INFORMATION AND COMPUTATION **124**, 103–112 (1996)
ARTICLE NO. 0008

Proving Congruence of Bisimulation in Functional Programming Languages

DOUGLAS J. HOWE

AT&T Bell Laboratories, 600 Mountain Avenue, Room 2B-438, Murray Hill, New Jersey 07974
E-mail: howe@research.att.com

We give a method for proving congruence of bisimulation-like equivalences in functional programming languages. The method applies to languages that can be presented as a set of expressions together with an evaluation relation. We use this method to show that some generalizations of Abramsky's applicative bisimulation are congruences whenever evaluation can be specified by a certain natural form of structured operational semantics. One of the generalizations handles nondeterminism and diverging computations.

1. INTRODUCTION

One way to view a functional programming language is as an *evaluation system*, which we define to be a set of terms together with an evaluation relation. The intention is that a term a evaluates to another term v, written $a \Rightarrow v$, if there is a computation starting with a that terminates with result v.

Program equivalence is central to reasoning about functional programs. The question arises of when two programs in an evaluation system are to be considered equivalent. There are two properties one might expect for a reasonable equivalence, at least when computation is deterministic. First, if two programs are equivalent and evaluation of one of them terminates (that is, produces a value), then so should evaluation of the other, and the two resulting values should be equivalent. Second, if two values v and v' are equivalent, then they should both be the same kind of value, and, furthermore, the components of v and v' should be equivalent. For example, if $v = \langle v_1, v_2 \rangle$ then v' should be a pair $\langle v'_1, v'_2 \rangle$ with v_1 equivalent to v'_1 and v_2 equivalent to v'_2.

The case where v and v' are functions is less clear. Suppose $v = \lambda x \cdot b$ and $v' = \lambda x \cdot b'$. In the case of the lazy λ-calculus, it is possible for a computation involving v or v' to substitute any term whatsoever for x. Thus if v and v' are equivalent, then for all terms e, $b[e/x]$ and $b'[e/x]$ should be equivalent.

If these properties are taken as a definition of program equivalence, then in the case of the lazy λ-calculus we get the *applicative bisimulation* of Abramsky (1990). The lazy λ-calculus can be viewed as an evaluation system $\mathscr{E}_\lambda$ by taking the following two rules as an inductive definition of evaluation.

$$\frac{f \Rightarrow \lambda x.b \quad b[a/x] \Rightarrow c}{f(a) \Rightarrow c} \qquad \frac{}{\lambda x.b \Rightarrow \lambda x.b}.$$

Applicative bisimulation can now be defined as the symmetric closure of the largest binary relation $\leqslant$ over closed terms such that if $a \leqslant a'$ and $a \Rightarrow \lambda x.b$ then there is a b' such that $a' \Rightarrow \lambda x.b'$ and for all closed e, $b[e/x] \leqslant b'[e/x]$. An equivalent formulation, which is more suggestive of the name "applicative bisimulation," is the following. Say that a *converges* if there is a c such that $a \Rightarrow c$. Then $\leqslant$ is the largest relation on closed terms such that if $a \leqslant a'$ and a converges then a' converges and for all closed e, $a(e) \leqslant a'(e)$.

Using this definition it is trivial to verify, for example, that $(\lambda x.b)(a) \sim b[a/x]$, since if one side evaluates to some value, then the other side evaluates to the same value. However, in order for bisimulation to be a useful program equivalence, we need to be able to substitute equals for equals.

We can do this because bisimulation is a *congruence*: if $a \sim b$ and $C[\cdot]$ is a context (a term with a hole) then $C[a] \sim C[b]$. This can be proved using a technique from Berry (1981). The basic idea is to show that two terms are bisimilar if and only if they are *observationally congruent*, in the sense that replacing one by the other in any context preserves observable results of evaluation. For $\mathscr{E}_\lambda$, the observable result of evaluating a term is simply the fact that the term has a value, and so a and b are observationally congruent if and only if for all contexts $C[\cdot]$, $C[a]$ converges and only if $C[b]$ does.

Similar proofs have been done for more complicated languages than $\mathscr{E}_\lambda$. The proofs are not particularly difficult, but they are all done from first principles and involve detailed analysis of the reduction of terms that involve contexts. See, for example, Talcott (1985), Bloom (1990) and Jagadeesan (1991).

This paper makes two main contributions. The first is a simpler and more general method for proving congruence of bisimulation-like equivalences for evaluation systems. It is

0890-5401/96 $18.00

more general because it does not require bisimulation to be the same as some form of observational congruence to which the known techniques can be applied. It is simpler because it eliminates reasoning about contexts and because most of the work in applying it to particular evaluation systems and simulations can be factored out and captured in some simple technical lemmas.

The second contribution is a formalism for specifying evaluation relations that guarantees that certain bisimulation-like equivalences are congruences. This formalism is in the general spirit of Plotkin (1981) and the "natural semantics" of Kahn (1987). An evaluation system whose evaluation can be defined using this formalism is called a *structured* evaluation system.

Section 2 defines evaluation systems and deals with some basic syntactic matters. So that our results are applicable to call-by-value languages, the syntax of evaluation systems distinguishes ordinary variables from those for which only values can be substituted. Our treatment of call-by-value is adapted from Ong (1992).

A binary relation $\leqslant$ over terms is a *precongruence* if it is a preorder (a reflexive, transitive relation) and if for all contexts $C[\,\cdot\,]$, if $a \leqslant b$ then $C[a] \leqslant C[b]$. One of the key ideas in our proof method is to define a derived relation, called the "precongruence candidate," such that a preorder is a precongruence if and only if it is the same as its precongruence candidate. The precongruence candidate is the subject of Section 3.

Section 4 gives a rather abstract treatment of bisimulation-like relations. We define a direct generalization of bisimulation (and its underlying preorder) for an arbitrary evaluation system, and give a characterization of when it is a congruence. However, this generalization is usually inadequate in the presence of nondeterminism, since it will not distinguish between, for example, the value 17 and a program that can nondeterministically choose either to return 17 or to start a diverging computation. There is no single natural notion of divergence for an arbitrary evaluation system, but we can define a version of bisimulation that is parameterized with respect to the notion of divergence, and prove a characterization of when this is a congruence. Section 4 also gives a simple example of our proof method, using it to prove that bisimulation in $\mathscr{E}_\lambda$ is a congruence.

Section 5 gives a brief account of observational congruence and proves a "context lemma": if the evaluation system is deterministic then bisimulation is the same as observational congruence.

Section 6 gives the formalism for structured evaluation systems and proves that the first generalization of bisimulation referred to above is a congruence. Section 7 defines a particular notion of divergence for *finitary* structured evaluation systems, and shows that for this notion, the second generalization of bisimulation is a congruence.

The final section discusses some related work.

2. EVALUATION SYSTEMS

Informally, an evaluation system is a set of terms together with an evaluation relation. Terms are formed from variables and from applications of operators to sequences of operands. An operand consists of a term together with a list of variables that bind in it. In this setting, $\lambda x.b$ and $f(a)$ from the λ-calculus become $\lambda(x.b)$ and $ap(f; a)$, where f and a are operands with empty lists of bound variables. We will usually use conventional notations for specific terms, when they exist, relying on context to indicate, for example, that $f(a)$ means $ap(f; a)$ and not the application of an operator f to an operand a.

Our approach to syntax is essentially the same as in Aczel (1978) (also described in Klop 1980, Remark 1.5), except for a slight modification to incorporate the idea of Ong (1992) for handling call-by-value computation.

To deal with call-by-value, the syntax of an evaluation system has two kinds of variables. Let Var_1 and Var_2 be disjoint infinite sets of variables, and let $Var = Var_1 \cup Var_2$. A variable is *ordinary* if $x \in Var_1$, otherwise it is a *value* variable. The intention is that during computation, only fully evaluated terms may be substituted for value variables. Two variables $x, y \in Var$ have the *same kind* if $x, y \in Var_1$ or $x, y \in Var_2$. We say that two sequences of variables $\bar{x} = x_1, ..., x_n$ and $\bar{x}' = x'_1, ..., x'_{n'}$ have the same kind if $n = n'$ and if for each i, $1 \leqslant i \leqslant n$, x_i and x'_i have the same kind.

The syntax of an evaluation system is specified by a signature $L = (O, \alpha)$ where O is a set of *operators* and α is a function assigning to each operator an *arity*, which is a sequence of sequences of distinct variables. The arity of an operator specifies the number of operands the operator takes, and the number and kind of binding variables for each operand. For example, a signature for the lazy λ-calculus is $L_\lambda = (O, \alpha)$ where $O = \{ap, \lambda\}$, $\alpha(ap) = (\varnothing, \varnothing)$ and $\alpha(\lambda) = ((x))$ for some ordinary variable x.

A *term* over L is either a variable or has the form $\tau(s_1;...; s_n)$ where $\tau \in O$, $\alpha(\tau) = (\bar{z}_1, ..., \bar{z}_n)$ for some $\bar{z}_1, ..., \bar{z}_n$, and each s_i is an *operand* of the form $\bar{x}_i.a_i$ where a_i is a term and $\bar{x}_i$ is a sequence of distinct variables having the same kind as $\bar{z}_i$.

We impose a binding structure on terms by specifying that in each operand $\bar{x}.a$, all free occurrences in a of variables in $\bar{x}$ become bound in $\bar{x}.a$. The usual notions of substitution and α-equality apply, with the additional restriction that for operands $\bar{x}.a$ and $\bar{x}'.a'$ to be α-equal, $\bar{x}$ and $\bar{x}'$ must have the same kind. We identify α-equal terms and operands.

L is a *call-by-value* signature if for all $\tau \in O$, $\alpha(\tau)$ contains only value variables. L is a *lazy* signature if for all $\tau \in O$, $\alpha(\tau)$ contains only ordinary variables. A term a is *spurious* if L is call-by-value and a contains an ordinary variable, or if L is lazy and a contains a value variable. We rule out spurious

terms in what follows for minor technical reasons having only to do with the context lemma in Section 5.

Let T^L be the set of non-spurious terms over L and T_0^L the set of closed terms. We drop the superscripts when they are clear from context. Suppose $\eta \subset T \times T$. For $a, b \in T$, write $a\,\eta\,b$ for $(a, b) \in \eta$. For operands s and s', define $s\,\eta\,s'$ if there exist operands $\bar{z}.b$ and $\bar{z}.b'$ such that $s = \bar{z}.b$, $s' = \bar{z}.b'$ and $b\,\eta\,b'$. If $\bar{s}$ and $\bar{s}'$ are operand sequences $s_1, ..., s_n$ and $s'_1, ..., s'_n$, respectively, then define $\bar{s}\,\eta\,\bar{s}'$ if $s_i\,\eta\,s'_i$ for each i, $1 \leqslant i \leqslant n$.

DEFINITION 2.1. An *evaluation system* is a pair $\mathscr{E} = (L, \Rightarrow)$ where L is a signature, $\Rightarrow \subset T_0 \times T_0$ and for all $a, v \in T_0$, if $a \Rightarrow v$, then for all v', $v \Rightarrow v'$ if and only if $v = v'$.

The relation $\Rightarrow$ is called the *evaluation relation* of the evaluation system, and a term $v \in T_0$ is a *value* if $v \Rightarrow v$. Note that v is a value if and only if there exists $a \in T_0$ such that $a \Rightarrow v$. Let V be the set of all values.

An example of an evaluation system is $\mathscr{E}_\lambda$, whose signature is L_λ and whose evaluation relation is defined in the introduction. The values of $\mathscr{E}_\lambda$ are all closed terms of the form $\lambda x.b$.

We extend relations on closed terms to open terms by substituting closed terms for free variables. To make this way of extending relations respect the distinction between ordinary and value variables, define a *closing substitution* to be a substitution of closed terms for all variables such that for all value variables x, $\sigma(x) \in V$.

DEFINITION 2.2. For $\eta \subset T_0 \times T_0$, define $\eta^\circ \subset T \times T$, the *extension of η to open terms*, by $a\,\eta^\circ\,a'$ if $\sigma(a)\,\eta\,\sigma(a')$ for every closing substitution σ. Also, for $\eta \subset T \times T$, define η_0, the restriction *of* η to closed terms, by $\eta_0 = \eta \cap (T_0 \times T_0)$.

Note that if $\eta \subset T_0 \times T_0$, then $(\eta^\circ)_0 = \eta$.

A relation $\eta \subset T \times T$ is *operator respecting* if for all $\tau(\bar{s})$, $\tau(\bar{s}') \in T$, if $\bar{s}\,\eta\,\bar{s}'$ then $\tau(\bar{s})\,\eta\,\tau(\bar{s}')$. An operator-respecting preorder is a *precongruence*, and an operator-respecting equivalence relation is a *congruence*. If η is defined to be a relation over closed terms then we say that η is operator respecting, is a precongruence, or is a congruence, respectively, if η° is.

As an example, consider $\mathscr{E}_\lambda$. A preorder $\leqslant$ over closed terms of $\mathscr{E}_\lambda$ is a precongruence if and only if

1. for all terms b, b', *if* $b \leqslant^\circ b'$ then $\lambda x.b \leqslant^\circ \lambda x.b'$, and
2. for all terms a, a', f, f', if $a \leqslant^\circ a'$ and $f \leqslant^\circ f'$ then $f(a) \leqslant^\circ f'(a')$.

3. THE PRECONGRUENCE CANDIDATE

This section gives the key technical idea in our proof method, which is the definition of an auxiliary relation called the "precongruence candidate." This is defined in terms of the preorder η we wish to prove a precongruence, and the proof of precongruence will involve showing η and its precongruence candidate to be the same.

Let $\eta \subset T_0 \times T_0$ be a preorder. The *precongruence candidate* for η, written $\hat{\eta}$, is a relation that, by definition, is operator-respecting and contains η°. Informally, $a\,\hat{\eta}\,b$ if b can be obtained from a via one bottom-up pass of replacements of subterms by terms that are larger under η. This idea is formalized in the following definition.

DEFINITION 3.1. Let $\eta \subset T_0 \times T_0$ be a preorder. Define $a\,\hat{\eta}\,b$, for $a, b \in T$, by induction on the size of a.

- For a variable x, if $x\,\eta^\circ\,b$ then $x\,\hat{\eta}\,b$.
- For $\tau(\bar{s}), \tau(\bar{s}') \in T$, if $\bar{s}\,\hat{\eta}\,\bar{s}'$ and $\tau(\bar{s}')\,\eta^\circ\,b$ then $\tau(\bar{s})\,\hat{\eta}\,b$.

The following lemma gives some basic properties of the precongruence candidate. We will often use these properties implicitly in the rest of the paper.

LEMMA 3.1. *If $\eta \subset T_0 \times T_0$ is a preorder then the following hold*:

1. *$\hat{\eta}$ is reflexive.*
2. *$\hat{\eta}$ is operator respecting.*
3. *$\eta^\circ \subset \hat{\eta}$.*
4. *If $a\,\hat{\eta}\,b$ and $b\,\eta^\circ\,c$, then $a\,\hat{\eta}\,c$.*

Proof. Property 1 follows by induction on term size, Definition 3.1, and the reflexivity of η. Property 2 is an immediate consequence of Definition 3.1 and the reflexivity of η. Property 3 follows from Definition 3.1 and property 1. Property 4 follows by Definition 3.1 and the transitivity of η. ∎

An important part of our proof method is to show that the precongruence candidate is preserved by computation: if $a\,\hat{\eta}\,b$ and we evaluate a to get a' then we can evaluate b getting a b' such that $b\,\hat{\eta}\,b'$. As computation typically involves substitution, it is important that $\hat{\eta}$ be preserved under substitution. A sufficient formulation of this property is the following.

LEMMA 3.2. *Let $\eta \subset T_0 \times T_0$ be a preorder and suppose $a, a', b, b' \in T$. Suppose that x is a variable and that if x is a value variable then a and a' are either values or value variables. If $a\,\hat{\eta}\,a'$ and $b\,\hat{\eta}\,b'$ then $b[a/x]\,\hat{\eta}\,b'[a'/x]$.*

Proof. The proof is by induction on the size of b. In the case where b is x, we have $x\,\eta^\circ\,b'$, so $a'\,\eta^\circ\,b'[a'/x]$ by definition of η°. Since $a\,\hat{\eta}\,a'$, $a\,\hat{\eta}\,b'[a'/x]$. In the case that b is a variable $y \neq x$, since $y\,\eta^\circ\,b'$ we have $y\,\eta^\circ\,b'[a'/x]$. Now suppose $b = \tau(\bar{s})$ and let $\bar{s}'$ be such that $\bar{s}\,\hat{\eta}\,\bar{s}'$ and $\tau(\bar{s}')\,\eta^\circ\,b'$. By the induction hypothesis, $\bar{s}[a/x]\,\hat{\eta}\,\bar{s}'[a'/x]$. Also, $\tau(\bar{s}')[a'/x]\,\eta^\circ\,b'[a'/x]$, so $\tau(\bar{s})[a/x]\,\hat{\eta}\,b'[a'/x]$. ∎

An immediate consequence of the above lemma is that $\hat{\eta} \subset (\hat{\eta}_0)^\circ$. This last fact will be used several times in the next section.

It is because of Lemma 3.2 that we cannot take $\hat{\eta}$ to be the simpler auxiliary relation used in Groote and Vaandrager (1992). That relation can be inductively defined by

1. $\eta^\circ \subset \hat{\eta}$ and
2. if $\bar{s}\ \hat{\eta}\ \bar{s}'$ then $\tau(\bar{s})\ \hat{\eta}\ \tau(\bar{s}')$.

Lemma 3.2 fails for this definition, as the following example shows. Suppose that $\mathscr{E} = \mathscr{E}_\lambda$, and define $a \approx a'$ if there exists a c such that $a \Rightarrow c$ and $a' \Rightarrow c$. Let $a = \lambda x.x$, $a' = (\lambda x.x)(\lambda x \cdot x)$, $b = \lambda x.\, y$, and $b' = (\lambda x.x)(\lambda x.\, y)$. Then $a \approx a'$ and $b \approx^\circ b'$, so $a \mathrel{\hat{\approx}} a'$ and $b \mathrel{\hat{\approx}} b'$, but $b[a/y] \not\mathrel{\hat{\approx}} b'[a'/y]$.

THEOREM 3.1. *Suppose $\eta \subset T_0 \times T_0$ is a preorder. Then the following are equivalent*:

1. *η° is a precongruence.*
2. *$\hat{\eta} \subset \eta^\circ$.*
3. *$\hat{\eta}_0 \subset \eta$.*

Proof. We show $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 1$. For the first implication, if η° is a precongruence, then by induction on the definition of $a\ \hat{\eta}\ b$, if $a\ \hat{\eta}\ b$ then $a\ \eta^\circ\ b$. For the second, if $\hat{\eta} \subset \eta^\circ$ then $\hat{\eta}_0 \subset (\eta^\circ)_0 = \eta$. Finally, if $\hat{\eta}_0 \subset \eta$ then, by Lemma 3.2, $\hat{\eta} \subset (\hat{\eta}_0)^\circ \subset \eta^\circ$, so $\hat{\eta} = \eta^\circ$. ∎

Define $v^\top$ to be the transpose of a binary relation v, so that $a\ v^\top\ b$ if and only if $b\ v\ a$. Denote the transitive closure of v by v^*.

The following technical lemma is used in later sections.

LEMMA 3.3. *Suppose that η is an equivalence relation. Then $\hat{\eta}^*$ is symmetric, and if $v \subset T_0 \times T_0$ is a preorder and $\hat{\eta}_0 \subset v$ then $\hat{\eta}_0 \subset v^\top$.*

Proof. A straightforward induction on the definition of $\hat{\eta}$ shows that $\hat{\eta} \subset \hat{\eta}^{*\top}$, so $\hat{\eta}^*$ is symmetric. By Lemma 3.2, $(\hat{\eta}^*)_0 = (\hat{\eta}_0)^*$, so we have

$$(\hat{\eta}_0)^\top \subset ((\hat{\eta}^*)_0)^\top = ((\hat{\eta}^*)^\top)_0 = (\hat{\eta}^*)_0 = (\hat{\eta}_0)^* \subset v^* = v. \quad ∎$$

4. SIMULATIONS

A property shared by the preorders our proof method applies to is that they are all *simulation relations* (or "simulations" for short), in a sense to be made precise shortly. In this section, we prove a few useful facts about simulations. These facts have little intrinsic interest; rather, they constitute an attempt to capture, at a more abstract level, parts of the proof method which would otherwise need to be repeated each time the method is applied to a concrete evaluation system. The main results are Theorems 4.1 and 4.2, which give characterizations, in terms of the precongruence candidate, of when two particular co-inductively-defined simulations are operator-respecting. As an example after Theorem 4.1, we apply the theorem to prove that applicative bisimulation in $\mathscr{E}_\lambda$ is a congruence.

Let $\mathscr{E} = (L, \Rightarrow)$ be an evaluation system.

DEFINITION 4.1. For $\eta \subset T \times T$, define $[\eta] \subset T_0 \times T_0$ by $a\ [\eta]\ a'$ if for all terms $\theta(\bar{s})$, if $a \Rightarrow \theta(\bar{s})$ then there exists $\bar{s}'$ such that $a' \Rightarrow \theta(\bar{s}')$ and $\bar{s}\ \eta\ \bar{s}'$.

Note that if c and c' are values and $c\ [\eta]\ c'$ then c and c' can be written as $\theta(\bar{t})$ and $\theta(\bar{t}')$, respectively, with $\bar{t}\ \eta\ \bar{t}'$. Thus if $c\ [\hat{\eta}]\ c'$ then $c\ \hat{\eta}\ c'$. Also, note that

$$[\eta_1] \circ [\eta_2] \subset [\eta_1 \circ \eta_2]$$

for all $\eta_1, \eta_2 \subset T \times T$.

DEFINITION 4.2. Suppose $\eta \subset T_0 \times T_0$. η is a *simulation* if η is a preorder and $\eta \subset [\eta^\circ]$. If β is a preorder, then η is a *β-simulation* if it is a simulation and $\eta \subset \beta$.

The parameter β is for handling divergence. In Section 6, we define $a\!\downarrow$ to mean that there are no non-terminating computations starting with a, where the notion of computation is based on a particular inductive presentation of the evaluation relation. One possibility for β is to define $a\ \beta\ a'$ if $a\!\downarrow$ implies $a'\!\downarrow$. In this section, however, there are some useful properties we can prove with minimal assumptions on β.

We now define some simulations of particular interest. We first give the definitions, then prove that the defined simulations exist.

DEFINITION 4.3. For $\beta \subset T_0 \times T_0$ a preorder, define $\leqslant_\beta$ to be the largest β-simulation. Let $\sim_\beta$ be the symmetric closure of $\leqslant_\beta$, and let $\leqslant = \leqslant_{T_0 \times T_0}$ and $\sim = \sim_{T_0 \times T_0}$.

In $\mathscr{E}_\lambda$, $\sim$ is the relation of applicative bisimulation we defined in the introduction.

DEFINITION 4.4 Let $\beta \subset T_0 \times T_0$ be an equivalence relation. A relation $\eta \subset T_0 \times T_0$ is a β-bisimulation if it is an equivalence relation and $\eta \subset \beta \cap [\eta^\circ] \cap [\eta^{\circ\top}]^\top$. Define $\simeq_\beta$ to be the largest β-bisimulation.

If $F \in T_0 \times T_0 \to T_0 \times T_0$ is monotone with respect to inclusion of relations as sets, then there exists $\tilde{\eta}$ which is the largest $\eta \subset T_0 \times T_0$ such that $\eta \subset F(\eta)$. It is straightforward to show that $\tilde{\eta}$ is the greatest fixed-point of F. The definition of $\tilde{\eta}$ is *co-inductive* and gives a corresponding principle of co-induction: to show that $\eta \subset \tilde{\eta}$ for some relation $\eta \subset T_0 \times T_0$, it suffices to show that $\eta \subset F(\eta)$.

The operation $\eta \mapsto [\eta]$ is monotone, so we can define $\leqslant_\beta$ and $\simeq_\beta$ as greatest fixed-points of monotone functions. They satisfy the fixed-point equations

$$\leqslant_\beta = \beta \cap [\leqslant_\beta^\circ]$$

and

$$\simeq_\beta = \beta \cap [\simeq_\beta^\circ] \cap [\simeq_\beta^{\circ\top}]^\top.$$

To finish showing that Definitions 4.3 and 4.4 are well-formed, we use co-induction to prove that the greatest fixed-point $\leqslant_\beta$ is a preorder, and that $\simeq_\beta$ is an equivalence relation.

Let I be the set of all (a, a) with $a \in T_0$. Trivially $I \subset \beta \cap [I^\circ]$, so by co-induction, $I \subset \leqslant_\beta$ and $\leqslant_\beta$ is reflexive. Using the fixed-point property of $\leqslant_\beta$, we have

$$\leqslant_\beta \circ \leqslant_\beta \subset (\beta \cap [\leqslant_\beta^\circ]) \circ (\beta \cap [\leqslant_\beta^\circ]) \subset \beta \cap [(\leqslant_\beta \circ \leqslant_\beta)^\circ]$$

so by co-induction, $\leqslant_\beta \circ \leqslant_\beta \subset \leqslant_\beta$ and $\leqslant_\beta$ is transitive.

The proof that $\simeq_\beta$ is a preorder is similar. To show $\simeq_\beta$ is symmetric it suffices to show $\simeq_\beta^\top \subset \simeq_\beta$. This follows by co-induction because, transposing the fixed-point property of $\simeq_\beta$, we have

$$\simeq_\beta^\top \subset \beta^\top \cap [\simeq_\beta^\circ]^\top \cap [\simeq_\beta^{\circ\top}] = \beta \cap [(\simeq_\beta^\top)^{\circ\top}]^\top \cap [(\simeq_\beta^\top)^\circ].$$

To state Theorem 4.1 it is useful to introduce some terminology relating evaluation and the precongruence candidate of a simulation.

DEFINITION 4.5. Suppose $v \subset T \times T$. An evaluation pair is a pair of terms (a, c) such that $a \Rightarrow c$. The evaluation pair (a, c) *respects* v if for all closed a' such that $a \, v \, a'$ there is a c' such that $a' \Rightarrow c'$ and $c[v] \, c'$. An evaluation pair $(\tau(\bar{s}), c)$ *respects* v *on subterms* if for all closed $\tau(\bar{s}')$ such that $\bar{s} \, v \, \bar{s}'$ there is a c' such that $\tau(\bar{s}') \Rightarrow c'$ and $c[v] \, c'$.

We say that the evaluation relation $\Rightarrow$ respects v (on subterms) if all evaluation pairs do. Note that $\Rightarrow$ respects v if and only if $v_0 \subset [v]$.

LEMMA 4.1. *Let η be a simulation. If an evaluation pair respects $\hat{\eta}$ on subterms then it respects $\hat{\eta}$.*

Proof. Let $(\tau(\bar{s}), c)$ be an evaluation pair and suppose $\tau(\bar{s}) \, \hat{\eta} \, b$. There exists $\bar{s}'$ such that $\bar{s} \, \hat{\eta} \, \bar{s}'$ and $\tau(\bar{s}') \, \eta^\circ \, b$. By Lemma 3.2 we may assume that $\tau(\bar{s}')$ is closed. Since $(\tau(\bar{s}), c)$ respects $\hat{\eta}$ on subterms, there is a c'' with $\tau(\bar{s}') \Rightarrow c''$ and $c \, \hat{\eta} \, c''$. Since η is a simulation, there is a c' with $b \Rightarrow c'$ and $c'' \, [\eta^\circ] \, c' . c \, [\hat{\eta}] \, c'$. ∎

THEOREM 4.1. *Let $\beta \subset T_0 \times T_0$ be a preorder and let $\eta = \leqslant_\beta$. η is a precongruence if and only if evaluation respects $\hat{\eta}$ and $\hat{\eta}_0 \subset \beta$.*

Proof. Suppose η is a precongruence. By Theorem 3.1, $\eta^\circ = \hat{\eta}$. By definition of η, $\hat{\eta}_0 = \eta \subset \beta$. We also have $\hat{\eta}_0 = \eta \subset [\eta^\circ] = [\hat{\eta}]$, so evaluation respects $\hat{\eta}$. For the converse, by Theorem 3.1 and the definition of $\leqslant_\beta$ it suffices to show $\hat{\eta}_0 \subset \beta \cap [(\hat{\eta}_0)^\circ]$. Since evaluation respects $\hat{\eta}$, $\hat{\eta}_0 \subset [\hat{\eta}] \subset [(\hat{\eta}_0)^\circ]$. ∎

EXAMPLE. We can use what we have developed so far to prove that applicative bisimulation in $\mathscr{E}_\lambda$ is a congruence. It suffices to show that $\leqslant$ is a precongruence.

By Lemma 4.1 and Theorem 4.1 it suffices to show that every evaluation pair respects $\hat{\leqslant}$ on subterms. We do this by induction on the definition of evaluation in $\mathscr{E}_\lambda$. In the case of the evaluation rule for application, we have $f(a) \Rightarrow c$, $f \Rightarrow \lambda x.b$ and $b[a/x] \Rightarrow c$. Suppose $f \mathrel{\hat{\leqslant}} f'$ and $a \mathrel{\hat{\leqslant}} a'$. By the induction hypothesis and Lemma 4.1, $(f, \lambda x.b)$ respects $\hat{\leqslant}$ so for some b', $f' \Rightarrow \lambda x.b'$ and $b \mathrel{\hat{\leqslant}} b'$. By Lemma 3.2, $b[a/x] \mathrel{\hat{\leqslant}} b'[a'/x]$. Now $(b[a/x], c)$ respects $\hat{\leqslant}$, so there exists c' such that $b'[a'/x] \Rightarrow c'$ *and* $c[\hat{\leqslant}] \, c'$. The case for the other evaluation rule is trivial.

LEMMA 4.2. *If $\beta \subset T_0 \times T_0$ is a preorder and $\eta \subset T_0 \times T_0$ is a symmetric β-simulation then $\eta \subset \simeq_\beta$,*

Proof. $\eta \subset [\eta^\circ]$ so $\eta = \eta^\top \subset [\eta^\circ]^\top = [\eta^{\circ\top}]^\top$. Since $\simeq_\beta$ is the largest β-bisimulation, $\eta \subset \simeq_\beta$. ∎

THEOREM 4.2. *Suppose that β is an equivalence relation and let $\eta = \simeq_\beta$. Then η is a congruence if and only if evaluation respects $\hat{\eta}$ and $\hat{\eta}_0 \subset \beta$.*

Proof. The forward direction is the same as in the proof of Theorem 4.1. Suppose evaluation respects $\hat{\eta}$ and $\hat{\eta}_0 \subset \beta$. Then $\hat{\eta}_0 \subset \beta \cap [\hat{\eta}]$ and we have

$$(\hat{\eta}^*)_0 = (\hat{\eta}_0)^* \subset \beta \cap [\hat{\eta}^*] \subset \beta \cap [((\hat{\eta}^*)_0)^\circ].$$

By Lemma 3.3, is symmetric and so $(\hat{\eta}^*)_0 \subset \eta$ by Lemma 4.2. Hence $\hat{\eta}_0 \subset \eta$ and we can apply Theorem 3.1. ∎

5. THE CONTEXT LEMMA

Let $\mathscr{E} = (L, \Rightarrow)$ be an evaluation system where $L = (O, \alpha)$, and let $\leqslant_1 = [T_0 \times T_0]$. For $a, b \in T$, define $a \leqslant_C b$ if $C[a] \leqslant_1 C[b]$ for all contexts $C[\cdot]$ such that $C[a]$ and $C[b]$ are closed. Define a to be *observationally congruent* to b, and write $a \sim_C b$, if $a \leqslant_C b$ and $b \leqslant_C a$. Thus a is observationally congruent to b if no "experiment" can distinguish a and b, where an experiment consists of placing a context around a term, evaluating, and observing whether evaluation terminated and, if so, what was the outermost operator of the resulting value.

Recall that we defined $\sim$ to be the symmetric closure of the largest simulation. Under certain conditions, $\sim^\circ$ is the same as observational congruence. It can fail to be so for three basic reasons.

Nondeterminism. For example, consider extending $\mathscr{E}_\lambda$ with an erratic choice operator $+$ whose evaluation is defined by

$$\frac{a \Rightarrow c}{a + b \Rightarrow c} \quad \frac{b \Rightarrow c}{a + b \Rightarrow c}.$$

Let k_1 and k_2 be two closed terms such that $k_1 \not\sim k_2$. Then

$$(\lambda x.k_1) + (\lambda x.k_2) \sim_C \lambda x.k_1 + k_2$$

but

$$(\lambda x.k_1)+(\lambda x.k_2) \not\sim \lambda x.k_1+k_2.$$

Missing value accessors. This occurs when the evaluation system does not have "enough destructors" to allow computations to access all components of values. For example, consider the evaluation system with the syntax of the λ-calculus but with evaluation defined by the single rule

$$\overline{\lambda x.b \Rightarrow \lambda x.b}.$$

Let k_1 and k_2 be closed terms such that $k_1 \not\sim k_2$. $\lambda x.k_1 \not\sim \lambda x.k_2$, but the terms are observationally congruent because no context can "expose" what is under the lambda abstraction.

Observational congruence not preserved under closing substitutions. For example, if we take the evaluation system of the previous paragraph and add, say, constants for the integers and an addition operator with appropriate evaluation rules, then $x+0 \not\sim^\circ x+1$. However, $x+0 \sim_C x+1$ because no context can perform a substitution of a closed term for x. For example, the context $(\lambda x.\cdot)(0)$ will not work because there is no evaluation rule for application.

If we rule out all of these possibilities then $\sim$ and $\sim_C$ are the same relation when $\sim$ is a congruence. To state this precisely, we need a few definitions.

Evaluation is *determinate* if $b=c$ whenever $a \Rightarrow b$ and $a \Rightarrow c$. Suppose $\theta \in O$ and let $n=|\alpha(\theta)|$. $\mathscr{E}$ is *computationally complete* for θ if for all i, $1 \leqslant i \leqslant n$, and for all closing substitutions σ, there is a context $C_{\theta,i,\sigma}[\cdot]$ such that for all values of the form $\theta(\bar{s})$, if the ith operand of $\bar{s}$ is $\bar{z}.b$ where $\bar{z}=\alpha(\theta)_i$, then

$$C_{\theta,i,\sigma}[\theta(\bar{s})] \sim \sigma(b).$$

$\mathscr{E}$ is computationally complete if it is computationally complete for each of its operators.

THEOREM 5.1 (Context Lemma). *If $\mathscr{E}$ is determinate and computationally complete, and if $\sim$ is a congruence and $\leqslant_C \subset ((\leqslant_C)_0)^\circ$, then $\sim_C = \sim^\circ$.*

Proof. Clearly $\sim^\circ \subset \sim_C$.

To prove $\sim_C \subset \sim^\circ$ it suffices to show that $(\leqslant_C)_0 \subset [((\leqslant_C)_0)^\circ]$. Suppose, then, that $a, b \in T_0$, $a \leqslant_C b$ and $a \Rightarrow \theta(\bar{s})$. Since $a \leqslant_1 b$, there exists $\bar{s}'$ such that $b \Rightarrow \theta(\bar{s}')$. Suppose $1 \leqslant i \leqslant |\alpha(\theta)|$ and let the ith members of $\bar{s}$ and $\bar{s}'$ be $\bar{z} \cdot b$ and $\bar{z} \cdot b'$, respectively, where $\bar{z}=\alpha(\theta)_i$. It suffices to show that for all closing substitutions, $\sigma(b) \leqslant_C \sigma(b')$. Since $\mathscr{E}$ is determinate, if $u \Rightarrow v$ then $u \sim v$, so

$$\begin{aligned}\sigma(b) &\sim C_{\theta,i,\sigma}[\theta(\bar{s})] \sim C_{\theta,i,\sigma}[a] \\ &\leqslant_C C_{\theta,i,\sigma}[b] \sim C_{\theta,i,\sigma}[\theta(\bar{s}')] \sim \sigma(b').\end{aligned}$$

The hypothesis that $\leqslant_C$ be preserved under closing substitutions is fairly weak. For example, it holds (given that $\sim^\circ$ is a congruence) if $\mathscr{E}$ contains both the lazy and call-by-value λ-calculi. In the case where $\mathscr{E}$ is call-by-value, it is sufficient that $\mathscr{E}$ contain the call-by-value λ-calculus, and similarly for the lazy case. Finding analogues of Theorem 5.1 for other languages, such as nondeterministic ones, requires changing the notion of observation.

6. STRUCTURED EVALUATION SYSTEMS

In this section, we give a formalism for specifying evaluation relations that guarantees that certain simulations are precongruences. Inference rules for evaluation are specified using an extension of the set of terms over $L=(O, \alpha)$.

For each sequence $\bar{x}$ of distinct variables we fix an infinite set of new variables which we call the *metavariables of arity* $\bar{x}$. Assume that the set of metavariables of arity $\varnothing$ is partitioned into an infinite set of *ordinary* metavariables and an infinite set of *value* metavariables. A *term schema* is built in the same way as a term, except that we also include expressions of the form $P[\bar{a}]$ where P is a metavariable of arity $\bar{x}$ and $\bar{a}$ is a list of term schemas such that $|\bar{a}|=|\bar{x}|$ and for each i, $1 \leqslant i \leqslant |\bar{x}|$, if x_i is a value variable then a_i is either a value variable or a value metavariable. We write P for $P[\ ]$ and use capital letters in term schemas exclusively for metavariables. A simple term schema has the form $\tau(\bar{x}_1.P_1[\bar{x}_1]; ...; \bar{x}_n.P_n[\bar{x}_n])$ where the P_i are distinct metavariables.

An *instantiation* is a partial map σ from metavariables to operands such that if $\sigma(P)$ is defined then it has the form $\bar{x}.b$ where $\bar{x}$ has the same kind as the arity of P and all free variables of b are in $\bar{x}$. The application of σ to term schemas is similar to ordinary substitution, except that if $\sigma(P)$ is $x_1, ..., x_n.b$, then $\sigma(P[a_1, ..., a_n]) = b[\sigma(a_1), ..., \sigma(a_n)/x_1, ..., x_n]$.

An *evaluation rule* is an inference rule whose formulas all have the form $a \Rightarrow b$ where a and b are term schemas with no free variables. Specifically, an evaluation rule r consists of a set I_r, a total well-founded relation $\leqslant_r$ over I_r, a family of formulas $\{a_i \Rightarrow b_i\}_{i \in I_r}$ indexed by I_r, and a formula $a \Rightarrow b$ called the conclusion of r. The formulas $a_i \Rightarrow b_i$, $i \in I_r$, are called the *premises* of r. In addition, we require the following.

1. a is a simple term schema.

2. For all $i \in I_r$, b_i is a metavariable or a simple term schema, and has no metavariables in common with a or with b_j for $j \neq i$.

3. For each $i \in I_r$ and metavariable P of a_i, P occurs in a or in b_j for some $j <_r i$. Every metavariable in b occurs in b_i for some $i \in I_r$.

4. For each metavariable P occurring in the rule, P is a value metavariable if and only if P is b_i for some $i \in I_r$.

5. If b is a metavariable then it is b_i for some $i \in I_r$.

Such a rule will be called a rule for τ if τ is the operator in a.

An *instance* of an evaluation rule is the result of applying to its premises and conclusion an instantiation σ whose domain contains the set of metavariables occurring in the rule. Let R be a set of evaluation rules. *Derivations* over R are trees of instances of rules from R where the children of a node r are in one-to-one correspondence with the premises of r, and each child's conclusion is the same as the corresponding premise. The root rule instance is called the *last step* of the derivation. The *conclusion* of a derivation is the conclusion of the last step. A derivation ∇ is a derivation *for* a term u if the conclusion of ∇ is $u \Rightarrow v$ for some v.

DEFINITION 6.1. A *structured evaluation system* consists of an evaluation system $\mathscr{E}$ and a set R of evaluation rules whose formulas are term schemas over $\mathscr{E}$, such that the evaluation relation of $\mathscr{E}$ is the same as the relation $\Rightarrow$ defined by $a \Rightarrow b$ if the formula $a \Rightarrow b$ has a derivation over R. A structured evaluation system is *finitary* if there are a finite number of rules for each operator and if each rule has a finite number of premises.

We will usually identify a structured evaluation system with its underlying evaluation system when the set of rules is clear from context.

For example, the following two rules can be used to give a finitary structured evaluation system for the call-by-value λ-calculus:

$$\frac{F \Rightarrow \lambda x.B[x] \quad A \Rightarrow V \quad B[V] \Rightarrow C}{F(A) \Rightarrow C} \qquad \frac{}{\lambda x \cdot B[x] \Rightarrow \lambda x.B[x]}.$$

(By $F(A)$ above, we mean $ap(F; A)$.) Below are two different possible rules for pairing, one call-by-value and one lazy:

$$\frac{A \Rightarrow A' \quad B \Rightarrow B'}{\langle A, B\rangle \Rightarrow \langle A', B'\rangle} \qquad \frac{}{\langle A, B\rangle \Rightarrow \langle A, B\rangle}.$$

By definition of evaluation system (Definition 2.1), a set of evaluation rules forms a structured evaluation system only if for all a, $v \in T_0$, if $a \Rightarrow v$, then for all v', $v \Rightarrow v'$ if and only if $v = v'$. This condition seems to be easy to verify on a case by case basis. However, we can also give a fairly general syntactic sufficient condition.

LEMMA 6.1. *A set R of evaluation rules forms a structured evaluation system if there is a subset of operators $K \subset O$ satisfying the following two conditions. If $a \Rightarrow b$ is the conclusion of a rule for some $\tau \in O \backslash K$, then b is a metavariable. For each $\theta \in K$ there is exactly one rule for θ in R, and it has the form*

$$\frac{\{S_i \Rightarrow T_i\}_{1 \leqslant i \leqslant n,\, S_i \neq T_i}}{\theta(\bar{x}_1.S_1[\bar{x}_1];\ldots;\bar{x}_n.S_n[\bar{x}_n]) \Rightarrow \theta(\bar{x}_1.T_1[\bar{x}_1];\ldots;\bar{x}_n.T_n[\bar{x}_n])},$$

where for each i, $1 \leqslant i \leqslant n$, either $S_i = T_i$ or the arity of S_i is $\varnothing$.

The operators in K are called *canonical* and the other operators are called *noncanonical*. If we eliminate either of the rules for pairing in the example rules above, then the remaining three rules satisfy the conditions of Lemma 6.1 when we take λ and pairing to be the canonical operators.

LEMMA 6.2. *Let $\mathscr{E}$ be an evaluation system, and suppose that η is a simulation over $\mathscr{E}$. Suppose that a is a term schema and σ, σ' are instantiations such that for every metavariable P in a, $\sigma(P)$ and $\sigma'(P)$ are defined, $\sigma(P)\ \hat{\eta}\ \sigma'(P)$, and if P is a value metavariable then $\sigma(P)$ and $\sigma'(P)$ are values. Then $\sigma(a)\ \hat{\eta}\ \sigma'(a)$.*

Proof. The proof is a straightforward induction on the size of a, using the definitions of instantiation and term schema, Lemma 3.2 and the fact that $\hat{\eta}$ is a precongruence. ∎

THEOREM 6.1. *In any structured evaluation system $\mathscr{E}$, evaluation respects $\hat{\eta}$ for all simulations η.*

Proof. The proof is essentially the same as the one for $\mathscr{E}_\lambda$. We show that all evaluation pairs respect $\hat{\eta}$ on subterms, by induction on derivations. Suppose that ∇ is a derivation of $\tau(\vec{s}) \Rightarrow c$ and that $\tau(\vec{s}')$ is a closed term such that $\vec{s}\ \hat{\eta}\ \vec{s}'$. Let r be a rule and σ an instantiation such that the last step in ∇ is $\sigma(r)$. Let r have conclusion $a \Rightarrow b$ and premises $\{a_i \Rightarrow b_i\}_{i \in I_r}$.

We build an instantiation σ' such that

1. the domain of σ' is the set of metavariables occurring in a or b_j for $j \in I_r$,

2. $\sigma'(a) = \tau(\vec{s}')$,

3. $\sigma'(a_j) \Rightarrow \sigma'(b_j)$ has a derivation for $j \in I_r$, and

4. for every P in the domain of σ', $\sigma(P)\ \hat{\eta}\ \sigma'(P)$ and, furthermore, if P is a value metavariable then $\sigma(P)\ [\hat{\eta}]\ \sigma'(P)$.

We build σ' by induction over I_r, at each stage $i \in I_r$ obtaining a σ' satisfying properties 1–4 above except that in properties 1 and 3 we add the restriction $j \leqslant_r i$. Note that property 3 above and properties 2, 3 and 4 of evaluation rules together imply that if P is a value metavariable in the domain of σ', then $\sigma(P)$ and $\sigma'(P)$ are values.

Suppose that σ' has been built for all $i' < i$. Because of property 3 of evaluation rules, all metavariables of a_i are in the domain of σ'. We have $\sigma(a_i) \Rightarrow \sigma(b_i)$, and,

by Lemma 6.2, $\sigma(a_i)\,\hat{\eta}\,\sigma'(a_i)$, so by the outer induction hypothesis and Lemma 4.1, there is a d such that $\sigma'(a_i) \Rightarrow d$ and $\sigma(b_i)\,[\hat{\eta}]\,d$. Since b_i is a metavariable or a simple term schema, we can extend σ' so that $\sigma'(b_i) = d$. Note that by properties 2 and 4 of evaluation rules, b_i contains value metavariables if and only if b_i is itself a value metavariable P, and in this case $\sigma'(P) = d$.

We thus get a derivation of $\tau(\vec{s}') \Rightarrow \sigma'(b)$. If b is a metavariable, then by properties 4 and 5 of evaluation rules, b must be b_i for some $i \in I_r$, and so $\sigma(b)\,[\hat{\eta}]\,\sigma'(b)$ by property 4 of σ' above. If b is not a metavariable, then $\sigma(b)\,[\hat{\eta}]\,\sigma'(b)$ follows by Lemma 6.2. ∎

The following is an immediate consequence of Theorems 4.1 and 6.1.

COROLLARY 6.1. *If $\mathscr{E}$ is a structured evaluation system then $\leqslant$ is a precongruence.*

7. DIVERGENCE

We now turn to a simulation that incorporates a notion of divergence for structured evaluation systems. For this section, assume that $\mathscr{E}$ is a finitary structured evaluation system with rule set R.

A natural way to implement evaluation in $\mathscr{E}$ is with the following recursive procedure. To evaluate a term u, find a rule r with conclusion $a \Rightarrow b$ and premises $\{a_i \Rightarrow b_i\}_{1 \leqslant i \leqslant n}$ such that *a matches u* in the sense that there exists an instantiation σ such that $\sigma(a) = u$. Recursively evaluate $\sigma(a_1)$, getting v_1. If b_1 matches v_1, extend σ so that $\sigma(b_1) = v$, and continue in this way with the remaining premises, in order. If all of the matches succeed, then the result of evaluating u is $\sigma(b)$. Otherwise, find another rule whose conclusion has a left-hand side matching u, trying all such rules if necessary to produce a result. If there is no such rule, evaluation of u fails.

This procedure essentially does a sequential goal-directed search for a derivation of $u \Rightarrow v$ for some v. Intermediate stages of this search can be viewed as incomplete, or partial, derivations, and a diverging computation can be viewed as an infinite partial derivation. We formalize this as follows.

DEFINITION 7.1. A *partial derivation* is a finitely branching ordered tree of instances of rules from R such that for any node whose subtrees are $\nabla_1, ..., \nabla_m$ and whose rule instance has conclusion $a \Rightarrow b$ and premises $\{a_i \Rightarrow b_i\}_{1 \leqslant i \leqslant n}$, we have $m \leqslant n$, the left-hand side of the conclusion of ∇_m is a_m, and for $1 \leqslant i < m$, ∇_i is a derivation with conclusion $a_i \Rightarrow b_i$.

When appropriate, we apply the terminology for derivations to partial derivations. Also, for emphasis we will sometimes refer to derivations as *complete* derivations. A partial derivation will be called complete if it is a complete derivation.

DEFINITION 7.2. We say that a closed term u *may diverge*, and write $u\uparrow$, if there is a partial derivation for u that has an infinite branch; otherwise, we say u *must converge* and write $u\downarrow$.

DEFINITION 7.3. Define $\beta_\downarrow \subset T_0 \times T_0$ by $a\,\beta\,b$ if $a\downarrow$ exactly when $b\downarrow$. Define $\simeq = \simeq_{\beta_\downarrow}$.

We will refer to $\simeq$ simply as bisimulation. Before we can prove that bisimulation is a congruence, we need to prove some technical lemmas about divergence.

Since $\mathscr{E}$ is finitary, all complete derivations are finite trees, so we can define a *rule list* of a partial derivation ∇ to be the preorder listing of a tree T of rules such that ∇ can be obtained from T by applying instantiations to the nodes of T. Note that if ρ is a rule list of ∇ then ∇ has an infinite branch if and only if ρ is infinite.

Define $R(u)$ to the the set of all ρ such that ρ is a rule list of some partial derivation for u.

LEMMA 7.1. *For all closed u, u may diverge if and only if $R(u)$ is infinite.*

Proof. The forward direction is trivial.

Conversely, suppose $R(u)$ is infinite. We show that $R(u)$ is the set of branches of a tree. Clearly if $\rho \in R(u)$ then every prefix of the sequence ρ is in $R(u)$. We also need to show that if ρ_∞ is an infinite sequence of rules such that every proper prefix of ρ_∞ is in $R(u)$, then $\rho_\infty \in R(u)$. To do this, we show that there exists a partial derivation ∇_∞ for u such that ρ_∞ is a rule list for ∇_∞.

For each prefix ρ of ρ_∞ there is a partial derivation ∇_ρ for u such that ρ is a rule list for ∇_ρ. Choose ρ maximizing the number of children of the root of ∇_ρ. Let i be the maximum number of children. Because of the properties of evaluation rules, if ρ' is a prefix of ρ_∞ which is longer than ρ, then the first $i-1$ subtrees of the root of $\nabla_{\rho'}$ exist and are the same as those of ∇_ρ, so if we remove the prefix ρ from ρ_∞, we get an infinite rule list such that every prefix is a rule list of a partial derivation for the left-hand side of the ith premise of the last step of ∇_ρ.

Repeatedly applying this construction gives us the required partial derivation ∇_∞.

Since the structured evaluation system is finitary, the tree $R(u)$ is finitely branching. The tree is also infinite, so it must have an infinite path, and so there is an infinite partial derivation for u. ∎

Because of the above lemma, we can make the following definition.

DEFINITION 7.4. If $u\downarrow$ then define $|u|$ to be the number of elements of $R(u)$.

LEMMA 7.2. *Let $\eta \subset T_0 \times T_0$ be a simulation. Suppose that $\eta \subset [\eta^{\circ\top}]^\top$ and that if $a\,\eta\,b$ and $a\downarrow$ then $b\downarrow$. If $u\downarrow$ and $u\,\hat{\eta}_0\,u'$ then $u'\downarrow$.*

Proof. The proof is similar to the proof of Theorem 6.1. We prove by induction on $|u|$ that if $u\downarrow$ and $u\ \hat{\eta}_0\ u'$ then $u'\downarrow$ and $u\ [\hat{\eta}^\top]^\top\ u'$. Write $u = \tau(\bar{s})$. There is a closed $\tau(\bar{s}')$ such that $\tau(\bar{s}')\ \eta\ u'$ and $\bar{s}\ \hat{\eta}\ \bar{s}'$. Suppose ∇ is a partial derivation for $\tau(\bar{s}')$, and suppose the last step in ∇ is $\sigma'(r)$. Let r have conclusion $a \Rightarrow b$ and premises $\{a_i \Rightarrow b_i\}_{1 \leqslant i \leqslant n}$. Let m be the number of children of the root of ∇, and let

$$I = \{i \mid 1 \leqslant i < m \vee (i = m \ \&\ \nabla\ \textit{complete})\}.$$

We build an instantiation σ such that

1. the domain of σ is the set of metavariables occurring in a or b_j for $j \in I$,
2. $\sigma(a) = \tau(\bar{s})$,
3. $\sigma(a_j) \Rightarrow \sigma(b_j)$ has a derivation for each $j \in I$, and
4. for every P in the domain of σ, $\sigma(P)\ \hat{\eta}\ \sigma'(P)$ and, furthermore, if P is a value metavariable then $\sigma(P)\ [\hat{\eta}]\ \sigma'(P)$.

We build σ by induction over I, at each stage $i \in I$ obtaining a σ satisfying the properties 1–4 above except that in properties 1 and 3 we add the restriction $j \leqslant i$. Suppose that σ has been built for all $i' < i$. There is a finite rule list ρ starting with r such that prepending ρ to any member of $R(\sigma(a_i))$ gives a member of $R(u)$, so $|\sigma(a_i)| < |u|$. Since $\sigma'(a_i) \Rightarrow \sigma'(b_i)$, by Lemma 6.2 and the induction hypothesis there is a d such that $\sigma(a_i) \Rightarrow d$ and $d\ [\hat{\eta}]\ \sigma'(b_i)$. Extend σ so that $\sigma(b_i) = d$.

Suppose that $u'\uparrow$. Take ∇ above to be an infinite partial derivation for u'. $|\sigma(a_m)| < |\tau(\bar{s})|$ so by Lemma 6.2 and the induction hypothesis, $\sigma'(a_m)\downarrow$, which is impossible since ∇ contains an infinite partial derivation for $\sigma'(a_m)$.

Now suppose that $u' \Rightarrow v'$. Since $\tau(\bar{s}')\ \eta\ u'$, by the assumption on η there is a c such that $\tau(\bar{s}') \Rightarrow c$ and $c\ [\eta^\circ]\ v'$. Now take ∇ to be a derivation of $\tau(\bar{s}') \Rightarrow c$. Our construction gives a derivation of $\sigma(a) \Rightarrow \sigma(b)$ with $\sigma(b)\ [\hat{\eta}]\ \sigma'(b) = c$. $c\ [\eta^\circ]\ v'$ so $\sigma(b)\ [\hat{\eta}]\ v'$. ∎

THEOREM 7.1. *If $\mathscr{E}$ is a structured evaluation system then $\simeq$ is a congruence.*

Proof. By Theorem 6.1, evaluation in $\mathscr{E}$ respects $\widehat{\simeq}$, so by Theorem 4.2, bisimulation is a congruence if $\widehat{\simeq}_0 \subset \beta_\downarrow$, and this follows by Lemma 7.2 and the second half of Lemma 3.3. ∎

8. RELATED WORK

The main ideas involved in proving congruence of the bisimulation relation $\sim$ (Definition 4.3) first appeared in Howe (1989). A version of the structured evaluation system formalism appeared in Howe (1991). Since Howe (1989), several extensions of the basic method have appeared in the literature.

Sands (1991) modifies the definition of $\leqslant$ to include a notion of "improvement" based on the structure of derivations, so that if $a \leqslant b$ then b can be viewed as a program that computes at least as efficiently as a. The precongruence proof for this preorder uses the same precongruence candidate and has the same inductive structure as the proof method described here.

Ong (1992) gives two extensions of our proof method. The first, mentioned earlier, is to call-by-value systems. Howe (1989) dealt only with lazy evaluation systems. As pointed out in Ong (1992), closing substitutions in Howe (1989) were defined to substitute arbitrary closed terms for free variables. Because of this, in the call-by-value λ-calculus, the two terms $\lambda x.I$ and $\lambda x.(\lambda y.I)(x)$, where $I = \lambda x.x$, are not bisimilar, since if a diverging term is substituted for x, the body of the first term converges while the body of the second does not. Ong's solution, which we have incorporated here, is to distinguish variables used for call-by-value, and to modify the definition of closing substitution appropriately.

The second extension in Ong (1992) is to handle divergence. This work was done independently of our own work on divergence, and there are a number of differences. Ong uses a different framework in place of evaluation systems. His is closer to the λ-calculus; in particular, all values have interpretations as functions. He does not give a particular formalism for computation, but instead deals with "must converge" and "may converge" as abstract properties and proves a theorem characterizing when bisimulation is a congruence. One of the main applications of the theorem is to a concurrent λ-calculus.

Our results about parameterized bisimulation hold when divergence is taken to be that of Ong's concurrent λ-calculus. However, the calculus is not directly specifiable as a structured evaluation system. The problem is that we treat nondeterminism in our rules as erratic. For example, consider the non-deterministic choice operator $+$ whose evaluation rules are

$$\frac{a \Rightarrow c}{a + b \Rightarrow c} \qquad \frac{b \Rightarrow c}{a + b \Rightarrow c}$$

There are two ways to interpret these rules. One is with respect to a sequential evaluator, so that $a + b$ is guaranteed to converge only if both a and b are. The other is to view these rules as specifying parallel evaluation of the two alternatives, so that if either of a or b is guaranteed to have a value then so is $a + b$. It should be straightforward to at least make an *ad hoc* extension to our formalism so that parallel evaluation can be specified, but this has not been worked out.

Ong's framework has an analogue of our restriction to *finitary* structured evaluation systems. An interesting question is whether our proof method can be extended to deal

with a more general kind of divergence, such as the one derived from considering a set of evaluation rules as a *co*-inductive definition.

Another study of concurrent λ-calculus is Sangiorgi (1994). This paper deals with languages where *reduction* is specified using a formalism similar to the *tyft* format of Groote and Vaandrager (1992). The paper leaves open the question of whether bisimulation is always congruence for this formalism. In Howe (1995), we use a slight modification of our method to answer this question in the affirmative.

The *tyft* format of Groote and Vaandrager (1992) for specifying state transition systems has variable occurrence restrictions similar to those in structured evaluation systems. There is a proof of congruence of bisimulation for this format, but the proof method does not seem to extend to our setting. See Section 3 for more on this.

Bloom (1990) defines an *observation calculus* and proves a general congruence theorem for it by reasoning about program contexts. The calculus requires that evaluation results be constants, and does not allow parts of intermediate results of evaluations to be substituted in one another.

Gordon (1994) extends our method to typed functional languages, and Ritter and Pitts (1995) extend it to languages with state.

ACKNOWLEDGMENTS

We owe special thanks to Stuart Allen and several anonymous referees for their many valuable comments on the presentation of this work. In particular, one of the referees suggested a substantial simplification of the proofs in Section 7.

Received April 8, 1992; final manuscript received June 6, 1995

REFERENCES

Abramsky, S. (1990), The lazy lambda calculus, *in* "Research Topics in Functional Programming" (D. A. Turner, Ed.), pp. 65–116, Addison–Wesley, Reading, MA.

Aczel, P. (1978), A general CR Theorem, preprint, Univ. of Manchester.

Berry, G. (1981), "Some Syntactic and Categorical Constructions of Lambda-Calculus Models," Technical Report 80, INRIA.

Bloom, B. (1990), Can LCF be topped? Flat lattice models of typed λ-calculus, *Inform. and Comput*. **87**, 264–301.

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

Groote, J. F., and Vaandrager, F. (1992), Structured operational semantics and bisimulation as a congruence, *Inform. and Comput*. **100**, 202–260.

Howe, D. J. (1989), Equality in lazy computation systems, *in* "Proceedings of the Fourth Annual Symposium on Logic in Computer Science," pp. 198–203, IEEE Computer Society, Los Alamitos, CA.

Howe, D. J. (1991), On computational open-endedness in Martin-Löf's type theory, *in* "Proceedings of the Sixth Annual Symposium on Logic in Computer Science," pp. 162–172, IEEE Computer Society, Los Alamitos, CA.

Howe, D. J. (1995), "A Note on Proving Congruence of Bisimulation in a Generalized Lambda Calculus," unpublished technical memo, AT&T Bell Laboratories.

Jagadeesan, R. (1991), "Investigations into Abstraction and Concurrency," Ph.D. thesis, Cornell Univ.

Kahn, G. (1987), Natural semantics, *in* "Proceedings of the Symposium on Theoretical Aspects of Computer Software," Lecture Notes in Computer Science, Vol. 247, p. 22–39, Springer-Verlag, Berlin/New York.

Klop, J. W. (1980), Combinatory reduction systems, "Mathematical Centre Tracts," Vol. 127, CWI, Amsterdam.

Ong, L. (1992), The concurrent lambda calculus I: A general pre-congruence theorem for applicative bisimulation, *in* "Proceedings on Seminars on Parallel Programming Systems," pp. 139–164, Department of Information Systems and Computer Science, National Univ. of Singapore.

Plotkin, G. (1981), "A Structural Approach to Operational Semantics," Technical Report, Computer Science Department, Aarhus Univ.

Ritter, E., and Pitts, A. M. (1995), A fully abstract translation between a λ-calculus with reference types and Standard ML, *in* "Proceedings of the Second International Conference on Typed Lambda Calculi and Applications," Lecture Notes in Computer Science, Vol. 902, pp. 397–413, Springer-Verlag, Berlin/New York.

Sands, D. (1991), Operational theories of improvement in functional languages, *in* "Proceedings of the Glasgow Functional Programming Workshop," pp. 298–311, Springer Workshops in Computing, Springer-Verlag, Berlin/New York

Sangiorgi, D. (1992), The lazy lambda calculus in a concurrency scenario, *Inform. and Comput*. **111**, 120–153.

Talcott, C. (1985), "The Essence of Rum: A Theory of the Intensional and Extensional Aspects of Lisp-Type Computation," Ph.D. thesis, Stanford Univ.