\alpha]M_2'' \sim^\circ_{\mathcal{R}_1} [\overline{V}_1'/\overline{x}_1][\overline{\sigma}_1'/\overline{\alpha}_1][\rho_1/\alpha]M_2'' : [\rho_1/\alpha]\rho_2,$$

i.e.,

$$\Delta_1 \vdash [([\overline{\sigma}_0/\overline{\alpha}_0]\rho_1)/\alpha]M_2 \sim^\circ_{\mathcal{R}_1} [([\overline{\sigma}_0'/\overline{\alpha}_0]\rho_1)/\alpha]M_2' : \tau.$$

(Recall $FTV(\rho_1) \subseteq \{\overline{\alpha}_0\}$ and $\Delta_1 = \{(\overline{\alpha}_1, \overline{\sigma}_1, \overline{\sigma}_1')\} \supseteq \Delta_0 = \{(\overline{\alpha}_0, \overline{\sigma}_0, \overline{\sigma}_0')\}$.) Again by the induction hypothesis, we have

$$\Delta_2 \vdash W \sim^\circ_{\mathcal{R}_2} W' : \tau$$

for some $\Delta_2 \supseteq \Delta_1$ and $\mathcal{R}_2 \supseteq \mathcal{R}_1$.

Sub-case $M_3 = x_{1_i}$. Then

$$\begin{aligned}V_{1_i} &= \Lambda\alpha.\,M_2\\ V_{1_i}' &= \Lambda\alpha.\,M_2'\end{aligned}$$

and $\tau_{1_i} = \forall\alpha.\,\rho_2$. Since we have

$$(\Lambda\alpha.\,M_2)[[\overline{\sigma}_1/\overline{\alpha}_1]\rho_1] \Downarrow W$$

and

$$(\Lambda\alpha.\,M_2')[[\overline{\sigma}_1'/\overline{\alpha}_1]\rho_1] \Downarrow W'$$

with

$$\Delta_1 \vdash \Lambda\alpha.\,M_2 \sim_{\mathcal{R}_1} \Lambda\alpha.\,M_2' : \forall\alpha.\,\rho_2,$$

we have

$$(\Delta_1, \mathcal{R}_1 \cup \{(W, W', \tau)\}) \in \sim$$

by Condition 3 of bisimulation. (Recall $\tau = [\rho_1/\alpha]\rho_2$.) Thus, we have

$$\Delta_1 \vdash W \sim^\circ_{\mathcal{R}_2} W' : \tau$$

for $\mathcal{R}_2 = \mathcal{R}_1 \cup \{(W, W', \tau)\}$ by the definition of $\sim^\circ$.

Case (E-App). Then M_0 has the form

$$M_0 \;=\; M_1 M_2$$

and the given evaluation derivations have the forms:

$$\frac{\begin{array}{c}[\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0]M_1 \Downarrow (\texttt{fix}\ f(x\!:\!\pi)\!:\!\rho = M_3)\\ [\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0]M_2 \Downarrow W_1\\ [W_1/x][(\texttt{fix}\ f(x\!:\!\pi)\!:\!\rho = M_3)/f]M_3 \Downarrow W\end{array}}{[\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0](M_1 M_2) \Downarrow W}$$

$$\frac{\begin{array}{c}[\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}_0]M_1 \Downarrow (\texttt{fix}\ f(x\!:\!\pi')\!:\!\rho' = M'_3)\\ [\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}_0]M_2 \Downarrow W'_1\\ [W'_1/x][(\texttt{fix}\ f(x\!:\!\pi')\!:\!\rho' = M'_3)/f]M'_3 \Downarrow W'\end{array}}{[\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}_0](M_1 M_2) \Downarrow W'}$$

By inversion of (T-App), we have

$$\overline{\alpha}_0, \overline{x}_0\!:\!\overline{\tau}_0 \;\vdash\; M_1 \;:\; \sigma \to \tau$$

and

$$\overline{\alpha}_0, \overline{x}_0\!:\!\overline{\tau}_0 \;\vdash\; M_2 \;:\; \sigma$$

for some σ. Thus, by the definition of $\sim^\circ$, we have

$$\Delta_0 \;\vdash\; [\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0]M_1 \;\sim^\circ_{\mathcal{R}_0}\; [\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}_0]M_1 \;:\; \sigma \to \tau.$$

Then, by the induction hypothesis, we have

$$\Delta_1 \;\vdash\; \texttt{fix}\ f(x\!:\!\pi)\!:\!\rho = M_3 \;\sim^\circ_{\mathcal{R}_1}\; \texttt{fix}\ f(x\!:\!\pi')\!:\!\rho' = M'_3 \;:\; \sigma \to \tau$$

for some $\Delta_1 \supseteq \Delta_0$ and $\mathcal{R}_1 \supseteq \mathcal{R}_0$. Therefore, by the definition of $\sim^\circ$, we have

$$\begin{aligned}\texttt{fix}\ f(x\!:\!\pi)\!:\!\rho = M_3 \;&=\; [\overline{V}_1/\overline{x}_1][\overline{\sigma}_1/\overline{\alpha}_1]M_4\\ \texttt{fix}\ f(x\!:\!\pi')\!:\!\rho' = M'_3 \;&=\; [\overline{V}'_1/\overline{x}_1][\overline{\sigma}'_1/\overline{\alpha}_1]M_4\end{aligned}$$

for some

$$\Delta_1 \;\vdash\; \overline{V}_1 \;\sim_{\mathcal{R}_1}\; \overline{V}'_1 \;:\; \overline{\tau}_1$$

and

$$\overline{\alpha}_1, \overline{x}_1\!:\!\overline{\tau}_1 \;\vdash\; M_4 \;:\; \sigma \to \tau$$

with $\Delta_1 = \{(\overline{\alpha}_1, \overline{\sigma}_1, \overline{\sigma}'_1)\}$.

Meanwhile, by weakening, we have

$$\overline{\alpha}_1, \overline{x}_0\!:\!\overline{\tau}_0 \;\vdash\; M_2 \;:\; \sigma.$$

Thus, by the definition of $\sim^\circ$, we have

$$\Delta_1 \;\vdash\; [\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0]M_2 \;\sim^\circ_{\mathcal{R}_1}\; [\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}_0]M_2 \;:\; \sigma$$

since $\Delta_1 \supseteq \Delta_0$ and $\mathcal{R}_1 \supseteq \mathcal{R}_0$. Then, by the induction hypothesis, we have

$$\Delta_2 \;\vdash\; W_1 \;\sim^\circ_{\mathcal{R}_2}\; W'_1 \;:\; \sigma$$

for some $\Delta_2 \supseteq \Delta_1$ and $\mathcal{R}_2 \supseteq \mathcal{R}_1$. Therefore, by the definition of $\sim^\circ$, we have

$$\begin{array}{rcl} W_1 & = & [\overline{V}_2/\overline{x}_2][\overline{\sigma}_2/\overline{\alpha}_2]M_5 \\ W_1' & = & [\overline{V}_2'/\overline{x}_2][\overline{\sigma}_2'/\overline{\alpha}_2]M_5 \end{array}$$

for some

$$\Delta_2 \vdash \overline{V}_2 \sim_{\mathcal{R}_2} \overline{V}_2' : \overline{\tau}_2$$

and

$$\overline{\alpha}_2, \overline{x}_2 : \overline{\tau}_2 \vdash M_5 : \sigma$$

with $\Delta_2 = \{(\overline{\alpha}_2, \overline{\sigma}_2, \overline{\sigma}_2')\}$.

Sub-case $M_4 = (\texttt{fix}\ f(x:\sigma):\tau = M_3'')$**.** By inversion of (T-Fix), we have

$$\overline{\alpha}_1, \overline{x}_1 : \overline{\tau}_1, f : \sigma \to \tau, x : \sigma \vdash M_3'' : \tau.$$

Thus, by weakening and the substitution lemma for values, we have

$$\overline{\alpha}_2, \overline{x}_1 : \overline{\tau}_1, \overline{x}_2 : \overline{\tau}_2 \vdash [(\texttt{fix}\ f(x:\sigma):\tau = M_3'')/f][M_5/x]M_3'' : \tau.$$

Then, by the definition of $\sim^\circ$, we have

$$\begin{array}{rcl} \Delta_2 & \vdash & [\overline{V}_1, \overline{V}_2/\overline{x}_1, \overline{x}_2][\overline{\sigma}_2/\overline{\alpha}_2][(\texttt{fix}\ f(x:\sigma):\tau = M_3'')/f][M_5/x]M_3'' \\ & \sim^\circ_{\mathcal{R}_2} & [\overline{V}_1', \overline{V}_2'/\overline{x}_1, \overline{x}_2][\overline{\sigma}_2'/\overline{\alpha}_2][(\texttt{fix}\ f(x:\sigma):\tau = M_3'')/f][M_5/x]M_3'' : \tau, \end{array}$$

i.e.,

$$\Delta_2 \vdash [W_1/x][(\texttt{fix}\ f(x:\pi):\rho = M_3)/f]M_3 \sim^\circ_{\mathcal{R}_2} [W_1'/x][(\texttt{fix}\ f(x:\pi'):\rho' = M_3')/f]M_3' : \tau.$$

Again by the induction hypothesis, we obtain

$$\Delta_3 \vdash W \sim^\circ_{\mathcal{R}_3} W' : \tau$$

for some $\Delta_3 \supseteq \Delta_2$ and $\mathcal{R}_3 \supseteq \mathcal{R}_2$.

Sub-case $M_4 = x_{1_i}$**.** Then

$$\begin{array}{rcl} V_{1_i} & = & \texttt{fix}\ f(x:\pi):\rho = M_3 \\ V_{1_i}' & = & \texttt{fix}\ f(x:\pi'):\rho' = M_3' \end{array}$$

and $\tau_{1_i} = \sigma \to \tau$. Since we have

$$(\texttt{fix}\ f(x:\pi):\rho = M_3)([\overline{V}_2/\overline{x}_2][\overline{\sigma}_2/\overline{\alpha}_2]M_5) \Downarrow W$$

and

$$(\texttt{fix}\ f(x:\pi'):\rho' = M_3')([\overline{V}_2'/\overline{x}_2][\overline{\sigma}_2'/\overline{\alpha}_2]M_5) \Downarrow W'$$

with

$$\Delta_2 \vdash \texttt{fix}\ f(x:\pi):\rho = M_3 \sim_{\mathcal{R}_2} \texttt{fix}\ f(x:\pi'):\rho' = M_3' : \sigma \to \tau,$$

we have

$$(\Delta_2, \mathcal{R}_2 \cup \{(W, W', \tau)\}) \in \sim$$

by Condition 2 of bisimulation. Thus, we have

$$\Delta_2 \vdash W \sim^\circ_{\mathcal{R}_3} W' : \tau$$

for $\mathcal{R}_3 = \mathcal{R}_2 \cup \{(W, W', \tau)\}$ by the definition of $\sim^\circ$.

Proofs of the other cases are similar. □

C. PROOF OF LEMMA 4.4

The lemma to prove was: If $\Delta_0 \vdash N \sim^\circ_{\mathcal{R}_0} N' : \tau$ then $N \Downarrow \iff N' \Downarrow$.

We assume $N \Downarrow W$ and prove $N' \Downarrow$ by induction on the derivation of $N \Downarrow W$. (The other direction follows by symmetry.) The argument is similar to the proof of Lemma 4.3, except that we are proving the *existence* of an evaluation derivation for N' by using the given evaluation derivation for N, instead of proving a property of given evaluation derivations for N and N'. We show just the most interesting case: the one for (E-Open).

By the definition of $\sim^\circ$, we have

$$N \;=\; [\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0]M_0$$

and

$$N' \;=\; [\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}_0]M_0$$

for some

$$\Delta_0 \;\vdash\; \overline{V}_0 \;\sim_{\mathcal{R}_0}\; \overline{V}'_0 \;:\; \overline{\tau}_0$$

and

$$\overline{\alpha}_0, \overline{x}_0 \,{:}\, \overline{\tau}_0 \;\vdash\; M_0 \;:\; \tau$$

with $\Delta_0 = \{(\overline{\alpha}_0, \overline{\sigma}_0, \overline{\sigma}'_0)\}$. In the case of (E-Open), M_0 has the form

$$M_0 \;=\; \texttt{open}\ M_1\ \texttt{as}\ \alpha, y\ \texttt{in}\ M_2$$

and the given evaluation derivation for N has the following form:

$$\frac{\begin{array}{c}[\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0]M_1 \Downarrow \texttt{pack}\ \rho_1, W_1\ \texttt{as}\ \exists\alpha.\,\rho_2 \\ {[W_1/y][\rho_1/\alpha][\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0]M_2} \Downarrow\end{array}}{[\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0](\texttt{open}\ M_1\ \texttt{as}\ \alpha, y\ \texttt{in}\ M_2) \Downarrow}$$

We aim to derive an evaluation of N' of a similar form:

$$\frac{\begin{array}{c}[\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}_0]M_1 \Downarrow \texttt{pack}\ \rho'_1, W'_1\ \texttt{as}\ \exists\alpha.\,\rho'_2 \\ {[W'_1/y][\rho'_1/\alpha][\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}_0]M_2} \Downarrow\end{array}}{[\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}_0](\texttt{open}\ M_1\ \texttt{as}\ \alpha, y\ \texttt{in}\ M_2) \Downarrow}$$

By inversion of (T-Open), we have

$$\overline{\alpha}_0, \overline{x}_0 \,{:}\, \overline{\tau}_0 \;\vdash\; M_1 \;:\; \exists\alpha.\,\rho''_2$$

and

$$\overline{\alpha}_0, \overline{x}_0 \,{:}\, \overline{\tau}_0, \alpha, y \,{:}\, \rho''_2 \;\vdash\; M_2 \;:\; \tau$$

with $\alpha \notin FTV(\tau)$. Thus, by the definition of $\sim^\circ$, we have

$$\Delta_0 \;\vdash\; [\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}]M_1 \;\sim^\circ_{\mathcal{R}_0}\; [\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}]M_1 \;:\; \exists\alpha.\,\rho''_2.$$

Therefore, by the induction hypothesis and Lemma 4.3, we have

$$[\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}_0]M_1 \Downarrow W'_2$$

for some

$$\Delta_1 \vdash \texttt{pack}\ \rho_1, W_1\ \texttt{as}\ \exists\alpha.\,\rho_2 \ \sim^\circ_{\mathcal{R}_1}\ W'_2 \ :\ \exists\alpha.\,\rho''_2$$

with $\Delta_1 \supseteq \Delta_0$ and $\mathcal{R}_1 \supseteq \mathcal{R}_0$. Then, by the definition of $\sim^\circ$, we have

$$\texttt{pack}\ \rho_1, W_1\ \texttt{as}\ \exists\alpha.\,\rho_2 \ =\ [\overline{V}_1/\overline{x}_1][\overline{\sigma}_1/\overline{\alpha}_1]M_3$$

and

$$W'_2 \ =\ [\overline{V}'_1/\overline{x}_1][\overline{\sigma}'_1/\overline{\alpha}_1]M_3$$

for some

$$\Delta_1 \vdash \overline{V}_1 \sim_{\mathcal{R}_1} \overline{V}'_1 \ :\ \overline{\tau}_1$$

and

$$\overline{\alpha}_1, \overline{x}_1 : \overline{\tau}_1 \vdash M_3 \ :\ \exists\alpha.\,\rho''_2$$

with $\Delta_1 = \{(\overline{\alpha}_1, \overline{\sigma}_1, \overline{\sigma}'_1)\}$.

Sub-case $M_3 = (\texttt{pack}\ \rho''_1, M_4\ \texttt{as}\ \exists\alpha.\,\rho''_2)$**.** Then W'_2 has the form

$$W'_2 \ =\ \texttt{pack}\ \rho'_1, W'_1\ \texttt{as}\ \exists\alpha.\,\rho'_2$$

where

$$\begin{aligned} W_1 &= [\overline{V}_1/\overline{x}_1][\overline{\sigma}_1/\overline{\alpha}_1]M_4 \\ W'_1 &= [\overline{V}'_1/\overline{x}_1][\overline{\sigma}'_1/\alpha_1]M_4 \end{aligned}$$

and

$$\begin{aligned} \rho_1 &= [\overline{\sigma}_1/\overline{\alpha}_1]\rho''_1 \\ \rho'_1 &= [\overline{\sigma}'_1/\overline{\alpha}_1]\rho''_1. \end{aligned}$$

Since we have

$$\overline{\alpha}_1, \overline{x}_1 : \overline{\tau}_1 \vdash M_4 \ :\ [\rho''_1/\alpha]\rho''_2$$

by inversion of (T-Pack), we have

$$\overline{\alpha}_1, \overline{x}_0 : \overline{\tau}_0, \overline{x}_1 : \overline{\tau}_1 \vdash [M_4/y][\rho''_1/\alpha]M_2 \ :\ \tau$$

by weakening and the substitution lemmas for types and terms. Therefore, by the definition of $\sim^\circ$, we have

$$\Delta_1 \vdash [\overline{V}_0, \overline{V}_1/\overline{x}_0, \overline{x}_1][\overline{\sigma}_1/\overline{\alpha}_1][M_4/y][\rho''_1/\alpha]M_2 \ \sim^\circ_{\mathcal{R}_1}\ [\overline{V}'_0, \overline{V}_1/\overline{x}_0, \overline{x}_1][\overline{\sigma}'_1/\overline{\alpha}_1][M_4/y][\rho''_1/\alpha]M_2 \ :\ \tau.$$

Since

$$\begin{aligned} & [\overline{V}_0, \overline{V}_1/\overline{x}_0, \overline{x}_1][\overline{\sigma}_1/\overline{\alpha}_1][M_4/y][\rho''_1/\alpha]M_2 \\ =\ & [([\overline{V}_1/\overline{x}_1][\overline{\sigma}_1/\overline{\alpha}_1]M_4)/y][([\overline{\sigma}_1/\overline{\alpha}_1]\rho''_1)/\alpha][\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0]M_2 \\ =\ & [W_1/y][\rho_1/\alpha][\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0]M_2 \end{aligned}$$

and

$$\begin{aligned}&[\overline{V}'_0, \overline{V}'_1/\overline{x}_0, \overline{x}_1][\overline{\sigma}'_1/\overline{\alpha}_1][M_4/y][\rho''_1/\alpha]M_2\\ =\ &[(([\overline{V}'_1/\overline{x}_1][\overline{\sigma}'_1/\overline{\alpha}_1]M_4)/y][(([\overline{\sigma}'_1/\overline{\alpha}_1]\rho''_1)/\alpha][\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}_0]M_2\\ =\ &[W'_1/y][\rho'_1/\alpha][\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}_0]M_2,\end{aligned}$$

we have

$$[W'_1/y][\rho'_1/\alpha][\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}_0]M_2 \ \Downarrow$$

by the induction hypothesis, i.e., $N' \Downarrow$.

Sub-case $M_3 = x_{1_i}$**.** Then $W'_2 = V'_{1_i}$ where

$$V_{1_i} \ = \ \mathtt{pack}\ \rho_1, W_1\ \mathtt{as}\ \exists\alpha.\,\rho_2$$

and $\tau_{1_i} = \exists\alpha.\,\rho''_2$. By Condition 1 of bisimulation, V'_{1_i} is a value of the existential type $[\overline{\sigma}'_1/\overline{\alpha}_1]\tau_{1_i} = \exists\alpha.\,([\overline{\sigma}'_1/\overline{\alpha}_1]\rho''_2)$, so it has the form

$$V'_{1_i} \ = \ \mathtt{pack}\ \rho'_1, W'_1\ \mathtt{as}\ \exists\alpha.\,\rho'_2.$$

Since

$$\Delta_1 \vdash V_{1_i} \sim_{\mathcal{R}_1} V'_{1_i} \ :\ \tau_{1_i},$$

we have the following two possibilities by Condition 4 of bisimulation.

Sub-sub-case $(\beta, \rho_1, \rho'_1) \in \Delta_1$ **and** $(W_1, W'_1, [\beta/\alpha]\rho''_2) \in \mathcal{R}_1$**.** Since we have

$$\overline{\alpha}_0, \overline{x}_0 : \overline{\tau}_0, \alpha, y : \rho''_2 \vdash M_2 \ :\ \tau$$

with $\alpha \notin FTV(\tau)$, we have

$$\overline{\alpha}_0, \overline{x}_0 : \overline{\tau}_0, \beta, y : [\beta/\alpha]\rho''_2 \vdash [\beta/\alpha]M_2 \ :\ \tau.$$

Then, we have

$$\Delta_1 \vdash [\overline{V}_0, W_1/\overline{x}_0, y][\overline{\sigma}_1/\overline{\alpha}_1][\beta/\alpha]M_2 \sim^{\circ}_{\mathcal{R}_1} [\overline{V}'_0, W'_1/\overline{x}_0, y][\overline{\sigma}'_1/\overline{\alpha}_1][\beta/\alpha]M_2 \ :\ \tau$$

by the definition of $\sim^{\circ}$. Since we have $\beta = \alpha_{1_i}$ with $\rho_1 = \sigma_{1_i}$ and $\rho'_1 = \sigma'_{1_i}$ for some i, we have

$$[\overline{V}_0, W_1/\overline{x}_0, y][\overline{\sigma}_1/\overline{\alpha}_1][\beta/\alpha]M_2 \ = \ [W_1/y][\rho_1/\alpha][\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0]M_2$$

and

$$[\overline{V}'_0, W'_1/\overline{x}_0, y][\overline{\sigma}'_1/\overline{\alpha}_1][\beta/\alpha]M_2 \ = \ [W'_1/y][\rho'_1/\alpha][\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}_0]M_2,$$

so we have

$$[W'_1/y][\rho'_1/\alpha][\overline{V}'_0/\overline{x}_0][\overline{\sigma}'_0/\overline{\alpha}_0]M_2 \ \Downarrow$$

by the induction hypothesis, i.e., $N' \Downarrow$.

Sub-sub-case $(\Delta_1 \uplus \{(\alpha, \rho_1, \rho'_1)\}, \mathcal{R}_1 \cup \{(W_1, W'_1, \rho''_2)\}) \in \sim$**.** Then we have

$$\Delta_2 \vdash [\overline{V}_0, W_1/\overline{x}_0, y][\overline{\sigma}_0, \rho_1/\overline{\alpha}_0, \alpha]M_2 \sim^{\circ}_{\mathcal{R}_2} [\overline{V}'_0, W'_1/\overline{x}_0, y][\overline{\sigma}'_0, \rho'_1/\overline{\alpha}_0, \alpha]M_2 \ :\ \tau$$

for $\Delta_2 = \Delta_1 \cup \{(\alpha, \rho_1, \rho_1')\}$ and $\mathcal{R}_2 = \mathcal{R}_1 \cup \{(W_1, W_1', \rho_2'')\}$ by the definition of $\sim^\circ$. Since we have

$$[\overline{V}_0, W_1/\overline{x}_0, y][\overline{\sigma}_0, \rho_1/\overline{\alpha}_0, \alpha]M_2 \ = \ [W_1/y][\rho_1/\alpha][\overline{V}_0/\overline{x}_0][\overline{\sigma}_0/\overline{\alpha}_0]M_2$$

and

$$[\overline{V}_0', W_1'/\overline{x}_0, y][\overline{\sigma}_0', \rho_1'/\overline{\alpha}_0, \alpha]M_2 \ = \ [W_1'/y][\rho_1'/\alpha][\overline{V}_0'/\overline{x}_0][\overline{\sigma}_0'/\overline{\alpha}_0]M_2,$$

we have

$$[W_1'/y][\rho_1'/\alpha][\overline{V}_0'/\overline{x}_0][\overline{\sigma}_0'/\overline{\alpha}_0]M_2 \ \Downarrow$$

by the induction hypothesis, i.e., $N' \Downarrow$. □

ACKNOWLEDGMENTS

We would like to thank Karl Crary, Andy Gordon, Bob Harper, Vassileios Koutavas, and Andrew Pitts for information and discussions on their work and its relationship to ours. Comments from anonymous reviewers, the members of the PL Club at the University of Pennsylvania, and Naoki Kobayashi helped us to sharpen the presentation. Reviewer 2 of JACM suggested the examples in Section 7.

REFERENCES

Abadi, M. and Fournet, C. 2001. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.* 104–115.

Abadi, M. and Gordon, A. D. 1998. A bisimulation method for cryptographic protocols. *Nordic Journal of Computing 5*, 267–303. Preliminary version appeared in *7th European Symposium on Programming*, *Lecture Notes in Computer Science*, Springer-Verlag, vol. 1381, pp. 12–26, 1998.

Abadi, M. and Gordon, A. D. 1999. A calculus for cryptographic protocols: The spi calculus. *Information and Computation 148,* 1, 1–70. Preliminary version appeared in *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pp. 36–47, 1997.

Abramsky, S. 1990. The lazy lambda calculus. In *Research Topics in Functional Programming*, D. A. Turner, Ed. Addison-Wesley, 65–117.

Ahmed, A. 2006. Step-indexed syntactic logical relations for recursive and quantified types. In *15th European Symposium on Programming.* 69–83.

Ahmed, A., Appel, A. W., and Virga, R. 2003. An indexed model of impredicative polymorphism and mutable references. `http://www.cs.princeton.edu/~amal/papers/impred.pdf`.

Appel, A. W. and McAllester, D. 2001. An indexed model of recursive types for foundational proof-carrying code. *ACM Transactions on Programming Languages and Systems 23,* 5, 657–683.

Berger, M., Honda, K., and Yoshida, N. 2003. Genericity and the pi-calculus. In *Foundations of Software Science and Computation Structures.* Lecture Notes in Computer Science, vol. 2620. Springer-Verlag, 103–119.

Bierman, G. M., Pitts, A. M., and Russo, C. V. 2000. Operational properties of Lily, a polymorphic linear lambda calculus with recursion. In *Higher Order Operational Techniques in Semantics.* Electronic Notes in Theoretical Computer Science, vol. 41. Elsevier Science.

Birkedal, L. and Harper, R. 1999. Relational interpretations of recursive types in an operational setting. *Information and Computation 155,* 1–2, 3–63. Summary appeared in *Theoretical Aspects of Computer Software*, *Lecture Notes in Computer Science*, Springer-Verlag, vol. 1281, pp. 458–490, 1997.

Boreale, M., De Nicola, R., and Pugliese, R. 2002. Proof techniques for cryptographic processes. *SIAM Journal on Computing 31,* 3, 947–986. Preliminary version appeared in *14th Annual IEEE Symposium on Logic in Computer Science*, pp. 157–166, 1999.

Borgström, J. and Nestmann, U. 2002. On bisimulations for the spi calculus. In *9th International Conference on Algebraic Methodology and Software Technology.* Lecture Notes in Computer Science, vol. 2422. Springer-Verlag, 287–303.

Bruce, K. B., Cardelli, L., and Pierce, B. C. 1999. Comparing object encodings. *Information and Computation 155,* 1–2, 108–133. Extended abstract appeared in *Theoretical Aspects of Computer Software*, Springer-Verlag, vol. 1281, pp. 415–338, 1997.

Crary, K. and Harper, R. 2007. Syntactic logical relations for polymorphic and recursive types. In *Computation, Meaning, and Logic: Articles dedicated to Gordon Plotkin.* Electronic Notes in Theoretical Computer Science, vol. 172. Elsevier Science, 259–299.

Gordon, A. D. 1995a. Bisimilarity as a theory of functional programming. mini-course. `http://research.microsoft.com/~adg/Publications/BRICS-NS-95-3.dvi.gz`.

Gordon, A. D. 1995b. Operational equivalences for untyped and polymorphic object calculi. In *Higher Order Operational Techniques in Semantics.* Cambridge University Press, 9–54.

Gordon, A. D. and Rees, G. D. 1995. Bisimilarity for $F_{<:}$. Draft.

Gordon, A. D. and Rees, G. D. 1996. Bisimilarity for a first-order calculus of objects with subtyping. In *Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.* 386–395.

Heintze, N. and Riecke, J. G. 1998. The SLam calculus: Programming with secrecy and integrity. In *Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.*

Howe, D. J. 1996. Proving congruence of bisimulation in functional programming languages. *Information and Computation 124,* 2, 103–112.

Hughes, D. J. 1997. Games and definability for System F. In *Twelfth Annual IEEE Symposium on Logic in Computer Science.* 76–86.

Koutavas, V. and Wand, M. 2006a. Bisimulations for untyped imperative objects. In *15th European Symposium on Programming.* 146–161.

Koutavas, V. and Wand, M. 2006b. Small bisimulations for reasoning about higher-order imperative programs. In *Proceedings of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.* 141–152.

Koutavas, V. and Wand, M. 2007. Reasoning about class behavior. In *2007 International Workshop on Foundations and Developments of Object-Oriented Languages.* `http://foolwood07.cs.uchicago.edu/program/koutavas.pdf`.

Melliés, P.-A. and Vouillon, J. 2005. Recursive polymorphic types and parametricity in an operational framework. In *20th Annual IEEE Symposium on Logic in Computer Science.* 82–91.

Meyer, A. R. and Sieber, K. 1988. Towards fully abstract semantics for local variables: Preliminary report. In *Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.* 191–203.

Milner, R. 1980. *A Calculus of Communicating Systems.* Number 92 in Lecture Notes in Computer Science. Springer-Verlag.

Milner, R. 1989. *Communication and Concurrency.* Prentice Hall.

Milner, R. 1999. *Communicating and Mobile Systems: The π-Calculus.* Cambridge University Press.

Mitchell, J. C. 1996. *Foundations for Programming Languages.* MIT Press.

Moggi, E. 1991. Notions of computation and monads. *Information and Computation 93,* 1, 55–92.

Morris, Jr., J. H. 1973a. Protection in programming languages. *Communications of the ACM 16,* 1, 15–21.

Morris, Jr., J. H. 1973b. Types are not sets. In *Proceedings of the 1st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages.* 120–124.

PIERCE, B. C. AND SANGIORGI, D. 2000. Behavioral equivalence in the polymorphic pi-calculus. *Journal of the ACM 47,* 3, 531–586. Extended abstract appeared in *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 1997, pp. 531–584.

PITTS, A. 2005. Typed operational reasoning. In *Advanced Topics in Types and Programming Languages*, B. C. Pierce, Ed. MIT Press, Chapter 7, 245–289. Preliminary version appeared as *Existential Types: Logical Relations and Operational Equivalence* in *Automata, Languages and Programming*, *Lecture Notes in Computer Science*, Springer-Verlag, vol. 1443, pp. 309–326, 1998.

PITTS, A. M. 2000. Parametric polymorphism and operational equivalence. *Mathematical Structures in Computer Science 10*, 321–359. Preliminary version appeared in *HOOTS II Second Workshop on Higher-Order Operational Techniques in Semantics*, *Electronic Notes in Theoretical Computer Science*, vol. 10, 1998.

PITTS, A. M. AND STARK, I. 1993. Observable properties of higher order functions that dynamically create local names, or: what's new? In *Mathematical Foundations of Computer Science.* Lecture Notes in Computer Science, vol. 711. Springer-Verlag, 122–141.

PITTS, A. M. AND STARK, I. 1998. Operational reasoning for functions with local state. In *Higher Order Operational Techniques in Semantics.* Cambridge University Press, 227–273.

SANGIORGI, D. 1992. Expressing mobility in process algebras: First-order and higher-order paradigm. Ph.D. thesis, University of Edinburgh.

SANGIORGI, D., KOBAYASHI, N., AND SUMII, E. 2007. Environmental bisimulations for higher-order languages. In *Twenty-Second Annual IEEE Symposium on Logic in Computer Science.* 293–302.

SUMII, E. AND PIERCE, B. C. 2004. A bisimulation for dynamic sealing. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.* 161–172.

WADLER, P. 1989. Theorems for free! In *Proceedings of the Fourth ACM SIGPLAN International Conference on Functional Programming Languages and Computer Architecture.* ACM, 347–359.

Received Month Year; revised Month Year; accepted Month Year