Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 155 (2006) 445–465
www.elsevier.com/locate/entcs

Normal Form Simulation for McCarthy's Amb

Soren B. Lassen[1]

Google, Inc.
Mountain View, CA

Abstract

This paper presents co-inductive operational theories of program refinement and equivalence, called whnf similarity and whnf simulation equivalence, for the λ-calculus extended with McCarthy's ambiguous choice operator *amb*. The associated whnf simulation co-induction proof principle is useful for establishing non-trivial equivalences and refinement relationships between programs. Whnf similarity is a pre-congruence and whnf simulation equivalence is a congruence and a conservative extension of the Lévy-Longo tree theory for the pure λ-calculus.

Keywords: ambiguous choice, λ-calculus, co-induction

McCarthy's ambiguity operator *amb* [13] is a "locally angelic" or "fair" non-deterministic choice operator. It embodies a form of fair parallel evaluation, which is a powerful programming idiom. Functional programming with *amb* is very expressive and subsumes several other well-known non-deterministic and parallel functional language extensions, including erratic choice, countable choice (random assignment), and parallel or. Some uses of *amb* in functional and parallel programming are described in [5,14,3].

This paper presents new operationally-based methods for reasoning about program refinement and equivalence between higher-order functional programs with *amb*. We introduce co-inductive theories for program refinement and equivalence, called weak head normal form (whnf) similarity and simulation equivalence. They extend the bisimulation characterization of Lévy-Longo tree equivalence from [21,8] to a call-by-name λ-calculus with *amb*. We show that whnf similarity is a pre-congruence and that whnf simulation equivalence

[1] Email: soren@google.com

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2005.11.068

is a congruence and a conservative extension of the Lévy-Longo tree theory for the λ-calculus. Moreover, we establish a useful whnf simulation up-to-context proof rule.

Whnf simulation equivalence is included in contextual equivalence. The inclusion is strict, because whnf simulation equivalence distinguishes some non-deterministic branching behaviors that are not distinguished by contextual equivalence. Even so, whnf simulation equivalence is an improvement over the existing operational theories for *amb*, called Kleene equivalence and cost equivalence [7,14,11], because it makes neither the unnecessary syntactic distinctions of Kleene equivalence nor the unnecessary operational distinctions of cost equivalence.

Outline

Section 1 discusses related work. Section 2 specifies syntax and operational semantics and defines contextual refinement and equivalence relations. Section 3 extends the operational semantics to open terms, needed for the definition of whnf similarity in section 4. In section 5 we prove that whnf similarity is a pre-congruence. Section 6 establishes a whnf simulation up-to-context proof rule. Section 7 discusses other (bi)simulation preorders and equivalences.

1 Background and related work

Fair non-determinism has been studied mostly in the context of dataflow networks, beginning with Park's seminal paper [20]. Panangaden, Stark, and Shanbhogue [19,17,18] established the relative expressiveness of McCarthy's *amb* and a number of fair merge primitives for dataflows.

The semantics of *amb* is impossible to capture using conventional semantic models, because *amb* is not monotone, let alone continuous, with regard to domain-theoretic orderings.[2] Traditionally, one gives meaning to non-deterministic constructs via powerdomains, which are domain-theoretic analogues of the powerset operator; see [22] for a survey. The denotational approach encounters well-known problems when attempting to model McCarthy's *amb*. For example, the Egli-Milner ordering is a natural preorder to consider (it combines domain-theoretic analogues of may and must behaviors),

[2] After the presentation of this paper at the MFPS XXI conference, Lassen, Levy, and Panangaden [10] formalized the non-monotonicity and discontinuity of *amb* relative to domain-theoretic orderings by proving that no least fixpoint semantics for *amb* satisfies the desired equational laws and respects may divergence.

but *amb* is not even monotonic with respect to this ordering, so the theory breaks down.

On the other hand, there are good structural operational semantics for higher-order functional languages with *amb* [14,2]. Moran's big-step structural operational semantics was used as the basis for the operationally-defined equational theories of Kleene equivalence and cost equivalence, reported in [7,14,11]. These theories are useful both for establishing basic equational laws and for bisimulation proofs of program equivalences between recursive functions and infinite data structures. Nonetheless, they are too discriminative—more discriminative than contextual equivalence—because they distinguish terms unnecessarily based on syntax and steps in the operational semantics.

There are good non-deterministic variations of Abramsky's applicative bisimulation theory [1] for erratic non-determinism [16,4,12,7]. Two of these, convex bisimulation and refinement simulation, are good candidates for a better operational theory for *amb* but it is an open problem whether they are (pre)congruences; see [11] for an explanation why we have failed to apply Howe's congruence proof method [4] to *amb*.

This paper sidesteps the open problems concerning applicative simulation for *amb* by using a different kind of simulation which we call normal form simulation. In our terminology, the bisimulation characterizations of Lévy-Longo and Böhm tree equivalences in [21,8] are instances of normal form simulation. Another example is the eager normal form simulation for the call-by-value λ-calculus in [9]. The normal form simulation co-induction proof rules are sometimes more useful than applicative simulation because they do not involve any universal quantification over function arguments. Moreover, in general, normal form simulation supports convenient simulation up-to-context proof rules [8], whereas the validity of general applicative simulation up-to-context is an open problem [6].

We show that our normal form simulation equivalence, whnf simulation equivalence, is included in applicative simulation equivalence but leave it as an open problem whether the inclusion is strict.

Carayol, Hirschkoff, and Sangiorgi [2] encode the λ-calculus extended with *amb* (the same language studied in the present paper) into the π-calculus and prove program equivalences between λ-terms with *amb* by showing that their encodings are bisimilar in the π-calculus. The induced equivalence on pure λ-terms (without *amb*) is Lévy-Longo tree equivalence. (This follows from the same characterization of the induced equivalence from a π-calculus encoding of the pure λ-calculus in [21].) The results in [2] are not directly comparable to ours. They consider an alternative definition of contextual equivalence between λ-terms with *amb*, based on an unconventional notion of "strong"

divergence, because it is impossible to encode *amb* compositionally into the π-calculus in a fashion that is faithful with respect to regular divergence. The authors argue that strong divergence is the appropriate behavioral observation to use in the definition of contextual equivalence. We prefer the conventional definition of contextual equivalence (see remark 2.4 below).

2 Syntax and operational semantics

We consider the pure untyped λ-calculus extended with *amb*:

$$\begin{array}{lllllllll} \text{Variables} & x, y, z & & & & & & & \\ \text{Terms} & t & ::= & x & \mid & \lambda x.\, t & \mid & t_1\, t_2 & \mid & amb(t_1, t_2) \end{array}$$

We identify terms up to renaming of bound variables and write $t_1[t_2/x]$ for the capture-free substitution of t_2 for the free occurrences of x in t_1.

The term $amb(t_1, t_2)$ non-deterministically chooses between t_1 and t_2. Informally, it does so by evaluating t_1 and t_2 in fair parallel and choosing the outcome of the branch that first evaluates to a result. It may only diverge if both threads of execution in the fair parallel evaluation may diverge.

We use the following standard combinators in the sequel: the combinators $I = \lambda x.\, x$ and $K = \lambda x.\, \lambda y.\, x$, the diverging term $\Omega = (\lambda x.\, x\, x)\, (\lambda x.\, x\, x)$, the Church numeral $0 = \lambda x.\, \lambda y.\, y$, successor $succ = \lambda x.\, \lambda y.\, \lambda z.\, y\, (x\, y\, z)$, and Turing's fixed point combinator $Fix = (\lambda z.\, \lambda x.\, x\, (z\, z\, x))\, (\lambda z.\, \lambda x.\, x\, (z\, z\, x))$.

Erratic choice can be defined in terms of *amb* as follows:

$$err(t_1, t_2) = amb(K\, t_1, K\, t_2)\, I.$$

The countable choice between all Church numerals can be defined as follows:

$$any = (Fix\, \lambda x.\, \lambda y.\, amb(y, x\, (succ\, y)))\, 0.$$

Fairness guarantees that *any* terminates. (With *err* in place of *amb*, evaluation may diverge by always choosing the right branch.)

We equip the language with a structural operational semantics in the form of an inductively defined big-step evaluation relation $t \rightsquigarrow w$ that relates closed terms t to closed λ-abstractions w.

$$\frac{}{\lambda x.\, t \rightsquigarrow \lambda x.\, t} \qquad \frac{t_1 \rightsquigarrow \lambda x.\, t \quad t[t_2/x] \rightsquigarrow w}{t_1\, t_2 \rightsquigarrow w} \qquad \frac{t_1 \rightsquigarrow w}{amb(t_1, t_2) \rightsquigarrow w} \qquad \frac{t_2 \rightsquigarrow w}{amb(t_1, t_2) \rightsquigarrow w}$$

A closed term t *may converge*, written $t\Downarrow$, iff $\exists w.\, t \rightsquigarrow w$.

If we expand the definition of $err(t_1, t_2)$, we see that the derived evaluation rules are identical to those for *amb*. To distinguish *amb* from erratic choice

we need to consider divergence behavior of terms or, dually, their must convergence behavior. We define a *must convergence* predicate $\downarrow$ on closed terms inductively by the rules:

$$\frac{}{\lambda x.\, t\downarrow} \qquad \frac{t_1\downarrow \quad \forall x, t.\ t_1 \rightsquigarrow \lambda x.\, t \Rightarrow t[{}^{t_2}\!/_{x}]\downarrow}{t_1\, t_2\downarrow} \qquad \frac{t_1\downarrow}{amb(t_1,t_2)\downarrow} \qquad \frac{t_2\downarrow}{amb(t_1,t_2)\downarrow}$$

We say that a closed term t *may diverge*, written $t\uparrow$, iff $\neg t\downarrow$.

The rules for *amb* are "fair" because they specify that $amb(t_1, t_2)$ must converge if one branch must converge, even if the other diverges. Operationally, this can be accomplished by evaluating the two branches in parallel with a fair scheduler. For instance, observe that $amb(I, \Omega)\downarrow$ and $any\downarrow$. By expanding the definition of $err(t_1, t_2)$ we derive the rule:

$$\frac{t_1\downarrow \quad t_2\downarrow}{err(t_1,t_2)\downarrow}$$

which highlights the difference between *err* and *amb*.

Proposition 2.1 *For any closed term t, $t\downarrow$ implies $t\Downarrow$.*

Proof. By induction on the derivation of $t\downarrow$. □

We now define a program refinement and a program equivalence relation that take both evaluation and divergence behavior into account. We need the following notation: A term context C is a term with a hole, $[\,]$, and $C[t]$ is the term obtained by filling t into the hole in C, possibly capturing free variables in t if the hole in C occurs under λ-abstractions.

Definition 2.2 t is *contextually refined* by t' iff, for all term contexts C such that $C[t]$ and $C[t']$ are closed terms, $C[t']\Downarrow \Rightarrow C[t]\Downarrow$ and $C[t']\uparrow \Rightarrow C[t]\uparrow$. Let *contextual equivalence* be the induced equivalence relation (mutual contextual refinement).

Since $\downarrow$ is the complement of $\uparrow$, we see that t is contextually equivalent to t' iff, for all term contexts C such that $C[t]$ and $C[t']$ are closed terms, $C[t]\Downarrow \Leftrightarrow C[t']\Downarrow$ and $C[t]\downarrow \Leftrightarrow C[t']\downarrow$.

Informally, contextual refinement orders terms according to their degree of determinism—*err* is a greatest lower bound operator and a term t is contextually refined by a term t' iff t is contextually equivalent to $err(t, t')$.

It is easy to show that contextual refinement is a pre-congruence and that contextual equivalence is a congruence, in fact, the largest congruence relation that discriminates terms with different may or must convergence behavior.

Remark 2.3 The test for may convergence in definition 2.2, $C[t']\Downarrow \Rightarrow C[t]\Downarrow$, is redundant for the call-by-value λ-calculus with *amb* [7, proposition 8.2.1]. This is not true for our call-by-name language. For example, Ω is not contextually refined by $err(\Omega, K\,\Omega)$, because only the latter may converge, but they are related by the "must" contextual preorder (and equivalence) without the test for may convergence. To see this, let

$$R = \{(\Omega, err(\Omega, K\,\Omega))\}^{\mathsf{SC}}$$

(see definition 5.1), then show that $t\ R\ t'$ and $t' \rightsquigarrow \lambda x.\, t'_0$ imply either $t'_0 = \Omega$ or $t \rightsquigarrow \lambda x.\, t_0$ for some t_0 such that $t_0\ R\ t'_0$, by induction on $t' \rightsquigarrow \lambda x.\, t'_0$. Use this to show that $t\ R\ t'$ and $t{\downarrow}$ imply $t'{\downarrow}$, by induction on $t{\downarrow}$, and from this conclude that Ω is below $err(\Omega, K\,\Omega)$ in the must contextual preorder.

However, if we extended our language with a "convergence tester" or "strict let" as in [11], then the test for may convergence in definition 2.2 does become redundant (by the same argument as in [7]).

Remark 2.4 In [2], an alternative "strong" divergence predicate is defined. A term t is *strongly divergent*, written $t{\uparrow}$, if it may evolve into a term t' such that $\neg t'\Downarrow$ (t "evolves into" t' means that t reduces to t' by weak head reduction and by parallel reduction of subterms of *amb*). In our framework of big-step operational semantics, strong divergence can be specified inductively as the smallest predicate closed under the rules:

$$\frac{t_1{\uparrow}}{t_1\,t_2{\uparrow}} \qquad \frac{t[t_2/\!\!/x]{\uparrow}}{t_1\,t_2{\uparrow}} \text{ if } t_1 \rightsquigarrow \lambda x.\, t \qquad \frac{t_1{\uparrow} \quad t_2{\uparrow}}{amb(t_1,t_2){\uparrow}} \qquad \frac{}{t{\uparrow}} \text{ if } \neg t\Downarrow$$

Carayol, Hirschkoff, and Sangiorgi argue that it is appropriate to distinguish strongly divergent terms from other ("weakly") divergent terms. However, the difference between strong and weak divergence reflects a difference in nondeterministic branching behavior that (conventional) contextual equivalence abstracts away from. In our view, the definition of contextual equivalence in terms of may and must convergence (compared to Carayol *et al*'s definition in terms of strong divergence) is well-motivated by conventional semantical notions: May and must convergence are partial and total correctness assertions, respectively, so contextual equivalence is crisply defined as the largest congruence that only equates terms that satisfy the same partial and total correctness assertions.

The next example shows that contextual equivalence is not a conservative extension of contextual equivalence for the pure (lazy) λ-calculus, a.k.a. the maximal semi-lazy λ-theory [15].

Example 2.5 $t = \lambda x.\, x\, \lambda y.\, x\, y$ and $t' = \lambda x.\, x\, x$ are contextually equivalent in the pure λ-calculus. This can be proved using the operational extensionality property of the lazy λ-calculus [1,15] and the fact that weak head reduction is deterministic. But t and t' are not contextually equivalent in the presence of *amb*: Let $C = amb(I, [\,]\, err(I, \Omega))\, I$, then $C[t]$ may diverge and $C[t']$ must converge.

Because of the quantification over all term contexts, it is difficult to prove that a term is contextually refined by another term or that two terms are contextually equivalent directly from the definitions of contextual refinement and equivalence. (See example 4.6 for an example of a direct proof.)

Our goal is to find useful proof methods for establishing refinement and equivalence relationships between terms. To this end we will introduce a novel simulation proof principle for reasoning about *amb*, namely normal form simulation. More specifically, since the operational semantics for our language evaluates terms to weak head normal form (whnf), we call the simulation preorder *whnf similarity* and the induced equivalence relation *whnf simulation equivalence* (or, in the nomenclature of [12], *mutual whnf similarity*).

3 Open operational semantics

In this section, we extend the operational semantics—the evaluation relation and the must convergence predicate—to operate on open terms. This is needed to define normal form simulation for *amb* in the next section, because normal form simulation is based on symbolic evaluation of both closed and open terms to normal form.

The weak head normal forms are the terms w generated by the grammar:

$$\begin{array}{lrcl} \text{Whnfs} & w & ::= & f \mid \lambda x.\, t \\ \lambda\text{-free hnfs} & f & ::= & x \mid f\, t \end{array}$$

(A "λ-free" hnf is a head normal form that is not a λ-abstraction.)

We extend the evaluation relation, $\rightsquigarrow$, to open terms by adding rules for *amb* to the deterministic weak head evaluation relation for open λ-terms in [8]. The extended evaluation relation relates (open and closed) terms to whnfs. It

is defined inductively as the smallest relation closed under the rules:

$$\text{(Eval-Var)}\ \frac{}{x \rightsquigarrow x} \qquad \text{(Eval-Lam)}\ \frac{}{\lambda x.\, t \rightsquigarrow \lambda x.\, t}$$

$$\text{(Eval-App.}\lambda\text{)}\ \frac{t_1 \rightsquigarrow \lambda x.\, t \quad t[t_2/x] \rightsquigarrow w}{t_1\, t_2 \rightsquigarrow w} \qquad \text{(Eval-App.f)}\ \frac{t_1 \rightsquigarrow f}{t_1\, t_2 \rightsquigarrow f\, t_2}$$

$$\text{(Eval-Amb.1)}\ \frac{t_1 \rightsquigarrow w}{amb(t_1, t_2) \rightsquigarrow w} \qquad \text{(Eval-Amb.2)}\ \frac{t_2 \rightsquigarrow w}{amb(t_1, t_2) \rightsquigarrow w}$$

A key property of "open evaluation" is the following substitution property.

Proposition 3.1 (Substitution) *$t_1[t_2/x] \rightsquigarrow w$ iff there exists a whnf w_1 such that $t_1 \rightsquigarrow w_1$ and $w_1[t_2/x] \rightsquigarrow w$.*

We omit the proof.

Next, we need to extend the definition of must convergence, $\downarrow$, to open terms. It is not obvious how to do this—in fact, more than anything else, the solution to this problem is the key to defining a congruent normal form simulation equivalence. As a guiding principle, must convergence on open terms should satisfy a substitution property analogous to proposition 3.1 for evaluation. But how? For instance, is $x\downarrow$ true or not? It should be true if we substitute I for x. It should be false if we substitute Ω for x. A more complex example is the term $amb(I, x)\, I$. It must converge if we substitute I or Ω for x, but not if we substitute $K\, \Omega$ for x.

The solution is to turn the predicate $t\downarrow$ on closed terms t into a relation $t\downarrow F$ between (open and closed) terms t and sets F of λ-free hnfs, where $t\downarrow F$ should be understood as a conditional assertion about the must convergence behavior of t: Suppose $t\downarrow F$ and $t\sigma$ is closed, for some substitution σ, then $t\sigma\downarrow$ if

- $\sigma(x)\downarrow$ for all $x \in F \cap dom(\sigma)$, and
- for all $f\, t' \in F$ and closed λ-abstractions w, $f\sigma \rightsquigarrow w$ implies $w\, (t'\sigma)\downarrow$;

and if t is closed then $t\downarrow\emptyset$ iff $t\downarrow$.

For instance, $x\downarrow\{x\}$. It means (trivially) that, for every closed term t, $t\downarrow$ implies $x[t/x]\downarrow$.

Less trivially, $amb(I, x)\, I\downarrow\{x\, I\}$. It means that, for every closed term t, $amb(I, t)\, I\downarrow$ if, for all w, $t \rightsquigarrow w$ implies $w\, I\downarrow$.

The $\downarrow$ relation is defined inductively by the following rules:

$$\text{(Conv-Var)}\ \frac{}{x{\downarrow}F}\ \text{if } x \in F \qquad \text{(Conv-Lam)}\ \frac{}{\lambda x.\, t{\downarrow}F}$$

$$\text{(Conv-App)}\ \frac{t_1{\downarrow}F \quad \forall t \in \{t[{}^{t_2}\!/\!_x] \mid t_1 \rightsquigarrow \lambda x.\, t\}.\ t{\downarrow}F}{t_1\, t_2{\downarrow}F}\ \text{if } \{f\, t_2 \mid t_1 \rightsquigarrow f\} \subseteq F$$

$$\text{(Conv-Amb.1)}\ \frac{t_1{\downarrow}F}{amb(t_1,t_2){\downarrow}F} \qquad \text{(Conv-Amb.2)}\ \frac{t_2{\downarrow}F}{amb(t_1,t_2){\downarrow}F}$$

Proposition 3.2 *$t{\downarrow}F$ implies $\exists w.\, t \rightsquigarrow w$.*

Proof. By induction on the derivation of $t{\downarrow}F$. □

Proposition 3.3 (Weakening) *$t{\downarrow}F$ and $F \subseteq F'$ imply $t{\downarrow}F'$.*

Proof. By induction on the derivation of $t{\downarrow}F$. □

Proposition 3.4 (Substitution) *$t_1[{}^{t_2}\!/\!_x]{\downarrow}F$ iff there exists F_1 such that $t_1{\downarrow}F_1$ and*

(i) *for all variables $y \in F_1$, $y[{}^{t_2}\!/\!_x]{\downarrow}F$, and*
(ii) *for all $t \in B(F_1, t_2, x)$, $t{\downarrow}F$, and*
(iii) *$A(F_1, t_2, x) \subseteq F$,*

where

$$A(F_1,t_2,x) = \{f'\,(t[{}^{t_2}\!/\!_x]) \mid \exists f.\ f\,t \in F_1\ \&\ f[{}^{t_2}\!/\!_x] \rightsquigarrow f'\},$$
$$B(F_1,t_2,x) = \{t_0[{}^{t[{}^{t_2}\!/\!_x]}\!/\!_y] \mid \exists f.\ f\,t \in F_1\ \&\ f[{}^{t_2}\!/\!_x] \rightsquigarrow \lambda y.\, t_0\}.$$

(In the first clause $y[{}^{t_2}\!/\!_x]{\downarrow}F$ means $t_2{\downarrow}F$, if $x = y$, otherwise $y \in F$.)

Proof. The left-to-right implication follows from propositions 5.8 and 5.11 below. The reverse implication follows by induction on the derivation of $t_1{\downarrow}F_1$. □

4 Whnf similarity and whnf simulation equivalence

Definition 4.1 A relation R between terms is a *whnf simulation* if $t\ R\ t'$ implies

(i) whenever $t' \rightsquigarrow \lambda x.\, t'_0$ there exists $\lambda x.\, t_0$ such that $t \rightsquigarrow \lambda x.\, t_0$ and $t_0\ R\ t'_0$, and
(ii) whenever $t' \rightsquigarrow f' = x\, t'_1 \ldots t'_n$ there exists $f = x\, t_1 \ldots t_n$ such that $t \rightsquigarrow f$ and $t_i\ R\ t'_i$ for all $i \in \{1, \ldots, n\}$, and

(iii) whenever $t{\downarrow}F$ there exists F' such that $t'{\downarrow}F'$ and, for every $f' = x\,t'_1 \ldots t'_n \in F'$, there exists $f = x\,t_1 \ldots t_n \in F$ such that $t_i\ R\ t'_i$ for all $i \in \{1, \ldots, n\}$.

A whnf simulation R is a *whnf bisimulation* if its reciprocal relation $R^{\mathrm{op}} = \{(t', t) \mid (t, t') \in R\}$ is also a whnf simulation.

Let *whnf similarity* be the largest whnf simulation. This is well-defined, because whnf simulations are closed under unions. Whnf similarity is a preorder: It is reflexive, because the identity term relation Id is a whnf (bi)simulation, and it is transitive, because the composition of two whnf simulations is a whnf simulation. Let *whnf simulation equivalence*, $\eqsim_{\mathrm{wh}}$, be the induced equivalence relation (mutual whnf similarity).

Proposition 4.2 *Pure λ-terms t and t' (without occurrences of amb) are Lévy-Longo tree equivalent iff $t \eqsim_{\mathrm{wh}} t'$. That is, $\eqsim_{\mathrm{wh}}$ is a conservative extension of Lévy-Longo tree equivalence for the pure λ-calculus.*

Proof. We use the "deterministic" whnf (bi)simulation in [21] (called open applicative bisimulation) and in [8]. To disambiguate, let us refer to any whnf simulation for our extended language (definition 4.1) as a "non-deterministic" whnf simulation. We also use the following two facts, which can be shown by induction on derivations:

- The restriction of the non-deterministic evaluation relation $t \rightsquigarrow w$, for our extended language, to pure λ-terms t is identical to the usual deterministic big-step evaluation relation between pure λ-terms and whnfs.
- For any pure λ-term t, $t{\downarrow}F$ iff there exists a whnf w such that $t \rightsquigarrow w$ and, if w is a λ-free hnf, $\{x, x\,t_1, x\,t_1\,t_2, \ldots, x\,t_1 \ldots t_n \mid w = x\,t_1 \ldots t_n\} \subseteq F$.

Using these facts, we see that any deterministic whnf simulation is also a non-deterministic whnf simulation and, moreover, the restriction of any non-deterministic whnf simulation to pure λ-terms is a deterministic whnf simulation. By co-induction we conclude that Lévy-Longo tree equivalence coincides with our non-deterministic whnf simulation equivalence on pure λ-terms. □

Example 4.3 Whnf simulation equivalence includes fundamental laws such as β equivalence and *amb* is associative, commutative, idempotent, and Ω is its unit:

$$
\begin{gathered}
(\lambda x.\, t_1)\, t_2 \eqsim_{\mathrm{wh}} t_1[t_2/x], \\
amb(t_1, amb(t_2, t_3)) \eqsim_{\mathrm{wh}} amb(amb(t_1, t_2), t_3), \\
amb(t_1, t_2) \eqsim_{\mathrm{wh}} amb(t_2, t_1), \\
amb(t, t) \eqsim_{\mathrm{wh}} t, \\
amb(t, \Omega) \eqsim_{\mathrm{wh}} amb(\Omega, t) \eqsim_{\mathrm{wh}} t.
\end{gathered}
$$

In all cases it is straightforward to check that the left hand side t and the right hand side t' evaluate to the same whnfs w and must converge to the same sets F of λ-free hnfs; therefore $\{(t, t')\} \cup Id$ is a whnf bisimulation (recall that Id is itself a whnf bisimulation) and, hence, $t \eqsim_{\text{wh}} t'$.

Example 4.4 Erratic choice is a greatest lower bound operator with respect to whnf similarity, because

$$\{(err(t_1, t_2), t_1), (err(t_1, t_2), t_2), (t, err(t_1, t_2)) \mid t \text{ is whnf similar to both } t_1 \text{ and } t_2\} \cup Id$$

is a whnf simulation.

Example 4.5 Consider the following encoding of lists:

$$nil = \lambda x_1 x_2.\, x_1, \qquad cons(t_1, t_2) = \lambda x_1 x_2.\, x_2\, t_1\, t_2,$$

where $\lambda x_1 x_2.\, t$ is shorthand for $\lambda x_1.\, \lambda x_2.\, t$.

The function

$$append = Fix\ \lambda y.\, \lambda z_1 z_2.\, z_1\ z_2\ \lambda x_1 x_2.\, cons(x_1, y\, x_2\, z_2)$$

takes two streams z_1 and z_2 and returns z_1 with z_2 appended to the end, if z_1 is finite.

If z_1 is infinite, one can show that $append\, z_1\, z_2 \eqsim_{\text{wh}} z_1$ by exhibiting a suitable whnf bisimulation. For instance, let $repeat\, x$ be the infinite list of x's defined as $repeat\, x = Fix\ \lambda y.\, cons(x, y)$, then

$$repeat\, x \eqsim_{\text{wh}} append\, (repeat\, x)\, \Omega$$

because

$$R_1 = \{(repeat\, x, append\, (repeat\, x)\, \Omega), (\lambda x_2.\, x_2\, x\, (repeat\, x), \lambda x_2.\, x_2\, x\, (append\, (repeat\, x)\, \Omega))\} \cup Id$$

is a whnf bisimulation.

A non-deterministic "bottom-avoiding" merge function on lists (cf. [11]) can be encoded as follows:

$$merge = Fix\ \lambda y.\, \lambda z_1 z_2.\, amb(z_1\ z_2\ \lambda x_1 x_2.\, cons(x_1, y\, x_2\, z_2), z_2\ z_1\ \lambda x_1 x_2.\, cons(x_1, y\, z_1\, x_2))$$

It is bottom-avoiding in the sense that

$$merge\, x\, \Omega \eqsim_{\mathrm{wh}} append\, x\, \Omega$$

which we can prove by verifying that

$$\begin{aligned} R_2 = \{&(merge\, x\, \Omega,\ append\, x\, \Omega),\\ &(\lambda x_1 x.\ cons(x_1, merge\, x\, \Omega), \lambda x_1 x.\ cons(x_1, append\, x\, \Omega)),\\ &(\lambda x.\ cons(x_1, merge\, x\, \Omega), \lambda x.\ cons(x_1, append\, x\, \Omega)),\\ &(cons(x_1, merge\, x\, \Omega), cons(x_1, append\, x\, \Omega)),\\ &(\lambda x_2.\ x_2\, x_1\, (merge\, x\, \Omega), \lambda x_2.\ x_2\, x_1\, (append\, x\, \Omega))\} \cup Id \end{aligned}$$

is a whnf bisimulation.

In the next section we prove that whnf similarity is a pre-congruence and therefore that whnf simulation equivalence is a congruence. An immediate corollary is that whnf similarity is included in contextual refinement and whnf simulation equivalence is included in contextual equivalence. The next example shows that the inclusions are strict.

Example 4.6 Let $t = amb(K\, I, K\, \Omega)$ and $t' = K\, err(I, \Omega)$. Then t is contextually equivalent to t' but t and t' are not whnf similar.

It is straightforward to show that t' is whnf similar to t. Hence t' is contextually refined by t. But t is not whnf similar to t', because $t' \rightsquigarrow \lambda x.\ err(I, \Omega)$ and t does not evaluate to any λ-abstraction $\lambda x.\, t_0$ with t_0 whnf similar to $err(I, \Omega)$.

To prove that t is contextually refined by t' we need to show that $C[t']\Downarrow \Rightarrow C[t]\Downarrow$ and $C[t]\downarrow \Rightarrow C[t']\downarrow$, for all closed term contexts C.

First, we prove

$$\forall x, t_0, w'.\ t_0[t'/x] \rightsquigarrow w' \Rightarrow P(x, t_0, w'), \tag{1}$$

where P is the predicate

$$\begin{aligned} P(x, t_0, w') \ \text{ iff } \ &(\exists y.\ w' = \lambda y.\ err(I, \Omega)\ \&\ t_0[t/x] \rightsquigarrow \lambda y.\ I\ \&\ t_0[t/x] \rightsquigarrow \lambda y.\ \Omega)\\ &\vee\ (\exists y, t_0'.\ w' = \lambda y.\ t_0'[t'/x]\ \&\ t_0[t/x] \rightsquigarrow \lambda y.\ t_0'[t/x]), \end{aligned}$$

by induction on the derivation of $t_0[t'/x] \rightsquigarrow w'$.

Case $t_0[t'/x] \rightsquigarrow w'$ is derived from $t_1' \rightsquigarrow \lambda z.\, t_3'$ and $t_3'[t_2'/z] \rightsquigarrow w'$, where $t_1'\, t_2' = t_0[t'/x]$. Either $t_0 = t_{01}\, t_{02}$, with $t_1' = t_{01}[t'/x]$ and $t_2' = t_{02}[t'/x]$, or $t_0 = x$ and $t_1' = \lambda z.\, t_3' = K$ and $t_2' = err(I, \Omega)$.

If $t_0 = t_{01}\, t_{02}$, by the induction hypothesis $P(x, t_{01}, \lambda z.\, t_3')$ holds, that is, either $t_3' = err(I, \Omega)$ and $t_{01}[t/x] \rightsquigarrow \lambda z.\, I$, in which case $w' = I$ and we deduce that $t_0[t/x] \rightsquigarrow I$, too, or $t_3' = t_{03}[t'/x]$ and $t_{01}[t/x] \rightsquigarrow \lambda z.\, t_{03}[t/x]$, in which case $t_3'[t_2'/z] = t_{04}[t'/x]$, where $t_{04} = t_{03}[t_{02}/z]$, and by the induction hypothesis $P(x, t_{04}, w')$ holds from which we deduce $P(x, t_0, w')$.

If $t_0 = x$, $P(x, t_0, w')$ holds because $w' = \lambda z.\, err(I, \Omega)$ and $t_0[t/x] \rightsquigarrow \lambda z.\, I$ and $t_0[t/x] \rightsquigarrow \lambda z.\, \Omega$, because $t_0[t/x] = t = amb(K\, I, K\, \Omega)$ and $K\, I \rightsquigarrow \lambda z.\, I$ and $K\, \Omega \rightsquigarrow \lambda z.\, \Omega$.

The other cases are simpler.

Next, we use (1) to prove

$$\forall x, t_0.\ t_0[t/x]{\downarrow} \Rightarrow t_0[t'/x]{\downarrow}, \tag{2}$$

by induction on the derivation of $t_0[t/x]{\downarrow}$.

Case $t_0[t/x] = t_1\, t_2$ and $t_0[t/x]{\downarrow}$ is derived from $t_1{\downarrow}$ and

$$\forall y, t_3.\ t_1 \rightsquigarrow \lambda y.\, t_3 \Rightarrow t_3[t_2/y]{\downarrow}. \tag{3}$$

In this case t_0 must be of the form $t_0 = t_{01}\, t_{02}$ with $t_1 = t_{01}[t/x]$ and $t_2 = t_{02}[t/x]$. Then $t_0[t'/x] = t_1'\, t_2'$ with $t_1' = t_{01}[t'/x]$ and $t_2' = t_{02}[t'/x]$. We must show $t_1'\, t_2'{\downarrow}$. By the induction hypothesis, $t_1'{\downarrow}$. It remains to be shown that

$$\forall y, t_3'.\ t_1' \rightsquigarrow \lambda y.\, t_3' \Rightarrow t_3'[t_2'/y]{\downarrow}. \tag{4}$$

From (1) we deduce that, whenever $t_1' \rightsquigarrow \lambda y.\, t_3'$, then $P(x, t_{01}, \lambda y.\, t_3')$. Expanding the definition of $P(x, t_{01}, \lambda y.\, t_3')$, we observe that it cannot be the case that $t_{01}[t/x] = t_1 \rightsquigarrow \lambda y.\, \Omega$ because of (3). Therefore t_3' must be of the form $t_3' = t_{03}[t'/x]$ and $t_1 \rightsquigarrow \lambda y.\, t_{03}[t/x]$. By (3), $(t_{03}[t/x])[t_2/y]{\downarrow}$. Since $(t_{03}[t/x])[t_2/y] = (t_{03}[t_{02}/y])[t/x]$, the induction hypothesis implies that $(t_{03}[t_{02}/y])[t'/x]{\downarrow}$. Since $(t_{03}[t_{02}/y])[t'/x] = t_3'[t_2'/y]$, we conclude (4), as required.

Case $t_0[t/x] = amb(t_1, t_2)$ and $t_0[t/x]{\downarrow}$ is derived from $t_1{\downarrow}$.

Either $t_0 = amb(t_{01}, t_{02})$, with $t_1 = t_{01}[t/x]$ and $t_2 = t_{02}[t/x]$, or $t_0 = x$ and $t_1 = K\, I$ and $t_2 = K\, \Omega$. In the first case, when $t_0 = amb(t_{01}, t_{02})$, we get $t_{01}[t'/x]{\downarrow}$, by the induction hypothesis, and derive $t_0[t'/x]{\downarrow}$. In the second case, when $t_0 = x$, we see that $t_0[t'/x]{\downarrow}$ because $t_0[t'/x] = K\, err(I, \Omega)$ and $K{\downarrow}$ and $K \rightsquigarrow \lambda z.\, \lambda y.\, z$ and $(\lambda y.\, z)[^{err(I,K)}/_z] = \lambda y.\, err(I, K){\downarrow}$.

The other cases are similar or simpler.

Finally, given any closed term context C, note that $C[t'] = C[x][t'/x]$ and $C[t] = C[x][t/x]$ for any fresh variable x. Therefore (1) implies $C[t']\Downarrow \Rightarrow C[t]\Downarrow$ and (2) implies $C[t]\downarrow \Rightarrow C[t']\downarrow$. Hence t is contextually refined by t'.

5 Pre-congruence

In this section we prove that whnf similarity is a pre-congruence. We already know that whnf similarity is a preorder. What remains to be shown is that it is also compatible, i.e., closed under contexts. For this purpose we introduce the substitutive context closure relational operator [6].

Definition 5.1 If R is a relation between terms, its *substitutive context closure*, R^{SC}, is the smallest relation that is compatible and substitutive and contains R. It is defined inductively by the rules:

$$\text{(SC-Var)}\ \frac{}{x\ R^{\mathsf{SC}}\ x} \qquad \text{(SC-Lam)}\ \frac{t\ R^{\mathsf{SC}}\ t'}{\lambda x.\, t\ R^{\mathsf{SC}}\ \lambda x.\, t'}$$

$$\text{(SC-App)}\ \frac{t_1\ R^{\mathsf{SC}}\ t_1' \quad t_2\ R^{\mathsf{SC}}\ t_2'}{t_1\, t_2\ R^{\mathsf{SC}}\ t_1'\, t_2'} \qquad \text{(SC-Amb)}\ \frac{t_1\ R^{\mathsf{SC}}\ t_1' \quad t_2\ R^{\mathsf{SC}}\ t_2'}{amb(t_1, t_2)\ R^{\mathsf{SC}}\ amb(t_1', t_2')}$$

$$\text{(SC-Subst)}\ \frac{t_1\ R^{\mathsf{SC}}\ t_1' \quad t_2\ R^{\mathsf{SC}}\ t_2'}{t_1[t_2/x]\ R^{\mathsf{SC}}\ t_1'[t_2'/x]} \qquad \text{(SC-Rel)}\ \frac{}{t\ R^{\mathsf{SC}}\ t'}\ \text{if } t\ R\ t'$$

Since the definition is symmetrical, R^{SC} is symmetric if R is.

Lemma 5.2 (Main Lemma) *If R is a whnf simulation then R^{SC} is a whnf simulation.*

Before we prove the main lemma, let us see how to use it to derive that whnf similarity is a pre-congruence.

Theorem 5.3 *Whnf similarity is a pre-congruence and whnf simulation equivalence is a congruence.*

Proof. Since whnf similarity is itself a whnf simulation, the main lemma says that its substitutive context closure is also a whnf simulation. By co-induction, we conclude that whnf similarity includes its substitutive context closure. This means that whnf similarity is compatible and substitutive, which is what we needed to conclude that whnf similarity is a pre-congruence and, hence, whnf simulation equivalence is a congruence. □

Corollary 5.4 *Whnf similarity is included in contextual refinement and whnf simulation equivalence is included in contextual equivalence.*

The proof of the main lemma comes in two parts. The first part deals with clauses (1) and (2) in the definition of whnf simulation. The second part deals with clause (3).

For the first part, define a "lower" whnf simulation to be a relation R such that $t\ R\ t'$ implies clauses (1) and (2) from the definition of whnf simulation. (For convenience, we invert the conventional definition in [12].)

Definition 5.5 R is a *lower whnf simulation* if $t\ R\ t'$ implies

(i) whenever $t' \rightsquigarrow \lambda x.\, t'_0$ there exists $\lambda x.\, t_0$ such that $t \rightsquigarrow \lambda x.\, t_0$ and $t_0\ R\ t'_0$, and

(ii) whenever $t' \rightsquigarrow f' = x\, t'_1 \ldots t'_n$ there exists $f = x\, t_1 \ldots t_n$ such that $t \rightsquigarrow f$ and $t_i\ R\ t'_i$ for all $i \in \{1, \ldots, n\}$.

Lemma 5.6 *If R is a lower whnf simulation then R^{SC} is a lower whnf simulation.*

Proof. (Outline) A simple extension of the proof of the main lemma for the deterministic lambda calculus in [8] to non-deterministic choice. The proof involves the definition of a variant of the evaluation relation, $t \overset{n}{\rightsquigarrow} w$, instrumented by the number n of β reductions in the evaluation of a term t to a whnf w. Proposition 3.1 extends to the instrumented evaluation relation and plays a key role in the proof. □

The definition of lower whnf simulation is more relaxed than the definition of whnf simulation, so any whnf simulation is also a lower whnf simulation. Therefore lemma 5.6 establishes the first part of the proof of the main lemma 5.2.

Remark 5.7 Based on the definition of lower whnf simulation, we can define a lower whnf similarity preorder. Lemma 5.6 implies that it is a pre-congruence. However, since the definition is based only on (may) evaluation, not divergence or must convergence, lower whnf similarity does not distinguish *amb* from *err*. In other words, it doesn't adequately capture the fair behaviour of *amb*.

For the purposes of the second part of the proof of the main lemma 5.2 we need to instrument the must convergence relation by the depth of the derivation tree, measured as the number of beta reductions. Since a must convergence derivation tree is not always finitely branching, its depth may not be finite. Therefore we use ordinals to measure the derivation depth. The instrumented must convergence relation $t{\downarrow}^a F$ relates terms t, sets F of λ-free

hnfs, and countable ordinals a. It it defined inductively by the rules:

$$\text{(Conv-Var)}\ \frac{}{x{\downarrow^a}F}\ \text{if } x \in F \qquad \text{(Conv-Lam)}\ \frac{}{\lambda x.\, t{\downarrow^a}F}$$

$$\text{(Conv-App)}\ \frac{t_1{\downarrow^a}F \quad \forall t \in \{t[{}^{t_2}\!/_x] \mid t_1 \rightsquigarrow \lambda x.\, t\}.\ t{\downarrow^{<a}}F}{t_1\, t_2{\downarrow^a}F}\ \text{if } \{f\, t_2 \mid t_1 \rightsquigarrow f\} \subseteq F$$

$$\text{(Conv-Amb.1)}\ \frac{t_1{\downarrow^a}F}{amb(t_1,t_2){\downarrow^a}F} \qquad \text{(Conv-Amb.2)}\ \frac{t_2{\downarrow^a}F}{amb(t_1,t_2){\downarrow^a}F}$$

where $t{\downarrow^{<a}}F$ in (Conv-App) means $\exists a' < a.\ t{\downarrow^{a'}}F$.

Proposition 5.8 *$t{\downarrow}F$ iff there exists a countable ordinal a such that $t{\downarrow^a}F$.*

Proof. The right to left implication is immediate because the rules for the binary must convergence relation $t{\downarrow}F$ are just the erasure of the instrumented rules for the ternary must convergence relation $t{\downarrow^a}F$.

For the left to right implication, in two passes we attach ordinal labels to the nodes of the derivation tree for $t{\downarrow}F$.

In the first pass, from leaves to root, label the nodes by ordinals in the following fashion. Label all (Conv-Var) and (Conv-Lam) nodes (the leaf nodes) by 0. Label all (Conv-Amb.1) and (Conv-Amb.2) nodes by the label of the root of their immediate sub-tree. Finally, label any (Conv-App) node with conclusion $t_1\, t_2{\downarrow}F'$ by the ordinal $\max\{a_{t_1}, \max\{a_{t_0} + 1 \mid t_0 \in S\}\}$ where $S = \{t[{}^{t_2}\!/_x] \mid t_1 \rightsquigarrow \lambda x.\, t\}$ and a_{t_1} is the label of the root of the first immediate sub-tree of the derivation of $t_1\, t_2{\downarrow}F'$ and $\{a_{t_0} \mid t_0 \in S\}$ are the labels of the roots of the other immediate sub-trees. Since the set S used in the labelling of each (Conv-App) node is countable, all the labels are countable ordinals, including the label a of the root of the derivation tree for $t{\downarrow}F$.

In the second pass, from root to leaves, increase the value of the labels as follows. Change the ordinal label of the root of the first immediate sub-tree of every (Conv-App) node to the (Conv-App) node's own label. Change the ordinal label of the root of the immediate sub-tree of every (Conv-Amb.1) node to the (Conv-Amb.1) node's own label. Similarly for (Conv-Amb.2). The resulting labelling of the derivation tree for $t{\downarrow}F$ constitutes a valid instrumentation in accordance with the rules above and the label a of the root is a countable ordinal. □

Remark 5.9 It is possible to refine the proof to show a stronger result, namely that $t{\downarrow}F$ iff there exists a *recursive* ordinal a such that $t{\downarrow^a}F$.

Proposition 5.10 (Weakening) *$t{\downarrow^a}F$ and $F \subseteq F'$ and $a \leq a'$ imply $t{\downarrow^{a'}}F'$.*

Proof. By induction on the derivation of $t{\downarrow}^a F$. □

Proposition 5.11 (Substitution) *$t_1[t_2/x]{\downarrow}^a F$ implies there exists F_1 such that $t_1{\downarrow}^a F_1$ and*

(i) *for all variables $y \in F_1$, $y[t_2/x]{\downarrow}^a F$, and*

(ii) *for all $t \in B(F_1, t_2, x)$, $t{\downarrow}^{<a} F$, and*

(iii) *$A(F_1, t_2, x) \subseteq F$,*

where $A(F_1, t_2, x)$ and $B(F_1, t_2, x)$ are defined as in proposition 3.4.

Proof. Fix t_2 and x and define the following predicate P on pairs of sets:

$$\begin{aligned} P(F_1, F) \text{ iff } &\text{(i) for all variables } y \in F_1,\ y[t_2/x]{\downarrow}^a F \\ &\text{(ii) for all } t \in B(F_1, t_2, x),\ t{\downarrow}^{<a} F, \text{ and} \\ &\text{(iii) } A(F_1, t_2, x) \subseteq F. \end{aligned}$$

Observe that $P(F_1, F)$ and $P(F_2, F)$ imply $P(F_1 \cup F_2, F)$.

The desired result follows if, for all t', F, and a,

$$t'{\downarrow}^a F \Rightarrow \forall t_1.\ t' = t_1[t_2/x] \Rightarrow \exists F_1.\ t_1{\downarrow}^a F_1 \ \&\ P(F_1, F),$$

which can be shown by induction on the derivation of $t'{\downarrow}^a F$. □

Proof. (Main Lemma 5.2) Assume R is a whnf simulation. To prove that R^{SC} is a whnf simulation we must show for all terms t and t', if $t\ R^{\mathsf{SC}}\ t'$ then

(i) whenever $t' \rightsquigarrow \lambda x.\, t'_0$ there exists $\lambda x.\, t_0$ such that $t \rightsquigarrow \lambda x.\, t_0$ and $t_0\ R^{\mathsf{SC}}\ t'_0$,

(ii) whenever $t' \rightsquigarrow f' = x\, t'_1 \ldots t'_n$ there exists $f = x\, t_1 \ldots t_n$ such that $t \rightsquigarrow f$ and $t_i\ R^{\mathsf{SC}}\ t'_i$ for all $i \in \{1, \ldots, n\}$, and

(iii) whenever $t{\downarrow}F$ there exists F' such that $t'{\downarrow}F'$ and, for every $f' = x\, t'_1 \ldots t'_n \in F'$, there exists $f = x\, t_1 \ldots t_n \in F$ such that $t_i\ R^{\mathsf{SC}}\ t'_i$ for all $i \in \{1, \ldots, n\}$.

Since R is a whnf simulation it is also a lower whnf simulation, so, by lemma 5.6, R^{SC} is a lower whnf simulation. Therefore the first and second clause hold.

For the third clause, the desired result follows, by proposition 5.8, if we can show

$$\begin{aligned} \forall \text{countable ordinals } a.\ \forall t, t'.\ \forall F.\ t\ R^{\mathsf{SC}}\ t' \ \&\ t{\downarrow}^a F \Rightarrow \\ \exists F'.\ t'{\downarrow}F' \ \&\ \forall f' = x\, t'_1 \ldots t'_n \in F' \Rightarrow \\ \exists f = x\, t_1 \ldots t_n \in F.\ t_i\ R^{\mathsf{SC}}\ t'_i \text{ for all } i \in \{1, \ldots, n\}. \end{aligned}$$

The proof is by induction on a (transfinite induction) and on the derivation of $t\ R^{\mathsf{SC}}\ t'$, ordered lexicographically. The argument is, in essence, analogous to the proof of lemma 5.6. □

6 Whnf simulation up to context

The main lemma 5.2 is analogous to the pre-congruence proof for whnf similarity for deterministic λ-calculus in [8]. We now apply the reasoning from [8] to derive from the proof of the main lemma a whnf simulation up-to-context proof rule for *amb*.

Definition 6.1 R is a *whnf simulation up-to-context* if $t\ R\ t'$ implies

(i) whenever $t' \rightsquigarrow \lambda x.\, t'_0$ there exists $\lambda x.\, t_0$ such that $t \rightsquigarrow \lambda x.\, t_0$ and $t_0\ R^{\mathsf{SC}}\ t'_0$,

(ii) whenever $t' \rightsquigarrow f' = x\, t'_1 \ldots t'_n$ there exists $f = x\, t_1 \ldots t_n$ such that $t \rightsquigarrow f$ and $t_i\ R^{\mathsf{SC}}\ t'_i$ for all $i \in \{1, \ldots, n\}$, and

(iii) whenever $t{\downarrow}F$ there exists F' such that $t'{\downarrow}F'$ and, for every $f' = x\, t'_1 \ldots t'_n \in F'$, there exists $f = x\, t_1 \ldots t_n \in F$ such that $t_i\ R^{\mathsf{SC}}\ t'_i$ for all $i \in \{1, \ldots, n\}$.

Note that definition 6.1 relaxes the three clauses in definition 4.1 by replacing R by R^{SC}.

A whnf simulation up-to-context R is a *whnf bisimulation up-to-context* if its reciprocal relation R^{op} is also a whnf simulation up-to-context.

Theorem 6.2 *Every whnf simulation up-to-context R is included in whnf similarity.*

Proof. If we revisit the proofs of lemma 5.6 and the main lemma 5.2, we find that they only use the weaker pre-condition that R is a whnf simulation up-to-context. The conclusion of the main lemma says that R^{SC} is a whnf simulation. Therefore, by co-induction, R^{SC} is included in whnf similarity and, by (SC-Rel), R is included in whnf similarity. □

By symmetry, every whnf bisimulation up-to-context is included in $\eqsim_{\mathrm{wh}}$.

Example 6.3 Recall from example 4.5 the whnf bisimulations R_1 and R_2 used to prove $\mathit{repeat}\, x \eqsim_{\mathrm{wh}} \mathit{append}\, (\mathit{repeat}\, x)\, \Omega$ and $\mathit{merge}\, x\, \Omega \eqsim_{\mathrm{wh}} \mathit{append}\, x\, \Omega$. We can prove the same equations much simpler by checking that

$$R'_1 = \{(\mathit{repeat}\, x, \mathit{append}\, (\mathit{repeat}\, x)\, \Omega)\},$$
$$R'_2 = \{(\mathit{merge}\, x\, \Omega, \mathit{append}\, x\, \Omega)\},$$

are whnf bisimulations up-to-context.

7 Closing remarks

7.1 Whnf bisimilarity

Whnf bisimulations are closed under union. Therefore there exists a greatest whnf bisimulation, which we can call *whnf bisimilarity*. It is also the greatest symmetric whnf simulation. It follows from the main lemma 5.2 that whnf bisimilarity is a congruence.

Since every whnf bisimulation is a whnf simulation, whnf bisimilarity is included in whnf simulation equivalence. The inclusion is strict, because whnf bisimilarity distinguishes more non-deterministic branching behaviors, as demonstrated by the following example.

Example 7.1 Let $t = K\, amb(I, K\,\Omega)$ and $t' = amb(t, K\, I)$. Then $t \simeq_{\mathrm{wh}} t'$ but t and t' are not whnf bisimilar.

7.2 Applicative similarity and bisimilarity

In [7,11] applicative similarity and bisimilarity are defined for call-by-value and call-by-name functional languages equipped with *amb*. For our language, we can define applicative similarity and bisimilarity as follows.

Definition 7.2 Let a relation R between terms be an *applicative simulation* if $t\ R\ t'$ implies, for all substitutions σ such that $t\sigma$ and $t'\sigma$ are closed,

(i) whenever $t'\sigma \rightsquigarrow \lambda x.\, t'_0$ there exists $\lambda x.\, t_0$ such that $t\sigma \rightsquigarrow \lambda x.\, t_0$ and $t_0\ R\ t'_0$, and

(ii) $t\sigma{\downarrow}$ implies $t'\sigma{\downarrow}$.

An applicative simulation R is an *applicative bisimulation* if its reciprocal relation R^{op} is also an applicative simulation. Let *applicative similarity* be the largest applicative simulation and *applicative bisimilarity* be the largest applicative bisimulation.

It is an open problem whether applicative similarity is a pre-congruence and whether applicative bisimilarity is a congruence, for our language as well as for the languages in [7,11].

Since whnf similarity is a pre-congruence, we can show that, if t and t' are whnf similar, so are $t\sigma$ and $t'\sigma$, for any substitution σ. From this it follows that whnf similarity is an applicative simulation. Hence whnf similarity is included in applicative similarity. Analogously, we see that whnf bisimilarity is included in applicative bisimilarity. We do not know if the inclusions are strict. (Obviously, if the inclusions are not strict, we have answered affirmatively the open question whether applicative similarity is a pre-congruence

and applicative bisimilarity is a congruence.)

7.3 Enf similarity and hnf similarity

It should be possible to adapt the whnf simulation theory from this paper to develop an analogous "eager" normal form (enf) simulation theory [9] for the call-by-value λ-calculus extended with *amb*.

Similarly, we can ask if it is possible to modify our "lazy" whnf simulation theory to obtain a different, "sensible" equational theory (that equates Ω and $K\,\Omega$), by adapting the head normal form (hnf) bisimulation characterization of Böhm tree equivalence from [8]. A key challenge here is how to define an operational semantics for *amb* based on evaluation to head normal form.

Acknowledgement

I thank the referees for explanations of related work and for helpful suggestions to improve the presentation.

References

[1] Abramsky, S., *The lazy lambda calculus*, in: D. Turner, editor, *Research Topics in Functional Programming*, Addison-Wesley, 1990 pp. 65–116.

[2] Carayol, A., D. Hirschkoff and D. Sangiorgi, *On the representations of McCarthy's* amb *in the π-calculus*, in: *EXPRESS'03*, Electronic Notes in Theoretical Computer Science **96**, 2004, pp. 73–89.

[3] Du Bois, A. R., R. F. Pointon, H.-W. Loidl and P. W. Trinder, *Implementing declarative parallel bottom-avoiding choice*, in: *14th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'02)* (2002), pp. 82–92.

[4] Howe, D. J., *Proving congruence of bisimulation in functional programming languages*, Information and Computation **124** (1996), pp. 103–112.

[5] Hughes, J. and J. O'Donnell, *Expressing and reasoning about non-deterministic functional programs*, in: *Proceedings of the Glasgow Functional Programming Workshop '89*, Workshops in Computing (1990), pp. 308–328.

[6] Lassen, S. B., *Relational reasoning about contexts*, in: A. D. Gordon and A. M. Pitts, editors, *Higher Order Operational Techniques in Semantics*, Publications of the Newton Institute, Cambridge University Press, 1998 pp. 91–135.

[7] Lassen, S. B., "Relational Reasoning about Functions and Nondeterminism," Ph.D. thesis, Department of Computer Science, University of Aarhus (1998), BRICS Dissertation Series DS-98-2.

[8] Lassen, S. B., *Bisimulation in untyped lambda calculus: Böhm trees and bisimulation up to context*, in: *MFPS XV*, Electronic Notes in Theoretical Computer Science **20** (1999), pp. 346–374.

[9] Lassen, S. B., *Eager normal form bisimulation*, in: *Proc. 20th Annual IEEE Symposium on Logic in Computer Science*, 2005, pp. 345–354.

[10] Lassen, S. B., P. B. Levy and P. Panangaden, *Divergence-least semantics of amb is Hoare* (2005), short presentation at the APPSEM II workshop, September 2005, Frauenchiemsee, Germany.

[11] Lassen, S. B. and A. K. Moran, *Unique fixed point induction for McCarthy's amb*, in: *Proceedings of the 24th International Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science **1672** (1999), pp. 198–208.

[12] Lassen, S. B. and C. S. Pitcher, *Similarity and bisimilarity for countable non-determinism and higher-order functions (extended abstract)*, in: *Second Workshop on Higher-Order Operational Techniques in Semantics (HOOTS II)*, Electronic Notes in Theoretical Computer Science **10** (1998), pp. 246–266.

[13] McCarthy, J., *A basis for a mathematical theory of computation*, in: *Computer Programming and Formal Systems*, North-Holland, Amsterdam, 1963 pp. 33–70.

[14] Moran, A. K., "Call-by-name, Call-by-need, and McCarthy's Amb," Ph.D. thesis, Department of Computing Science, Chalmers University of Technology and University of Gothenburg (1998).

[15] Ong, C.-H. L., *Lazy lambda calculus: Theories, models and local structure characterization*, in: *19th ICALP*, Lecture Notes in Computer Science **623** (1992), pp. 487–498.

[16] Ong, C.-H. L., *Non-determinism in a functional setting*, in: *Proc. 8th Annual IEEE Symposium on Logic in Computer Science, Montreal*, 1993, pp. 275–286.

[17] Panangaden, P., *McCarthy's amb cannot implement fair merge*, in: *Proc. 8th FST&TCS Conference*, Lecture Notes in Computer Science **338** (1988), pp. 348–363.

[18] Panangaden, P. and V. Shanbhogue, *The expressive power of indeterminate dataflow primitives*, Information and Computation **98** (1992), pp. 99–131.

[19] Panangaden, P. and E. W. Stark, *Computations, residuals, and the power of indeterminancy*, in: *15th ICALP*, Lecture Notes in Computer Science **317** (1988), pp. 439–454.

[20] Park, D., *The fairness problem and nondeterministic computing networks*, in: *Foundations of Computer Science IV*, Mathematical Centre Tracts **159** (1983), pp. 133–161.

[21] Sangiorgi, D., *The lazy lambda calculus in a concurrency scenario*, Information and Computation **111** (1994), pp. 120–153.

[22] Søndergaard, H. and P. Sestoft, *Non-determinism in functional languages*, Computer Journal **35** (1992), pp. 514–523.