Bisimilarity for a First-Order Calculus of Objects with Subtyping*

Andrew D. Gordon and Gareth D. Rees
University of Cambridge Computer Laboratory
http://www.cl.cam.ac.uk/users/{adg,gdr11}/

January 1996

Abstract

Bisimilarity (also known as 'applicative bisimulation') has attracted a good deal of attention as an operational equivalence for λ-calculi. It approximates or even equals Morris-style contextual equivalence and admits proofs of program equivalence via co-induction. It has an elementary construction from the operational definition of a language. We consider bisimilarity for one of the typed object calculi of Abadi and Cardelli. By defining a labelled transition system for the calculus in the style of Crole and Gordon and using a variation of Howe's method we establish two central results: that bisimilarity is a congruence, and that it equals contextual equivalence. So two objects are bisimilar iff no amount of programming can tell them apart. Our third contribution is to show that bisimilarity soundly models the equational theory of Abadi and Cardelli. This is the first study of contextual equivalence for an object calculus and the first application of Howe's method to subtyping. By these results, we intend to demonstrate that operational methods are a promising new direction for the foundations of object-oriented programming.

*Version of March 25, 1996. A technical summary of this report appears in the proceedings of the 23rd ACM symposium on Principles of Programming Languages, St Petersburg Beach, Florida, January 1996, pages 386–395.

Contents

List of Tables

1 Motivation

Abadi and Cardelli (1994a, 1994b, 1994c) present a number of related calculi that formalise aspects of object-oriented programming languages, including method update (the ability to modify the behaviour of an object by altering one of its methods) and object subsumption (the ability to emulate an object with an object that has more methods). They give equational theories for their calculi, present a denotational semantics based on partial equivalence relations for the largest calculus and show that the equational theory is sound. Their object calculi form an extremely simple yet clearly object-oriented setting in which to seek type systems that support styles of object-oriented programming found in full-blown languages. Hence they are an important subject of research.

Abadi and Cardelli's goal was to study type systems for objects by abandoning complex encodings of objects as λ-terms and to study primitive objects in their own right. Our goal here is to study operational equivalence of objects in its own right, instead of via denotational semantics, another kind of encoding.

We work with $\mathbf{FOb_1}_{<:\mu}$ (Abadi and Cardelli 1994c), a first-order stateless object calculus including a ground type of Booleans, objects, recursive types and functions. We take Morris-style contextual equivalence (Morris 1968) to be the natural operational equivalence on objects: two programs are equivalent unless there is a distinguishing context of ground type such that when each program is placed in the context, one converges but the other diverges. Following earlier work on functional languages (Crole and Gordon 1995; Gordon 1995), we define a CCS-style labelled transition system for the object calculus and replay the definition of (strong) bisimilarity from CCS. Our bisimilarity descends from a line of work on operational equivalence for functional calculi beginning with Abramsky's applicative bisimulation (Abramsky and Ong 1993). The new elements here are subtyping and objects. Using an extension of the method of Howe (1989) we prove Theorem 1, that bisimilarity is a congruence. Theorem 2, that bisimilarity equals contextual equivalence, then follows easily. The quantification over all contexts makes contextual equivalence hard to establish directly. The purpose of the labelled transition system, justified by Theorem 2, is to admit CCS-style bisimulation proofs of contextual equivalence. We use this style of proof—a form of co-induction—for our final result, Theorem 3, that bisimilarity soundly models all of Abadi and Cardelli's equational theory.

We briefly examine equivalences between two example objects suggested by Abadi and Cardelli. One of the equivalences follows by co-induction but not from their equational theory.

Our framework appears to be robust. Our main results continue to hold when we extend the pure object calculus with functions, records, variants and dynamic types. We leave a study of polymorphic types as future work, but see Rees (1994).

Section 2 introduces syntax and operational semantics of the object calculus $\mathbf{Ob}_{\mathbf{1}<:\mu}$ and its extension with functions, $\mathbf{FOb}_{\mathbf{1}<:\mu}$. Contextual equivalence is defined in Section 3, and we prove it equal to a form of bisimilarity in Section 4. In Section 5 we show that contextual equivalence is a model of Abadi and Cardelli's equational theory, and we consider one of their examples in Section 6. Section 7 extends $\mathbf{FOb}_{\mathbf{1}<:\mu}$ with records, variants and dynamic types. We investigate the connection between divergence and contextual equivalence in Section 8. In Section 9 we briefly discuss other forms of operational equivalence. In Section 10 we exercise our theory by proving that Abadi and Cardelli's embedding of functions as objects is sound, but not fully abstract. We review related work in Section 11 and conclude in Section 12.

For the sake of clarity, unless a proof is particularly compelling or interesting, we postpone it to Appendix A. Although the main text deals mainly with the language $\mathbf{FOb}_{\mathbf{1}<:\mu}$, in Appendix A we prove all the results for the full language incorporating the extensions given in Section 7.

2 An object calculus

This section recalls Abadi and Cardelli's $\mathbf{FOb}_{\mathbf{1}<:\mu}$, a stateless calculus with subtyping, including `Top`, one ground type `Bool`, object types, recursive types and function types.

The syntax is a λ-calculus extended with expressions for Booleans, conditionals, **object formation**, $[\ell_1 = \varsigma(x_1{:}A_1)e_1, \ldots, \ell_n = \varsigma(x_n{:}A_n)e_n]$ (where each $\varsigma(x_i{:}A_i)e_i$ is a **method**), **method selection**, $a.\ell$, **method update**, $a.\ell \Leftarrow \varsigma(x{:}A)e$, and recursive types. We use the metavariables x and X for variables and type variables, and e and E for possibly open expressions and types, respectively. Let I stand for finite indexing sets and ℓ for labels, drawn from some countably infinite set. Formally, the grammars of **types**, E, and **expressions**, e, are given as follows.

$$\begin{array}{rcl} E & ::= & X \mid \texttt{Top} \mid \texttt{Bool} \mid [\ell_i{:}E_i]_{i\in I} \mid \mu(X)E \mid E \to E' \\ e & ::= & x \mid \texttt{true} \mid \texttt{false} \mid \texttt{if}(e,e,e) \\ & & \mid [\ell_i = \varsigma(x_i{:}E_i)e_i]_{i\in I} \mid e.\ell \mid e.\ell \Leftarrow \varsigma(x{:}E)e \\ & & \mid \texttt{fold}(E,e) \mid \texttt{unfold}(e) \\ & & \mid \lambda(x{:}E)e \mid (e\, e') \end{array}$$

Let $fv(e)$ be the set of free variables of the expression e, and $ftv(e)$ and $ftv(E)$ be the sets of free type variables of the expression e and the type E, respectively. We identify expressions and types up to alpha-conversion, denoted by $\equiv$.

An **environment**, Γ, is a finite list of assignments of closed types to variables, $x{:}A$, followed by a finite list of type variable bounds, $X <: E$. Let Γ, X, Γ' be short for $\Gamma, X <: \texttt{Top}, \Gamma'$. Let $Dom(\Gamma)$ be the set of variables and type variables bound or assigned in Γ. The static semantics of the calculus consists of five inductively defined judgments: $\Gamma \vdash \diamond$ (the environment Γ is well-formed), $E \succ Y$ (the type E is formally contractive in variable Y), $\Gamma \vdash E$ (the type expression E is well-formed), $\Gamma \vdash E <: E'$ (the type E is a subtype of E'), and $\Gamma \vdash e : A$ (the expression e has the closed type A). These judgments are given inductively by the rules in Tables 1, 2, 3, 4 and 5. We follow a metavariable convention based on the following sets.

$$\begin{array}{rcl} A, B \in \mathit{Type} & \stackrel{\text{def}}{=} & \{E \mid \varnothing \vdash E\} \\ a, b \in \mathit{Prog}(A) & \stackrel{\text{def}}{=} & \{e \mid \varnothing \vdash e : A\} \end{array}$$

By **program** we specifically mean a closed expression contained in $Prog(A)$ for some A, respectively. As usual we write simply $a_1, a_2, \ldots, a_n{:}A$ to mean $\{a_1, a_2, \ldots, a_n\} \subseteq Prog(A)$ and $A <: B$ for $\varnothing \vdash A <: B$.

Here are basic facts about the static semantics.

$$\frac{}{\varnothing \vdash \diamond}\ (\text{Env } \varnothing) \qquad \frac{\Gamma \vdash \diamond \quad \varnothing \vdash A \quad x \notin \mathit{Dom}(\Gamma)}{\Gamma, x{:}A \vdash \diamond}\ (\text{Env } x)$$

$$\frac{\Gamma \vdash E \quad X \notin \mathit{Dom}(\Gamma)}{\Gamma, X <: E \vdash \diamond}\ (\text{Env } X{<:})$$

Table 1: Well-formed environments

$$\frac{X \neq Y}{X \succ Y} \qquad \frac{}{\texttt{Top} \succ Y} \qquad \frac{}{\texttt{Bool} \succ Y} \qquad \frac{}{[\ell_i{:}E_i]_{i \in I} \succ Y} \qquad \frac{E \succ Y}{\mu(X)E \succ Y}$$

Table 2: Formal contractivity

$$\frac{\Gamma, X <: E, \Gamma' \vdash \diamond}{\Gamma, X <: E, \Gamma' \vdash X}\ (\text{Type } X{<:}) \qquad \frac{\Gamma \vdash \diamond}{\Gamma \vdash \texttt{Top}}\ (\text{Type Top})$$

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash \texttt{Bool}}\ (\text{Type Bool}) \qquad \frac{\Gamma \vdash E_i\ (\forall i \in I) \quad \ell_i \text{ distinct}}{\Gamma \vdash [l_i : E_i]_{i \in I}}\ (\text{Type Object})$$

$$\frac{\Gamma, X <: \texttt{Top} \vdash E \quad E \succ X}{\Gamma \vdash \mu(X)E}\ (\text{Type Rec } <:) \qquad \frac{\Gamma \vdash E \quad \Gamma \vdash E'}{\Gamma \vdash E \to E'}\ (\text{Type Arrow})$$

Table 3: Well-formed types

$$\frac{\Gamma \vdash E}{\Gamma \vdash E <: E}\ (\text{Sub Refl}) \qquad \frac{\Gamma \vdash E_1 <: E_2 \quad \Gamma \vdash E_2 <: E_3}{\Gamma \vdash E_1 <: E_3}\ (\text{Sub Trans})$$

$$\frac{\Gamma, X <: E, \Gamma' \vdash \diamond}{\Gamma, X <: E, \Gamma' \vdash X <: E}\ (\text{Sub } X) \qquad \frac{\Gamma \vdash E}{\Gamma \vdash E <: \mathtt{Top}}\ (\text{Sub Top})$$

$$\frac{J \subseteq I \quad \Gamma \vdash E_i\ (\forall i \in I)}{\Gamma \vdash [\ell_i : E_i]_{i \in I} <: [\ell_i : E_i]_{i \in J}}\ (\text{Sub Object})$$

$$\frac{\Gamma \vdash \mu(X_1)E_1 \quad \Gamma \vdash \mu(X_2)E_2 \quad \Gamma, X_2 <: \mathtt{Top}, X_1 <: X_2 \vdash E_1 <: E_2}{\Gamma \vdash \mu(X_1)E_1 <: \mu(X_2)E_2}\ (\text{Sub Rec})$$

$$\frac{\Gamma \vdash E_1' <: E_1 \quad \Gamma \vdash E_2 <: E_2'}{\Gamma \vdash E_1 \to E_2 <: E_1' \to E_2'}\ (\text{Sub Arrow})$$

Table 4: Subtyping relation

Lemma 1

(1) *If* $\Gamma \vdash E$ *then* $ftv(E) \subseteq Dom(\Gamma)$ *and* $\Gamma \vdash \diamond$.

(2) *If* $\Gamma \vdash E <: E'$ *then* $ftv(E) \cup ftv(E') \subseteq Dom(\Gamma)$ *and* $\Gamma \vdash \diamond$.

(3) *If* $\Gamma \vdash e : A$ *then* $ftv(e) \cup fv(e) \subseteq Dom(\Gamma)$ *and* $\Gamma \vdash \diamond$.

(4) *If* $\Gamma \vdash [\ell_i = \varsigma(x_i{:}A_i)e_i]_{i \in I} : B$ *then the* A_i *are identical and each* $A_i <: B$.

(5) *If* $A <: B$ *then* $Prog(A) \subseteq Prog(B)$.

(6) *If* $A <: B$ *and both* A *and* B *are object types, they must have the forms* $[\ell_i{:}A_i]_{i \in I}$ *and* $[\ell_j{:}A_j]_{j \in J}$, *respectively, with* $J \subseteq I$.

(7) *If* $\Gamma, X', X <: X' \vdash E <: E'$ *and* $\Gamma \vdash A <: A'$ *then* $\Gamma \vdash E[A/X] <: E'[A'/X']$.

(8) *Subtyping is decidable.*

(9) *If* $\Gamma \vdash E <: E'$ *and* $\Gamma \vdash E' <: E$ *then* $E \equiv E'$.

Abadi and Cardelli discuss the $\mathbf{FOb}_{\mathbf{1}<:\mu}$ type system at length. A few points are worth noting here. The pure object calculus $\mathbf{Ob}_{\mathbf{1}<:\mu}$ can be obtained by removing functions from $\mathbf{FOb}_{\mathbf{1}<:\mu}$. Numbers, lists and so on can

$$\frac{\Gamma, x{:}A, \Gamma' \vdash \diamond}{\Gamma, x{:}A, \Gamma' \vdash x : A}\ (\text{Val } x) \qquad \frac{\Gamma \vdash e_1 : \texttt{Bool} \quad \Gamma \vdash e_2 : A \quad \Gamma \vdash e_3 : A}{\Gamma \vdash \texttt{if}(e_1, e_2, e_3) : A}\ (\text{Val If})$$

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash \texttt{true} : \texttt{Bool}}\ (\text{Val True}) \qquad \frac{\Gamma \vdash \diamond}{\Gamma \vdash \texttt{false} : \texttt{Bool}}\ (\text{Val False})$$

$$\frac{\Gamma, x_i{:}A \vdash e_i : A_i\ (\forall i \in I) \quad A \equiv [\ell_i{:}A_i]_{i \in I} \quad \Gamma \vdash \diamond \quad \ell_i \text{ distinct}}{\Gamma \vdash [\ell_i = \varsigma(x_i{:}A)e_i]_{i \in I} : A}\ (\text{Val Object})$$

$$\frac{\Gamma \vdash e : [\ell_i{:}A_i]_{i \in I} \quad j \in I}{\Gamma \vdash e.\ell_j : A_j}\ (\text{Val Select})$$

$$\frac{A \equiv [\ell_i{:}A_i]_{i \in I} \quad \Gamma \vdash e : A \quad \Gamma, x{:}A \vdash e' : A_j \quad j \in I}{\Gamma \vdash e.\ell_j \Leftarrow \varsigma(x{:}A)e' : A}\ (\text{Val Update})$$

$$\frac{A \equiv \mu(X)E \quad \Gamma \vdash e : E[{}^{A}\!/_{X}] \quad \varnothing \vdash A}{\Gamma \vdash \texttt{fold}(A, e) : A}\ (\text{Val Fold})$$

$$\frac{A \equiv \mu(X)E \quad \Gamma \vdash e : A}{\Gamma \vdash \texttt{unfold}(e) : E[{}^{A}\!/_{X}]}\ (\text{Val Unfold})$$

$$\frac{\Gamma, x{:}B \vdash e : A}{\Gamma \vdash \lambda(x{:}B)e : B \to A}\ (\text{Val Fun}) \qquad \frac{\Gamma \vdash e_1 : B \to A \qquad \Gamma \vdash e_2 : B}{\Gamma \vdash (e_1\, e_2) : A}\ (\text{Val Appl})$$

$$\frac{\Gamma \vdash e : A_1 \quad \Gamma \vdash A_1 <: A_2}{\Gamma \vdash e : A_2}\ (\text{Val Subsumption})$$

Table 5: Type assignment

$$\frac{}{\texttt{if}(\texttt{true}, a_1, a_2) \mapsto a_1}\ \text{(Red If True)} \qquad \frac{}{\texttt{if}(\texttt{false}, a_1, a_2) \mapsto a_2}\ \text{(Red If False)}$$

$$\frac{a \equiv [\ell_i = \varsigma(x_i{:}A_i)e_i]_{i\in I} \quad j \in I}{a.\ell_j \mapsto e_j[a/x_j]}\ \text{(Red Select)}$$

$$\frac{\begin{array}{c} a \equiv [\ell_i = \varsigma(x_i{:}A_i)e_i]_{i\in I} \\ a' \equiv [\ell_j = \varsigma(x{:}A_j)e, \ell_i = \varsigma(x_i{:}A_i)e_i]_{i\in I-\{j\}} \quad j \in I \end{array}}{a.\ell_j \Leftarrow \varsigma(x{:}B)e \mapsto a'}\ \text{(Red Update)}$$

$$\frac{}{\texttt{unfold}(\texttt{fold}(A, v)) \mapsto v}\ \text{(Red Unfold)} \qquad \frac{}{((\lambda(x{:}A)e)\, a) \mapsto e[a/x]}\ \text{(Red App)}$$

$$\frac{a \mapsto b}{\mathcal{E}[a] \mapsto \mathcal{E}[b]}\ \text{(Red Experiment)}$$

Table 6: Single-step reduction

be encoded in $\mathbf{FOb_{1<:\mu}}$ and functions can be encoded in the pure calculus $\mathbf{Ob_{1<:\mu}}$. Thus $\mathbf{FOb_{1<:\mu}}$ is essentially PCF plus subtyping, recursive types and objects. The contractive constraint on recursive types implies that any closed type can be decomposed into the form $\mu(X_1)\ldots\mu(X_n)E$ where E is one of $\texttt{Top}$, $\texttt{Bool}$, $[\ell_i{:}E_i]_{i\in I}$, or $E \to E'$. Whenever $A <: B$ and both A and B are object types, they must have the forms $[\ell_i{:}A_i]_{i\in I}$ and $[\ell_j{:}A_j]_{j\in J}$, respectively, with $J \subseteq I$. In other words, an object type is **invariant** in its component types, that is, neither covariant nor contravariant. This is necessary because methods have a contravariant dependence on self, but as a consequence only nonvariant functions may be encoded in $\mathbf{Ob_{1<:\mu}}$. The primitive functions in $\mathbf{FOb_{1<:\mu}}$ have the usual subtyping: contravariant in the domain and covariant in the codomain.

We need the following substitution and bound weakening lemmas, which are standard.

Lemma 2 *If* $\Gamma, x{:}A, \Gamma' \vdash e : B$ *and* $\Gamma \vdash e' : A$, *then* $\Gamma, \Gamma' \vdash e[e'/x] : B$.

Lemma 3 *If* $\Gamma, x{:}A, \Gamma' \vdash e : B$ *and* $A' <: A$ *then* $\Gamma, x{:}A', \Gamma' \vdash e : B$ *too.*

Abadi and Cardelli present a many-step deterministic evaluation relation, $\rightsquigarrow$, for $\mathbf{FOb_{1<:\mu}}$. For our purposes, it is more convenient to reformulate it as

a single-step reduction relation. To specify the evaluation strategy we need the following notion of a context. Let '$-$' be a distinguished variable used to stand for a **hole** in a program. Let a **context** be an expression e such that the only free variable, if any, is $-$. If e is a context and a is a program, we write $e[a]$ short for the program $e[a/-]$. These are not variable-capturing contexts.

For each type A define the set $\mathit{Value}(A) \subseteq \mathit{Prog}(A)$ (with typical members u, v) of **values** of the following forms.

$$\texttt{true} \quad \texttt{false} \quad [\ell_i = \varsigma(x_i{:}B)e_i]_{i\in I} \quad \texttt{fold}(B, v) \quad \lambda(x{:}B)e$$

Let Value be the set of values at any type, that is, $\bigcup\{\mathit{Value}(A) \mid A \in \mathit{Type}\}$.

The notation $a \mapsto b$ means that a reduces to b in a single step of reduction; it is defined inductively by the rules in Table 6. The rule (Red Experiment) gives a reduction strategy. An **experiment**, $\mathcal{E}$, is a context with one hole, of one of the following forms.

$$\begin{array}{c} \texttt{if}(-, a_1, a_2) \quad -.\ell \quad -.\ell \Leftarrow \varsigma(x{:}A)e \\ \texttt{unfold}(-) \quad \texttt{fold}(A, -) \quad (-\,a) \end{array}$$

The relation $\mapsto$ is weak in the sense that it is closed only under experiments, not arbitrary contexts. There are no experiments to allow reduction under ς- or λ-abstractions. (Red Update) preserves the property that whenever an object $[\ell_i = \varsigma(x_i{:}A_i)e_i]_{i\in I}$ has type A, each $A_i <: A$ and indeed all the A_i's are identical.

Experiments are just atomic evaluation contexts (Felleisen and Friedman 1986); every program can be decomposed uniquely as follows.

Lemma 4 *If $a{:}A$ there is a unique list of experiments $\mathcal{E}_1, \ldots, \mathcal{E}_n$, $n \geq 0$, and value v such that $a \equiv \mathcal{E}_1[\ldots\mathcal{E}_n[v]\ldots]$.*

Lemma 5 *The values are the normal forms of $\mapsto$, that is, whenever a is a program, $a \in \mathit{Value}$ iff $\neg\exists b(a \mapsto b)$.*

The reduction rules are deliberately type independent, in the sense that although they manipulate type information contained in programs, they are not contingent on the form of the types they manipulate. For instance, (Red Update) allows for the type bounds, A_i, to be distinct although we know them to be identical. Hence ill-typed expressions can be reduced. This is a redundancy in the dynamic semantics, but is harmless because we are only interested in statically typable programs.

We can easily prove determinacy and subject reduction.

Lemma 6 *If $a \mapsto b$ and $a \mapsto c$, then $b \equiv c$.*

Lemma 7 *If $a{:}A$ and $a \mapsto b$ then $b{:}A$.*

As usual, let the relation $\mapsto^*$ be the reflexive and transitive closure of $\mapsto$. We now recover a many-step evaluation relation, $\Downarrow$, and three standard predicates as follows.

$$\begin{array}{rcll} a \mapsto & \stackrel{\text{def}}{=} & \exists b(a \mapsto b) & \text{`}a \text{ reduces'} \\ a \Downarrow b & \stackrel{\text{def}}{=} & a \mapsto^* b \;\&\; \neg(b \mapsto) & \text{`}a \text{ evaluates to } b\text{'} \\ a{\Downarrow} & \stackrel{\text{def}}{=} & \exists b(a \Downarrow b) & \text{`}a \text{ converges'} \\ a{\Uparrow} & \stackrel{\text{def}}{=} & \forall b(a \mapsto^* b \;\Rightarrow b \mapsto) & \text{`}a \text{ diverges'} \end{array}$$

We have $a{\Downarrow}$ iff not $a{\Uparrow}$. We have recovered Abadi and Cardelli's many-step evaluation relation; $a \Downarrow b$ in our notation corresponds to $\varnothing \vdash a \rightsquigarrow b$ in theirs. The proof is standard.

We introduced $\mathbf{FOb}_{\mathbf{1}<:\mu}$ without recursive programs, but we can define them using objects; for example, let $\mu(x{:}A)e$ abbreviate the expression $[\ell = \varsigma(s{:}[\ell{:}A])e[^{s.\ell}\!/_{x}]].\ell$. We have $\mu(x{:}A)e \mapsto e[^{\mu(x:A)e}\!/_{x}]$. Hence there is a divergent program at every type. Let Ω^A be $\mu(x{:}A)x$; $\Omega^A \mapsto \Omega^A$ so $\Omega^A{\Uparrow}$. The definability of Ω^A and the contractivity condition on recursive types together imply the following lemma, which guarantees a value and a divergent program at every type.

Lemma 8 $\forall A \in \mathit{Type}\ \exists a_1, a_2{:}A\ (a_1{\Downarrow}\ \&\ a_2{\Uparrow})$.

3 Contextual equivalence

We take Morris' contextual equivalence (Morris 1968; Plotkin 1977), also known as 'observational congruence' (Meyer and Cosmadakis 1988), to be the natural operational equivalence on objects. First we introduce the idea of a relation between expressions of matching types. Let a **proved program** be a pair a_A such that $a{:}A$. Let P and Q range over proved programs. Let *Rel* be the universal relation on proved programs of the same type, given as follows.

$$Rel \stackrel{\text{def}}{=} \{(a_A, b_A) \mid a{:}A \;\&\; b{:}A\}$$

We use $\mathcal{R}$ and $\mathcal{S}$ for subsets of *Rel*, that is, relations on proved programs that respect typing. If $\mathcal{R} \subseteq Rel$, then for any type A, define $\mathcal{R}_A = \{(a, b) \mid (a_A, b_A) \in \mathcal{R}\}$. So the notation '$a\,\mathcal{R}_A\,b$' means that $(a_A, b_A) \in \mathcal{R}$.

Instead of using relations on proved programs, we could have used binary relations on expressions indexed by types. We use proved programs because when defining bisimilarity we can work simply in the complete lattice of subsets of *Rel* ordered by $\subseteq$, rather than of indexed sets.

For each closed type A, let **A-contextual equivalence**, $\overset{A}{\simeq} \subseteq Rel$, be the relation on proved programs given by the following.

$$a \overset{A}{\simeq}_B b \quad \text{iff} \quad \text{whenever } -{:}B \vdash e : A,\ e[a] \Downarrow \text{ iff } e[b] \Downarrow \text{ too.}$$

In other words, two programs are A-contextually equivalent iff their behaviour is the same whenever they are placed in a larger program, e, of type A.

Proposition 9

(1) *Suppose there exists a context e such that $-{:}A \vdash e : B$ and for all programs $a{:}A$, we have $a\Downarrow$ iff $e[a]\Downarrow$. Then $\overset{B}{\simeq} \subseteq \overset{A}{\simeq}$.*

(2) *If $A <: B$ then $\overset{B}{\simeq} \subseteq \overset{A}{\simeq}$.*

(3) *For all A, $\overset{\texttt{Top}}{\simeq} \subseteq \overset{A}{\simeq}$.*

(4) *For all B, $\overset{B}{\simeq} \subseteq \overset{\texttt{Bool}}{\simeq}$.*

(5) $\overset{\texttt{Top}}{\simeq} \subset \overset{\texttt{Bool}}{\simeq}$.

Proof

(1) Suppose for some some type C and programs a and b, we have $a \overset{B}{\simeq}_C b$. We must prove that $a \overset{A}{\simeq}_C b$. Suppose for some context e' that $-{:}C \vdash e' : A$. By symmetry, it is enough to show that if $e'[a]\Downarrow$ then $e'[b]\Downarrow$. Consider the context $e[e'/-]$ satisfying $-{:}C \vdash e[e'/-] : B$, where e is as given in the statement of the proposition. Then $e[e'[a]]\Downarrow$, and since $a \overset{B}{\simeq}_C b$, we have $e[e'[b]]\Downarrow$; hence $e'[b]\Downarrow$ as required.

(2) A corollary of part (1), taking $e \equiv -$ and using subsumption.

(3) A corollary of part (2), since $A <: \mathtt{Top}$.

(4) A corollary of part (1), taking $e \equiv \mathtt{if}(-, v_B, v_B)$ for some value v_B of type B (we know such a value must exist by Lemma 8).

(5) We have $\neg(\mathtt{true} \overset{\mathtt{Top}}{\simeq}_{\mathtt{Top}} \Omega^{\mathtt{Top}})$ but $\mathtt{true} \overset{\mathtt{Bool}}{\simeq}_{\mathtt{Top}} \Omega^{\mathtt{Top}}$. The former is immediate; the latter is trivial once we have Theorem 2. ■

Of this family of equivalence relations, $\overset{\mathtt{Top}}{\simeq}$ makes the most distinctions between programs, and $\overset{\mathtt{Bool}}{\simeq}$ the fewest. The only substantial difference among the relations is the set of types at which termination is distinguishable from non-termination.

In a call-by-value language, we can construct a context satisfying part (1) for any two types A and B, namely $((\lambda(x{:}A)v_B)\,-)$ where v_B is some value of type B. Hence, in a call-by-value language, the $\overset{A}{\simeq}$'s are equal.

We choose to take $\overset{\mathtt{Bool}}{\simeq}$ as our notion of operational equivalence for this study for three reasons. First, it is the notion of contextual equivalence used by Plotkin (1977) and is standard in studies of the language PCF. Second, it is the most generous of the A-contextual equivalences. Third, since one of our motivations is to validate the equational theory of Abadi and Cardelli, we must choose a contextual equivalence in which their equations are sound. In particular, the rule

$$\frac{\Gamma \vdash e : A \quad \Gamma \vdash e' : B}{\Gamma \vdash e \leftrightarrow e' : \mathtt{Top}}\ (\text{Eq Top})$$

must hold, so $\overset{\mathtt{Top}}{\simeq}$ is inappropriate. We will see later that (Eq Top) holds for $\overset{\mathtt{Bool}}{\simeq}$.

Another candidate was $\overset{[]}{\simeq}$, which does satisfy (Eq Top) and would be the most natural choice in the language without Booleans. It is finer-grained that $\overset{\mathtt{Bool}}{\simeq}$; it distinguishes $[\,]$ and $\Omega^{[]}$ whereas $\overset{\mathtt{Bool}}{\simeq}$ identifies them.

We will use **contextual equivalence**, $\simeq \subseteq \mathit{Rel}$, to stand for $\overset{\texttt{Bool}}{\simeq}$, and define **contextual order**, $\precsim \subseteq \mathit{Rel}$, as follows.

$$a \precsim_A b \quad \text{iff} \quad \text{whenever } -:A \vdash e : \texttt{Bool},\ e[a] \Downarrow \text{ implies } e[b] \Downarrow \text{ too.}$$

Note that $a \simeq_A b$ iff $a \precsim_A b$ and $b \precsim_A a$.

It is easy to show that two programs are contextually distinct: just exhibit a single context that tells them apart. But to show equivalence requires a quantification over all contexts. The point of the next section is to characterise contextual equivalence co-inductively as a kind of bisimilarity, and hence to admit CCS-style bisimulation proofs of equivalence. Contextual equivalence is defined in terms of one-off tests consisting of composite contexts; bisimilarity is defined in terms of multiple atomic observations on objects. When proving programs equal it is often easier to consider a series of atomic observations rather than all possible contexts.

4 Bisimilarity

4.1 Labelled transitions

We will define a labelled transition system that characterises the atomic observations one can make of a proved program. The notation $P \xrightarrow{\alpha} Q$ means that the proved program P does an **action** α and becomes another proved program Q. As usual we write $P \xrightarrow{\alpha}$ mean that there is some Q with $P \xrightarrow{\alpha} Q$.

The simplest labelled transition system to characterise contextual equivalence co-inductively is the following.

$$\frac{-{:}A \vdash \mathcal{E} : B}{a_A \xrightarrow{\mathcal{E}} \mathcal{E}[a]_B} \text{ (Trans Exper)} \qquad \frac{a{\Downarrow}}{a_{\texttt{Bool}} \xrightarrow{\texttt{val}} \mathbf{0}} \text{ (Trans Val)}$$

where an action is either an experiment, $\mathcal{E}$, or `val`, and $\mathbf{0}$ is disjoint from the set of programs. We could prove that CCS-style bisimilarity according to this labelled transition system equals contextual equivalence. This is a direct generalisation of Milner's context lemma for PCF (Milner 1977). We can do better than this by describing a labelled transition system in which there are fewer transitions available to proved programs: this reduction in size will simplify some of our proofs.

We divide the types of $\mathbf{FOb_{1<:\mu}}$ into two classes, **active** and **passive**. Only `Bool` is active. `Top`, object types, recursive types and function types are passive. At active types a program must converge to a value before it can be observed; at passive types a program does actions unconditionally, whether or not it converges. The observable actions, $\alpha \in Act$, take the following forms.

$$\texttt{true} \quad \texttt{false} \quad \ell \quad \ell \Leftarrow \varsigma(x)e \quad \texttt{unfold} \quad @a$$

These actions correspond to the actions of the general labelled transition system, except that actions of the form $\texttt{if}(-, a, b)$ are missing, and type annotations are missing from the update actions.

The labelled transition system we shall work with is the family of relations $(\xrightarrow{\alpha} \mid \alpha \in Act)$ given inductively by the rules in Table 7, such that whenever $P \xrightarrow{\alpha} Q$, each of P and Q is a proved program. Let $\mathbf{0}$ be $a_{\texttt{Top}}$, for some arbitrary program $a{:}\texttt{Top}$. The purpose of $\mathbf{0}$ is that it has no actions; after observing ground data there is nothing more to observe.

We can characterise the observable actions at each type. For each type A, define the set $Act(A) \subseteq Act$ as follows.

$$Act(\texttt{Top}) \;=\; \{\}$$

$$\frac{v \in \{\texttt{true}, \texttt{false}\}}{v_{\texttt{Bool}} \xrightarrow{v} \mathbf{0}} \text{ (Trans Bool)}$$

$$\frac{A \equiv [\ell_i{:}A_i]_{i \in I} \quad j \in I}{a_A \xrightarrow{\ell_j} a.\ell_j{}_{A_j}} \text{ (Trans Select)}$$

$$\frac{A \equiv [\ell_i{:}A_i]_{i \in I} \quad j \in I \quad x{:}A \vdash e : A_j}{a_A \xrightarrow{\ell_j \Leftarrow \varsigma(x)e} a.\ell_j \Leftarrow \varsigma(x{:}A)e_A} \text{ (Trans Update)}$$

$$\frac{A \equiv \mu(X)E}{a_A \xrightarrow{\texttt{unfold}} \texttt{unfold}(a)_{E[A/X]}} \text{ (Trans Unfold)}$$

$$\frac{b{:}B}{a_{B \to A} \xrightarrow{@b} (a\, b)_A} \text{ (Trans App)}$$

$$\frac{A \text{ active} \qquad a \mapsto a' \qquad a'_A \xrightarrow{\alpha} b_B}{a_A \xrightarrow{\alpha} b_B} \text{ (Trans Red)}$$

Table 7: Rules of the labelled transition system

$$\begin{array}{rcl} Act(\texttt{Bool}) & = & \{\texttt{true}, \texttt{false}\} \\ Act([\ell_i{:}A_i]_{i\in I}) & = & \{\ell_i, \ell_i \Leftarrow \varsigma(x)e \mid i \in I \;\&\; x{:}[\ell_i{:}A_i]_{i\in I} \vdash e : A_i\} \\ Act(\mu(X)E) & = & \{\texttt{unfold}\} \\ Act(A \to B) & = & \{@a \mid a{:}A\} \end{array}$$

Lemma 10 *$\alpha \in Act(A)$ iff $\exists a{:}A(a_A \xrightarrow{\alpha})$.*

We can make more observations at a subtype than a supertype.

Lemma 11 *If $A <: B$ then $Act(B) \subseteq Act(A)$.*

An alternative formulation of a labelled transition system for $\mathbf{FOb}_{\mathbf{1}<:\mu}$ would be to remove the rules (Trans Red) and (Trans Bool) and substitute instead the following rule.

$$\frac{a \Downarrow v \quad v \in \{\texttt{true}, \texttt{false}\}}{a_{\texttt{Bool}} \xrightarrow{v} \mathbf{0}} \text{ (Trans Bool')}$$

However, we adopt the more general rule (Trans Red) so that when we introduce more active types into the language, the general rule will apply to them all. In particular, the following lemma must hold in any extension of this system.

The following is a trivial fact for $\mathbf{FOb}_{\mathbf{1}<:\mu}$, as `Bool` is the only active type, but it holds in the extensions of $\mathbf{Ob}_{\mathbf{1}<:\mu}$ we will consider in Section 7, in which more types are active.

Lemma 12 *If A is active then $a_A \xrightarrow{\alpha} b_B$ iff $\exists v \in Value(a \Downarrow v \;\&\; v_A \xrightarrow{\alpha} b_B)$.*

At passive types, divergent programs have actions, but they always lead to divergent programs.

Lemma 13 *If $a{:}A$, $a\Uparrow$ and $a_A \xrightarrow{\alpha} b_B$ then $b\Uparrow$.*

The actions on passive types are almost in the form of experiments. We introduce the idea of erasure on experiments, $\mathcal{E}\downarrow$, to make this precise.

$$\begin{array}{rcl} (-.\ell)\downarrow & = & \ell \\ (-.\ell \Leftarrow \varsigma(x{:}A)e)\downarrow & = & \ell \Leftarrow \varsigma(x)e \\ (\texttt{unfold}(-))\downarrow & = & \texttt{unfold} \\ (-\,a)\downarrow & = & @a \end{array}$$

The erasure operation removes the type annotation from the update action, ensuring that actions are uniform at different types. This property is required for the proof of Lemma 11. Given the erasure operator, we can recover a limited form of the rule (Trans Exper) for passive types.

Lemma 14 *If A is passive then $a_A \xrightarrow{\alpha} b_B$ implies $\exists \mathcal{E}(\alpha \equiv \mathcal{E}\downarrow \,\&\, b \equiv \mathcal{E}[a])$.*

Hence, the transitions at passive type in our labelled transition system correspond to the ones derivable from the (Trans Exper) rule, except that the latter incorporate subsumption. Our labelled transition system is image-singular, in the following sense.

Lemma 15 *If $P \xrightarrow{\alpha} Q$ and $P \xrightarrow{\alpha} Q'$, then $Q \equiv Q'$.*

Because of subsumption, the more general system is not image-singular. For example, the following two transitions are derivable from (Trans Exper).

$$\begin{array}{lcl} a_{[\ell:\mathtt{Bool}]} & \xrightarrow{-.\ell} & a.\ell_{\mathtt{Bool}} \\ a_{[\ell:\mathtt{Bool}]} & \xrightarrow{-.\ell} & a.\ell_{\mathtt{Top}} \end{array}$$

4.2 Definition of bisimilarity

The **derivation tree** of a proved program P is the potentially infinite tree whose nodes are proved programs, whose arcs are labelled transitions, and which is rooted at P. Following Milner (1989), we wish to regard two proved programs as behaviourally equivalent iff their derivation trees are the same when we ignore the syntactic structure of the programs labelling the nodes and the ordering of the arcs from each node. We formalise this idea in the standard way. First define two functions $[-], \langle - \rangle : \wp(Rel) \to \wp(Rel)$ by

$$\begin{array}{lcl} [\mathcal{S}] & \stackrel{\text{def}}{=} & \{(P,Q) \mid \text{whenever } P \xrightarrow{\alpha} P' \text{ there is } Q' \\ & & \qquad \text{with } Q \xrightarrow{\alpha} Q' \text{ and } P' \,\mathcal{S}\, Q'\} \\ \langle \mathcal{S} \rangle & \stackrel{\text{def}}{=} & [\mathcal{S}] \cap [\mathcal{S}^{\text{op}}]^{\text{op}} \end{array}$$

These are both monotone functions on $\wp(Rel)$. Let a relation $\mathcal{S} \subseteq Rel$ be a **bisimulation** iff $\mathcal{S} \subseteq \langle \mathcal{S} \rangle$. Let **bisimilarity**, $\sim \subseteq Rel$, be the union of all the bisimulations. By the Tarski–Knaster theorem, bisimilarity is the greatest fixpoint of $\langle - \rangle$. In other words, bisimilarity is the greatest relation to satisfy the following: whenever $(P,Q) \in Rel$, $P \sim Q$ iff

(1) $P \xrightarrow{\alpha} P' \Rightarrow \exists Q'(Q \xrightarrow{\alpha} Q' \,\&\, P' \sim Q')$

(2) $Q \xrightarrow{\alpha} Q' \Rightarrow \exists P'(P \xrightarrow{\alpha} P' \,\&\, P' \sim Q')$.

If $\mathcal{S}$ is a bisimulation, $\mathcal{S} \subseteq \sim$ by definition of $\sim$; this is the co-induction principle associated with bisimilarity.

We shall need the preorder form of bisimilarity. Let relation $\mathcal{S} \subseteq Rel$ be a **simulation** iff $\mathcal{S} \subseteq [\mathcal{S}]$. **Similarity**, $\lesssim \subseteq Rel$, is the greatest fixpoint of $[-]$, that is, the union of all simulations.

4.3 Basic properties of bisimilarity

We can easily establish the following using co-induction.

Proposition 16

(1) $\lesssim$ *is a preorder and* $\sim$ *an equivalence relation.*

(2) $\sim = \lesssim \cap \lesssim^{\mathrm{op}}$.

Part (2) depends on image-singularity, Lemma 15. This property fails in a nondeterministic calculus such as CCS.

4.4 Bisimilarity and subtyping

Whenever $A <: B$, we would expect that if two programs are equal at the subtype, A, that they will be equal at the supertype, B. To prove this, we need the following lemma.

Lemma 17 *Relation* $\{(a_B, b_B) \mid \exists A(\varnothing \vdash A <: B \;\&\; a \sim_A b)\}$ *is a simulation.*

The following is a simple corollary by co-induction.

Proposition 18

(1) *If* $a \lesssim_A b$ *and* $A <: B$ *then* $a \lesssim_B b$.

(2) *If* $a \sim_A b$ *and* $A <: B$ *then* $a \sim_B b$.

In the following section we introduce the idea of a relation being a congruence, and in the next we prove that bisimilarity is one.

4.5 Congruence and precongruence

A congruence is an equivalence that is preserved by all contexts. To state this formally we must begin with a few preliminary definitions. Let a **substitution** be a function $\vec{\sigma} \stackrel{\mathrm{def}}{=} a_1/x_1, \ldots, a_n/x_n$ $(n \geq 0)$ from expressions to expressions, which substitutes programs for free variables. The application of a substitution $\vec{\sigma}$ to an expression e is written $e[\vec{\sigma}]$. A substitution $\vec{\sigma} \equiv a_1/x_1, \ldots, a_n/x_n$ is a Γ-closure for an environment $\Gamma \equiv x_1{:}A_1, \ldots, x_n{:}A_n$ iff each $a_i{:}A_i$. Let a **proved expression** be a triple (Γ, e, A) such that Γ is 'closed' (it contains no type variable bounds; that is, it only contains assignments of closed types to program variables), $A \in \mathit{Type}$ and $\Gamma \vdash e : A$. If the relation $\mathcal{R} \subseteq \mathit{Rel}$ then its **open extension**, $\mathcal{R}^\circ$, is the relation on proved expressions such that $(\Gamma, e, A)\,\mathcal{R}^\circ\,(\Gamma', e', A')$ iff $A \equiv A'$, $\Gamma \equiv \Gamma'$ and $e[\vec{\sigma}]\,\mathcal{R}\,e'[\vec{\sigma}]$ for all Γ-closures $\vec{\sigma}$. Open extension is a monotonic operator on relations.

Lemma 19 *If $\mathcal{R} \subseteq \mathcal{S}$ then $\mathcal{R}^\circ \subseteq \mathcal{S}^\circ$.*

For instance, Rel° is the universal relation on pairs of proved expressions with matching types and environments. As a notational convention, if $\mathcal{R} \subseteq Rel^\circ$ we write $\Gamma \vdash e \,\mathcal{R}\, e' : A$ to mean that $((\Gamma, e, A), (\Gamma, e', A)) \in \mathcal{R}$.

If $\mathcal{R} \subseteq Rel^\circ$ then its **compatible refinement** (Gordon 1994) is the relation $\widehat{\mathcal{R}} \subseteq Rel^\circ$, given by the rules in Table 8, that relates two expressions if they share the same outermost syntactic constructor, and their immediate sub-expressions are pairwise related by $\mathcal{R}$. Each compatible refinement rule corresponds to a type assignment rule, and vice versa, except for (Val Subsumption). We say a relation $\mathcal{R} \subseteq Rel^\circ$ is a **precongruence** iff it satisfies the following two rules.

$$\frac{\Gamma \vdash e \leftrightarrow e' : A \qquad A <: B}{\Gamma \vdash e \leftrightarrow e' : B}\ (\text{Eq Subsum}) \qquad \frac{\Gamma \vdash e \mathrel{\widehat{\leftrightarrow}} e' : A}{\Gamma \vdash e \leftrightarrow e' : A}\ (\text{Eq Comp})$$

This definition of precongruence can easily be shown equivalent to a more conventional one based on substitution into variable-capturing contexts. If, in addition, a precongruence is an equivalence relation, we say it is a **congruence**.

Proposition 20 *Suppose $\leftrightarrow$ is a preorder. Then satisfaction of* (Eq Comp) *and* (Eq Subsum) *is equivalent to satisfaction of the following rule*

$$\frac{\Gamma \vdash \mathcal{C}[\bullet_A] : B \qquad \Gamma, \Gamma' \vdash e \leftrightarrow e' : A}{\Gamma \vdash \mathcal{C}[e] \leftrightarrow \mathcal{C}[e'] : B}\ (\text{Eq Cong})$$

where $\mathcal{C}$ is a variable-capturing context, that is, an expression with a single hole of the form $\bullet_A$ (subject to the extra type assignment axiom $\Gamma \vdash \bullet_A : A$), with the hole in the scope of binders for the environment Γ'.

4.6 Bisimilarity is a congruence

We will show that the open extension of bisimilarity is a congruence. Since bisimilarity is the symmetrisation of similarity, Proposition 16(2), it is enough to prove that similarity is a precongruence. We do so using a form of Howe's method (1989). We define an auxiliary relation, $\lesssim^\bullet$, which by definition is a precongruence, and prove that $\lesssim^\bullet = \lesssim^\circ$. Let the **precongruence candidate**, $\lesssim^\bullet \subseteq Rel^\circ$, be the least relation closed under the following rule.

$$\frac{\Gamma \vdash e \mathrel{\widehat{\lesssim^\bullet}} e'' : A \quad A <: B \quad \Gamma \vdash e'' \lesssim^\circ e' : B}{\Gamma \vdash e \lesssim^\bullet e' : B}\ (\text{Cand Def})$$

$$\frac{}{\Gamma, x{:}A, \Gamma' \vdash x \mathrel{\widehat{\mathcal{R}}} x : A}\ (\text{Comp } x)$$

$$\frac{}{\Gamma \vdash \mathtt{true} \mathrel{\widehat{\mathcal{R}}} \mathtt{true} : \mathtt{Bool}}\ (\text{Comp True})$$

$$\frac{}{\Gamma \vdash \mathtt{false} \mathrel{\widehat{\mathcal{R}}} \mathtt{false} : \mathtt{Bool}}\ (\text{Comp False})$$

$$\frac{\Gamma \vdash e_1 \mathrel{\mathcal{R}} e_1' : \mathtt{Bool} \quad \Gamma \vdash e_i \mathrel{\mathcal{R}} e_i' : A \quad (2 \leq i \leq 3)}{\Gamma \vdash \mathtt{if}(e_1, e_2, e_3) \mathrel{\widehat{\mathcal{R}}} \mathtt{if}(e_1', e_2', e_3') : A}\ (\text{Comp If})$$

$$\frac{A \equiv [\ell_i{:}B_i] \quad \Gamma, x_i{:}A \vdash e_i \mathrel{\mathcal{R}} e_i' : B_i\ (\forall i \in I)}{\Gamma \vdash [\ell_i = \varsigma(x_i{:}A)e_i]_{i \in I} \mathrel{\widehat{\mathcal{R}}} [\ell_i = \varsigma(x_i{:}A)e_i']_{i \in I} : A}\ (\text{Comp Object})$$

$$\frac{\Gamma \vdash e \mathrel{\mathcal{R}} e' : [\ell_i{:}A_i]_{i \in I} \quad j \in I}{\Gamma \vdash e.\ell_j \mathrel{\widehat{\mathcal{R}}} e'.\ell_j : A_j}\ (\text{Comp Select})$$

$$\frac{A \equiv [\ell_i{:}B_i] \quad \Gamma \vdash e_1 \mathrel{\mathcal{R}} e_1' : A \quad \Gamma, x{:}A \vdash e_2 \mathrel{\mathcal{R}} e_2' : B_j \quad j \in I}{\Gamma \vdash e_1.\ell_j \Leftarrow \varsigma(x{:}A)e_2 \mathrel{\widehat{\mathcal{R}}} e_1'.\ell_j \Leftarrow \varsigma(x{:}A)e_2' : A}\ \left(\begin{array}{c}\text{Comp}\\ \text{Update}\end{array}\right)$$

$$\frac{A \equiv \mu(X)E \quad \Gamma \vdash e \mathrel{\mathcal{R}} e' : E[{}^{A}\!/_{X}]}{\Gamma \vdash \mathtt{fold}(A, e) \mathrel{\widehat{\mathcal{R}}} \mathtt{fold}(A, e') : A}\ (\text{Comp Fold})$$

$$\frac{A \equiv \mu(X)E \quad \Gamma \vdash e \mathrel{\mathcal{R}} e' : A}{\Gamma \vdash \mathtt{unfold}(e) \mathrel{\widehat{\mathcal{R}}} \mathtt{unfold}(e') : E[{}^{A}\!/_{X}]}\ (\text{Comp Unfold})$$

$$\frac{\Gamma, x{:}B \vdash e \mathcal{R} e' : A}{\Gamma \vdash \lambda(x{:}B)e \mathrel{\widehat{\mathcal{R}}} \lambda(x{:}B)e' : B \to A}\ (\text{Comp Fun})$$

$$\frac{\Gamma \vdash e_1 \mathcal{R} e_1' : B \to A \quad \Gamma \vdash e_2 \mathcal{R} e_2' : B}{\Gamma \vdash (e_1\, e_2) \mathrel{\widehat{\mathcal{R}}} (e_1'\, e_2') : A}\ (\text{Comp App})$$

Table 8: Compatible refinement

Since $\lesssim^\bullet$ is defined by exactly one rule, it is valid upwards, that is, whenever $\Gamma \vdash e \lesssim^\bullet e' : B$, there is some expression e'' and some type A with $\Gamma \vdash e \widehat{\lesssim^\bullet} e'' : A$ and $A <: B$ and also $\Gamma \vdash e'' \lesssim^\circ e' : B$. We can easily prove the following properties of $\lesssim^\bullet$ by standard methods.

Lemma 21 *Relation $\lesssim^\bullet$ is reflexive, and the following rules are valid.*

$$\frac{\Gamma \vdash e \lesssim^\circ e' : A}{\Gamma \vdash e \lesssim^\bullet e' : A}\ \text{(Cand Sim)} \qquad \frac{\Gamma \vdash e \widehat{\lesssim^\bullet} e' : A}{\Gamma \vdash e \lesssim^\bullet e' : A}\ \text{(Cand Comp)}$$

$$\frac{\Gamma \vdash e \lesssim^\bullet e'' : A \quad \Gamma \vdash e'' \lesssim^\circ e' : A}{\Gamma \vdash e \lesssim^\bullet e' : A}\ \text{(Cand Right)}$$

$$\frac{\begin{array}{l}\Gamma, x{:}B \vdash e_1 \lesssim^\bullet e_1' : A \\ \Gamma \vdash e_2 \lesssim^\bullet e_2' : B\end{array}}{\Gamma \vdash e_1[e_2/x] \lesssim^\bullet e_1'[e_2'/x] : A}\ \text{(Cand Subst)}$$

Moreover, $\lesssim^\bullet$ is the least relation closed under (Cand Comp), (Cand Right) *and* (Eq Subsum).

Unlike previous applications of Howe's method, we need to prove that $\lesssim^\bullet$ satisfies (Eq Subsum). Given that (Eq Subsum) holds for similarity, Proposition 18, the following new properties are easy to prove.

Lemma 22 *Both* (Eq Subsum) *and the rule*

$$\frac{A <: B \quad \Gamma, x{:}B, \Gamma' \vdash e \leftrightarrow e' : C}{\Gamma, x{:}A, \Gamma' \vdash e \leftrightarrow e' : C}\ \text{(Eq Asm Subsum)}$$

hold for $\lesssim^\circ$ and $\lesssim^\bullet$.

The following lemma is the heart of the precongruence proof. The proof is detailed but follows the standard pattern.

Lemma 23 *Relation $\mathcal{S} = \{(a_A, {a'}_A) \mid \varnothing \vdash a \lesssim^\bullet a' : A\}$ is a simulation.*

Theorem 1 *The open extension of bisimilarity is a congruence.*

Proof By Lemma 23, $\mathcal{S}$ is a simulation, and hence $\mathcal{S} \subseteq \lesssim$ by co-induction. Open extension is monotone, Lemma 19, so $\mathcal{S}^\circ \subseteq \lesssim^\circ$. Now $\lesssim^\bullet \subseteq \mathcal{S}^\circ$ follows by (Cand Subst) and the reflexivity of $\lesssim^\bullet$. Hence we have $\lesssim^\bullet \subseteq \lesssim^\circ$. But (Cand Sim) provides the reverse inclusion, so in fact $\lesssim^\bullet = \lesssim^\circ$ and hence $\lesssim^\circ$ is a precongruence. By appeal to symmetry, $\sim^\circ$ is a congruence. ■

4.7 Bisimilarity equals contextual equivalence

The proof of our main result, Theorem 2, follows the standard pattern.

Lemma 24 *Both* $\lesssim \subseteq \sqsubseteq_{\lesssim}$ *and* $\sim \subseteq \simeq$.

Lemma 25 *Contextual order,* $\sqsubseteq_{\lesssim}$*, is a simulation.*

Theorem 2 $\sim = \simeq$.

Proof Apply co-induction to Lemma 25 and combine with Lemma 24. ■

4.8 Soundness of reduction

The relationship between operational semantics and equivalence in $\mathbf{FOb}_{\mathbf{1}<:\mu}$ is subtle. The following facts are straightforward to state and prove.

Proposition 26 *For any type* A*,*

(1) $\forall a, b{:}A(a \mapsto b \Rightarrow a \sim_A b)$*;*

(2) $\forall a, v{:}A(a \Downarrow v \Rightarrow a \sim_A v)$*;*

(3) $\forall a{:}A(a\Uparrow \Rightarrow a \sim_A \Omega^A)$.

We postpone a classification of the types for which part (3) can be strengthened from $\Rightarrow$ to iff until Section 8.

$$\frac{A' \equiv [\ell_i{:}B_i]_{i\in I} \quad A'' \equiv [\ell_i{:}B_i, \ell_j{:}B_j]_{i\in I, j\in J} \quad I \cap J = \varnothing \quad \Gamma, x_i{:}A' \vdash e_i : B_i\,(i \in I) \quad \Gamma, x_j{:}A'' \vdash e_j : B_j\,(j \in J)}{\Gamma \vdash [\ell_i = \varsigma(x_i{:}A')e_i]_{i\in I} \leftrightarrow [\ell_i = \varsigma(x_i{:}A'')e_i]_{i\in I\cup J} : A'} \text{ (Eq Sub Object)}$$

$$\frac{A \equiv \mu(X)E \quad \Gamma \vdash e : A}{\Gamma \vdash \mathtt{fold}(A, \mathtt{unfold}(e)) \leftrightarrow e : A} \text{ (Eval Fold)}$$

$$\frac{\Gamma \vdash e : A \to B \quad x \notin \mathit{Dom}(\Gamma)}{\Gamma \vdash \lambda(x{:}A)(e\,x) \leftrightarrow e : A \to B} \text{ (Eval Eta)}$$

Table 9: Fragment of the equational theory of $\mathbf{FOb}_{\mathbf{1}<:\mu}$

5 Validating the equational theory

Abadi and Cardelli (1994c) present an equational theory for $\mathbf{FOb}_{\mathbf{1}<:\mu}$. Their relation is essentially the relation $\leftrightarrow \subseteq \mathit{Rel}^\circ$ that is inductively defined by the rules in Table 9, together with rules of equivalence, compatibility, closure under evaluation, (Eq Top) and (Eq Subsum). The theory is given in full in Appendix B. We show that the open extension of bisimilarity is closed under the relevant equational rules and hence $\leftrightarrow \subseteq \sim^\circ$.

Most of the $\mathbf{FOb}_{\mathbf{1}<:\mu}$ equational theory is easy to verify. The equivalence rules follow for bisimilarity follow from Proposition 16 (1). The congruence rules follow directly from Theorem 1. The evaluation rules follow from Proposition 26. (Eq Top) follows from Proposition 31(1). (Eq Subsum) follows from Lemma 22. (Eval Fold) and (Eval Eta) can easily be proved by co-induction. (Eq Sub Object) appears to be most easily proved via a direct proof that the two objects are contextually equivalent. We give the full proofs in Appendix A.6. Since $\sim^\circ$ is closed under all the rules inductively defining the equational theory, it is sound in the following sense.

Theorem 3 $\leftrightarrow \subseteq \sim^\circ$.

Bisimilarity is no panacea—witness the direct proof of (Eq Sub Object)—but the bisimulation proofs of most of the equational rules would appear to be simpler than direct proofs of contextual equivalence.

The converse of Theorem 3 fails; see Proposition 29 below for an example. In any case since the calculus presented here is Turing-powerful, no

recursively enumerable equational theory such as $\leftrightarrow$ could be complete for operational equivalence.

Strictly speaking, the theory of this paper does not cover all of Abadi and Cardelli's equations because their equational theory covers open expressions of open type. We have only considered open expressions of closed type. Open expressions of open type are not useful in $\mathbf{FOb}_{\mathbf{1}<:\mu}$ because there are no polymorphic functions (though of course such expressions are useful in richer systems considered by Abadi and Cardelli). It should be possible to extend our notion of open extension to include expressions of open type, and correspondingly to extend the compatible refinement and congruence candidate relations and thus derive the exact forms of their equations. We postpone this until a future study of polymorphism.

6 Example

We consider an example given by Abadi and Cardelli. A **field** is a degenerate method that does not depend on its self parameter. Let $e'.\ell := e$ be short for $e'.\ell \Leftarrow \varsigma(x)e$, and $[\ell = e, \ldots]$ be short for $[\ell = \varsigma(x{:}A)e, \ldots]$, for some $x \notin fv(e)$. Define a type A and two objects a and b as follows.

$$\begin{aligned} A &\stackrel{\text{def}}{=} [x{:}\mathtt{Bool}, f{:}\mathtt{Bool}] \\ a{:}A &\stackrel{\text{def}}{=} [x = \mathtt{true}, f = \mathtt{true}] \\ b{:}A &\stackrel{\text{def}}{=} [x = \mathtt{true}, f = \varsigma(s{:}A)s.x] \end{aligned}$$

Proposition 27 *Not* $a \sim_A b$.

Proof Let the context e be $-.x := \Omega^{\mathtt{Bool}}.f$. Both $e[a]$ and $e[b]$ are programs of type $\mathtt{Bool}$, but $e[a] \Downarrow \mathtt{true}$ whereas $e[b]\Uparrow$. Hence the two are contextually distinct, therefore not bisimilar. ■

Proposition 28 $a \sim_{[x:\mathtt{Bool}]} b$.

Proof Using (Eq Sub Object) we can prove both $[x = \mathtt{true}] \sim_{[x:\mathtt{Bool}]} a$ and $[x = \mathtt{true}] \sim_{[x:\mathtt{Bool}]} b$, and therefore $a \sim_{[x:\mathtt{Bool}]} b$ by transitivity. ■

Proposition 29 $a \sim_{[f:\mathtt{Bool}]} b$.

Proof Let P and Q be $a_{[f:\mathtt{Bool}]}$ and $b_{[f:\mathtt{Bool}]}$ respectively. Here are all their possible transitions.

(1) $P \xrightarrow{f} P'$ with $P' \equiv (a.f)_{\mathtt{Bool}} \sim \mathtt{true}_{\mathtt{Bool}}$.

(2) $Q \xrightarrow{f} P'$ with $P' \equiv (b.f)_{\mathtt{Bool}} \sim \mathtt{true}_{\mathtt{Bool}}$.

(3) $P \xrightarrow{f \Leftarrow \varsigma(x)e} P'$ with $P' \sim [x = \mathtt{true}, f = \varsigma(x{:}A)e]_{[f:\mathtt{Bool}]}$.

(4) $Q \xrightarrow{f \Leftarrow \varsigma(x)e} Q'$ with $Q' \sim [x = \mathtt{true}, f = \varsigma(x{:}A)e]_{[f:\mathtt{Bool}]}$.

In each case, whenever $P \xrightarrow{\alpha} P'$ there is Q' with $Q \xrightarrow{\alpha} Q'$ and $P' \sim Q'$, and vice versa. Hence $(P, Q) \in \langle\sim\rangle$ and since $\sim = \langle\sim\rangle$ we have $a \sim_{[f:\mathtt{Bool}]} b$. ■

This example, in a form using natural numbers rather than Booleans, is given in section 4.3 of Abadi and Cardelli (1994c). Proposition 29 does not follow from the equational theory $\leftrightarrow$. We expect it would follow by a direct proof of contextual equivalence (similar to the one we needed for (Eq Sub Object)) but the bisimulation proof above is much simpler.

7 Extensions

Each lemma in the proof of Theorem 2 is a property of a relation which is proved by induction or co-induction on some set of rules. If a new syntactical construct is added to the language, we must check that each property is unaffected; this amounts to adding a single case to each induction or co-induction argument.

To demonstrate the modularity of our approach, we extend $\mathbf{FOb}_{\mathbf{1}<:\mu}$ with the following first-order types.

- Type `Dynamic` (Table 10); values of this type are pairs of a programs with its type. The language here is a subset of that considered by Abadi, Cardelli, Pierce, and Plotkin (1989), but with subtyping. `Dynamic` type is active.

  The operational semantics of terms of dynamic type uses the subtyping relation to decide whether (Red Dynamic Match) or (Red Dynamic Default) applies. These rules thus depend on the decidability of subtyping; if we were to extend the type system to incorporate F$\leq$, in which subtyping is undecidable (Pierce 1994), we would have to choose a different destructor for terms of dynamic type.

  Our motivation for including dynamic types is to be able to derive more subtyping relations than would be possible using $\mathbf{FOb}_{\mathbf{1}<:\mu}$ only. Suppose `Nat` is a type of natural numbers. Given the two types

$$\begin{array}{rcl} A & \stackrel{\text{def}}{=} & \mu(X)[x{:}\mathtt{Nat}, dx{:}\mathtt{Nat} \to X] \\ B & \stackrel{\text{def}}{=} & \mu(X)[x, y{:}\mathtt{Nat}, dx, dy{:}\mathtt{Nat} \to X] \end{array}$$

  which represent moveable one-dimensional and two-dimensional points (the dx and dy functions take a natural number and return an object in which the appropriate co-ordinate has been updated), the expected subtyping $B <: A$ does not hold because of object invariance. Moreover, if that single subtyping relation were added to the language, the property of subject reduction property would fail (Abadi and Cardelli 1994c). However, we can represent these points using dynamic types, for example,

$$\begin{array}{rcl} A' & \stackrel{\text{def}}{=} & [x{:}\mathtt{Nat}, dx{:}\mathtt{Nat} \to \mathtt{Dynamic}] \\ B' & \stackrel{\text{def}}{=} & [x, y{:}\mathtt{Nat}, dx, dy{:}\mathtt{Nat} \to \mathtt{Dynamic}] \end{array}$$

  and now the relation $B' <: A'$ holds. However, the encoding is inelegant: every time we update a point, we have to use a `typecase` expression

$$\frac{}{\Gamma \vdash \texttt{Dynamic}}\ \text{(Type Dynamic)}$$

$$\frac{\Gamma \vdash e : A}{\Gamma \vdash \texttt{tag}(A, e) : \texttt{Dynamic}}\ \text{(Val Dynamic)}$$

$$\frac{\Gamma \vdash e : \texttt{Dynamic} \quad \Gamma, x{:}B \vdash e' : A \quad \Gamma \vdash e'' : A}{\Gamma \vdash \texttt{typecase}(e, (x{:}B)e', e'') : A}\ \text{(Val Typecase)}$$

$$\frac{A <: A'}{\texttt{typecase}(\texttt{tag}(A, e), (x{:}A')e', e'') \mapsto e'[e/x]}\ \text{(Red Dynamic Match)}$$

$$\frac{\neg(A <: A')}{\texttt{typecase}(\texttt{tag}(A, e), (x{:}A')e', e'') \mapsto e''}\ \text{(Red Dynamic Default)}$$

$$\frac{}{\texttt{tag}(A, e)_{\texttt{Dynamic}} \xrightarrow{\texttt{tag}A} e_A}\ \text{(Trans Dynamic)}$$

$$\frac{\Gamma \vdash e\ \widehat{\mathcal{R}}\ e' : A}{\Gamma \vdash \texttt{tag}(A, e)\ \widehat{\mathcal{R}}\ \texttt{tag}(A, e') : \texttt{Dynamic}}\ \text{(Comp Dynamic)}$$

$$\frac{\Gamma \vdash e_1 \mathcal{R} e_1' : \texttt{Dynamic} \quad \Gamma, x{:}B \vdash e_2 \mathcal{R} e_2' : A \quad \Gamma \vdash e_3 \mathcal{R} e_3' : A}{\Gamma \vdash \texttt{typecase}(e_1, (x{:}B)e_2, e_3)\ \widehat{\mathcal{R}}\ \texttt{typecase}(e_1', (x{:}B)e_2', e_3') : A}\ \left(\begin{array}{c}\text{Comp}\\ \text{Typecase}\end{array}\right)$$

$$\begin{array}{rcl} \mathcal{E} & ::= & \texttt{typecase}(-, (x{:}A)e', e'') \\ v & ::= & \texttt{tag}(A, e) \\ \alpha & ::= & \texttt{tag}A \in \mathit{Type} \end{array}$$

Table 10: Rules for dynamic types

to convert the resulting value from dynamic type back to an point. To represent these types elegantly, we need a type system that is 'polymorphic in self', such as that given in Abadi and Cardelli (1994b).

- **Records** (Table 11); these are essentially objects without method update, and with covariant subtyping, as described in Cardelli (1989). Record types are passive.
- **Variants** (Table 12), also known as unions or finite sum types, as described in Cardelli (1989). Variant record types are active.

The proofs in the appendix are for the language $\mathbf{FOb}_{\mathbf{1}<:\mu}$ extended with these types. We leave second-order extensions, in particular polymorphism, for future work. We believe our results would routinely extend to eager constructs, that is, call-by-value functions, records that evaluate their components, and so on, but we have not checked the details.

$$\frac{\Gamma \vdash E_i \, (\forall i \in I)}{\Gamma \vdash \mathtt{record}(E_i)_{i \in I}} \text{ (Type Record)}$$

$$\frac{J \subseteq I \quad \Gamma \vdash E_j <: E'_j \quad (\forall j \in J)}{\Gamma \vdash \mathtt{record}(\ell_i{:}E_i)_{i \in I} <: \mathtt{record}(\ell_j{:}E'_j)_{j \in J}} \text{ (Sub Record)}$$

$$\frac{\Gamma \vdash e_i : E_i \, (\forall i \in I)}{\Gamma \vdash \mathtt{record}(\ell_i = e_i)_{i \in I} : \mathtt{record}(\ell_i{:}E_i)_{i \in I}} \text{ (Val Record)}$$

$$\frac{\Gamma \vdash e : \mathtt{record}(\ell_i{:}E_i)_{i \in I} \quad j \in I}{\Gamma \vdash e.\ell_j : E} \text{ (Val Select)}$$

$$\frac{j \in I}{\mathtt{record}(\ell_i = a_i)_{i \in I}.\ell_j \mapsto a_j} \text{ (Red Record)}$$

$$\frac{A \equiv \mathtt{record}(\ell_i{:}A_i)_{i \in I} \quad j \in I \quad \mathcal{E} \equiv -.\ell_j \quad B \equiv A_j}{\mathtt{record}(\ell_i = a_i)_{i \in I\, A} \xrightarrow{\mathcal{E}\downarrow} \mathcal{E}[a]_B} \text{ (Trans Record)}$$

$$\frac{\Gamma \vdash e_i \mathcal{R} e'_i : A_i \, (\forall i \in I)}{\Gamma \vdash \mathtt{record}(\ell_i = e_i)_{i \in I} \, \widehat{\mathcal{R}} \, \mathtt{record}(\ell_i = e'_i)_{i \in I} : \mathtt{record}(\ell_i{:}A_i)_{i \in I}} \text{ (Comp Record)}$$

$$\frac{\Gamma \vdash e_1 \mathcal{R} e'_1 : \mathtt{record}(\ell_i{:}A_i)_{i \in I} \quad j \in I}{\Gamma \vdash e_1.\ell_j \, \widehat{\mathcal{R}} \, e'_1.\ell_j : A_j} \text{ (Comp Select)}$$

$$\begin{array}{rcl} \mathcal{E} & ::= & -.\ell_i \\ v & ::= & \mathtt{record}(\ell_i = a_i)_{i \in I} \end{array}$$

Table 11: Rules for records

$$\frac{\Gamma \vdash E_i\,(\forall i \in I)}{\Gamma \vdash \mathtt{variant}(E_i)_{i\in I}}\ \text{(Type Variant)}$$

$$\frac{I \subseteq J \quad \Gamma \vdash E_i <: E'_i\,(\forall i \in I)}{\Gamma \vdash \mathtt{variant}(\ell_i{:}E_i)_{i\in I} <: \mathtt{variant}(\ell_j{:}E'_j)_{j\in J}}\ \text{(Sub Variant)}$$

$$\frac{j \in I \quad \Gamma \vdash e : E_j \quad \Gamma \vdash E_i\,(\forall i \in I)}{\Gamma \vdash \mathtt{variant}(\ell_j = e) : \mathtt{variant}(\ell_i{:}E_i)_{i\in I}}\ \text{(Val Variant)}$$

$$\frac{\Gamma \vdash e : \mathtt{variant}(\ell_i{:}E_i)_{i\in I} \quad \Gamma, x{:}E_i \vdash e_i : E\,(\forall i \in \{1,\ldots,n\})}{\Gamma \vdash \mathtt{case}(e,(x)e_1,\ldots,(x)e_n) : E}\ \text{(Val Case)}$$

$$\frac{}{\mathtt{case}(\mathtt{variant}(\ell_j = a_j),(x)e_1,\ldots,(x)e_n) \mapsto e_j[a_j/x]}\ \text{(Red Case)}$$

$$\frac{A \equiv \mathtt{variant}(\ell_i{:}A_i)_{i\in I} \quad j \in I}{\mathtt{variant}(\ell_j = a)_A \xrightarrow{\ell_j} a_{A_j}}\ \text{(Trans Variant)}$$

$$\frac{j \in I \quad \Gamma \vdash e\mathcal{R}e' : A_j}{\Gamma \vdash \mathtt{variant}(\ell_j = e)\,\widehat{\mathcal{R}}\,\mathtt{variant}(\ell_j = e') : \mathtt{variant}(\ell_i{:}A_i)_{i\in I}}\ \text{(Comp Variant)}$$

$$\frac{\Gamma \vdash e\mathcal{R}e' : \mathtt{variant}(\ell_i{:}A_i)_{i\in I} \quad \Gamma, x_i{:}A_i \vdash e_i\mathcal{R}e'_i : A\,(\forall i \in I)}{\Gamma \vdash \mathtt{case}(e,(x)e_1,\ldots,(x)e_n)\,\widehat{\mathcal{R}}\,\mathtt{case}(e',(x)e'_1,\ldots,(x)e'_n) : A}\ \text{(Comp Case)}$$

$$\begin{array}{rcl} \mathcal{E} & ::= & \mathtt{case}(-,(x)e_1,\ldots,(x)e_n) \\ v & ::= & \mathtt{variant}(\ell_j = a_j) \end{array}$$

Table 12: Rules for variant records

8 Operational adequacy

Proposition 26(3) strengthened from '$\Rightarrow$' to 'iff' is an important property of equality and divergence. In the setting of the equality induced by a denotational semantics it is known as **computational adequacy** (see Pitts (1994), for instance) or **complete adequacy** (Meyer and Cosmadakis 1988).

We can characterise the types with this property as follows. Let a type A be **total** iff $\forall a{:}A \exists v{:}A(a \sim_A v)$, that is, every program equals a value. Otherwise we say A is **partial**. The following proposition says that the partial types are exactly those that are computationally adequate.

Proposition 30 *For any type A, the following are equivalent.*

(1) *A partial*

(2) $\forall a{:}A(a \sim \Omega^A \Rightarrow a{\Uparrow})$

(3) $\forall a, b{:}A(a{\Downarrow} \;\&\; a \sim b \Rightarrow b{\Downarrow})$

In contrast to partial types, at a total type A, equality to Ω^A cannot imply divergence because there is a value equal to every program, including Ω^A. Total types are important for our study because we want Abadi and Cardelli's (Eq Top) rule to hold. It asserts that all programs at type `Top` are equal. It holds because there are no transitions at type `Top`. So at least `Top` is total. Clearly `Bool` is partial. But object types may be either. $[\ell{:}\texttt{Bool}]$ is partial but [] and $[\ell{:}[]]$, for instance, are total. In order to compute which types are total, we introduce a further distinction, between singular and plural types.

Let a type A be **singular** iff $\forall a, b{:}A(a \sim_A b)$, that is, any two programs at type A are equal. Otherwise we say A is **plural**, that is, there are at least two distinct programs at type A.

We can compute the singular types as follows. Let SP be the computable function given by structural recursion on argument E using the equations in Table 13. If E is a type, then $SP(E)$ either equals the constant S or a pair (v, e) where v is a value (possibly with free type variables) and e is an expression. We assume a total ordering on the set of method labels.

Proposition 31

(1) $SP(A) = S \Rightarrow \forall a, b{:}A(a \sim_A b)$

(2) $SP(A) \neq S \Rightarrow \exists v{:}A(v \Downarrow v \;\&\; v \not\sim \Omega^A)$

(3) *A is singular iff $SP(A) = S$.*

$$SP(X) = S$$

$$SP(\texttt{Top}) = S$$

$$SP(\texttt{Bool}) = (\texttt{true}, -)$$

$$SP([\ell_i{:}E_i]_{i\in I}) = \begin{cases} S & \text{if } SP(E_i) = S \text{ for all } i \in I \\ ([\ell_j{=}v_j, \ell_i{=}\Omega^{E_i}]_{i\in I-\{j\}}, \\ \ e_j[-.\ell_j]) & \text{if } \ell_j \text{ is the least label with } SP(E_j) = (v_j, e_j) \end{cases}$$

$$SP(\mu(X)E) = \begin{cases} S & \text{if } SP(E) = S \\ (\texttt{fold}(\mu, v[\mu/X]), & \text{if } SP(E) = (v, e) \text{ and} \\ \ e[\mu/X][\texttt{unfold}(-)]) & \mu \equiv \mu(X)E \end{cases}$$

$$SP(E_1 \to E_2) = \begin{cases} S & \text{if } SP(E_2) = S \\ (\lambda(x{:}E_1)v_2, e_2[-(\Omega^{E_1})]) & \text{if } SP(E_2) = (v_2, e_2) \end{cases}$$

$$SP(\texttt{Dynamic}) = (\texttt{tag}(\texttt{Bool}, \texttt{true}), \texttt{typecase}(-, (x{:}\texttt{Bool})\texttt{true}, \texttt{true}))$$

$$SP(\texttt{record}(\ell_i{:}E_i)_{i\in I}) = \begin{cases} S & \text{if } SP(E_i) = S \text{ for all } i \in I \\ (\texttt{record}(\ell_j{=}v_j, \ell_i{=} & \text{if } \ell_j \text{ is the least label} \\ \Omega^{E_i})_{i\in I-\{j\}}, e_j[-.\ell_j]) & \text{with } SP(E_j) = (v_j, e_j) \end{cases}$$

$$SP(\texttt{variant}(\ell_i{:}E_i)_{i\in I)} = (\texttt{variant}(\ell_1 = \Omega^{E_1}), \texttt{case}(-, (x)\texttt{true}, \ldots, (x)\texttt{true}))$$

Table 13: The $SP(-)$ function

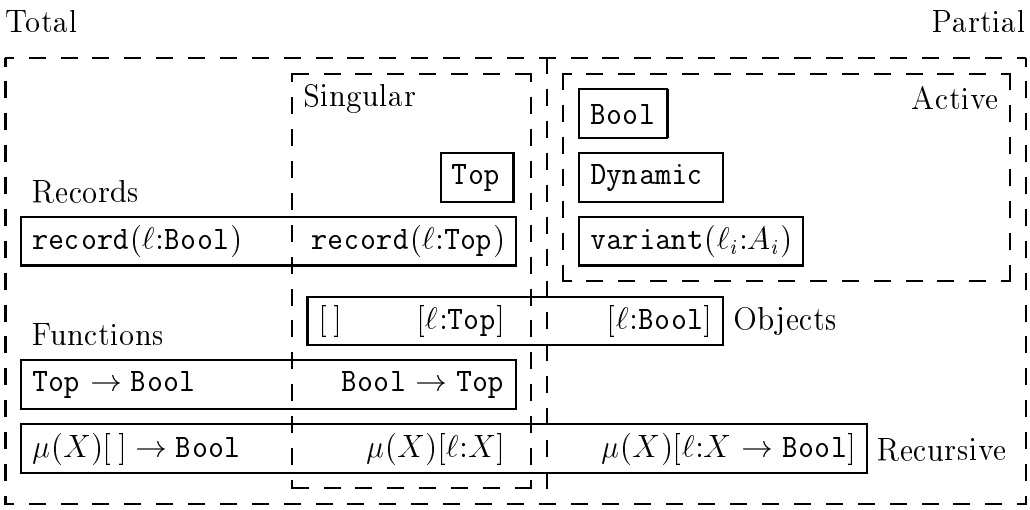

Figure 1: Classification of types in $\mathbf{Ob}_{\mathbf{1}<:\mu}$

A type is singular if neither it nor any of its subexpressions is `Bool`. So $\mu(X)[\ell_1$:`Top`$, \ell_2{:}X]$ is singular but $\mu(X)[\ell_1$:`Bool`$, \ell_2{:}X]$ is not. Intuitively, all programs are equal at a singular type because there are no `Bool`-contexts to tell them apart.

Now we can compute whether a type is partial or total as follows.

Proposition 32 *Suppose A is a type. If $SP(A) = S$ then A is total. Otherwise A takes one of the following forms:*

(1) $\mu(X_1) \ldots \mu(X_n)$`Bool`*, partial;*

(2) $\mu(X_1) \ldots \mu(X_n)[\ell_i{:}E_i]$*, partial.*

(3) $\mu(X_1) \ldots \mu(X_n)E_1 \to E_2$*, total;*

(4) $\mu(X_1) \ldots \mu(X_n)$`Dynamic`*, partial.*

(5) $\mu(X_1) \ldots \mu(X_n)$`record`$(\ell_i{:}E_i)$*, total.*

(6) $\mu(X_1) \ldots \mu(X_n)$`variant`$(\ell_i{:}E_i)$*, partial.*

Object types are either singular or partial; a total object type must be singular. If we had unrestricted call-by-value functions, every type in the language would be partial. Any value, even at type `Top`, could be distinguished from Ω. But then (Eq Top) would be invalid. Figure 1 illustrates the relationships between these classifications of types.

9 Other equivalence relations

Our theory is based on characterising contextual equivalence as a form of bisimilarity. We considered several other forms of operational equivalence.

9.1 Contextual equivalence using capturing contexts

In Section 3, we defined contextual equivalence for closed expressions only. In extending the relation to open expressions, we have two choices; one is to use the relation $\simeq^\circ$, the other is to use contexts with a single hole that captures free variables; that is, we define a relation $\approx^1$ as follows.

$$e \approx^1 e' \quad \text{iff} \quad \begin{cases} \text{for all capturing contexts } \mathcal{C} \text{ s.t.} \\ \mathcal{C}[e]{:}\texttt{Bool} \text{ and } \mathcal{C}[e']{:}\texttt{Bool} \text{ then} \\ \mathcal{C}[e] \Downarrow \text{ iff } \mathcal{C}[e'] \Downarrow \end{cases}$$

We can easily show that that $\approx^1$ contains contextual equivalence for both $\mathbf{FOb}_{\mathbf{1}<:\mu}$ and $\mathbf{Ob}_{\mathbf{1}<:\mu}$; the reverse inclusion holds for the former but fails for the latter. The only bound variables in $\mathbf{Ob}_{\mathbf{1}<:\mu}$ are self-parameters, of object type, so $x{:}\texttt{Bool} \vdash x \approx^1 \texttt{true}$ holds vacuously, but of course $x{:}\texttt{Bool} \vdash x \not\approx^\circ \texttt{true}$, because $\texttt{false}/x$ is an $x{:}\texttt{Bool}$-closure. However, if $\mathbf{Ob}_{\mathbf{1}<:\mu}$ is extended with functions (as in $\mathbf{FOb}_{\mathbf{1}<:\mu}$) or with a `let`-construct at arbitrary type we can prove that the two equivalences are equal.

Proposition 33 *In the presence of functions or a* `let`*-construct,* $\simeq^\circ = \approx^1$.

9.2 Record-style bisimilarity

Thinking of objects as records, we considered a simple equivalence, $\approx^2$, that equates two objects if selections of their methods are pairwise bisimilar.

$$a \approx^2_{[\ell_i:A_i](i\in I)} b \quad \text{iff} \quad \begin{cases} a \Downarrow \text{ iff } b \Downarrow; \text{ and} \\ a.\ell \approx^2_{A_i} b.\ell \text{ for all } i \in I \end{cases}$$

This relation is not discriminatory enough because it has too narrow a notion of observation on objects. It would be correct for a record calculus, but it ignores the possibility of method update. For example, it equates a and b from Section 6 at type $[x{:}\texttt{Bool}, f{:}\texttt{Bool}]$, but we know from Proposition 27 that they are contextually distinct at that type. Abadi and Cardelli (1994c) reject a record-style semantics for their calculus for similar reasons.

9.3 Applicative bisimulation

Howe (1989) defines a format for applicative bisimulation, $\approx^3$, for a general class of untyped λ-calculi. Here is a natural way to express this format in a typed setting.

$$a \approx^3_A b \quad \text{iff} \quad \begin{array}{l} \text{whenever } a \Downarrow u \text{ then } \exists v \text{ s.t. } b \Downarrow v \\ \qquad \text{and } \varnothing \vdash u \mathrel{\widehat{\approx^{3\circ}}} v : A; \text{ and vice versa.} \end{array}$$

Unfortunately, this format is too discriminatory for this object calculus. It distinguishes between the two programs a and b from Section 6 at the type $[f{:}\mathtt{Bool}]$, whereas Proposition 29 shows they are contextually equivalent.

Two λ-calculus functions $\lambda(x{:}A)e$ and $\lambda(x{:}A)e'$ are equal if and only if e and e' are equal for any expression of the correct type that may be substituted for x. However, for the methods $\varsigma(s{:}A)\mathtt{true}$ and $\varsigma(s{:}A)s.x$ to be equal, their bodies only need to be equal when particular values of s are substituted for x: namely the objects a and b themselves.

10 Encoding functions as objects

We will consider an encoding of the λ-calculus in the object calculus and use the bisimilarity relation to show that this encoding is sound.

We take as the language for this study the simply-typed λ-calculus together with Booleans. The relations of contextual equivalence and bisimilarity are defined as before, and the two relations may be shown to be equal using the same proof as before. Now we consider the encoding $\langle\!\langle - \rangle\!\rangle$, given by the following rules.

$$\begin{array}{rcl}
\langle\!\langle \mathtt{Bool} \rangle\!\rangle & = & \mathtt{Bool} \\
\langle\!\langle A \to B \rangle\!\rangle & = & [\mathtt{a}{:}\langle\!\langle A \rangle\!\rangle, \mathtt{v}{:}\langle\!\langle B \rangle\!\rangle] \\
\langle\!\langle x \rangle\!\rangle & = & x.\mathtt{a} \\
\langle\!\langle \mathtt{true} \rangle\!\rangle & = & \mathtt{true} \\
\langle\!\langle \mathtt{false} \rangle\!\rangle & = & \mathtt{false} \\
\langle\!\langle \lambda(x{:}A)e \rangle\!\rangle & = & [\mathtt{a} = \varsigma(s{:}\langle\!\langle A \to B \rangle\!\rangle)s.\mathtt{a}, \mathtt{v} = \varsigma(x{:}\langle\!\langle A \to B \rangle\!\rangle)\langle\!\langle e \rangle\!\rangle] \\
\langle\!\langle e_1 e_2 \rangle\!\rangle & = & \langle\!\langle e_1 \rangle\!\rangle.\mathtt{a} \Leftarrow \varsigma(s{:}\langle\!\langle A \to B \rangle\!\rangle)\langle\!\langle e_2 \rangle\!\rangle.\mathtt{v} \\
\langle\!\langle \mathtt{if}(e_1, e_2, e_3) \rangle\!\rangle & = & \mathtt{if}(\langle\!\langle e_1 \rangle\!\rangle, \langle\!\langle e_2 \rangle\!\rangle, \langle\!\langle e_3 \rangle\!\rangle))
\end{array}$$

To show that this encoding is sound, we need to show that any reduction of a λ-calculus expression can be matched by the reduction of its translation in the object calculus. Unfortunately, the expected property 'if $a \mapsto b$ then $\langle\!\langle a \rangle\!\rangle \mapsto^+ \langle\!\langle b \rangle\!\rangle$' fails. For example, let $a \equiv ((\lambda(x{:}\mathtt{Bool})\lambda(y{:}\mathtt{Bool})x)\,\mathtt{true})$ and $b \equiv \lambda(y{:}\mathtt{Bool})\mathtt{true}$. Omitting the type annotations for clarity, and letting $o \equiv [\mathtt{a} = \varsigma(s)s.\mathtt{a}, \mathtt{v} = \varsigma(y)x.\mathtt{a}]$ we have

$$\begin{array}{rcl}
\langle\!\langle a \rangle\!\rangle & \mapsto^+ & [\mathtt{a} = \varsigma(s)s.\mathtt{a}, \mathtt{v} = \varsigma(y)[\mathtt{a} = \mathtt{true}, \mathtt{v} = \varsigma(x)o].\mathtt{a}] \\
\langle\!\langle b \rangle\!\rangle & \equiv & [\mathtt{a} = \varsigma(s)s.\mathtt{a}, \mathtt{v} = \varsigma(y)\mathtt{true}].
\end{array}$$

So $a \mapsto b$ but $\langle\!\langle a \rangle\!\rangle \not\mapsto^+ \langle\!\langle b \rangle\!\rangle$. However, the translation of a differs from the translation of b only in that where the latter has `true`, the former has the unevaluated selection $[\mathtt{a} = \mathtt{true}, \ldots].\mathtt{a}$, and in fact the congruence property of bisimilarity immediates gives $\langle\!\langle a \rangle\!\rangle \sim \langle\!\langle b \rangle\!\rangle$. To make this idea precise, we introduce the relation $\Gamma \vdash e < e' : A$ on expressions in the object calculus, defined by induction on the two rules

$$\frac{}{\varnothing \vdash a < \delta(a,b) : A} \qquad \frac{\Gamma \vdash e \mathrel{\widehat{<}} e' : A}{\Gamma \vdash e < e' : A}$$

where $\delta(a,b) \stackrel{\text{def}}{=} [\mathtt{a} = a, \mathtt{v} = \varsigma(s)b].\mathtt{a}$, for suitably typed programs a and b. Clearly $\delta(a,b) \mapsto a$. The intuition is that $a < b$ if b has the same structure as a, but contains zero or more additional unevaluated method selections.

Now we can prove that if a λ-calculus program converges, then it is soundly modelled by its translation into the object calculus, in the following sense.

Lemma 34 *Let a be a λ-calculus program with $a{:}A$. Then*

(1) $a \Downarrow v \Rightarrow \exists u(\langle\!\langle a \rangle\!\rangle \Downarrow u \;\&\; \langle\!\langle v \rangle\!\rangle <_{\langle\!\langle A \rangle\!\rangle} u)$

(2) $\langle\!\langle a \rangle\!\rangle \Downarrow u \Rightarrow \exists v(a \Downarrow v \;\&\; \langle\!\langle v \rangle\!\rangle <_{\langle\!\langle A \rangle\!\rangle} u)$

The soundness of the translation is a simple corollary.

Theorem 4 *If $\langle\!\langle a \rangle\!\rangle \simeq_{\langle\!\langle A \rangle\!\rangle} \langle\!\langle b \rangle\!\rangle$ then $a \simeq_A b$.*

Proof Suppose for some λ-calculus context $-{:}A \vdash e$ that we have $e[a]\Downarrow$. We must show $e[b]\Downarrow$. By Proposition 34(1) $\langle\!\langle e[a] \rangle\!\rangle \Downarrow$; from $\langle\!\langle a \rangle\!\rangle \simeq_{\langle\!\langle A \rangle\!\rangle} \langle\!\langle b \rangle\!\rangle$ we have $\langle\!\langle e[b] \rangle\!\rangle \Downarrow$; hence by Proposition 34(2) $e[b]\Downarrow$. ■

However, the translation is not fully abstract, because we can exhibit two programs that are equivalent, but whose translations are contextually inequivalent.

Lemma 35 *There exist λ-calculus terms a, b such that $a \sim b$ but not $\langle\!\langle a \rangle\!\rangle \sim \langle\!\langle b \rangle\!\rangle$.*

Proof Take the following programs.

$$\begin{array}{rcl} a & \equiv & \lambda(x{:}\mathtt{Bool} \to \mathtt{Bool})(x\,\mathtt{true}) \\ b & \equiv & \lambda(x{:}\mathtt{Bool} \to \mathtt{Bool})\mathtt{if}((x\,\mathtt{false}), (x\,\mathtt{true}), (x\,\mathtt{true})) \end{array}$$

The only transition from a is $a \xrightarrow{@c} (a\,c)$ for some $c{:}\mathtt{Bool} \to \mathtt{Bool}$, with $(a\,c) \mapsto (c\,\mathtt{true})$. We have $b \xrightarrow{@c} (b\,c)$ with $(b\,c) \mapsto \mathtt{if}((c\,\mathtt{false}), (c\,\mathtt{true}), (c\,\mathtt{true}))$, and since our simply-typed λ-calculus is determinstic and normalising, then either $(c\,\mathtt{false}) \Downarrow \mathtt{true}$ or $(c\,\mathtt{false}) \Downarrow \mathtt{false}$; in either case $(b\,c) \mapsto^* (c\,\mathtt{true})$, so $a \sim b$.

However, $\langle\!\langle a \rangle\!\rangle$ and $\langle\!\langle b \rangle\!\rangle$ are distinguished by the ς-calculus context $-.\mathtt{a} \Leftarrow \varsigma(s{:}\langle\!\langle \mathtt{Bool} \to \mathtt{Bool} \rangle\!\rangle)[\mathtt{a} = \Omega^{\mathtt{Bool}}, \mathtt{v} = \varsigma(s{:}\langle\!\langle \mathtt{Bool} \to \mathtt{Bool} \rangle\!\rangle)\mathtt{if}(s.\mathtt{a}, s.\mathtt{a}, \Omega^{\mathtt{Bool}})]$. ■

So our translation of the simply-typed λ-calculus without recursion is not fully abstract.

The particular counterexample depends on the fact that our choice of simply-typed λ-calculus is normalising, so it is natural to ask about the case

when the λ-calculus is extended with recursive terms (thus obtaining a form of PCF). In that case, there exists a divergent λ-calculus term Ω^A for each λ-calculus type A, so the counterexample in Lemma 35 fails because the context $(-\,\lambda(y)\mathtt{if}(y{:}\mathtt{Bool}, \mathtt{true}, \Omega^{\mathtt{Bool}}))$ distinguishes the programs a and b.

However it is simple to construct a counterexample that works. For example, take the following programs.

$$\begin{aligned} a' &\equiv \lambda(x{:}\mathtt{Bool})\Omega^{\mathtt{Bool}} \\ b' &\equiv \Omega^{\mathtt{Bool}\to\mathtt{Bool}} \end{aligned}$$

It is easy to show that $a' \sim b'$, but the context $-.(\mathtt{v} \Leftarrow \varsigma(s{:}\langle\!\langle \mathtt{Bool} \to \mathtt{Bool} \rangle\!\rangle)\mathtt{true}).\mathtt{v}$ distinguishes the translations $\langle\!\langle a \rangle\!\rangle$ and $\langle\!\langle b \rangle\!\rangle$.

11 Related work

Most prior work on the theoretical underpinnings for object-oriented programming uses denotational semantics (Gunter and Mitchell 1994), which provides fixpoint induction for reasoning about programs. Co-induction cannot always take the place of fixpoint induction, but Mason, Smith, and Talcott (1994) show how to derive fixpoint induction in a purely operational setting. Breazu-Tannen, Gunter, and Scedrov (1990) is one of the few papers to establish computational adequacy for a denotational semantics in the presence of subtyping. One conclusion of our study is that in spite of its elementary construction, bisimilarity is a useful operational model for an object calculus. Although much can be done purely operationally, it would be worthwhile to research the connections between contextual equivalence and the PER model for $\mathbf{Ob}_{\mathbf{1}<:\mu}$.

Walker (1995) and Jones (1993) show how to encode objects in the π-calculus. Following their approach, we could translate $\mathbf{Ob}_{\mathbf{1}<:\mu}$ into the π-calculus, but we expect, based on Sangiorgi (1994), that the equivalence generated by the encoding would be finer grained than contextual equivalence. Agha, Mason, Smith, and Talcott (1992) studied untyped actors, a form of objects, with side-effects and concurrency. We consider the extension of our results to the imperative object calculus of Abadi and Cardelli (1995) to be important future work. In the presence of dynamic state all known definitions of bisimilarity are finer grained than contextual equivalence (Stark 1994) but nonetheless we expect bisimilarity to be useful for imperative objects. $\mathbf{Ob}_{\mathbf{1}<:\mu}$ is also studied by Palsberg (1994), who presents a complete type inference algorithm. $\mathbf{Ob}_{\mathbf{1}<:\mu}$ is based on fixed-length objects; Mitchell, Honsell, and Fisher (1993) have developed a λ-calculus of extensible objects. They too define a simple operational semantics, analogous to our $\mapsto$ relation. We expect our theory of bisimilarity could be reworked for their calculus. We are aware of only two other studies of bisimilarity and subtyping. Pierce and Sangiorgi (1995) investigate type annotations on names in the π-calculus. Maung (1993), like us, used a labelled transition system and a notion of similarity to express object properties. He proved that similarity of his objects implies a notion of substitutability.

12 Conclusion

Contextual equivalence formally captures the idea that two programs are equal iff no amount of programming can tell them apart. We characterised contextual equivalence as a form of bisimilarity. We validated Abadi and Cardelli's equational theory. Furthermore, we showed that bisimilarity admits CCS-style proofs of equivalence, going beyond the equational theory. Our work builds on previous studies of bisimilarity for functional calculi (Abramsky and Ong 1993; Howe 1989; Crole and Gordon 1995; Gordon 1995). This is the first use of Howe's method in the presence of subsumption and the first study of contextual equivalence for an object calculus. The chief difficulties were in defining a labelled transition system that correctly dealt with method update and subsumption. Our main results extend to function, dynamic, record and variant types. In all we claim that operational methods are a promising new direction for the foundations of object-oriented programming.

Milner (1989) showed that bisimilarity is a useful theory of concurrent processes. Analogously, our work shows that bisimilarity is a useful theory of objects with subtyping. We have shown that from elementary foundations it captures intuitive operational arguments about objects.

Acknowledgements

Gordon holds a Royal Society University Research Fellowship. Rees holds an EPSRC Research Studentship. We thank Martín Abadi, Luca Cardelli and Andy Pitts for many useful conversations about this work. We are grateful to Paul Taylor for the use of his commutative diagrams macros to typeset some of the diagrams.

References

Abadi, M. and L. Cardelli (1994a, June). A semantics of object types. In **Proceedings of the 9th IEEE Symposium on Logic in Computer Science**, pp. 332–341. IEEE Computer Society Press.

Abadi, M. and L. Cardelli (1994b). A theory of primitive objects: Second-order systems. In **Proceedings of European Symposium on Programming**, Volume 788 of **Lecture Notes in Computer Science**, pp. 1–25. Springer-Verlag.

Abadi, M. and L. Cardelli (1994c, April). A theory of primitive objects: Untyped and first-order systems. In **Theoretical Aspects of Computer Software**, pp. 296–320. Springer-Verlag.

Abadi, M. and L. Cardelli (1995). An imperative object calculus. In **TAPSOFT'95: Theory and Practice of Software Development**, Volume 915 of **Lecture Notes in Computer Science**, pp. 471–485. Springer-Verlag.

Abadi, M. and L. Cardelli (1996). **A Theory of Objects**. To appear.

Abadi, M., L. Cardelli, B. Pierce, and G. Plotkin (1989). Dynamic typing in a statically-typed language. In **Sixteenth ACM Symposium on Principles of Programming Languages**.

Abramsky, S. and L. Ong (1993). Full abstraction in the lazy lambda calculus. **Information and Computation 105**, 159–267. Available as Technical Report 259, University of Cambridge Computer Laboratory.

Agha, G., I. Mason, S. Smith, and C. Talcott (1992, August 24–27,). Towards a theory of actor computation. In **CONCUR'92: Third International Conference on Concurrency Theory, Stony Brook, New York**, Volume 630 of **Lecture Notes in Computer Science**, pp. 565–579. Springer-Verlag.

Amadio, R. and L. Cardelli (1990). Subtyping recursive types. Technical Report 62, DEC Systems Research Center, Palo Alto.

Breazu-Tannen, V., C. A. Gunter, and A. Scedrov (1990, June). Computing with coercions. In **Proceedings of the 1990 ACM Conference on Lisp and Functional Programming**, pp. 44–60.

Cardelli, L. (1989, May 24,). Typeful programming. Technical Report 45, DEC Systems Research Center, Palo Alto.

Compagnoni, A. B. (1994). **Higher-Order Subtyping with Intersection Types**. Ph. D. thesis, Catholic University of Nijmegen.

Crole, R. L. and A. D. Gordon (1995). A sound metalogical semantics for input/output effects. In **CSL'94 Computer Science Logic, Kazimierz, Poland, September 1994**, Volume 933 of **Lecture Notes in Computer Science**, pp. 339–353. Springer-Verlag. Full version submitted to **Mathematical Structures in Computer Science**.

Felleisen, M. and D. Friedman (1986). Control operators, the SECD-machine, and the λ-calculus. In **Formal Description of Programming Concepts III**, pp. 193–217. North-Holland.

Gordon, A. D. (1994). **Functional Programming and Input/Output**. Cambridge University Press.

Gordon, A. D. (1995). Bisimilarity as a theory of functional programming. In **Eleventh Annual Conference on Mathematical Foundations of Programming Semantics**, Volume 1 of **Electronic Notes in Theoretical Computer Science**. Elsevier Science Publishers B.V. Available at http://www.elsevier.nl:80/mcs/tcs/pc/covvol1.htm.

Gunter, C. A. and J. C. Mitchell (Eds.) (1994). **Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design**. MIT Press, Cambridge, Mass.

Howe, D. J. (1989). Equality in lazy computation systems. In **Proceedings of the 4th IEEE Symposium on Logic in Computer Science**, pp. 198–203.

Jones, C. (1993). A pi-calculus semantics for an object-based design notation. In **CONCUR'93: Fourth International Conference on Concurrency Theory**, Volume 715 of **Lecture Notes in Computer Science**, pp. 158–172. Springer-Verlag.

Mason, I. A., S. F. Smith, and C. L. Talcott (1994). From operational semantics to domain theory. Submitted for publication.

Maung, I. (1993). Simulation, subtyping and substitutability. Technical Report UBC 93/5, Department of Computing, University of Brighton.

Meyer, A. R. and S. S. Cosmadakis (1988, July). Semantical paradigms: Notes for an invited lecture. In **Proceedings of the 3rd IEEE Symposium on Logic in Computer Science**, pp. 236–253.

Milner, R. (1977). Fully abstract models of typed lambda-calculi. **Theoretical Computer Science 4**, 1–23.

Milner, R. (1989). **Communication and Concurrency**. Prentice-Hall International.

Mitchell, J. C., F. Honsell, and K. Fisher (1993). A lambda calculus of objects and method specialization. In **Proceedings of the Eighth IEEE Symposium on Logic in Computer Science, Montreal**, pp. 26–38.

Morris, J. H. (1968, December). **Lambda-Calculus Models of Programming Languages**. Ph. D. thesis, MIT.

Palsberg, J. (1994). Efficient inference of object types. In **Proceedings of the 9th IEEE Symposium on Logic in Computer Science**, pp. 186–195.

Pierce, B. and D. Sangiorgi (1995). Typing and subtyping for mobile processes. **Mathematical Structures in Computer Science**. To appear. Summary in **Proceedings of the 8th IEEE Conference on Logic in Computer Science**, pp. 376–385 (1993).

Pierce, B. C. (1994, August). Bounded quantification is undecidable. **Information and Computation 113**(1), 131–165.

Pitts, A. M. (1994). Computational adequacy via 'mixed' inductive definitions. In **Proceedings Mathematical Foundations of Programming Semantics IX, New Orleans 1993**, Volume 802 of **Lecture Notes in Computer Science**, pp. 72–82. Springer-Verlag.

Plotkin, G. D. (1977). LCF considered as a programming language. **Theoretical Computer Science 5**, 223–255.

Rees, G. (1994, April). Observational equivalence for a polymorphic lambda calculus. University of Cambridge Computer Laboratory. http://www.cl.cam.ac.uk/users/gdr11/equivalence.dvi.

Sangiorgi, D. (1994, May). The lazy lambda calculus in a concurrency scenario. **Information and Computation 111**(1), 120–153.

Stark, I. D. B. (1994, December). **Names and Higher-Order Functions**. Ph. D. thesis, University of Cambridge Computer Laboratory.

Walker, D. (1995, 1 February). Objects in the π-calculus. **Information and Computation 116**(2), 253–271.

A Proofs

A.1 Operational semantics

Proof of Lemma 1

(1) *If* $\Gamma \vdash E$ *then* $ftv(E) \subseteq Dom(\Gamma)$ *and* $\Gamma \vdash \diamond$.

(2) *If* $\Gamma \vdash E <: E'$ *then* $ftv(E) \cup ftv(E') \subseteq Dom(\Gamma)$ *and* $\Gamma \vdash \diamond$.

(3) *If* $\Gamma \vdash e : A$ *then* $ftv(e) \cup fv(e) \subseteq Dom(\Gamma)$ *and* $\Gamma \vdash \diamond$.

(4) *If* $\Gamma \vdash [\ell_i = \varsigma(x_i{:}A_i)e_i]_{i \in I} : B$ *then the* A_i *are identical and each* $A_i <: B$.

(5) *If* $A <: B$ *then* $Prog(A) \subseteq Prog(B)$.

(6) *If* $A <: B$ *and both* A *and* B *are object types, they must have the forms* $[\ell_i{:}A_i]_{i \in I}$ *and* $[\ell_j{:}A_j]_{j \in J}$, *respectively, with* $J \subseteq I$.

(7) *If* $\Gamma, X', X <: X' \vdash E <: E'$ *and* $\Gamma \vdash A <: A'$ *then* $\Gamma \vdash E[A/X] <: E'[A'/X']$.

(8) *Subtyping is decidable.*

(9) *If* $\Gamma \vdash E <: E'$ *and* $\Gamma \vdash E' <: E$ *then* $E \equiv E'$.

The proofs of parts (1) to (7) are routine. For part (8), we give an inductive definition of an alternative subtyping relation, in which all the rules are syntax-directed, and show that the new definition is the same as the old. Part (9) is a simple corollary of part (8). Our algorithmic presentation of the subtyping relation was inspired by Compagnoni (1994). See Amadio and Cardelli (1990) for an algorithm to decide subtypings in a richer system where a recursive type is considered equivalent to its unfolding.

The new subtyping relation, $<:'$, is defined by the rules in Table 14. We observe that the rules are in syntax-directed form; that is, a subtyping relation $\Gamma \vdash E <:' E'$ matches at most one of the rules. Hence, if we can show that $<: = <:'$, and if we can show that application of the rules terminates for any triple (Γ, E_1, E_2) with $\Gamma \vdash E_1$ and $\Gamma \vdash E_2$, then the definition of $<:'$ gives an algorithm for deciding whether two types are in the subtype relation.

The following inclusion is immediate.

Lemma 36 *The rules for* $<:'$ *are all derivable rules for* $<:$; *hence* $<:' \subseteq <:$.

Proof The rules (Sub′ Refl X), (Sub′ Refl Bool) and (Sub′ Refl Rec) follow from (Sub Refl). The (Sub′ Trans X) follows from (Sub X) and (Sub Trans). The rules (Sub′ Top), (Sub′ Object) and (Sub′ Arrow) are identical to (Sub Top), (Sub Object) and (Sub Arrow) respectively. The rule (Sub′ Rec) differs from (Sub Rec) only in a side condition. ■

$$\frac{\Gamma, X <: E, \Gamma' \vdash \diamond}{\Gamma, X <: E, \Gamma' \vdash X <:' X} \text{ (Sub}' \text{ Refl } X)$$

$$\frac{\Gamma, X <: E, \Gamma' \vdash E <:' E' \quad E' \not\equiv \texttt{Top} \quad E' \not\equiv X}{\Gamma, X <: E, \Gamma' \vdash X <:' E'} \text{ (Sub}' \text{ Trans } X)$$

$$\frac{\Gamma \vdash E}{\Gamma \vdash E <:' \texttt{Top}} \text{ (Sub}' \text{ Top)}$$

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash \texttt{Bool} <:' \texttt{Bool}} \text{ (Sub}' \text{ Refl Bool)}$$

$$\frac{J \subseteq I \quad \Gamma \vdash E_i \, (\forall i \in I)}{\Gamma \vdash [\ell_i : E_i]_{i \in I} <:' [\ell_i : E_i]_{i \in J}} \text{ (Sub}' \text{ Object)}$$

$$\frac{\Gamma, X <: \texttt{Top} \vdash E}{\Gamma \vdash \mu(X)E <:' \mu(X)E} \text{ (Sub}' \text{ Refl Rec)}$$

$$\frac{\begin{array}{c}\Gamma \vdash \mu(X_1)E_1 \quad \Gamma \vdash \mu(X_2)E_2 \\ \Gamma, X_2 <: \texttt{Top}, X_1 <: X_2 \vdash E_1 <:' E_2 \quad \mu(X_1)E_1 \not\equiv \mu(X_2)E_2\end{array}}{\Gamma \vdash \mu(X_1)E_1 <:' \mu(X_2)E_2} \text{ (Sub}' \text{ Rec)}$$

$$\frac{\Gamma \vdash E_1' <:' E_1 \quad \Gamma \vdash E_2 <:' E_2'}{\Gamma \vdash E_1 \to E_2 <:' E_1' \to E_2'} \text{ (Sub}' \text{ Arrow)}$$

Table 14: Syntax-directed subtyping relation

To prove the reverse inclusion, we must establish that the rules for $<:$ are derivable for $<:'$. The difficult case is (Sub Trans); the other rules are straightforward to derive.

Lemma 37 *The rules* (Sub Refl)*,* (Sub X)*,* (Sub Top)*,* (Sub Object)*,* (Sub Rec) *and* (Sub Arrow) *are derivable for* $<:'$*.*

Proof For (Sub Refl), we prove the result by induction on the derivation of $\Gamma \vdash E$.

For (Sub X), then if $E \equiv \mathtt{Top}$, then the result follows by (Sub Top). Otherwise, given the hypothesis, then $\Gamma, X <: E, \Gamma' \vdash E$ by (Env X); hence $\Gamma, X <: E, \Gamma' \vdash E <:' E$ by (Sub Refl); hence $\Gamma, X <: E, \Gamma' \vdash X <:' E$ by (Sub Trans X).

The rules (Sub Top), (Sub Object) and (Sub Arrow) are identical to (Sub$'$ Top), (Sub$'$ Object) and (Sub$'$ Arrow) respectively.

The rule (Sub Rec) follows from (Sub$'$ Refl Rec) and (Sub$'$ Rec) together. ■

It suffices to prove (Sub Trans) for $<:'$. The most obvious proof method is induction on the derivation of the subtyping judgment. The cases are easy except for (Sub$'$ Rec), when the judgments we derive are initially in the wrong form to apply the induction hypothesis; that is, we deduce

$$\Gamma, X_2 <: \mathtt{Top}, X_1 <: X_2 \vdash E_1 <:' E_2 \tag{1}$$
$$\Gamma, X_3 <: \mathtt{Top}, X_2 <: X_3 \vdash E_2 <:' E_3 \tag{2}$$

and wish to derive $\Gamma, X_3 <: \mathtt{Top}, X_1 <: X_3 \vdash E_1 <:' E_3$ by the induction hypothesis. However, we cannot do this because the environments are different.

Our solution will involve induction on a certain weighting of subtyping judgments; in the case for (Sub$'$ Rec) we will derive the following judgments,

$$\Gamma, X_3 <: \mathtt{Top}, X_2 <: X_3, X_1 <: X_2 \vdash E_1 <:' E_2 \tag{3}$$
$$\Gamma, X_3 <: \mathtt{Top}, X_2 <: X_3, X_1 <: X_2 \vdash E_2 <:' E_3 \tag{4}$$

from (1) and (2). Since (3) and (4) involve the same environment we apply the induction hypothesis and the case for (Sub$'$ Rec) easily follows. The implication (2)⇒(4) is just weakening, since the type variable X_1 is not free in E_2 or E_3. To establish (1)⇒(3), we prove the following lemma.

Lemma 38

(1) $\Gamma, X <: E'', \Gamma' \vdash \diamond$ *iff* $\Gamma, Y <: E'', X <: Y, \Gamma' \vdash \diamond$ *for some variable* $Y \notin Dom(\Gamma, X <: E'', \Gamma')$.

(2) $\Gamma, X <: E'', \Gamma' \vdash E$ *iff* $\Gamma, Y <: E'', X <: Y, \Gamma' \vdash E$ *for some variable* $Y \notin Dom(\Gamma, X <: E'', \Gamma')$.

(3) $\Gamma, X <: E'', \Gamma' \vdash E <:' E'$ *iff* $\Gamma, Y <: E'', X <:' Y, \Gamma' \vdash E <:' E'$ *for some variable* $Y \notin Dom(\Gamma, X <: E'', \Gamma')$.

Proof In each case, the proof is by rule induction. Parts (1) and (2) are routine; in part (3) we must take care in the case (Sub′ Trans X) when $X = Y$; then we must apply (Sub′ Trans X) twice to deduce the required result. ■

We note that in going from $\Gamma, X<:\texttt{Top}, \Gamma' \vdash E<:'E'$ to $\Gamma, Y<:\texttt{Top}, X<:Y, \Gamma' \vdash E <:' E'$ we may increase the depth of the derivation, because the rule (Sub′ Trans X) may be applied more times in the derivation of the latter than in the derivation of the former. Hence we cannot now prove the validity of the case (Sub′ Rec) in the proof of (Sub′ Trans) by simply inducting on the depth of derivation of the subtyping judgment, because the judgment (3) may have a longer derivations than the judgment (1), and so the induction hypothesis will not go through.

Instead of a simple induction on depth of inference, we induct on the value of the function $w(\Gamma, E, E') = \langle p, q\rangle$ which computes a weight (a pair of natural numbers) for the triple (Γ, E, E') where we have $\Gamma \vdash E$ and $\Gamma \vdash E'$. The weight w is given by the following definition.

$$\begin{array}{rcl} w(\Gamma, E, E') & = & \langle \text{size of } E', |\Gamma'| \rangle \\ & & \quad \text{where } \Gamma' \text{ is the least initial segment} \\ & & \quad \text{of } \Gamma \text{ with } ftv(E) \subseteq Dom(\Gamma')\rangle \end{array}$$

Weights are considered to be ordered lexicographically. We extend weights to subtyping judgments by setting $w(\Gamma \vdash E<:'E') = w(\Gamma, E, E')$. By inspection of the rules in Table 14 we note that in each rule, the weight of the hypothesis of the rule is always less than the weight of the conclusion of the rule. For each rule except (Sub′ Trans X) the first component of the weight goes down from conclusion to hypothesis; for (Sub′ Trans X) the first component remains the same, and the second component decreases (for the environment $\Gamma, X <: E, \Gamma'$ to be well-formed we must have $\Gamma \vdash E$, which has a shorter environment than the judgment $\Gamma, X <: E \vdash X$).

The weight w establishes that the rules in Table 14 give a terminating algorithm; that is, for any well-formed environment Γ and any two types E and E' with $\Gamma \vdash E$ and $\Gamma \vdash E'$, the application of the rules eventually produces a proof of $\Gamma \vdash E <:' E'$ or else reaches a point where no rule applies.

To establish the correctness of the algorithm, we will need the following property of w.

Lemma 39 *Let $\mathcal{J}$ be the judgment $\Gamma, X <: \mathtt{Top}, \Gamma' \vdash E <:' E'$ and let $\mathcal{K}$ be the judgment $\Gamma, Y <: \mathtt{Top}, X <: Y, \Gamma' \vdash E <:' E'$ for some $Y \notin fv(\Gamma) \cup fv(\Gamma') \cup \{X\}$. Then if $\mathcal{J}$ is derivable with $w(\mathcal{J}) = \langle p, q \rangle$, then $\mathcal{K}$ is derivable with $w(\mathcal{K}) = \langle p, q + n \rangle$ for some $n \geq 0$.*

Proof That $\mathcal{K}$ is derivable follows from Lemma 38. That $w(\mathcal{K}) = \langle p, q+n \rangle$ is proved by induction on the derivation of $\mathcal{J}$. ■

Now the proof that (Sub Trans) is a derived rule for $<:'$ is simple.

Lemma 40 (Sub Trans) *If $\Gamma \vdash E_1 <:' E_2$ and $\Gamma \vdash E_2 <:' E_3$ then $\Gamma \vdash E_1 <:' E_3$.*

Proof By induction on the weight $w(\Gamma \vdash E_1 <:' E_2)$, considered to be ordered lexicographically. The interesting case is (Sub Rec). Here we have the judgments

$$\Gamma \vdash \mu(X_1)E_1 <:' \mu(X_2)E_2 \tag{5}$$
$$\Gamma \vdash \mu(X_2)E_2 <:' \mu(X_3)E_3 \tag{6}$$

with $w(5) = \langle p, q \rangle$. By (Sub′ Rec) we deduce the judgments

$$\Gamma, X_2 <: \mathtt{Top}, X_1 <: X_2 \vdash E_1 <:' E_2 \tag{7}$$
$$\Gamma, X_3 <: \mathtt{Top}, X_2 <: X_3 \vdash E_1 <:' E_2 \tag{8}$$

with $w(7) = \langle p-1, q \rangle$. By Lemma 38 and Lemma 39 we derive the judgment

$$\Gamma, X_3 <: \mathtt{Top}, X_2 <: X_3, X_1 <: X_2 \vdash E_1 <:' E_2 \tag{9}$$

with $w(9) = \langle p-1, q+n \rangle$ for some $n \geq 0$ (since $X_3 \notin fv(\Gamma) \cup \{X_1, X_2\}$). By weakening we derive

$$\Gamma, X_3 <: \mathtt{Top}, X_2 <: X_3, X_1 <: X_2 \vdash E_1 <:' E_2 \tag{10}$$

(since $X_1 \notin fv(\Gamma) \cup \{X_3, X_2\}$). Since $\langle p-1, q+n \rangle < \langle p, q \rangle$ we can apply the induction hypothesis and deduce

$$\Gamma, X_3 <: \mathtt{Top}, X_2 <: X_3, X_1 <: X_2 \vdash E_1 <:' E_3 \tag{11}$$

so by Lemma 38(3) we can get rid of the variable X_2 and establish

$$\Gamma, X_3 <: \mathtt{Top}, X_1 <: X_3 \vdash E_1 <:' E_3. \tag{12}$$

Hence $\Gamma \vdash \mu(X_1)E_1 <:' \mu(X_2)E_2$ as required. ■

Now we can establish the main result:

Proof of Lemma 1(8) *Subtyping is decidable.*

Proof Combining Lemmas 36, 37 and 40 gives us the identity $<:' = <:$, and then the syntax-directed rules for $<:'$ give us an algorithm for deciding the validity of a candidate subtyping relation. ■

Proof of Lemma 1(9) *If* $\Gamma \vdash E <: E'$ *and* $\Gamma \vdash E' <: E$ *then* $E \equiv E'$.

Proof By inspection of the rules for the relation $<:'$. ■

Proof of Lemma 2 *If* $\Gamma, x{:}E', \Gamma' \vdash e : E$ *and* $\Gamma \vdash e' : E'$, *then* $\Gamma, \Gamma' \vdash e[e'/x] : E$.

Proof By induction on the derivation of $\Gamma, x{:}E', \Gamma' \vdash e : E$. ■

Proof of Lemma 3 *If* $\Gamma, x{:}A, \Gamma' \vdash e : B$ *and* $A' <: A$ *then* $\Gamma, x{:}A', \Gamma' \vdash e : B$ *too.*

Proof By induction on the derivation of $\Gamma, x{:}A, \Gamma' \vdash e : B$. ■

Proof of Lemma 4 *If* $a{:}A$ *there is a unique list of experiments* $\mathcal{E}_1, \ldots, \mathcal{E}_n$ *and value* v *such that* $a \equiv \mathcal{E}_1[\ldots \mathcal{E}_n[v] \ldots]$.

Proof By induction on the derivation of $a{:}A$. ■

Proof of Lemma 5 *The values are the normal forms of* $\mapsto$, *that is, whenever* a *is a program,* $a \in \mathit{Value}$ *iff* $\neg(a \mapsto)$.

Proof The forward direction is easy to check; we just observe that no reduction rule applies to any value. For the reverse direction, we prove by induction on the structure of the derivation of $a{:}A$ that if $a \notin \mathit{Value}$, then $a \mapsto$. Consider the last rule used in the derivation of $a{:}A$. It cannot have been (Val x), because a is a closed expression, and cannot have been (Val True), (Val False), (Val Object), (Val Fun), (Val Dynamic), (Val Record) or (Val Variant) because $a \notin \mathit{Value}$.

(Val Subsumption) Here $a{:}A'$ for some $A' <: A$. By the induction hypothesis, if $a \notin \mathit{Value}$, then $a \mapsto$.

(Val If) Here $a \equiv \mathtt{if}(a_1, a_2, a_3)$ with $a_1{:}\mathtt{Bool}$. If $a_1 \in \mathit{Value}$, then by its type we must have either $a_1 \equiv \mathtt{true}$ or $a_1 \equiv \mathtt{false}$ and then either $a \mapsto a_2$ by (Red If True) or $a \mapsto a_3$ by (Red If False) respectively. Otherwise a_1 is not a result and by the induction hypothesis $a_1 \mapsto a_1'$. Hence $a \mapsto \mathtt{if}(a_1', a_2, a_3)$ by (Red Experiment).

(Val Select) **or** (Val Update) Here $a \equiv b.\delta$ where $b{:}[\ell_i{:}B_i]_{i\in I}$ and $j \in I$ and either $\delta \equiv \ell_j$ or $\delta \equiv \ell_j \Leftarrow \varsigma(x{:}A)e$, respectively. If $b \notin \mathit{Value}$, then by the induction hypothesis $b \mapsto b'$; hence $a \mapsto b'.\delta$ by (Red Experiment). Otherwise $b \in \mathit{Value}$, and by its type we must have $b \equiv [\ell_i = \varsigma(x_i{:}B_i)e_i]_{i\in I}$ and by (Red Select) or (Red Update) we have either $a \mapsto e_j[{}^{b}\!/\!_{x}]$ or $a \mapsto [\ell_j = \varsigma(x{:}B_j)e, \ell_i = \varsigma(x_i{:}B_i)e_i]_{i\in I-\{j\}}$, respectively.

(Val Fold) Here $a \equiv \mathtt{fold}(A, a')$ and $a'{:}A$. If $a \notin \mathit{Value}$, then $a' \notin \mathit{Value}$, and by the induction hypothesis, $a' \mapsto a''$. Hence $a \mapsto \mathtt{fold}(A, a'')$ by (Red Experiment).

(Val Unfold) Here $A \equiv E[{}^{A}\!/\!_{X}]$, $a \equiv \mathtt{unfold}(a')$ and $a'{:}\mu(X)E$. If $a' \in \mathit{Value}$, then by its type we must have $a' \equiv \mathtt{fold}(A', a'')$ with $a'' \in \mathit{Value}$. Then by (Red Unfold) $a \mapsto a''$. Otherwise $a' \notin \mathit{Value}$, and by the induction hypothesis we have $a' \mapsto a''$. Hence $a \mapsto \mathtt{unfold}(a'')$ by (Red Experiment).

(Val Appl) Here $a \equiv (a_1\, a_2)$, with $a_1{:}B \to A$. If $a_1 \in \mathit{Value}$, then $a_1 \equiv \lambda(x{:}B')e$ with $B <: B'$ and then $a \mapsto e[{}^{a_2}\!/\!_{x}]$ by (Red App). Otherwise $a_1 \mapsto a_1'$ by the induction hypothesis, and then $a \mapsto (a_1'\, a_2)$ by (Red Experiment).

(Val Typecase) Here $a \equiv \mathtt{typecase}(a_1, (x{:}B)e, a_3)$ with $a_1{:}\mathtt{Dynamic}$. If $a_1 \in \mathit{Value}$, then by its type we must have $a_1 \equiv \mathtt{tag}(C, a_1')$. Now either $C <: B$, in which case $a \mapsto e[{}^{a_1'}\!/\!_{x}]$ by (Red Dynamic Match), or else $\neg(C <: B)$, in which case $a \mapsto a_3$ by (Red Dynamic Default). Otherwise, $a_1 \notin \mathit{Value}$, hence by the induction hypothesis $a_1 \mapsto a_1'$; hence by (Red Experiment) $a \mapsto \mathtt{typecase}(a_1', (x{:}B)e, a_3)$.

(Val Select) Here $a \equiv a'.\ell_j$ with $a'{:}\mathtt{record}(\ell_i{:}A_i)_{i\in I}$ and $j \in I$. If $a' \in \mathit{Value}$, then by its type we must have $a' \equiv \mathtt{record}(\ell_i = a_i')_{i\in I}$, in which case $a \mapsto a_j'$ by (Red Record). Otherwise, $a' \notin \mathit{Value}$, hence by the induction hypothesis $a' \mapsto a''$; hence by (Red Experiment) $a \mapsto a''.\ell_j$.

(Val Case) Here $a \equiv \mathtt{case}(a', (x{:})e_1, \ldots, (x)e)$ with $a'{:}\mathtt{variant}(\ell_i{:}A_i)_{i\in I}$. If $a' \in \mathit{Value}$, then by its type we must have $a' \equiv \mathtt{variant}(\ell_j = a')$ for

some $j \in I$, in which case $a \mapsto e_j[a'/x]$ by (Red Case). Otherwise, $a' \notin \mathit{Value}$, hence by the induction hypothesis $a' \mapsto a''$; hence by (Red Experiment) $a \mapsto \mathtt{case}(a'', (x)e_1, \ldots, (x)e)$. ■

Proof of Lemma 6 *If $a \mapsto b$ and $a \mapsto c$, then $b \equiv c$.*

Proof By rule induction on the reduction $a \mapsto b$. We note that the rules (Red If True) and (Red If False) apply only if $a \equiv \mathtt{if}(v, a_1, a_2)$, where v is `true` or `false` respectively; the rules (Red Select) and (Red Update) apply only if $a \equiv v.\delta$, where v is an object; the rule (Red Unfold) applies only if $a \equiv \mathtt{unfold}(v)$ where v is $\mathtt{fold}(A, u)$; the rule (Red App) applies only if $a \equiv (v\, a')$ where $v \equiv \lambda(x{:}A)e$; the rules (Red Dynamic Match) and (Red Dynamic Default) apply only if $a \equiv \mathtt{typecase}(v, (x{:}A)e, a')$ where $v \equiv \mathtt{tag}(A', a'')$ (and because subtyping is decidable, Proposition 1(8), only one can apply); the rule (Red Record) applies only if $a \equiv v.\ell_j$ where $v \equiv \mathtt{record}(\ell_i = a_i)_{i \in I}$ and $j \in I$; and the rule (Red Case) applies only if $a \equiv \mathtt{case}(v, (x)e_1, \ldots, (x)e_n)$ where $v \equiv \mathtt{variant}(\ell_j = v')$. But in each case, (Red Experiment) does not apply because $v \in \mathit{Value}$; and no other rule matches the outermost form of a. Therefore $a \mapsto c$ must follow from the same rule as $a \mapsto b$, and in each rule the outcome of the reduction depends only on the form of a. Hence $b \equiv c$. If (Red Experiment) does apply, then $a \equiv \mathcal{E}[a']$, $b \equiv \mathcal{E}[b']$ and $c \equiv \mathcal{E}[c']$ with $a' \mapsto b'$ and $a' \mapsto c'$. By the induction hypothesis, $b' \equiv c'$; hence $b \equiv c$. ■

Proof of Lemma 7 *If $a{:}A$ and $a \mapsto b$ then $b{:}A$.*

Proof By rule induction on the derivation of the typing $a{:}A$. Having found one form of b, we appeal implicitly to Lemma 6 to deduce that there are no other forms for b. Consider the last rule used in the derivation of $a{:}A$. It cannot have been (Val x) because a is a closed expression, and it cannot have been (Val True), (Val False), (Val Object), (Val Fun), (Val Dynamic), (Val Record) or (Val Variant) because $a \mapsto$.

(Val Subsumption) Here we have a type A' such that $a{:}A'$ and $A' <: A$. By the induction hypothesis $b{:}A'$ and by (Val Subsumption) $b{:}A$.

(Val If) Here $a \equiv \mathtt{if}(a', a_2, a_3)$ and $a'{:}\mathtt{Bool}$ and $a_1{:}A$ and $a_2{:}A$. Then either (i) $a' \mapsto a''$ or else (ii) $a' \in \mathit{Value}$. In case (i), by (Red Experiment) $b \equiv \mathtt{if}(a'', a_2, a_3)$ and by the induction hypothesis $a''{:}\mathtt{Bool}$; hence by (Val If) $b{:}A$. In case (ii), by the type $a' \equiv \mathtt{true}$ or $a' \equiv \mathtt{false}$; then by (Red If True) or (Red If False) respectively we have $b \equiv a_2$ or $b \equiv a_3$ respectively; in either case $b{:}A$.

(Val Select) Here $a \equiv a'.\ell_j$ with $a'{:}[\ell_i{:}A_i]$ and $A \equiv A_j$. Then either (i) $a' \mapsto b'$ or else (ii) $a' \in \mathit{Value}$. In case (i), by (Red Experiment) $b \equiv b'.\ell_j$ and by the induction hypothesis we have $b'{:}[\ell_i{:}A_i]$; hence by (Val Select) we have $b{:}A_j \equiv A$. In case (ii), by the type $a' \equiv [\ell_i = \varsigma(x{:}A_i)e_i]_{i\in I}$; by (Red Select) $b \equiv b_j[a/x]$; by (Val Object) we have $a{:}[\ell_i{:}A_i] \vdash b_j : A_j$; and $b{:}A_j \equiv A$ by Lemma 2.

(Val Update) This case is similar to that for (Val Select).

(Val Fold) Here $a \equiv \mathtt{fold}(A, a')$ and $A \equiv \mu(X)E$ and $a'{:}E[A/X]$. Since $a \mapsto$ it must be the case that $a' \notin \mathit{Value}$ and hence $a' \mapsto a''$. By (Red Experiment) $b \equiv \mathtt{fold}(A, a'')$; by the induction hypothesis $a''{:}E[A/X]$ and by (Val Fold) $b{:}A$.

(Val Unfold) Here $a \equiv \mathtt{unfold}(a')$ and $A \equiv E[\mu(X)E/X]$ and $a'{:}\mu(X)E$. Then either (i) $a' \mapsto a''$ or else (ii) $a' \in \mathit{Value}$. In case (i), by (Red Experiment) $b \equiv \mathtt{unfold}(a'')$; by the induction hypothesis $a''{:}\mu(X)E$; hence by (Val Unfold) $b{:}A$. In case (ii), by the type of a' we must have $a' \equiv \mathtt{fold}(\mu(X)E, a'')$; hence by (Red Unfold) $b \equiv a''$ and by (Val Unfold) $b{:}A$.

(Val Appl) Here $a \equiv (a_1\, a_2)$, with $a_1{:}B \to A$ and $a_2{:}B$. Then either (i) $a_1 \mapsto a_1'$ or else (ii) $a_1 \in \mathit{Value}$. In case (i), by (Red Experiment), $b \equiv (a_1'\, a_2)$, and by the induction hypothesis $a_1'{:}B \to A$; hence by (Val Appl) $b{:}A$. In case (ii), by the type $a_1 \equiv \lambda(x{:}B)e$ with $x{:}B \vdash e : A$ and by (Red App) we have $b \equiv e[a_2/x]$, and $b{:}A$ by Lemma 2.

(Val Typecase) Here $a \equiv \mathtt{typecase}(a_1, (x{:}B)e, a_3)$ wand we have $a_1{:}\mathtt{Dynamic}$ and $x{:}B \vdash e{:}A$ and $a_3{:}A$. Then either (i) $a_1 \mapsto a_1'$ or else (ii) $a_1 \in \mathit{Value}$. In case (i), by (Red Experiment), $b \equiv \mathtt{typecase}(a_1', (x{:}B)e, a_3)$, and by the induction hypothesis $a_1'{:}\mathtt{Dynamic}$; hence by (Val Typecase) $b{:}A$. In case (ii), by the type $a_1 \equiv \mathtt{tag}(C, a_1')$, and either $\neg(C <: B)$, in which case $b \equiv a_3$, or else $C <: B$, in which case $b \equiv e[a_1'/x]$ by (Red Dynamic Match), and hence $b{:}A$ by Lemma 2 and Lemma 3.

(Val Select) Here $a \equiv a'.\ell_j$ with $a'{:}\mathtt{record}(\ell_i{:}A_i)_{i\in I}$ and $j \in I$ and $A \equiv A_j$. Then either (i) $a' \mapsto a''$ or else (ii) $a' \in \mathit{Value}$. In case (i), by (Red Experiment), $b \equiv a''.\ell_j$, and by the induction hypothesis $a''{:}\mathtt{record}(\ell_i{:}A_i)_{i\in I}$; hence by (Val Select) $b{:}A$. In case (ii), by the type $a' \equiv \mathtt{record}(\ell_i = a_i')_{i\in I}$, and $b \equiv a_j'$ by (Red Record), and hence $b{:}A$.

(Val Case) Here $a \equiv \texttt{case}(a', (x)e_1, \ldots, (x)e)$ with a':$\texttt{variant}(\ell_i{:}A_i)_{i\in I}$ and $x{:}A_i \vdash e_i{:}A$ for each $i \in I$. Then either (i) $a' \mapsto a''$ or else (ii) $a' \in \mathit{Value}$. In case (i), by (Red Experiment), $b \equiv \texttt{case}(a'', (x)e_1, \ldots, (x)e)$, and by the induction hypothesis a':$\texttt{variant}(\ell_i{:}A_i)_{i\in I}$; hence by (Val Case) $b{:}A$. In case (ii), by the type $a' \equiv \texttt{variant}(\ell_j = a'')$ for some $j \in I$, and $b \equiv e_j[a''/x]$ by (Red Case), and hence $b{:}A$ by Lemma 2. ■

Proof of Lemma 8 $\forall A \in \mathit{Type}\ \exists a_1, a_2{:}A\ (a_1{\Downarrow}\ \&\ a_2{\Uparrow})$.

Proof Let a_2 be Ω^A at each type A. The type A must take the form $\mu(X_1)\ldots\mu(X_n)B$, so let a_1 be $\texttt{fold}(A_1, \ldots\texttt{fold}(A_n, b)\ldots)$, where the A_i's are unravellings of A, and b is $\texttt{true}$ if B is $\texttt{Top}$ or $\texttt{Bool}$, $[\ell_i = \Omega^{B_i}]_{i\in I}$ if B is $[\ell_i{:}B_i]_{i\in I}$, $\lambda(x{:}B_1)\Omega^{B_2}$ if B is $B_1 \to B_2$, $\texttt{tag}(\texttt{true}, \texttt{Bool})$ if B is $\texttt{Dynamic}$, $\texttt{variant}(\ell_1 = \Omega^{B_1})$ if B is $\texttt{variant}(\ell_i{:}B_i)_{i\in I}$ or $\texttt{record}(\ell_i = \Omega^{B_i})_{i\in I}$ if B is $\texttt{record}(\ell_i{:}B_i)_{i\in i}$. We know that B can have no other form by the contractivity condition in (Type Rec $<:$). ■

A.2 Labelled transition system

Proof of Lemma 10 $\alpha \in \mathit{Act}(A)$ *iff* $\exists a{:}A(a_A \xrightarrow{\alpha})$.

Proof By inspection. ■

Proof of Lemma 11 *If* $\varnothing \vdash A <: B$ *then* $\mathit{Act}(B) \subseteq \mathit{Act}(A)$.

Proof The proof is by rule induction on the derivation of $\varnothing \vdash A{<:}B$. The only interesting case is (Sub Object), in which case we have $A \equiv [\ell_i{:}A_i]_{i\in I}$ and $B \equiv [\ell_j{:}B_j]_{j\in J}$, with $J \subseteq I$. We consider an arbitrary $\alpha \in \mathit{Act}(B)$. If α is ℓ_j then $\alpha \in \mathit{Act}(A)$ because $J \subseteq I$. Otherwise α is $\ell_j \Leftarrow \varsigma(x)e$, and $x{:}B \vdash e : A_j$; so $x{:}A \vdash e : A_j$ by Lemma 3; hence $\alpha \in \mathit{Act}(A)$ too, because $J \subseteq I$. ■

Proof of Lemma 12 *If* A *is active then* $a_A \xrightarrow{\alpha} b_B$ *iff* $\exists v \in \mathit{Value}(a \Downarrow v\ \&\ v_A \xrightarrow{\alpha} b_B)$.

Proof If A is active then we have one of the three cases $A \equiv \texttt{Bool}$, $A \equiv \texttt{Dynamic}$ or $A \equiv \texttt{variant}(\ell_i{:}A_i)_{i\in I}$. The result follows by induction on the length of the reduction $a \Downarrow v$, using (Trans Bool), (Trans Dynamic) or (Trans Variant) respectively for the base case and (Trans Red) for the induction step. ■

Proof of Lemma 13 *If* $a{:}A$ *and* $a{\Uparrow}$ *and* $a_A \xrightarrow{\alpha} b_B$ *then* $b{\Uparrow}$.

Proof If $a\Uparrow$ and $a_A \xrightarrow{\alpha}$ then the type A must be passive, in which case there must be some experiment $\mathcal{E}$ such that $\alpha \equiv \mathcal{E}\downarrow$ and $b \equiv \mathcal{E}[A]$. But since each experiment is strict in its hole, $b\Uparrow$ too. ■

Proof of Lemma 14 *If A is passive then $a_A \xrightarrow{\alpha} b_B$ implies $\exists\mathcal{E}(\alpha \equiv \mathcal{E}\downarrow \,\&\, b \equiv \mathcal{E}[a])$.*

Proof By inspection of the rules in Table 7 and the definition of erasure. ■

Proof of Lemma 15 *If $P \xrightarrow{\alpha} Q$ and $P \xrightarrow{\alpha} Q'$, then $Q \equiv Q'$.*

Proof By inspection of the rules in Table 7. ■

A.3 Bisimilarity and subtyping

Proof of Proposition 16

(1) *$\lesssim$ is a preorder and $\sim$ is an equivalence relation.*

(2) *$\sim \;=\; \lesssim \cap \lesssim^{\mathrm{op}}$.*

Proof Part (1) is a standard result for bisimilarity and similarity; see Proposition 4.2 of Milner (1989), for instance. Part (2) depends on the image-singularity of the labelled transition system, Lemma 15, and is easily proved by co-induction. ■

Lemma 41 *Suppose $\varnothing \vdash A <: B$, $\varnothing \vdash a : A$ and $\alpha \in Act(B)$. Then*

(1) *whenever $a_A \xrightarrow{\alpha} {a'}_{A'}$ then there exist a'', B' such that $a_B \xrightarrow{\alpha} {a''}_{B'}$ and $A' <: B'$ and $a' \sim_{A'} a''$; and*

(2) *whenever $a_B \xrightarrow{\alpha} {a'}_{B'}$ then there exist a'', A' such that $a_A \xrightarrow{\alpha} {a''}_{A'}$ and $A' <: B'$ and $a' \sim_{A'} a''$.*

Proof We prove part (1) by an analysis of the way in which $a_A \xrightarrow{\alpha} {a'}_{A'}$ is derived. In every case, a' and a'' are identical, except when $\alpha \equiv \ell \Leftarrow \varsigma(x)e$, in which case a' and a'' differ only in type annotations, that is, $a' \equiv a.\ell \Leftarrow \varsigma(x{:}A)e$ and $a' \equiv a.\ell \Leftarrow \varsigma(x{:}B)e$.

Now, either $a\Uparrow$, in which case by part (3) of Proposition 26 we have $a \sim_A \Omega^A$, hence by Lemma 13 we have $a' \sim_{A'} \Omega^{A'} \sim_{A'} a''$ as required; or else $a \Downarrow v$. Then $v \equiv [\ell_i = \varsigma(x_i{:}A_i)e_i]_{i\in I}$ with $\varnothing \vdash A_i <: A$; also $\alpha \equiv \ell_j \Leftarrow (x)e$ and $a' \equiv v.\ell_j \Leftarrow \varsigma(x{:}A)e_j$ and $A' \equiv A$. Now since $\alpha \in Act(B)$ and $a \Downarrow$,

then it must be the case that $a_B \xrightarrow{\alpha} a''_B$ where $a'' \equiv v.\ell_j \Leftarrow \varsigma(x{:}B)e$. The two programs a' and a'' differ only in the type annotation on the parameter in the updating method. However, this annotation is altered by reduction using (Red Update) to conform to the existing annotations on the type, that is, $a' \mapsto^* a''' \equiv [\ell_j = \varsigma(x{:}A_j)e, \ell_i = \varsigma(x_i{:}A_i)e_i]_{(i \in I - \{j\})}$ and also $a'' \mapsto^* a'''$. By repeated application of part (1) of Proposition 26 $a' \sim_{A'} a''' \sim_{A'} a''$ as required.

Part (2) follows by a symmetric argument. ■

Proof of Lemma 17 *$\mathcal{S} \stackrel{\text{def}}{=} \{(a_B, b_B) \mid \exists A(\varnothing \vdash A <: B \ \&\ a \sim_A b)\}$ is a simulation.*

Proof Suppose that $a_B \,\mathcal{S}\, b_B$. Then there is a type A such that $\varnothing \vdash A <: B$ and $a \sim_A b$. Consider any transition $a_B \xrightarrow{\alpha} a'_{B'}$. By Lemma 11 $\alpha \in Act(A)$. Then by Lemma 41 (2) $a_A \xrightarrow{\alpha} a''_{A'}$ with $\varnothing \vdash A' <: B'$ and $a' \sim_{A'} a''$. Then from the definition of bisimulation, $b_A \xrightarrow{\alpha} b''_{A'}$ with $a'' \sim_{A'} b''$. Hence by Lemma 41 (1) $b_B \xrightarrow{\alpha} b'_{B''}$ with $b'' \sim_{A'} b''$ and $\varnothing \vdash A' <: B''$. But $B' \equiv B''$ because we have $a_B \xrightarrow{\alpha} a'_{B'}$ and $b_B \xrightarrow{\alpha} b'_{B''}$ with the same transition α. By the transitivity of $\sim$ we have $a' \sim_{A'} b'$ and from this and the above we have $a'_{B'} \,\mathcal{S}\, b'_{B'}$. Hence $\mathcal{S}$ is a simulation. ■

A.4 Properties of $\lesssim^\bullet$

Proof of Lemma 19 *If $\mathcal{R} \subseteq \mathcal{S}$ then $\mathcal{R}^\circ \subseteq \mathcal{S}^\circ$.*

Proof Suppose $\mathcal{R} \subseteq \mathcal{S}$ and $\Gamma \vdash e \,\mathcal{R}^\circ\, e' : A$. Then for any Γ-closure, $\vec{\sigma}$, we have $e[\vec{\sigma}] \,\mathcal{R}_A\, e'[\vec{\sigma}]$. Hence $e[\vec{\sigma}] \,\mathcal{S}_A\, e'[\vec{\sigma}]$; hence $\Gamma \vdash e \,\mathcal{S}^\circ\, e' : A$. ■

For instance, if Id is the identity relation on programs (up to alpha-conversion, of course) then Id° is the identity relation on proved programs,

$$\{((\Gamma, e, A), (\Gamma, e, A)) \mid \Gamma \vdash e : A\},$$

and furthermore, $Id^\circ = \widehat{Id^\circ}$. The inclusion $\widehat{Id^\circ} \subseteq Id^\circ$ is immediate from inspection of the rules in Table 8, while the reverse inclusion follows from the result that $\Gamma \vdash e : A$ implies $\Gamma \vdash e \,\widehat{Id^\circ}\, e : A$, which is proved by induction on the derivation of $\Gamma \vdash e : A$.

Proof of Proposition 20 *Suppose $\leftrightarrow$ is a preorder. Then satisfaction of* (Eq Comp) *and* (Eq Subsum) *is equivalent to satisfaction of the following rule*

$$\frac{\Gamma \vdash \mathcal{C}[\bullet_A] : B \qquad \Gamma, \Gamma' \vdash e \leftrightarrow e' : A}{\Gamma \vdash \mathcal{C}[e] \leftrightarrow \mathcal{C}[e'] : B} \text{ (Eq Cong)}$$

where $\mathcal{C}$ is a variable-capturing context, that is, an expression with a single hole of the form $\bullet_A$ (subject to the extra type assignment axiom $\Gamma \vdash \bullet_A : A$), with the hole in the scope of binders for the environment Γ'.

Proof Each instance of (Eq Comp) can be derived from (Eq Cong), using reflexivity and transitivity if need be. If $A <: B$ we can derive $\Gamma \vdash \bullet_A : B$ using (Val Subsumption) so (Eq Subsum) is a special case of (Eq Cong).

For the other direction, we can show that (Eq Comp) and (Eq Subsum) together imply (Eq Cong) by induction on the depth of inference of $\Gamma \vdash \mathcal{C}[\bullet_A] : B$. If $\mathcal{C}[\bullet_A] = \bullet_A$ then $\Gamma \vdash e \leftrightarrow e' : A$ is required, which follows by assumption since Γ' must be empty (because there are no binders in the context). Otherwise one of the (Val $-$) rules must apply.

(Val Fun) Here $\mathcal{C} = \lambda(x{:}B_1)\mathcal{C}_0$, $B \equiv B_1 \to B_2$, $\Gamma, x{:}B_1 \vdash \mathcal{C}_0[\bullet] : B_2$ by a shorter inference, $\Gamma' = x{:}B_1, \Gamma''$ and context $\mathcal{C}_0$ captures the variables in Γ''. By induction hypothesis $\Gamma, x{:}B_1 \vdash \mathcal{C}_0[e] \leftrightarrow \mathcal{C}_0[e'] : B_2$ so then $\Gamma \vdash \mathcal{C}[e] \leftrightarrow \mathcal{C}[e'] : B$ follows by (Eq Comp).

(Val App) Here either $\mathcal{C} = \mathcal{C}_0\, e_0$ or $e_0\, \mathcal{C}_0$ for some expression e_0 and smaller context $\mathcal{C}_0$ (remember there is exactly one hole in $\mathcal{C}$). In the first case we have $\Gamma \vdash \mathcal{C}_0 : C \to B$ and $\Gamma \vdash e_0 : C$ for some type C. By induction hypothesis we have $\Gamma \vdash \mathcal{C}_0[e] \leftrightarrow \mathcal{C}_0[e'] : C \to B$ and then by reflexivity and (Eq Comp) $\Gamma \vdash \mathcal{C}_0[e]\, e_0 \leftrightarrow \mathcal{C}_0[e']\, e_0 : B$ follows. The other case, with $\mathcal{C} = e_0\, \mathcal{C}_0$, is similar.

(Val Subsum) There is some type B' with $B' <: B$ and $\Gamma \vdash \mathcal{C}[\bullet_A] : B'$ by a shorter inference. Then $\Gamma \vdash \mathcal{C}[e] \leftrightarrow \mathcal{C}[e'] : B'$ by induction hypothesis, and $\Gamma \vdash \mathcal{C}[e] \leftrightarrow \mathcal{C}[e'] : B$ by (Eq Subsum).

The other cases are similar. ■

Proof of Lemma 21 *Rules* (Cand Sim), (Cand Comp), (Cand Right) *and* (Cand Subst) *are valid. Moreover, $\lesssim^\bullet$ is the least relation closed under* (Cand Comp) *and* (Cand Right).

Proof First note that $\lesssim^\circ$ is a preorder, since similarity is a preorder and by definition of open extension. Given then that $\lesssim^\circ$ is reflexive, reflexivity of $\lesssim^\bullet$ follows by proving $\Gamma \vdash e \lesssim^\bullet e : A$ by induction on the derivation of $\Gamma \vdash e : A$. Rule (Cand Comp) follows at once from (Cand Def). Now the reflexivity of $\lesssim^\bullet$ means that $Id^\circ \subseteq \lesssim^\bullet$. Compatible refinement is monotone, so we have $\widehat{Id^\circ} \subseteq \widehat{\lesssim^\bullet}$, which is to say $Id^\circ \subseteq \widehat{\lesssim^\bullet}$. Thus we have $\Gamma \vdash e \mathrel{\widehat{\lesssim^\bullet}} e : A$ whenever $\Gamma \vdash e : A$, and hence (Cand Sim) follows from (Cand Def).

For (Cand Right), suppose that $\Gamma \vdash e \lesssim^\bullet e'' : A$ and $\Gamma \vdash e'' \lesssim^\circ e' : A$. By (Cand Def) there must be a type A' and an expression e''' with $\Gamma \vdash e \mathrel{\widehat{\lesssim^\bullet}} e''' : A'$ and $A' <: A$ and $\Gamma \vdash e''' \lesssim^\circ e'' : A$. By transitivity of $\lesssim^\circ$ we have $\Gamma \vdash e''' \lesssim^\circ e' : A$, and hence by (Cand Def) that $\Gamma \vdash e \lesssim^\bullet e' : A$ as required.

(Cand Subst) follows by a routine rule induction on the judgment $\Gamma, x{:}B \vdash e_1 \lesssim^\bullet e_1' : A$.

Finally suppose that $\mathcal{R}$ is another relation to satisfy (Cand Comp), (Cand Right) and (Eq Subsum). We must show that $\lesssim^\bullet \subseteq \mathcal{R}$. We prove by rule induction that $\Gamma \vdash e \lesssim^\bullet e' {:} A$ implies $\Gamma \vdash e \mathcal{R} e' {:} A$. Suppose that $\Gamma \vdash e \lesssim^\bullet e' {:} A$, and hence by (Cand Def) that there is a type A' and an expression e'' with $\Gamma \vdash e \mathrel{\widehat{\lesssim^\bullet}} e'' : A'$ and $A' <: A$ and $\Gamma \vdash e'' \lesssim^\circ e' : A$. By the induction hypothesis we have $\Gamma \vdash e \mathrel{\widehat{\mathcal{R}}} e'' : A'$. So by (Cand Comp) for $\mathcal{R}$ we have $\Gamma \vdash e \mathrel{\mathcal{R}} e'' : A'$, by (Eq Subsum) we have $\Gamma \vdash e \mathcal{R} e'' : A$ and then by (Cand Right) for $\mathcal{R}$ that $\Gamma \vdash e \mathrel{\mathcal{R}} e' : A$ as required. ■

Proof of Lemma 22 *Each of the two rules* (Eq Subsum) *and* (Eq Asm Subsum) *is valid for both* $\lesssim^\circ$ *and* $\lesssim^\bullet$.

Proof First we validate the rules for $\lesssim^\circ$. From Lemma 17 it follows by considering an arbitrary Γ-closure that (Eq Subsum) is valid for $\lesssim^\circ$. As for (Eq Asm Subsum), suppose that $A <: B$ and $\Gamma, x{:}B, \Gamma' \vdash e \lesssim^\circ e' : C$, which is to say that for any Γ-closure $\vec{\sigma}$, any $b{:}B$ and any Γ'-closure $\vec{\sigma}'$, that $e[\vec{\sigma}, {}^{b}\!/\!_{x}, \vec{\sigma}'] \lesssim e'[\vec{\sigma}, {}^{b}\!/\!_{x}, \vec{\sigma}']$. To validate the rule we must show the same thing, except with $b{:}A$. But if $b{:}A$ we know by subsumption that $b{:}B$ too, and hence the desired relation holds.

Now we can validate (Eq Subsum) for $\lesssim^\bullet$. Suppose that $\Gamma \vdash e \lesssim^\bullet e' {:} A$ and $A <: B$. By (Cand Def) there must be a mediating type A' and expression e'' such that $\Gamma \vdash e \mathrel{\widehat{\lesssim^\bullet}} e'' : A'$ and $A' <: A$ and $\Gamma \vdash e'' \lesssim^\circ e' : A$. We have $A' <: B$ and from (Eq Subsum) for $\lesssim^\circ$ we have $\Gamma \vdash e'' \lesssim^\circ e' : B$; hence $\Gamma \vdash e \lesssim^\bullet e' : B$ via (Cand Def).

Finally, suppose that $A <: B$ and $\Gamma, x{:}B, \Gamma' \vdash e \lesssim^\bullet e' {:} C$. We prove by rule induction on the latter that $\Gamma, x{:}A, \Gamma' \vdash e \lesssim^\bullet e' : C$ holds too. From (Cand Def) there must be a type C' and an expression e'' such that $\Gamma, x{:}B, \Gamma' \vdash e \mathrel{\widehat{\lesssim^\bullet}} e'' {:} C'$ and $C' <: C$ and $\Gamma, x{:}B, \Gamma' \vdash e'' \lesssim^\circ e' {:} C$. Either both e and e'' are the same variable, say y, or they both have the same outermost constructor with pairwise related sub-expressions. In the first case, whether $y = x$ or not we can use (Comp x) to derive $\Gamma, x{:}A, \Gamma' \vdash y \mathrel{\widehat{\lesssim^\bullet}} y : C'$. Then by (Eq Asm Subsum) for $\lesssim^\circ$ we have $\Gamma, x{:}A, \Gamma' \vdash y \lesssim^\circ e' {:} C$, hence $\Gamma, x{:}A, \Gamma' \vdash e \lesssim^\bullet e' {:} C$.

In the second case, where e and e'' share the same outermost constructor with sub-expressions related pairwise by $\lesssim^\bullet$, it follows by induction hypothesis that $\Gamma, x{:}A, \Gamma' \vdash e \mathrel{\widehat{\lesssim^\bullet}} e'' : C'$, and then via the same argument as for the

first case we have $\Gamma, x{:}A, \Gamma' \vdash e \lesssim^\bullet e' : C$. Hence (Eq Asm Subsum) is valid for $\lesssim^\bullet$. ■

Lemma 42 *If $a \mathcal{S}_A b$ and $a \mapsto a'$ then $a' \mathcal{S}_A b$ too.*

Proof We proceed by a rule induction on the reduction $a \mapsto a'$. In any case we know from (Cand Def) there is a type A' and a program c such that $\varnothing \vdash a \mathrel{\widehat{\lesssim^\bullet}} c : A'$ and $A' <: A$ and $\varnothing \vdash c \lesssim^\circ b : A$.

(Red Experiment) There is an experiment $\mathcal{E}$ and programs a_1 and a_1' such that $a \equiv \mathcal{E}[a_1]$, $a' \equiv \mathcal{E}[a_1']$ and $a_1 \mapsto a_1'$. Since $]eic\mathcal{E}[a_1] \mathrel{\widehat{\lesssim^\bullet}} c[A']$ we can show, by considering each possible form of the experiment $\mathcal{E}$, that there is a program c_1, an experiment $\mathcal{E}_1$, and a type B such that $c \equiv \mathcal{E}_1[c_1]$ and $\varnothing \vdash a_1 \lesssim^\bullet c_1 : B$. Now by the induction hypothesis it follows that $\varnothing \vdash b_1 \lesssim^\bullet c_1 : B$ too. We can check that each possible form of $\mathcal{E}_1$ must be such that $\varnothing \vdash \mathcal{E}[b_1] \mathrel{\widehat{\lesssim^\bullet}} \mathcal{E}_1[c_1] : A'$ (the interesting cases are (Comp Fun) and (Comp Update), and hence by (Cand Def) and $\varnothing \vdash c \lesssim^\circ b : A$ we have $\varnothing \vdash a' \lesssim^\bullet b : A$.

(Red If True)

- From $a \mapsto a'$ by (Red If True) we have $a \equiv \mathtt{if}(\mathtt{true}, a_2, a_3)$ and $a' \equiv a_2$.
- From $\varnothing \vdash a \mathrel{\widehat{\lesssim^\bullet}} c : A'$ by (Comp If) $c \equiv \mathtt{if}(c_1, c_2, c_3)$ with $\varnothing \vdash \mathtt{true} \lesssim^\bullet c_1 : \mathtt{Bool}$ and $\varnothing \vdash a_i \lesssim^\bullet c_i : A'$ for $2 \leq i \leq 3$.
- From (Cand Def) there must be a type $B <: \mathtt{Bool}$ and a program u such that $\varnothing \vdash \mathtt{true} \mathrel{\widehat{\lesssim^\bullet}} u : B$ and $\varnothing \vdash u \lesssim^\circ c_1 : \mathtt{Bool}$.
- By (Comp True) $B \equiv \mathtt{Bool}$ and $u \equiv \mathtt{true}$.

Now we can appeal to the similarity, $\varnothing \vdash u \lesssim^\circ c_1 : \mathtt{Bool}$.

- Since $u \xrightarrow{\mathtt{true}} \mathbf{0}$, then c_1 must match that transition, hence by Lemma 12 $c_1 \Downarrow \mathtt{true}$, so by (Red If True) $c \mapsto^* c_2$.

Putting these together with appeal to (Eq Subsum) for $\lesssim^\bullet$, we have the following relations at the type A.

$$a' \equiv a_2 \lesssim^\bullet c_2 \sim c \lesssim b$$

and so by (Cand Right) and transitivity of $\lesssim$ we have $a' \mathcal{S}_A b$.

(Red If False) Similar to the case for (Red If True).

(Red Select)

- From $a \mapsto a'$ by (Red Select), we must have $a \equiv v.\ell_j$ where $v \equiv [\ell_i = \varsigma(x_i{:}B_i)e_i]_{i\in I}$ and $j \in I$ and $a' \equiv e_j[a/x_j]$.
- By Lemma 1(4), there's some type B such that each $B_i \equiv B$.
- Since $\varnothing \vdash a \mathrel{\widehat{\lesssim^\bullet}} c : A'$, then by (Comp Select), c must have the same outermost structure as a: that is, $c \equiv c_1.\ell_j$ and $\varnothing \vdash v \lesssim^\bullet c_1 : [\ell_i{:}A_i]_{i\in I'}$ and $A_j \equiv A'$ and $j \in I'$.
- By (Cand Def) there must be a type $B' <: [\ell_i{:}A_i]_{i\in I'}$ and a program u such that $\varnothing \vdash v \mathrel{\widehat{\lesssim^\bullet}} u : B'$ and $\varnothing \vdash u \lesssim^\circ c_1 : [\ell_i{:}A_i]_{i\in I'}$.
- By (Comp Object) we must have $B' \equiv B \equiv [\ell_i{:}A_i]_{i\in I}$ and hence $j \in I' \subseteq I$. The program u must have the same outermost form as v, that is, $u \equiv [\ell_i = \varsigma(x_i{:}B')e'_i]_{i\in I}$ and $x_i{:}B' \vdash e_i \lesssim^\bullet e'_i : A_i$ for each $i \in I$.
- By (Cand Comp) we have $\varnothing \vdash v \lesssim^\bullet u : B'$.
- From $x_j{:}B' \vdash e_j \lesssim^\bullet e'_j : A_j$ and (Cand Subst) we have $\varnothing \vdash e_j[v/x_j] \lesssim^\bullet e'_j[u/x_j] : A_j$.

Now we can appeal to the similarity, $\varnothing \vdash u \lesssim^\circ c_1 : B$.

- Since $j \in J$, we can derive $u_B \xrightarrow{\ell_j} (u.\ell_j)_{A_j}$ from (Trans Select). Since $u \lesssim_B c_1$, value c_1 can match this transition, so we get that $u.\ell_j \lesssim_{A_j} c_1.\ell_j$.

Putting these together we have the following relations, at type A.

$$a' \lesssim^\bullet e'_j[u/x_j] \sim u.\ell_j \lesssim c_1.\ell_j \equiv c \lesssim b$$

and so by (Cand Right) and transitivity of $\lesssim$ we have $a'\mathcal{S}_A b$.

(Red Update)

- From $a \mapsto a'$ by (Red Update) we must have $a \equiv v.\ell_j \Leftarrow \varsigma(x{:}B)e$, where $v \equiv [\ell_i = \varsigma(x_i{:}B_i)e_i]_{i\in I}$ and $a' \equiv [\ell_j = \varsigma(x{:}B_j)e, \ell_i = \varsigma(x_i{:}B_i)e_i]_{i\in I-\{j\}}$.
- By Lemma reflemma:static-facts, there is some type B' such that each $B_i \equiv B'$.
- From $\varnothing \vdash a \mathrel{\widehat{\lesssim^\bullet}} c : A'$ by (Comp Update) we must have $B \equiv A' \equiv [\ell_i{:}A_i]_{i\in I'}$, with $j \in I' \subseteq I$. The program c must have the same outermost form as a, namely $c \equiv c_1.\ell_j \Leftarrow \varsigma(x{:}A')e'$ with $\varnothing \vdash v \lesssim^\bullet c_1 : A'$ and $x_j{:}A' \vdash e \lesssim^\bullet e' : A_j$.

- By (Cand Def), there's some type $B'' <: A'$ and some program u such that $\varnothing \vdash v \mathrel{\widehat{\lesssim^\bullet}} u : B''$ and $\varnothing \vdash e \lesssim^\circ c_1 : A'$.
- By (Comp Object) $B'' \equiv [\ell_i{:}A_i]_{i\in I}$, so $I = I'$ and $B'' \equiv A'$ and u has the same outermost form as v, that is, $u \equiv [\ell_i = \varsigma(x_i{:}A')e'_i]_{i\in i}$ and $x_i{:}A' \vdash e_i \lesssim^\bullet e'_i : A_i$ for all $i \in I$.
- By (Eq Asm Subsum) for $\lesssim^\bullet$ we have $x_j{:}A' \vdash e \lesssim^\bullet e' : A_j$ and by (Comp Object) we have $\varnothing \vdash a' \mathrel{\widehat{\lesssim^\bullet}} c' : A'$ where $c' \equiv [\ell_i = \varsigma(x_i{:}A')e'_i, \ell_j = \varsigma(x_j{:}A')e']_{i\in I-\{j\}}$.

Now we appeal to the similarity, $\varnothing \vdash u \lesssim^\circ c_1 : A'$.

- Let action α be $\ell_j \Leftarrow \varsigma(x)e'$. By (Trans Update) we have $u_{A'} \xrightarrow{\alpha} (u.\ell_j \Leftarrow \varsigma(x{:}A')e')_{A'}$. Hence there is c_2 such that $(c_1)_{A'} \xrightarrow{\alpha} (c_2)_{A'}$ and $u.\ell_j \Leftarrow \varsigma(x{:}B)e' \sim_{A'} c_2 \sim_{A'} c_1.\ell_j \Leftarrow \varsigma(x{:}A')e' \equiv c$.

Putting these together yields

$$\varnothing \vdash a' \mathrel{\widehat{\lesssim^\bullet}} c' : A' \quad A' <: A \quad \varnothing \vdash c' \lesssim u.\ell_j \Leftarrow \varsigma(x{:}A')e' \sim c \lesssim b : A$$

and hence by (Cand Def) and transitivity of $\lesssim^\bullet$, $a' \mathcal{S}_A b$ as required.

(Red Unfold)

- From $a \mapsto a'$ by (Red Unfold) we have $a \equiv \texttt{unfold}(v)$ for some value $v \equiv \texttt{fold}(B, v')$ and some type B, with $a' \equiv v'$.
- From $\varnothing \vdash a \mathrel{\widehat{\lesssim^\bullet}} c : A'$ by (Comp Unfold) there must be some type expression E such that $A' \equiv E[\mu(X)E/X]$ and $c \equiv \texttt{unfold}c_1$ and $\varnothing \vdash \texttt{fold}(B, v) \lesssim^\bullet c_1 : \mu(X)E$.
- From (Cand Def), there must be a type $B' <: \mu(X)E$ and a program u such that $\varnothing \vdash v \mathrel{\widehat{\lesssim^\bullet}} u : B'$ and $\varnothing \vdash u \lesssim^\circ c_1 : \mu(X)E$.
- By (Comp Fold), $B \equiv B' \equiv \mu(X)E'$ for some type expresson E', and $u \equiv \texttt{fold}(B, u')$ and $\varnothing \vdash v' \lesssim^\bullet u' : E'[\mu(X)E'/X]$.
- By (Sub Rec) we have $X{<:}\texttt{Top} \vdash E'{<:}E$; hence by Lemma 1(7) we have $\varnothing \vdash E'[\mu(X)E'/X] <: E[\mu(X)E/X] \equiv A'$, so by (Eq Subsum) for $\lesssim^\bullet$ we have $\varnothing \vdash v' \lesssim^\bullet u' : A'$.

Now we can appeal to the similarity, $\varnothing \vdash u \lesssim^\circ c_1 : \mu(X)E$.

- Since $u_{\mu(X)E} \xrightarrow{\texttt{unfold}} \texttt{unfold}(u)_{A'}$, then $c_{1\,\mu(X)E} \xrightarrow{\texttt{unfold}} \texttt{unfold}(c_1)_{A'}$ with $\varnothing \vdash \texttt{unfold}(u) \lesssim \texttt{unfold}(c_1) : A'$.

Putting these together with appeal to (Eq Subsum) for $\lesssim^\bullet$ and $\lesssim$, we have the following relations at the type A.

$$a' \equiv v' \lesssim^\bullet u' \sim \mathtt{unfold}(u) \lesssim \mathtt{unfold}(c_1) \equiv c \lesssim b$$

and so by (Cand Right) and transitivity of $\lesssim$ we have $a' \mathcal{S}_A b$.

(Red App)

- From $a \mapsto a'$ by (Red App) we must have $a \equiv (v\, a_2)$ and $v \equiv \lambda(x{:}B)e$ and $a' \equiv e[a_2/x]$.
- From $\varnothing \vdash a \mathrel{\widehat{\lesssim^\bullet}} c : A'$ by (Comp App) we have that c has the same outermost form as a, that is, $c \equiv (c_1\, c_2)$ where $\varnothing \vdash v \lesssim^\bullet c_1{:}B \to A'$ and $\varnothing \vdash a_2 \lesssim^\bullet c_2 : B$.
- By (Cand Def) here must be a type $B' <: B \to A'$ and a program u such that $\varnothing \vdash v \mathrel{\widehat{\lesssim^\bullet}} u : B'$ and $\varnothing \vdash u \lesssim^\circ c_1 : B \to A'$.
- By (Comp Fun), $u \equiv \lambda(x{:}B)e'$ and $B' \equiv B \to A''$ for some type $A'' <: A'$ and $x{:}B \vdash e \lesssim^\bullet e' : A''$.
- By (Cand Subst) $\varnothing \vdash e[a_2/x] \lesssim^\bullet e'[c_2/x] : A''$.

Now we can appeal to the similarity $\varnothing \vdash u \lesssim^\circ c_1 : B \to A'$.

- By (Trans App), $u_{B\to A'} \xrightarrow{@c_2} u(c_2)_{A'}$, so $c_{1\,B\to A'} \xrightarrow{@c_2} c_1(c_2)_{A'}$. Hence by (Eq Subsum) for $\lesssim^\circ$ we have $u(c_2) \lesssim^\circ_A c_1(c_2)$.

Putting these together, we have the following relations at type A.

$$a' \equiv e[a_2/x] \lesssim^\bullet f[c_2/x] \sim u(c_2) \lesssim c_1(c_2) \equiv c \lesssim b$$

and so by (Cand Right) and transitivity of $\lesssim$ we have $a' \mathcal{S}_A b$.

(Red Dynamic Default) Here $a \equiv \mathtt{typecase}(\mathtt{tag}(C, a_1), (x{:}B)a_2, a_3)$ where $a_1{:}C$ and $\neg(C <: B)$. From (Comp Typecase), program c has the same outer form as a, that is, $c \equiv \mathtt{typecase}(c_1, (x{:}B)c_2, c_3)$ with $\varnothing \vdash \mathtt{tag}(C, a_1) \lesssim^\bullet c_1 : \mathtt{Dynamic}$ and $x{:}B \vdash a_2 \lesssim^\bullet c_2 : A'$ and $\varnothing \vdash a_3 \lesssim^\bullet c_3 : A'$. Now we appeal to the similarity, $\varnothing \vdash c \lesssim b : A$. Suppose $c_3 \xrightarrow{\alpha} c_3'$, then by (Trans Red), $c \xrightarrow{\alpha} c_3'$ too; hence $b \xrightarrow{\alpha} b'$ with $c_3' \lesssim b'$; hence by appeal to (Eq Subsum) for $\lesssim^\bullet$, we have, at the type A, $a' \equiv a_3 \lesssim^\bullet c_3 \lesssim b$ and so by (Cand Right) $a' \mathcal{S}_A b$.

(Red Dynamic Match)

- From $a \mapsto a'$ and (Red Dynamic Match), $a \equiv \texttt{typecase}(a_1, (x{:}B)a_2, a_3)$ and a_1 is $\texttt{tag}(C, a_1')$ and $a_1'{:}C$ and $C <: B$.
- From $\varnothing \vdash a \mathrel{\widehat{\lesssim^\bullet}} c : A'$ by (Comp Typecase), $c \equiv \texttt{typecase}(c_1, (x{:}B)c_2, c_3)$ with $\varnothing \vdash a_1 \lesssim^\bullet c_1 : \texttt{Dynamic}$ and $x{:}B \vdash a_2 \lesssim^\bullet c_2 : A'$ and $\varnothing \vdash a_3 \lesssim^\bullet c_3 : A'$.
- From (Cand Def) and (Comp Dynamic) there must be a value u_1 such that $\varnothing \vdash a_1 \mathrel{\widehat{\lesssim^\bullet}} u_1 : A''$ and $\varnothing \vdash u_1 \lesssim^\circ c_1 : \texttt{Dynamic}$ for some type $A'' <: \texttt{Dynamic}$. But there is no subtype of $\texttt{Dynamic}$, so $A'' \equiv \texttt{Dynamic}$.
- By (Comp Dynamic) u_1 must have the same outer shape as a_1, that is, u_1 is $\texttt{tag}(C, u_1')$ where $\varnothing \vdash a_1 \lesssim^\bullet u_1' : C$.

Now we can appeal to the similarity, $\varnothing \vdash u_1 \lesssim^\circ c_1 : \texttt{Dynamic}$.

- Since $u_1 \xrightarrow{\texttt{tag}C} u_1'$, then $c_1 \xrightarrow{\texttt{tag}C} c_1'$ with $\varnothing \vdash u_1' \lesssim^\circ c_1' : C$; hence $c_1 \Downarrow \texttt{tag}(C, c_1')$; hence by (Red Dynamic Match) $c \Downarrow c_2[{}^{c_1}\!/_x]$.
- We have $x{:}B \vdash a_2 \lesssim^\bullet c_2{:}A'$, and by (Cand Right) we have $a_1' \lesssim^\bullet c_1'$. By (Eq Assum Subsum) we deduce $x{:}C \vdash a_2 \lesssim^\bullet c_2 : A'$; hence by (Cand Subst) $\varnothing \vdash a' \equiv a_2[{}^{a_1}\!/_x] \lesssim^\bullet c_2[{}^{c_1}\!/_x]{:}A'$ and by (Eq Subsum) we have the same relation at the type A.

Putting these together we have the following relations at the type A.

$$a' \equiv a_2[{}^{a_1}\!/_x] \lesssim^\bullet c_2[{}^{c_1}\!/_x] \sim c \lesssim b$$

and so by (Cand Right) and transitivity of $\lesssim$ we have $a' \mathcal{S}_A b$.

(Red Record)

- From $a \mapsto a'$ by (Red Record) we have $a \equiv v.\ell_j$ where $v \equiv \texttt{record}(\ell_i = a_i)_{i \in I}$, $j \in I$ and $a' \equiv a_j$.
- From $\varnothing \vdash a \mathrel{\widehat{\lesssim^\bullet}} c : A'$ by (Comp Select) we have that $c \equiv c_1.\ell_j$, $\varnothing \vdash v \lesssim^\bullet c_1 : B$ where $B \equiv \texttt{record}(\ell_i{:}A_i)_{i \in I'}$, $A_j \equiv A$ and $j \in I' \subseteq I$.
- By (Cand Def) there exists a type $B' <: B$ and a program u with $\varnothing \vdash v \mathrel{\widehat{\lesssim^\bullet}} u : B'$ and $\varnothing \vdash u \lesssim^\circ c_1 : B$.
- By (Comp Record), $I = I'$, $B' \equiv B \equiv \texttt{record}(\ell_i{:}A_i)_{i \in I}$, $u \equiv \texttt{record}(\ell_i = u_i)_{i \in I}$ and $\varnothing \vdash a_i \lesssim^\bullet u_i : A_i$ for each $i \in I$.

Now we can appeal to the similarity, $\varnothing \vdash u_1 \lesssim^\circ c_1 : B$.

- By (Trans Record), $u_B \xrightarrow{\ell_j} u.\ell_{j\,A_j}$ and by the similarity c_1 can match this transition with $c_{1B} \xrightarrow{\ell_j} c_1.\ell_{j\,A_j}$ and $u.\ell_j \lesssim_{A_j} c_1.\ell_j$.

By appeal to (Eq Subsum) for $\lesssim^\bullet$, we have the following relations at the type A.

$$a' \equiv a_j \lesssim^\bullet u_j \lesssim u.\ell_j \lesssim c_1.\ell_j \equiv c \lesssim b$$

and so by (Cand Right) and transitivity of $\lesssim$ we have $a' \mathcal{S}_A b$.

(Red Case)

- From $a \mapsto a'$ by (Red Case) $a \equiv \mathtt{case}(v, (x)e_1, \ldots, (x)e_n)$ where $v \equiv \mathtt{variant}(\ell_j = v')$ for some $j \in I$, and $a' \equiv e_j[v'/x]$.
- From $\varnothing \vdash a \mathrel{\widehat{\lesssim^\bullet}} c : A'$ by (Comp Case) $c \equiv \mathtt{case}(c_1, (x)c'_1, \ldots, (x)c'_n)$ with $\varnothing \vdash v \lesssim^\bullet c_1 : B$ where $B \equiv \mathtt{variant}(\ell_i{:}A_i)_{i\in I}$ and $x{:}A_i \vdash e_i \lesssim^\bullet c'_i : A'$ for each $i \in I$.
- From (Cand Def) there must be a type $B' <: B$ and a program u_1 such that $\varnothing \vdash v \mathrel{\widehat{\lesssim^\bullet}} u_1 : B'$ and $\varnothing \vdash u_1 \lesssim^\circ c_1 : B$.
- By (Comp Variant) u_1 must have the same outermost form as v, that is, u_1 is $\mathtt{variant}(\ell_j = u')$ where $\varnothing \vdash v' \lesssim^\bullet u' : A_j$.
- By (Cand Comp) we have $\varnothing \vdash v \lesssim^\bullet u : B$, and by (Cand Subst) and (Eq Subsum) we have $\varnothing \vdash e_j[v'/x] \lesssim^\bullet c'_j[u'/x] : A$.

Now we can appeal to the similarity, $\varnothing \vdash u_1 \lesssim^\circ c_1 : B$.

- Since $u_1 \xrightarrow{\ell_j} u'$, then $c_1 \xrightarrow{\ell_j} c'$ with $\varnothing \vdash u' \lesssim^\circ c' : A_j$; hence $c_1 \Downarrow \mathtt{variant}(\ell_j = c')$; hence by (Red Case) $c \Downarrow c'_j[c'/x]$.

Putting these together we have the following relations at the type A.

$$a' \equiv e_j[v'/x] \lesssim^\bullet c'_j[u'/x] \sim c \lesssim b$$

and so by (Cand Right) and transitivity of $\lesssim$ we have $a' \mathcal{S}_A b$. ■

Proof of Lemma 23 *If $P \,\mathcal{S}\, Q$ and $P \xrightarrow{\alpha} P'$ there is Q' with $Q \xrightarrow{\alpha} Q'$ and $P' \,\mathcal{S}\, Q'$.*

Proof The proof of (2) is by rule induction on the transition $P \xrightarrow{\alpha} P'$. Suppose proved programs P and Q are a_A and b_A respectively, with $\varnothing \vdash a \lesssim^\bullet b : A$. In each case there is a program c with $\varnothing \vdash a \lesssim^\bullet c : A$ and $\varnothing \vdash c \lesssim^\circ b : A$. We consider the transition rules in turn.

(Trans Red) Here $P \xrightarrow{\alpha} P'$ derives from $a \mapsto a'$ and ${a'}_A \xrightarrow{\alpha} P'$. By Lemma 42, ${a'}_A \mathcal{S} Q$, so by induction hypothesis there is Q' with $Q \xrightarrow{\alpha} Q'$ and $P' \lesssim^\bullet Q'$.

(Trans Select, Update, Unfold, App and Record) Here $a_A \xrightarrow{\mathcal{E}\downarrow} \mathcal{E}[a]_B$, and also $c_A \xrightarrow{\mathcal{E}\downarrow} \mathcal{E}[c]_B$ unconditionally. Since $\lesssim^\bullet$ is a precongruence, it follows that $\mathcal{E}[a] \lesssim^\bullet_B \mathcal{E}[c]$, and we can fill in the following diagram,

$$\begin{array}{ccccccc} P & \equiv & a_A & \lesssim^\bullet & c_A & \lesssim & Q \\ & & \mathcal{E}\downarrow \big\downarrow & & \mathcal{E}\downarrow \big\downarrow & & \big\downarrow \mathcal{E}\downarrow \\ P' & \equiv & \mathcal{E}[a]_B & \lesssim^\bullet & \mathcal{E}[c]_B & \lesssim & Q' \end{array}$$

to yield $P' \mathcal{S} Q'$ as required.

(Trans Bool) Here $A \equiv \texttt{Bool}$ and we take $P \xrightarrow{\texttt{true}} \mathbf{0}$ (the case for `false` is similar); hence $a \equiv \texttt{true}$; hence $c \equiv \texttt{true}$ too; hence $c_A \xrightarrow{\texttt{true}} \mathbf{0}$ too, and we can fill in the following diagram,

$$\begin{array}{ccccccc} P & \equiv & a_A & \lesssim^\bullet & c_A & \lesssim & Q \\ & & \texttt{true} \big\downarrow & & \texttt{true} \big\downarrow & & \big\downarrow \texttt{true} \\ P' & \equiv & \mathbf{0} & \lesssim^\bullet & \mathbf{0} & \lesssim & Q' \end{array}$$

to yield $P' \mathcal{S} Q'$ as required.

(Trans Dynamic) Here $A \equiv \texttt{Dynamic}$ and $a \equiv \texttt{tag}(A', a')$, and $a_A \xrightarrow{\texttt{tag}A'} {a'}_{A'}$. Since $\varnothing \vdash a \mathrel{\widehat{\lesssim^\bullet}} c : \texttt{Dynamic}$ it follows by (Comp Dynamic) that $c \equiv \texttt{tag}((,A)', c')$ with $\varnothing \vdash a' \lesssim^\bullet c' : A'$; hence $c_A \xrightarrow{\texttt{tag}A'} {c'}_{A'}$ too; hence $Q \xrightarrow{\texttt{tag}A'} Q'$, and we can fill in the following diagram,

$$\begin{array}{ccccccc} P & \equiv & a_A & \lesssim^\bullet & c_A & \lesssim & Q \\ & & \texttt{tag}A' \big\downarrow & & \texttt{tag}A' \big\downarrow & & \big\downarrow \texttt{tag}A' \\ P' & \equiv & {a'}_{A'} & \lesssim^\bullet & {c'}_{A'} & \lesssim & Q' \end{array}$$

to yield $P' \mathcal{S} Q'$ as required.

(Trans Variant) Here $A \equiv \mathtt{variant}(\ell_i{:}A_i)_{i\in I}$ and $a \equiv \mathtt{variant}(\ell_j = a')$ for some $j \in I$, and $a_A \xrightarrow{\ell_j} a'_{A_j}$. Since $\varnothing \vdash a \mathrel{\widehat{\lesssim^\bullet}} c : A$ it follows by (Comp Variant) that $c \equiv \mathtt{variant}(\ell_j = c')$ with $\varnothing \vdash a' \lesssim^\bullet c' : A_j$; hence $c_A \xrightarrow{\ell_j} c'_{A_j}$ too; hence $Q \xrightarrow{\ell_j} Q'$, and we can fill in the following diagram,

$$\begin{array}{ccccccc} P & \equiv & a_A & \lesssim^\bullet & c_A & \lesssim & Q \\ & & \downarrow{\scriptstyle \ell_j} & & \downarrow{\scriptstyle \ell_j} & & \downarrow{\scriptstyle \ell_j} \\ P' & \equiv & a'_{A_j} & \lesssim^\bullet & c'_{A_j} & \lesssim & Q' \end{array}$$

to yield $P' \mathrel{\mathcal{S}} Q'$ as required. ■

A.5 Bisimilarity equals contextual equivalence

Proof of Lemma 24 *Both $\lesssim \subseteq \sqsubseteq$ and $\sim \subseteq \simeq$*

Proof First we shall prove $\lesssim \subseteq \sqsubseteq$. Suppose that $a \lesssim_A b$. To see that $a \sqsubseteq_A b$ we must consider an arbitrary expression e such that $x{:}A \vdash e{:}\mathtt{Bool}$ and show that $e[a/x]\Downarrow$ implies $e[b/x]\Downarrow$. By supposition we have $\varnothing \vdash a \lesssim^\circ b : A$, and so by the substitution property of $\lesssim^\circ$ (Cand Subst) it follows that $\varnothing \vdash e[a/x] \lesssim^\circ e[b/x]{:}\mathtt{Bool}$. In fact we have $e[a/x] \lesssim_B e[b/x]$. Since $e[a/x]\Downarrow v$, it must be able to do the transition v, hence $e[b/x]$ must match it, and therefore $e[b/x]\Downarrow$ too (by Lemma 12). Having now shown that $\lesssim \subseteq \sqsubseteq$, $\sim \subseteq \simeq$ follows by symmetry. ■

Proof of Lemma 25 *Contextual order is a simulation.*

Proof Suppose then that $a \sqsubseteq_A b$ and that $a_A \xrightarrow{\alpha} P$. To show that $\sqsubseteq$ is a simulation we must find some Q such that $b_A \xrightarrow{\alpha} Q$ with $P \sqsubseteq Q$. We prove this by induction on the derivation of $a_A \xrightarrow{\alpha} P$.

(Trans Red) Here $a \mapsto a'$ and $a'_A \xrightarrow{\alpha} P$. By part (1) of Proposition 26, $a \sim_A a'$, and by Lemma 24, $a' \sqsubseteq_A a$; hence $a' \sqsubseteq_A b$, and so by the induction hypothesis there exists Q such that $b_A \xrightarrow{\alpha} Q$ and $P \sqsubseteq Q$.

(Trans Select, Update, Unfold, App and Record) Here $a_A \xrightarrow{\mathcal{E}\downarrow} \mathcal{E}[a]_B$, and hence $-{:}A \vdash \mathcal{E} : B$, and so $b_A \xrightarrow{\mathcal{E}\downarrow} \mathcal{E}[b]_B$ unconditionally. By the substitution property of contextual order, we deduce $\mathcal{E}[a] \sqsubseteq_B \mathcal{E}[b]$.

(Trans Bool) Take $\alpha \equiv \mathtt{true}$, $a_A \equiv \mathtt{true}_{\mathtt{Bool}}$ and $P \equiv \mathbf{0}$ (the case for $\mathtt{false}$ is similar). Consider the context $C \equiv \mathtt{if}(-, \mathtt{true}, \Omega^{\mathtt{Bool}})$. Since $C[a] \Downarrow \mathtt{true}$, we must have $C[b] \Downarrow$ too; therefore $b \Downarrow \mathtt{true}$ and thus by repeated applciation of (Trans Red) we have $b_A \xrightarrow{\alpha} Q$, where $Q \equiv \mathbf{0}$ and thus $P \sqsubseteq\!\!\!\approx Q$ as required.

(Trans Dynamic) Here we have $A \equiv \mathtt{Dynamic}$ and $a \Downarrow \mathtt{tag}(A', a')$, and $a_A \xrightarrow{\mathtt{tag}A'} a'_{A'}$. Consider the context $C \equiv \mathtt{typecase}(-, (x{:}A')\mathtt{true}, \Omega^{\mathtt{Bool}})$. Since $C[a] \Downarrow \mathtt{true}$ we must have $C[b] \Downarrow$; so we must have $b \Downarrow \mathtt{tag}(A'', b')$ with $A'' <: A'$. By considering the context $C \equiv \mathtt{typecase}(-, (x{:}A'')\Omega^{\mathtt{Bool}}, \mathtt{true})$ we deduce that $A' <: A''$ and hence by the antisymmetry property of subtyping we have $A' \equiv A''$. By repeated application of (Trans Red) and by (Trans Dynamic) we have $b_A \xrightarrow{\mathtt{tag}A'} b'_{A'}$. Now suppose there were some context C' such that $C'[a'] \Downarrow$ and $C'[b']\Uparrow$; then consider the context $C'' \equiv \mathtt{typecase}(-, (x{:}A')C'[x], \mathtt{true})$. We have $C''[a] \Downarrow$ and $C''[b]\Uparrow$. But this is contradictory to $a \sqsubseteq\!\!\!\approx_A b$; so there is no such context C'; hence $a' \sqsubseteq\!\!\!\approx_{A'} b'$.

(Trans Variant) Here we have $A \equiv \mathtt{variant}(\ell_i{:}A_i)_{i\in I}$ and $a \Downarrow \mathtt{variant}(\ell_j = a')$ and $a_A \xrightarrow{\ell_j} a'_{A_j}$. Consider the context $C \equiv \mathtt{case}(-, (x)\Omega^{\mathtt{Bool}}, \ldots, (x)\mathtt{true}, \ldots, (x)\Omega^{\mathtt{Bool}})$. Since $C[a] \Downarrow \mathtt{true}$ we must have $C[b] \Downarrow$; so we must have $b \Downarrow \mathtt{variant}(\ell_j = b')$, and by repeated application of (Trans Red) and by (Trans Variant) we have $b_A \xrightarrow{\ell_j} b'_{A_j}$. Now suppose there were some context C' such that $C'[a'] \Downarrow$ and $C'[b']\Uparrow$; then consider the context $C'' \equiv \mathtt{case}(-, (x)\Omega^{\mathtt{Bool}}, \ldots, (x)C'[x], \ldots, (x)\Omega^{\mathtt{Bool}})$. We have $C''[a] \Downarrow$ and $C''[b]\Uparrow$. But this is contradictory to $a \sqsubseteq\!\!\!\approx_A b$; so there is no such context C'; hence $a' \sqsubseteq\!\!\!\approx_{A'} b'$.

All cases considered, it follows that contextual order is a simulation. ■

A.6 Validation of the equational theory

Lemma 43 (Eval Fold) *Let $A \equiv \mu(X)E$ and suppose $\Gamma \vdash e : A$. Then $\Gamma \vdash \mathtt{fold}(A, \mathtt{unfold}(e)) \sim^{\circ} e : A$.*

Proof Let $B \equiv E[{}^{A}\!/\!_{X}]$. By definition of open extension it is enough to show that that pair

$$(\mathtt{fold}(A, \mathtt{unfold}(a))_A, a_A)$$

is an element of $\sim$ for any program $a{:}A$. The only transitions of these two programs are $\mathtt{fold}(A, \mathtt{unfold}(a))_A \xrightarrow{\mathtt{unfold}} \mathtt{unfold}(\mathtt{fold}(A, \mathtt{unfold}(a)))_B$, and

$a_A \stackrel{\texttt{unfold}}{\longrightarrow} \texttt{unfold}(a)_B$, where $\texttt{unfold}(\texttt{fold}(A, \texttt{unfold}(a))) \sim_B \texttt{unfold}(e)$ by (Eval Unfold). Hence the pair is a member of $\langle\sim\rangle = \sim$. ■

It only remains to validate (Eq Sub Object), the following rule.

$$\frac{A' \equiv [\ell_i{:}B_i]_{i\in I} \quad A'' \equiv [\ell_i{:}B_i, \ell_j{:}B_j]_{i\in I, j\in J} \quad I \cap J = \varnothing \quad \Gamma, x_i{:}A' \vdash e_i : B_i\ (i \in I) \quad \Gamma, x_j{:}A'' \vdash e_j : B_j\ (j \in J)}{\Gamma \vdash [\ell_i = \varsigma(x_i{:}A')e_i]_{i\in I} \leftrightarrow [\ell_i = \varsigma(x_i{:}A'')e_i]_{i\in I\cup J} : A'} \text{ (Eq Sub Object)}$$

We shall validate (Eq Sub Object) via a direct proof of contextual equivalence, similar to the Activity Lemma of Plotkin (1977). We shall need the following definitions.

Let there be a collection of distinguished variables $\{-_n \mid n \in \mathbb{N}\}$, which we call **indexed holes**. Suppose B is a type and $\vec{A} = A_1, \ldots, A_n$ is a list of types. Let an $(\vec{A}, B)$**-context** be an expression e of type B containing indexed holes with types in the list $\vec{A}$, that is, $-_1{:}A_1, \ldots, -_n{:}A_n \vdash e : B$. If e is an $(\vec{A}, B)$-context we write $e[e_1, \ldots, e_n]$ as short for $e[{}^{e_1}/_{-1}] \cdots [{}^{e_n}/_{-n}]$. Finally, iff $\mathcal{R} \subseteq \mathit{Rel}$ let its **context closure** be the relation $\mathcal{R}^{\mathcal{C}} \subseteq \mathit{Rel}$ given by

$$\mathcal{R}^{\mathcal{C}}_A \stackrel{\text{def}}{=} \{(e[\vec{a}], e[\vec{b}]) \mid \exists \vec{B}(e \text{ is an } (\vec{B}, A)\text{-context } \&\ a_i\ \mathcal{R}_B\ b_i \text{ for each } i)\}.$$

Let (R1) and (R2) be the following properties of a relation $\mathcal{R} \subseteq \mathit{Rel}$.

(R1) Whenever $a\ \mathcal{R}_A\ b$ both a and b are values.

(R2) Whenever $a\ \mathcal{R}_A\ b$ and experiment $\mathcal{E}$ satisfies $-{:}A \vdash \mathcal{E} : B$ there are a' and b' such that $\mathcal{E}[a] \mapsto a'$, $\mathcal{E}[b] \mapsto b'$ and $a'\ \mathcal{R}^{\mathcal{C}}_B\ b'$.

We shall show in general that any relation possessing these two properties is contained in contextual equivalence, and then prove (Eq Sub Object) by exhibiting such a relation.

Lemma 44 *Suppose $\mathcal{R} \subseteq \mathit{Rel}$ is a relation satisfying* (R1) *and* (R2)*. Then whenever $a\ \mathcal{R}^{\mathcal{C}}_A\ b$ and $a \mapsto a'$ there is b' such that $b \mapsto b'$ and $a'\ \mathcal{R}^{\mathcal{C}}_A\ b'$.*

Proof By induction on the derivation of $a \mapsto a'$. From $a\ \mathcal{R}^{\mathcal{C}}_A\ b$ there must be a $(\vec{B}, A)$-context e, family $(a_i, b_i) \in \mathcal{R}_{B_i}$ such that $a \equiv e[\vec{a}]$ and $b \equiv e[\vec{b}]$. Since $e[\vec{a}]$ can reduce and each a_i must be a value by (R1), context e cannot be an indexed hole or a value but must equal $\mathcal{E}[e']$ where $\vec{-}{:}\vec{B}, -{:}C \vdash \mathcal{E}{:}A$ and $\vec{-}{:}\vec{B} \vdash e' : C$ for some type C, and where $\mathcal{E}$ takes the form of an experiment. Exactly one of the following cases must hold.

(A) Reduction $a \mapsto a'$ obtained via (Red Experiment), experiment $\mathcal{E}[\vec{a}]$ and reduction $e'[\vec{a}] \mapsto a''$, so that $a' \equiv \mathcal{E}[\vec{a}][a'']$.

(B) Expression e' takes the form of a value, and so a rule other than (Red Experiment) applies.

(C) Expression e' is $-_j$, one of the indexed holes.

We examine each case in turn.

(A) Since $e'[\vec{a}]\ \mathcal{R}_C^{\mathcal{C}}\ e'[\vec{b}]$ the induction hypothesis implies there is b'' such that $e'[\vec{b}] \mapsto b''$ and $a''\ \mathcal{R}_C^{\mathcal{C}}\ b''$. We may derive $b \mapsto \mathcal{E}[\vec{b}][b'']$ and $a' = \mathcal{E}[\vec{a}][a'']\ \mathcal{R}_A^{\mathcal{C}}\ \mathcal{E}[\vec{b}][b'']$ by definition of context closure. (Since a'' and b'' need not be present in $\vec{a}$ and $\vec{b}$ respectively we are here making essential use of multiple-hole contexts.)

(B) In this case a case analysis of all the (Red $-$) rules shows there is a $(\vec{B}, A)$-context e'' such that $\forall \vec{c}{:}\vec{B}(e[\vec{c}] \mapsto e''[\vec{c}])$. (See Lemma 4.16 of Gordon (1994, p45) for the proof of a similar lemma.) In particular $a' \equiv e''[\vec{a}]$ and $b \mapsto e''[\vec{b}]$ with $e''[\vec{a}]\ \mathcal{R}_A^{\mathcal{C}}\ e''[\vec{b}]$.

(C) Here $a \equiv \mathcal{E}[a_j] \mapsto a'$. Given $a_j\ \mathcal{R}_{B_j}\ b_j$, fact (R2) and determinacy there is b' such that $\mathcal{E}[b_j] \mapsto b'$ with $a'\ \mathcal{R}_A^{\mathcal{C}}\ b'$ as required. ■

Proposition 45 *If $\mathcal{R}$ satisfies* (R1) *and* (R2) *then $\mathcal{R} \subseteq \simeq$.*

Proof We shall show that $\mathcal{R} \subseteq \lesssim$; that $\mathcal{R}^{\mathrm{op}} \subseteq \lesssim$, and hence $\mathcal{R} \subseteq \simeq$, follows by a symmetic argument. Suppose then that $a\ \mathcal{R}_A\ b$, $-{:}A \vdash e : \mathtt{Bool}$ and that $e[a]{\Downarrow}$. We must prove that $e[b]{\Downarrow}$. Clearly $e[a]\ \mathcal{R}_{\mathtt{Bool}}^{\mathcal{C}}\ e[b]$ (by turning $-$ into $-_1$). Suppose that $e[a] \mapsto^* \mathtt{true}$. By iterating Lemma 44 there is c such that $e[b] \mapsto^* c$ and $\mathtt{true}\ \mathcal{R}_{\mathtt{Bool}}^{\mathcal{C}}\ c$. It follows by (R1) that c must be a value, so we have $e[b]{\Downarrow}$ as required. (The same argument applies when $e[a] \mapsto^* \mathtt{false}$). ■

Now to validate any closed instance of (Eq Sub Object), let $\mathcal{R} \subseteq \mathit{Rel}$ be the relation such that $a'\ \mathcal{S}_{A'}\ a''$ iff $A' \equiv [\ell_i{:}B_i]_{i \in I}$, and a' and a'' take the forms

$$[\ell_i = \varsigma(x_i{:}A')e_i]_{i \in I} \text{ and } [\ell_i = \varsigma(x_i{:}A'')e_i]_{i \in I \cup J}$$

respectively, where $A'' \equiv [\ell_i{:}B_i, \ell_j{:}B_j]_{i \in I, j \in J}$ with $I \cap J = \varnothing$ and $x_i{:}A' \vdash e_i{:}B_i$ for each $i \in I$ and $x_j{:}A'' \vdash e_j : B_j$ for each $j \in J$.

Proposition 46 *Relation $\mathcal{R}$ possesses properties* (R1) *and* (R2).

Proof (R1) is immediate. For (R2) suppose $a\mathcal{R}_A b$. Then A must be $[\ell_i{:}B_i]_{i\in I}$, and a and b must take the forms in the definition above. There are two classes of experiments to consider.

(1) $\mathcal{E} = -.\ell_i$ where $i \in I$. By (Red Select) $a \mapsto e_i[a/x_i]$ and $b \mapsto e_i[b/x_i]$ and so $a\,\mathcal{R}^{\mathcal{C}}_{B_i}\,b$ (by treating x_i as a hole).

(2) $\mathcal{E} = -.\ell_j \Leftarrow \varsigma(x{:}C)e$ with $C = [\ell_k{:}B_k]_{k\in K}$, $K \subseteq I$, $j \in K$ and $x{:}C \vdash e : B_j$. By (Red Update) we have

$$\begin{array}{rcl} a & \mapsto & [\ell_j = \varsigma(x{:}A')e, \ell_i = \varsigma(x_i{:}A')e_i]_{i\in I-\{j\}} \\ b & \mapsto & [\ell_j = \varsigma(x{:}A'')e, \ell_i = \varsigma(x_i{:}A'')e_i]_{i\in I\cup J-\{j\}} \end{array}$$

and the resulting programs are paired by $\mathcal{R}_{A'} \subseteq \mathcal{R}^{\mathcal{C}}_{A'}$. ■

By Proposition 45 $\mathcal{R} \subseteq \simeq$. By choosing suitable closing substitutions this amounts to a validation of the rule (Eq Sub Object).

A.7 Properties of types

Proof of Proposition 26

(1) $\forall a, b{:}A(a \mapsto b \Rightarrow a \sim_A b)$;

(2) $\forall a, v{:}A(a \Downarrow v \Rightarrow a \sim_A v)$;

(3) $\forall a{:}A(a\Uparrow \Rightarrow a \sim_A \Omega^A)$.

Proof

(1) We show that the relation $\mathcal{S} \stackrel{\text{def}}{=} \{(a_A, b_A) \mid a \mapsto b\} \cup Id$ (where $Id \stackrel{\text{def}}{=} \{(a_A, a_A)\}$) is a bisimulation. Suppose $a_A \xrightarrow{\alpha} a'_B$. Suppose A is active. Then by Lemma 12 it must be the case that $a \Downarrow v$ and $v_A \xrightarrow{\alpha} a'_B$. By Lemma 6 $b \Downarrow v$ too; hence $b_A \xrightarrow{\alpha} a'_B$, and we have $a'_B\,Id\,a'_B$. If A is passive and $-{:}A \vdash \mathcal{E} : B$, then $a_A \xrightarrow{\mathcal{E}\downarrow} \mathcal{E}[a]_B$, and then $b_A \xrightarrow{\mathcal{E}\downarrow} \mathcal{E}[b]_B$ unconditionally. Since $\mathcal{E}[a] \mapsto \mathcal{E}[b]$ by (Red Experiment), then $\mathcal{E}[a]\,\mathcal{S}_B\,\mathcal{E}[b]$. In either case the converse follows by a similar argument, and so $\mathcal{S}$ is a bisimulation.

(2) Immediate from (1) since $\sim$ is a preorder.

(3) We show that the relation $\mathcal{S} \stackrel{\text{def}}{=} \{(a_A, b_A) \mid a{\Uparrow} \;\&\; b{\Uparrow}\}$ is a simulation. Suppose $a\,\mathcal{S}_A\,b$. If A is active, by Lemma 12 it has no transitions. If A is passive, suppose $a_A \xrightarrow{\mathcal{E}\downarrow} \mathcal{E}[a]_B$. Since A is passive, we can derive $b_A \xrightarrow{\mathcal{E}\downarrow} \mathcal{E}[b]_B$ too. By (Red Experiment) $\mathcal{E}[a]{\Uparrow}$ and $\mathcal{E}[b]{\Uparrow}$, so they are related by $\mathcal{S}$. Hence $\mathcal{S}$ is a simulation, and in fact is a bisimulation, as it is symmetric. Hence (3) follows. ■

Proof of Proposition 30 *For any type A, the following are equivalent.*

(1) *A partial*

(2) $\forall a{:}A(a \sim \Omega^A \Rightarrow a{\Uparrow})$

(3) $\forall a, b{:}A(a{\Downarrow} \;\&\; a \sim b \Rightarrow b{\Downarrow})$

Proof

(1) ⇒ (2) For a contradiction suppose that A partial, and there is some a with $a \sim \Omega$ and $a{\Downarrow}$. So there is a value v with $a \Downarrow v$; hence $v \sim \Omega$. Now any convergent program of type A equals a value. And any divergent programs at type A equal Ω, which equals the value v. So A is total. Contradiction.

(2) ⇒ (3) For a contradiction, suppose $a{\Downarrow}$, $a \sim_A b$ and $b{\Uparrow}$. From the latter, $b \sim_A \Omega^A$. Since $a{\Downarrow}$, the contrapositive of (2) implies that $a \not\sim_A \Omega^A$. But by transitivity, $b \not\sim_A \Omega^A$. Contradiction.

(3) ⇒ (1) Suppose some value equals Ω^A. From (3) it would follow that $\Omega^A{\Downarrow}$, but this is impossible. Hence no value equals Ω^A, so A is partial. ■

Lemma 47 *If $SP(E_1) = S$ and $SP(E_2) = S$ then $SP(E_1[^{E_2}/_X]) = S$.*

Proof Structural induction on E_1. ■

Lemma 48 *$\mathcal{S} = \{(a_A, b_A) \mid a, b{:}A \;\&\; SP(A) = S\}$ is a simulation.*

Proof By a case analysis of A. If pair $(a_A, b_A) \in \mathcal{S}$, A is passive, so any transition from a can be matched by its partner b, with their successors also paired in $\mathcal{S}$. The case for $A \equiv \mu(X)E$ needs Lemma 47. ■

Lemma 49 *If $\vec{X} \vdash E$ and $SP(E) = (v, e)$ then*

(1) *$\vec{X} \vdash v : E$ and $\vec{X}, {-}{:}E \vdash e : \texttt{Bool}$;*

(2) $e[v] \Downarrow \texttt{true}$ *and* e *is an evaluation context.*

Proof

(1) By structural induction on E.

(2) By structural induction on E. Clearly e is always an evaluation context. In the cases where $SP(E) = (v, e)$ we can compute the evaluation behaviour of $e[v]$ as follows.

$(\texttt{Bool})$ $-[\texttt{true}] \equiv \texttt{true} \Downarrow \texttt{true}$.

$([\ell_i{:}E_i]_{i\in I})$ $e_j[[\ell_j = v_j, \ldots].\ell_j] \mapsto e_j[v_j]$ and $e_j[v_j] \Downarrow \texttt{true}$ by IH.

$(\mu(X)E)$ $e'[\texttt{unfold}(\texttt{fold}(v'))] \mapsto e'[v']$ and $e'[v'] \Downarrow \texttt{true}$ by IH.

$(E_1 \to E_2)$ $e_2[\lambda(x{:}E_1)v_2(\Omega^{E_1})] \mapsto e_2[v_2]$ and $e_2[v_2] \Downarrow \texttt{true}$ by IH.

$(\texttt{Dynamic})$ $\texttt{typecase}((\texttt{Bool}, \texttt{true}), (x{:}\texttt{Bool})\texttt{true}, \texttt{true}) \Downarrow \texttt{true}$.

$(\texttt{variant}(\ell_i{:}E_i)_{i\in I})$
$\texttt{case}(\texttt{variant}(\ell_1 = \Omega^{E_1}), (x)\texttt{true}, \ldots, (x)\texttt{true}) \Downarrow \texttt{true}$.

$(\texttt{record}(\ell_i{:}E_i)_{i\in I})$
$e_j[\texttt{record}(\ell_j = v_j, \ell_i = \Omega^{E_i})_{i\in I-\{j\}}.\ell_j] \mapsto e_j[v_j] \Downarrow \texttt{true}$ by IH. ■

Proof of Proposition 31

(1) $SP(A) = S \Rightarrow \forall a, b{:}A(a \sim_A b)$

(2) $SP(A) \neq S \Rightarrow \exists v{:}A(v \Downarrow v \;\&\; v \not\sim \Omega^A)$

(3) A *is singular iff* $SP(A) = S$.

Proof Combine Lemmas 48 and 49. (3) is a corollary of (1) and (2). ■

Proof of Proposition 32 *Suppose A is a type. If $SP(A) = S$ then A is total. Otherwise A takes one of the following forms:*

(1) $\mu(X_1)\ldots\mu(X_n)\texttt{Bool}$, *partial;*

(2) $\mu(X_1)\ldots\mu(X_n)[\ell_i{:}E_i]$, *partial.*

(3) $\mu(X_1)\ldots\mu(X_n)E_1 \to E_2$, *total;*

(4) $\mu(X_1)\ldots\mu(X_n)\texttt{Dynamic}$, *partial.*

(5) $\mu(X_1)\ldots\mu(X_n)\texttt{record}(\ell_i{:}E_i)$, *total.*

(6) $\mu(X_1)\ldots\mu(X_n)\texttt{variant}(\ell_i{:}E_i)$, *partial.*

Proof By Lemma 8, any singular type is total.

(1) Any value of type A takes the form $\texttt{fold}(A_1,\ldots\texttt{fold}(A_n,v)\ldots)$ where v is `true` or `false` and each A_i is an unravelling of A. We can always tell Ω^A apart from such a value using the context $\texttt{unfold}(\ldots\texttt{unfold}(-)\ldots)$ consisting of n `unfold`'s.

(2) Any value of type A takes the form $[\ell_i = \varsigma(x_i)e_i]_{i\in I}$, surrounded by n `fold`'s as in part (2). Since $SP(A) \neq S$ there must be a $j \in I$ with $SP(E_j) = (v_j, e_j)$, $e_j[v_j] \Downarrow \texttt{true}$ and e_j an evaluation context. Let e be the context

$$e_j[\texttt{unfold}(\ldots\texttt{unfold}(-)\ldots).\ell_j := v_j.\ell_j]$$

and we have $-{:}A \vdash e : \texttt{Bool}$. For any value $v{:}A$ we have $e[v] \Downarrow \texttt{true}$ but $e[\Omega^A]\Uparrow$. Hence A is partial.

(3) Using the following eta laws we can construct a value equal to any $a{:}\mu(X_1)\ldots\mu(X_n)E_1 \to E_2$.

$$\begin{array}{lcll} a & \sim & \lambda(x{:}A_1)a(x) & \text{if } a{:}A_1 \to A_2 \\ a & \sim & \texttt{fold}(\mu, \texttt{unfold}(a)) & \text{if } a{:}\mu \equiv \mu(X)E \end{array}$$

(4) As in part (1), using the context

$$\texttt{typecase}(\texttt{unfold}(\ldots\texttt{unfold}(-)\ldots),\\ (x{:}\texttt{Bool})\texttt{true}, \texttt{true}).$$

(5) As in part (2), using the following additional eta law if a has the type $\texttt{record}(\ell_i{:}E_i)_{i\in I}$.

$$a \sim \texttt{record}(\ell_i = a.\ell_i)_{i\in I}$$

(6) As in part (3), using the context

$$\texttt{case}(\texttt{unfold}(\ldots\texttt{unfold}(-)\ldots),\\ (x)\texttt{true}, \ldots, (x)\texttt{true}).$$

Thus we have an algorithm for determining whether a type is partial or total. ■

A.8 Other equivalence relations

Proof of Proposition 33 *In the presence of functions or a* `let`*-construct,* $\simeq^\circ = \approx^1$.

Proof

$\simeq^\circ \subseteq \approx^1$ Suppose $\Gamma \vdash e \simeq^\circ_A e'$ and also $\mathcal{C}[e] \Downarrow$. Now $\simeq^\circ$ is equal to $\sim^\circ$ and the open extension of bisimilarity is a congruence, so we have $\mathcal{C}[e] \sim_{\mathtt{Bool}} \mathcal{C}[e']$. So $\mathcal{C}[e'] \Downarrow$ too, and the converse follows by a symmetric argument.

$\approx^1 \subseteq \simeq^\circ$ Suppose $\Gamma \vdash e \approx^1_A e'$. We will show that for any Γ-closure $\vec{\sigma} \equiv a_1/x_1, \ldots, a_n/x_n$, that $e[\vec{\sigma}] \sqsubseteq_{\approx_A} e'[\vec{\sigma}]$. Choose any capturing context $\mathcal{C}$ such that $\mathcal{C}[e[\vec{\sigma}]] \Downarrow$, and let $\mathcal{D} \stackrel{\text{def}}{=} ((\lambda(x_1, \ldots, x_n).-)\, a_1 \ldots a_n)$. Then by repeated application of (Red App) we have $\mathcal{D}[e] \mapsto^* e[\vec{\sigma}]$, so $\mathcal{D}[e] \sqsubseteq_{\approx_A} e[\vec{\sigma}]$; and contextual order is a precongruence, so $\mathcal{C}[e[\vec{\sigma}]] \sqsubseteq_{\approx_{\mathtt{Bool}}} \mathcal{C}[\mathcal{D}[e]]$. By a similar argument we have $\mathcal{C}[e'[\vec{\sigma}]] \sqsubseteq_{\approx_{\mathtt{Bool}}} \mathcal{C}[\mathcal{D}[e']]$. Now $\mathcal{C}[\mathcal{D}[-]]$ is a capturing context, so by the definition of $\approx^1$, if $\mathcal{C}[\mathcal{D}[e]] \Downarrow$, then $\mathcal{C}[\mathcal{D}[e']] \Downarrow$ too. So if $\mathcal{C}[e[\sigma]] \Downarrow$, then $\mathcal{C}[e'[\vec{\sigma}]] \Downarrow$ too, so $e[\vec{\sigma}] \sqsubseteq_{\approx_A} e'[\vec{\sigma}]$. Hence $\Gamma \vdash e \sqsubseteq^\circ_{\approx_A} e'$. A symmetric argument shows $\Gamma \vdash e' \sqsubseteq^\circ_{\approx_A} e$, so we have $\Gamma \vdash e \simeq^\circ_A e'$ as required. ■

A.9 Encoding the λ-calculus in the object calculus

In this section, for the sake of brevity, we write $\delta^m(a)$ to be short for $\delta(\ldots\delta(a, b_m)\ldots, b_1)$, for $m \geq 0$ and arbitrary programs $b_1, \ldots, b_m$.

Lemma 50 *Let* $\langle\!\langle\Gamma\rangle\!\rangle, x{:}\langle\!\langle A\rangle\!\rangle \vdash \langle\!\langle e\rangle\!\rangle < e' : \langle\!\langle B\rangle\!\rangle$ *and* $\varnothing \vdash \langle\!\langle a\rangle\!\rangle < a' : \langle\!\langle A\rangle\!\rangle$ *and* $b \equiv [\mathtt{a} = a', \mathtt{v} = c]$. *Then we have* $\langle\!\langle\Gamma\rangle\!\rangle \vdash \langle\!\langle e[a/x]\rangle\!\rangle < e'[b/x] : \langle\!\langle B\rangle\!\rangle$.

Proof By induction on the derivation of $\Gamma, x{:}A \vdash e : B$.

(Val x) Here $e \equiv x$ (if $e \equiv y \not\equiv x$ the result is obvious). From $\langle\!\langle\Gamma\rangle\!\rangle \vdash x.\mathtt{a} < e' : \langle\!\langle B\rangle\!\rangle$ it follows that $e' \equiv x.\mathtt{a}$ (since $x.\mathtt{a}$ is not a closed program). Then $\Gamma \vdash \langle\!\langle a\rangle\!\rangle < b.\mathtt{a} : \langle\!\langle B\rangle\!\rangle$ follows by the definition of $<$.

(Val App) Here $e \equiv (e_1\, e_2)$. From $\langle\!\langle\Gamma\rangle\!\rangle, x{:}\langle\!\langle A\rangle\!\rangle \vdash \langle\!\langle e_1\rangle\!\rangle.\mathtt{a} := \langle\!\langle e_2\rangle\!\rangle.\mathtt{v} < e' : \langle\!\langle B\rangle\!\rangle$ it follows that e' has the form

$$\delta^l(\delta^m(\delta^n(e'_1).\mathtt{a} := \delta^p(e'_2)).\mathtt{v})$$

with $\langle\!\langle\Gamma\rangle\!\rangle \vdash \langle\!\langle e_1\rangle\!\rangle < e_1' : \langle\!\langle C \to B\rangle\!\rangle$ and $\langle\!\langle\Gamma\rangle\!\rangle \vdash \langle\!\langle e_2\rangle\!\rangle < e_2' : \langle\!\langle C\rangle\!\rangle$ and $l, m, n, p \geq 0$. By the induction hypothesis we have $\langle\!\langle\Gamma\rangle\!\rangle \vdash \langle\!\langle e_1[a/x]\rangle\!\rangle < e_1'[b/x] : \langle\!\langle C \to B\rangle\!\rangle$ and $\langle\!\langle\Gamma\rangle\!\rangle \vdash \langle\!\langle e_2[a/x]\rangle\!\rangle < e_2'[b/x] : \langle\!\langle C\rangle\!\rangle$. So

$$\langle\!\langle\Gamma\rangle\!\rangle \vdash \langle\!\langle e[a/x]\rangle\!\rangle < \delta^l(\delta^m(\delta^n(e_1'[b/x]).\mathtt{a} := \delta^p(e_2'[b/x])).\mathtt{v}) : \langle\!\langle B\rangle\!\rangle$$

as required.

(Val Fun) Here $e \equiv \lambda(y{:}C')e_1$ and $A \equiv C' \to C$. Assume without loss of generality that $x \neq y$. Since $\langle\!\langle\Gamma\rangle\!\rangle, x{:}\langle\!\langle A\rangle\!\rangle \vdash [\mathtt{a} = \varsigma(s)s.\mathtt{a}, \mathtt{v} = \varsigma(y)\langle\!\langle e_1\rangle\!\rangle] < e' : \langle\!\langle B\rangle\!\rangle$ e' has the form

$$\delta^m([\mathtt{a} = \varsigma(s)s.\mathtt{a}, \mathtt{v} = \varsigma(y)e_1'])$$

with $\langle\!\langle\Gamma\rangle\!\rangle, x{:}\langle\!\langle A\rangle\!\rangle \vdash \langle\!\langle e_1\rangle\!\rangle < e_1' : \langle\!\langle C\rangle\!\rangle$ and $m \geq 0$. By the induction hypothesis, $\langle\!\langle\Gamma\rangle\!\rangle \vdash \langle\!\langle e_1[a/x]\rangle\!\rangle < e_1'[b/x] : \langle\!\langle C\rangle\!\rangle$ so

$$\begin{aligned}\langle\!\langle\Gamma\rangle\!\rangle \vdash [\mathtt{a} = \varsigma(s)s.\mathtt{a}, \mathtt{v} = \varsigma(y)\langle\!\langle e_1[a/x]\rangle\!\rangle] < \\ \delta^m([\mathtt{a} = \varsigma(s)s.\mathtt{a}, \mathtt{v} = \varsigma(y)e_1'[b/x]]) : \langle\!\langle B\rangle\!\rangle\end{aligned}$$

as required.

The other cases (Val True), (Val False) and (Val If) are similar. ■

Lemma 51 *If a is a λ-calculus program and $a{:}A$, then if $a \mapsto a'$ and $\varnothing \vdash \langle\!\langle a\rangle\!\rangle < b : \langle\!\langle A\rangle\!\rangle$ then there exists b' such that $b \mapsto^+ b'$ and $\varnothing \vdash \langle\!\langle a'\rangle\!\rangle < b' : \langle\!\langle A\rangle\!\rangle$.*

Proof By induction on the derivation of $a \mapsto a'$.

(Red Beta) Here $a \equiv ((\lambda(x{:}B)e)\, a_2)$ and $a' \equiv e[a_2/x]$. Since

$$\varnothing \vdash [\mathtt{a} = \varsigma(s)s.\mathtt{a}, \mathtt{v} = \varsigma(x{:}\langle\!\langle B \to A\rangle\!\rangle)\langle\!\langle e\rangle\!\rangle].\mathtt{a} := \langle\!\langle a_2\rangle\!\rangle.\mathtt{v} < b : \langle\!\langle A\rangle\!\rangle,$$

then it follows (using the fact that $\delta^n(a) \mapsto^n a$ for any a and n) that

$$b \mapsto^* [\mathtt{a} = b_2, \mathtt{v} = \varsigma(x{:}\langle\!\langle B \to A\rangle\!\rangle)b_1].\mathtt{v}$$

where $\varnothing \vdash \langle\!\langle a_2\rangle\!\rangle < b_2 : \langle\!\langle B\rangle\!\rangle$ and $x{:}\langle\!\langle B \to A\rangle\!\rangle \vdash \langle\!\langle e\rangle\!\rangle < b_1 : \langle\!\langle A\rangle\!\rangle$. So $b \mapsto^+ b'$ where $b' \equiv b_1[[\mathtt{a} = b_2, \mathtt{v} = \varsigma(x{:}\langle\!\langle B \to A\rangle\!\rangle)b_1]/x]$ and $\varnothing \vdash \langle\!\langle e[a_2/x]\rangle\!\rangle < b' : \langle\!\langle A\rangle\!\rangle$ by Lemma 50.

(Red Experiment) Suppose $\mathcal{E} \equiv (-\, a_2)$ (the case when $\mathcal{E} \equiv \mathtt{if}(-, a_2, a_3)$ is similar). Then $a \equiv (a_1\, a_2)$ and $a_1 \mapsto a_1'$ and $a' \equiv (a_1'\, a_2)$. From $\varnothing \vdash \langle\!\langle a_1\rangle\!\rangle.\mathtt{a} := \langle\!\langle a_2\rangle\!\rangle.\mathtt{v} < b : \langle\!\langle A\rangle\!\rangle$ it follows that $b \mapsto^* b_1.\mathtt{a} := b_2.\mathtt{v}$ with $\varnothing \vdash \langle\!\langle a_1\rangle\!\rangle < b_1 : \langle\!\langle B \to A\rangle\!\rangle$ and $\varnothing \vdash \langle\!\langle a_2\rangle\!\rangle < b_2 : \langle\!\langle B\rangle\!\rangle$. By the induction hypothesis there exists b_1' such that $b_1 \mapsto^+ b_1'$ and $\varnothing \vdash \langle\!\langle a_1'\rangle\!\rangle < b_1' : \langle\!\langle B \to A\rangle\!\rangle$. So $b \mapsto^+ b'$ where $b' \equiv b_1'.\mathtt{a} := b_2.\mathtt{v}$ and $\varnothing \vdash \langle\!\langle (a_1'\, a_2)\rangle\!\rangle < b' : \langle\!\langle A\rangle\!\rangle$.

The remaining cases, for (Red If True) and (Red If False), are similar to the above. ■

Proof of Lemma 34 *Let a be a λ-calculus program with $a{:}A$. Then*

(1) $a \Downarrow v \Rightarrow \exists u(\langle\langle a\rangle\rangle \Downarrow u \;\&\; \langle\langle v\rangle\rangle <_{\langle\langle A\rangle\rangle} u)$

(2) $\langle\langle a\rangle\rangle \Downarrow u \Rightarrow \exists v(a \Downarrow v \;\&\; \langle\langle v\rangle\rangle <_{\langle\langle A\rangle\rangle} u)$

Proof For part (1), let $a \ll b$ if there is some type A such that $\varnothing \vdash \langle\langle a\rangle\rangle < b : \langle\langle A\rangle\rangle$. Then by Lemma 51, the following diagram commutes.

$$\begin{array}{ccccccccc} a & \mapsto & a_1 & \mapsto & a_2 & \mapsto & \ldots & \mapsto & v \\ \Mapsto & & \Mapsto & & \Mapsto & & & & \Mapsto \\ \langle\langle a\rangle\rangle & \mapsto^+ & b_1 & \mapsto^+ & b_2 & \mapsto^+ & \ldots & \mapsto^+ & b_n \end{array}$$

Then from $v \ll b_n$ it follows that there exists a value u such that $b_n \mapsto^* u$ and $v \ll u$.

For part (2), it suffices to show that $\langle\langle a\rangle\rangle\Downarrow$ implies $a\Downarrow$. For if so, we have $\langle\langle a\rangle\rangle \Downarrow u$ and $a \Downarrow v$, and $\langle\langle v\rangle\rangle <_{\langle\langle A\rangle\rangle} u$ follows from part (1) and the determinacy of reduction in the object calculus.

We prove the contrapositive, that $a\Uparrow$ implies $\langle\langle a\rangle\rangle\Uparrow$. Suppose $a\Uparrow$. By Lemma 51, we can fill in the following diagram.

$$\begin{array}{ccccccc} a & \mapsto & a_1 & \mapsto & a_2 & \mapsto & \ldots \\ \Mapsto & & \Mapsto & & \Mapsto & & \\ \langle\langle a\rangle\rangle & \mapsto^+ & b_1 & \mapsto^+ & b_2 & \mapsto^+ & \ldots \end{array}$$

Hence $\langle\langle a\rangle\rangle\Uparrow$ as required. ■

B The equational theory of $\mathbf{FOb}_{\mathbf{1}<:\mu}$

These equational rules are taken from Abadi and Cardelli (1996), and comprise rules of symmetry, transitivity, congruence and reduction, plus (Eq Top), (Eq Sub Object), (Eval Fold) and (Eval Eta).

$$\frac{\Gamma \vdash e \leftrightarrow e' : A}{\Gamma \vdash e' \leftrightarrow e : A} \text{ (Eq Symm)}$$

$$\frac{\Gamma \vdash e \leftrightarrow e' : A \quad \Gamma \vdash e' \leftrightarrow e'' : A}{\Gamma \vdash e \leftrightarrow e'' : A} \text{ (Eq Trans)}$$

$$\frac{\Gamma, x{:}A, \Gamma' \vdash \diamond}{\Gamma, x{:}A, \Gamma' \vdash x \leftrightarrow x : A} \text{ (Eq } x\text{)}$$

$$\frac{\Gamma \vdash e \leftrightarrow e' : A \quad \Gamma \vdash A <: B}{\Gamma \vdash e \leftrightarrow e' : B} \text{ (Eq Subsumption)}$$

$$\frac{\Gamma \vdash e : A \quad \Gamma \vdash e' : B}{\Gamma \vdash e \leftrightarrow e' : \mathtt{Top}} \text{ (Eq Top)}$$

$$\frac{A \equiv [\ell_i{:}A_i]_{i\in I} \quad \Gamma, x_i{:}A \vdash e_i \leftrightarrow e_i' : A_i \ (i \in I)}{\Gamma \vdash [\ell_i = \varsigma(x{:}A)e_i]_{i\in I} \leftrightarrow [\ell_i = \varsigma(x{:}A)e_i']_{i\in I} : A} \text{ (Eq Object)}$$

$$\frac{\Gamma \vdash e \leftrightarrow e' : [\ell_i{:}A_i]_{i\in i} \quad j \in I}{\Gamma \vdash e.\ell_j \leftrightarrow e'.\ell_j : A_j} \text{ (Eq Select)}$$

$$\frac{\Gamma \vdash e_1 \leftrightarrow e_1' : [\ell_i{:}A_i]_{i\in i} \quad \Gamma, x{:}A \vdash e_2 \leftrightarrow e_2' : A_j \quad j \in I}{\Gamma \vdash e_1.\ell_j \Leftarrow \varsigma(x{:}A)e_2 \leftrightarrow e_1'.\ell_j \Leftarrow \varsigma(x{:}A)e_2' : A_j} \text{ (Eq Update)}$$

$$\frac{\begin{array}{c} A' \equiv [\ell_i{:}B_i]_{i\in I} \quad A'' \equiv [\ell_i{:}B_i, \ell_j{:}B_j]_{i\in I, j\in J} \quad I \cap J = \varnothing \\ \Gamma, x_i{:}A' \vdash e_i : B_i \ (i \in I) \quad \Gamma, x_j{:}A'' \vdash e_j : B_j \ (j \in J) \end{array}}{\Gamma \vdash [\ell_i = \varsigma(x_i{:}A')e_i]_{i\in I} \leftrightarrow [\ell_i = \varsigma(x_i{:}A'')e_i]_{i\in I\cup J} : A'} \text{ (Eq Sub Object)}$$

$$\frac{A \equiv [\ell_i{:}A_i]_{i\in I} \quad e \equiv [\ell_i = \varsigma(x_i{:}A')e_i]_{i\in J} \quad \Gamma \vdash e : A \quad j \in I \subseteq J}{\Gamma \vdash e.\ell_j \leftrightarrow e_j[e/x_j] : A_j} \text{ (Eval Select)}$$

$$\frac{\begin{array}{c} A \equiv [\ell_i{:}A_i]_{i\in I} \quad e \equiv [\ell_i = \varsigma(x_i{:}A')e_i]_{i\in J} \\ \Gamma \vdash e : A \quad \Gamma, x{:}A \vdash e' : A_j \quad j \in I \subseteq J \end{array}}{\Gamma \vdash e.\ell_j \Leftarrow \varsigma(x{:}A)e' \leftrightarrow [\ell_i = \varsigma(x_i{:}A')e_i, \ell_j = \varsigma(x{:}A')e']_{i\in J-\{j\}} : A} \begin{pmatrix}\text{Eval}\\ \text{Update}\end{pmatrix}$$

$$\frac{A \equiv \mu(X)E \quad \Gamma \vdash e \leftrightarrow e' : E[{}^{A}/_{X}]}{\Gamma \vdash \texttt{fold}(A, e) \leftrightarrow \texttt{fold}(A, e') : A} \text{ (Eq Fold)}$$

$$\frac{A \equiv \mu(X)E \quad \Gamma \vdash e \leftrightarrow e' : A}{\Gamma \vdash \texttt{unfold}(e) \leftrightarrow \texttt{unfold}(e') : E[{}^{A}/_{X}]} \text{ (Eq Unfold)}$$

$$\frac{A \equiv \mu(X)E \quad \Gamma \vdash e : A}{\Gamma \vdash \texttt{fold}(A, \texttt{unfold}(e)) \leftrightarrow e : A} \text{ (Eval Fold)}$$

$$\frac{A \equiv \mu(X)E \quad \Gamma \vdash e : E[{}^{A}/_{X}]}{\Gamma \vdash \texttt{unfold}(\texttt{fold}(A, e)) \leftrightarrow e) : E[{}^{A}/_{X}]} \text{ (Eval Unfold)}$$

$$\frac{\Gamma, x{:}A \vdash e \leftrightarrow e' : B}{\Gamma \vdash \lambda(x{:}A)e \leftrightarrow \lambda(x{:}A)e' : A \to B} \text{ (Eq Fun)}$$

$$\frac{\Gamma \vdash e_1 \leftrightarrow e_1' : A \to B \quad \Gamma \vdash e_2 \leftrightarrow e_2' : A}{\Gamma \vdash (e_1\, e_2) \leftrightarrow (e_1'\, e_2') : B} \text{ (Eq Appl)}$$

$$\frac{\Gamma \vdash \lambda(x{:}A)e : A \to B \quad \Gamma \vdash e' : A}{\Gamma \vdash ((\lambda(x{:}A)e)\, e') \leftrightarrow e[{}^{e'}/_{x}] : B} \text{ (Eval Beta)}$$

$$\frac{\Gamma \vdash e : A \to B \quad x \notin \mathit{Dom}(\Gamma)}{\Gamma \vdash \lambda(x{:}A)(e\, x) \leftrightarrow e : A \to B} \text{ (Eval Eta)}$$

C Notation

- Sets of expressions

Type	set of closed types
$\mathit{Exp}(\Gamma, A)$	set of expressions of type A in environment Γ
$\mathit{Prog}(A)$	set of programs of type A
$\mathit{Value}(A)$	set of values of type A

- Meta-variables in expressions

E	a possibly open type
e	a possibly open expression
Γ	an environment
X	a type variable
x	an expression variable
A, B, C	closed types
a, b, c	programs (closed expressions)
u, v	values (programs which do not reduce)
ℓ	label identifying a field of an object, record or variant
I, J	finite indexing sets (with which labels are indexed)
i, j	indexing variables ranging over the contents of I, J
$\mathcal{E}$	an experiment (atomic evaluation context)
$-$	distinguished free variable in experiments and contexts

- Proved programs and relations between them

Rel	the universal relation on proved programs of equal type
Id	the identity relation on proved programs
P, Q	proved programs (pairs of programs and their types)
$\mathcal{R}, \mathcal{S}$	relations on programs (subsets of Rel)
a_A	the proved program a at type A
$\mathcal{R}_A$	the subset of $\mathcal{R}$ relating programs of type A
$\mathcal{R}^\circ$	the open extension of $\mathcal{R}$
$\widehat{\mathcal{R}}$	the compatible refinement of $\mathcal{R}$

- Operational semantics

$a \mapsto b$	program a reduces to b in one step of computation
$a \mapsto$	program a can take (at least) one step of computation
$a \Downarrow b$	program a evaluates to the value b
$a\Downarrow$	the evaluation of a converges
$a\Uparrow$	the evaluation of a diverges
$\precsim$	contextual order
$\simeq$	contextual equivalence
$\overset{A}{\simeq}$	contextual equivalence, termination observable at type A

- Labelled transition system

α	an action in the labelled transition system
$\mathbf{0}$	the proved program with no actions
$\downarrow$	the erasure operation from experiments to actions
$P \xrightarrow{\alpha} Q$	P does the action α, becoming Q

- Similarity and bisimilarity

$[-]$	the similarity functional
$\langle - \rangle$	the bisimilarity functional
$\lesssim$	the similarity relation
$\sim$	the bisimilarity relation
$\lesssim^{\bullet}$	the candidate relation for precongruence

- Equivalence relations

$\equiv$	equality up to alpha-conversion
$\leftrightarrow$	metavariable for an equivalence relation on expressions
$\approx^1$	contextual equivalence using capturing contexts
$\approx^2$	record-style bisimilarity
$\approx^3$	applicative bisimulation

- Miscellaneous

$\vec{\sigma}$	a substitution of closed terms for expression variables