5

Howe's method for higher-order languages

ANDREW PITTS

5.1 Introduction

A fundamental requirement for any notion of equality between programs is that it be a *congruence*, in other words an equivalence relation that is compatible with the various syntactical constructs of the programming language. Being an equivalence relation, in particular being transitive, allows one to use a familiar technique of equational reasoning, namely the deduction of an equality $P \simeq P'$ via a chain of intermediate equalities, $P \simeq P_1 \simeq \cdots \simeq P_n \simeq P'$. Being compatible enables the use of an even more characteristic technique of equational reasoning, substitution of equals for equals, whereby an equality between compound phrases, $C[P] \simeq C[P']$, is deduced from an equality between sub-phrases, $P \simeq P'$. If one regards the meaning of a program to be its $\simeq$-equivalence class, then compatibility says that this semantics is *compositional* – the meaning of a compound phrase is a function of the meanings of its constituent sub-phrases.

This book is concerned with coinductively defined notions of program equality based on the notion of bisimulation. For these, the property of being an equivalence relation is easy to establish, whereas the property of compatibility can sometimes be difficult to prove and indeed in some cases is false.[1] This is particularly the case for languages involving *higher-order* features, that is, ones permitting functions and processes to be data that can be manipulated by functions and processes. For example, the language might feature parameterised programs $P(x)$ where the parameter x can be instantiated not with simple data (booleans, numbers, names, finite trees, ...), but with other programs Q. In

[1] The classic example is the failure of weak bisimilarity to be preserved by summation in Milner's CCS [Mil89, section 5.3].

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

this case, although it is usually easy to see that a bisimilarity $\simeq$ satisfies

$$\forall Q.\ P(x) \simeq P'(x) \ \Rightarrow\ P(Q) \simeq P'(Q),$$

for compatibility of $\simeq$ we have to establish the stronger property

$$\forall Q, Q'.\ P(x) \simeq P'(x) \ \wedge\ Q \simeq Q' \ \Rightarrow\ P(Q) \simeq P'(Q').$$

This is often hard to prove directly from the definition of $\simeq$.

In this chapter we present a method for establishing such congruence properties of coinductively defined program equalities, due originally to Howe [How89]. It is one that has proved to be particularly effective in the presence of higher-order functions. Howe's method was originally developed partly to give a direct proof that Abramsky's notion of *applicative bisimilarity* is a congruence for his 'lazy' λ-calculus [Abr90]. The latter consists of the terms of the untyped λ-calculus equipped with a non-strict, or call-by-name evaluation strategy.[2] Abramsky's original proof of congruence is via a denotational semantics of the language,[3] whereas Howe's method provides a more direct route to this purely syntactic result, just making use of the language's operational semantics. Such syntactical methods can often be very brittle: small changes to the language's syntax or operational semantics may break the method. This has proved not to be the case for Howe's method; essentially the same technique has been applied, first by Howe and then by others, to quite a variety of higher-order languages, both typed and untyped, featuring both sequential functions and concurrent processes, and with a variety of different operational semantics. Although far from being a universal panacea for proofs of congruence, Howe's method certainly deserves the space we give it in this book.

Chapter outline Howe's definitive account of his method [How96] uses a general framework that can be specialised to a number of different functional languages and evaluation strategies. Here we prefer to explain the method by giving some concrete examples of it in action. To see the wood from the trees, the examples are as syntactically simple as possible. In fact we use applicative similarity and bisimilarity for the untyped λ-calculus with a call-by-value evaluation strategy as the running example; these notions are explained in Sections 5.2 and 5.3. Then in Section 5.4 we see how far we can get with

[2] The terminology 'lazy' is now more commonly used as a synonym for a call-by-need evaluation strategy (as used by the Haskell functional programming language, for example) rather than for the call-by-name strategy.

[3] It is a corollary of his 'domain theory in logical form' [Abr91]; see [Pit97a] for an alternative denotational proof making use of logical relations.

a proof of the congruence property for applicative similarity directly from its coinductive definition. This motivates the use of Howe's 'precongruence candidate' construction, which is explained in Section 5.5 and used to prove that applicative similarity for the call-by-value λ-calculus is a precongruence and hence that applicative bisimilarity is a congruence.

Section 5.6 explains an important consequence of this congruence result, namely that applicative bisimilarity for this language coincides with a Morris-style [Mor69] contextual equivalence, in which terms of the language are identified if they behave the same in all contexts (with respect to some fixed notion of observation). The coincidence of these two notions of program equality depends upon the deterministic nature of evaluation in call-by-value λ-calculus. So in Section 5.7 we consider what happens if we add a non-deterministic feature; we use 'erratic' binary choice as our example. As Howe observed, in such a non-deterministic setting his precongruence candidate construction can still be used to prove congruence results for bisimilarities, via a finesse involving transitive closure.

In Section 5.8 we illustrate a perhaps less well-known application of Howe's method, namely to proving 'context lemmas' [Mil77] for contextual equivalences. In general, such lemmas tell us that two terms behave the same in all contexts, that is, are contextually equivalent iff they behave the same in some restricted collection of contexts. Context lemmas are a useful way of establishing certain basic properties of contextual equivalence – for example, that terms with the same reduction behaviour are contextually equivalent. We consider the restriction to contexts that are 'Uses of Closed Instantiations' and the associated 'CIU-equivalence' of Mason and Talcott [MT91]. Specifically, we use Howe's method to show that for call-by-value λ-calculus with erratic choice, and for the observational scenario based on 'may-terminate', CIU-equivalence is a congruence and (hence) coincides with contextual equivalence. This result is representative of the way Howe's method can be used to establish context lemmas for contextual equivalences.

Section 5.9 briefly considers the call-by-name version of all these results. Section 5.10 summarises what is involved in Howe's method and then in Section 5.11 we outline some other applications of it that occur in the literature and assess its limitations.

Table 5.1 gives the notation we use for the various types of preorder and equivalence considered in this chapter.

Prerequisites The reader is assumed to have some familiarity with the basic notions of λ-calculus and its use in programming language semantics; see [Pie02, chapter 5], for example. The first few paragraphs of the next section summarise what we need of this material.

Table 5.1. *Preorders and equivalences used in this chapter.*

		call-by-value	call-by-name
applicative	similarity	$\lesssim_{\mathrm{v}}$	$\lesssim_{\mathrm{n}}$
	bisimilarity	$\simeq_{\mathrm{v}}$	$\simeq_{\mathrm{n}}$
contextual	preorder	$\leq_{\mathrm{v}}$	$\leq_{\mathrm{n}}$
	equivalence	$=_{\mathrm{v}}$	$=_{\mathrm{n}}$
CIU	preorder	$\leq^{\mathrm{ciu}}_{\mathrm{v}}$	$\leq^{\mathrm{ciu}}_{\mathrm{n}}$
	equivalence	$=^{\mathrm{ciu}}_{\mathrm{v}}$	$=^{\mathrm{ciu}}_{\mathrm{n}}$

5.2 Call-by-value λ-calculus

We take the set Λ of λ-*terms* and the subset $V \subseteq \Lambda$ of λ-*values* to be given by the following grammar

$$\begin{aligned} e \in \Lambda &::= v \mid e\, e \\ v \in V &::= x \mid \lambda x.\, e \end{aligned} \tag{5.1}$$

where x ranges over a fixed, countably infinite set *Var* of *variables*. An occurrence of a variable x in a λ-term e is a *free occurrence* if it is not within the scope of any λx. The *substitution* of e for all free occurrences of x in e' is denoted $e'\{e/x\}$; more generally a simultaneous substitution is written $e'\{e_1, \ldots, e_n/x_1, \ldots, x_n\}$. As usual when making a substitution $e'\{e/x\}$, we do not want free occurrences in e of a variable x' to become bound by occurrences of $\lambda x'$ in e'; in other words, substitution should be 'capture-avoiding'. To achieve this we identify the syntax trees generated by the above grammar up to α-*equivalence* of λ-bound variables: thus the elements of Λ and V are really α-equivalence classes of abstract syntax trees, but we refer to a class by naming one of its representatives. For example if x, y, z are distinct variables, then $\lambda y.\, x$ and $\lambda z.\, x$ are equal λ-terms (being α-equivalent) and $(\lambda y.\, x)\{y/x\}$ is $\lambda z.\, y$, not $\lambda y.\, y$.

The finite set of variables occurring free in $e \in \Lambda$ is denoted $\mathsf{fv}(e)$. If $\overline{x} \in \wp_{\mathrm{fin}}(\mathit{Var})$ is a finite set of variables, we write

$$\begin{aligned} \Lambda(\overline{x}) &\stackrel{\mathrm{def}}{=} \{e \in \Lambda \mid \mathsf{fv}(e) \subseteq \overline{x}\} \\ V(\overline{x}) &\stackrel{\mathrm{def}}{=} \{v \in V \mid \mathsf{fv}(v) \subseteq \overline{x}\} \end{aligned} \tag{5.2}$$

for the subsets of λ-terms and λ-values whose free variables are all in $\overline{x}$. Thus $\Lambda(\emptyset)$ is the subset of *closed* λ-terms and $V(\emptyset)$ is the set of closed λ-abstractions.

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

Call-by-value evaluation of closed λ-terms is the binary relation $\Downarrow_{\mathrm{v}} \subseteq \Lambda(\emptyset) \times V(\emptyset)$ inductively defined by the following two rules.

$$\frac{}{v \Downarrow_{\mathrm{v}} v} \tag{Val}$$

$$\frac{e_1 \Downarrow_{\mathrm{v}} \lambda x.\, e \qquad e_2 \Downarrow_{\mathrm{v}} v \qquad e\{v/x\} \Downarrow_{\mathrm{v}} v'}{e_1\, e_2 \Downarrow_{\mathrm{v}} v'}. \tag{Cbv}$$

Exercise 5.2.1 Consider the closed λ-values

$$\begin{aligned} \mathit{one} &\stackrel{\text{def}}{=} \lambda x.\, \lambda y.\, x\ y \\ \mathit{two} &\stackrel{\text{def}}{=} \lambda x.\, \lambda y.\, x\ (x\ y) \\ \mathit{succ} &\stackrel{\text{def}}{=} \lambda n.\, \lambda x.\, \lambda y.\, x\ (n\ x\ y). \end{aligned}$$

(These are the first two 'Church numerals' and the successor function for Church numerals; see [Pie02, sect. 5.2], for example.) For all $u, v, w \in V(\emptyset)$, show that $\mathit{succ}\ \mathit{one}\ u\ v \Downarrow_{\mathrm{v}} w$ if and only if $\mathit{two}\ u\ v \Downarrow_{\mathrm{v}} w$. Is it the case that $\mathit{succ}\ \mathit{one} \Downarrow_{\mathrm{v}} \mathit{two}$ holds?

Exercise 5.2.2 Prove that evaluation is *deterministic*, that is, satisfies

$$e \Downarrow_{\mathrm{v}} v_1 \ \wedge\ e \Downarrow_{\mathrm{v}} v_2 \ \Rightarrow\ v_1 = v_2. \tag{5.3}$$

Show that it is also partial, in the sense that for some closed λ-term $e \in \Lambda(\emptyset)$ there is no v for which $e \Downarrow_{\mathrm{v}} v$ holds. [Hint: consider $e = (\lambda x.\, x\ x)(\lambda x.\, x\ x)$, for example.]

Call-by-value evaluation is sometimes called *strict evaluation*, in contrast with non-strict, or *call-by-name* evaluation. We will consider the latter in Section 5.9.

5.3 Applicative (bi)similarity for call-by-value λ-calculus

We wish to define an equivalence relation $\simeq \subseteq \Lambda(\emptyset) \times \Lambda(\emptyset)$ that equates two closed λ-terms $e_1, e_2 \in \Lambda(\emptyset)$ if they have the same behaviour with respect to call-by-value evaluation. But what does 'behaviour' mean in this case? It is too simplistic to require that e_1 and e_2 both evaluate to the same λ-value (or both diverge); for example we might hope that the λ-term *succ one* from Exercise 5.2.1 is behaviourally equivalent to the λ-value *two* even though it does not evaluate to it under call-by-value evaluation. To motivate the kind of behavioural equivalence we are going to consider, recall that a λ-abstraction $\lambda x.\, e$ is a notation for a *function*; and indeed via the operation of substitution it

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

determines a function on closed λ-values, mapping $v \in V(\emptyset)$ to $e\{v/x\} \in \Lambda(\emptyset)$. So if $e_1 \simeq e_2$ and $e_i \Downarrow_{\mathrm{v}} \lambda x_i.\, e_i'$ (for $i = 1, 2$), it seems reasonable to require that $\lambda x_1.\, e_1'$ and $\lambda x_2.\, e_2'$ determine the same functions $V(\emptyset) \to \Lambda(\emptyset)$ modulo $\simeq$, in the sense that $e_1'\{v/x_1\} \simeq e_2'\{v/x_2\}$ should hold for all $v \in V(\emptyset)$. This property of an equivalence relation $\simeq$ is not yet a definition of it, because of the circularity it involves. We can break that circularity by taking the *greatest*[4] relation with this property, in other words by making a coinductive definition.

Definition 5.3.1 (Applicative simulation) A relation $\mathcal{S} \subseteq \Lambda(\emptyset) \times \Lambda(\emptyset)$ is a (*call-by-value*) *applicative simulation* if for all $e_1, e_2 \in \Lambda(\emptyset)$, $e_1\ \mathcal{S}\ e_2$ implies

$$e_1 \Downarrow_{\mathrm{v}} \lambda x.\, e_1' \ \Rightarrow\ \exists e_2'.\ e_2 \Downarrow_{\mathrm{v}} \lambda x.\, e_2' \ \wedge\ \forall v \in V(\emptyset).\ e_1'\{v/x\}\ \mathcal{S}\ e_2'\{v/x\}. \tag{v-Sim}$$

It is an *applicative bisimulation* if both $\mathcal{S}$ and its reciprocal $\mathcal{S}^{op} = \{(e, e') \mid e'\ \mathcal{S}\ e\}$ are applicative simulations. Note that the union of a family of applicative (bi)simulations is another such. So there is a largest applicative simulation, which we call *applicative similarity* and write as $\lesssim_{\mathrm{v}}$; and there is a largest applicative bisimulation, which we call *applicative bisimilarity* and write as $\simeq_{\mathrm{v}}$.

Example 5.3.2 It follows from Exercise 5.2.1 that

$$\{(\mathit{succ}\ \mathit{one}, \mathit{two})\} \cup \{(e, e) \mid e \in \Lambda(\emptyset)\}$$

is an applicative bisimulation. Hence *succ one* and *two* are applicatively bisimilar, $\mathit{succ}\ \mathit{one} \simeq_{\mathrm{v}} \mathit{two}$.

Definition 5.3.1 (and Theorem 5.6.5 below) extends smoothly to many applied λ-calculi. For example, it provides a useful tool for proving equivalences between pure functional programs involving algebraic data types; see [Gor95] and [Pit97b, section 3].

Applicative similarity is by definition the greatest post-fixed point for the monotone endofunction $F : \wp(\Lambda(\emptyset) \times \Lambda(\emptyset)) \to \wp(\Lambda(\emptyset) \times \Lambda(\emptyset))$ that maps $\mathcal{S}$ to

$$F(\mathcal{S}) \stackrel{\text{def}}{=} \{(e_1, e_2) \mid \forall x, e_1'.\ e_1 \Downarrow_{\mathrm{v}} \lambda x.\, e_1' \ \Rightarrow\ \exists e_2'.\ e_2 \Downarrow_{\mathrm{v}} \lambda x.\, e_2' \wedge \forall v.\ e_1'\{v/x\}\ \mathcal{S}\ e_2'\{v/x\}\}.$$

[4] Taking the *least* such relation does not lead anywhere interesting, since that least relation is empty.

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

By the Knaster–Tarski fixed-point theorem [San12, theorem 2.3.21], it is also the greatest fixed point $\mathtt{gfp}(F)$ and hence we have:

$$\begin{aligned} & e_1 \lesssim_{\mathrm{v}} e_2 \\ & \quad \Leftrightarrow \forall x, e_1'.\, e_1 \Downarrow_{\mathrm{v}} \lambda x.\, e_1' \Rightarrow \exists e_2'.\, e_2 \Downarrow_{\mathrm{v}} \lambda x.\, e_2' \wedge \forall v.\, e_1'\{v/x\} \lesssim_{\mathrm{v}} e_2'\{v/x\}. \end{aligned} \tag{5.4}$$

Similarly, applicative bisimulation satisfies:

$$\begin{aligned} & e_1 \simeq_{\mathrm{v}} e_2 \\ & \quad \Leftrightarrow \forall x, e_1'.\ e_1 \Downarrow_{\mathrm{v}} \lambda x.\, e_1' \Rightarrow \exists e_2'.\ e_2 \Downarrow_{\mathrm{v}} \lambda x.\, e_2' \wedge \forall v.\ e_1'\{v/x\} \simeq_{\mathrm{v}} e_2'\{v/x\} \\ & \qquad \wedge \forall x, e_2'.\ e_2 \Downarrow_{\mathrm{v}} \lambda x.\, e_2' \Rightarrow \exists e_1'.\ e_1 \Downarrow_{\mathrm{v}} \lambda x.\, e_1' \wedge \forall v.\ e_1'\{v/x\} \simeq_{\mathrm{v}} e_2'\{v/x\}. \end{aligned} \tag{5.5}$$

Exercise 5.3.3 Show that the identity relation $\{(e, e) \mid e \in \Lambda(\emptyset)\}$ is an applicative bisimulation. Show that if $\mathcal{S}_1$ and $\mathcal{S}_2$ are applicative (bi)simulations, then so is their composition $\mathcal{S}_1 \circ \mathcal{S}_2 = \{(e_1, e_3) \mid \exists e_2.\ e_1\ \mathcal{S}_1\ e_2\ \wedge\ e_2\ \mathcal{S}_2\ e_3\}$. Deduce that $\lesssim_{\mathrm{v}}$ is a preorder (reflexive and transitive relation) and that $\simeq_{\mathrm{v}}$ is an equivalence relation (reflexive, symmetric and transitive relation).

Because evaluation is deterministic (Exercise 5.2.2), applicative bisimulation is the equivalence relation generated by the preorder $\lesssim_{\mathrm{v}}$:

Proposition 5.3.4 *For all* $e_1, e_2 \in \Lambda(\emptyset)$, $e_1 \simeq_{\mathrm{v}} e_2$ *holds iff* $e_1 \lesssim_{\mathrm{v}} e_2$ *and* $e_2 \lesssim_{\mathrm{v}} e_1$.

Proof Since $\simeq_{\mathrm{v}}$ and $\simeq_{\mathrm{v}}^{op}$ are both applicative bisimulations, and hence both applicative simulations, they are contained in the greatest one, $\lesssim_{\mathrm{v}}$. Thus $e_1 \simeq_{\mathrm{v}} e_2$ implies $e_1 \lesssim_{\mathrm{v}} e_2$ and $e_2 \lesssim_{\mathrm{v}} e_1$.

Conversely, since $\lesssim_{\mathrm{v}}$ satisfies property (v-Sim) and $\Downarrow_{\mathrm{v}}$ satisfies (5.3), it follows that $\lesssim_{\mathrm{v}} \cap \lesssim_{\mathrm{v}}^{op}$ is an applicative bisimulation and hence is contained in $\simeq_{\mathrm{v}}$. Thus $e_1 \lesssim_{\mathrm{v}} e_2$ and $e_2 \lesssim_{\mathrm{v}} e_1$ together imply $e_1 \simeq_{\mathrm{v}} e_2$. □

Exercise 5.3.5 Show that

$$(\forall v \in V(\emptyset).\ e_1 \Downarrow_{\mathrm{v}} v\ \Rightarrow\ e_2 \Downarrow_{\mathrm{v}} v)\ \Rightarrow\ e_1 \lesssim_{\mathrm{v}} e_2. \tag{5.6}$$

[Hint: show that the union of $\{(e_1, e_2) \mid \forall v.\ e_1 \Downarrow_{\mathrm{v}} v\ \Rightarrow\ e_2 \Downarrow_{\mathrm{v}} v\}$ with the identity relation is an applicative simulation.]

Deduce that β_{v}-equivalence is contained in applicative bisimilarity, that is, $(\lambda x.\, e)v \simeq_{\mathrm{v}} e\{v/x\}$.

Exercise 5.3.6 Show by example that η-equivalence is not contained in applicative bisimilarity, in other words that $e \simeq_{\mathrm{v}} \lambda x.\, e\, x$ (where $x \notin \mathsf{fv}(e)$) does not always hold. What happens if e is a λ-value?

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

Exercise 5.3.7 Show that $\lambda x.\, e_1 \lesssim_{\mathrm{v}} \lambda x.\, e_2$ holds iff $e_1\{v/x\} \lesssim_{\mathrm{v}} e_2\{v/x\}$ holds for all $v \in V(\emptyset)$.

5.4 Congruence

We noted in Exercise 5.3.3 that applicative bisimilarity is an equivalence relation. To qualify as a reasonable notion of semantic equality for λ-terms, $\simeq_{\mathrm{v}}$ should not only be an equivalence relation, but also be a congruence; in other words it should also respect the way λ-terms are constructed. There are two such constructions: formation of application terms $(e_1, e_2 \mapsto e_1\, e_2)$ and formation of λ-abstractions $(x, e \mapsto \lambda x.\, e)$. The second is a variable-binding operation and to make sense of the statement that it respects applicative bisimilarity we first have to extend $\simeq_{\mathrm{v}}$ to a binary relation between all λ-terms, open as well as closed. To do this we will regard the free variables in a λ-term as implicitly λ-bound and use the property of $\lesssim_{\mathrm{v}}$ noted in Exercise 5.3.7.

Definition 5.4.1 (Open extension of applicative (bi)similarity) Given a finite set of variables $\overline{x} = \{x_1, \ldots, x_n\} \in \wp_{\mathrm{fin}}(Var)$ and λ-terms $e, e' \in \Lambda(\overline{x})$, we write

$$\overline{x} \vdash e \lesssim_{\mathrm{v}} e' \tag{5.7}$$

if $e\{v_1, \ldots, v_n/x_1, \ldots, x_n\} \lesssim_{\mathrm{v}} e'\{v_1, \ldots, v_n/x_1, \ldots, x_n\}$ holds for all $v_1, \ldots, v_n \in V(\emptyset)$. Similarly

$$\overline{x} \vdash e \simeq_{\mathrm{v}} e' \tag{5.8}$$

means $e\{v_1, \ldots, v_n/x_1, \ldots, x_n\} \simeq_{\mathrm{v}} e'\{v_1, \ldots, v_n/x_1, \ldots, x_n\}$ holds for all $v_1, \ldots, v_n \in V(\emptyset)$.

Definition 5.4.2 (λ-term relations) The relations (5.7) and (5.8) are examples of what we call a *λ-term relation*, by which we mean a set $\mathcal{R}$ of triples $(\overline{x}, e, e')$ where $\overline{x} \in \wp_{\mathrm{fin}}(Var)$ and $e, e' \in \Lambda(\overline{x})$. We will use mixfix notation and write $\overline{x} \vdash e\ \mathcal{R}\ e'$ to indicate that $(\overline{x}, e, e') \in \mathcal{R}$. We call a λ-term relation $\mathcal{R}$ *symmetric* if

$$\forall \overline{x} \in \wp_{\mathrm{fin}}(Var).\, \forall e, e' \in \Lambda(\overline{x}).\ \overline{x} \vdash e\ \mathcal{R}\ e' \ \Rightarrow\ \overline{x} \vdash e'\ \mathcal{R}\ e, \tag{Sym}$$

transitive if

$$\forall \overline{x} \in \wp_{\mathrm{fin}}(Var).\, \forall e, e', e'' \in \Lambda(\overline{x}).\ \overline{x} \vdash e\ \mathcal{R}\ e' \ \wedge\ \overline{x} \vdash e'\ \mathcal{R}\ e'' \ \Rightarrow\ \overline{x} \vdash e\ \mathcal{R}\ e'' \tag{Tra}$$

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

and *compatible* if

$$\forall \overline{x} \in \wp_{\text{fin}}(Var).\ x \in \overline{x} \ \Rightarrow\ \overline{x} \vdash x\ \mathcal{R}\ x \tag{Com1}$$

$$\forall \overline{x} \in \wp_{\text{fin}}(Var).\ \forall x \in Var - \overline{x}.\ \forall e, e' \in \Lambda(\overline{x} \cup \{x\}).\ \overline{x} \cup \{x\} \vdash e\ \mathcal{R}\ e' \ \Rightarrow\ \overline{x} \vdash \lambda x.e\ \mathcal{R}\ \lambda x.e' \tag{Com2}$$

$$\forall \overline{x} \in \wp_{\text{fin}}(Var).\ \forall e_1, e_1', e_2, e_2' \in \Lambda(\overline{x}).\ \overline{x} \vdash e_1\ \mathcal{R}\ e_1' \ \wedge\ \overline{x} \vdash e_2\ \mathcal{R}\ e_2' \ \Rightarrow\ \overline{x} \vdash e_1 e_2\ \mathcal{R}\ e_1' e_2'. \tag{Com3}$$

Property (Com1) is a special case of *reflexivity*:

$$\forall \overline{x} \in \wp_{\text{fin}}(Var).\ \forall e \in \Lambda(\overline{x}).\ \overline{x} \vdash e\ \mathcal{R}\ e. \tag{Ref}$$

Indeed, it is not hard to see that a compatible λ-term relation has to be reflexive. We say that $\mathcal{R}$ is a *precongruence* if it has the properties (Tra), (Com1), (Com2) and (Com3); and we call it a *congruence* if it also satisfies property (Sym).

Exercise 5.4.3 Prove that every compatible λ-term relation is reflexive. Deduce that property (Com3) implies

$$\forall \overline{x} \in \wp_{\text{fin}}(Var).\ \forall e_1, e_1', e_2 \in \Lambda(\overline{x}).\ \overline{x} \vdash e_1\ \mathcal{R}\ e_1' \ \Rightarrow\ \overline{x} \vdash e_1 e_2\ \mathcal{R}\ e_1' e_2 \tag{Com3L}$$

$$\forall \overline{x} \in \wp_{\text{fin}}(Var).\ \forall e_1, e_1', e_2 \in \Lambda(\overline{x}).\ \overline{x} \vdash e_2\ \mathcal{R}\ e_2' \ \Rightarrow\ \overline{x} \vdash e_1 e_2\ \mathcal{R}\ e_1 e_2'. \tag{Com3R}$$

Show that if $\mathcal{R}$ is transitive, then (Com3L) and (Com3R) together imply (Com3).

Remark 5.4.4 (Motivating Howe's method) We will eventually prove that the open extension of applicative similarity is a precongruence relation (Theorem 5.5.5). It is illuminating to see how far we can get with a direct proof of this before we have to introduce the machinery of Howe's method. Combining Exercise 5.3.3 with the way the open extension is defined from the relation on closed λ-terms in Definition 5.4.1, it follows that properties (Tra), (Ref), and hence also (Com1), hold when $\mathcal{R}$ is $\lesssim_{\text{v}}$. In the same way, the property in Exercise 5.3.7 implies (Com2). So to complete the proof that $\lesssim_{\text{v}}$ is a precongruence, we just have to prove that it has property (Com3); and for this it is sufficient to prove the special case when $\overline{x} = \emptyset$, that is, to prove

$$\forall e_1, e_1', e_2, e_2' \in \Lambda(\emptyset).\ e_1 \lesssim_{\text{v}} e_1' \ \wedge\ e_2 \lesssim_{\text{v}} e_2' \ \Rightarrow\ e_1 e_2 \lesssim_{\text{v}} e_1' e_2'. \tag{$*$}$$

Since we know that $\lesssim_{\text{v}}$ is a preorder, proving $(*)$ is equivalent to proving the following two special cases (cf. Exercise 5.4.3).

$$\forall e_1, e_1' \in \Lambda(\emptyset).\ e_1 \lesssim_{\text{v}} e_1' \ \Rightarrow\ \forall e_2 \in \Lambda(\emptyset).\ e_1 e_2 \lesssim_{\text{v}} e_1' e_2 \tag{$*_1$}$$

$$\forall e_2, e_2' \in \Lambda(\emptyset).\ e_2 \lesssim_{\text{v}} e_2' \ \Rightarrow\ \forall e_1 \in \Lambda(\emptyset).\ e_1 e_2 \lesssim_{\text{v}} e_1 e_2'. \tag{$*_2$}$$

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

Given the coinductive definition of $\lesssim_{\mathrm{v}}$ (Definition 5.3.1), an obvious strategy for proving these properties is to show that

$$\mathcal{S}_1 \stackrel{\text{def}}{=} \{(e_1\, e_2, e_1'\, e_2) \mid e_1 \lesssim_{\mathrm{v}} e_1' \wedge e_2 \in \Lambda(\emptyset)\}$$
$$\mathcal{S}_2 \stackrel{\text{def}}{=} \{(e_1\, e_2, e_1\, e_2') \mid e_1 \in \Lambda(\emptyset) \wedge e_2 \lesssim_{\mathrm{v}} e_2'\}$$

are contained in applicative simulations; for then $\mathcal{S}_i$ is contained in the largest such, $\lesssim_{\mathrm{v}}$, which gives $(*_i)$. We leave the proof of this for $\mathcal{S}_1$ as a straightforward exercise (Exercise 5.4.5). Proving $(*_2)$ by the same method as in that exercise is not so easy. Let us see why. It would suffice to show that

$$e_2 \lesssim_{\mathrm{v}} e_2' \tag{5.9}$$
$$e_1\, e_2 \Downarrow_{\mathrm{v}} \lambda x.\, e \tag{5.10}$$

implies

$$\exists e'.\ e_1\, e_2' \Downarrow_{\mathrm{v}} \lambda x.\, e' \wedge \forall v \in V(\emptyset).\ e\{v/x\} \lesssim_{\mathrm{v}} e'\{v/x\}. \tag{$\dagger$}$$

The syntax-directed nature of the rules (Val) and (Cbv) inductively defining $\Downarrow_{\mathrm{v}}$ mean that (5.10) holds because for some e_3 and e_4 we have

$$e_1 \Downarrow_{\mathrm{v}} \lambda x.\, e_3 \tag{5.11}$$
$$e_2 \Downarrow_{\mathrm{v}} \lambda x.\, e_4 \tag{5.12}$$
$$e_3\{\lambda x.\, e_4/x\} \Downarrow_{\mathrm{v}} \lambda x.\, e. \tag{5.13}$$

Then since $\lesssim_{\mathrm{v}}$ is an applicative simulation, from (5.9) and (5.12) we conclude that there is some e_4' with

$$e_2' \Downarrow_{\mathrm{v}} \lambda x.\, e_4' \tag{5.14}$$

and $\forall v.\ e_4\{v/x\} \lesssim_{\mathrm{v}} e_4'\{v/x\}$, which by Exercise 5.3.7 implies

$$\lambda x.\, e_4 \lesssim_{\mathrm{v}} \lambda x.\, e_4'. \tag{5.15}$$

So to prove $(\dagger)$ we just have to show that

$$\exists e'.\ e_3\{\lambda x.\, e_4'/x\} \Downarrow_{\mathrm{v}} \lambda x.\, e' \wedge \forall v.\ e\{v/x\} \lesssim_{\mathrm{v}} e'\{v/x\}. \tag{$\dagger\dagger$}$$

In view of (5.13) and the fact that $\lesssim_{\mathrm{v}}$ is an applicative simulation, to prove $(\dagger\dagger)$ we just need to show that $e_3\{\lambda x.\, e_4/x\} \lesssim_{\mathrm{v}} e_3\{\lambda x.\, e_4'/x\}$. This would follow from (5.15) if we knew that applicative similarity has the following substitution property:

$$\forall \overline{x} \in \wp_{\mathrm{fin}}(Var).\, \forall x \in Var - \overline{x}.\, \forall e \in \Lambda(\overline{x} \cup \{x\}).\, \forall v, v' \in V(\overline{x}).$$
$$\overline{x} \vdash v \lesssim_{\mathrm{v}} v' \ \Rightarrow\ \overline{x} \vdash e\{v/x\} \lesssim_{\mathrm{v}} e\{v'/x\}. \tag{5.16}$$

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

Unfortunately the property $(*_2)$ that we are trying to prove is very similar to this property (consider taking e to be $e_1\, x$) and we are stuck. To get unstuck, in the next section we use Howe's method of constructing a relation $(\lesssim_{\mathrm{v}})^H$ out of $\lesssim_{\mathrm{v}}$ that has property (5.16) by construction and which can be proved equal to $\lesssim_{\mathrm{v}}$ via an induction over evaluation $\Downarrow_{\mathrm{v}}$ (the key Lemma 5.5.4).

Exercise 5.4.5 Prove property $(*_1)$ by showing that

$$\{(e_1\, e_2, e_1'\, e_2) \mid e_1 \lesssim_{\mathrm{v}} e_1' \ \wedge\ e_2 \in \Lambda(\emptyset)\} \cup \lesssim_{\mathrm{v}}$$

is an applicative simulation.

5.5 Howe's construction

Given a λ-term relation $\mathcal{R}$, the λ-term relation $\mathcal{R}^H$ is inductively defined from $\mathcal{R}$ by the following rules. It is an instance of what Howe calls the 'precongruence candidate' [How96, sect. 3].

$$\frac{\overline{x} \vdash x\ \mathcal{R}\ e}{\overline{x} \vdash x\ \mathcal{R}^H\ e} \tag{How1}$$

$$\frac{\overline{x} \cup \{x\} \vdash e_1\ \mathcal{R}^H\ e_1' \qquad \overline{x} \vdash \lambda x.e_1'\ \mathcal{R}\ e_2 \qquad x \notin \overline{x}}{\overline{x} \vdash \lambda x.e_1\ \mathcal{R}^H\ e_2} \tag{How2}$$

$$\frac{\overline{x} \vdash e_1\ \mathcal{R}^H\ e_1' \qquad \overline{x} \vdash e_2\ \mathcal{R}^H\ e_2' \qquad \overline{x} \vdash e_1'\, e_2'\ \mathcal{R}\ e_3}{\overline{x} \vdash e_1\, e_2\ \mathcal{R}^H\ e_3}. \tag{How3}$$

We will need the following general properties of $(\_)^H$ whose proof we leave as an exercise.

Lemma 5.5.1

(i) *If $\mathcal{R}$ is reflexive, then $\mathcal{R}^H$ is compatible.*
(ii) *If $\mathcal{R}$ is transitive, then $\overline{x} \vdash e_1\ \mathcal{R}^H\ e_2$ and $\overline{x} \vdash e_2\ \mathcal{R}\ e_3$ imply $\overline{x} \vdash e_1\ \mathcal{R}^H\ e_3$.*
(iii) *If $\mathcal{R}$ is reflexive and transitive, then $\overline{x} \vdash e_1\ \mathcal{R}\ e_2$ implies $\overline{x} \vdash e_1\ \mathcal{R}^H\ e_2$.*

□

Definition 5.5.2 (Substitutivity and closure under substitution)
A λ-term relation $\mathcal{R}$ is called (*value*) *substitutive* if for all $\overline{x} \in \wp_{\mathrm{fin}}(Var)$, $x \in Var - \overline{x}$, $e, e' \in \Lambda(\overline{x} \cup \{x\})$ and $v, v' \in V(\overline{x})$

$$\overline{x} \cup \{x\} \vdash e\ \mathcal{R}\ e' \ \wedge\ \overline{x} \vdash v\ \mathcal{R}\ v' \ \Rightarrow\ \overline{x} \vdash e\{v/x\}\ \mathcal{R}\ e'\{v'/x\}. \tag{Sub}$$

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

Note that if $\mathcal{R}$ is also reflexive, then this implies

$$\overline{x} \cup \{x\} \vdash e \;\mathcal{R}\; e' \;\wedge\; v \in V(\overline{x}) \;\Rightarrow\; \overline{x} \vdash e\{v/x\} \;\mathcal{R}\; e'\{v/x\}. \tag{Cus}$$

We say that $\mathcal{R}$ is *closed under value-substitution* if it satisfies (Cus).

Note that because of the way the open extension of applicative similarity and bisimilarity are defined (Definition 5.4.1), evidently they are closed under value-substitution. It is less clear that they are substitutive; this will follow from the coincidence of $\lesssim_{\mathrm{v}}$ with $(\lesssim_{\mathrm{v}})^H$ that we prove below, because of the following result.

Lemma 5.5.3 *If $\mathcal{R}$ is reflexive, transitive and closed under value-substitution, then $\mathcal{R}^H$ is substitutive and hence also closed under value-substitution.*

Proof Property (Sub) for $\mathcal{R}^H$ follows from property (Cus) for $\mathcal{R}$ by induction on the derivation of $\overline{x} \cup \{x\} \vdash e \;\mathcal{R}^H\; e'$ from the rules (How1)–(How3), using Lemma 5.5.1(iii) in case e is a variable. □

Specialising Lemmas 5.5.1 and 5.5.3 to the case $\mathcal{R} = \lesssim_{\mathrm{v}}$, we now give the key property needed to show that $(\lesssim_{\mathrm{v}})^H$ is in fact equal to $\lesssim_{\mathrm{v}}$.

Lemma 5.5.4 (Key lemma, version 1) *If $e_1 \Downarrow_{\mathrm{v}} \lambda x.\, e$ and $\emptyset \vdash e_1 \;(\lesssim_{\mathrm{v}})^H\; e_2$, then $e_2 \Downarrow_{\mathrm{v}} \lambda x.\, e'$ holds for some e' satisfying $\{x\} \vdash e \;(\lesssim_{\mathrm{v}})^H\; e'$.*

Proof We show that

$$\begin{aligned}\mathcal{E} \stackrel{\text{def}}{=} \{(e_1, \lambda x.\, e) \;\mid\; & \forall e_2.\; \emptyset \vdash e_1 \;(\lesssim_{\mathrm{v}})^H\; e_2 \\ & \Rightarrow\; \exists e'.\; e_2 \Downarrow_{\mathrm{v}} \lambda x.\, e' \;\wedge\; \{x\} \vdash e \;(\lesssim_{\mathrm{v}})^H\; e'\}\end{aligned}$$

is closed under the two rules (Val) and (Cbv) inductively defining $\Downarrow_{\mathrm{v}}$.

Closure under (Val): When $e_1 = \lambda x.\, e$, for any e_2 if $\emptyset \vdash \lambda x.\, e \;(\lesssim_{\mathrm{v}})^H\; e_2$ holds, it must have been deduced by an application of rule (How2) to

$$\{x\} \vdash e \;(\lesssim_{\mathrm{v}})^H\; e_1' \tag{5.17}$$

$$\lambda x.\, e_1' \lesssim_{\mathrm{v}} e_2 \tag{5.18}$$

for some e_1'. Since $\lesssim_{\mathrm{v}}$ is an applicative simulation, from (5.18) it follows that $e_2 \Downarrow_{\mathrm{v}} \lambda x.\, e'$ holds for some e' with

$$\{x\} \vdash e_1' \lesssim_{\mathrm{v}} e'. \tag{5.19}$$

Applying Lemma 5.5.1(ii) to (5.17) and (5.19) gives $\{x\} \vdash e \;(\lesssim_{\mathrm{v}})^H\; e'$. So we do indeed have $(\lambda x.\, e, \lambda x.\, e) \in \mathcal{E}$.

Closure under (Cbv): Suppose

$$(e_1, \lambda x.\, e_2), (e_1', \lambda x.\, e_2'), (e_2\{\lambda x.\, e_2'/x\}, \lambda x.\, e_3) \in \mathcal{E}.$$

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

We have to show that $(e_1\, e_1', \lambda x.\, e_3) \in \mathcal{E}$. For any e, if $\emptyset \vdash e_1\, e_1'\, (\lesssim_{\mathrm{v}})^H\, e$ holds it must have been deduced by an application of rule (How3) to

$$\emptyset \vdash e_1\, (\lesssim_{\mathrm{v}})^H\, e_4 \tag{5.20}$$

$$\emptyset \vdash e_1'\, (\lesssim_{\mathrm{v}})^H\, e_4' \tag{5.21}$$

$$e_4\, e_4' \lesssim_{\mathrm{v}} e \tag{5.22}$$

for some e_4, e_4'. Since $(e_1, \lambda x.\, e_2), (e_1', \lambda x.\, e_2') \in \mathcal{E}$, it follows from (5.20) and (5.21) that

$$e_4 \Downarrow_{\mathrm{v}} \lambda x.\, e_5 \tag{5.23}$$

$$\{x\} \vdash e_2\, (\lesssim_{\mathrm{v}})^H\, e_5 \tag{5.24}$$

$$e_4' \Downarrow_{\mathrm{v}} \lambda x.\, e_5' \tag{5.25}$$

$$\{x\} \vdash e_2'\, (\lesssim_{\mathrm{v}})^H\, e_5'$$

hold for some e_5, e_5', and hence also that

$$\emptyset \vdash \lambda x.\, e_2'\, (\lesssim_{\mathrm{v}})^H\, \lambda x.\, e_5'. \tag{5.26}$$

Now we can apply the substitutivity property of $(\lesssim_{\mathrm{v}})^H$ (Lemma 5.5.3) to (5.24) and (5.26) to get $\emptyset \vdash e_2\{^{\lambda x.\, e_2'}/_x\}\, (\lesssim_{\mathrm{v}})^H\, e_5\{^{\lambda x.\, e_5'}/_x\}$. From this and the fact that $(e_2\{^{\lambda x.\, e_2'}/_x\}, \lambda x.\, e_3) \in \mathcal{E}$, it follows that for some e_6 it is the case that

$$e_5\{^{\lambda x.\, e_5'}/_x\} \Downarrow_{\mathrm{v}} \lambda x.\, e_6 \tag{5.27}$$

$$\{x\} \vdash e_3\, (\lesssim_{\mathrm{v}})^H\, e_6. \tag{5.28}$$

Applying rule (Cbv) to (5.23), (5.25) and (5.27), we have that $e_4\, e_4' \Downarrow_{\mathrm{v}} \lambda x.\, e_6$. Then since $\lesssim_{\mathrm{v}}$ is an applicative simulation, from this and (5.22) we get that $e \Downarrow_{\mathrm{v}} \lambda x.\, e'$ holds for some e' with

$$\{x\} \vdash e_6 \lesssim_{\mathrm{v}} e' \tag{5.29}$$

and hence also with $\{x\} \vdash e_3\, (\lesssim_{\mathrm{v}})^H\, e'$ (by Lemma 5.5.1(ii) on (5.28) and (5.29)). Therefore we do indeed have $(e_1\, e_1', \lambda x.\, e_3) \in \mathcal{E}$. □

Theorem 5.5.5 (Applicative bisimilarity is a congruence) *The open extension of applicative similarity is a precongruence relation and hence the open extension of applicative bisimilarity is a congruence relation.*

Proof By Lemma 5.5.3, $(\lesssim_{\mathrm{v}})^H$ is closed under value-substitution:

$$\{x\} \vdash e\, (\lesssim_{\mathrm{v}})^H\, e' \;\Rightarrow\; \forall v \in V(\emptyset).\; e\{^v/_x\}\, (\lesssim_{\mathrm{v}})^H\, e'\{^v/_x\}.$$

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

Therefore the key Lemma 5.5.4 implies that $(\lesssim_{\mathrm{v}})^H$ restricted to closed λ-terms is an applicative simulation. So it is contained in the largest one, $\lesssim_{\mathrm{v}}$:

$$\emptyset \vdash e_1 \; (\lesssim_{\mathrm{v}})^H \; e_2 \;\Rightarrow\; e_1 \lesssim_{\mathrm{v}} e_2.$$

Using closure of $(\lesssim_{\mathrm{v}})^H$ under value-substitution once again, this lifts to open terms:

$$\overline{x} \vdash e_1 \; (\lesssim_{\mathrm{v}})^H \; e_2 \;\Rightarrow\; \overline{x} \vdash e_1 \lesssim_{\mathrm{v}} e_2.$$

In view of Lemma 5.5.1(iii), this means that the λ-term relations $(\lesssim_{\mathrm{v}})^H$ and $\lesssim_{\mathrm{v}}$ are equal. Since $(\lesssim_{\mathrm{v}})^H$ is compatible by Lemma 5.5.1(i) and since we already know that $\lesssim_{\mathrm{v}}$ is transitive, it follows that $\lesssim_{\mathrm{v}} = (\lesssim_{\mathrm{v}})^H$ is a precongruence relation. □

5.6 Contextual equivalence

One of the important consequences of Theorem 5.5.5 is that applicative bisimilarity coincides with the standard notion of *contextual equivalence* for the call-by-value λ-calculus. A λ-term *context* C is a syntax tree with a unique[5] 'hole' $[\cdot]$

$$C \in \mathit{Con} ::= [\cdot] \;\big|\; \lambda x.\, C \;\big|\; C\, e \;\big|\; e\, C \tag{5.30}$$

and $C[e]$ denotes the λ-term that results from filling the hole with a λ-term e:

$$\begin{aligned} [\cdot][e] &= e \\ (\lambda x.\, C)[e] &= \lambda x.\, C[e] \\ (C\, e')[e] &= C[e]\, e' \\ (e'\, C)[e] &= e'\, C[e]. \end{aligned} \tag{5.31}$$

We also write $C[C']$ for the context resulting from replacing the occurrence of $[\cdot]$ in the syntax tree C by the tree C'.

Remark 5.6.1 (Contexts considered too concrete) Note that (5.31) is a 'capturing' form of substitution – free variables in e may become bound in $C[e]$; for example, $(\lambda x.\, -)[x] = \lambda x.\, x$. This means that it does not make sense to identify contexts up to renaming of λ-bound variables, as we do with λ-terms. For example, if x and x' are distinct variables, then $C_1 = \lambda x.\, [\cdot]$ and $C_2 = \lambda x'.\, [\cdot]$

[5] In the literature one also finds treatments that use contexts with finitely many (including zero) occurrences of the hole. While it does not affect the associated notion of contextual equivalence, the restriction to 'linear' contexts that we use here makes for some technical simplifications.

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

are distinct contexts that give different results when their hole is filled with x: $C_1[x] = \lambda x.\, x \neq \lambda x'.\, x = C_2[x]$. The concreteness of the above notion of 'context' forces us to take care with the actual names of bound variables. This might not seem so bad for the λ-calculus, because it is syntactically so simple. However it becomes increasingly irksome if one is producing fully formalised and machine-checked proofs, or if one is dealing with more complicated programming languages with more complex forms of binding. Several different remedies have been proposed for this problem of overly concrete representations of contexts. We explain one of them in Remark 5.6.7, but postpone discussing it until we have given a more-or-less classical development of contextual equivalence based on the above, rather too concrete notion of context.

Continuing the 'hygiene' of keeping track of free variables through use of the sets $\Lambda(\overline{x})$ indexed by sets $\overline{x}$ of variables, and following [CH07, fig. 7], let us inductively define subsets $Con(\overline{x}; \overline{x}')$ of contexts by the rules:

$$\frac{}{[\cdot] \in Con(\overline{x}; \overline{x})} \tag{Con1}$$

$$\frac{C \in Con(\overline{x}; \overline{x}' \cup \{x\}) \qquad x \notin \overline{x}'}{\lambda x.\, C \in Con(\overline{x}; \overline{x}')} \tag{Con2}$$

$$\frac{C \in Con(\overline{x}; \overline{x}') \qquad e \in \Lambda(\overline{x}')}{C\, e \in Con(\overline{x}; \overline{x}')} \tag{Con3}$$

$$\frac{e \in \Lambda(\overline{x}') \qquad C \in Con(\overline{x}; \overline{x}')}{e\, C \in Con(\overline{x}; \overline{x}')} \tag{Con4}$$

For example, if $x \notin \overline{x}$ then $\lambda x.\, [\cdot] \in Con(\overline{x} \cup \{x\}; \overline{x})$. The role in the above definition of the double indexing over both $\overline{x}$ and $\overline{x}'$ becomes clearer once one notes the following properties, (5.32) and (5.33). They are easily proved, the first by induction on the derivation of $C \in Con(\overline{x}; \overline{x}')$ from the rules (Con1)–(Con4), the second by induction on the derivation of $C' \in Con(\overline{x}'; \overline{x}'')$.

$$e \in \Lambda(\overline{x}) \;\wedge\; C \in Con(\overline{x}; \overline{x}') \;\Rightarrow\; C[e] \in \Lambda(\overline{x}') \tag{5.32}$$

$$C \in Con(\overline{x}; \overline{x}') \;\wedge\; C' \in Con(\overline{x}'; \overline{x}'') \;\Rightarrow\; C'[C] \in Con(\overline{x}; \overline{x}''). \tag{5.33}$$

In particular, the elements of $Con(\overline{x}; \emptyset)$ are *closing contexts* for λ-terms with free variables in $\overline{x}$: if $e \in \Lambda(\overline{x})$ and $C \in Con(\overline{x}; \emptyset)$, then $C[e] \in \Lambda(\emptyset)$ is a closed λ-term, which we can consider evaluating.

Definition 5.6.2 (Contextual equivalence) The *contextual preorder* with respect to call-by-value evaluation is the λ-term relation given by

$$\overline{x} \vdash e_1 \preceq_{\mathrm{v}} e_2 \quad \stackrel{\text{def}}{=} \quad \forall C \in Con(\overline{x}; \emptyset).\; C[e_1]{\Downarrow_{\mathrm{v}}} \;\Rightarrow\; C[e_2]{\Downarrow_{\mathrm{v}}}$$

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

where in general we write $e\Downarrow_{\mathrm{v}}$ to mean that $e \Downarrow_{\mathrm{v}} v$ holds for some v. The λ-term relation of *contextual equivalence*, $\overline{x} \vdash e_1 =_{\mathrm{v}} e_2$, holds iff $\overline{x} \vdash e_1 \leq_{\mathrm{v}} e_2$ and $\overline{x} \vdash e_2 \leq_{\mathrm{v}} e_1$.

Exercise 5.6.3 Show that $\leq_{\mathrm{v}}$ is a precongruence relation (Definition 5.4.2). [Hint: for property (Com2), given $C \in Con(\overline{x}; \emptyset)$ consider $C[\lambda x.[\cdot]] \in Con(\overline{x} \cup \{x\}; \emptyset)$; for property (Com3), given $C \in Con(\overline{x}; \emptyset)$ consider $C[e_1\,[\cdot]]$ and $C[[\cdot]\,e_2']$ and use Exercise 5.4.3.]

Lemma 5.6.4 *If $\overline{x} \vdash e_1 \lesssim_{\mathrm{v}} e_2$ and $C \in Con(\overline{x}; \overline{x}')$, then $\overline{x}' \vdash C[e_1] \lesssim_{\mathrm{v}} C[e_2]$.*

Proof By induction on the derivation of $C \in Con(\overline{x}; \overline{x}')$ from the rules (Con1)–(Con4), using Theorem 5.5.5. □

Theorem 5.6.5 (Applicative bisimilarity is contextual equivalence) *For all $\overline{x} \in \wp_{\mathrm{fin}}(Var)$ and $e_1, e_2 \in \Lambda(\overline{x})$, $\overline{x} \vdash e_1 \lesssim_{\mathrm{v}} e_2$ iff $\overline{x} \vdash e_1 \leq_{\mathrm{v}} e_2$ (and hence $\overline{x} \vdash e_1 \simeq_{\mathrm{v}} e_2$ iff $\overline{x} \vdash e_1 =_{\mathrm{v}} e_2$).*

Proof We first prove

$$\overline{x} \vdash e_1 \lesssim_{\mathrm{v}} e_2 \;\Rightarrow\; \overline{x} \vdash e_1 \leq_{\mathrm{v}} e_2. \tag{5.34}$$

If $\overline{x} \vdash e_1 \lesssim_{\mathrm{v}} e_2$, then for any $C \in Con(\overline{x}; \emptyset)$ by Lemma 5.6.4 we have $C[e_1] \lesssim_{\mathrm{v}} C[e_2]$. Since $\lesssim_{\mathrm{v}}$ is an applicative bisimulation, this means in particular that $C[e_1]\Downarrow_{\mathrm{v}}$ implies $C[e_2]\Downarrow_{\mathrm{v}}$. So by definition, $\overline{x} \vdash e_1 \leq_{\mathrm{v}} e_2$.

To prove the converse of (5.34), first note that it suffices to show that $=_{\mathrm{v}}$ restricted to closed λ-terms is an applicative simulation and hence contained in $\lesssim_{\mathrm{v}}$. For then, if we have $\overline{x} \vdash e_1 \leq_{\mathrm{v}} e_2$, by repeated use of the congruence property (Com2) (which we know holds of $=_{\mathrm{v}}$ from Exercise 5.6.3), we get $\emptyset \vdash \lambda\overline{x}.\, e_1 =_{\mathrm{v}} \lambda\overline{x}.\, e_2$ and hence $\lambda\overline{x}.\, e_1 \lesssim_{\mathrm{v}} \lambda\overline{x}.\, e_2$; but then we can use Exercise 5.3.7 to deduce that $\overline{x} \vdash e_1 \lesssim_{\mathrm{v}} e_2$.

So we just have to show that $\{(e_1, e_2) \mid \emptyset \vdash e_1 \leq_{\mathrm{v}} e_2\}$ has the applicative simulation property. If $\emptyset \vdash e_1 \leq_{\mathrm{v}} e_2$ and $e_1 \Downarrow_{\mathrm{v}} \lambda x.\, e_1'$, then $C[e_1]\Downarrow_{\mathrm{v}}$ with $C = [\cdot]$, so $C[e_2]\Downarrow_{\mathrm{v}}$, that is, $e_2 \Downarrow_{\mathrm{v}} \lambda x.\, e_2'$ for some e_2'. From Exercise 5.3.5 we thus have $e_i \simeq_{\mathrm{v}} \lambda x.\, e_i'$ and hence by (5.34) that

$$\emptyset \vdash \lambda x.\, e_1' =_{\mathrm{v}} e_1 \leq_{\mathrm{v}} e_2 =_{\mathrm{v}} \lambda x.\, e_2'.$$

So $\emptyset \vdash \lambda x.\, e_1' \leq_{\mathrm{v}} \lambda x.\, e_2'$ and hence for any $v \in V(\emptyset)$ we can use the congruence property of $\leq_{\mathrm{v}}$ (Exercise 5.6.3) to deduce that $\emptyset \vdash (\lambda x.\, e_1')\, v \leq_{\mathrm{v}} (\lambda x.\, e_2')\, v$. From Exercise 5.3.5 and (5.34), we have $\emptyset \vdash (\lambda x.\, e_i')\, v =_{\mathrm{v}} e_i'\{v/x\}$ and therefore $\emptyset \vdash e_1'\{v/x\} \leq_{\mathrm{v}} e_2'\{v/x\}$. Thus $\{(e_1, e_2) \mid \emptyset \vdash e_1 \leq_{\mathrm{v}} e_2\}$ does indeed have property (v-Sim). □

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

As a corollary of the coincidence of contextual equivalence with applicative bisimulation, we have the following extensionality properties of λ-terms modulo $=_{\mathrm{v}}$.

Corollary 5.6.6 (Extensionality) *Given variables* $\overline{x} = \{x_1, \ldots, x_n\}$ *and* λ*-terms* $e, e' \in \Lambda(\overline{x})$*, then* $\overline{x} \vdash e =_{\mathrm{v}} e'$ *iff* $\emptyset \vdash \lambda\overline{x}.\, e =_{\mathrm{v}} \lambda\overline{x}.\, e'$ *iff*

$$\forall v_1 \ldots, v_n \in V(\emptyset).\ \emptyset \vdash e\{v_1, \ldots, v_n/x_1, \ldots, x_n\} =_{\mathrm{v}} e\{v_1, \ldots, v_n/x_1, \ldots, x_n\}.$$

Proof Just note that these properties hold of $\simeq_{\mathrm{v}}$ (by definition of the open extension in Definition 5.4.1 and using Exercise 5.3.7); so we can apply Theorem 5.6.5. □

Remark 5.6.7 ('Context free' contextual equivalence) We noted in Remark 5.6.1 that the notion of context is unpleasantly concrete – it prevents one from working uniformly at the level of α-equivalence classes of syntax trees. In fact one can dispense with contexts entirely and work instead with a coinductive characterisation of the contextual preorder and equivalence phrased in terms of λ-term relations. (This is an instance of the 'relational' approach to contextual equivalence first proposed by Gordon [Gor98] and Lassen [Las98a, Las98b].) The contextual preorder turns out to be the largest λ-term relation $\mathcal{R}$ that is both compatible (Definition 5.4.2) and *adequate* (for call-by-value evaluation), which by definition means

$$\forall e.\, e' \in \Lambda(\emptyset).\ \emptyset \vdash e\ \mathcal{R}\ e' \ \Rightarrow\ (e{\Downarrow_{\mathrm{v}}} \ \Rightarrow\ e'{\Downarrow_{\mathrm{v}}}). \qquad \text{(v-Adeq)}$$

To see this, let $\mathbb{CA}$ be the collection of all compatible and adequate λ-term relations and let

$$\leq_{\mathrm{v}}^{\mathrm{ca}} \ \stackrel{\mathrm{def}}{=}\ \bigcup \mathbb{CA}. \qquad (5.35)$$

We first want to show that $\leq_{\mathrm{v}}^{\mathrm{ca}} \in \mathbb{CA}$ and hence that $\leq_{\mathrm{v}}^{\mathrm{ca}}$ is the largest compatible and adequate λ-term relation. Note that the identity relation $\{(\overline{x}, e, e) \mid e \in \Lambda(\overline{x})\}$ is in $\mathbb{CA}$; so $\leq_{\mathrm{v}}^{\mathrm{ca}}$ is reflexive and hence in particular satisfies compatibility property (Com1). It is clear that properties (Com2) and (v-Adeq) are closed under taking unions of relations, so that $\leq_{\mathrm{v}}^{\mathrm{ca}}$ has these properties; but the same is not true for property (Com3). However, if $\mathcal{R}_1, \mathcal{R}_2 \in \mathbb{CA}$ then it is easy to see that the composition $\mathcal{R}_1 \circ \mathcal{R}_2$ is also in $\mathbb{CA}$; hence $\leq_{\mathrm{v}}^{\mathrm{ca}} \circ \leq_{\mathrm{v}}^{\mathrm{ca}} \subseteq \leq_{\mathrm{v}}^{\mathrm{ca}}$, that is, $\leq_{\mathrm{v}}^{\mathrm{ca}}$ is transitive. So by Exercise 5.4.3, for (Com3) it is enough to show that $\leq_{\mathrm{v}}^{\mathrm{ca}}$ satisfies (Com3L) and (Com3R); and this is straightforward, since unlike (Com3) these properties clearly are closed under taking unions of relations.

So the largest compatible and adequate λ-term relation not only exists, but is reflexive and transitive.

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

To see that $\preceq_{\mathrm{v}}^{\mathrm{ca}}$ coincides with the contextual preorder, first note that it is immediate from its definition that $\preceq_{\mathrm{v}}$ is adequate; and we noted in Exercise 5.6.3 that it is a precongruence. So $\preceq_{\mathrm{v}} \in \mathbb{CA}$ and hence $\preceq_{\mathrm{v}} \subseteq \preceq_{\mathrm{v}}^{\mathrm{ca}}$. Since $\preceq_{\mathrm{v}}^{\mathrm{ca}}$ is a precongruence we can prove

$$\overline{x} \vdash e_1 \preceq_{\mathrm{v}}^{\mathrm{ca}} e_2 \ \wedge \ C \in \mathit{Con}(\overline{x}; \overline{x}') \ \Rightarrow \ \overline{x}' \vdash C[e_1] \preceq_{\mathrm{v}}^{\mathrm{ca}} C[e_2]$$

in the same way that Lemma 5.6.4 was proved. Thus

$$\begin{aligned}
&\overline{x} \vdash e_1 \preceq_{\mathrm{v}}^{\mathrm{ca}} e_2 \\
&\quad \Rightarrow \ \forall C \in \mathit{Con}(\overline{x}; \emptyset).\ \emptyset \vdash C[e_1] \preceq_{\mathrm{v}}^{\mathrm{ca}} C[e_2] \\
&\quad \Rightarrow \ \forall C \in \mathit{Con}(\overline{x}; \emptyset).\ C[e_1]{\Downarrow_{\mathrm{v}}} \ \Rightarrow \ C[e_2]{\Downarrow_{\mathrm{v}}} \quad \text{since } \preceq_{\mathrm{v}}^{\mathrm{ca}} \text{ is adequate} \\
&\quad \Rightarrow \ \overline{x} \vdash e_1 \preceq_{\mathrm{v}} e_2.
\end{aligned}$$

So altogether we have that $\preceq_{\mathrm{v}}^{\mathrm{ca}}$ is equal to $\preceq_{\mathrm{v}}$.

Exercise 5.6.8 Show that $=_{\mathrm{v}}$ is the largest λ-term relation that is both compatible and *bi-adequate*:

$$\forall e.\, e' \in \Lambda(\emptyset).\ \emptyset \vdash e \ \mathcal{R} \ e' \ \Rightarrow \ (e{\Downarrow_{\mathrm{v}}} \ \Leftrightarrow \ e'{\Downarrow_{\mathrm{v}}}). \qquad \text{(Bi-adeq)}$$

Exercise 5.6.9 Adapt the proof of Theorem 5.6.5 to give a direct proof (that is, a proof making no use of $\preceq_{\mathrm{v}}$) of the fact that $\lesssim_{\mathrm{v}}$ coincides with the relation $\preceq_{\mathrm{v}}^{\mathrm{ca}}$ defined in (5.35).

5.7 The transitive closure trick

We noted in Proposition 5.3.4 that because the evaluation relation $\Downarrow_{\mathrm{v}}$ is deterministic, applicative bisimilarity is the symmetrisation of applicative similarity: $\simeq_{\mathrm{v}} = \lesssim_{\mathrm{v}} \cap \lesssim_{\mathrm{v}}^{op}$. We used this observation in Section 5.5 to deduce that $\simeq_{\mathrm{v}}$ is a congruence by using Howe's method to show that $\lesssim_{\mathrm{v}}$ is a precongruence. What happens if we add features to the λ-calculus that cause $\simeq_{\mathrm{v}}$ to be different from $\lesssim_{\mathrm{v}} \cap \lesssim_{\mathrm{v}}^{op}$? Can one apply Howe's method directly to $\simeq_{\mathrm{v}}$ to deduce that it is a congruence in that case? Note that Howe's 'precongruence candidate' construction $(\_)^H$ has an asymmetric nature: $\mathcal{R}^H$ is obtained from $\mathcal{R}$ by inductively closing under the compatibility properties (Com1)–(Com3) at the same time composing with $\mathcal{R}$ *on the right*. (This is needed to get the proof of the key Lemma 5.5.4 to go through.) So if we apply $(\_)^H$ to $\simeq_{\mathrm{v}}$ we do not get a symmetric relation, therefore it cannot coincide with $\simeq_{\mathrm{v}}$ and hence in particular we cannot hope to transfer the congruence properties of $(\simeq_{\mathrm{v}})^H$ to $\simeq_{\mathrm{v}}$ in quite the same way that we did for applicative similarity. However,

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

as Howe observed [How96, lemma 3.3], the transitive closure of $(\simeq_{\mathrm{v}})^H$ is a symmetric relation and this fact is sufficient to deduce congruence of $\simeq_{\mathrm{v}}$ via $(\simeq_{\mathrm{v}})^H$. We will illustrate this refinement of the method by adding an 'erratic' *choice operator* ($\oplus$) to the call-by-value λ-calculus, which does indeed cause $\simeq_{\mathrm{v}}$ to be different from $\lesssim_{\mathrm{v}} \cap \lesssim_{\mathrm{v}}^{op}$ (see Exercise 5.7.1).

Let the set $\Lambda^{\oplus}$ of $\lambda^{\oplus}$*-terms* and the subset $V^{\oplus} \subseteq \Lambda^{\oplus}$ of $\lambda^{\oplus}$*-values* consist of λ-terms and λ-values (modulo α-equivalence) extended with a binary operation $\oplus$:

$$\begin{aligned} e \in \Lambda^{\oplus} &::= v \mid e\,e \mid e \oplus e, \\ v \in V^{\oplus} &::= x \mid \lambda x.\, e. \end{aligned} \tag{5.36}$$

We write $\Lambda^{\oplus}(\overline{x})$ (respectively, $V^{\oplus}(\overline{x})$) for the subset of $\lambda^{\oplus}$-terms (respectively, $\lambda^{\oplus}$-values) whose free variables are in the finite set $\overline{x}$ of variables. A $\lambda^{\oplus}$*-term relation* $\mathcal{R}$ is a family of binary relations $\overline{x} \vdash (\_)\ \mathcal{R}\ (\_)$ on $\Lambda^{\oplus}(\overline{x})$, as $\overline{x}$ ranges over $\wp_{\mathrm{fin}}(\mathit{Var})$ (cf. Definition 5.4.2). $\mathcal{R}$ is a *congruence* if it has the properties (Sym), (Tra), (Com1)–(Com3) and an additional compatibility property for the new term constructor:

$$\begin{aligned} &\forall \overline{x} \in \wp_{\mathrm{fin}}(\mathit{Var}).\ \forall e_1, e_1', e_2, e_2' \in \Lambda(\overline{x}). \\ &\quad \overline{x} \vdash e_1\ \mathcal{R}\ e_1' \ \wedge\ \overline{x} \vdash e_2\ \mathcal{R}\ e_2' \ \Rightarrow\ \overline{x} \vdash e_1 \oplus e_2\ \mathcal{R}\ e_1' \oplus e_2'. \end{aligned} \tag{Com4}$$

(As before, satisfaction of (Com1)–(Com4) implies that $\mathcal{R}$ is reflexive; so a congruence is in particular an equivalence relation.)

We extend call-by-value evaluation to a relation $\Downarrow_{\mathrm{v}} \subseteq \Lambda^{\oplus}(\emptyset) \times \Lambda^{\oplus}(\emptyset)$ by adding to (Val) and (Cbv) the two rules

$$\frac{e_i \Downarrow_{\mathrm{v}} v}{e_1 \oplus e_2 \Downarrow_{\mathrm{v}} v} \quad (i = 1, 2). \tag{Ch}$$

The definitions of call-by-value applicative similarity ($\lesssim_{\mathrm{v}}$) and bisimilarity ($\simeq_{\mathrm{v}}$) on closed $\lambda^{\oplus}$-terms are as in Definition 5.3.1 except that $\Lambda(\emptyset)$ is replaced by the extended set $\Lambda^{\oplus}(\emptyset)$; and these relations are extended to $\lambda^{\oplus}$-term relations as in Definition 5.4.1, using closed $\lambda^{\oplus}$-value-substitutions.

Exercise 5.7.1 Defining

$$\begin{aligned} v_1 &\stackrel{\mathrm{def}}{=} \lambda x.\, x \\ v_2 &\stackrel{\mathrm{def}}{=} \lambda y.\, ((\lambda x.\, x\, x)\, (\lambda x.\, x\, x)) \\ v &\stackrel{\mathrm{def}}{=} \lambda z.\ (v_1 \oplus v_2) \\ e &\stackrel{\mathrm{def}}{=} v \oplus (\lambda z.\, v_1) \oplus (\lambda z.\, v_2), \end{aligned}$$

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

show that $v \lesssim_{\mathrm{v}} e \lesssim_{\mathrm{v}} v$, but that $v \not\simeq_{\mathrm{v}} e$. (Compare this with the examples in [San12, chapter 5], such as Exercise 5.8.11, showing the difference between trace equivalence and bisimilarity for labelled transition systems; recall that testing equivalence implies trace equivalence.)

We wish to prove that call-by-value applicative bisimilarity is a congruence for $\lambda^{\oplus}$-terms. We can still do so using Howe's precongruence candidate construction $(\_)^H$, which in this setting sends a $\lambda^{\oplus}$-term relation $\mathcal{R}$ to the $\lambda^{\oplus}$-term relation $\mathcal{R}^H$ inductively defined by the rules (How1)–(How3) from Section 5.5 together with an extra rule for the new binary operation, $\oplus$:

$$\frac{\overline{x} \vdash e_1 \ \mathcal{R}^H \ e_1' \qquad \overline{x} \vdash e_2 \ \mathcal{R}^H \ e_2' \qquad \overline{x} \vdash e_1' \oplus e_2' \ \mathcal{R} \ e_3}{\overline{x} \vdash e_1 \oplus e_2 \ \mathcal{R}^H \ e_3}. \tag{How4}$$

The rule follows the same pattern as for the existing λ-calculus constructs and does not disturb the validity of the general properties of $(\_)^H$ given in Lemmas 5.5.1 and 5.5.3. More delicate is the analogue of the key Lemma 5.5.4, since it depends not only upon the syntactical form of $e \oplus e'$, but also upon its operational semantics, as defined by the rules (Ch).

Lemma 5.7.2 (Key lemma, version 2) *For all* $e_1, \lambda x.\, e, e_2 \in \Lambda^{\oplus}(\emptyset)$

$$\begin{aligned} e_1 \Downarrow_{\mathrm{v}} \lambda x.\, e \ \wedge \ \emptyset \vdash e_1 \ (\simeq_{\mathrm{v}})^H \ e_2 \\ \Rightarrow \ \exists e' \in \Lambda^{\oplus}.\ e_2 \Downarrow_{\mathrm{v}} \lambda x.\, e' \ \wedge \ \{x\} \vdash e \ (\simeq_{\mathrm{v}})^H \ e'. \end{aligned} \tag{5.37}$$

Proof This can be proved by showing that

$$\begin{aligned} \mathcal{E} \stackrel{\text{def}}{=} \{(e_1, \lambda x.\, e) \ \mid \ \forall e_2.\ \emptyset \vdash e_1 \ (\simeq_{\mathrm{v}})^H \ e_2 \\ \Rightarrow \ \exists e'.\ e_2 \Downarrow_{\mathrm{v}} \lambda x.\, e' \ \wedge \ \{x\} \vdash e \ (\simeq_{\mathrm{v}})^H \ e'\} \end{aligned}$$

is closed under the rules (Val), (Cbv) and (Ch) inductively defining $\Downarrow_{\mathrm{v}}$ for $\lambda^{\oplus}$-terms. The proof of closure under (Val) and (Cbv) is exactly as in the proof of Lemma 5.5.4. We give the proof of closure under the $i = 1$ case of (Ch), the argument for the other case being symmetric.

So suppose $(e_1, \lambda x.\, e) \in \mathcal{E}$. We have to show for any $e_1' \in \Lambda^{\oplus}(\emptyset)$ that $(e_1 \oplus e_1', \lambda x.\, e) \in \mathcal{E}$. For any e_2, if $\emptyset \vdash e_1 \oplus e_1' \ (\simeq_{\mathrm{v}})^H \ e_2$ holds it must have been deduced by an application of rule (How4) to

$$\emptyset \vdash e_1 \ (\simeq_{\mathrm{v}})^H \ e_3 \tag{5.38}$$

$$\emptyset \vdash e_1' \ (\simeq_{\mathrm{v}})^H \ e_3' \tag{5.39}$$

$$e_3 \oplus e_3' \simeq_{\mathrm{v}} e_2 \tag{5.40}$$

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

for some e_3, e_3'. Since $(e_1, \lambda x.\, e) \in \mathcal{E}$, it follows from (5.38) that for some e'' it is the case that

$$e_3 \Downarrow_{\mathrm{v}} \lambda x.\, e'' \tag{5.41}$$

$$\overline{x} \vdash e\ (\simeq_{\mathrm{v}})^H\ e''. \tag{5.42}$$

Applying (Ch) to (5.41), we get $e_3 \oplus e_3' \Downarrow_{\mathrm{v}} \lambda x.\, e''$. Since $\simeq_{\mathrm{v}}$ is an applicative bisimulation, it follows from this and (5.40) that

$$e_2 \Downarrow_{\mathrm{v}} \lambda x.\, e' \tag{5.43}$$

$$\overline{x} \vdash e'' \simeq_{\mathrm{v}} e'. \tag{5.44}$$

Applying Lemma 5.5.1(ii) to (5.42) and (5.44), we get $\overline{x} \vdash e\ (\simeq_{\mathrm{v}})^H\ e'$. Therefore we do indeed have $(e_1 \oplus e_1', \lambda x.\, e) \in \mathcal{E}$. □

The lemma implies that $(\simeq_{\mathrm{v}})^H$ is an applicative simulation; but to get the applicative *bi*simulation property we have to move to its transitive closure.

Definition 5.7.3 (Transitive closure) The *transitive closure* of a $\lambda^{\oplus}$-term relation $\mathcal{R}$, is the $\lambda^{\oplus}$-term relation $\mathcal{R}^+$ inductively defined by the rules

$$\frac{\overline{x} \vdash e\ \mathcal{R}\ e'}{\overline{x} \vdash e\ \mathcal{R}^+\ e'} \qquad \frac{\overline{x} \vdash e\ \mathcal{R}^+\ e' \qquad \overline{x} \vdash e'\ \mathcal{R}^+\ e''}{\overline{x} \vdash e\ \mathcal{R}^+\ e''}. \tag{TC}$$

Exercise 5.7.4 Show that if $\mathcal{R}$ is compatible, then so is $\mathcal{R}^+$. Show that if $\mathcal{R}$ is closed under value-substitution, then so is $\mathcal{R}^+$.

Lemma 5.7.5 *If a $\lambda^{\oplus}$-term relation $\mathcal{R}$ is an equivalence relation, then so is $(\mathcal{R}^H)^+$.*

Proof Being a transitive closure, $(\mathcal{R}^H)^+$ is of course transitive; and since $\mathcal{R}$ is reflexive, by Lemma 5.5.1(i) $\mathcal{R}^H$ is reflexive and hence so is $(\mathcal{R}^H)^+$. So the real issue is the symmetry property. To prove $\overline{x} \vdash e\ (\mathcal{R}^H)^+\ e' \ \Rightarrow\ \overline{x} \vdash e'\ (\mathcal{R}^H)^+\ e$, it suffices to prove

$$\overline{x} \vdash e\ \mathcal{R}^H\ e' \ \Rightarrow\ \overline{x} \vdash e'\ (\mathcal{R}^H)^+\ e \tag{5.45}$$

and then argue by induction on the derivation of $\overline{x} \vdash e\ (\mathcal{R}^H)^+\ e'$ from the rules (TC). To prove (5.45), one argues by induction on the derivation of $\overline{x} \vdash e\ \mathcal{R}^H\ e'$ from the rules (How1)–(How4), using Lemma 5.5.1(iii). In addition to that lemma, for closure under (How2)–(How4) one has to use the fact that $(\mathcal{R}^H)^+$ satisfies (Com2)–(Com4) respectively, which follows from Lemma 5.5.1(i) and Exercise 5.7.4. □

Theorem 5.7.6 *Call-by-value applicative bisimilarity is a congruence relation for $\lambda^{\oplus}$-terms.*

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

Proof We show that $\simeq_{\mathrm{v}}$ is equal to $((\simeq_{\mathrm{v}})^H)^+$. The former is an equivalence relation and the latter is a compatible $\lambda^{\oplus}$-term relation by Lemma 5.5.1(i) and Exercise 5.7.4; so altogether $\simeq_{\mathrm{v}} = ((\simeq_{\mathrm{v}})^H)^+$ is a congruence relation.

We have $\simeq_{\mathrm{v}} \subseteq (\simeq_{\mathrm{v}})^H \subseteq ((\simeq_{\mathrm{v}})^H)^+$ by Lemma 5.5.1(iii) and by definition of $(\_)^+$. For the reverse inclusion, since $((\simeq_{\mathrm{v}})^H)^+$ is closed under value-substitution (Lemma 5.5.3 and Exercise 5.7.4), it suffices to show that its restriction to closed $\lambda^{\oplus}$-terms is a call-by-value applicative bisimulation. From Lemma 5.7.2 we get

$$e_1 \Downarrow_{\mathrm{v}} \lambda x.e \ \wedge\ \emptyset \vdash e_1\ (\simeq_{\mathrm{v}})^H\ e_2 \\ \Rightarrow\ \exists e' \in \Lambda^{\oplus}.\ e_2 \Downarrow_{\mathrm{v}} \lambda x.e' \wedge \forall e'' \in \Lambda^{\oplus}(\emptyset).\ \emptyset \vdash e\{e''/x\}\ ((\simeq_{\mathrm{v}})^H)^+\ e'\{e''/x\}$$

(using the fact that $((\simeq_{\mathrm{v}})^H)^+$ is closed under value-substitution and contains ${\simeq_{\mathrm{v}}}^H$). Therefore

$$e_1 \Downarrow_{\mathrm{v}} \lambda x.e \ \wedge\ \emptyset \vdash e_1\ ((\simeq_{\mathrm{v}})^H)^+\ e_2 \\ \Rightarrow \exists e' \in \Lambda^{\oplus}.\ e_2 \Downarrow_{\mathrm{v}} \lambda x.e' \wedge \forall e'' \in \Lambda^{\oplus}(\emptyset).\ \emptyset \vdash e\{e''/x\}\ ((\simeq_{\mathrm{v}})^H)^+\ e'\{e''/x\}.$$

So $\emptyset \vdash (\_)\ ((\simeq_{\mathrm{v}})^H)^+\ (\_)$ is an applicative simulation; and since by Lemma 5.7.5 it is a symmetric relation, it is in fact an applicative bisimulation, as required. □

5.8 CIU-equivalence

Howe's method has been applied most often in the literature to prove congruence properties of various kinds of bisimilarity. It is less well known that it also provides a useful method for proving congruence of 'CIU' equivalences. This form of equivalence was introduced by Mason and Talcott in their work on contextual equivalence of impure functional programs [MT91, MT92], but is prefigured by Milner's context lemma for contextual equivalence for the pure functional language PCF [Mil77]. In both cases the form of program equivalence of primary concern is the kind studied in Section 5.6, namely contextual equivalence. The quantification over all contexts that occurs in its definition (recall Definition 5.6.2) makes it hard to prove either specific instances of, or general properties of contextual equivalence. Therefore one seeks alternative characterisations of contextual equivalence that make such tasks easier. One approach is to prove a 'context lemma' to the effect that quantification over all contexts can be replaced by quantification over a restricted class of contexts without affecting the associated contextual equivalence. As we now explain, the

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

coincidence of CIU-equivalence with contextual equivalence is exactly such a result.

We take the acronym 'CIU' to stand for a permutation of '*U*ses of *C*losed *I*nstantiations': the 'closed instantiations' part refers to the fact that CIU-equivalence is first defined on closed terms and then extended to open ones via closing substitutions (just as for applicative bisimulation in Definition 5.4.1); the 'uses' part refers to the fact that closed terms are identified when they have the same evaluation behaviour in any context that 'uses' its hole, that is, in any Felleisen-style *evaluation context* [FH92]. Thus CIU-equivalence is a form of contextual equivalence in which restrictions are placed upon the kind of contexts in which operational behaviour of program phrases is observed. The restrictions make it easier to establish some properties of CIU-equivalence compared with contextual equivalence itself, for which there are no restrictions upon the contexts. Once one has proved that CIU-equivalence is a congruence, it is straightforward to see that it actually coincides with contextual equivalence. Therefore the notion of CIU-equivalence is really a means of establishing properties of contextual equivalence, namely the ones that stem from the reduction properties of terms, such as those in Exercise 5.8.5.

We illustrate these ideas in this section by defining CIU-equivalence for call-by-value λ-calculus with erratic choice ($\oplus$), using Howe's method to prove that it is a congruence and hence that it coincides with contextual equivalence.

We will adopt the more abstract, relational approach to contextual equivalence outlined in Remark 5.6.7: the contextual preorder $\leq_{\mathrm{v}}$ for the call-by-value $\lambda^{\oplus}$-calculus is the largest $\lambda^{\oplus}$-term relation that is both compatible (that is, satisfies (Com1)–(Com4)) and adequate (that is, satisfies (v-Adeq)). At the same time, rather than using Felleisen's notion of 'evaluation context', which in this case would be

$$E ::= [\cdot] \;\big|\; E\, e \;\big|\; v\, E\,,$$

we will use an 'inside out' representation of such contexts as a stack of basic *evaluation frames* (which in this case are $[\cdot]e$ and $v[\cdot]$). Such a representation of evaluation contexts yields a convenient, structurally inductive characterisation of the 'may evaluate to some value in call-by-value' property used in the definition of contextual equivalence (Lemma 5.8.3); see [Pit02, section 4] for a more detailed explanation of this point.

Definition 5.8.1 (Frame stacks) The set $Stk^{\oplus}$ of call-by-value *frame stacks* is given by

$$s \in Stk^{\oplus} ::= \mathtt{nil} \;\big|\; [\cdot]e :: s \;\big|\; v[\cdot] :: s. \tag{5.46}$$

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

As for terms and values, we write $Stk^{\oplus}(\overline{x})$ for the subset of frame stacks whose free variables are in the finite set $\overline{x}$. Given $s \in Stk^{\oplus}(\overline{x})$ and $e \in \Lambda^{\oplus}(\overline{x})$, we get a term $E_s[e] \in \Lambda^{\oplus}(\overline{x})$, defined by:

$$\begin{cases} E_{\mathtt{nil}}[e] & \stackrel{\text{def}}{=} e \\ E_{[\cdot]e' :: s}[e] & \stackrel{\text{def}}{=} E_s[e\, e'] \\ E_{v[\cdot] :: s}[e] & \stackrel{\text{def}}{=} E_s[v\, e]. \end{cases} \tag{5.47}$$

Definition 5.8.2 (Transition and may-termination) The binary relation of call-by-value *transition* between pairs $(s\ ,\ e)$ of frame stacks and closed $\lambda^{\oplus}$-terms

$$(s\ ,\ e) \to_{\mathrm{v}} (s'\ ,\ e')$$

is defined by cases as follows:

$$\begin{aligned} ((\lambda x.\, e)[\cdot] :: s\ ,\ v) &\to_{\mathrm{v}} (s\ ,\ e\{v/x\}) && (\to_{\mathrm{v}}1) \\ ([\cdot]e :: s\ ,\ v) &\to_{\mathrm{v}} (v[\cdot] :: s\ ,\ e) && (\to_{\mathrm{v}}2) \\ (s\ ,\ e\, e') &\to_{\mathrm{v}} ([\cdot]e' :: s\ ,\ e) && (\to_{\mathrm{v}}3) \\ (s\ ,\ e_1 \oplus e_2) &\to_{\mathrm{v}} (s\ ,\ e_i) \qquad \text{for } i = 1, 2. && (\to_{\mathrm{v}}4) \end{aligned}$$

We write $(s\ ,\ e)\downarrow_{\mathrm{v}}^{(n)}$ if for some closed value $v \in V^{\oplus}(\emptyset)$ there exists a sequence of transitions $(s\ ,\ e) \to_{\mathrm{v}} \cdots \to_{\mathrm{v}} (\mathtt{nil}\ ,\ v)$ of length less than or equal to n; and we write $(s\ ,\ e)\downarrow_{\mathrm{v}}$ and say that $(s\ ,\ e)$ *may terminate* if $(s\ ,\ e)\downarrow_{\mathrm{v}}^{(n)}$ holds for some $n \geq 0$.

The relationship between this termination relation and 'may evaluate to some value in call-by-value', $e\Downarrow_{\mathrm{v}} \stackrel{\text{def}}{=} \exists v.\ e \Downarrow_{\mathrm{v}} v$, is as follows.

Lemma 5.8.3 *For all closed frame stacks $s \in Stk^{\oplus}(\emptyset)$ and closed $\lambda^{\oplus}$-terms $e \in \Lambda^{\oplus}(\emptyset)$, $(s\ ,\ e)\downarrow_{\mathrm{v}}$ iff $E_s[e]\Downarrow_{\mathrm{v}}$. In particular $e\Downarrow_{\mathrm{v}}$ holds iff $(\mathtt{nil}\ ,\ e)\downarrow_{\mathrm{v}}$.*

Proof The result can be deduced from the following properties of transition, evaluation and termination, where $\to_{\mathrm{v}}^*$ stands for the reflexive-transitive closure of $\to_{\mathrm{v}}$ and $(\_)\ @\ (\_)$ stands for the operation of appending two lists.

$$(s'\ ,\ E_s[e]) \to_{\mathrm{v}}^* (s\ @\ s'\ ,\ e) \tag{5.48}$$

$$(s'\ ,\ E_s[e])\downarrow_{\mathrm{v}} \ \Rightarrow\ (s\ @\ s'\ ,\ e)\downarrow_{\mathrm{v}} \tag{5.49}$$

$$e \Downarrow_{\mathrm{v}} v \ \Rightarrow\ (s\ ,\ e) \to_{\mathrm{v}}^* (s\ ,\ v) \tag{5.50}$$

$$(s\ ,\ e)\downarrow_{\mathrm{v}}^{(n)} \ \Rightarrow\ \exists v.\ e \Downarrow_{\mathrm{v}} v \ \wedge\ (s\ ,\ v)\downarrow_{\mathrm{v}}^{(n)}. \tag{5.51}$$

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

Properties (5.48) and (5.49) are proved by induction on the length of the list s; (5.50) by induction on the derivation of $e \Downarrow_{\mathrm{v}} v$; and (5.51) by induction on n (and then by cases according to the structure of e).

Therefore if $(s\ ,\ e)\downarrow_{\mathrm{v}}$, then $(\mathtt{nil}\ ,\ E_s[e])\downarrow_{\mathrm{v}}$ by (5.48) and hence $E_s[e]\Downarrow_{\mathrm{v}}$ by (5.51). Conversely, if $E_s[e] \Downarrow_{\mathrm{v}} v$, then $(\mathtt{nil}\ ,\ E_s[e])\downarrow_{\mathrm{v}}$ by (5.50) and hence $(s\ ,\ e)\downarrow_{\mathrm{v}}$ by (5.49). □

Definition 5.8.4 (CIU-equivalence) Given $e, e' \in \Lambda^{\oplus}(\emptyset)$, we define

$$e \leq_{\mathrm{v}}^{\mathrm{ciu}} e' \stackrel{\mathrm{def}}{=} \forall s \in Stk^{\oplus}(\emptyset).\ (s\ ,\ e)\downarrow_{\mathrm{v}} \Rightarrow (s\ ,\ e')\downarrow_{\mathrm{v}}.$$

This is extended to a $\lambda^{\oplus}$-term relation, called the (call-by-value) *CIU-preorder*, via closing value-substitutions:

$$\overline{x} \vdash e \leq_{\mathrm{v}}^{\mathrm{ciu}} e' \stackrel{\mathrm{def}}{=} \forall v_1, \ldots, v_n \in V^{\oplus}(\emptyset).\ e\{v_1, \ldots, v_n/x_1, \ldots, x_n\} \leq_{\mathrm{v}}^{\mathrm{ciu}} e'\{v_1, \ldots, v_n/x_1, \ldots, x_n\}$$

(where $\overline{x} = \{x_1, \ldots, x_n\}$). *CIU-equivalence* is the symmetrisation of this relation:

$$\overline{x} \vdash e =_{\mathrm{v}}^{\mathrm{ciu}} e' \stackrel{\mathrm{def}}{=} \overline{x} \vdash e \leq_{\mathrm{v}}^{\mathrm{ciu}} e' \wedge \overline{x} \vdash e' \leq_{\mathrm{v}}^{\mathrm{ciu}} e'.$$

It is clear from its definition that the $\lambda^{\oplus}$-term relation $\leq_{\mathrm{v}}^{\mathrm{ciu}}$ is a preorder (reflexive and transitive) and hence that $=_{\mathrm{v}}^{\mathrm{ciu}}$ is an equivalence relation.

Exercise 5.8.5 Show that β_{v}-equivalence holds up to $=_{\mathrm{v}}^{\mathrm{ciu}}$:

$$\overline{x} \vdash (\lambda x.\, e)\, v =_{\mathrm{v}}^{\mathrm{ciu}} e\{v/x\}.$$

Show that $e_1 \oplus e_2$ is the least upper bound of e_1 and e_2 with respect to the CIU-preorder, $\leq_{\mathrm{v}}^{\mathrm{ciu}}$.

We will use Howe's method to show that $\leq_{\mathrm{v}}^{\mathrm{ciu}}$ is a compatible $\lambda^{\oplus}$-term relation and deduce that it coincides with the contextual preorder (and hence that $=_{\mathrm{v}}^{\mathrm{ciu}}$ coincides with contextual equivalence). As for previous applications of Howe's method, we know that $(\leq_{\mathrm{v}}^{\mathrm{ciu}})^H$ is a compatible $\lambda^{\oplus}$-term relation containing $\leq_{\mathrm{v}}^{\mathrm{ciu}}$ (Lemma 5.5.1). So for compatibility of $\leq_{\mathrm{v}}^{\mathrm{ciu}}$, we just need to show $(\leq_{\mathrm{v}}^{\mathrm{ciu}})^H \subseteq \leq_{\mathrm{v}}^{\mathrm{ciu}}$. Since $\leq_{\mathrm{v}}^{\mathrm{ciu}}$ is defined on open terms by taking closing value-substitutions, both it and $(\leq_{\mathrm{v}}^{\mathrm{ciu}})^H$ are closed under value-substitution (Lemma 5.5.3); so it suffices to show for closed $\lambda^{\oplus}$-terms that

$$\emptyset \vdash e\ (\leq_{\mathrm{v}}^{\mathrm{ciu}})^H\ e' \Rightarrow e \leq_{\mathrm{v}}^{\mathrm{ciu}} e'$$

that is,

$$\emptyset \vdash e\ (\leq_{\mathrm{v}}^{\mathrm{ciu}})^H\ e' \wedge (s\ ,\ e)\downarrow_{\mathrm{v}} \Rightarrow (s\ ,\ e')\downarrow_{\mathrm{v}}.$$

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

We do this by proving the following analogue of the key Lemma 5.5.4. It uses an extension of the Howe precongruence candidate construction $(\_)^H$ to (closed) frame stacks: given a $\lambda^{\oplus}$-term relation $\mathcal{R}$, the relation $s\ \mathcal{R}^H\ s'$ is inductively defined by:

$$\frac{}{\mathtt{nil}\ \mathcal{R}^H\ \mathtt{nil}} \qquad \frac{\emptyset \vdash e\ \mathcal{R}^H\ e' \qquad s\ \mathcal{R}^H\ s'}{[\cdot]e :: s\ \mathcal{R}^H\ [\cdot]e' :: s'}$$
$$\frac{\{x\} \vdash e\ \mathcal{R}^H\ e' \qquad s\ \mathcal{R}^H\ s'}{(\lambda x.\, e)[\cdot] :: s\ \mathcal{R}^H\ (\lambda x.\, e')[\cdot] :: s'}. \tag{How-Stk}$$

Lemma 5.8.6 (Key lemma, version 3)

$$\forall s, e, s', e'.\ s\ (\leq_{\mathrm{v}}^{\mathrm{ciu}})^H\ s' \wedge \emptyset \vdash e\ (\leq_{\mathrm{v}}^{\mathrm{ciu}})^H\ e' \wedge (s\ ,\ e)\downarrow_{\mathrm{v}}^{(n)} \Rightarrow (s'\ ,\ e')\downarrow_{\mathrm{v}}. \tag{5.52}$$

Proof The proof is by induction on n. The base case $n = 0$ is straightforward, so we concentrate on the induction step. Assume (5.52) holds and that

$$s_1\ (\leq_{\mathrm{v}}^{\mathrm{ciu}})^H\ s' \tag{5.53}$$
$$\emptyset \vdash e_1\ (\leq_{\mathrm{v}}^{\mathrm{ciu}})^H\ e' \tag{5.54}$$
$$(s_1\ ,\ e_1) \to_{\mathrm{v}} (s_2\ ,\ e_2) \tag{5.55}$$
$$(s_2\ ,\ e_2)\downarrow_{\mathrm{v}}^{(n)}. \tag{5.56}$$

We have to show that $(s'\ ,\ e')\downarrow_{\mathrm{v}}$ and do so by analysing (5.55) against the four possible cases ($\to_{\mathrm{v}}1$)–($\to_{\mathrm{v}}4$) in the definition of the transition relation.

Case ($\to_{\mathrm{v}}1$): So $s_1 = (\lambda x.\, e)[\cdot] :: s_2$, $e_1 = \lambda x.\, e_1'$ and $e_2 = e\{\lambda x.\, e_1'/x\}$, for some x, e and e_1'. Thus (5.53) must have been deduced by applying (How-Stk) to

$$\{x\} \vdash e\ (\leq_{\mathrm{v}}^{\mathrm{ciu}})^H\ e' \tag{5.57}$$
$$s_2\ (\leq_{\mathrm{v}}^{\mathrm{ciu}})^H\ s_2' \tag{5.58}$$

where $(\lambda x.\, e')[\cdot] :: s_2' = s'$. Since $e_1 = \lambda x.\, e_1'$, (5.54) must have been deduced by an application of (How2) to

$$\{x\} \vdash e_1'\ (\leq_{\mathrm{v}}^{\mathrm{ciu}})^H\ e_1'' \tag{5.59}$$
$$\lambda x.\, e_1''\ \leq_{\mathrm{v}}^{\mathrm{ciu}}\ e' \tag{5.60}$$

for some e_1''. From Lemma 5.5.3 we have that $(\leq_{\mathrm{v}}^{\mathrm{ciu}})^H$ is substitutive; so since (5.59) implies $\emptyset \vdash \lambda x.\, e_1'\ (\leq_{\mathrm{v}}^{\mathrm{ciu}})^H\ \lambda x.\, e_1''$, from this and (5.57) we get $\emptyset \vdash e\{\lambda x.\, e_1'/x\}\ (\leq_{\mathrm{v}}^{\mathrm{ciu}})^H\ e'\{\lambda x.\, e_1''/x\}$, that is, $\emptyset \vdash e_2\ (\leq_{\mathrm{v}}^{\mathrm{ciu}})^H\ e'\{\lambda x.\, e_1''/x\}$. Applying the induction hypothesis (5.52) to this, (5.58) and (5.56) gives $(s_2'\ ,\ e'\{\lambda x.\, e_1''/x\})\downarrow_{\mathrm{v}}$ and hence also $((\lambda x.\, e')[\cdot] :: s_2'\ ,\ \lambda x.\, e_1'')\downarrow_{\mathrm{v}}$. From this and

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

(5.60) we get $(\!|(\lambda x.\, e')[\cdot] :: s_2' \;,\; e'|\!)\downarrow_{\mathrm{v}}$. In other words $(\!|s' \;,\; e'|\!)\downarrow_{\mathrm{v}}$, as required for the induction step.

Case ($\rightarrow_{\mathrm{v}}2$): So $s_1 = [\cdot]e_2 :: s$, $e_1 = \lambda x.\, e_1'$ and $s_2 = (\lambda x.\, e_1')[\cdot] :: s$, for some s, x and e_1'. Thus (5.53) must have been deduced by applying (How-Stk) to

$$s \; (\leq_{\mathrm{v}}^{\mathrm{ciu}})^H \; s_2' \tag{5.61}$$

$$\emptyset \vdash e_2 \; (\leq_{\mathrm{v}}^{\mathrm{ciu}})^H \; e_2' \tag{5.62}$$

$$\tag{5.63}$$

where $[\cdot]e_2' :: s_2' = s'$. Since $e_1 = \lambda x.\, e_1'$, (5.54) must have been deduced by an application of (How2) to

$$\{x\} \vdash e_1' \; (\leq_{\mathrm{v}}^{\mathrm{ciu}})^H \; e_1'' \tag{5.64}$$

$$\lambda x.\, e_1'' \leq_{\mathrm{v}}^{\mathrm{ciu}} e' \tag{5.65}$$

for some e_1''. By (How-Stk) on (5.61) and (5.64), we get $s_2 \; (\leq_{\mathrm{v}}^{\mathrm{ciu}})^H \; (\lambda x.\, e_1'')[\cdot] :: s_2'$. Applying the induction hypothesis (5.52) to this, (5.62) and (5.56) gives $(\!|(\lambda x.\, e_1'')[\cdot] :: s_2' \;,\; e_2'|\!)\downarrow_{\mathrm{v}}$ and hence also $(\!|[\cdot]e_2' :: s_2' \;,\; \lambda x.\, e_1''|\!)\downarrow_{\mathrm{v}}$. From this and (5.65) we get $(\!|[\cdot]e_2' :: s_2' \;,\; e'|\!)\downarrow_{\mathrm{v}}$. In other words $(\!|s' \;,\; e'|\!)\downarrow_{\mathrm{v}}$, as required for the induction step.

Case ($\rightarrow_{\mathrm{v}}3$): So $e_1 = e_2\, e$ and $s_2 = [\cdot]e :: s_1$, for some e. Thus (5.54) must have been deduced by applying (How3) to

$$\emptyset \vdash e_2 \; (\leq_{\mathrm{v}}^{\mathrm{ciu}})^H \; e_2'' \tag{5.66}$$

$$\emptyset \vdash e \; (\leq_{\mathrm{v}}^{\mathrm{ciu}})^H \; e'' \tag{5.67}$$

$$e_2''\, e'' \leq_{\mathrm{v}}^{\mathrm{ciu}} e' \tag{5.68}$$

for some e_2'' and e''. Applying (How-Stk) to (5.67) and (5.53) we get $s_2 \; (\leq_{\mathrm{v}}^{\mathrm{ciu}})^H \; [\cdot]e'' :: s'$. Applying the induction hypothesis (5.52) to this, (5.66) and (5.56) gives $(\!|[\cdot]e'' :: s' \;,\; e_2''|\!)\downarrow_{\mathrm{v}}$ and hence also $(\!|s' \;,\; e_2''\, e''|\!)\downarrow_{\mathrm{v}}$. From this and (5.68) we get $(\!|s' \;,\; e'|\!)\downarrow_{\mathrm{v}}$, as required for the induction step.

Case ($\rightarrow_{\mathrm{v}}4$): We treat the $i = 1$ subcase of ($\rightarrow_{\mathrm{v}}4$), the argument for the $i = 2$ subcase being symmetric. So $s_1 = s_2$ and $e_1 = e_2 \oplus e$, for some e. Thus (5.54) must have been deduced by applying (How4) to

$$\emptyset \vdash e_2 \; (\leq_{\mathrm{v}}^{\mathrm{ciu}})^H \; e_2'' \tag{5.69}$$

$$\emptyset \vdash e \; (\leq_{\mathrm{v}}^{\mathrm{ciu}})^H \; e''$$

$$e_2'' \oplus e'' \leq_{\mathrm{v}}^{\mathrm{ciu}} e' \tag{5.70}$$

for some e_2'' and e''. Since $s_1 = s_2$, we can apply the induction hypothesis (5.52) to (5.53), (5.69) and (5.56) to conclude that $(\!|s' \;,\; e_2''|\!)\downarrow_{\mathrm{v}}$ and hence also that

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

$(s'\ ,\ e_2'' \oplus e') \downarrow_{\mathrm{v}}$. From this and (5.70) we get $(s'\ ,\ e') \downarrow_{\mathrm{v}}$, as required for the induction step. □

Theorem 5.8.7 (CIU-equivalence is contextual equivalence) *For all* $\overline{x} \in \wp_{\mathrm{fin}}(Var)$ *and* $e_1, e_2 \in \Lambda^{\oplus}(\overline{x})$, $\overline{x} \vdash e_1 \leq_{\mathrm{v}}^{\mathrm{ciu}} e_2$ *iff* $\overline{x} \vdash e_1 \leq_{\mathrm{v}} e_2$ *(and hence* $\overline{x} \vdash e_1 =_{\mathrm{v}}^{\mathrm{ciu}} e_2$ *iff* $\overline{x} \vdash e_1 =_{\mathrm{v}} e_2$*).*

Proof Since $\leq_{\mathrm{v}}^{\mathrm{ciu}}$ is reflexive, so is the $\lambda^{\oplus}$-term relation $(\leq_{\mathrm{v}}^{\mathrm{ciu}})^H$ and hence so is its extension to closed frame stacks. So we can take $s = s'$ in Lemma 5.8.6 and conclude that $\emptyset \vdash e\ (\leq_{\mathrm{v}}^{\mathrm{ciu}})^H\ e' \ \Rightarrow\ e \leq_{\mathrm{v}}^{\mathrm{ciu}} e'$. As we remarked before the lemma, this is enough to show that $(\leq_{\mathrm{v}}^{\mathrm{ciu}})^H = \leq_{\mathrm{v}}^{\mathrm{ciu}}$ and hence that $\leq_{\mathrm{v}}^{\mathrm{ciu}}$ is compatible. It is immediate from Lemma 5.8.3 that it satisfies the adequacy property (v-Adeq) from Remark 5.6.7. So $\leq_{\mathrm{v}}^{\mathrm{ciu}}$ is contained in the largest compatible adequate $\lambda^{\oplus}$-term relation:

$$\overline{x} \vdash e \leq_{\mathrm{v}}^{\mathrm{ciu}} e' \ \Rightarrow\ \overline{x} \vdash e \leq_{\mathrm{v}} e'. \tag{5.71}$$

To prove the converse of this, first note that since the contextual preorder is compatible, if $\emptyset \vdash e \leq_{\mathrm{v}} e'$ then $\emptyset \vdash E_s[e] \leq_{\mathrm{v}} E_s[e']$ (by induction on the length of the list s, using (5.47)); hence by adequacy of $\leq_{\mathrm{v}}$ and Lemma 5.8.3 we have

$$\emptyset \vdash e \leq_{\mathrm{v}} e' \ \Rightarrow\ \emptyset \vdash e \leq_{\mathrm{v}}^{\mathrm{ciu}} e'.$$

So if $\overline{x} \vdash e \leq_{\mathrm{v}} e'$, then by compatibility of $\leq_{\mathrm{v}}$ we have $\emptyset \vdash \lambda\overline{x}.\, e \leq_{\mathrm{v}} \lambda\overline{x}.\, e'$ and hence also $\emptyset \vdash \lambda\overline{x}.\, e \leq_{\mathrm{v}}^{\mathrm{ciu}} \lambda\overline{x}.\, e'$. Then supposing $\overline{x} = \{x_1, \ldots, x_n\}$, for any closed values $v_1, \ldots, v_n$, we can use the compatibility of $\leq_{\mathrm{v}}^{\mathrm{ciu}}$ (established in the first part of this proof) together with the validity of β_{v}-equivalence (Exercise 5.8.5) to get $\emptyset \vdash e\{v_1, \ldots, v_n/x_1, \ldots, x_n\} \leq_{\mathrm{v}}^{\mathrm{ciu}} e'\{v_1, \ldots, v_n/x_1, \ldots, x_n\}$. Therefore we do indeed have that $\overline{x} \vdash e \leq_{\mathrm{v}} e'$ implies $\overline{x} \vdash e \leq_{\mathrm{v}}^{\mathrm{ciu}} e'$. □

Remark 5.8.8 Theorem 5.8.7 serves to establish some basic properties of contextual equivalence for the call-by-value $\lambda^{\oplus}$-calculus (such as β_{v}-equivalence, via Exercise 5.8.5). Nevertheless, in this call-by-value setting its usefulness is somewhat limited.[6] This is because of the presence in frame stacks of evaluation frames of the form $(\lambda x.\, e)[\cdot]$. For closed values v we have $(\lambda x.\, e)v \Downarrow_{\mathrm{v}}$ iff $e\{v/x\} \Downarrow_{\mathrm{v}}$; so testing may-termination with respect to such evaluation frames is essentially the same as testing may-termination in an arbitrary context. In the call-by-name setting the CIU theorem gives a more useful characterisation of contextual equivalence; see Example 5.9.1. Nevertheless, such 'context

[6] In fact the properties of $=_{\mathrm{v}}$ for this calculus are rather subtle. For example, it is possible to show that $\lesssim_{\mathrm{v}} \cap \lesssim_{\mathrm{v}}^{op}$ is strictly contained in $=_{\mathrm{v}}$: the untyped nature of the calculus means there are expressions with unbounded non-deterministic behaviour (via the usual encoding of fixpoint recursion and arithmetic), thereby allowing one to encode Lassen's example separating mutual similarity from contextual equivalence [Las98b, pp. 72 and 90].

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

lemmas' cannot help us with deeper properties that a contextual equivalence may have (such as the extensionality and representation independence results considered in [Pit02]), where bisimilarities, logical relations and related notions come into their own. However, the weakness of CIU theorems is also their strength; because they only give a weak characterisation of contextual equivalence, they seem to hold (and are relatively easy to prove via Howe's method) in the presence of a very wide range of programming language features where other more subtle analyses are not easily available or are more troublesome to develop; see [PS08, appendix A], for example.

5.9 Call-by-name equivalences

In this section we consider briefly the call-by-name version of the results described so far. Apart from changing the evaluation relation and using arbitrary substitutions instead of value-substitutions, the essential features of Howe's method for proving congruence results remains the same.

Syntax We will use the $\lambda^{\oplus}$-calculus from Section 5.7, that is, the untyped λ-calculus extended with erratic choice, $\oplus$. Recall from there that a $\lambda^{\oplus}$-term relation is *compatible* if it satisfies (Com1)–(Com4); is a *precongruence* if it is compatible and satisfies (Tra); and is a *congruence* if it is a precongruence and satisfies (Sym). Howe's 'precongruence candidate' construction is similarly unchanged: it sends a $\lambda^{\oplus}$-term relation $\mathcal{R}$ to the $\lambda^{\oplus}$-term relation $\mathcal{R}^H$ inductively defined by rules (How1)–(How4).

Operational semantics *Call-by-name evaluation* of closed $\lambda^{\oplus}$-terms is the binary relation $\Downarrow_{\mathrm{n}} \subseteq \Lambda^{\oplus}(\emptyset) \times V^{\oplus}(\emptyset)$ inductively defined by the rules

$$\frac{}{v \Downarrow_{\mathrm{n}} v} \qquad \text{(Val)}$$

$$\frac{e_1 \Downarrow_{\mathrm{n}} \lambda x.\, e \qquad e\{e_2/x\} \Downarrow_{\mathrm{n}} v}{e_1\, e_2 \Downarrow_{\mathrm{n}} v} \qquad \text{(Cbn)}$$

$$\frac{e_i \Downarrow_{\mathrm{n}} v}{e_1 \oplus e_2 \Downarrow_{\mathrm{n}} v}. \quad \text{(i=1,2)} \qquad \text{(Ch)}$$

The *call-by-name transition relation*, $(s\ ,\ e) \to_{\mathrm{n}} (s'\ ,\ e')$, is given by

$$(s\ ,\ e\, e') \to_{\mathrm{n}} ([\cdot]e' :: s\ ,\ e) \qquad (\to_{\mathrm{n}}2)$$

$$(s\ ,\ e_1 \oplus e_2) \to_{\mathrm{n}} (s\ ,\ e_i) \qquad \text{for } i = 1, 2 \qquad (\to_{\mathrm{n}}3)$$

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

where now s ranges over *call-by-name frame stacks*

$$s ::= \mathtt{nil} \;\big|\; [\cdot]e :: s. \tag{5.72}$$

Defining

$$e{\Downarrow_{\mathrm{n}}} \stackrel{\text{def}}{=} \exists v \in V^{\oplus}(\emptyset).\ e \Downarrow_{\mathrm{n}} v \tag{5.73}$$

$$(s\ ,\ e){\downarrow_{\mathrm{n}}} \stackrel{\text{def}}{=} \exists v \in V^{\oplus}(\emptyset).\ (s\ ,\ e) \to_{\mathrm{n}} \cdots \to_{\mathrm{n}} (\mathtt{nil}\ ,\ v) \tag{5.74}$$

the analogue of Lemma 5.8.3 holds and in particular we have

$$e{\Downarrow_{\mathrm{n}}} \;\Leftrightarrow\; (\mathtt{nil}\ ,\ e){\downarrow_{\mathrm{n}}}. \tag{5.75}$$

Applicative (bi)similarity A relation $\mathcal{S} \subseteq \Lambda^{\oplus}(\emptyset) \times \Lambda^{\oplus}(\emptyset)$ is a *call-by-name applicative simulation* if for all $e_1, e_2 \in \Lambda^{\oplus}(\emptyset)$, $e_1\ \mathcal{S}\ e_2$ implies

$$e_1 \Downarrow_{\mathrm{n}} \lambda x.\, e_1' \;\Rightarrow\; \exists e_2'.\ e_2 \Downarrow_{\mathrm{n}} \lambda x.\, e_2' \;\wedge\; \forall e \in \Lambda^{\oplus}(\emptyset).\ e_1'\{e/x\}\ \mathcal{S}\ e_2'\{e/x\}. \tag{n-Sim}$$

We write $\lesssim_{\mathrm{n}}$ for the largest such relation and call it *call-by-name applicative similarity*. A relation $\mathcal{B} \subseteq \Lambda^{\oplus}(\emptyset) \times \Lambda^{\oplus}(\emptyset)$ is a call-by-name applicative *bisimulation* if both $\mathcal{B}$ and its reciprocal $\mathcal{B}^{op}$ are call-by-name applicative simulations. We write $\simeq_{\mathrm{n}}$ for the largest such relation and call it *call-by-name applicative bisimilarity*. We extend $\lesssim_{\mathrm{n}}$ and $\simeq_{\mathrm{n}}$ to $\lambda^{\oplus}$-term relations via closing substitutions:

$$\overline{x} \vdash e \lesssim_{\mathrm{n}} e' \stackrel{\text{def}}{=} \forall e_1, \ldots, e_n \in \Lambda^{\oplus}(\emptyset).\ e\{e_1, \ldots, e_n/x_1, \ldots, x_n\} \lesssim_{\mathrm{n}} e\{e_1, \ldots, e_n/x_1, \ldots, x_n\} \tag{5.76}$$

$$\overline{x} \vdash e \simeq_{\mathrm{n}} e' \stackrel{\text{def}}{=} \forall e_1, \ldots, e_n \in \Lambda^{\oplus}(\emptyset).\ e\{e_1, \ldots, e_n/x_1, \ldots, x_n\} \simeq_{\mathrm{n}} e\{e_1, \ldots, e_n/x_1, \ldots, x_n\} \tag{5.77}$$

(where $\overline{x} = \{x_1, \ldots, x_n\}$).

With these definitions, the proof that $\lesssim_{\mathrm{n}}$ is a precongruence and that $\simeq_{\mathrm{n}}$ is a congruence proceeds as before, by using Howe's method to show that these $\lambda^{\oplus}$-term relations are compatible. For $\mathcal{R} = \lesssim_{\mathrm{n}}$ or $\mathcal{R} = \simeq_{\mathrm{n}}$ one proves the key lemma

$$e_1 \Downarrow_{\mathrm{n}} \lambda x.\, e \;\wedge\; \emptyset \vdash e_1\ \mathcal{R}^H\ e_2 \;\Rightarrow\; \exists e'.\ e_2 \Downarrow_{\mathrm{n}} \lambda x.\, e' \;\wedge\; \{x\} \vdash e\ \mathcal{R}^H\ e' \tag{5.78}$$

by induction on the derivation of $e_1 \Downarrow_{\mathrm{n}} \lambda x.\, e$. In case $\mathcal{R} = \lesssim_{\mathrm{n}}$, this gives $\lesssim_{\mathrm{n}} = (\lesssim_{\mathrm{n}})^H$ and so $\lesssim_{\mathrm{n}}$ inherits the compatibility property from $(\lesssim_{\mathrm{n}})^H$. In case $\mathcal{R} = \simeq_{\mathrm{n}}$, one uses the transitive closure trick from Section 5.7 to deduce from

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

(5.78) that $\simeq_{\mathrm{n}} = ((\lesssim_{\mathrm{n}})^H)^+$ and so $\simeq_{\mathrm{n}}$ inherits the compatibility property from $((\simeq_{\mathrm{n}})^H)^+$.

Contextual equivalence Taking the relational approach outlined in Remark 5.6.7, we can define the *call-by-name contextual preorder*, $=_{\mathrm{n}}$, to be the largest $\lambda^{\oplus}$-term relation $\mathcal{R}$ that is both compatible and *adequate for call-by-name evaluation*:

$$\forall e.\, e' \in \Lambda^{\oplus}(\emptyset).\ \emptyset \vdash e\ \mathcal{R}\ e' \ \Rightarrow\ (e{\Downarrow_{\mathrm{n}}} \ \Rightarrow\ e'{\Downarrow_{\mathrm{n}}}) \qquad \text{(n-Adeq)}$$

(with the proviso that the existence of a largest such relation is not immediately obvious, but can be deduced as in the remark). *Call-by-name contextual equivalence* is the symmetrisation of this preorder: $=_{\mathrm{n}} \stackrel{\mathrm{def}}{=} \leq_{\mathrm{n}} \cap \leq_{\mathrm{n}}^{op}$.

CIU-equivalence Two closed $\lambda^{\oplus}$-terms are in the *call-by-name CIU-preorder*, $e \leq_{\mathrm{n}}^{\mathrm{ciu}} e'$, if for all call-by-name frame stacks s we have that $(s\ ,\ e)\downarrow_{\mathrm{n}}$ implies $(s\ ,\ e')\downarrow_{\mathrm{n}}$. This is extended to a $\lambda^{\oplus}$-term relation via closing substitutions:

$$\overline{x} \vdash e \leq_{\mathrm{n}}^{\mathrm{ciu}} e' \ \stackrel{\mathrm{def}}{=}\ \forall e_1, \ldots, e_n \in \Lambda^{\oplus}(\emptyset).\ e\{e_1, \ldots, e_n/x_1, \ldots, x_n\} \leq_{\mathrm{n}}^{\mathrm{ciu}} e'\{e_1, \ldots, e_n/x_1, \ldots, x_n\}$$

(where $\overline{x} = \{x_1, \ldots, x_n\}$). *Call-by-name CIU-equivalence* is the symmetrisation of this relation: $=_{\mathrm{n}}^{\mathrm{ciu}} \stackrel{\mathrm{def}}{=} \leq_{\mathrm{n}}^{\mathrm{ciu}} \cap (\leq_{\mathrm{n}}^{\mathrm{ciu}})^{op}$.

The proof that $\leq_{\mathrm{n}}^{\mathrm{ciu}}$ is compatible is via the key lemma

$$\forall s, e, s', e'.\ s\ (\leq_{\mathrm{n}}^{\mathrm{ciu}})^H\ s' \wedge \emptyset \vdash e\ (\leq_{\mathrm{n}}^{\mathrm{ciu}})^H\ e' \wedge (s\ ,\ e)\downarrow_{\mathrm{n}}^{(n)} \ \Rightarrow\ (s'\ ,\ e')\downarrow_{\mathrm{n}} \qquad (5.79)$$

which, like Lemma 5.8.6, is proved by induction on the number n of steps of transition in $(s\ ,\ e)\downarrow_{\mathrm{n}}^{(n)}$ and then by cases on the definition of the call-by-name transition relation $\rightarrow_{\mathrm{n}}$.

Compatibility of $\leq_{\mathrm{n}}^{\mathrm{ciu}}$ gives the call-by-name *CIU-theorem*:

$$\leq_{\mathrm{n}}^{\mathrm{ciu}} = \leq_{\mathrm{n}} \qquad \text{and} \qquad =_{\mathrm{n}}^{\mathrm{ciu}} = =_{\mathrm{n}} \qquad (5.80)$$

as in the proof of Theorem 5.8.7, except that one uses validity of β-equivalence

$$\overline{x} \vdash (\lambda x.\, e)e' =_{\mathrm{n}}^{\mathrm{ciu}} e\{e'/x\} \qquad (5.81)$$

rather than β_{v}-equivalence. Because call-by-name frame stacks have a rather simple form (they are just a list of closed terms waiting to be applied as arguments for a function), this CIU-theorem is more useful than the call-by-value version. The following example illustrates this.

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

Example 5.9.1 It is not hard to see that $\lambda x_1.\,\lambda x_2.\,(x_1 \oplus x_2) \not\lesssim_{\mathrm{n}} (\lambda x_1.\,\lambda x_2.\,x_1) \oplus (\lambda x_1.\,\lambda x_2.\,x_2)$. However, it is the case that

$$\lambda x_1.\,\lambda x_2.\,(x_1 \oplus x_2) \leq_{\mathrm{n}} (\lambda x_1.\,\lambda x_2.\,x_1) \oplus (\lambda x_1.\,\lambda x_2.\,x_2) \tag{5.82}$$

(indeed the two terms are contextually equivalent). This can be shown by appealing to the CIU-theorem and checking that

$$(s\ ,\ \lambda x_1.\,\lambda x_2.\,(x_1 \oplus x_2))\downarrow_{\mathrm{n}} \Rightarrow\ (s\ ,\ (\lambda x_1.\,\lambda x_2.\,x_1) \oplus (\lambda x_1.\,\lambda x_2.\,x_2))\downarrow_{\mathrm{n}} \tag{5.83}$$

holds for all call-by-name frame stacks s. For if $s = \mathtt{nil}$ or $s = [\cdot]e :: \mathtt{nil}$, then the right-hand side of the implication in (5.83) holds. Whereas if $s = [\cdot]e_1 :: [\cdot]e_2 :: s'$ is a frame stack of length two or more, then the first three transitions of each configuration are

$$\begin{aligned}
&(s\ ,\ \lambda x_1.\,\lambda x_2.\,(x_1 \oplus x_2)) \to_{\mathrm{n}} \cdot \to_{\mathrm{n}} (s'\ ,\ e_1 \oplus e_2) \to_{\mathrm{n}} (s'\ ,\ e_i)\\
&(s\ ,\ (\lambda x_1.\,\lambda x_2.\,x_1) \oplus (\lambda x_1.\,\lambda x_2.\,x_2)) \to_{\mathrm{n}} (s\ ,\ \lambda x_1.\,\lambda x_2.\,x_i) \to_{\mathrm{n}} \cdot \to_{\mathrm{n}} (s'\ ,\ e_i)
\end{aligned}$$

for $i = 1, 2$; and hence

$$\begin{aligned}
&(([\cdot]e_1 :: [\cdot]e_2 :: s'\ ,\ \lambda x_1.\,\lambda x_2.\,(x_1 \oplus x_2))\downarrow_{\mathrm{n}}\\
\Rightarrow\ &(s'\ ,\ e_1)\downarrow_{\mathrm{n}} \vee (s'\ ,\ e_2)\downarrow_{\mathrm{n}}\\
\Rightarrow\ &([\cdot]e_1 :: [\cdot]e_2 :: s'\ ,\ (\lambda x_1.\,\lambda x_2.\,x_1) \oplus (\lambda x_1.\,\lambda x_2.\,x_2))\downarrow_{\mathrm{n}}.
\end{aligned}$$

Remark 5.9.2 The relationship between the various call-by-name equivalences considered in this section is:

$$(\simeq_{\mathrm{n}}) \subsetneq (\lesssim_{\mathrm{n}} \cap \lesssim_{\mathrm{n}}^{op}) \subsetneq (=_{\mathrm{n}}) = (=_{\mathrm{n}}^{\mathrm{ciu}}).$$

The first inclusion is a consequence of the definition of applicative (bi)similarity; it is proper because of Example 5.7.1 (which works the same for call-by-name as for call-by-value). The second inclusion is a corollary of the compatibility property for $\lesssim_{\mathrm{n}}$; and Example 5.9.1 shows that it is strict.

Exercise 5.9.3 Show that unlike for call-by-name contextual equivalence, the terms $\lambda x_1.\,\lambda x_2.\,(x_1 \oplus x_2)$ and $(\lambda x_1.\,\lambda x_2.\,x_1) \oplus (\lambda x_1.\,\lambda x_2.\,x_2)$ are not call-by-value contextually equivalent. [Hint: consider the call-by-value evaluation behaviour of $t(\lambda x_1.\,\lambda x_2.\,(x_1 \oplus x_2))$ and $t((\lambda x_1.\,\lambda x_2.\,x_1) \oplus (\lambda x_1.\,\lambda x_2.\,x_2))$, where $t \stackrel{\mathrm{def}}{=} \lambda z.\,z\,z\,v_2\,v_2\,v_1\,v_1$ with v_1 and v_2 as in Example 5.7.1.]

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

5.10 Summary

Having seen Howe's method in action in a few different settings, let us summarise what it involves.

(1) One has a notion $\mathcal{R}$ of program equivalence or preordering that one wishes to show is compatible with the constructs of the programming language:

$$P_1 \ \mathcal{R} \ P_2 \ \Rightarrow \ C[P_1] \ \mathcal{R} \ C[P_2].$$

Higher-order features of the language make a direct proof difficult, often because any proof of compatibility requires a proof of a closely related substitutivity property

$$P_1 \ \mathcal{R} \ P_2 \ \Rightarrow \ P\{^{P_1}/_x\} \ \mathcal{R} \ P\{^{P_2}/_x\}$$

(maybe with some restriction as to what P_1 and P_2 range over).

(2) Howe's precongruence candidate construction builds a relation $\mathcal{R}^H$ that is easily seen to have the required compatibility and substitutivity properties. In the case when $\mathcal{R}$ is a preorder, we get the desired properties of $\mathcal{R}$ by showing $\mathcal{R} = \mathcal{R}^H$. In the case when $\mathcal{R}$ is an equivalence, we get the desired properties of $\mathcal{R}$ by showing that it is equal to the transitive closure $(\mathcal{R}^H)^+$ of the precongruence candidate. (Use of transitive closure overcomes the inherent asymmetry in the definition of $\mathcal{R}^H$.)

(3) The key step in proving $\mathcal{R} = \mathcal{R}^H$ or $\mathcal{R} = (\mathcal{R}^H)^+$ is to show that $\mathcal{R}^H$ is contained in $\mathcal{R}$. This involves a delicate proof by induction over the operational semantics of programs.

5.11 Assessment

Howe's method was originally developed to prove congruence properties of bisimilarities for untyped functional languages. The method has since been successfully applied to bisimilarities for higher-order languages with types [Gor95, Pit97b], objects [Gor98], local state [JR99] and concurrent communication [FHJ98, Bal98, GH05].

The transfer of the method from functions to process calculi has highlighted a couple of drawbacks of Howe's method, the first more important than the second.

First, the operational semantics of processes is usually specified in terms of transitions labelled with actions, rather than in terms of evaluation to canonical form. Evaluation-based applicative bisimilarities correspond to transition-based *delay bisimilarities* in which externally observable actions

https://doi.org/10.1017/CBO9780511792588.006 Published online by Cambridge University Press

are matched by actions preceded by a number of internal ('silent') actions; see [San12, chapter 4]. Howe's method works well for proving congruence properties for such delay bisimilarities; but it does not appear to work for the more common *weak bisimilarities* in which externally observable actions are matched by actions both preceded and followed by a number of internal actions. For these reasons Jeffrey and Rathke [JR00] advocate replacing Howe's method with the use of Sangiorgi's trigger semantics from the higher-order π-calculus [San96]; see also [JR05, appendix A]. Another more recent approach uses the notion of *environment bisimulation* introduced by Sangiorgi, Kobayashi and Sumii [SKS07, Sum09], who show how to obtain congruence (and 'bisimilarity up to') results for higher-order languages without the need to invoke Howe's method.

Secondly, the induction proof in step (3) of the method (that is, 'key lemmas' like Lemma 5.7.2) relies heavily on the syntax-directed nature of the definition of the precongruence candidate $\mathcal{R}^H$; this does not interact well with a quotient by a structural congruence that sometimes forms part of the operational semantics of processes; see [PR98], for example.

Despite these caveats, Howe's method is unusually robust compared with many techniques based on operational semantics, witnessed by the fact that it can be applied to prove congruence properties for a variety of different forms of program equivalence. We believe it deserves a place in the semanticist's tool kit.

Bibliography

[Abr90] S. Abramsky. The lazy λ-calculus. In D.A. Turner, editor, *Research Topics in Functional Programming*, chapter 4, pages 65–117. Addison Wesley, 1990.

[Abr91] S. Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, **51**:1–77, 1991.

[Bal98] M. Baldamus. *Semantics and logic of higher-order processes: Characterizing late context bisimulation*. PhD thesis, Fachbereich 13 – Informatik, Technishen Universität Berlin, 1998.

[CH07] K. Crary and R. Harper. Syntactic logical relations for polymorphic and recursive types. In L. Cardelli, M. Fiore, and G. Winskel, editors, *Computation, Meaning and Logic, Articles dedicated to Gordon Plotkin*, volume 172 of *Electronic Notes in Theoretical Computer Science*, pages 259–299. Elsevier, 2007.

[FH92] M. Felleisen and R. Hieb. The revised report on the syntactic theories of sequential control and state. *Theoretical Computer Science*, **103**:235–271, 1992.

[FHJ98] W. Ferreira, M. Hennessy, and A. Jeffrey. A theory of weak bisimulation for core CML. *Journal of Functional Programming*, **8**(5):447–491, 1998.

[GH05] J.C. Godskesen and T. Hildebrandt. Extending Howe's method to early bisimulations for typed mobile embedded resources with local names. In *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science 25th International Conference, Hyderabad, India, December 15–18, 2005. Proceedings*, volume 3821 of *Lecture Notes in Computer Science*, pages 140–151. Springer, 2005.

[Gor95] A.D. Gordon. Bisimilarity as a theory of functional programming. In *Eleventh Conference on the Mathematical Foundations of Programming Semantics, New Orleans, 1995*, volume 1 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1995.

[Gor98] A.D. Gordon. Operational equivalences for untyped and polymorphic object calculi. In Gordon and Pitts [GP98], pages 9–54.

[GP98] A.D. Gordon and A.M. Pitts, editors. *Higher Order Operational Techniques in Semantics*. Cambridge University Press, 1998.

[How89] D.J. Howe. Equality in lazy computation systems. In *4th Annual Symposium on Logic in Computer Science*, pages 198–203. IEEE Computer Society Press, 1989.

[How96] D.J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, **124**(2):103–112, 1996.

[JR99] A. Jeffrey and J. Rathke. Towards a theory of bisimulation for local names. In *14th Annual Symposium on Logic in Computer Science*, pages 56–66. IEEE Computer Society Press, 1999.

[JR00] A. Jeffrey and J. Rathke. A theory of bisimulation for a fragment of concurrent ML with local names. In *Proceedings of the 15th Annual Symposium on Logic in Computer Science*, pages 311–321. IEEE Computer Society Press, 2000.

[JR05] A. Jeffrey and J. Rathke. Contextual equivalence for higher-order π-calculus revisited. *Logical Methods in Computer Science*, **1**:(1), April 2005.

[Las98a] S.B. Lassen. Relational reasoning about contexts. In Gordon and Pitts [GP98], pages 91–135.

[Las98b] S.B. Lassen. *Relational reasoning about functions and nondeterminism*. PhD thesis, Department of Computer Science, University of Aarhus, 1998.

[Mil77] R. Milner. Fully abstract models of typed lambda-calculi. *Theoretical Computer Science*, **4**:1–22, 1977.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mor69] J. Morris. *Lambda calculus models of programming languages*. PhD thesis, MIT, 1969.

[MT91] I.A. Mason and C.L. Talcott. Equivalence in functional languages with effects. *Journal of Functional Programming*, **1**:287–327, 1991.

[MT92] I.A. Mason and C.L. Talcott. Inferring the equivalence of functional programs that mutate data. *Theoretical Computer Science*, **105**:167–215, 1992.

[Pie02] B.C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

[Pit97a] A.M. Pitts. A note on logical relations between semantics and syntax. *Logic Journal of the Interest Group in Pure and Applied Logics*, **5**(4):589–601, 1997.

[Pit97b] A.M. Pitts. Operationally-based theories of program equivalence. In P. Dybjer and A.M. Pitts, editors, *Semantics and Logics of Computation*, Publications of the Newton Institute, pages 241–298. Cambridge University Press, 1997.

[Pit02] A.M. Pitts. Operational semantics and program equivalence. In G. Barthe, P. Dybjer, and J. Saraiva, editors, *Applied Semantics, Advanced Lectures*, volume 2395 of *Lecture Notes in Computer Science, Tutorial*, pages 378–412. Springer, 2002. International Summer School, APPSEM 2000, Caminha, Portugal, September 9–15, 2000.

[PR98] A.M. Pitts and J.R.X. Ross. Process calculus based upon evaluation to committed form. *Theoretical Computer Science*, **195**:155–182, 1998.

[PS08] A.M. Pitts and M.R. Shinwell. Generative unbinding of names. *Logical Methods in Computer Science*, **4**(1:4):1–33, 2008.

[San96] D. Sangiorgi. Bisimulation for higher-order process calculi. *Information and Computation*, **131**(2):141–178, 1996.

[San12] D. Sangiorgi. *An Introduction to Bisimulation and Coinduction*. Cambridge University Press, 2012.

[SKS07] D. Sangiorgi, N. Kobayashi, and E. Sumii. Environmental bisimulations for higher-order languages. In *Twenty-Second Annual IEEE Symposium on Logic In Computer Science (LICS'07)*, pages 293–302. IEEE Computer Society Press, July 2007.

[Sum09] E. Sumii. A complete characterization of observational equivalence in polymorphic lambda-calculus with general references. In *Computer Science Logic, Proceedings of 18th EACSL Annual Conference (CSL 2009)*, volume 5771 of *Lecture Notes in Computer Science*, pages 455–469 Springer, 2009.

[Win93] G. Winskel. *The Formal Semantics of Programming Languages*. Foundations of Computing. The MIT Press, 1993.