Open Bisimulation for Aspects*

Radha Jagadeesan, Corin Pitcher, and James Riely

School of CTI, DePaul University, Chicago, IL 60604, USA
{rjagadeesan,cpitcher,jriely}@cs.depaul.edu

Abstract. We define bisimilarity for an aspect extension of the untyped lambda calculus and prove that it is sound and complete for contextual reasoning about programs. The language we study is very small, yet powerful enough to encode mutable references and a range of temporal pointcuts. We extend formal studies of Open Modules to this more general setting. Examples suggest that aspects are amenable to techniques developed for stateful higher-order programs. To our knowledge, this is the first study of coinductive reasoning principles for aspect programs.

1 Introduction

Aspects have emerged as a powerful tool in the design and development of systems [5, 10, 31, 32, 42, 53]. A standard example from the AspectJ tutorials suffices to introduce the basic vocabulary: Suppose class L realizes a useful library, and we want to obtain timing information about a method f of L. With aspects this can be done by writing *advice* specifying that, whenever f is called, the current time should be logged, f should be executed, and then the current time should again be logged. Aspects permit the profiling code to be localized in the advice, transferring the responsibility for coordinating the advice and base code to a compiler or runtime environment. In writing the logging advice, one must identify the pieces of code, using *pointcuts*, that need to be logged — in [20] this is called *quantification*. Furthermore, the developer of the library does not need explicit knowledge about advice that may be written in the future — this is called *obliviousness* in [20]. Aspect-orientation is paradigm-independent and has been realized in object-oriented [31, 65], imperative [15] and functional languages [18, 67].

Our focus in this paper is on the intersubstitutivity of programs written in an aspect-oriented extension of a functional language: when can one program fragment be substituted for another without altering the observable behavior of the program? A basic tool that has been used to address this question for other programming paradigms has been coinduction, in the form of bisimulation principles. While the origins of bisimulation trace back to concurrency theory (see [59, 60] for a comprehensive historical survey and detailed bibliography), bisimulation principles have proven to be quite useful to address program equality in several paradigms, e.g., higher-order languages (see [23, 55] for a detailed treatment with historical context), even in the presence of existential types [64] or state [29, 36], and object-oriented languages [24, 34].

* Based on extended abstract published in AOSD 07, March 12-16, 2007, Vancouver, Canada, 1-59593-615-7/07/03

This paper brings aspect-based languages within the ambit of this technique. Our formal techniques and results suggest that aspects are no more intractable than stateful higher-order programs. In first order languages with first order references, when reasoning about programs, the environment has only two ways to interact with a program: either via global shared variables or by invoking the program (that can of course result in changes in encapsulated private state of the program). In higher-order languages with higher-order references, a program can also "leak" local state externally via higher-order mechanisms providing the environment a third way to interact with a program. Our results suggest that mechanisms that address this feature of higher-order languages with state may be adapted to an aspect framework with dynamic aspects.

Bisimulation. We study a core untyped lambda calculus, enhanced with aspects and named functions. Advice is first class in our calculus: it can be created and added dynamically while a program is running. The language can code mutable higher-order references and expressive pointcuts such as cflow and regular event patterns.

We describe a bisimulation principle based on a labelled transition system for aspect programs. We show that bisimulation is sound and complete for contextual equivalence.

We demonstrate the usability of the bisimulation principle via examples using the encoding of mutable variables — we show that several of the program equalities suggested by Meyer and Sieber [46] are validated by our bisimulation principle.

Application to Open Modules. Aspect-Aware Interfaces [33] enhance the usual signature information of modules with the pointcuts that are exported by the module and visible to the clients of the module. This enhancement of traditional signatures facilitates extra reasoning by providing bounds on the use of advice. An Open Module [6] delineates conditions about when it is permissible to replace the implementation of a module with another.

The formal treatment of Open Modules [6] only permits call pointcuts, whereas the implementation of Open Modules in AspectJ [52] also permits cflow pointcuts. Recent research has explored more expressive pointcut languages, such as those which match the entire computation history using regular patterns [9] or nested word languages [7, 8]. We use our bisimulation principle to bridge this expressiveness gap.

Our core calculus supports mechanisms to delimit the scope of the program where a function can be advised. We do this by providing named primitive pointcuts. Each function and advice declaration is associated with a primitive pointcut. Advice applies to a function only if its associated primitive pointcut is the same as that of the function. We use normal scoping mechanisms to control the knowledge of primitive pointcuts. The use of named primitive pointcuts as a separate construct permits the scope of the "advise"-access to vary separately from the standard scope of direct access to the function reference.

This framework permits the use of our bisimulation principle to establish conditions under which implementations can be changed without affecting clients, even in the presence of dynamic aspects and an expressive collection of history-sensitive pointcuts.

Organization of this paper. After a discussion of related work, we present the core language in Section 3, including a definition of contextual equivalence and examples.

In Section 4, we describe the LTS and our notion of bisimulation; this section also contains examples that illustrate the use of the bisimulation. In Section 5, we state the foundational properties that hold. The bulk of the proofs are deferred to the appendices.

2 Related work

Core calculi for aspect-based languages have been explored in a variety of settings. For example, [13, 27] are based on class-oriented calculi; in [14], a parametric description of a wide range of aspect languages, is based on the object calculus [1]; and [56] integrates aspect and object-oriented languages. Our calculus builds on descriptions of aspects in higher-order functional languages [18, 67].

[69] provides a denotational semantics for a calculus with dynamic join points, pointcut designators and advice. Our focus is on operational reasoning and proof rules. We refer the reader to [36] for a comparison of the operational and denotational approaches to stateful higher-order languages.

[45] provide the semantics of dynamic join points by translating into a core functional language with simple matching features. Our approach complements this work by providing reasoning tools for a core functional language with aspects.

Formal static reasoning via type systems has been explored for functional [43] and object-oriented [28] aspect languages. Typing considerations are orthogonal to our primary focus, and we elide them to lighten the presentation.

Model-checking techniques have been explored to analyze the behavior of individual aspect programs [41, 62, 66]. Our paper is complementary to this research. Our study provides formal foundations for compositional proof principles that are of use in model-checking aspect programs. The utility of this approach is already suggested in [41].

There has also been research into facilitating reasoning by controlling obliviousness. For example, information flow methods have been used to create type systems that ensure that aspects do not affect the return value [16] — for some security applications, these superficially drastic sounding restrictions are appropriate. In this general spirit, albeit with less impact on obliviousness, the named primitive pointcuts of our calculus can be viewed as ways to control interference between aspects and between aspects and other code. Our primitive pointcuts are directly inspired by Open Modules [6] (see also [51]) and are a formal device to model some features of Aspect-Aware Interfaces [33]. There are two different views about where such names can originate: (a) as programming annotation, written by the programmer (a view arguably in tension with uninhibited obliviousness), or (b) a tool derived annotation, derived from an analysis of the context of the program. In this paper, we do not take a viewpoint on this debate; instead, we focus on the support to reasoning that is afforded by such annotations.

Broadly speaking, bisimulation approaches to higher-order languages fall into the following main categories, depending on the kinds of tests that are permitted.

The first approach is usually termed applicative bisimulation. Some of the historical landmarks on this route are the initial definition of applicative bisimulation for lazy lambda calculus [4], the presentation using a labelled transition system [22] and a general method to show that applicative bisimulation is a congruence [25]. In this approach,

two terms, say M_1 and M_2, that agree on convergence behavior are tested for bisimilarity by providing them identical arguments and testing the resulting computation ($M_1\ N$ and $M_2\ N$) coinductively for bisimilarity. Extensions to account for imperative features were developed in [29].

In the second approach, the tests are enhanced. So, two lambda terms (say M_1 and M_2) are tested by providing them arguments that are derived from identical contexts (say $D[\cdot]$) with holes filled by bisimilar terms (say N_1 and N_2) and testing the resulting computation ($M_1\ D[N_1]$ and $M_2\ D[N_2]$) coinductively for bisimilarity. The complexity and number of tests is controlled by restricting attention to value contexts, i.e., $D[\cdot]$ such that $D[N_1]$ and $D[N_2]$ are values. [64] develops this approach for a language including existential types; [36] develops this general framework for a higher-order language with imperative features. Class equivalences [35], and the object calculus [34] are also tackled by these methods. This general approach is now termed environmental bisimulation and its metatheory has been studied recently [61].

Our approach is inspired by open bisimulation [58], and ENF-bisimulation [38, 39]. In this approach, two lambda terms (say M_1 and M_2) are tested by providing them arguments that are symbolic names (say ϕ) and testing the resulting computation ($M_1\ \phi$ and $M_2\ \phi$) coinductively for bisimilarity. Any two terms with different symbols in the primary function position (say $\phi\ M$ and $\psi\ N$, where $\phi \neq \psi$ for arbitrary M and N) are considered different. Our approach to stateful programs is in particular closely related to the concurrently and independently developed treatment of sequential control and state [63] following this approach. Furthermore, the precise relationship of this style to game-theoretic semantics of programming languages [3, 26] has by now been formalized [40].

In comparison to applicative bisimulation, the more elementary congruence proofs of our approach suggest that our open-bisimulation based approach addresses stateful features more directly. In contrast to the environmental approaches to higher-order languages, our methods do not need to address the contextual closure of programs and equivalences of values in this closure. However, the price paid by our approach is the explicit maintenance of extra contexts and transitions for book-keeping mechanisms. We develop congruence results and bisimulation-upto results to lighten this burden. In Section 4, we present a detailed comparison of our definitions with the two approaches.

In summary, the examples in the paper suggest that our treatment is good enough to capture and formalize intuitions crystallized by observation of the source code. However, we do not have any results that support the (semi-)automatic derivation of witnessing relations. That investigation remains open to future study.

3 Language

Our calculus builds on descriptions of aspects in higher-order functional languages [18, 67]. Advice may be loaded dynamically; several recent aspect language implementations support such dynamic aspects, eg, [11]. Primitive pointcuts are named and scoped: a programmer may limit the scope over which a function is advisable by controlling the scope of the associated primitive pointcut. In this respect, our language has some of the expressiveness of the module language of [6], in a simpler setting. Each function

declaration is associated with a primitive pointcut and advice applies to a function only if its associated primitive pointcut is that of the function. One may view possession of the name of a function as a form of *read access* and possession of the primitive pointcut of a function as a form of *write access*. We formalize this intuition when encoding references in Example 7.

The language is an untyped lambda calculus extended with function declarations in the style of ML and with advice over declared functions. The difference between abstractions and declared functions can be detected contextually. For example, consider $(\lambda\_.0)$ and $(\textbf{fun}\ \mathsf{f@p}=\lambda\_.0;\mathsf{f})$, which declares f at primitive pointcut p and returns f. The first expression results immediately in an abstraction. The second results in the name f, which is only resolved to an abstraction when applied. The difference is observable when the primitive pointcut p is used to declare advice, as, for example, in the context $(\textbf{adv}\ \mathsf{p}=\lambda\_.1;[\text{–}]\ ())$; here [–] is the "hole" to be filled by a term. The context declares advice at p then applies the hole to the unit value; evaluation results in 0 when the hole is filled with $(\lambda\_.0)$, but 1 when filled with $(\textbf{fun}\ \mathsf{f@p}=\lambda\_.0;\mathsf{f})$. A function declared at a bound primitive pointcut is unadvisable outside the scope of the binder; thus, $(\lambda\_.0)$ and $(\textbf{pcd}\ \mathsf{p};\textbf{fun}\ \mathsf{f@p}=\lambda\_.0;\mathsf{f})$ are contextually indistinguishable.

In the rest of this section, we formalize the syntax (Subsection 3.1) and dynamics (Subsections 3.2 and 3.3) of this core calculus. Subsection 3.4 defines contextual equivalence. Subsection 3.5 provides simple examples to illustrate the definitions. Subsection 3.6 discusses Open Modules and temporal pointcuts.

3.1 Syntax

We divide names into two countably infinite and mutually disjoint sets: variables and primitive pointcuts. In this study, primitive pointcuts are second-class entities; we discuss the motivation for this decision in Example 11.

SYNTAX

$f,g,h,x,y,z,\phi,\psi,\theta$	Variable Names
p,q,r	Primitive Pointcut Descriptors
$A,B ::=$	Declarations
$\mathsf{pcd}\ p$	Primitive Pointcut Descriptor ($dn = \{p\}$)
$\mathsf{fun}\ f\mathsf{@}p=U$	Function ($dn = \{f\}$)
$\mathsf{adv}\ p=\lambda z.U$	Advice ($dn = \{\}$, z bound in U)
$U,V,W ::=$	Values
x	Variable
$\lambda x.M$	Abstraction (x bound in M)
$M,N,L ::=$	Terms
U	Value
$A;M$	Declaration ($dn(A)$ bound in M)
$\mathsf{let}\ x=M;N$	Sequence (x bound in N)
$U\ V$	Application

The name declared by a declaration is given by the function *dn*, defined in the syntax table above. We assume the usual notion of free names, recovered by the function *fn*. While recursive uses of f are not allowed in $\mathsf{fun}\ f@p = U$, this is not a limitation; see Example 6. We identify terms up to renaming of bound names and write $M[x := U]$ for the capture-avoiding substitution of U for x in M, and similarly for $M[p := q]$. Thus $\mathsf{pcd}\ p\,;\, M$ is identical to $\mathsf{pcd}\ q\,;\, M[p := q]$ for any $q \notin \mathit{fn}(M)$.

We use the following discipline for variable names, when feasible. (The distinctions, while useful in many cases, are blurred when discussing congruence.)

- z is used for *proceed variables* bound in the body of advice;
- x-y are used for variables bound in abstractions and let-expressions, other than as a proceed variable;
- f-h are used for variables bound by function declarations;
- ϕ-θ are used for free function variables.

Variables x-y are resolved, in the standard way, during evaluation (Subsection 3.3). Variables z and f-h are resolved during function lookup (Subsection 3.2). The variables ϕ-θ are unresolvable; these are used in the LTS semantics (Section 4).

In examples, we use the unit value $()$, booleans (tru and fls), integers and pairs of values. These represent the standard Church encodings [54] (where $()$ is any value); thus 0, tru and fst are encoded as the combinator $(\lambda x.\lambda y.x)$, fls and snd are $(\lambda x.\lambda y.y)$, and 1 is $(\lambda x.\lambda y.y\,x)$. The church encoded constants may not satisfy all the equations one would like; we use them only for parsimony. Primitive constants may be added to the definition of the labelled transition system in Section 4 in the standard way [21]. We also use other well-known combinators, such as the divergent term $\Omega \triangleq (\lambda x.x\,x)\,(\lambda x.x\,x)$.

We use syntax sugar for application in the style of Moggi [48]; for example, $(M\,N) \triangleq (\mathsf{let}\ x = M\,;\, \mathsf{let}\ y = N\,;\, x\,y)$, where $x \notin \mathit{fn}(N)$. We adopt the same convention for operators on booleans, naturals and pairs. We write $\_$ for a bound variable that does not occur free in its scope; we abbreviate $(\mathsf{let}\ \_ = M\,;\, N)$ as $(M\,;\, N)$ and $(\lambda\_.M)$ as $(\lambda\,.M)$.

In examples, we sometimes write $(\mathsf{fun}\ f = U)$ as shorthand for $(\mathsf{pcd}\ p\,;\, \mathsf{fun}\ f@p = U)$, when p is not of interest. We also occasionally write declarations as terms, with the meaning that A, as a term, abbreviates $(A\,;\, ())$.

3.2 Lookup

In this Subsection, we describe function lookup, which determines the body of an advised function from a declaration sequence (notation $\vec{A}(f) = U$). We write $\vec{A}$ for *declaration sequences*, with "$\cdot$" representing the empty sequence, and "$;$" the element separator. An *evaluation configuration* is a pair of a declaration sequence and a term, written $\vec{A}/M$. To motivate the formal definition of lookup, we first present a few examples.

Example 1. Let $\vec{\mathsf{A}}$ be defined as follows.

$$\begin{array}{lll} \vec{\mathsf{A}} = \mathbf{adv}\ \mathsf{p} = \lambda z.\mathsf{V}\,; & & \mathsf{V} = \lambda y.(z\,y) + 1 \\ \quad \mathbf{fun}\ \mathsf{f@p} = \mathsf{W}\,; & \text{where} & \mathsf{W} = \lambda\,.5 \\ \quad \mathbf{adv}\ \mathsf{p} = \lambda z.\mathsf{U} & & \mathsf{U} = \lambda x.(z\,x) * 3. \end{array}$$

When one looks up f in the context of $\vec{\mathsf{A}}$, the result is

$$\vec{\mathsf{A}}(\mathsf{f}) = \mathsf{U}\big[\mathsf{z} := \mathsf{V}[\mathsf{z} := \mathsf{W}]\big] = \lambda \mathsf{x}.((\lambda \mathsf{y}.((\lambda.5)\,\mathsf{y}) + 1)\,\mathsf{x}) * 3.$$

The top-level term is U: the last (or most recently) declared advice which effects f (via the primitive pointcut p). The proceed variable z of U is bound to the rest of the advice which effects f, in this case V. Substitutions layer in this way to the last piece of advice, which proceeds to the function body, in this case W.

Evaluation of f() proceeds as follows.

$$\begin{aligned} \cdot/\vec{\mathsf{A}}\,;\mathsf{f}() &\twoheadrightarrow \vec{\mathsf{A}}/((\lambda \mathsf{y}.((\lambda.5)\,\mathsf{y}) + 1)())*3 \\ &\twoheadrightarrow \vec{\mathsf{A}}/(((\lambda.5)())+1)*3 \\ &\twoheadrightarrow \vec{\mathsf{A}}/18. \end{aligned}$$

Lookup is a partial function on names. For example, using the declarations above, $\vec{\mathsf{A}}(\mathsf{g})$ is undefined, and thus the evaluation configuration $\vec{\mathsf{A}}/\mathsf{g}()$ is stuck. □

Example 2. Note that advice may ignore the definition of the underlying function or of other advice — both referenced via z. As an example, consider

$$\begin{array}{lll} \vec{\mathsf{B}} = \textbf{adv}\ \mathsf{p} = \lambda \mathsf{z}.\mathsf{V}\,; & & \mathsf{V} = \lambda.7 \\ \quad \textbf{fun}\ \mathsf{f@p} = \mathsf{W}\,; & \text{where} & \mathsf{W} = \lambda.5 \\ \quad \textbf{adv}\ \mathsf{p} = \lambda \mathsf{z}.\mathsf{U} & & \mathsf{U} = \lambda \mathsf{x}.(\mathsf{z}\,\mathsf{x}) * 3. \end{array}$$

In this case

$$\vec{\mathsf{B}}(\mathsf{f}) = \mathsf{U}\big[\mathsf{z} := \mathsf{V}[\mathsf{z} := \mathsf{W}]\big] = \lambda \mathsf{x}.((\lambda.7)\,\mathsf{x}) * 3$$

and evaluation of f() proceeds as follows.

$$\cdot/\vec{\mathsf{B}}\,;\mathsf{f}() \twoheadrightarrow \vec{\mathsf{B}}/((\lambda.7)())*3 \twoheadrightarrow \vec{\mathsf{B}}/21$$ □

Lookup is defined using two auxiliary functions: *body* and *advise*. Whereas we identify terms up to renaming of bound names, the same does not hold for names declared in a declaration sequence. (This treatment is motivated by the definition of *body*, by which a primitive pointcut may escape its scope.) Instead, we require that declaration sequences be *well formed*, ie, that each name be declared at most once. For example, the declaration sequence (**fun** f@p = U ; **fun** f@q = V) is not well formed.

Definition 3 (Well formedness). A declaration sequence "$\vec{A}\,;\,B$" is *well formed* if $dn(B)$ does not occur in $\vec{A}$ and $\vec{A}$ is well formed. The empty sequence is also well formed.

An evaluation configuration $\vec{A}/M$ is *well formed* if $\vec{A}$ is well formed. □

Note that in a well-formed evaluation configuration $\vec{A}/M$, there may be names that occur free in M that are not declared in $\vec{A}$ (cf. Example 1).

The partial function $body(f, \vec{A})$ is defined whenever f is declared in $\vec{A}$; when defined, *body* returns both the value of the function and the primitive pointcut at which f is declared in $\vec{A}$.

$$\begin{aligned} body(f, \cdot) &\triangleq \text{undefined} \\ body(f, \mathsf{pcd} \cdots\,;\vec{A}) &\triangleq body(f, \vec{A}) \end{aligned}$$

$$\begin{aligned}
\mathit{body}(f, \mathsf{fun}\ f@p = U\,;\vec{A}) &\triangleq \langle p, U\rangle \\
\mathit{body}(f, \mathsf{fun}\ g@p = U\,;\vec{A}) &\triangleq \mathit{body}(f,\vec{A}),\ \text{where } f \neq g \\
\mathit{body}(f, \mathsf{adv}\ \cdots\,;\vec{A}) &\triangleq \mathit{body}(f,\vec{A})
\end{aligned}$$

The total function $\mathit{advise}(p, U, \vec{A})$ returns a value that applies to U the advice declared in $\vec{A}$ for p.

$$\begin{aligned}
\mathit{advise}(p, U, \cdot) &\triangleq U \\
\mathit{advise}(p, U, \mathsf{pcd}\ q\,;\vec{A}) &\triangleq \mathit{advise}(p, U, \vec{A}),\ \text{where } p \neq q \\
\mathit{advise}(p, U, \mathsf{fun}\ \cdots\,;\vec{A}) &\triangleq \mathit{advise}(p, U, \vec{A}) \\
\mathit{advise}(p, U, \mathsf{adv}\ p = \lambda z.V\,;\vec{A}) &\triangleq \mathit{advise}(p, V[z := U], \vec{A}) \\
\mathit{advise}(p, U, \mathsf{adv}\ q = \lambda z.V\,;\vec{A}) &\triangleq \mathit{advise}(p, U, \vec{A}),\ \text{where } p \neq q
\end{aligned}$$

Finally, the partial function $\vec{A}(f)$ is defined as follows.

$$\vec{A}(f) \triangleq \begin{cases} \mathit{advise}(p, V, \vec{A}) & \text{if } \mathit{body}(f, \vec{A}) = \langle p, V\rangle \\ \text{undefined} & \text{otherwise} \end{cases}$$

3.3 Dynamics

Following [19], reduction is defined using *evaluation contexts*, defined as follows.

$$\mathscr{E}, \mathscr{F}, \mathscr{G} \ ::=\ [-] \mid \mathsf{let}\ x = \mathscr{E}\,;N$$

As usual, $[-]$ is the "hole" to be filled by a term. Reduction is defined inductively as a binary relation between well formed configurations, using four axiom schemas.

REDUCTION $(\vec{A}/M \to \vec{A}'/M')$

$\vec{A}/\mathscr{E}[B\,;M]$	$\to \vec{A}\,;B/\mathscr{E}[M]$	if $\mathit{dn}(B) \notin \mathit{dn}(\vec{A}) \cup \mathit{fn}(\mathscr{E})$
$\vec{A}/\mathscr{E}[\mathsf{let}\ x = U\,;N]$	$\to \vec{A}/\mathscr{E}\big[N[x := U]\big]$	
$\vec{A}/\mathscr{E}[f\ V]$	$\to \vec{A}/\mathscr{E}\big[U\ V\big]$	if $\vec{A}(f) = U$
$\vec{A}/\mathscr{E}[(\lambda x.M)\ V]$	$\to \vec{A}/\mathscr{E}\big[M[x := V]\big]$	

The first axiom is structural, regulating the scope of declarations. Recall that we allow renaming of bound variables in terms, but not declaration sequences. Since the set of names is infinite, evaluation configurations of the form $\vec{A}/\mathscr{E}[B\,;M]$ may always reduce, fixing a "fresh" name for $\mathit{dn}(B)$.

The axiom for sequencing is standard, reducing $\mathsf{let}\ x = M\,;N$ only when M is a value. The use of contexts makes structural rules unnecessary. For example, if $\vec{A}/\mathscr{E}[f\ V] \to \vec{A}/U\ V$, then $\vec{A}/\mathsf{let}\ x = \mathscr{E}[f\ V]\,;N \to \vec{A}/\mathsf{let}\ x = \mathscr{E}[U\ V]\,;N$ by choosing the context $\mathscr{E}' = \mathsf{let}\ x = \mathscr{E}\,;N$.

There are three possibilities for an application $\vec{A}/\mathscr{E}[U\ V]$: (1) If U is a function name f and $\vec{A}(f)$ is defined, then evaluation proceeds to $\vec{A}/\mathscr{E}[\vec{A}(f)\ V]$. (2) If U is an abstraction then evaluation proceeds call-by-value using U; this is the standard beta-reduction axiom. (3) Otherwise evaluation is stuck.

Write $\twoheadrightarrow$ for the reflexive transitive closure of $\to$.

Example 4. Consider the following evaluation configuration.

$$\cdot / \textbf{fun}\ \mathsf{id@p} = \lambda \mathsf{x}.\mathsf{x}; \textbf{adv}\ \mathsf{p} = \lambda \mathsf{z}.\lambda \mathsf{y}.\mathsf{z}\,\mathsf{z}\,\mathsf{y}; (\lambda \mathsf{f}.\mathsf{f}\ 5)\ \mathsf{id}.$$

Using the axiom for declarations twice this reduces to

$$\twoheadrightarrow \textbf{fun}\ \mathsf{id@p} = \lambda \mathsf{x}.\mathsf{x}; \textbf{adv}\ \mathsf{p} = \lambda \mathsf{z}.\lambda \mathsf{y}.\mathsf{z}\,\mathsf{z}\,\mathsf{y} / (\lambda \mathsf{f}.\mathsf{f}\ 5)\ \mathsf{id}$$

which the axiom for application further reduces to

$$\rightarrow \textbf{fun}\ \mathsf{id@p} = \lambda \mathsf{x}.\mathsf{x}; \textbf{adv}\ \mathsf{p} = \lambda \mathsf{z}.\lambda \mathsf{y}.\mathsf{z}\,\mathsf{z}\,\mathsf{y} / \mathsf{id}\ 5.$$

Note that id is treated as a pure name when passed as an argument; it is only resolved at the point of application, where the axioms for lookup and beta-reduction yield

$$\begin{array}{ll} & \textbf{fun}\ \mathsf{id@p} = \lambda \mathsf{x}.\mathsf{x}; \textbf{adv}\ \mathsf{p} = \lambda \mathsf{z}.\lambda \mathsf{y}.\mathsf{z}\,\mathsf{z}\,\mathsf{y} / (\lambda \mathsf{y}.(\lambda \mathsf{x}.\mathsf{x})\ (\lambda \mathsf{x}.\mathsf{x})\ \mathsf{y})\ 5 \\ \rightarrow & \textbf{fun}\ \mathsf{id@p} = \lambda \mathsf{x}.\mathsf{x}; \textbf{adv}\ \mathsf{p} = \lambda \mathsf{z}.\lambda \mathsf{y}.\mathsf{z}\,\mathsf{z}\,\mathsf{y} / (\lambda \mathsf{x}.\mathsf{x})\ (\lambda \mathsf{x}.\mathsf{x})\ 5 \\ \rightarrow & \textbf{fun}\ \mathsf{id@p} = \lambda \mathsf{x}.\mathsf{x}; \textbf{adv}\ \mathsf{p} = \lambda \mathsf{z}.\lambda \mathsf{y}.\mathsf{z}\,\mathsf{z}\,\mathsf{y} / (\lambda \mathsf{x}.\mathsf{x})\ 5 \\ \rightarrow & \textbf{fun}\ \mathsf{id@p} = \lambda \mathsf{x}.\mathsf{x}; \textbf{adv}\ \mathsf{p} = \lambda \mathsf{z}.\lambda \mathsf{y}.\mathsf{z}\,\mathsf{z}\,\mathsf{y} / 5. \end{array}$$

□

3.4 Contextual equivalence

Contextual equivalence is defined with respect to a primitive notion of observation; two terms are related if they yield the same observations in all contexts. Following [17, 30], we assume a distinguished function name signal and take a call to this function to be a primitive observation.

Definition 5. A *(general) context* is any term with a single hole:

$$\mathscr{C} ::= [\text{–}] \mid A; \mathscr{C} \mid \mathsf{let}\ x = \mathscr{C}; N \mid \mathsf{let}\ x = M; \mathscr{C} \mid \mathscr{C}\ N \mid M\ \mathscr{C}.$$

Write $M \not\downarrow$ if $M \twoheadrightarrow \mathscr{E}[\mathsf{signal}\ U]$ for some evaluation context $\mathscr{E}$ and value U. For terms M and N in which signal does not occur, define $M \leqq N$ if for every context $\mathscr{C}$,

$$\mathscr{C}[M] \not\downarrow \text{ implies } \mathscr{C}[N] \not\downarrow.$$

Two terms M and N are *contextually equivalent* ($M \equiv N$) if $M \leqq N$ and $N \leqq M$. □

For lambda calculi, the primitive observation for contextual equivalence is usually taken to be termination [49]. Because our language includes side-effects, however, a finer observation is necessary for completeness (Appendix E). For example there is no context which, on the basis of termination alone, can distinguish $((\textbf{adv}\ \mathsf{p} = \lambda.\lambda.1); \mathsf{f}(); \Omega)$ from $((\textbf{adv}\ \mathsf{p} = \lambda.\lambda.2); \mathsf{f}(); \Omega)$, since both terms are divergent. (Ω is defined toward the end of Subsection 3.1.) Using signal, these can be distinguished by the context

$$(\textbf{fun}\ \mathsf{g@p} = \lambda.0); (\textbf{fun}\ \mathsf{f} = \lambda.\textbf{if}\ \mathsf{g}() == 1\ \textbf{then}\ \mathsf{signal}()\ \textbf{else}\ \Omega); [\text{–}].$$

3.5 Simple examples

Example 6 (Recursive use of function names). Note that in a function definition $(\mathsf{fun}\ f@p = U)$, the function name f is not bound in U. Recursive uses of f in U may be accommodated by writing the definition as

$$(\mathsf{fun}\ f@p = V)\,;\,(\mathsf{adv}\ p = \lambda.U)$$

where V is "dummy" value; the advice on f does not proceed and therefore V is ignored.

For example, one definition of Ω is "poisonpill ()", where poisonpill is the divergent function "**fun** poisonpill = λ . poisonpill ()". This may be written in our language as

$$(\mathbf{fun}\ \mathsf{poisonpill@p} = \lambda.()\,);\,(\mathbf{adv}\ \mathsf{p} = \lambda.\lambda.\mathsf{poisonpill}\ ())$$

where we have chosen "λ . ()" as the dummy value. □

Example 7 (References). We show how to code ML-style references as syntax sugar in the language of terms. The example demonstrates the unsurprising fact that dynamically loaded advice is a form of mutability.

We model references as a pair of functions, where the first is used for reading and the second for writing; the first is locally advisable, whereas the second is not. If p and f do not occur free in U, then define the following.

$$\begin{aligned}\mathsf{ref}\ U &\triangleq \mathsf{pcd}\ p\,;\,(\mathsf{fun}\ f@p = \lambda.U)\,;\,(f,\,\lambda x.\mathsf{adv}\ p = \lambda.\lambda.x)\\ !U &\triangleq (\mathsf{fst}\ U)\,()\\ U := V &\triangleq (\mathsf{snd}\ U)\,V\,;\,()\end{aligned}$$

We can code the imperative factorial as

$$\begin{aligned}&\mathbf{fun}\ \mathsf{fac} = (\lambda x.(\mathbf{let}\ \mathsf{y} = \mathsf{ref}\ 1)\,;\,(\mathbf{fun}\ \mathsf{loop} = \mathsf{U})\,;\,\mathsf{loop}\,\mathsf{x}),\ \text{ where}\\ &\mathsf{U} = \lambda x.\mathbf{if}\ (\mathsf{x} \le 1)\ \mathbf{then}\ (!\mathsf{y})\ \mathbf{else}\ (\mathsf{y} := !\mathsf{y} * \mathsf{x};\,\mathsf{loop}\ (\mathsf{x}-1)).\end{aligned}$$

Eliding the definitions of fac, loop, and p, fac 2 evaluates as

$$\begin{aligned}\cdot/\mathsf{fac}\ 2 &\twoheadrightarrow \mathbf{fun}\ \mathsf{f@p} = \lambda.1/\mathsf{loop}\ 2\\ &\twoheadrightarrow \mathbf{fun}\ \mathsf{f@p} = \lambda.1\,;\,\mathbf{adv}\ \mathsf{p} = \lambda.2/\mathsf{loop}\ 1\\ &\twoheadrightarrow \mathbf{fun}\ \mathsf{f@p} = \lambda.1\,;\,\mathbf{adv}\ \mathsf{p} = \lambda.2/\mathsf{f}\,()\\ &\twoheadrightarrow \mathbf{fun}\ \mathsf{f@p} = \lambda.1\,;\,\mathbf{adv}\ \mathsf{p} = \lambda.2/2.\end{aligned}$$

The result is 2, as expected. □

Example 8 (Contexts may need to test a value more than once). It is important that a context may store a value and test it more than once. For example, the terms $(\lambda.0)$, which always returns 0, and

$$\mathbf{let}\ \mathsf{b} = \mathsf{ref}\ \mathsf{tru}\,;\,(\lambda.\mathbf{if}\ !\mathsf{b}\ \mathbf{then}\ (\mathsf{b} := \mathsf{fls}\,;\,0)\ \mathbf{else}\ 1),$$

which returns 0 exactly once, can be distinguished by the context

$$\mathbf{let}\ \mathsf{x} = [\text{–}]\,;\,\mathsf{x}()\,;\,\mathbf{if}\ \mathsf{x}() = 0\ \mathbf{then}\ \mathsf{signal}()\ \mathbf{else}\ \Omega.$$ □

Example 9 (Contexts can observe advice order). To show some of the subtleties of contextual reasoning, here is an example where a context inserts itself in the middle of an advice list.

$$\mathscr{E} = \textsf{let x} = [\_]\,\textsf{; adv p} = \lambda \textsf{z}.\textsf{V}\,\textsf{; x()}$$

Consider

$$\mathscr{E}[\textsf{fun f@p} = \lambda \textsf{x}.0\,\textsf{; adv p} = \lambda \textsf{z}.\textsf{U}_1\,\textsf{; } (\lambda\,.(\textsf{adv p} = \lambda \textsf{z}.\textsf{U}_2\,\textsf{; f 0}))]$$

which evaluates to

$$\cdots\,\textsf{; U}_2\big[\textsf{z} := \textsf{V}[\textsf{z} := \textsf{U}_1[\textsf{z} := \lambda \textsf{x}.0]]\big].$$

Here the context has inserted the advice V between two bits of user advice $\textsf{U}_2$ and $\textsf{U}_1$. Using $\textsf{V} = (\lambda \textsf{x}.\textbf{if } \textsf{x} = 1 \textbf{ then } \textsf{signal()} \textbf{ else } \Omega)$, the context can distinguish the following pairs of advice, which cannot be distinguished without advising p.

$$\begin{array}{ll} \textsf{U}_1 = \lambda \textsf{x}.\textsf{z}\,(\textsf{x}+0) & \textsf{U}_1' = \lambda \textsf{x}.\textsf{z}\,(\textsf{x}+1) \\ \textsf{U}_2 = \lambda \textsf{x}.\textsf{z}\,(\textsf{x}+1) & \textsf{U}_2' = \lambda \textsf{x}.\textsf{z}\,(\textsf{x}+0) \end{array}$$

When composed, both pairs of advice add 3 to the function's argument. The advice V observes the intermediate result of the computation. □

Example 10 (Indistinguishability of functions). Functions with the same body declared at the same primitive pointcut are indistinguishable. The following terms are contextually equivalent for any M.

$$\begin{array}{l} \textsf{fun f@p} = \lambda \textsf{x}.M\,\textsf{; fun g@p} = \lambda \textsf{x}.M\,\textsf{; (f, g)} \\ \textsf{fun h@p} = \lambda \textsf{x}.M\,\textsf{; (h, h)} \end{array}$$

One can prove the equivalence using the method presented in the Section 4. □

3.6 Open Modules and temporal pointcuts

In this Subsection, we consider encodings of Open Modules, as proposed by Aldrich [6]. Open Modules extend ML-style modules to support two methods for controlling aspects:

- a distinction between internal and external function calls — only external calls are advisable from outside the module; and
- explicit pointcut declaration in module interfaces — only declared pointcuts may be used externally.

The first feature is handled in the operational semantics of [6] by renaming the function and creating a fresh declaration of the original name to invoke it. This kind of renaming can be achieved in compilation; here, we write programs directly in the form such a compiler would produce.

The second feature is more subtle, and we address it in two ways.

- We provide distinct binders for functions and primitive pointcuts; these may be viewed respectively as read and write capabilities, which may be handled independently. We treat primitive pointcuts as second class, since they are intended to delimit the static scope of mutability.
- We allow dynamically loaded advice. In addition to encoding state (Example 7), dynamically loaded advice allows us to create expressive "pointcuts" and to communicate them selectively as abstractions (Examples 13 and 14).

Example 11 (Open Modules). To get a sense of our approach, consider a concrete example: a math module with one advisable function fac. Internal and external calls to fac are distinguished so that only external calls may be advised.

```
module type MATH = sig
  val fac : int → int
  pointcut pfac : int → int
end;;
module Math : MATH = struct
  let rec fac = fun n → if n<1 then 1 else n*fac(n-1)
  pointcut pfac = call(fac)
end;;
open Math;;
let main = fun _→ fac 5;;
```

We view the module as providing two functions: the first is fac itself; the second is the pointcut pfac. A call to pfac will place advice on external calls to fac. In a module system, the calls to pfac occur in the compiler, rather than at runtime, but this phase distinction is an implementation convenience rather than a necessity.

The example can be coded in our language as follows. (Recall from Subsection 3.1 that (**fun** f = U) is syntactic sugar for (**pcd** p ; **fun** f@p = U), where p is a fresh primitive pointcut descriptor.)

```
fun Math = λ.
  fun fac′ = λn.if n<1 then 1 else n*fac′(n-1);
  pcd pfac′;
  fun fac@pfac′ = fac′;
  (fac, λy.adv pfac′ = λz.λx.y z x);
let (fac, pfac) = Math ();
fun main = λ.fac 5
```

The functions fac and pfac, recovered from Math, correspond exactly to the functions provided by the module above. For example, to count the number of calls to fac using a reference c, one might proceed as follows.

```
let c = ref 0;
pfac (λz.λx.c := !c+1; z x)
```

□

Remark 12 (Modularity results). In the above example, whereas fac is publicly advisable, fac′ is private to Math. To see that internal calls to fac′ are unadvisable, note

that one could exchange the body of fac' given here with that from Example 7 and the result would be contextually equivalent to the original (compare with Section 5.2 of [6]). In fact the following general result holds. Let

$$\mathscr{C} = \textbf{pcd}\ \mathsf{p}\,;\ \textbf{fun}\ \mathsf{f@p} = [\text{–}]\,;\ \mathsf{f}$$
$$\mathscr{D} = \textbf{fun}\ \mathsf{g@q} = [\text{–}].$$

Then,

$$\mathscr{C}[\mathsf{U}] \equiv \mathscr{C}[\mathsf{V}] \text{ implies } \mathscr{D}\big[\mathscr{C}[\mathsf{U}]\big] \equiv \mathscr{D}\big[\mathscr{C}[\mathsf{V}]\big].$$

This follows immediately from the fact that $\equiv$ is a congruence (Section 5). This general result allows any function to be defined in such a way that external calls are advisable, while internal ones are not. The remarkable power of contextual reasoning guarantees that the internal body can be substituted with any locally equivalent body without effecting the overall observable behavior. □

The previous encoding can be extended to richer pointcut languages, while still maintaining the modularity results.

Example 13 (cflow). The AspectJ pointcut `call(f) && cflow(g)` detects calls to f in the context of a call to g. Such a pointcut is exported from the following module.

```
fun FcflowG = λ .
  pcd pf ; fun f@pf = ··· ;
  pcd pg ; fun g@pg = ··· ;
  let b = ref fls ; // call to g active
  adv pg = λz.λx.let b′ = !b ; b := tru ; let y = z x ; b := b′ ; y ;
  (f, g, λy.adv pf = λz.λx.if !b then y z x else z x) ;
let (f, g, pf_cflow_g) = FcflowG () ;
```

The local boolean reference b is used to record whether a call to g is active. Whenever g is called, the advice at pg sets b to tru, proceeds to the body of g, then resets b. Whenever f is called the advice at pf first checks b before proceeding to the body of f.

A user may advise "f in the context of g", by calling the pf_cflow_g with advice $\lambda z.\lambda x.\cdots$. However, no other pointcuts are exposed. This generalizes the technique of Aldrich, and indeed the congruence results (c.f. Remark 12) apply equally to such terms. □

Nested word languages [7, 8] are a subset of context free languages with good closure properties that capture sensitivity to both the call-stack (as in cflow) and other history (as in regular patterns [9]). Pointcuts based on nested word languages arise naturally in examples in security (access control) and document processing (XML transducers). Since the operational semantics of nested word languages pushes exactly one stack symbol upon reading a call symbol and pops exactly one stack symbol upon reading a return symbol, such pointcut languages are addressable by implementation methods developed for cflow and regular patterns. The next example illustrates the ingredients of a translation from temporal pointcuts specified via nested-word languages.

Example 14 (History-sensitive access control). Abadi and Fournet [2] argue for history-sensitive access control mechanisms more expressive than the stack inspection mechanisms found in Java and C#. For example, consider a policy stating that advice on a sensitive function rm (eg, for file deletion) should be executed only if an (untrusted) function un has never been invoked in the past, *and* no call to f is still active. This policy for an access control failure is specified as a nested word language over symbols drawn from calls to, and returns from, un, rm and f. Using EBNF syntax, let balanced and opencalls be defined as follows.

$$\begin{array}{l} \mathsf{balanced} ::= \big((\mathit{call}(.)\ \ \mathsf{balanced}\ \ \mathit{ret}(.))\big)* \\ \mathsf{opencalls} ::= \big(\mathsf{balanced} \mid \mathit{call}(.)\big)* \end{array}$$

The property of interest can then be written as

$$\big(\big(\underbrace{.*\ \mathit{call}(\mathsf{un})\ \ .*}_{\text{un called}}\big) \mid \big(\underbrace{\mathsf{opencalls}\ \ \mathit{call}(\mathsf{f})\ \ \mathsf{opencalls}}_{\mathit{call}(\mathsf{f})\ \text{active}}\big)\big)\ \ \mathit{call}(\mathsf{rm}).$$

Following Example 13, we can export a pointcut matching the negation of this property of the call history.

```
fun Hsac = λ .
  pcd pun ; fun un@pun = ··· ;
  pcd pf ; fun f@pf = ··· ;
  pcd prm ; fun rm@prm = ··· ;
  let b1 = ref fls ; // call to f active
  adv pf = λz.λx.let b′ = !b1 ; b1 := tru ; let y = z x ; b1 := b′ ; y ;
  let b2 = ref fls ; // call un occurred
  adv pun = λz.λx.b2 := tru ; z x ;
  (f, un, rm, λy.adv prm = λz.λx.if !b1 or !b2 then z x else y z x) ;
let (f, un, rm, pf_hsac) = Hsac () ;
```

Advice attached using pf_hsac applies only in the specified conditions, and no other pointcuts are exposed. Again, the congruence results (c.f. Remark 12) apply equally to such terms. □

3.7 Access control and type enforcement

We demonstrate how Type Enforcement (TE) [12, 68] policies — a form of history-sensitive mandatory access control popularized in the NSA's Security-Enhanced Linux (SELinux) [44] — can be encoded as temporal advice. TE policies associate types with code and other resources to be protected; henceforth we call these "TE types" to avoid confusion with the usual notion of type found in programming languages. Also, the runtime system associates a current TE type with running code, which determines its privileges: access control decisions are based upon the current TE type and the TE type associated with the resource being accessed. The current TE type evolves as new code is invoked, based upon (1) the current TE type, (2) the TE type associated with the

new code, (3) the TE policy, and (4) constraints imposed by the caller. The mechanism permits access control policies that are sensitive to the history of the code that has been executed and constraints imposed by that code.

Example 15 (Web server). As an example policy, consider a web-server permitted to listen on ports 80 and 8080 if run by a system administrator, but only upon port 8080 if executed by an ordinary user. In this scenario, access privileges depend on both the original identity (system administrator or user) and the code (the web-server) that is running.

To encode this policy, we allow the current TE type to range over $\{\mathsf{adm}, \mathsf{usr}, \mathsf{ws\_adm}, \mathsf{ws\_usr}\}$, the TE type for the web-server code is $\mathsf{ws\_exe}$, and the TE types associated to the ports are $\{\mathsf{port}_{80}, \mathsf{port}_{8080}\}$. Initially the current TE type is adm or usr, then when the web-server is executed, the policy causes the current TE type to change from adm to $\mathsf{ws\_adm}$, or from usr to $\mathsf{ws\_usr}$. In addition, the policy permits

- adm and usr to execute code of TE type $\mathsf{ws\_exe}$;
- $\mathsf{ws\_adm}$ to access ports of TE type $\mathsf{port}_{80}, \mathsf{port}_{8080}$;
- $\mathsf{ws\_usr}$ to access ports of TE type port_{8080}.

With this policy, the desired security property is a non-interference property, namely that the code running as usr cannot be influenced by new connections on port 80, even after executing other code. □

The TE mechanism can be implemented with advice, where protected resources are modeled as functions that can be advised. To define the advice, we require:

- A finite set of current TE types T and a finite set of TE types E for executable code, not necessarily disjoint.
- An "allow" relation $\mathit{allow} \subseteq T \times E \times T$ describes when code can execute/access a function and transition to a new TE type, i.e., if the current TE type is t then a function marked with TE code type e can be invoked successfully and transition to current TE type t' if $\mathit{allow}(t, e, t')$.
- An "automatic transition" map $\mathit{auto} : T \times E \to T$ describes TE type transitions that occur automatically when a new function is executed, i.e., if the current TE type is t and a function marked with TE code type e is invoked successfully, then it is executed with TE type $\mathit{auto}(t, e)$.
- A finite set of primitive pointcuts Q and a map $\mathit{type} : Q \to E$.

Although we do not do so here, it is straightforward to also incorporate non-automatic transitions (c.f. the SELinux function $\mathsf{setexeccon}$) that allow a caller to choose, subject to *allow*, a TE type other than the default "automatic" TE type.

We consider declarations $\mathsf{A}_{\mathsf{curr}}(t)$ representing a private variable curr storing the current TE type initialized with t. The scope of the private variable curr extends over advice A_q, one for each primitive pointcut q, which checks whether a call to a function at q is permitted and updates the current TE type before the call takes place. A free variable fail is invoked when an access control check fails. The coding for updating the current TE type uses the same strategy adopted for cflow in Example 13, i.e., the caller's current TE type is stored before proceeding, and restored afterwards.

$$\mathsf{A}_{\mathsf{curr}}(t) \triangleq \mathsf{pcd}\ p, \mathsf{fun}\ \mathsf{curr@}p = \lambda\,.\,t$$

$$
\begin{array}{rcl}
\vec{\mathsf{A}}_Q & \triangleq & (\mathsf{A}_q \mid q \in Q) \\
\mathsf{A}_q & \triangleq & \mathsf{adv}\ q = \lambda z.\lambda x.\mathsf{L}_{z,x,q} \\
\mathsf{L}_{z,x,q} & \triangleq & \mathsf{let\ next} = \mathit{auto}(\,!\mathsf{curr}, \mathit{type}(q))\,; \\
& & \quad \mathsf{if}\ \mathit{allow}(\,!\mathsf{curr}, \mathit{type}(q), \mathsf{next})\ \mathsf{then} \\
& & \quad\quad \mathsf{let\ prev} = \,!\mathsf{curr}\,;\ \mathsf{curr} := \mathsf{next}\,; \\
& & \quad\quad \mathsf{let\ y} = z\,x\,;\ \mathsf{curr} := \mathsf{prev}\,;\ \mathsf{y} \\
& & \quad \mathsf{else}\quad \mathsf{fail}\ ()
\end{array}
$$

With the TE policy described in Example 15, and using TE code types as primitive pointcuts, suppose we are given a function webserver@ws_exe that starts a webserver on a port given as an argument, and functions $\mathsf{listen}_{80}@\mathsf{port}_{80}$, $\mathsf{listen}_{8080}@\mathsf{port}_{8080}$ that create listening sockets on ports 80 and 8080 respectively. We have:

$$
\begin{array}{rcl}
T & \triangleq & \{\mathsf{adm}, \mathsf{usr}, \mathsf{ws\_adm}, \mathsf{ws\_usr}, \mathsf{sys}, \mathsf{port}_{80}, \mathsf{port}_{8080}, \} \\
E & \triangleq & \{\mathsf{ws\_exe}, \mathsf{port}_{80}, \mathsf{port}_{8080}\}
\end{array}
$$

With the *allow* relation specified by:

$$
\begin{array}{ll}
\mathit{allow}(\mathsf{adm}, \mathsf{ws\_exe}, \mathsf{ws\_adm}) & \mathit{allow}(\mathsf{usr}, \mathsf{ws\_exe}, \mathsf{ws\_usr}) \\
\mathit{allow}(\mathsf{ws\_adm}, \mathsf{port}_{80}, \mathsf{sys}) & \\
\mathit{allow}(\mathsf{ws\_adm}, \mathsf{port}_{8080}, \mathsf{sys}) & \mathit{allow}(\mathsf{ws\_usr}, \mathsf{port}_{8080}, \mathsf{sys})
\end{array}
$$

Here the *auto* type transitions are exactly those allowed by *allow*. In the general type enforcement model, *auto* type transitions need not be the same as those allowed by *allow*, because *allow* can include non-automatic transitions.

Now, in the absence of $\mathit{allow}(\mathsf{ws\_usr}, \mathsf{port}_{80}, \mathsf{sys})$, the advice implementing the TE policy prevents the web-server from accessing port 80 when invoked with a current TE type of usr, i.e., if webserver attempts to invoke listen_{80} in the following program, the advice implementing the TE policy will cause fail to be invoked instead, because the invocation of webserver will cause the current TE type to change to ws_usr.

$$\mathsf{A}_{\mathsf{curr}}(\mathsf{usr})\,;\ \vec{\mathsf{A}}_Q\,;\ \mathsf{webserver}\ (80)$$

In this example, we see that the body of $\mathsf{listen}_{80}@\mathsf{port}_{80}$ is irrelevant to computation beginning with TE type usr. To formalize this non-interference property, we first define reachability $\mathit{reach}(t,e)$ of a TE type e from a TE type t to be the least relation such that:

- $\exists t'.\ \mathit{allow}(t,e,t')$ implies $\mathit{reach}(t,e)$
- $\exists t', e'.\ \mathit{allow}(t,e',t')$ and $\mathit{reach}(t',e)$ implies $\mathit{reach}(t,e)$

Reachability $\mathit{reach}(t,q)$ of a primitive pointcut from a TE type t is then defined to hold exactly when $\mathit{reach}(t,\mathit{type}(q))$. In the example above, the TE code type port_{80} is not reachable from usr.

The desired non-interference property is that we can take a program that declares functions at public primitive pointcuts, impose advice for type enforcement on those public primitive pointcuts, then arbitarily change the bodies of functions declared at primitive pointcuts unreachable from the initial TE type without changing the behavior of the program. In this already long paper, we elide the treatment and formal proof of this property for our encoding.

4 Labeled transition system and bisimulation

In this section, we present the bisimilarity relation following the LTS style pioneered by Gordon [22, 23], in particular in the style of presentation of Jeffrey and Rathke for Concurrent ML [29]. In contrast to this prior work, our intuitions are guided by open bisimulation and address aspect features. The technical consequence of this difference is that our proof that bisimilarity is a congruence is a direct proof based on a direct analysis of substitutions rather than following these papers in being based on Sangiorgi [57] or Howe [25].

The rest of this section is organized as follows. In Subsection 4.1, we describe the ideas of our LTS for the restricted case of the pure untyped lambda calculus without aspects or declarations. This treatment of a familiar calculus is intended to motivate the use of symbolic functions and advice in the LTS and to provide core intuitions for the following Subsections. In Subsection 4.2 we adapt the operational semantics of earlier sections to deal with symbolic data such as functions and advice. In Subsection 4.3, we provide a description of the LTS for the full calculus, and follow with a definition of the bisimilarity relation in Subsection 4.4. Subsection 4.5 makes the intuitions of our model concrete by a series of examples.

4.1 An introduction to open bisimulation

In this Subsection, we provide an snapshot of our approach by briefly describing an LTS for the pure untyped call-by-value lambda calculus.

We briefly recall the LTS approach [22] to applicative bisimulation for the pure untyped call-by-value lambda calculus

- A non-value term M has a τ transition to M' if M reduces in one step to M'.
- A value U (eg. $\lambda x.M$) has a transition labeled U' to the application $U\ U'$.

Two terms are bisimilar if the associated transition systems are bisimilar, i.e., reduction of one term terminates iff reduction of the other term terminates, and, if the terms reduce to values, then the results of applying both values to the same value are again bisimilar.

Our approach is inspired by open bisimulation [58] and ENF-bisimulation [38, 39]. (The reader can view this Subsection, in isolation, as a presentation of ENF-bisimulation-upto-η using an LTS.) Following our conventions, we use ϕ and ψ for variables that occur free in terms.

- A non-value term M has a τ transition to M' if M reduces in one step to M'.
- Values U have transitions labeled $\mathsf{app}\ \phi$ (where ϕ is fresh) to the application $U\ \phi$.
- Terms can now be of the form $\mathscr{E}[\phi\ U]$, for some evaluation context $\mathscr{E}$, where ϕ is an uninterpreted symbol. These terms have additional transitions (similar to ENF-bisimulation [38, 39]):
  - A transition labeled $\mathsf{fcall}\ \phi$ to U.
  - Transitions labeled $\mathsf{ret}\ \psi$ to $\mathscr{E}[\psi]$ for a fresh environment variable ψ.

Again, two terms are bisimilar if the associated transition systems are bisimilar.

The fcall ϕ transitions ensure that if the application $\mathscr{E}[\phi\ U]$ is bisimilar to the application $\mathscr{E}'[\phi'\ U']$ then $\phi = \phi'$ and U is bisimilar to U'. Similarly, the ret ψ transitions ensure that if $\mathscr{E}[\phi\ U]$ is bisimilar to $\mathscr{E}'[\phi'\ U']$ then $\mathscr{E}[\psi]$ is bisimilar to $\mathscr{E}'[\psi]$.

We extend this approach to open bisimulation by adding the featurss required to accommodate state and symbolic advice.

4.2 Symbolic functions and symbolic advice

The LTS must allow functions and advice to be defined by the environment, influencing a term. To accommodate context functions, we need only extend our notion of well-formedness to allow occurrences of free variables representing these functions. As noted earlier, we use ϕ, ψ to indicate these free variables; we sometimes refer to these as *symbolic function names* because they are uninterpreted in the term.

To accommodate context advice, we assume a countably infinite set of *symbolic advice names*, α, β, disjoint from the sets of variable names and primitive pointcuts.

SYMBOLIC ADVICE

α, β	Symbolic Advice Names
$A, B ::= \cdots \mid \mathsf{adv}\ p = \alpha$	Symbolic Advice Declaration
$U, V, W ::= \cdots \mid \alpha\texttt{<}U\texttt{>}$	Symbolic Advice Call

A symbolic advice call α<U> binds the proceed variable for α to U. Symbolic advice call typically occurs with a further application to the argument of the function being advised. Thus α<U> V indicates that the context is executing advice α with proceed value U and argument V.

Note that if $A = \mathsf{fun}\ f\texttt{@}p\texttt{=}\phi$, then by our previous definition of lookup $\vec{A}(f) = \langle p, \phi \rangle$; thus no extensions are required to handle symbolic functions. For symbolic advice, we extend the definition of *advise* as follows.

$$advise(q, U, \mathsf{adv}\ p = \alpha\,;\vec{A}) \triangleq \begin{cases} advise(q, \alpha\texttt{<}U\texttt{>}, \vec{A}) & \text{if } p = q \\ advise(q, U, \vec{A}) & \text{otherwise} \end{cases}$$

Example 16 (Evaluation with symbolic names). Let

$$\vec{\mathsf{A}} = \textbf{pcd}\ \mathsf{p}\,;\ \textbf{fun}\ \mathsf{f@p} = \phi\,;\ \textbf{adv}\ \mathsf{p} = \alpha\,;\ \textbf{adv}\ \mathsf{p} = \lambda \mathsf{z}.\lambda \mathsf{x}.(\mathsf{z}\ \mathsf{x}) * 3.$$

Evaluation of `f()` proceeds as follows.

$$\begin{aligned} \vec{\mathsf{A}}/\mathsf{f}() &\twoheadrightarrow \vec{\mathsf{A}}/(\lambda \mathsf{x}.(\alpha\texttt{<}\phi\texttt{>}\ \mathsf{x}) * 3)() \\ &\twoheadrightarrow \vec{\mathsf{A}}/(\alpha\texttt{<}\phi\texttt{>}())*3 \end{aligned}$$

Evaluation is now stuck; intuitively, control is given to the context that defined α.

Note that if evaluation arrives at an application `f()`, then the result is ϕ; again evaluation is stuck, this time giving the context control through the undefined body of ϕ. □

The following special form of substitution is used in proofs.

Definition 17. Substitution of an abstraction for symbolic advice "$M[\alpha := \lambda z.U]$" is defined homorphically over terms, with the only interesting case being for calls to symbolic advice:

$$(\alpha\texttt{<}V\texttt{>})[\alpha := \lambda z.U] \triangleq U[z := (V[\alpha := \lambda z.U])] \qquad \square$$

4.3 The LTS

We now define the states of the LTS. For namespace management, these include a *symbol environment*, which binds all symbolic function and advice names, and a *symbol declaration*, which may declare primitive pointcuts, functions and advice.

LTS SYNTAX

$\mathbf{M},\mathbf{N} ::= \vec{A}/\vec{\mathscr{E}}/M/\vec{U}$	Configuration
$\Gamma ::= \cdot \mid \phi,\Gamma \mid \alpha,\Gamma$	Symbol Environment
$\Delta ::= \cdot \mid A,\Delta$	Symbol Declaration

In a configuration $\vec{A}/\vec{\mathscr{E}}/M/\vec{U}$, we refer to M as the *active term*.

With respect to evaluation configurations, the new elements are the list of contexts $\vec{\mathscr{E}}$ and the list of values $\vec{U}$. The contexts $\vec{\mathscr{E}}$ model the call stack: it will be used in a manner consistent with the stack discipline. The list $\vec{U}$ includes all values that have been released/leaked to the environment during evaluation of the term. Thus, the values in $\vec{U}$ are available for the environment to inspect and use. Formally, $\vec{U}$ is a way to account for the imperative/state features of the calculus. These modeling ideas follow prior research [29, 30, 36, 64].

We define the LTS relative to a *symbol environment* Γ and *symbol declaration* Δ. In Subsection 4.4, we will define bisimilarity as $\Gamma;\Delta \vdash \mathbf{M} \sim \mathbf{N}$. The symbol environment is used to manage names in the LTS, in particular to ensure two bisimilar terms may always make transitions with the same labels. The symbol declaration, likewise, ensures that both contexts in a bisimulation have the same observation power. (We describe how to derive an initial state from a term in Definition 20.)

The target symbol environment/declaration of a transition is determined by the source symbol environment/declaration and the label of the transition.

Definition 18 (LTS state). In a configuration $\vec{A}/\vec{\mathscr{E}}/M/\vec{U}$, $dn(\vec{A})$ are bound in $\vec{\mathscr{E}}/M/\vec{U}$. (The let binders in $\vec{\mathscr{E}}$ are not in scope in M or $\vec{U}$ and thus are not binding.)

A *state* of the LTS is a triple $\Gamma;\Delta \vdash \mathbf{M}$, where the names listed in Γ are bound in Δ and $\mathbf{M}$ and $dn(\Delta)$ are bound in $\mathbf{M}$. A state is *well formed* if no name occurs free, and no name is declared more than once in $\Gamma,\Delta,\vec{A}$. $\square$

By way of contrast with evaluation configurations, note that we require a well formed LTS state to be closed. In the sequel, we assume that all LTS states are well-formed.

Names introduced by the context appear on the left side of the turnstile. Advice is unnamed, and thus advice introduced by the context appears on the right side of the turnstile in $\vec{A}$. Declarations introduced dynamically by the context appear on the right side of the turnstile in $\vec{A}$. Thus $\vec{A}$ will include symbolic advice laid down by the

environment — so, in general, it can be an interleaving of definitions of advice placed by the term and by the context.

We now present the labels used in the LTS.

LTS LABELS

$\varkappa ::= \tau \mid \kappa$		All Labels
$\kappa ::=$		Visible Labels
	$\mathsf{fcall}\ \phi$	Term calls context function ϕ
	$\mathsf{acall}\ \alpha$	Term calls context advice α
	$\mathsf{ret}\ \phi$	Context returns to term with result ϕ ($dn = \{\phi\}$)
	$\mathsf{app}\ \phi$	Context calls term with argument ϕ ($dn = \{\phi\}$)
	put	Context saves value
	$\mathsf{get}\ i$	Context restores value
	$\mathsf{fun}\ f@p = \phi$	Context declares function ($dn = \{f, \phi\}$)
	$\mathsf{adv}\ p = \alpha$	Context declares advice ($dn = \{\alpha\}$)

In Gordon's terminology [22], the visible labels fcall and acall are *active* (representing actions initiated by the term), whereas the remaining visible labels are *passive* (representing actions initiated by the environment). The choice of labels is determined by the possibilities available to the context to interact with the term. The examples in this section show how the labels correspond precisely to such contexts — an informal argument that underlies the completeness theorem (Appendix E).

LTS

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U} \xrightarrow{\tau} \Gamma;\Delta \vdash \vec{B}/\vec{\mathscr{E}}/N/\vec{U} \quad \text{if } \Delta,\vec{A}/M \to \Delta,\vec{B}/N$$

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/\mathscr{F}[\phi\ V]/\vec{U} \xrightarrow{\mathsf{fcall}\ \phi} \Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}},\mathscr{F}/V/\vec{U}$$

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/\mathscr{F}[\alpha\langle V\rangle\ W]/\vec{U} \xrightarrow{\mathsf{acall}\ \alpha} \Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}},\mathscr{F}/W/\vec{U},V$$

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}},\mathscr{F}/V/\vec{U} \xrightarrow{\mathsf{ret}\ \phi} \Gamma,\phi;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/\mathscr{F}[\phi]/\vec{U}$$

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/V/\vec{U} \xrightarrow{\mathsf{app}\ \phi} \Gamma,\phi;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/V\ \phi/\vec{U}$$

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/V/\vec{U} \xrightarrow{\mathsf{put}} \Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/V/\vec{U},V$$

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/V/\vec{U} \xrightarrow{\mathsf{get}\ i} \Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/U_i/\vec{U} \quad \text{if } 1 \le i \le |\vec{U}|$$

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/V/\vec{U} \xrightarrow{\mathsf{fun}\ f@p=\phi} \Gamma,\phi;\Delta,\mathsf{fun}\ f@p = \phi \vdash \vec{A}/\vec{\mathscr{E}}/V/\vec{U},f \quad \text{if } p \in dn(\Delta)$$

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/V/\vec{U} \xrightarrow{\mathsf{adv}\ p=\alpha} \Gamma,\alpha;\Delta \vdash \vec{A}\,;\,\mathsf{adv}\ p = \alpha/\vec{\mathscr{E}}/V/\vec{U} \qquad \text{if } p \in dn(\Delta)$$

The fact that configurations must be well-formed ensures that, in the rules for ret and app, the name ϕ must be fresh (i.e., must not occur in $\Gamma \cup dn(\Delta) \cup dn(\vec{A})$; likewise for the names ϕ and f in the rule for fun and α in the rule for adv. Well-formedness also ensures that in the rules for fcall and acall, ϕ and α must occur in Γ.

The LTS has several significant properties:

Call-by-value invariant. The LTS rules enforce a call-by-value invariant. This is seen by noting that precedence is afforded to internal reductions of the term. So, all rules except the first three are applicable to a state $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U}$ only if M is a value.

Applicative tests. app ϕ performs applicative tests. Rather than providing a term as an argument for the applicative test, this rule provides a fresh symbolic argument ϕ.

Stack of evaluation contexts. In the pure lambda calculus setting of Subsection 4.1, the rules for fcall and ret reflect the absence of interference between the caller and the callee in a purely functional language — the testing of the evaluation context and the callee argument is done separately. Thus, there was no need to track the evaluation context in the LTS for the pure lambda calculus.

In contrast, the LTS for the full calculus has to permit the environment an opportunity to inspect the arguments before the term continues evaluation — this is meaningful for the full calculus because of state changes caused by the dynamic laying down of advice. This is done in our LTS by the use of the stack of evaluation contexts $\mathscr{E}$.

fcall ϕ pushes the current evaluation context into $\mathscr{E}$. The active term becomes the argument to the call V. The transition ret ϕ returns a symbolic value ϕ to the top evaluation frame, $\mathscr{F}$, of the stack $\vec{\mathscr{E}}, \mathscr{F}$ and moves it into the current-term position, popping $\mathscr{F}$ from the top of the stack. (This stack discipline would have to be liberalized to address a language with control operators.)

Note that calls to signal (from Subsection 3.4) are treated like any other call, and thus generate labels of the form fcall signal.

Symbolic advice tests. In the rule for acall, the argument V is added to the list of values that are available for the environment to inspect and use. As in the case for fcall, the active term is changed to the argument, in this case W.

Environment value tests. put and get enable the movement of values between $\vec{U}$, the list of values leaked to the environment, and the active position of the configuration. put permits an evaluated value to be saved for use by the environment. get permits the environment to interact with a saved argument by moving it into the active term position. This rule leaves a copy of the restored term in $\vec{U}$. The label on this rule carries the position i in $\vec{U}$ that is being restored. Conceptually, put and get ensure that $\vec{U}$ is closed under structural rules.

New name tests. The rules for fun and adv permit the environment to add new function names and new advice. The first rule is necessary for bookkeeping; it allows the context to create an unbounded number of new function names; new names are added to the list of values $\vec{U}$ to maintain the invariant that functions declared in Δ can be inspected by the environment. The second rule is needed for more than bookkeeping. Since the order of advice matters, the rule for adv $p = \alpha$ also has to insert it into the list of advice declarations being carried in $\vec{A}$.

4.4 Bisimulation

Define $\twoheadrightarrow$ to be the reflexive transitive closure of $\xrightarrow{\tau}$. On visible labels define the weak labeled transition relation $\overset{\kappa}{\Longrightarrow}$ as $\twoheadrightarrow\xrightarrow{\kappa}$.

Note in the definition of the LTS ($\Gamma;\Delta \vdash \mathbf{M} \xrightarrow{\varkappa} \Gamma';\Delta' \vdash \mathbf{M}'$), that the symbol environment and declaration in the residual ($\Gamma';\Delta'$) are uniquely determined by the initial state ($\Gamma;\Delta$) and label ($\varkappa$). This leads us to define bisimilarity as a family of relations between configurations, written $\Gamma;\Delta \vdash \mathbf{M} \sim \mathbf{N}$. It is technically convenient to require that bisimilar configurations have equal length lists of contexts and values. (Alternatively,

we could prove that these invariants hold for bisimulations derived from the initial configurations of Definition 20.)

Definition 19. We say that a configuration $\vec{A}/\vec{\mathscr{E}}/M/\vec{U}$ has sort $\langle \Gamma, \Delta, m, n\rangle$ if $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U}$ is well-formed, the length of $\vec{\mathscr{E}}$ is m, and the length of $\vec{U}$ is n.

We define similarity, $\lesssim$, as the largest family of $\langle \Gamma, \Delta, m, n\rangle$-indexed relations over configurations such that

$$\Gamma;\Delta \vdash \mathbf{M} \lesssim \mathbf{N} \text{ and } \Gamma;\Delta \vdash \mathbf{M} \xrightarrow{\kappa} \Gamma';\Delta' \vdash \mathbf{M}'$$

imply that for some $\mathbf{N}'$

$$\Gamma;\Delta \vdash \mathbf{N} \xrightarrow{\kappa} \Gamma';\Delta' \vdash \mathbf{N}' \text{ and } \Gamma';\Delta' \vdash \mathbf{M}' \lesssim \mathbf{N}'.$$

$\Gamma;\Delta$-*bisimilarity*, $\sim$ is defined as two way similarity:

$$\Gamma;\Delta \vdash \mathbf{M} \sim \mathbf{N} \text{ if } \Gamma;\Delta \vdash \mathbf{M} \lesssim \mathbf{N} \text{ and } \Gamma;\Delta \vdash \mathbf{N} \lesssim \mathbf{M}. \qquad \square$$

For a fixed sort, $\langle \Gamma, \Delta, m, n\rangle$-indexed relations over configurations form a lattice ordered by set inclusion. The product of all these lattices over all sorts yields a lattice structure on sort-indexed families of relations. Indexed-(bi)similarity can also be formalized as the greatest fixed point of a monotone operator on this lattice. As is standard, Definition 19 describes $\lesssim$ (resp. $\sim$) as the prefixed point of this monotone operator.

Relational composition of two sort-indexed families of relations is defined by pointwise composition. For example, let $\mathscr{R} = \{\mathscr{R}_{\langle \Gamma,\Delta,m,n\rangle}\}$ and $\mathscr{S} = \{\mathscr{S}_{\langle \Gamma,\Delta,m,n\rangle}\}$ then $\mathscr{R} \circ \mathscr{S}$ is the sort-indexed family $\{\mathscr{R}_{\langle \Gamma,\Delta,m,n\rangle} \circ \mathscr{S}_{\langle \Gamma,\Delta,m,n\rangle}\}$. Since the tests of open bisimilarity are only names, standard proofs for first order calculi [47] apply here to yield that $\lesssim$ (resp. $\sim$) is closed under composition. Thus, $\lesssim$ (resp. $\sim$) is a preorder (resp. an equivalence relation).

Bisimilarity is insensitive to the addition of irrelevant new names to Γ, i.e., if $\Gamma;\Delta \vdash \mathbf{M} \sim \mathbf{N}$ and $\Gamma' \cap \Gamma = \emptyset$, then $\Gamma,\Gamma';\Delta \vdash \mathbf{M} \sim \mathbf{N}$. Symmetrically, bisimilarity is also insensitive to the removal of irrelevant new names from Γ, i.e., names in Γ that are not free in the rest of the configuration can be removed. The proofs follow by noting that the names in Γ are only typing constraints; so, addition or removal of names does not alter the transition capabilities of a configuration.

Bisimilarity on configurations relates to terms as follows.

Definition 20. Write "$\Gamma;\Delta \vdash M \sim N$" if $\Gamma;\Delta \vdash (\cdot/\cdot/M/\vec{f}) \sim (\cdot/\cdot/N/\vec{f})$, where $\vec{f}$ are the function names bound in Δ, in declaration order.

Write "$M \sim N$" if $(\vec{\phi},\vec{\alpha});(\mathsf{pcd}\ \vec{p}\,;\ \mathsf{adv}\ \vec{p} = \vec{\alpha}) \vdash M \sim N$, where $fn(M,N) = \{\vec{\phi},\vec{p}\}$. $\square$

The function symbols $\vec{\phi}$ detect function calls by the term. The primitive pointcut declarations $\mathsf{pcd}\ \vec{p}$ bind the free primitive pointcuts in the term. The advice declarations $\mathsf{adv}\ \vec{p} = \vec{\alpha}$ detect any call to a new function declared at a visible primitive pointcut (by the term). Functions can be introduced by fun transitions to detect any new advice declared at a visible primitive pointcut (by the term).

4.5 Simple examples

The first examples show that bisimilarity yields a β_v, η_v theory, i.e., that the call-by-value versions of β- and η-equality preserve bisimilarity.

Example 21 (β_v preserves bisimilarity). A standard LTS proof shows that prefixing by τ preserves bisimilarity. So, since:

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/(\lambda x.M)\ U/\vec{U} \xrightarrow{\tau} \Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M[x := U]/\vec{U},$$

and there are no other transitions to consider, we have

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/(\lambda x.M)\ U/\vec{U} \sim \vec{A}/\vec{\mathscr{E}}/M[x := U]/\vec{U}.$$

Thus, β_v preserves bisimilarity. □

Example 22 (η_v preserves bisimilarity). η_v holds, i.e., in the case where x is not free in U, we have

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/U/\vec{U} \sim \vec{A}/\vec{\mathscr{E}}/\lambda x.Ux/\vec{U}.$$

The key case in establishing the above bisimulation is to note that the transition

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/U/\vec{U} \xrightarrow{\mathsf{app}\ \phi} \Gamma,\phi;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/U\ \phi/\vec{U}$$

on the LHS is matched by the following sequence from the RHS

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/\lambda x.Ux/\vec{U} \xrightarrow{\mathsf{app}\ \phi} \Gamma,\phi;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/(\lambda x.Ux)\ \phi/\vec{U}$$

and that

$$\Gamma,\phi;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/(\lambda x.Ux)\ \phi/\vec{U} \xrightarrow{\tau} \Gamma,\phi;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/U\ \phi/\vec{U}.$$ □

The other cases are addressed by a simple bisimulation relation that relates configurations that are identical except that the occurrences of U in the active term and the value list can be replaced by $(\lambda x.Ux$.

Bisimilarity is not a trivial relation: for example, it distinguishes the Church booleans from one another, and likewise the Church numerals.

Example 23 (Bisimulation is sensitive to advice order). We revisit Example 9, noting that M and M' are distinguished by bisimilarity, where

$$\mathsf{M} = \textbf{fun}\ \mathsf{f@p} = \lambda \mathsf{x}.0\,;\ \textbf{adv}\ \mathsf{p} = \lambda \mathsf{z}.\mathsf{U}_1\,;\ (\lambda\,.(\textbf{adv}\ \mathsf{p} = \lambda \mathsf{z}.\mathsf{U}_2\,;\ \mathsf{f}\ 0))$$
$$\mathsf{M}' = \textbf{fun}\ \mathsf{f@p} = \lambda \mathsf{x}.0\,;\ \textbf{adv}\ \mathsf{p} = \lambda \mathsf{z}.\mathsf{U}'_1\,;\ (\lambda\,.(\textbf{adv}\ \mathsf{p} = \lambda \mathsf{z}.\mathsf{U}'_2\,;\ \mathsf{f}\ 0))$$

and

$$\mathsf{U}_1 = \lambda \mathsf{x}.\mathsf{z}\ (\mathsf{x}+0) \qquad \mathsf{U}'_1 = \lambda \mathsf{x}.\mathsf{z}\ (\mathsf{x}+1)$$
$$\mathsf{U}_2 = \lambda \mathsf{x}.\mathsf{z}\ (\mathsf{x}+1) \qquad \mathsf{U}'_2 = \lambda \mathsf{x}.\mathsf{z}\ (\mathsf{x}+0).$$

The term M can perform the following transitions in the LTS, whereas M' can make all transitions but the last one. The transitions correspond to the context presented in Example 9.

τ	*Term places advice* U_1 *and returns* $(\lambda.(\mathbf{adv}\ \mathsf{p} = \lambda \mathsf{z}.\mathsf{U}_2\,;\mathsf{f}\ 0))$
$\mathsf{adv}\ \mathsf{p} = \alpha$	*Context defines advice*
$\mathsf{app}\ \phi$	*Context calls* $(\lambda.(\mathbf{adv}\ \mathsf{p} = \lambda \mathsf{z}.\mathsf{U}_2\,;\mathsf{f}\ 0))$ *with ignored argument* ϕ
τ^*	*Term places advice* U_2 *and calls* f *with argument* 0
$\mathsf{acall}\ \alpha$	*Term calls* α *with argument with argument* 1
$\mathsf{app}\ \psi_{\mathsf{succ}}$	*Context provides first argument to church encoding of* 1
$\mathsf{app}\ \psi_{\mathsf{zero}}$	*Context provides second argument to church encoding of* 1
$\mathsf{fcall}\ \psi_{\mathsf{succ}}$	*Church encoding of* 1 *identifies itself by calling back to* ψ_{succ} □

As demonstrated in Example 21, the order of evaluation and multiplicity of use of "internal" functions are not necessarily detectable. Bisimilarity can, however, detect the order and multiplicity of calls to symbolic functions created by the environment. This corresponds to the opponent (the context) having state.

Example 24 (Detecting order). Consider the following terms.

$$\mathbf{let}\ \mathsf{x} = \phi\,()\,;\ \mathbf{let}\ \mathsf{y} = \psi\,()\,;\ ()$$
$$\mathbf{let}\ \mathsf{y} = \psi\,()\,;\ \mathbf{let}\ \mathsf{x} = \phi\,()\,;\ ()$$

The LTSs for these terms are immediately distinguished by the initially enabled transition, namely $\mathsf{fcall}\ \phi$ for the first term and $\mathsf{fcall}\ \psi$ for the second. □

Example 25 (Detecting multiplicity). Consider the following terms.

$$\mathbf{let}\ \mathsf{x} = \phi\,()\,;\ \mathbf{let}\ \mathsf{y} = \phi\,()\,;\ ()$$
$$\mathbf{let}\ \mathsf{y} = \phi\,()\,;\ ()$$

The LTSs for the first term may perform the following sequence of transitions: $\mathsf{fcall}\ \phi$, $\mathsf{ret}\ \psi$, $\mathsf{fcall}\ \phi$. The second term can match the first two of these transitions, but not the third. □

The distinctions made in Examples 24 and 25 (which are necessary in the full language with imperative features) hold even if all of the terms involved are purely functional, i.e., have no aspects.

Example 26 (The use of get and put rules). Consider the following terms.

$$\mathsf{M} = \vec{\mathsf{A}}\,;\mathsf{U} \qquad \vec{\mathsf{A}} = \mathbf{pcd}\ \mathsf{p}\,;\ \mathbf{fun}\ \mathsf{f@p} = \lambda.\mathsf{fls}\,;$$
$$\mathsf{V} = \lambda.\mathsf{tru} \qquad \mathsf{U} = \lambda.\mathbf{let}\ \mathsf{x} = \mathbf{not}(\mathsf{f}\ ())\,;(\mathbf{adv}\ \mathsf{p} = \lambda.\lambda.\mathsf{x})\,;\mathsf{x}$$

V is a function that always returns tru, whereas U (the function returned by M) will alternately return tru and fls, because of the state changes caused by the aspect in U. The terms can be distinguished by the context

$$\mathscr{E} = \mathbf{let}\ \mathsf{y} = [\_]\,;\mathsf{y}\,()\,;\mathsf{y}\,()$$

since $\mathscr{E}[\mathsf{V}]$ yields tru and $\mathscr{E}[\mathsf{M}]$ yields fls.

Clearly, this distinction relies crucially on the use of U twice. In the following example, we essentially show that the LTS is expressive enough to code the distinguishing context $\mathscr{E}$ by using $\mathsf{put}, \mathsf{get}$ tests to permit multiple tests of terms.

Using the definitions above, the behavior of $\mathscr{E}$ can be simulated in the LTS using the $\mathsf{put}, \mathsf{get}$ rules as follows. Consider the initial configuration $\cdot;\cdot \vdash \cdot/\cdot/\mathsf{M}/\cdot$, which has τ transitions to $\cdot;\cdot \vdash \vec{\mathsf{A}}/\cdot/\mathsf{U}/\cdot$. This configuration in turn has a put labeled transition to

$$\cdot;\cdot \vdash \vec{\mathsf{A}}/\cdot/\mathsf{U}/\mathsf{U},$$

which in turn has an $\mathsf{app}\ \phi$ labeled transition to

$$\phi;\cdot \vdash \vec{\mathsf{A}}/\cdot/\mathsf{U}\ \phi/\mathsf{U}.$$

A few τ transitions from this configuration yields

$$\phi;\cdot \vdash \vec{\mathsf{A}}\,;\, \mathbf{adv}\ \mathsf{f} = \lambda\,.\,\lambda\,.\,\mathsf{tru}/\cdot/\mathsf{tru}/\mathsf{U}.$$

To reevaluate U, we use a $\mathsf{get}\ 1$ transition to get

$$\phi;\cdot \vdash \vec{\mathsf{A}}\,;\, \mathbf{adv}\ \mathsf{f} = \lambda\,.\,\lambda\,.\,\mathsf{tru}/\cdot/\mathsf{U}/\mathsf{U}.$$

An $\mathsf{app}\ \psi$ labeled transition yields

$$\phi, \psi;\cdot \vdash \vec{\mathsf{A}}\,;\, \mathbf{adv}\ \mathsf{f} = \lambda\,.\,\lambda\,.\,\mathsf{tru}/\cdot/\mathsf{U}\ \psi/\mathsf{U}.$$

This second evaluation of U takes place in the context of the aspect that has been laid down. A few τ transitions from this configuration yields

$$\phi, \psi;\cdot \vdash \vec{\mathsf{A}}\,;\, \mathbf{adv}\ \mathsf{f} = \lambda\,.\,\lambda\,.\,\mathsf{tru}\,;\, \mathbf{adv}\ \mathsf{f} = \lambda\,.\,\lambda\,.\,\mathsf{fls}/\cdot/\mathsf{fls}/\mathsf{U}. \qquad \square$$

Much of the related work is formalized in terms of references, rather than advisable functions. In the next example, we discuss some of the subtleties, using the work of Meyer and Sieber [46] as the basis for comparison.

Example 27 (Primitive references versus advisable functions). For a free reference variable x, Meyer-Sieber [46] validate the equivalence $\mathsf{!x}\,;\,\mathsf{!x} \overset{\text{MS}}{=} \mathsf{!x}$. In our encoding of references, this translates roughly to the *in*equivalence demonstrated in Example 25. The difference arises from the weak assumptions one can make about functions relative to references; indeed the equivalence is valid in our language for *bound* references, where stronger assumptions are manifest:

$$\mathbf{let}\ \mathsf{x} = \mathsf{ref}\ 0\,;\,\mathsf{!x}\,;\,\mathsf{!x} \sim \mathbf{let}\ \mathsf{x} = \mathsf{ref}\ 0\,;\,\mathsf{!x}.$$

Unwinding the definition of references, this is roughly

$$\mathbf{pcd}\ \mathsf{p}\,;\,\mathbf{fun}\ \mathsf{f@p} = \lambda\,.0\,;\,\mathsf{f}()\,;\,\mathsf{f}() \sim \mathbf{pcd}\ \mathsf{p}\,;\,\mathbf{fun}\ \mathsf{f@p} = \lambda\,.0\,;\,\mathsf{f}().$$

But the equivalence does not hold when p is available to the context, since calls to f are then observable. Let $\Delta = \mathbf{pcd}\ \mathsf{p}, \mathbf{fun}\ \mathsf{f@p} = \lambda\,.0$. Then

$$\cdot;\Delta \vdash \mathsf{f}()\,;\,\mathsf{f}() \not\sim \mathsf{f}().$$

Interestingly, the equivalence does hold after an assignment, i.e., declaration of non-proceeding advice. Let $\mathsf{A} = \textbf{adv}\ \mathsf{p} = \lambda.\lambda.1$, then

$$\cdot;\Delta \vdash \mathsf{A};\mathsf{f}();\mathsf{f}() \sim \mathsf{A};\mathsf{f}()$$

which corresponds to $(\mathsf{x}:=1;\ !\mathsf{x};\ !\mathsf{x}) \overset{\text{MS}}{=} (\mathsf{x}:=1;\ !\mathsf{x})$.

Note also that for pure references $!\mathsf{x};\Omega \overset{\text{MS}}{=} \Omega$, whereas the corresponding result for functions does not hold: $\mathsf{f}();\Omega \overset{\text{MS}}{\neq} \Omega$. □

4.6 A reasoning principle

To simplify reasoning about bisimilarity, we develop an upto principle that eliminates the need to

- replicate values in bisimulations, eg, arising from a get 1 then a put transition.
- include terms that do not interact with the state, if they occur in the same position on each side of the bisimulation, and to

The following definition formalizes the above notion of replicated values.

Definition 28. $\mathscr{R}^{\bullet}_{\mathtt{dup}}$ is the least relation such that $\mathscr{R} \subseteq \mathscr{R}^{\bullet}_{\mathtt{dup}}$ and if

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/U,\vec{U}\ \ \mathscr{R}^{\bullet}_{\mathtt{dup}}\ \ \vec{B}/\vec{\mathscr{F}}/N/V,\vec{V}$$

then

$$\Gamma,\Gamma';\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/U,\vec{U},U\ \ \mathscr{R}^{\bullet}_{\mathtt{dup}}\ \ \vec{B}/\vec{\mathscr{F}}/N/V,\vec{V},V, \text{ and}$$
$$\Gamma,\Gamma';\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U}\ \ \mathscr{R}^{\bullet}_{\mathtt{dup}}\ \ \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$$ □

We say that a term (resp. evaluation context) is *state-free* over a symbol environment $(\Gamma;\Delta)$ if every free name is contained in Γ and the term (resp. evaluation context) contains *no* declaration subterms. The following definition formalizes the addition of identical state-free evaluation contexts (or values) to a relation on configurations.

Definition 29. $\mathscr{R}^{\bullet}_{\mathtt{sf}}$ is the least relation such that $\mathscr{R} \subseteq \mathscr{R}^{\bullet}_{\mathtt{sf}}$ and, for L (resp. W, $\mathscr{E}$) a state-free term (resp. value, context) for $(\Gamma,\Gamma';\Delta)$, we have that if

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U}\ \ \mathscr{R}^{\bullet}_{\mathtt{sf}}\ \ \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$$

then the following hold.

$$\Gamma,\Gamma';\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/W,\vec{U}\ \ \mathscr{R}^{\bullet}_{\mathtt{sf}}\ \ \vec{B}/\vec{\mathscr{F}}/N/W,\vec{V}$$
$$\Gamma,\Gamma';\Delta \vdash \vec{A}/\mathscr{E},\vec{\mathscr{E}}/M/\vec{U}\ \ \mathscr{R}^{\bullet}_{\mathtt{sf}}\ \ \vec{B}/\mathscr{E},\vec{\mathscr{F}}/N/\vec{V}$$ □

Moreover, if M and N are values:

$$\Gamma,\Gamma';\Delta \vdash \vec{A}/\vec{\mathscr{E}}/L/\vec{U}\ \ \mathscr{R}^{\bullet}_{\mathtt{sf}}\ \ \vec{B}/\vec{\mathscr{F}}/L/\vec{V}$$

Let $\mathscr{R}^{\bullet} = \mathscr{R}^{\bullet}_{\mathtt{dup}} \cup \mathscr{R}^{\bullet}_{\mathtt{sf}}$. Let $\twoheadleftarrow\!\!\twoheadrightarrow$ be the reflexive, transitive closure of the least symmetric relation containing $\xrightarrow{\tau}$. The following upto-technique is used to prove equivalences in Subsection 4.7.

Lemma 30. *Let $\mathscr{R}$ be a $\langle \Gamma, \Delta, m, n\rangle$-indexed relation on configurations. Suppose that*

$$\Gamma;\Delta \vdash \mathbf{M} \ \mathscr{R}\ \mathbf{N} \text{ and } \Gamma;\Delta \vdash \mathbf{M} \xrightarrow{\kappa} \Gamma';\Delta' \vdash \mathbf{M}'$$

implies there exists $\mathbf{N}'$ such that

$$\Gamma;\Delta \vdash \mathbf{N} \xrightarrow{\kappa} \Gamma';\Delta' \vdash \mathbf{N}' \text{ and } \Gamma';\Delta' \vdash \mathbf{M}' \ (\twoheadleftarrow\twoheadrightarrow;\mathscr{R}^{\bullet};\twoheadleftarrow\twoheadrightarrow)\ \mathbf{N}'.$$

Then $(\twoheadleftarrow\twoheadrightarrow;\mathscr{R}^{\bullet};\twoheadleftarrow\twoheadrightarrow) \subseteq \sim$.

PROOF SKETCH. The complex statement of the lemma masks its simple (but tedious) proof. As we have discussed already in Example 21, a simple and standard LTS proof shows that prefixing by τ preserves bisimilarity. So, $(\twoheadleftarrow\twoheadrightarrow;\sim;\twoheadleftarrow\twoheadrightarrow) \subseteq \sim$. Thus, the essence of the above lemma is that $\mathscr{R}^{\bullet} \subseteq \sim$. The proof formalizes the following intuitive observations:

- Transitions from duplicated values are already available in the starting configuration. So, duplicating values in the value list does not alter bisimilarity reasoning.
- State-free terms are "functional" in the sense that transitions from state-free terms are not dependent on the configuration. So, addition of identical state-free terms to the value list does not alter bisimilarity reasoning. □

One very useful consequence of the lemma is that $\sim^{\bullet} \subseteq \sim$. The results of Section 5 imply that $\sim$ is sound for a more general version of Definition 29: i.e., if $fn(U)$ and $fn(\mathscr{E})$ are bound by Γ and Δ and if $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U} \ \sim\ \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$ then,

$$\Gamma;\Delta \vdash \vec{A}/\mathscr{E},\vec{\mathscr{E}}/M/U,\vec{U} \ \sim\ \vec{B}/\mathscr{E},\vec{\mathscr{F}}/N/U,\vec{V}.$$

However, this more general property of $\sim$ is not necessarily sound as part of an upto-proof technique.

4.7 Examples with local store and higher-order functions

Examples 31 and 32 illustrate equivalences involving local state and higher-order functions — originally due to Meyer and Sieber [46]. The proofs provided here exemplify the techniques needed to address examples 1–5 and example 7 from [46]. Their example 6 involves the equality of locations, and the encoding used here does not accommodate such tests. To better illustrate the LTS, examples are written in our language directly rather than using the syntactic sugar for references in Example 7. We use the standard Church numerals for encoding natural numbers and operators in the lambda calculus. In the bisimulation candidate relation, we explicitly describe these arithmetic encodings to avoid addressing issues of normal forms of Church numerals. In a larger application, the systematic way to proceed would be to move to a typed setting and enrich the transition system (and hence the bisimulation relation) with new constants.

Example 31 (Local store). This example shows that local declaration of a primitive pointcut and function at that primitive pointcut(providing local store) does not affect computation. Consider the following terms.

$$\mathsf{M} = \mathsf{x} \qquad \mathsf{N} = \textbf{pcd}\ \mathsf{p}; \textbf{fun}\ \mathsf{f@p} = \lambda.0; \mathsf{x}$$

We wish to prove that $\lambda x.M \sim \lambda x.N$. By congruence (Theorem 36) it suffices to show $M \sim N$. Define the relation $\mathscr{R}$ as

$$x;\cdot \vdash (\cdot/\cdot/x/\cdot) \;\; \mathscr{R} \;\; (\vec{A}/\cdot/x/\cdot)$$

where $\vec{A} = (\textbf{pcd}\ p;\ \textbf{fun}\ f@p = \lambda.0)$.

The only possible transition labels are $\mathsf{app}\ \phi$ and put.

$$\begin{array}{ccc} x;\cdot \vdash \cdot/\cdot/x/\cdot & \xrightarrow{\mathsf{app}\ \phi} & x, \phi; \cdot \vdash \cdot/\cdot/x\ \phi/\cdot \\ \mathscr{R}\big\{ & & \big\}\mathscr{R}^{\bullet}_{\mathtt{sf}} \\ x;\cdot \vdash \vec{A}/\cdot/x/\cdot & \xrightarrow{\mathsf{app}\ \phi} & x, \phi; \cdot \vdash \vec{A}/\cdot/x\ \phi/\cdot \end{array} \qquad \begin{array}{ccc} x;\cdot \vdash \cdot/\cdot/x/\cdot & \xrightarrow{\mathsf{put}} & x;\cdot \vdash \cdot/\cdot/x/x \\ \mathscr{R}\big\{ & & \big\}\mathscr{R}^{\bullet}_{\mathtt{sf}} \\ x;\cdot \vdash \vec{A}/\cdot/x/\cdot & \xrightarrow{\mathsf{put}} & x;\cdot \vdash \vec{A}/\cdot/x/x \end{array}$$

By Lemma 30, $x;\cdot \vdash (\cdot/\cdot/x/\cdot) \sim (\cdot/\cdot/\vec{A}\,;x/\cdot)$. □

Example 32 (Higher-order functions). This example demonstrates reasoning about a call to an unknown procedure. Define M and N as follows.

$$\begin{aligned} M &= x\,(\lambda.())\,;\,() \\ N &= \textbf{pcd}\ p;\ \textbf{fun}\ f@p = \lambda.0; \\ &\quad\ x\,(\lambda.(\textbf{let}\ y = f\,()\,;\,(\textbf{adv}\ p = \lambda.\lambda.y+2)\,;\,()))\,; \\ &\quad\ \textbf{if}\ ((f\,()\ \textbf{mod}\ 2) = 0)\ \textbf{then}\ ()\ \textbf{else}\ \Omega \end{aligned}$$

In M, the external procedure x is invoked with a functional argument without side effects. In N, x is invoked with an argument that advises the local function f — corresponding to incrementing a local reference by two — thus maintaining the invariant that a call to f yields an even number.

In our proof, we prove the local invariant of evenness separately, without referring to the external function call. The bisimulation principle allows us to modularly add the external function.

To prove $\lambda x.M \sim \lambda x.N$ it suffices to show that $M \sim N$ (as before, by congruence). We use the following definitions.

$$\begin{aligned} U &= \lambda.() \\ \mathscr{E} &= [\text{–}]\,;\,() \\ \vec{A} &= \textbf{pcd}\ p;\ \textbf{fun}\ f@p = \lambda.0 \\ V &= \lambda.(\textbf{let}\ y = f\,()\,;\,(\textbf{adv}\ p = \lambda.\lambda.y+2)\,;\,()) \\ \mathscr{F} &= [\text{–}]\,;\,\textbf{if}\ ((f\,()\ \textbf{mod}\ 2) = 0)\ \textbf{then}\ ()\ \textbf{else}\ \Omega \\ \vec{B}_0 &\text{ is the empty advice list} \\ \vec{B}_n &= \vec{B}_{n-1}\,;\,(\textbf{adv}\ p = \lambda.\lambda.\mathsf{Dbl}_n) \\ \mathsf{Dbl}_0 &= \lambda f.\lambda x.x \\ \mathsf{Dbl}_{n+1} &= \lambda f.\lambda x.f(f(\mathsf{Dbl}_n\ f\ x)) \end{aligned}$$

Here the term Dbl_n represents $2n$, although it is not syntactically identical to the Church numeral for $2n$, and we are making use of the convention mentioned earlier for application of a term (rather than application of a value). So, $M = \mathscr{E}[x\ U]$ and $N = \vec{A}\,;\mathscr{F}[x\ V]$. We first prove two purely local results without the external call, to show that the tests under consideration (as given by $\mathscr{E}, \mathscr{F}$) do not distinguish $(\vec{A}, \vec{B}_m)$ and $(\vec{A}, \vec{B}_n)$ for any m, n.

(1) $\cdot;\cdot \vdash \vec{\mathsf{A}}, \vec{\mathsf{B}}_m/\mathscr{E}/\mathsf{V}/\mathsf{V}$ and $\cdot;\cdot \vdash \vec{\mathsf{A}}, \vec{\mathsf{B}}_n/\mathscr{F}/\mathsf{V}/\mathsf{V}$ are bisimilar, for all m,n.
(2) $\cdot;\cdot \vdash \vec{\mathsf{A}}, \vec{\mathsf{B}}_m/\mathscr{E}/\mathsf{V}/\mathsf{V}$ and $\cdot;\cdot \vdash \vec{\mathsf{A}}, \vec{\mathsf{B}}_n/\mathscr{E}/\mathsf{U}/\mathsf{U}$ are bisimilar, for all m,n.

We address (1); the proof for (2) is identical and omitted.

Let m,n range over all non-negative integers. Define $\mathscr{R}$ as follows.

$$\cdot;\cdot \vdash (\vec{\mathsf{A}}, \vec{\mathsf{B}}_m/\mathscr{F}/\mathsf{V}/\mathsf{V}) \;\mathscr{R}\; (\vec{\mathsf{A}}, \vec{\mathsf{B}}_n/\mathscr{E}/\mathsf{V}/\mathsf{V})$$

There are three possibilities for the transition system labels. Let $\stackrel{\kappa}{\Longrightarrow}$ be the sequential composition of $\stackrel{\kappa}{\longrightarrow}$ and $\twoheadleftarrow\!\!\twoheadrightarrow$.

Case $\mathsf{put}, \mathsf{get}\,1$**:** For put, we have the following.

$$\begin{array}{ccc} \cdot;\cdot \vdash \vec{\mathsf{A}}, \vec{\mathsf{B}}_m/\mathscr{E}/\mathsf{V}/\mathsf{V} & \xrightarrow{\mathsf{put}} & \cdot;\cdot \vdash \vec{\mathsf{A}}, \vec{\mathsf{B}}_m/\mathscr{E}/\mathsf{V}/\mathsf{V},\mathsf{V} \\ \mathscr{R}\big\{ & & \big\}\mathscr{R}^\bullet_{\mathsf{dup}} \\ \cdot;\cdot \vdash \vec{\mathsf{A}}, \vec{\mathsf{B}}_n/\mathscr{F}/\mathsf{V}/\mathsf{V} & \xrightarrow{\mathsf{put}} & \cdot;\cdot \vdash \vec{\mathsf{A}}, \vec{\mathsf{B}}_n/\mathscr{F}/\mathsf{V}/\mathsf{V},\mathsf{V} \end{array}$$

Similarly for $\mathsf{get}\,1$.

Case $\mathsf{app}\,\phi$**:** Using the operational semantics,

$$\begin{array}{l} \cdot;\cdot \vdash \vec{\mathsf{A}}, \vec{\mathsf{B}}_m/\mathscr{E}/\mathsf{V}/\mathsf{V} \stackrel{\mathsf{app}\,\phi}{\twoheadleftarrow\!\!\twoheadrightarrow} \cdot;\cdot \vdash \vec{\mathsf{A}}, \vec{\mathsf{B}}_{m+1}/\mathscr{E}/()/\mathsf{V} \\ \cdot;\cdot \vdash \vec{\mathsf{A}}, \vec{\mathsf{B}}_n/\mathscr{F}/\mathsf{V}/\mathsf{V} \stackrel{\mathsf{app}\,\phi}{\twoheadleftarrow\!\!\twoheadrightarrow} \cdot;\cdot \vdash \vec{\mathsf{A}}, \vec{\mathsf{B}}_{n+1}/\mathscr{F}/()/\mathsf{V} \end{array}$$

and thus we have the following.

$$\begin{array}{ccc} \cdot;\cdot \vdash \vec{\mathsf{A}}, \vec{\mathsf{B}}_m/\mathscr{E}/\mathsf{V}/\mathsf{V} & \stackrel{\mathsf{app}\,\phi}{\Longrightarrow} & \phi;\cdot \vdash \vec{\mathsf{A}}, \vec{\mathsf{B}}_{m+1}/\mathscr{E}/()/\mathsf{V} \\ \mathscr{R}\big\{ & & \big\}\mathscr{R}^\bullet \\ \cdot;\cdot \vdash \vec{\mathsf{A}}, \vec{\mathsf{B}}_n/\mathscr{F}/\mathsf{V}/\mathsf{V} & \stackrel{\mathsf{app}\,\phi}{\Longrightarrow} & \phi;\cdot \vdash \vec{\mathsf{A}}, \vec{\mathsf{B}}_{n+1}/\mathscr{F}/()/\mathsf{V} \end{array}$$

Case $\mathsf{ret}\,\phi$**:** Use the invariant that for any m, the function call $\mathsf{f}\;()$ in advice context $\vec{\mathsf{A}}, \vec{\mathsf{B}}_m$ evaluates to an even number.

$$\begin{array}{ccc} \cdot;\cdot \vdash \vec{\mathsf{A}}, \vec{\mathsf{B}}_m/\mathscr{E}/\mathsf{V}/\mathsf{V} & \stackrel{\mathsf{ret}\,\phi}{\Longrightarrow} & \phi;\cdot \vdash \vec{\mathsf{A}}, \vec{\mathsf{B}}_m/\cdot/()/\mathsf{V} \\ \mathscr{R}\big\{ & & \big\}\mathscr{R}^\bullet \\ \cdot;\cdot \vdash \vec{\mathsf{A}}, \vec{\mathsf{B}}_n/\mathscr{F}/\mathsf{V}/\mathsf{V} & \stackrel{\mathsf{ret}\,\phi}{\Longrightarrow} & \phi;\cdot \vdash \vec{\mathsf{A}}, \vec{\mathsf{B}}_n/\cdot/()/\mathsf{V} \end{array}$$

This concludes the case analysis. Therefore, by Lemma 30, $\mathscr{R}^\bullet$, and hence $\mathscr{R}$ also, is contained in bisimilarity.

Now, using transitivity of bisimilarity on (1) and (2) yields

$$\cdot;\cdot \vdash (\vec{\mathsf{A}}/\mathscr{E}/\mathsf{U}/\cdot) \sim (\vec{\mathsf{A}}/\mathscr{F}/\mathsf{V}/\cdot).$$

Since x is not free in either configuration, we have

$$\mathsf{x};\cdot \vdash (\vec{\mathsf{A}}/\mathscr{E}/\mathsf{U}/\cdot) \sim (\vec{\mathsf{A}}/\mathscr{F}/\mathsf{V}/\cdot).$$

From Example 31, since we have that $\sim^{\bullet}_{\mathtt{sf}} \subseteq \sim$ and that $\mathscr{E}$ and U are state-free for x:

$$\mathsf{x};\cdot \vdash (\vec{\mathsf{A}}/\mathscr{E}/\mathsf{U}/\cdot) \sim (\cdot/\mathscr{E}/\mathsf{U}/\cdot).$$

Using transitivity of $\sim$

$$\mathsf{x};\cdot \vdash (\vec{\mathsf{A}}/\mathscr{F}/\mathsf{V}/\cdot) \sim (\cdot/\mathscr{E}/\mathsf{U}/\cdot).$$

Since

$$(\mathsf{x};\cdot \vdash \cdot/\cdot/\mathscr{E}[\mathsf{x}\,\mathsf{U}]/\cdot) \xrightarrow{\mathsf{fcall}\,\mathsf{x}}\!\!\!\!\rightarrow (\mathsf{x};\cdot \vdash \cdot/\mathscr{E}/\mathsf{U}/\cdot), \text{ and}$$
$$(\mathsf{x};\cdot \vdash \cdot/\cdot/\vec{\mathsf{A}}\,;\mathscr{F}[\mathsf{x}\,\mathsf{V}]/\cdot) \xrightarrow{\mathsf{fcall}\,\mathsf{x}}\!\!\!\!\rightarrow (\mathsf{x};\cdot \vdash \vec{\mathsf{A}}/\mathscr{F}/\mathsf{V}/\cdot),$$

the required result,

$$\mathsf{x};\cdot \vdash (\cdot/\cdot/\vec{\mathsf{A}}\,;\mathscr{F}[\mathsf{x}\,\mathsf{V}]/\cdot) \sim (\cdot/\cdot/\mathscr{E}[\mathsf{U}]/\cdot),$$

follows since both sides have only weak $\mathsf{fcall}\,\mathsf{x}$ transitions to bisimilar targets. □

5 Results

Bisimilarity is sound and complete relative to observational congruence. Completeness is discussed in Appendix E. Here we sketch a proof of soundness, focusing on the technical novelties of our analysis; details are deferred to the appendices.

In the rest of this Section, we show that $\sim$ is a congruence. From this it is straightforward to show that $M \sim N$ implies $M \equiv N$.

The key component of the proof is a substitution lemma that validates substitution of equals-for-equals for contexts that do not capture variables: the reader might want to view this semantically as an instance of the composition principles underlying game semantics [3, 26], and syntactically as our variant of the delayed substitutions of the SECD machine [37]. This is stated and proved in Section A.

5.1 Bisimulation is a congruence

The following notion of *compatibility* captures some useful properties of the initial configurations of Definition 20 and those reachable from them.

Definition 33. A pair of LTS configurations $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U}$ and $\Gamma;\Delta \vdash \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$ are *compatible* if the following hold.

- All advice in Δ is symbolic advice of the form $\mathsf{adv}\ p = \alpha$.
- If $\mathsf{pcd}\ p \in \Delta$, then there exists $\mathsf{adv}\ p = \alpha \in \Delta$.
- For each $\mathsf{adv}\ p = \alpha \in \Delta$, it is the sole occurrence of α in Δ.
- If $\mathsf{fun}\ f\mathtt{@}p = \phi \in \Delta$ then there exists $1 \le i \le \min(|\vec{U}|, |\vec{V}|)$ such that $\vec{U}_i = \vec{V}_i = f$ □

The next two lemmas provide the infrastructure required to reason separately about the active term and the remaining pieces of a configuration. Lemma 34 permits the substitution of identical terms for values in the active term spot of bisimilar configurations, while maintaining bisimilarity. Lemma 35 is dual.

Lemma 34 (Inclusion of identical terms). *Consider compatible configurations* $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/U/\vec{U}$ *and* $\Gamma;\Delta \vdash \vec{B}/\vec{\mathscr{F}}/V/\vec{V}$*, and a term* L *such that* $fn(L) \subseteq \Gamma \cup dn(\Delta)$*. Then*

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/U/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/V/\vec{V}$$

implies

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/L/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/L/\vec{V}.$$

PROOF. See Appendix B. □

Lemma 35 (Inclusion of identical contexts). *Consider compatible configurations* $\Gamma;\Delta \vdash \cdot/\cdot/M/\vec{U}$ *and* $\Gamma;\Delta \vdash \cdot/\cdot/N/\vec{V}$*, and a well-formed LTS configuration* $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/()/\vec{W}$*, where no symbolic advice occurs in* M*,* $\vec{U}$*,* N*,* $\vec{V}$*. Then*

$$\Gamma;\Delta \vdash \cdot/\cdot/M/\vec{U} \sim \cdot/\cdot/N/\vec{V}$$

implies

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U},\vec{W} \sim \vec{A}/\vec{\mathscr{E}}/N/\vec{V},\vec{W}.$$

PROOF. See Appendix C. □

Given this machinery, the proof that bisimulation is a congruence (and is therefore sound for contextual equivalence) is quite routine.

Theorem 36 (Congruence of bisimilarity). *Consider values* $U_1 \sim U_1'$*,* $U_2 \sim U_2'$*, and* $U \sim U'$*, and terms* $M \sim M'$*,* $M_1 \sim M_1'$ *and* $M_2 \sim M_2'$*, that contain no symbolic advice. Then we have the following.*

$$U_1\ U_2 \sim U_1'\ U_2' \qquad \lambda x.M \sim \lambda x.M'$$

$$\mathsf{let}\ x = M_1\ ;\ M_2 \sim \mathsf{let}\ x = M_1'\ ;\ M_2' \qquad \mathsf{fun}\ f@p = U\ ;\ M \sim \mathsf{fun}\ f@p = U'\ ;\ M'$$

$$\mathsf{pcd}\ p\ ;\ M \sim \mathsf{pcd}\ p\ ;\ M' \qquad \mathsf{adv}\ p = \lambda z.U\ ;\ M \sim \mathsf{adv}\ p = \lambda z.U'\ ;\ M'$$

PROOF. See Appendix D. □

6 Conclusion

This paper is a step towards leveling the formal playing field between aspects and other programming paradigms. To our knowledge, we have presented the first description of bisimularity for aspect languages. We contribute new operational techniques to show that bisimilarity is a congruence. Our results complement ongoing research in the aspect community on the design and implementation of aspect languages.

On the one hand, our methods and techniques are those that are needed to address stateful higher order programming languages. This is already seen in the basic format of our transition systems. Just as the interface of a (perhaps higher order) stateful program includes the global variables that are being used in the program, our LTS embodies the distinction between external (visible and advisable) pointcuts and the internal

(hidden and unadvisable) pointcuts. Building further on analogies with higher order stateful computation, our bisimulation principle combines techniques used to address mobile processes (open bisimulation), names in the nu-calculus (via tracking leaked secrets in the LTS) and the lambda calculus (ENF-bisimulation). Our results suggest that from a purely theoretical viewpoint of studying general properties of the programming language, aspects are no more difficult to address formally than well-studied classical issues of higher-order imperative programs. We demonstrate the utility of our results by bridging the formal gap that exists between the foundations and the realizations of Open Modules. In particular, bisimulation is a congruence for a rich collection of temporal pointcuts including those that are realized in current implementations of Open Modules in aspect languages.

On the other hand, this message is tempered by its applicability to individual programs. Even theoretically speaking, our results do not directly yield a compositional translation of aspect programs into a higher order imperative language that preserves and reflects program equality. Our results are only a first step in terms of the practically important task of reasoning about concrete programs. We hope that they will serve as the conceptual infrastructure required to develop and validate "reasoning patterns" to address the common idioms of aspect-oriented programs.

Acknowledgements. We gratefully acknowledge suggestions by anonymous referees. James Riely was supported by NSF Career 0347542. Radha Jagadeesan and Corin Pitcher were supported by NSF Cybertrust 0430175.

Bibliography

[1] M. Abadi and L. Cardelli. *A Theory of Objects*. Springer Verlag, 1996.

[2] M. Abadi and C. Fournet. Access control based on execution history. In *Proceedings of the Network and Distributed System Security Symposium Conference*, 2003.

[3] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Inf. Comput.*, 163(2):409–470, 2000.

[4] S. Abramsky and C.-H. L. Ong. Full abstraction in the lazy lambda calculus. *Inf. Comput.*, 105(2):159–267, 1993.

[5] M. Aksit, K. Wakita, J. Bosch, L. Bergmans, and A. Yonezawa. Abstracting object-interactions using composition-filters. In *Object-based distributed processing, LNCS*, 1993.

[6] J. Aldrich. Open modules: Modular reasoning about advice. In A. P. Black, editor, *ECOOP*, volume 3586 of *Lecture Notes in Computer Science*, pages 144–168. Springer, 2005.

[7] R. Alur. The benefits of exposing calls and returns. In M. Abadi and L. de Alfaro, editors, *CONCUR*, volume 3653 of *Lecture Notes in Computer Science*, pages 2–3. Springer, 2005.

[8] R. Alur and P. Madhusudan. Adding nesting structure to words. In O. H. Ibarra and Z. Dang, editors, *Developments in Language Theory*, volume 4036 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2006.

[9] P. Avgustinov, E. Bodden, E. Hajiyev, L. Hendren, O. Lhoták, O. de Moor, N. Ongkingco, D. Sereni, G. Sittampalam, and J. Tibble. Aspects for trace monitoring. In K. Havelund, M. Nunez, G. Rosu, and B. Wolff, editors, *Formal Approaches to Testing Systems and Runtime Verification (FATES/RV)*, Lecture Notes in Computer Science. Springer, 2006.

[10] L. Bergmans. *Composing Concurrent Objects - Applying Composition Filters for the Development and Reuse of Concurrent Object-Oriented Programs*. Ph.D. thesis, University of Twente, 1994.

[11] C. Bockisch, M. Haupt, M. Mezini, and K. Ostermann. Virtual machine support for dynamic join points. In *AOSD*, pages 83–92, 2004.

[12] W. E. Boebert and R. Y. Kain. A practical alternative to hierarchical integrity policies. In *Proceedings of the Eighth National Computer Security Conference*, 1985.

[13] C. Clifton and G. T. Leavens. Minimao: An imperative core language for studying aspect-oriented reasoning. *Sci. Comput. Program.*, 63(3):321–374, 2006.

[14] C. Clifton, G. T. Leavens, and M. Wand. Parameterized aspect calculus: A core calculus for the direct study of aspect-oriented languages. At http://www.cs.iastate.edu/~cclifton/papers/TR03-13.pdf, 2003.

[15] Y. Coady, G. Kiczales, M. J. Feeley, and G. Smolyn. Using AspectC to improve the modularity of path-specific customization in operating system code. In *ESEC / SIGSOFT FSE*, pages 88–98, 2001.

[16] D. S. Dantas and D. Walker. Harmless advice. In Morrisett and Jones [50], pages 383–396.

[17] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34(1–2):83–133, Nov. 1984.

[18] C. Dutchyn, D. B. Tucker, and S. Krishnamurthi. Semantics and scoping of aspects in higher-order languages. *Sci. Comput. Program.*, 63(3):207–239, 2006.

[19] M. Felleisen, D. P. Friedman, E. Kohlbecker, and B. Duba. A syntactic theory of sequential control. *Theor. Comput. Sci.*, 52(3):205–237, 1987.

[20] R. Filman and D. Friedman. Aspect-oriented programming is quantification and obliviousness. In *Workshop on Advanced Separation of Concerns*, 2000.

[21] A. D. Gordon. A tutorial on co-induction and functional programming. In *In Glasgow functional programming workshop*, pages 78–95. Springer, 1994.

[22] A. D. Gordon. Bisimilarity as a theory of functional programming. *Electr. Notes Theor. Comput. Sci.*, 1, 1995.

[23] A. D. Gordon. Operational equivalences for untyped and polymorphic object calculi. In A. D. Gordon and A. M. Pitts, editors, *Higher-Order Operational Techniques in Semantics*, Publications of the Newton Institute, pages 9–54. Cambridge University Press, 1998.

[24] A. D. Gordon and G. D. Rees. Bisimilarity for a first-order calculus of objects with subtyping. In *POPL*, pages 386–395, 1996.

[25] D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Inf. Comput.*, 124(2):103–112, 1996.

[26] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II, and III. *Inf. Comput.*, 163(2):285–408, 2000.

[27] R. Jagadeesan, A. Jeffrey, and J. Riely. An untyped calculus of aspect oriented programs. In *Conference Record of ECOOP 03: The European Conference on Object-Oriented Programming*, volume 2743 of *Lecture Notes in Computer Science*, 2003.
[28] R. Jagadeesan, A. Jeffrey, and J. Riely. Typed parametric polymorphism for aspects. *Sci. Comput. Program.*, 63(3):267–296, 2006.
[29] A. Jeffrey and J. Rathke. A theory of bisimulation for a fragment of concurrent ML with local names. *Theor. Comput. Sci.*, 323(1-3):1–48, 2004. Preliminary version appeared in IEEE LICS 1999.
[30] A. Jeffrey and J. Rathke. Java Jr: Fully abstract trace semantics for a core Java language. In *ESOP*, volume 3444 of *LNCS*, pages 423–438. Springer, 2005.
[31] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An overview of AspectJ. *Lecture Notes in Computer Science*, 2072:327–355, 2001.
[32] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *European Conference on Object-Oriented Programming (ECOOP)*, 1997.
[33] G. Kiczales and M. Mezini. Aspect-oriented programming and modular reasoning. In *ICSE '05: Proceedings of the 27th international conference on software engineering*, pages 49–58, New York, NY, USA, 2005. ACM Press.
[34] V. Koutavas and M. Wand. Bisimulations for untyped imperative objects. In P. Sestoft, editor, *Proc. ESOP 2006*, volume 3924 of *Lecture Notes in Computer Science*, pages 146–161. Springer, Mar. 2006.
[35] V. Koutavas and M. Wand. Proving class equivalence. submitted for publication, July 2006.
[36] V. Koutavas and M. Wand. Small bisimulations for reasoning about higher-order imperative programs. In Morrisett and Jones [50], pages 141–152.
[37] P. J. Landin. The mechanical evaluation of expressions. *Computer Journal*, 6(4):308–320, Jan. 1964.
[38] S. Lassen. Eager normal form bisimulation. In *LICS*, pages 345–354. IEEE Computer Society, 2005.
[39] S. B. Lassen. Head normal form bisimulation for pairs and the $\lambda\mu$-calculus. In *LICS*, pages 297–306. IEEE Computer Society, 2006.
[40] S. B. Lassen and P. B. Levy. Typed normal form bisimulation. In J. Duparc and T. A. Henzinger, editors, *CSL*, volume 4646 of *Lecture Notes in Computer Science*, pages 283–297. Springer, 2007.
[41] H. C. Li, S. Krishnamurthi, and K. Fisler. Modular verification of open features using three-valued model checking. *Autom. Softw. Eng.*, 12(3):349–382, 2005.
[42] K. J. Lieberherr. *Adaptive Object-Oriented Software: The Demeter method with propagation patterns*. PWS Publishing Company, 1996.
[43] J. Ligatti, D. Walker, and S. Zdancewic. A type-theoretic interpretation of pointcuts and advice. *Sci. Comput. Program.*, 63(3):240–266, 2006.
[44] P. A. Loscocco and S. D. Smalley. Meeting critical security objectives with Security-Enhanced Linux. In *Proceedings of the 2001 Ottawa Linux Symposium*, 2001.
[45] H. Masuhara, G. Kiczales, and C. Dutchyn. A compilation and optimization model for aspect-oriented programs. In G. Hedin, editor, *CC*, volume 2622 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 2003.

[46] A. R. Meyer and K. Sieber. Towards fully abstract semantics for local variables. In *POPL*, pages 191–203, 1988.

[47] R. Milner. *Communication and concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[48] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.

[49] J. H. Morris. *Lambda-Calculus Models of Programming Languages*. PhD thesis, MIT, Dec. 1968.

[50] J. G. Morrisett and S. L. P. Jones, editors. *Proceedings of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2006, Charleston, South Carolina, USA, January 11-13, 2006*. ACM, 2006.

[51] S. Nakajima and T. Tamai. Lightweight formal analysis of aspect-oriented models. In *UML2004 Workshop on Aspect-Oriented Modeling*, 2004.

[52] N. Ongkingco, P. Avgustinov, J. Tibble, L. Hendren, O. de Moor, and G. Sittampalam. Adding open modules to AspectJ. In *AOSD '06: Proceedings of the 5th international conference on Aspect-oriented software development*, pages 39–50, New York, NY, USA, 2006. ACM Press.

[53] H. Ossher and P. Tarr. Multi-dimensional separation of concerns and the hyperspace approach. In *Proceedings of the Symposium on Software Architectures and Component Technology: The State of the Art in Software Development*, 2001.

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

[55] 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.

[56] H. Rajan and K. J. Sullivan. Classpects: unifying aspect- and object-oriented language design. In G.-C. Roman, W. G. Griswold, and B. Nuseibeh, editors, *ICSE*, pages 59–68. ACM, 2005.

[57] D. Sangiorgi. *Expressing Mobility in Process Algebras: First Order and Higher Order Paradigms*. PhD thesis, University of Edinburgh, 1993.

[58] D. Sangiorgi. A theory of bisimulation for the pi-calculus. *Acta Inf.*, 33(1):69–97, 1996.

[59] D. Sangiorgi. Bisimulation: From the origins to today. In *LICS*, pages 298–302. IEEE Computer Society, 2004.

[60] D. Sangiorgi. The bisimulation proof method: Enhancements and open problems. In R. Gorrieri and H. Wehrheim, editors, *FMOODS*, volume 4037 of *Lecture Notes in Computer Science*, pages 18–19. Springer, 2006.

[61] D. Sangiorgi, N. Kobayashi, and E. Sumii. Environmental bisimulations for higher-order languages. In *LICS*, pages 293–302. IEEE Computer Society, 2007.

[62] M. Sihman and S. Katz. Model checking applications of aspects and superimpositions. In *Foundations of Aspect Languages*, 2003.

[63] K. Støvring and S. B. Lassen. A complete, co-inductive syntactic theory of sequential control and state. In M. Hofmann and M. Felleisen, editors, *POPL*, pages 161–172. ACM, 2007.

[64] E. Sumii and B. C. Pierce. A bisimulation for dynamic sealing. In *POPL*, 2004. Full version in Theoretical Computer Science 375 (2007), 169-192.

[65] P. L. Tarr and H. Ossher. Hyper/J: Multi-dimensional separation of concerns for Java. In *ICSE*, pages 729–730, 2001.

[66] N. Ubayashi and T. Tamai. Aspect-oriented programming with model checking. In *AOSD '02: Proceedings of the 1st international conference on Aspect-oriented software development*, pages 148–154, New York, NY, USA, 2002. ACM Press.

[67] D. Walker, S. Zdancewic, and J. Ligatti. A theory of aspects. In C. Runciman and O. Shivers, editors, *ICFP*, pages 127–139. ACM, 2003.

[68] K. M. Walker, D. F. Sterne, M. L. Badger, M. J. Petkac, D. L. Shermann, and K. A. Oostendorp. Confining root programs with Domain and Type Enforcement (DTE). In *Proceedings of the Sixth USENIX UNIX Security Symposium*, 1996.

[69] M. Wand, G. Kiczales, and C. Dutchyn. A semantics for advice and dynamic join points in aspect-oriented programming. *TOPLAS*, 26(5):890–910, September 2004.

A Substitution lemma

This appendix develops the proof of Theorem 49, which states that substitution preserves bisimilarity. The substitutions that we consider provide two kinds of substitution information on a LTS configuration:

- Which value from the list of values in a configuration is substituted for a variable? This information is indicated by the positional index in $\vec{U}$ in a LTS configuration.
- Which contexts in the list of evaluation contexts are composed with other evaluation contexts and the active term? This information is specified by an integer stack.

Definition 37. An *extended substitution* σ consists of a partial function from variables (symbolic function names or symbolic advice names) to integers, a total order upon the domain of the partial function, and a non-empty stack of natural numbers.

Let ξ range over symbolic function names (metavariable ϕ) and symbolic advice names (metavariable α). If ξ is in the domain of the partial function of σ, we use $\sigma(\xi)$ for the value of the partial function of σ. We use $(\xi \mapsto k) \uplus \sigma$ for the operation of extending the domain of the partial function of σ to include ξ and placing ξ at the bottom of the total order on the domain of the new partial function. This operation is undefined if ξ is already in the domain of the original partial function σ.

In the non-empty stack of natural numbers, written $(\vec{m}, m)$ with m at the top, we require that $m_i > 0$, for all $1 \leq i \leq |\vec{m}|$, but allow $m = 0$. We write $|\sigma|$ for the length of the stack and $sum(\sigma)$ for the sum of the values on the stack. □

Definition 38. Given contexts $\mathscr{E}$ and $\mathscr{F}$, define $\mathscr{E} \circ \mathscr{F}$ as $\mathscr{E}[\mathscr{F}]$. Observe that $\circ$ is associative, with $[-]$ as a left and right identity.

$$Z(\mathscr{E}_1, \ldots, \mathscr{E}_n) \triangleq \mathscr{E}_1 \circ \cdots \circ \mathscr{E}_n$$

$$Z_{m_1,\ldots,m_k}(\mathscr{E}_{\langle 1,1\rangle}, \ldots, \mathscr{E}_{\langle 1,m_1\rangle}, \ \ldots, \ \mathscr{E}_{\langle k,1\rangle}, \ldots, \mathscr{E}_{\langle k,m_k\rangle}) \triangleq Z(\mathscr{E}_{\langle 1,1\rangle}, \ldots, \mathscr{E}_{\langle 1,m_1\rangle}), \ \ldots, \ Z(\mathscr{E}_{\langle k,1\rangle}, \ldots, \mathscr{E}_{\langle k,m_k\rangle})$$ □

Note that $Z(\,\cdot\,) = [-]$ and that $Z_\sigma(\vec{\mathscr{E}})$ returns a sequence of contexts of length $|\sigma|$. Given that all but the rightmost element of σ must be nonzero, we may conclude that $|Z_\sigma(\vec{\mathscr{E}})| \leq |\vec{\mathscr{E}}| + 1$.

Definition 39. σ is *valid* for $\Gamma; \Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U}$ if the following hold.

- The domain of σ is a subset of Γ.
- If α is an advice variable in the domain of σ, $U_{\sigma(\alpha)}$ is of the form $\lambda z.U$.
- If the total order for the domain of σ is $\xi_1, \ldots, \xi_n$, then, for all $1 \leq k \leq j \leq n$, we have ξ_k is not free in $U_{\sigma(\xi_j)}$.
- $sum(\sigma)$ equals the length of $\vec{\mathscr{E}}$. □

Definition 40. If σ is valid for $\Gamma; \Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U}$, we define:

$$[\Gamma; \Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U}]_\sigma \triangleq \Gamma'; \Delta \vdash \vec{A}'/\vec{\mathscr{E}}''/\mathscr{E}''[M']/\vec{U}''$$

where the primed elements on the right-hand side are derived as follows.

- Γ' is obtained from Γ by deleting every variable in the domain of σ.
- For every other metavariable χ, the single-primed version χ' is derived by substituting $U_{\sigma(\phi)}$ for ϕ in the configuration. The substitution is carried out following the total order of the variables given by σ (the least variable substituted first).
- $\vec{\mathscr{E}}'', \mathscr{E}'' = Z_\sigma(\mathscr{E}')$.
- $\vec{U}''$ is obtained from $\vec{U}'$ by deleting values at positions in the substitution from σ.

In addition, we write $[\vec{A}/\vec{\mathscr{E}}/M/\vec{U}]_\sigma$ when the context is uninteresting. □

For example, if $\mathbf{M} = \vec{A}/\mathscr{E}_1, \mathscr{E}_2, \mathscr{E}_3, \mathscr{E}_4, \mathscr{E}_5/M/\vec{U}$ then $[\mathbf{M}]_{3,2,0} = \vec{A}/\mathscr{E}_1[\mathscr{E}_2[\mathscr{E}_3]], \mathscr{E}_4[\mathscr{E}_5]/M/\vec{U}$, and $[\mathbf{M}]_{2,1,2} = \vec{A}/\mathscr{E}_1[\mathscr{E}_2], \mathscr{E}_3/\mathscr{E}_4[\mathscr{E}_5[M]]/\vec{U}$.

Definition 41. Write $\Gamma;\Delta \vdash \mathbf{M} \approx_\sigma \mathbf{N}$ if there exists $\Gamma';\Delta' \vdash \mathbf{M}' \sim \mathbf{N}'$ such that σ is valid for both $\Gamma';\Delta' \vdash \mathbf{M}'$ and $\Gamma';\Delta' \vdash \mathbf{N}'$, and we have $\Gamma;\Delta \vdash \mathbf{M} = [\Gamma';\Delta' \vdash \mathbf{M}']_\sigma$ and $\Gamma;\Delta \vdash \mathbf{N} = [\Gamma';\Delta' \vdash \mathbf{N}']_\sigma$. □

Two configurations are related by $\approx_\sigma$ if they are in the σ-image of configurations that are related by $\sim$. Theorem 49 formalizes the claim that bisimilarity is closed under substitution by stating that the relation $\approx = \bigcup_\sigma \approx_\sigma$ is a bisimulation.

To prove the substitution result, we first note that reduction is preserved by an extended substitution.

Lemma 42. *Given an extended substitution σ valid for $\Gamma;\Delta \vdash \mathbf{M}$, if:*

$$\Gamma;\Delta \vdash \mathbf{M} \xrightarrow{\tau} \Gamma;\Delta \vdash \mathbf{M}'$$

then:

$$[\Gamma;\Delta \vdash \mathbf{M}]_\sigma \xrightarrow{\tau} [\Gamma;\Delta \vdash \mathbf{M}']_\sigma$$

PROOF. Reduction is preserved by substitution of values for variables in Γ, and by the addition of an evaluation context to a term (reduction is specified in terms of evaluation contexts).

Next we prove two lemmas that are used to choose bisimilar configurations in certain forms. Lemma 43 is used to eliminate trivial evaluation contexts (of the form $[-]$) from the evaluation context stack when they are applied directly to a value. Lemma 46 is used to remove substitutions that substitute one variable for another.

More specifically, Lemma 43 shows that for any $\approx$-related configurations, if the underlying pair of bisimilar configurations has a value on the left-hand side configuration, then there is another underlying pair of bisimilar configurations where the first non-trivial evaluation context from the evaluation context stack has been moved to the active term (to cause a reduction), or there is no non-trivial evaluation context. This manipulation is limited to the right-most m evaluation contexts when the stack from the extended substitution is $(\vec{m}, m)$. To indicate when a reduction is possible, we write $\vec{A}/M \rightarrow$ when there exist $\vec{B}, N$ such that $\vec{A}/M \rightarrow \vec{B}/N$.

Lemma 43. *Given an extended substitution σ valid for $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/U/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$, there exists an extended substitution σ' valid for $\Gamma';\Delta \vdash \vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}' \sim \vec{B}'/\vec{\mathscr{F}}'/N'/\vec{V}'$ such that:*

1. $[\vec{A}/\vec{\mathscr{E}}/U/\vec{U}]_\sigma = [\vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}']_{\sigma'}$
2. $[\Gamma;\Delta \vdash \vec{B}/\vec{\mathscr{F}}/N/\vec{V}]_\sigma \xrightarrow{\tau}{}^* [\Gamma';\Delta \vdash \vec{B}'/\vec{\mathscr{F}}'/N'/\vec{V}']_{\sigma'}$
3. *One of the following holds:*
   (a) ***Reduction:*** $\vec{A}'/M' \rightarrow$.
   (b) ***Value:*** *The* σ' *stack has the form* $(\vec{m},0)$ *and* M' *is a value.*

PROOF. The proof is by induction on m, where the stack from σ is $(\vec{m},m)$. If $m = 0$ the original extended substitution and configurations satisfy requirements (1)-(3), and we are done. If $m > 0$ then $\vec{\mathscr{E}}$ can be decomposed as $\vec{\mathscr{E}} = (\vec{\mathscr{E}}'', \mathscr{E}'')$, and the bisimilarity $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/U/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$ yields the existence of $\vec{C}$ and V such that $\vec{B}/N \twoheadrightarrow \vec{C}/V$ and $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/U/\vec{U} \sim \vec{C}/\vec{\mathscr{F}}/V/\vec{V}$. Now applying put then ret ϕ transitions to both sides (ϕ fresh), where $\vec{\mathscr{F}} = (\vec{\mathscr{F}}'', \mathscr{F}'')$, we have:

$$\Gamma,\phi;\Delta \vdash \vec{A}/\vec{\mathscr{E}}''/\mathscr{E}''[\phi]/\vec{U},U \sim \vec{C}/\vec{\mathscr{F}}''/\mathscr{F}''[\phi]/\vec{V},V$$

We obtain the extended substitution σ'' by replacing the stack of $(\phi \mapsto |\vec{U}|+1) \uplus \sigma$ with $(\vec{m},m-1)$. Then $[\vec{A}/\vec{\mathscr{E}}''/\mathscr{E}''[\phi]/\vec{U},U]_{\sigma''} = [\vec{A}/\vec{\mathscr{E}}/U/\vec{U}]_\sigma$ and $[\vec{C}/\vec{\mathscr{F}}''/\mathscr{F}''[\phi]/\vec{V},V]_{\sigma''} = [\vec{C}/\vec{\mathscr{F}}/V/\vec{V}]_\sigma$. There are two cases depending on whether or not $\mathscr{E}'' = [-]$.

Case $\mathscr{E}'' \neq [-]$. When $\mathscr{E}'' \neq [-]$ there exists $\mathscr{E}'''$ such that $\mathscr{E}'' = \mathscr{E}'''[\mathsf{let}\ x = [-]\ ;\ M'']$, so we have the reduction:

$$\vec{A}/\mathscr{E}''[\phi] = \vec{A}/\mathscr{E}'''[\mathsf{let}\ x = \phi\ ;\ M''] \twoheadrightarrow \vec{A}/\mathscr{E}'''[M''[x := \phi]]$$

Thus we define $\sigma' = \sigma''$, $\Gamma' = (\Gamma,\phi)$, and:

$$\begin{array}{l}(\vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}') = (\vec{A}/\vec{\mathscr{E}}''/\mathscr{E}''[\phi]/\vec{U},U)\\ (\vec{B}'/\vec{\mathscr{F}}'/N'/\vec{V}') = (\vec{C}/\vec{\mathscr{F}}''/\mathscr{F}''[\phi]/\vec{V},V)\end{array}$$

Requirements (1) and (3)(a) are satisfied immediately. Using Lemma 42 and the earlier reduction $\vec{B}/N \twoheadrightarrow \vec{C}/V$, we deduce that (2) is also satisfied:

$$\begin{array}{ll} & [\Gamma;\Delta \vdash \vec{B}/\vec{\mathscr{F}}/N/\vec{V}]_\sigma\\ \xrightarrow{\tau}{}^* & [\Gamma;\Delta \vdash \vec{C}/\vec{\mathscr{F}}/V/\vec{V}]_\sigma\\ = & [\Gamma,\phi;\Delta \vdash \vec{C}/\vec{\mathscr{F}}''/\mathscr{F}''[\phi]/\vec{V},V]_{\sigma''}\\ = & [\Gamma';\Delta \vdash \vec{B}'/\vec{\mathscr{F}}'/N'/\vec{V}']_{\sigma'}\end{array}$$

Case $\mathscr{E}'' = [-]$. Now $\mathscr{E}''[\phi] = \phi$ is a value and the top element of the stack of σ'' is strictly smaller than that of σ, so the induction hypothesis can be applied to σ'' and $\Gamma,\phi;\Delta \vdash \vec{A}/\vec{\mathscr{E}}''/\phi/\vec{U},U \sim \vec{C}/\vec{\mathscr{F}}''/\mathscr{F}''[\phi]/\vec{V},V$ to yield σ' valid for $\Gamma';\Delta \vdash \vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}' \sim \vec{B}'/\vec{\mathscr{F}}'/N'/\vec{V}'$ such that:

- $[\vec{A}/\vec{\mathscr{E}}''/\phi/\vec{U},U]_{\sigma''} = [\vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}']_{\sigma'}$
- $[\Gamma,\phi;\Delta \vdash \vec{C}/\vec{\mathscr{F}}''/\mathscr{F}''[\phi]/\vec{V},V]_{\sigma''} \xrightarrow{\tau}{}^* [\Gamma';\Delta \vdash \vec{B}'/\vec{\mathscr{F}}'/N'/\vec{V}']_{\sigma'}$
- • Either $\vec{A}'/M' \rightarrow$.
  • Or the σ' stack has the form $(\vec{m},0)$ and M' is a value.

For (1), by transitivity of equality we have $[\vec{A}/\vec{\mathscr{E}}/U/\vec{U}]_\sigma = [\vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}']_{\sigma'}$. For (2), using Lemma 42 and the earlier reduction $\vec{B}/N \twoheadrightarrow \vec{C}/V$, we deduce that:

$$\begin{array}{rl} & [\Gamma;\Delta \vdash \vec{B}/\vec{\mathscr{F}}/N/\vec{V}]_\sigma \\ \xrightarrow{\tau}{}^* & [\Gamma;\Delta \vdash \vec{C}/\vec{\mathscr{F}}/V/\vec{V}]_\sigma \\ = & [\Gamma,\phi;\Delta \vdash \vec{C}/\vec{\mathscr{F}}''/\mathscr{F}''[\phi]/\vec{V},V]_{\sigma''} \\ \xrightarrow{\tau}{}^* & [\Gamma';\Delta \vdash \vec{B}'/\vec{\mathscr{F}}'/N'/\vec{V}']_{\sigma'} \end{array}$$

Finally, (3) is satisfied immediately by the results of the induction hypothesis above. This completes the proof of Lemma 43.

In order to prove that substitutions of variables for variables can be eliminated, we introduce the notion of a variable chain length in a substitution and value list. This measures sequences of substitutions of a variable with another variable in the value list. Such sequences transfer control between values without making reductions. We formalize the notion of variable chain length as follows.

Definition 44. Given σ valid for $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U}$, and $\xi \in \Gamma$, define the natural number $\mathsf{varchain}(\xi,\sigma,\vec{U})$ by:

$$\mathsf{varchain}(\xi,\sigma,\vec{U}) \triangleq \begin{cases} 1+\mathsf{varchain}(\psi,\sigma,\vec{U}) & \text{if } (\xi=\phi \wedge U_{\sigma(\phi)}=\psi) \\ & \quad \vee (\xi=\alpha \wedge U_{\sigma(\alpha)}=\lambda x.\psi) \\ 1+\mathsf{varchain}(\beta,\sigma,\vec{U}) & \text{if } (\xi=\phi \wedge U_{\sigma(\phi)}=\beta\texttt{<}U\texttt{>}) \\ & \quad \vee (\xi=\alpha \wedge U_{\sigma(\alpha)}=\lambda x.\beta\texttt{<}U\texttt{>}) \\ 0 & \text{otherwise} \end{cases}$$

To simplify arguments about the two different kinds of call transitions, we introduce new terminology.

Definition 45. We say that $\Gamma;\Delta \vdash \mathbf{M}$ has a call transition on ξ if either $\Gamma;\Delta \vdash \mathbf{M} \xrightarrow{\mathsf{fcall}\ \xi}$ or $\Gamma;\Delta \vdash \mathbf{M} \xrightarrow{\mathsf{acall}\ \xi}$.

Lemma 46 shows that for any $\approx$-related configurations, if the underlying pair of bisimilar configurations has a call transition on the left-hand side configuration, then there is another underlying pair of bisimilar configurations where the call transition has turned into a reduction (due to substitutions from the value list for the variable in the call), or the call transition is unaffected by the substitution.

Lemma 46. *Consider an extended substitution σ valid for $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$. If $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U}$ has a call transition on ξ then there exists an extended substitution σ' valid for $\Gamma';\Delta \vdash \vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}' \sim \vec{B}'/\vec{\mathscr{F}}'/N'/\vec{V}'$ such that:*

1. $[\vec{A}/\vec{\mathscr{E}}/M/\vec{U}]_\sigma = [\vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}']_{\sigma'}$
2. $[\Gamma;\Delta \vdash \vec{B}/\vec{\mathscr{F}}/N/\vec{V}]_\sigma \xrightarrow{\tau}{}^* [\Gamma';\Delta \vdash \vec{B}'/\vec{\mathscr{F}}'/N'/\vec{V}']_{\sigma'}$
3. *One of the following holds:*
   (a) ***Reduction:*** $\vec{A}'/M' \longrightarrow$.
   (b) ***Call:*** $\Gamma';\Delta \vdash \vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}'$ *has a call transition on ξ' and $\xi' \notin dom(\sigma')$.*

PROOF. By induction on $\mathsf{varchain}(\xi,\sigma,\vec{U})$. If $\mathsf{varchain}(\xi,\sigma,\vec{U})=0$, then $\xi \notin dom(\sigma)$ and the original extended substitution and configurations satisfy requirements (1)-(3), so we are done. Now suppose $\mathsf{varchain}(\xi,\sigma,\vec{U})>0$, so $\xi \in dom(\sigma)$. The given bisimilarity $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$ allows us to deduce the existence of $\vec{C}$ and L such that $\vec{B}/N \twoheadrightarrow \vec{C}/L$, the configuration $\Gamma;\Delta \vdash \vec{C}/\vec{\mathscr{F}}/L/\vec{V}$ has a call transition on ξ, and $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U} \sim \vec{C}/\vec{\mathscr{F}}/L/\vec{V}$. There are two cases to consider: ξ may be an ordinary variable ϕ or an advice variable α.

Case ξ is an ordinary variable. If ξ is an ordinary variable ϕ, the call transition is $\mathsf{fcall}\ \phi$, and there exist evaluation contexts $\mathscr{E}$, $\mathscr{F}$ and values U, V such that $M = \mathscr{E}[\phi\ U]$, $L = \mathscr{F}[\phi\ V]$, and we have the following sequence (where ψ is fresh):

$$\begin{array}{rl} & \Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U} \\ = & \Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/\mathscr{E}[\phi\ U]/\vec{U} \\ \xrightarrow{\mathsf{fcall}\ \phi} & \Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}},\mathscr{E}/U/\vec{U} \\ \xrightarrow{\mathsf{put}} & \Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}},\mathscr{E}/U/\vec{U},U \\ \xrightarrow{\mathsf{get}\ \sigma(\phi)} & \Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}},\mathscr{E}/U_{\sigma(\phi)}/\vec{U},U \\ \xrightarrow{\mathsf{app}\ \psi} & \Gamma,\psi;\Delta \vdash \vec{A}/\vec{\mathscr{E}},\mathscr{E}/U_{\sigma(\phi)}\ \psi/\vec{U},U \end{array}$$

Since $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U} \sim \vec{C}/\vec{\mathscr{F}}/L/\vec{V}$ and $L = \mathscr{F}[\phi\ V]$, there is a corresponding sequence from $\Gamma;\Delta \vdash \vec{C}/\vec{\mathscr{F}}/L/\vec{V}$ that yields:

$$\Gamma,\psi;\Delta \vdash \vec{A}/\vec{\mathscr{E}},\mathscr{E}/U_{\sigma(\phi)}\ \psi/\vec{U},U \sim \vec{C}/\vec{\mathscr{F}},\mathscr{F}/V_{\sigma(\phi)}\ \psi/\vec{V},V$$

We obtain the extended substitution σ'' by replacing the stack of $(\psi \mapsto |\vec{U}|+1) \uplus \sigma$ with $(\vec{m},m+1)$, where the stack from σ is $(\vec{m},m)$. Then:

$$[\vec{A}/\vec{\mathscr{E}},\mathscr{E}/U_{\sigma(\phi)}\ \psi/\vec{U},U]_{\sigma''} = [\vec{A}/\vec{\mathscr{E}}/\mathscr{E}[U_{\sigma(\phi)}\ U]/\vec{U}]_{\sigma} = [\vec{A}/\vec{\mathscr{E}}/\mathscr{E}[\phi\ U]/\vec{U}]_{\sigma} = [\vec{A}/\vec{\mathscr{E}}/M/\vec{U}]_{\sigma}$$

Similarly $[\vec{C}/\vec{\mathscr{F}},\mathscr{F}/V_{\sigma(\phi)}\ \psi/\vec{V},V]_{\sigma''} = [\vec{C}/\vec{\mathscr{F}}/L/\vec{V}]_{\sigma}$. There are two subcases, depending on whether or not the value $U_{\sigma(\phi)}$ is a λ-abstraction. In the first subcase, if $U_{\sigma(\phi)} = \lambda x.M''$ is a λ-abstraction, then $\vec{A}/(U_{\sigma(\phi)}\ \psi) \twoheadrightarrow \vec{A}/(M''[x := \psi])$. Therefore we set $\sigma' = \sigma''$, $\Gamma' = (\Gamma,\psi)$, and:

$$\begin{array}{l} (\vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}') = (\vec{A}/\vec{\mathscr{E}},\mathscr{E}/U_{\sigma(\phi)}\ \psi/\vec{U},U) \\ (\vec{B}'/\vec{\mathscr{F}}'/N'/\vec{V}') = (\vec{C}/\vec{\mathscr{F}},\mathscr{F}/V_{\sigma(\phi)}\ \psi/\vec{V},V) \end{array}$$

Requirements (1) and (3) are immediately satisfied. Requirement (2) follows using Lemma 42 and the earlier reduction $\vec{B}/N \twoheadrightarrow \vec{C}/L$:

$$\begin{array}{rl} & [\Gamma;\Delta \vdash \vec{B}/\vec{\mathscr{F}}/N/\vec{V}]_{\sigma} \\ \xrightarrow{\tau}{}^{*} & [\Gamma;\Delta \vdash \vec{C}/\vec{\mathscr{F}}/L/\vec{V}]_{\sigma} \\ = & [\Gamma,\psi;\Delta \vdash \vec{C}/\vec{\mathscr{F}},\mathscr{F}/V_{\sigma(\phi)}\ \psi/\vec{V},V]_{\sigma''} \\ = & [\Gamma';\Delta \vdash \vec{B}'/\vec{\mathscr{F}}'/N'/\vec{V}']_{\sigma'} \end{array}$$

This completes the first subcase. For the second subcase, since $U_{\sigma(\phi)}$ is not a λ-abstraction, it must be either a variable x or a symbolic advice call $\alpha\langle U''\rangle$. We define ξ'' by either

$\xi'' = x$ or $\xi'' = \alpha$ as appropriate. Then $\Gamma, \phi; \Delta \vdash \vec{A}/\vec{\mathscr{E}}, \mathscr{E}/U_{\sigma(\phi)}\ \psi/\vec{U}, U$ has a call transition on ξ''. Moreover, the variable chain length for ξ'' is strictly smaller than that for ϕ because:

$$\mathsf{varchain}(\phi, \sigma, \vec{U}) = 1 + \mathsf{varchain}(\xi'', \sigma, \vec{U}) = 1 + \mathsf{varchain}(\xi'', \sigma'', (\vec{U}, U))$$

Applying the induction hypothesis to σ'' and:

$$\Gamma, \psi; \Delta \vdash \vec{A}/\vec{\mathscr{E}}, \mathscr{E}/U_{\sigma(\phi)}\ \psi/\vec{U}, U \sim \vec{C}/\vec{\mathscr{F}}, \mathscr{F}/V_{\sigma(\phi)}\ \psi/\vec{V}, V$$

yields σ' and: $\Gamma'; \Delta \vdash \vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}' \sim \vec{B}'/\vec{\mathscr{F}}'/N'/\vec{V}'$ such that:

- $[\vec{A}/\vec{\mathscr{E}}, \mathscr{E}/U_{\sigma(\phi)}\ \psi/\vec{U}, U]_{\sigma''} = [\vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}']_{\sigma'}$
- $[\Gamma, \psi; \Delta \vdash \vec{C}/\vec{\mathscr{F}}, \mathscr{F}/V_{\sigma(\phi)}\ \psi/\vec{V}, V]_{\sigma''} \xrightarrow{\tau}{}^{*} [\Gamma'; \Delta \vdash \vec{B}'/\vec{\mathscr{F}}'/N'/\vec{V}']_{\sigma'}$
- - Either $\vec{A}'/M' \rightarrow$.
  - Or $\Gamma'; \Delta \vdash \vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}'$ has a call transition on ξ' and $\xi' \notin dom(\sigma')$.

Hence, requirement (3) is satisfied immediately, and requirement (1) follows by:

$$[\vec{A}/\vec{\mathscr{E}}/M/\vec{U}]_{\sigma} = [\vec{A}/\vec{\mathscr{E}}, \mathscr{E}/U_{\sigma(\phi)}\ \psi/\vec{U}, U]_{\sigma''} = [\vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}']_{\sigma'}$$

Requirement (2) follows using Lemma 42 and the earlier reduction $\vec{B}/N \twoheadrightarrow \vec{C}/L$:

$$\begin{array}{ll} & [\Gamma; \Delta \vdash \vec{B}/\vec{\mathscr{F}}/N/\vec{V}]_{\sigma} \\ \xrightarrow{\tau}{}^{*} & [\Gamma; \Delta \vdash \vec{C}/\vec{\mathscr{F}}/L/\vec{V}]_{\sigma} \\ = & [\Gamma, \psi; \Delta \vdash \vec{C}/\vec{\mathscr{F}}, \mathscr{F}/V_{\sigma(\phi)}\ \psi/\vec{V}, V]_{\sigma''} \\ \xrightarrow{\tau}{}^{*} & [\Gamma'; \Delta \vdash \vec{B}'/\vec{\mathscr{F}}'/N'/\vec{V}']_{\sigma'} \end{array}$$

This completes the second subcase (when $U_{\sigma(\phi)}$ is not a λ-abstraction), and also the case when ξ is a variable ϕ.

Case ξ is an advice variable. In the second case ξ is an advice variable α, so the call transition is $\mathsf{acall}\ \alpha$, and there exist evaluation contexts $\mathscr{E}$, $\mathscr{F}$ and values U, V such that $M = \mathscr{E}[\alpha\langle W_1\rangle\ U]$ and $L = \mathscr{F}[\alpha\langle W_2\rangle\ V]$. In addition, σ is valid for both configurations, so both $U_{\sigma(\alpha)}$ and $V_{\sigma(\alpha)}$ must be abstractions with bodies that are also values, i.e., there exist W_3 and W_4 such that $U_{\sigma(\alpha)} = \lambda x.W_3$ and $V_{\sigma(\alpha)} = \lambda x.W_4$. Thus we have the following sequence (where ψ_1, ψ_2 are fresh):

$$\begin{array}{ll} & \Gamma; \Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U} \\ = & \Gamma; \Delta \vdash \vec{A}/\vec{\mathscr{E}}/\mathscr{E}[\alpha\langle W_1\rangle\ U]/\vec{U} \\ \xrightarrow{\mathsf{acall}\ \alpha} & \Gamma; \Delta \vdash \vec{A}/\vec{\mathscr{E}}, \mathscr{E}/U/\vec{U}, W_1 \\ \xrightarrow{\mathsf{put}} & \Gamma; \Delta \vdash \vec{A}/\vec{\mathscr{E}}, \mathscr{E}/U/\vec{U}, W_1, U \\ \xrightarrow{\mathsf{get}\ \sigma(\alpha)} & \Gamma; \Delta \vdash \vec{A}/\vec{\mathscr{E}}, \mathscr{E}/U_{\sigma(\alpha)}/\vec{U}, W_1, U \\ \xrightarrow{\mathsf{app}\ \psi_1} & \Gamma, \psi_1; \Delta \vdash \vec{A}/\vec{\mathscr{E}}, \mathscr{E}/U_{\sigma(\alpha)}\ \psi_1/\vec{U}, W_1, U \\ = & \Gamma, \psi_1; \Delta \vdash \vec{A}/\vec{\mathscr{E}}, \mathscr{E}/(\lambda x.W_3)\ \psi_1/\vec{U}, W_1, U \\ \xrightarrow{\tau} & \Gamma, \psi_1; \Delta \vdash \vec{A}/\vec{\mathscr{E}}, \mathscr{E}/W_3[x := \psi_1]/\vec{U}, W_1, U \\ \xrightarrow{\mathsf{app}\ \psi_2} & \Gamma, \psi_1, \psi_2; \Delta \vdash \vec{A}/\vec{\mathscr{E}}, \mathscr{E}/(W_3[x := \psi_1])\ \psi_2/\vec{U}, W_1, U \end{array}$$

In the above we use the fact that $W_3[x := \psi_1]$ is a value to justify the app ψ_2 transition. Since $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U} \sim \vec{C}/\vec{\mathscr{F}}/L/\vec{V}$ and $L = \mathscr{F}[\alpha{<}W_2{>}\, V]$, there is a corresponding sequence from $\Gamma;\Delta \vdash \vec{C}/\vec{\mathscr{F}}/L/\vec{V}$, and because reduction is deterministic we conclude:

$$\Gamma, \psi_1, \psi_2; \Delta \vdash \vec{A}/\vec{\mathscr{E}}, \mathscr{E}/(W_3[x := \psi_1])\ \psi_2/\vec{U}, W_1, U \sim \vec{C}/\vec{\mathscr{F}}, \mathscr{F}/(W_4[x := \psi_1])\ \psi_2/\vec{V}, W_2, V$$

We obtain the extended substitution σ'' by replacing the stack of the following extended substitution with $(\vec{m}, m+1)$, where the stack from σ is $(\vec{m}, m)$:

$$(\psi_2 \mapsto |\vec{U}| + 2) \uplus ((\psi_1 \mapsto |\vec{U}| + 1) \uplus \sigma)$$

The extended substitution σ'' is valid for the configurations above because ψ_1, ψ_2 were fresh. With this substitution we have:

$$\begin{aligned}
&[\vec{A}/\vec{\mathscr{E}}, \mathscr{E}/(W_3[x := \psi_1])\ \psi_2/\vec{U}, W_1, U]_{\sigma''} \\
&= [\vec{A}/\vec{\mathscr{E}}, \mathscr{E}/(W_3[x := W_1])\ U/\vec{U}, W_1, U]_{\sigma''} \\
&= [\vec{A}/\vec{\mathscr{E}}/\mathscr{E}[(W_3[x := W_1])\ U]/\vec{U}]_{\sigma} \\
&= [\vec{A}/\vec{\mathscr{E}}/\mathscr{E}[(W_3[x := (W_1[\alpha := \lambda x.W_3])])\ U]/\vec{U}]_{\sigma} \\
&= [\vec{A}/\vec{\mathscr{E}}/\mathscr{E}[\alpha{<}W_1{>}[\alpha := \lambda x.W_3]\ U]/\vec{U}]_{\sigma} \\
&= [\vec{A}/\vec{\mathscr{E}}/\mathscr{E}[\alpha{<}W_1{>}[\alpha := U_{\sigma(\alpha)}]\ U]/\vec{U}]_{\sigma} \\
&= [\vec{A}/\vec{\mathscr{E}}/\mathscr{E}[\alpha{<}W_1{>}\, U]/\vec{U}]_{\sigma} \\
&= [\vec{A}/\vec{\mathscr{E}}/M/\vec{U}]_{\sigma}
\end{aligned}$$

Similarly:

$$[\vec{C}/\vec{\mathscr{F}}, \mathscr{F}/(W_4[x := \psi_1])\ \psi_2/\vec{V}, W_2, V]_{\sigma''} = [\vec{C}/\vec{\mathscr{F}}/L/\vec{V}]_{\sigma}$$

There are two subcases, depending on whether or not the value $W_3[x := \psi_1]$ is a λ-abstraction. Both subcases use the same reasoning as the corresponding subcases when ξ is an ordinary variable, i.e., when $W_3[x := \psi_1]$ is a λ-abstraction, a beta reduction applies immediately and we are done; and when $W_3[x := \psi_1]$ is not a λ-abstraction, the induction hypothesis is applied to σ'' and the bisimilar pair of configurations above, yielding the desired results by transitivity. This completes the case when ξ is an advice variable, and completes the proof of Lemma 46.

We now have an immediate corollary that is used to provide a suitable underlying bisimilar configuration when given a $\approx$-related configuration.

Corollary 47. *Given an extended substitution σ valid for $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$, there exists an extended substitution σ' valid for $\Gamma';\Delta \vdash \vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}' \sim \vec{B}'/\vec{\mathscr{F}}'/N'/\vec{V}'$ such that:*

1. $[\vec{A}/\vec{\mathscr{E}}/M/\vec{U}]_{\sigma} = [\vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}']_{\sigma'}$
2. $[\Gamma;\Delta \vdash \vec{B}/\vec{\mathscr{F}}/N/\vec{V}]_{\sigma} \xrightarrow{\tau}{}^{*} [\Gamma';\Delta \vdash \vec{B}'/\vec{\mathscr{F}}'/N'/\vec{V}']_{\sigma'}$
3. *One of the following holds:*
   (a) ***Reduction:*** $\vec{A}'/M' \longrightarrow$.
   (b) ***Value:*** *The σ' stack has the form $(\vec{m}, 0)$ and M' is a value.*
   (c) ***Call:*** $\Gamma';\Delta \vdash \vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}'$ *has a call transition on ξ' and $\xi' \notin dom(\sigma')$.*

PROOF. Consider $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U}$. If $\vec{A}/M \not\rightarrow$, the original extended substitution and configurations satisfy requirements (1)-(3), and we are done. Otherwise M is a value or $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U}$ has a call transition on some ξ. In the first case, the result follows immediately by Lemma 43. In the second case, the result follows immediately by Lemma 46. This completes the proof of Corollary 47.

Note that underlying reductions in (3)(a) of Corollary 47 can arise in three different ways. This corresponds to the different ways in which $\approx$-related configurations may have reductions.

Lemma 48 provides the core of the substitution result in Theorem 49. It shows that a weak transition from one side of a $\approx$-related configuration is matched by a weak transition from the other side, and the results are also $\approx$-related. This proof relies upon the mimicking of internal τ-reductions using LTS transitions. One way to view the case of the proof for application is as our version of the delayed substitutions of the SECD machine [37].

Lemma 48. *Given an extended substitution σ valid for $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$, if there is a weak $\kappa(\neq\tau)$ transition to a configuration $\mathbf{M}$:*

$$[\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U}]_\sigma \stackrel{\kappa}{\Rightarrow} \Gamma_1;\Delta_1 \vdash \mathbf{M}$$

then there exists a configuration $\mathbf{N}$ such that:

$$[\Gamma;\Delta \vdash \vec{B}/\vec{\mathscr{F}}/N/\vec{V}]_\sigma \stackrel{\kappa}{\Rightarrow} \Gamma_1;\Delta_1 \vdash \mathbf{N}$$

and:

$$\Gamma_1;\Delta_1 \vdash \mathbf{M} \approx \mathbf{N}$$

PROOF. By induction on the length of the reduction sequence in $[\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U}]_\sigma \stackrel{\kappa}{\Rightarrow} \Gamma_1;\Delta_1 \vdash \mathbf{M}$. Using Corollary 47, there exists an extended substitution σ' valid for $\Gamma';\Delta \vdash \vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}' \sim \vec{B}'/\vec{\mathscr{F}}'/N'/\vec{V}'$ such that:

- $[\vec{A}/\vec{\mathscr{E}}/M/\vec{U}]_\sigma = [\vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}']_{\sigma'}$
- $[\Gamma;\Delta \vdash \vec{B}/\vec{\mathscr{F}}/N/\vec{V}]_\sigma \stackrel{\tau}{\rightarrow}{}^* [\Gamma';\Delta \vdash \vec{B}'/\vec{\mathscr{F}}'/N'/\vec{V}']_{\sigma'}$
- One of the following holds:
  - ***Reduction:*** $\vec{A}'/M' \not\rightarrow$.
  - ***Value:*** The σ' stack has the form $(\vec{m},0)$ and M' is a value.
  - ***Call:*** $\Gamma';\Delta \vdash \vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}'$ has a call transition on ξ' and $\xi' \notin dom(\sigma')$.

Then we have:

$$[\Gamma';\Delta \vdash \vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}']_{\sigma'} = [\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U}]_\sigma \stackrel{\kappa}{\Rightarrow} \Gamma_1;\Delta_1 \vdash \mathbf{M}$$

We now consider the three possibilities given by Corollary 47: *Reduction*, *Value*, and *Call*. The *Reduction* case uses the induction hypothesis, and the *Value* and *Call* cases are the base cases where there are no reduction steps.

Case: Reduction. If $\vec{A}'/M' \to \vec{A}''/M''$ for some $\vec{A}''$ and M'', then by Lemma 42 we have:

$$[\Gamma';\Delta \vdash \vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}']_{\sigma'} \xrightarrow{\tau} [\Gamma';\Delta \vdash \vec{A}''/\vec{\mathscr{E}}'/M''/\vec{U}']_{\sigma'}$$

Moreover, reduction is deterministic, and the previous weak transition may be factored as:

$$[\Gamma';\Delta \vdash \vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}']_{\sigma'} \xrightarrow{\tau} [\Gamma';\Delta \vdash \vec{A}''/\vec{\mathscr{E}}'/M''/\vec{U}']_{\sigma'} \xRightarrow{\kappa} \Gamma_1;\Delta_1 \vdash \mathbf{M}$$

Since reduction is contained in bisimilarity, we also deduce $\Gamma';\Delta \vdash \vec{A}''/\vec{\mathscr{E}}'/M''/\vec{U}' \sim \vec{B}'/\vec{\mathscr{F}}'/N'/\vec{V}'$ from $\Gamma';\Delta \vdash \vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}' \sim \vec{B}'/\vec{\mathscr{F}}'/N'/\vec{V}'$. Thus we may apply the induction hypothesis to the strictly shorter reduction sequence:

$$[\Gamma';\Delta \vdash \vec{A}''/\vec{\mathscr{E}}'/M''/\vec{U}']_{\sigma'} \xRightarrow{\kappa} \Gamma_1;\Delta_1 \vdash \mathbf{M}$$

This yields a configuration $\mathbf{N}$ such that:

$$[\Gamma';\Delta \vdash \vec{B}'/\vec{\mathscr{F}}'/N'/\vec{V}']_{\sigma'} \xRightarrow{\kappa} \Gamma_1;\Delta_1 \vdash \mathbf{N}$$

and:

$$\Gamma_1;\Delta_1 \vdash \mathbf{M} \approx \mathbf{N}$$

The result follows from the previous reduction sequence by transitivity:

$$[\Gamma;\Delta \vdash \vec{B}/\vec{\mathscr{F}}/N/\vec{V}]_{\sigma} \xrightarrow{\tau}{}^{*} [\Gamma';\Delta \vdash \vec{B}'/\vec{\mathscr{F}}'/N'/\vec{V}']_{\sigma'} \xRightarrow{\kappa} \Gamma_1;\Delta_1 \vdash \mathbf{N}$$

This completes the *Reduction* case.

Case: Value. If the σ' stack has the form $(\vec{m},0)$ and M' is a value, then there exist $\vec{C}'$ and V' such that $\vec{B}'/N' \twoheadrightarrow \vec{C}'/V'$ and:

$$\Gamma';\Delta \vdash \vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}' \sim \vec{C}'/\vec{\mathscr{F}}'/V'/\vec{V}'$$

Next, define:

$$\Gamma'';\Delta \vdash \vec{A}''/\vec{\mathscr{E}}''/M''/\vec{U}'' \triangleq [\Gamma';\Delta \vdash \vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}']_{\sigma'}$$
$$\Gamma'';\Delta \vdash \vec{B}''/\vec{\mathscr{F}}''/N''/\vec{V}'' \triangleq [\Gamma';\Delta \vdash \vec{C}'/\vec{\mathscr{F}}'/V'/\vec{V}']_{\sigma'}$$

Now the top of the stack of σ' is empty and M', V' are values, so M'' and N'' are also values. Therefore κ must be a transition label with one of the following forms: $\mathsf{ret}\ \phi$, $\mathsf{app}\ \phi$, put, $\mathsf{get}\ i$, $\mathsf{fun}\ f@p = \phi$, $\mathsf{adv}\ p = \alpha$ (where all names other p are fresh).

Subcase: $\kappa = \mathsf{ret}\ \phi$. In this subcase, we must have the decomposition $\vec{\mathscr{E}}'' = (\vec{\mathscr{E}}''', \mathscr{E}''')$, for some $\vec{\mathscr{E}}'''$, $\mathscr{E}'''$, so that:

$$\Gamma'';\Delta \vdash \vec{A}''/\vec{\mathscr{E}}''/M''/\vec{U}'' \xrightarrow{\mathsf{ret}\ \phi} \Gamma'',\phi;\Delta \vdash \vec{A}''/\vec{\mathscr{E}}'''/\mathscr{E}'''[\phi]/\vec{U}'' = \Gamma_1;\Delta_1 \vdash \mathbf{M}$$

Similarly, we must also have $\vec{\mathscr{F}}'' = (\vec{\mathscr{F}}''', \mathscr{F}''')$, for some $\vec{\mathscr{F}}'''$, $\mathscr{F}'''$, as well as the following transition:

$$\Gamma'';\Delta \vdash \vec{B}''/\vec{\mathscr{F}}''/N''/\vec{V}'' \xrightarrow{\mathsf{ret}\ \phi} \Gamma'',\phi;\Delta \vdash \vec{B}''/\vec{\mathscr{F}}'''/\mathscr{F}'''[\phi]/\vec{V}''$$

We then define $\mathbf{N} \triangleq (\vec{B}''/\vec{\mathscr{F}}'''/\mathscr{F}'''[\phi]/\vec{V}'')$. To establish $\Gamma_1;\Delta_1 \vdash \mathbf{M} \approx \mathbf{N}$, we also use $\mathsf{ret}\ \phi$ transitions on the underlying bisimilar configurations. Since $|\vec{\mathscr{E}}''| = |(\vec{\mathscr{E}}''', \mathscr{E}''')| > 0$, the stack of σ' must have the form $((\vec{m},m),0)$, where $m > 0$ by definition, so $|\vec{\mathscr{E}}'| = \mathit{sum}(\sigma') > 0$. Hence, $\vec{\mathscr{E}}' = (\vec{\mathscr{G}}_1, \mathscr{G}_1)$, for some $\vec{\mathscr{G}}_1$, $\mathscr{G}_1$, and we have the transition:

$$\Gamma';\Delta \vdash \vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}' \xrightarrow{\mathsf{ret}\ \phi} \Gamma', \phi;\Delta \vdash \vec{A}'/\vec{\mathscr{G}}_1/\mathscr{G}_1[\phi]/\vec{U}'$$

By bisimilarity, we have $\vec{\mathscr{F}}' = (\vec{\mathscr{G}}_2, \mathscr{G}_2)$, for some $\vec{\mathscr{G}}_2$, $\mathscr{G}_2$, and the transition:

$$\Gamma';\Delta \vdash \vec{C}'/\vec{\mathscr{F}}'/V'/\vec{V}' \xrightarrow{\mathsf{ret}\ \phi} \Gamma', \phi;\Delta \vdash \vec{C}'/\vec{\mathscr{G}}_2/\mathscr{G}_2[\phi]/\vec{V}'$$

such that:

$$\Gamma';\Delta \vdash \vec{A}'/\vec{\mathscr{G}}_1/\mathscr{G}_1[\phi]/\vec{U}' \sim \vec{C}'/\vec{\mathscr{G}}_2/\mathscr{G}_2[\phi]/\vec{V}'$$

We obtain a new extended substitution σ'' from σ' by changing the stack from $((\vec{m},m),0)$ to $(\vec{m},m-1)$ (recall that $m > 0$ so $m-1 \geq 0$). The extended substitution σ'' is valid for the pair of bisimilar configurations above because $\mathit{sum}(\sigma') = \mathit{sum}(\sigma'') + 1$ and $|\vec{\mathscr{E}}'| = |\vec{\mathscr{G}}_1| + 1$, $|\vec{\mathscr{F}}'| = |\vec{\mathscr{G}}_2| + 1$. With the extended substitution σ'' we have:

$$\begin{array}{ll} & \mathbf{M} \\ = & (\vec{A}''/\vec{\mathscr{E}}'''/\mathscr{E}'''[\phi]/\vec{U}'') \\ = & [\vec{A}'/\vec{\mathscr{G}}_1/\mathscr{G}_1[\phi]/\vec{U}']_{\sigma''} \\ \approx & [\vec{C}'/\vec{\mathscr{G}}_2/\mathscr{G}_2[\phi]/\vec{V}']_{\sigma''} \\ = & (\vec{B}''/\vec{\mathscr{F}}'''/\mathscr{F}'''[\phi]/\vec{V}'') \\ = & \mathbf{N} \end{array}$$

Finally, by Lemma 42 and $\vec{B}'/N' \twoheadrightarrow \vec{C}'/V'$:

$$\begin{array}{ll} & [\Gamma';\Delta \vdash \vec{B}'/\vec{\mathscr{F}}'/N'/\vec{V}']_{\sigma'} \\ \xrightarrow{\tau}{}^* & [\Gamma';\Delta \vdash \vec{C}'/\vec{\mathscr{F}}'/V'/\vec{V}']_{\sigma'} \\ = & \Gamma'';\Delta \vdash \vec{B}''/\vec{\mathscr{F}}''/N''/\vec{V}'' \\ \xrightarrow{\mathsf{ret}\ \phi} & \Gamma_1;\Delta_1 \vdash \mathbf{N} \end{array}$$

Together with $\Gamma_1;\Delta_1 \vdash \mathbf{M} \approx \mathbf{N}$, this completes the subcase.

Subcases: $\kappa \in \{\mathsf{app}\ \phi, \mathsf{put}, \mathsf{get}\ i, \mathsf{fun} f @ p = \phi, \mathsf{adv}\ p = \alpha\}$. In each of these subcases, similar reasoning to that for the $\mathsf{ret}\ \phi$ subcase aboves applies. The (simplifying) difference is that these transitions leave the evaluation stack unchanged, and so $\sigma'' \triangleq \sigma'$.

This completes the *Value* case.

Case: Call. $\Gamma';\Delta \vdash \vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}'$ has a call transition on ξ' and $\xi' \notin \mathit{dom}(\sigma')$. The subcases when $\kappa = \mathsf{fcall}\ \phi$ and $\kappa = \mathsf{acall}\ \alpha$ are very similar. We present the latter subcase and omit the former.

Subcase: $\kappa = \mathsf{acall}\ \alpha$. Since there is a call transition on $\xi' = \alpha$, we have $M' = \mathscr{E}'[\alpha\langle W_1 \rangle W_1']$ and:

$$\Gamma';\Delta \vdash \vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}' \xrightarrow{\mathsf{acall}\ \alpha} \Gamma';\Delta \vdash \vec{A}'/\vec{\mathscr{E}}', \mathscr{E}'/W_1'/\vec{U}', W_1$$

By bisimilarity we have $\vec{B}'/N' \twoheadrightarrow \vec{C}'/L'$, for $L' = \mathscr{F}'[\alpha\!<\!W_2\!>\! W_2']$, and:

$$\Gamma';\Delta \vdash \vec{B}'/\vec{\mathscr{F}}'/N'/\vec{V}' \xrightarrow{\mathsf{acall}\ \alpha}\!\!\!\!\twoheadrightarrow \Gamma';\Delta \vdash \vec{C}'/\vec{\mathscr{F}}',\mathscr{F}'/W_2'/\vec{V}',W_2$$

such that:

$$\Gamma';\Delta \vdash \vec{A}'/\vec{\mathscr{E}}',\mathscr{E}'/W_1'/\vec{U}',W_1 \sim \vec{C}'/\vec{\mathscr{F}}',\mathscr{F}'/W_2'/\vec{V}',W_2$$

Now suppose that the stack of σ' is $(\vec{m},m)$ and that $Z_{\sigma'}(\vec{\mathscr{E}}') = (\vec{\mathscr{E}}'',\mathscr{E}'')$. We write $(\cdot)^\dagger$ for the result of applying the σ' substitutions from $\vec{U}'$ on a term, value list, evaluation context list, and declarations list. We obtain the new extended substitution σ'' from σ' by modifying the stack from $(\vec{m},m)$ to $((\vec{m},m+1),0)$, so $Z_{\sigma''}(\vec{\mathscr{E}}',\mathscr{E}') = (\vec{\mathscr{E}}'',\mathscr{E}''[\mathscr{E}'],[\text{–}])$. The extended substitution σ'' is valid for the bisimilar configurations above. Now since $\alpha \notin dom(\sigma')$ we have, for some Γ_1:

$$\begin{array}{rl}
 & [\Gamma';\Delta \vdash \vec{A}'/\vec{\mathscr{E}}'/M'/\vec{U}']_{\sigma'} \\
= & \Gamma_1;\Delta \vdash \vec{A}'^\dagger/\vec{\mathscr{E}}''^\dagger/\mathscr{E}''[M']^\dagger/\vec{U}'^\dagger \\
= & \Gamma_1;\Delta \vdash \vec{A}'^\dagger/\vec{\mathscr{E}}''^\dagger/\mathscr{E}''[\mathscr{E}'[\alpha\!<\!W_1\!>\! W_1']]^\dagger/\vec{U}'^\dagger \\
\xrightarrow{\mathsf{acall}\ \alpha} & \Gamma_1;\Delta \vdash \vec{A}'^\dagger/\vec{\mathscr{E}}''^\dagger,\mathscr{E}''[\mathscr{E}']^\dagger/W_1'^\dagger/\vec{U}',W_1{}^\dagger \\
= & [\Gamma';\Delta \vdash \vec{A}'/\vec{\mathscr{E}}',\mathscr{E}'/W_1'/\vec{U}',W_1]_{\sigma''} \\
= & \Gamma_1;\Delta \vdash \mathbf{M}
\end{array}$$

Similarly, with the additional use of Lemma 42:

$$[\Gamma';\Delta \vdash \vec{B}'/\vec{\mathscr{F}}'/N'/\vec{V}']_{\sigma'} \xrightarrow{\mathsf{acall}\ \alpha}\!\!\!\!\twoheadrightarrow [\Gamma';\Delta \vdash \vec{C}'/\vec{\mathscr{F}}',\mathscr{F}'/W_2'/\vec{V}',W_2]_{\sigma''}$$

Thus we set $\mathbf{N} \triangleq [\Gamma';\Delta \vdash \vec{C}'/\vec{\mathscr{F}}',\mathscr{F}'/W_2'/\vec{V}',W_2]_{\sigma''}$ and we know $\Gamma_1;\Delta \vdash \mathbf{M} \approx \mathbf{N}$ from:

$$\Gamma';\Delta \vdash \vec{A}'/\vec{\mathscr{E}}',\mathscr{E}'/W_1'/\vec{U}',W_1 \sim \vec{C}'/\vec{\mathscr{F}}',\mathscr{F}'/W_2'/\vec{V}',W_2$$

This completes the $\mathsf{acall}\ \alpha$ subcase, the *Call* case, and the proof of Lemma 48.

Finally, we state and prove the substitution result.

Theorem 49 (Substitution). *The relation* $\approx\ = \bigcup_\sigma \approx_\sigma$ *is a bisimulation.*

PROOF. We show that $\approx$ is a bisimulation by coinduction. Consider $\Gamma;\Delta \vdash \mathbf{M} \approx \mathbf{N}$. For the left-to-right direction, if there is a weak transition $\Gamma;\Delta \vdash \mathbf{M} \xrightarrow{\kappa'}\!\!\!\!\twoheadrightarrow \Gamma';\Delta' \vdash \mathbf{M}'$ for $\kappa' \neq \tau$, then, by Lemma 48, there exists a configuration $\mathbf{N}'$ such that: $\Gamma;\Delta \vdash \mathbf{N} \xrightarrow{\kappa'}\!\!\!\!\twoheadrightarrow \Gamma';\Delta' \vdash \mathbf{N}'$ and $\Gamma';\Delta' \vdash \mathbf{M}' \approx \mathbf{N}'$. For the right-to-left direction, since bisimilarity is symmetric, $\approx$ is also symmetric. Hence, if there is a weak transition $\Gamma;\Delta \vdash \mathbf{N} \xrightarrow{\kappa''}\!\!\!\!\twoheadrightarrow \Gamma'';\Delta'' \vdash \mathbf{N}''$ for $\kappa'' \neq \tau$, then, by Lemma 48 and two uses of symmetry, there exists a configuration $\mathbf{M}''$ such that: $\Gamma;\Delta \vdash \mathbf{M} \xrightarrow{\kappa''}\!\!\!\!\twoheadrightarrow \Gamma'';\Delta'' \vdash \mathbf{M}''$ and $\Gamma'';\Delta'' \vdash \mathbf{M}'' \approx \mathbf{N}''$.

B Identity extension for terms

This section sketches the proof of Lemma 34. We first sketch an auxiliary lemma concerning the addition of a fresh public PCD and initial advice to bisimilar configurations.

Lemma 50. *If* $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$ *and* p, α *are fresh, then:* $\Gamma,\alpha;\Delta,\mathsf{pcd}\ p,\mathsf{adv}\ p=\alpha \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$.

PROOF. A straightforward bisimulation proof using Lemma 30. The bisimulation contains not only the configuration with the addition of α and $\mathsf{pcd}\ p,\mathsf{adv}\ p=\alpha$, but also functions and advice (at p) that can be added by the environment. The values resulting from looking up those functions are identical on both sides and state-free, and thus Lemma 30 allows them to be safely ignored. □

For the proof sketch of Lemma 34 in the rest of this section, we assume that:

- $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$
- M,N are values.
- All names in $fn(L)$ are bound in $\Gamma;\Delta$.
- $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U}$ and $\Gamma;\Delta \vdash \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$ are compatible.

The proof proceeds by structural induction on L.

Case x, x not bound in Δ. Already established in Lemma 30, which, amongst other things, shows that bisimilarity is closed under replacement of the active term by a state-free term with free variables bound in Γ. The variable x is such a state-free term.

Case f, f bound in Δ. $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U}$ and $\Gamma;\Delta \vdash \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$ have no τ-reductions.

The configurations are compatible, so let i be the index of the value list that has f in $\vec{U}$ and $\vec{V}$. Using transition $\mathsf{get}\ i$, we get:

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/f/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/f/\vec{V}$$

Case U V. The induction hypothesis on U yields:

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/U/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/U/\vec{V}$$

and hence using put:

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/U/\vec{U},U \sim \vec{B}/\vec{\mathscr{F}}/U/\vec{V},U$$

Using induction hypothesis on V yields:

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/V/\vec{U},U \sim \vec{B}/\vec{\mathscr{F}}/V/\vec{V},U$$

and hence using put

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/V/\vec{U},U,V \sim \vec{B}/\vec{\mathscr{F}}/V/\vec{V},U,V$$

The term $x_1\ x_2$ is state-free, so Lemma 30 yields:

$$\Gamma,x_1,x_2;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/x_1\ x_2/\vec{U},U,V \sim \vec{B}/\vec{\mathscr{F}}/x_1\ x_2/\vec{V},U,V$$

Consider substitution σ with stack $((),0)$ and partial function given by $\{x_i \mapsto i+|\vec{U}| \mid i=1,2\}$. Using Theorem 49 yields the required result.

Case $\mathsf{pcd}\ p\,;L$. Applying Lemma 50 to:

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/U/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/V/\vec{V}$$

gives:

$$\Gamma,\alpha;\Delta,\mathsf{pcd}\ p,\mathsf{adv}\ p=\alpha \vdash \vec{A}/\vec{\mathscr{E}}/U/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/V/\vec{V}$$

By the induction hypothesis on L:

$$\Gamma,\alpha;\Delta,\mathsf{pcd}\ p,\mathsf{adv}\ p=\alpha \vdash \vec{A}/\vec{\mathscr{E}}/L/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/L/\vec{V}$$

A bisimulation proof establishes:

$$\Gamma,\alpha;\Delta \vdash \vec{A},\mathsf{pcd}\ p,\mathsf{adv}\ p=\alpha/\vec{\mathscr{E}}/L/\vec{U} \sim \vec{B},\mathsf{pcd}\ p,\mathsf{adv}\ p=\alpha/\vec{\mathscr{F}}/L/\vec{V}$$

And, with $W = \lambda z.\lambda x.z\,x$, a second bisimulation proof using Lemma 30 yields:

$$\Gamma,\alpha;\Delta \vdash \vec{A},\mathsf{pcd}\ p,\mathsf{adv}\ p=\alpha/\vec{\mathscr{E}}/L/\vec{U},W \sim \vec{B},\mathsf{pcd}\ p,\mathsf{adv}\ p=\alpha/\vec{\mathscr{F}}/L/\vec{V},W$$

Substitution of W for α using Theorem 49 gives:

$$\Gamma;\Delta \vdash \vec{A},\mathsf{pcd}\ p,\mathsf{adv}\ p=W/\vec{\mathscr{E}}/L/\vec{U},W \sim \vec{B},\mathsf{pcd}\ p,\mathsf{adv}\ p=W/\vec{\mathscr{F}}/L/\vec{V},W$$

A final bisimulation proof shows:

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/\mathsf{pcd}\ p\,;L/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/\mathsf{pcd}\ p\,;L/\vec{V}$$

Case $\mathsf{fun}\ f\mathbin{@}p=U\,;L$. Using induction on U we deduce that:

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/U/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/U/\vec{V}$$

and hence:

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/U/\vec{U},U \sim \vec{B}/\vec{\mathscr{F}}/U/\vec{V},U$$

Using $\mathsf{fun}\ f\mathbin{@}p=\phi$ on both sides, we get:

$$\Gamma,\phi;\Delta,\mathsf{fun}\ f\mathbin{@}p=\phi \vdash \vec{A}/\vec{\mathscr{E}}/L/\vec{U},U \sim \vec{B}/\vec{\mathscr{F}}/L/\vec{V},U$$

Consider substitution σ with stack $((),0)$ and partial function given by $\{\phi \mapsto 1+|\vec{U}|\}$. Using Theorem 49 yields the required result.

Case $\mathsf{adv}\ p=U\,;L$. Similar to above, but using $\mathsf{adv}\ p=\alpha$ transitions instead of $\mathsf{fun}\ f\mathbin{@}p=\phi$.

Case $\mathsf{let}\ x=L_1\,;L_2$. For a fixed L_2, define a bisimulation candidate $\mathscr{R}$ to be the least relation containing bisimilarity and such that:

1. If $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$, then
   $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/\mathsf{let}\ x=M\,;L_2/\vec{U}\ \ \mathscr{R}\ \ \vec{B}/\vec{\mathscr{F}}/\mathsf{let}\ x=N\,;L_2/\vec{V}$.
2. If $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}},\mathscr{E},\vec{\mathscr{E}}'/M/\vec{U} \sim \vec{B}/\vec{\mathscr{F}},\mathscr{F},\vec{\mathscr{F}}'/N/\vec{V}$, then
   $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}},\mathsf{let}\ x=\mathscr{E}\,;L_2,\vec{\mathscr{E}}'/M/\vec{U}\ \ \mathscr{R}\ \ \vec{B}/\vec{\mathscr{F}},\mathsf{let}\ x=\mathscr{F}\,;L_2,\vec{\mathscr{F}}'/N/\vec{V}$.

We sketch the argument that $\mathscr{R}$ is in fact a bisimulation.

For (1), if $\vec{A}/M \twoheadrightarrow \vec{A}'/U$, then we can deduce $\vec{B}/N \twoheadrightarrow \vec{B}'/V$, for some $\vec{B}'$, V, and so we have both $\vec{A}/\mathsf{let}\ x = M\ ;\ L_2 \twoheadrightarrow \vec{A}'/L_2[x := U]$ and $\vec{B}/\mathsf{let}\ x = N\ ;\ L_2 \twoheadrightarrow \vec{B}'/L_2[x := V]$. We also have a bisimilarity $\Gamma;\Delta \vdash \vec{A}'/\vec{\mathscr{E}}/U/\vec{U} \sim \vec{B}'/\vec{\mathscr{F}}/V/\vec{V}$, and via introduction of x (assumed fresh, otherwise rename bound variables) and corresponding put transitions we have $\Gamma,x;\Delta \vdash \vec{A}'/\vec{\mathscr{E}}/U/\vec{U},U \sim \vec{B}'/\vec{\mathscr{F}}/V/\vec{V},V$. Applying the induction hypothesis for L_2 yields $\Gamma,x;\Delta \vdash \vec{A}'/\vec{\mathscr{E}}/L_2/\vec{U},U \sim \vec{B}'/\vec{\mathscr{F}}/L_2/\vec{V},V$. Consider substitution σ with stack $((),0)$ and partial function given by $\{x \mapsto 1 + |\vec{U}|\}$. Using Theorem 49 yields $\Gamma;\Delta \vdash \vec{A}'/\vec{\mathscr{E}}/L_2[x := U]/\vec{U} \sim \vec{B}'/\vec{\mathscr{F}}/L_2[x := V]/\vec{V}$.

For (1), when a call transition with evaluation context $\mathscr{E}$ occurs during the reduction of $\vec{A}/M$, there is a corresponding call transition (possibly after some reduction) with evaluation context $\mathscr{F}$ from $\vec{B}/N$ due to the underlying bisimilarity. To reflect, in $\mathscr{R}$, the corresponding call transitions that take place for $\vec{A}/\mathsf{let}\ x = M\ ;\ L_2$ and $\vec{B}/\mathsf{let}\ x = N\ ;\ L_2$, (2) is used with evaluation contexts $\mathsf{let}\ x = \mathscr{E}\ ;\ L_2$ and $\mathsf{let}\ x = \mathscr{F}\ ;\ L_2$ respectively.

For (2), the interesting case is for $\mathsf{ret}\ \phi$ transitions when $\vec{\mathscr{E}}' = \vec{\mathscr{F}}' = ()$. In this case, the $\mathsf{let}\ x = \mathscr{E}\ ;\ L_2$ and $\mathsf{let}\ x = \mathscr{F}\ ;\ L_2$ contexts are restored and (1) is used to continue.

Finally, once $\mathscr{R}$ is known to be a bisimulation, we note that the induction hypothesis for L_1 yields:

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/L_1/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/L_1/\vec{V}$$

and so:

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/\mathsf{let}\ x = L_1\ ;\ L_2/\vec{U}\ \ \mathscr{R}\ \ \vec{B}/\vec{\mathscr{F}}/\mathsf{let}\ x = L_1\ ;\ L_2/\vec{V}$$

Since $\mathscr{R}$ is a bisimulation, we conclude:

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/\mathsf{let}\ x = L_1\ ;\ L_2/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/\mathsf{let}\ x = L_1\ ;\ L_2/\vec{V}$$

Case $\lambda x.L$. Consider the relation that consists of all compatible pairs of configurations $(\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M'/\vec{U},\Gamma;\Delta \vdash \vec{B}/\vec{\mathscr{F}}/N'/\vec{V})$ such that there exists:

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/L_1/\vec{U}' \sim \vec{B}/\vec{\mathscr{F}}/L_2/\vec{V}'$$

such that:

- $\vec{U}'$ (resp. $\vec{V}'$) is obtained from $\vec{U}$ (resp. $\vec{V}$) by deleting all occurrences of $\lambda x.L$.
- The possibilities for L_1, L_2 are as follows
  - $L_1 = M'$ and $L_2 = N'$
  - Both M', N' are values, and one of the following hold:
    - $L_1 = L_2 = \lambda x.L$
    - $L_1 = L_2 = L[x := \phi]$

The required result follows from showing that this relation is a bisimulation. The straightforward proof to show this uses inductive hypothesis on L at all configurations having $L_1 = L_2 = L[x := \phi]$ in the active term position.

Inclusion of identical values and identical evaluation contexts. Since the first time when values from the value list (or contexts from the context list) can be moved into active position is when the term in the active position has become a value, and hence in the realm of applicability of Lemma 34, the addition of identical contexts and values can be done in slightly more general situations.

Corollary 51 (to Lemma 34). *If:*

- $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$
- *All names in* $fn(U), \mathscr{E}, \mathscr{E}'$ *are bound in* Γ, Δ
- $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U}$ *and* $\Gamma;\Delta \vdash \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$ *are compatible.*

then:

$$\Gamma;\Delta \vdash \vec{A}/\mathscr{E}', \vec{\mathscr{E}}, \mathscr{E}/M/\vec{U}, U \sim \vec{B}/\mathscr{E}', \vec{\mathscr{F}}, \mathscr{E}/N/\vec{V}, U$$

Corollary 51 is used in the proof of Lemma 35 given in Section C.

C Inclusion of identical contexts

In this section we prove Lemma 35. The proof relies on the following auxiliary lemma that allows the addition of new common symbolic advice $\mathsf{adv}\ p = \beta$ before existing common symbolic advice $\mathsf{adv}\ p = \alpha$.

Lemma 52. *If:*

- $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$
- $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U}$ *and* $\Gamma;\Delta \vdash \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$ *are compatible.*
- $\Delta = \Delta_1, \mathsf{adv}\ p = \alpha, \Delta_2$.
- α *does not occur in* Δ_1, Δ_2, $\vec{A}/\vec{\mathscr{E}}/M/\vec{U}$, $\vec{B}/\vec{\mathscr{F}}/N/\vec{V}$.

Then, for fresh β*:*

$$\Gamma, \beta; \Delta_1, \mathsf{adv}\ p = \beta, \mathsf{adv}\ p = \alpha, \Delta_2 \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$$

PROOF. By bisimulation. We first define a function $\{|\}$ that replaces occurrences of α<U> with α<β<U>> in terms, values, contexts, and declarations, i.e., $\{|\}$ is the homomorphic, capture-avoiding extension of:

$$\{|\alpha\text{<}U\text{>}|\} = \alpha\text{<}\beta\text{<}\{|U|\}\text{>>}$$

Although $\{|\alpha|\} = \alpha$ this fact is not needed in the sequel because of the restrictions on occurrences of α. Consider $\vec{C}$ and L such that all occurrences of α in $\vec{C}$ and L have the form α<U>, and where there are no occurrences of β. Then the reduction:

$$\Delta_1, \mathsf{adv}\ p = \alpha, \Delta_2, \vec{C}/L \rightarrow \Delta_1, \mathsf{adv}\ p = \alpha, \Delta_2, \vec{C}'/L'$$

holds iff the following reduction holds:

$$\Delta_1, \mathsf{adv}\ p = \beta, \mathsf{adv}\ p = \alpha, \Delta_2, \{|\vec{C}|\}/\{|L|\} \rightarrow \Delta_1, \mathsf{adv}\ p = \beta, \mathsf{adv}\ p = \alpha, \Delta_2, \{|\vec{C}'|\}/\{|L'|\}$$

The interesting case is for lookup of a function f defined at p, i.e., if the first lookup yields W, then the second lookup yields $\{|W|\}$:

$$(\Delta_1, \mathsf{adv}\ p = \beta, \mathsf{adv}\ p = \alpha, \Delta_2, \vec{C})(f) = \{|(\Delta_1, \mathsf{adv}\ p = \alpha, \Delta_2, \vec{C})(f)|\}$$

We also define a relation R between pairs of pairs of terms by:

$$\begin{array}{lll} R((M',N'),(M,N)) & \Leftrightarrow & (M' = M \wedge N' = N) \vee \\ & & (M \text{ and } N \text{ are values} \wedge M' = \beta\texttt{<}M\texttt{>} \wedge N' = \beta\texttt{<}N\texttt{>}) \vee \\ & & (M \text{ and } N \text{ are values} \wedge M' = \beta\texttt{<}M\texttt{>}\ \phi \wedge N' = \beta\texttt{<}N\texttt{>}\ \phi) \end{array}$$

Now define the bisimulation candidate $\mathscr{R}$ by:

$$\Gamma, \beta; \Delta_1, \mathsf{adv}\ p = \beta, \mathsf{adv}\ p = \alpha, \Delta_2 \vdash (\{|\vec{A}|\}/\{|\vec{\mathscr{E}}|\}/M'/\vec{U}')\ \mathscr{R}\ (\{|\vec{B}|\}/\{|\vec{\mathscr{F}}|\}/N'/\vec{V}')$$

whenever:

1. $\Gamma; \Delta_1, \mathsf{adv}\ p = \alpha, \Delta_2 \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$
2. $\Gamma; \Delta_1, \mathsf{adv}\ p = \alpha, \Delta_2 \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U}$ and $\Gamma; \Delta_1, \mathsf{adv}\ p = \alpha, \Delta_2 \vdash \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$ are compatible.
3. $R((M',N'),(\{|M|\},\{|N|\}))$ and $R((U_i',V_i'),(\{|U_i|\},\{|V_i|\}))$, for all $1 \le i \le |\vec{U}| = |\vec{V}| = |\vec{U}'| = |\vec{V}'|$.
4. β does not occur in Δ_1, Δ_2, $\vec{A}/\vec{\mathscr{E}}/M/\vec{U}$, $\vec{B}/\vec{\mathscr{F}}/N/\vec{V}$, and α only occurs in the form $\alpha\texttt{<}U\texttt{>}$.

It can be verified that $\mathscr{R}$ is a bisimulation. We present the three non-trivial cases. For the first case, consider $M' = \mathscr{G}_1'[\alpha\texttt{<}W_1'\texttt{>}\ W_2']$, so the only transition possible is:

$$\begin{array}{l} \Gamma, \beta; \Delta_1, \mathsf{adv}\ p = \beta, \mathsf{adv}\ p = \alpha, \Delta_2 \vdash \{|\vec{A}|\}/\{|\vec{\mathscr{E}}|\}/\mathscr{G}_1'[\alpha\texttt{<}W_1'\texttt{>}\ W_2']/\vec{U}' \xrightarrow{\mathsf{acall}\ \alpha} \\ \Gamma, \beta; \Delta_1, \mathsf{adv}\ p = \beta, \mathsf{adv}\ p = \alpha, \Delta_2 \vdash \{|\vec{A}|\}/\{|\vec{\mathscr{E}}|\}, \mathscr{G}_1'/W_2'/\vec{U}', W_1' \end{array}$$

Since $\mathscr{G}_1'[\alpha\texttt{<}W_1'\texttt{>}\ W_2']$ is not a value, we know that $\mathscr{G}_1'[\alpha\texttt{<}W_1'\texttt{>}\ W_2'] = \{|M|\}$, so M must have form $\mathscr{G}_1[\alpha\texttt{<}W_1\texttt{>}\ W_2]$ and $\mathscr{G}_1' = \{|\mathscr{G}_1|\}$, $W_1' = \beta\texttt{<}\{|W_1|\}\texttt{>}$, and $W_2' = \{|W_2|\}$. The transition above is thus:

$$\begin{array}{l} \Gamma, \beta; \Delta_1, \mathsf{adv}\ p = \beta, \mathsf{adv}\ p = \alpha, \Delta_2 \vdash \{|\vec{A}|\}/\{|\vec{\mathscr{E}}|\}/\{|\mathscr{G}_1|\}[\alpha\texttt{<}\beta\texttt{<}\{|W_1|\}\texttt{>}\ \{|W_2|\}\texttt{>}]/\vec{U}' \xrightarrow{\mathsf{acall}\ \alpha} \\ \Gamma, \beta; \Delta_1, \mathsf{adv}\ p = \beta, \mathsf{adv}\ p = \alpha, \Delta_2 \vdash \{|\vec{A}|\}/\{|\vec{\mathscr{E}}|\}, \{|\mathscr{G}_1|\}/\{|W_2|\}/\vec{U}', \beta\texttt{<}\{|W_1|\}\texttt{>} \end{array}$$

In addition, we have the transition:

$$\begin{array}{l} \Gamma; \Delta_1, \mathsf{adv}\ p = \alpha, \Delta_2 \vdash \vec{A}/\vec{\mathscr{E}}/\mathscr{G}_1[\alpha\texttt{<}W_1\texttt{>}\ W_2]/\vec{U} \xrightarrow{\mathsf{acall}\ \alpha} \\ \Gamma; \Delta_1, \mathsf{adv}\ p = \alpha, \Delta_2 \vdash \vec{A}/\vec{\mathscr{E}}, \mathscr{G}_1/W_2/\vec{U}, W_1 \end{array}$$

By assumption:

$$\Gamma; \Delta_1, \mathsf{adv}\ p = \alpha, \Delta_2 \vdash \vec{A}/\vec{\mathscr{E}}/\mathscr{G}_1[\alpha\texttt{<}W_1\texttt{>}\ W_2]/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$$

Hence, there exist $\mathscr{G}_2$, W_3, and W_4 such that:

$$\begin{array}{l} \Gamma; \Delta_1, \mathsf{adv}\ p = \alpha, \Delta_2 \vdash \vec{B}/\vec{\mathscr{F}}/N/\vec{V} \xrightarrow{\mathsf{acall}\ \alpha} \\ \Gamma; \Delta_1, \mathsf{adv}\ p = \alpha, \Delta_2 \vdash \vec{B}_1/\vec{\mathscr{F}}, \mathscr{G}_2/W_4/\vec{V}, W_3 \end{array}$$

And:

$$\Gamma;\Delta_1,\mathsf{adv}\ p=\alpha,\Delta_2 \vdash \vec{A}/\vec{\mathscr{E}},\mathscr{G}_1/W_2/\vec{U},W_1 \sim \vec{B}_1/\vec{\mathscr{F}},\mathscr{G}_2/W_4/\vec{V},W_3$$

The preservation of reduction by $\{|\,|\}$ yields:

$$\begin{array}{l}\Gamma,\beta;\Delta_1,\mathsf{adv}\ p=\beta,\mathsf{adv}\ p=\alpha,\Delta_2 \vdash \{|\vec{B}|\}/\{|\vec{\mathscr{F}}|\}/\{|N|\}/\vec{V}' \xrightarrow{\mathsf{acall}\ \alpha}\!\!\!\!\!\rightarrow \\ \Gamma,\beta;\Delta_1,\mathsf{adv}\ p=\beta,\mathsf{adv}\ p=\alpha,\Delta_2 \vdash \{|\vec{B}_1|\}/\{|\vec{\mathscr{F}}|\},\{|\mathscr{G}_2|\}/\{|W_4|\}/\vec{V}',\beta\!<\!\{|W_3|\}\!>\end{array}$$

Finally, the fact that R allows introduction of β around top-level values establishes the case:

$$\begin{array}{l}\Gamma,\beta;\Delta_1,\mathsf{adv}\ p=\beta,\mathsf{adv}\ p=\alpha,\Delta_2 \vdash (\{|\vec{A}|\}/\{|\vec{\mathscr{E}}|\},\{|\mathscr{G}_1|\}/\{|W_2|\}/\vec{U}',\beta\!<\!\{|W_1|\}\!>) \\ \qquad\qquad \mathscr{R}(\{|\vec{B}_1|\}/\{|\vec{\mathscr{F}}|\},\{|\mathscr{G}_2|\}/\{|W_4|\}/\vec{V}',\beta\!<\!\{|W_3|\}\!>)\end{array}$$

For the second case, suppose M and N are values and $M' = \beta\!<\!\{|M|\}\!>$, $N' = \beta\!<\!\{|N|\}\!>$. The non-trivial transition is:

$$\begin{array}{l}\Gamma,\beta;\Delta_1,\mathsf{adv}\ p=\beta,\mathsf{adv}\ p=\alpha,\Delta_2 \vdash \{|\vec{A}|\}/\{|\vec{\mathscr{E}}|\}/\beta\!<\!\{|M|\}\!>/\vec{U}' \xrightarrow{\mathsf{app}\ \phi}\!\!\!\!\rightarrow \\ \Gamma,\beta;\Delta_1,\mathsf{adv}\ p=\beta,\mathsf{adv}\ p=\alpha,\Delta_2 \vdash \{|\vec{A}|\}/\{|\vec{\mathscr{E}}|\}/\beta\!<\!\{|M|\}\!>\ \phi/\vec{U}'\end{array}$$

There is a similar transition for $N' = \beta\!<\!\{|N|\}\!>$, and the targets are in the bisimulation $\mathscr{R}$ by definition of R. For the third case, suppose M and N are values and $M' = \beta\!<\!\{|M|\}\!>\ \phi$, $N' = \beta\!<\!\{|N|\}\!>\ \phi$. The only possible transition is:

$$\begin{array}{l}\Gamma,\beta;\Delta_1,\mathsf{adv}\ p=\beta,\mathsf{adv}\ p=\alpha,\Delta_2 \vdash \{|\vec{A}|\}/\{|\vec{\mathscr{E}}|\}/\beta\!<\!\{|M|\}\!>\ \phi/\vec{U}' \xrightarrow{\mathsf{acall}\ \beta}\!\!\!\!\rightarrow \\ \Gamma,\beta;\Delta_1,\mathsf{adv}\ p=\beta,\mathsf{adv}\ p=\alpha,\Delta_2 \vdash \{|\vec{A}|\}/\{|\vec{\mathscr{E}}|\},[-]/\phi/\vec{U}',\{|M|\}\end{array}$$

There is a similar transition for $N' = \beta\!<\!\{|N|\}\!>\ \phi$. By assumption:

$$\Gamma;\Delta_1,\mathsf{adv}\ p=\alpha,\Delta_2 \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$$

Using the definition of bisimulation in conjunction with Corollary 51, we can deduce:

$$\Gamma;\Delta_1,\mathsf{adv}\ p=\alpha,\Delta_2 \vdash \vec{A}/\vec{\mathscr{E}},[-]/\phi/\vec{U},M \sim \vec{B}/\vec{\mathscr{F}},[-]/\phi/\vec{V},N$$

It follows that:

$$\begin{array}{l}\Gamma;\Delta_1,\mathsf{adv}\ p=\beta,\mathsf{adv}\ p=\alpha,\Delta_2 \vdash (\{|\vec{A}|\}/\{|\vec{\mathscr{E}}|\},[-]/\phi/\vec{U}',\{|M|\}) \\ \qquad\qquad \mathscr{R}(\{|\vec{B}|\}/\{|\vec{\mathscr{F}}|\},[-]/\phi/\vec{V}',\{|N|\})\end{array}$$

Hence $\mathscr{R}$ is a bisimulation. The result follows immediately from the original hypothesis preventing occurrences of α, and that $\{|\,|\}$ is the identity on such terms, values, contexts, and declarations. □

Proof of Lemma 35. For the proof of Lemma 35 in the rest of this section, a simple bisimulation proof allows us to assume without loss of generality that in $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/()/\vec{W}$, that is to be added to $\Gamma;\Delta \vdash \cdot/\cdot/M/\vec{U} \sim \cdot/\cdot/N/\vec{V}$, the declarations $\vec{A}$ have the form $\vec{A}_1,\vec{A}_2,\vec{A}_3$ where ($\vec{q}$ and $\vec{r}$ may be bound in $\vec{A}_1$ or Δ):

$$\vec{A}_1 = \mathsf{pcd}\ \vec{p}$$

$$\vec{A}_2 = \mathsf{fun}\ \vec{f}@\vec{q} = \vec{W}'$$
$$\vec{A}_3 = \mathsf{adv}\ \vec{r} = \vec{W}''$$

Repeated use of Lemma 50 and a simple bisimulation to move primitive pointcuts to the left of the common declaration list, allows $\vec{A}_1$ to be added along with initial symbolic advice $\vec{A}_4 = (\mathsf{adv}\ \vec{p} = \vec{\alpha})$ (where $\vec{\alpha}$ are fresh):

$$\Gamma, \vec{\alpha}; \Delta, \vec{A}_1, \vec{A}_4 \vdash \cdot/\cdot/M/\vec{U} \sim \cdot/\cdot/N/\vec{V}$$

Note that this pair of configurations is also compatible. The next steps of the proof introduce symbolic function and advice bodies first, and then substitute their actual bodies from $\vec{A}_2$ and $\vec{A}_3$. First, function definitions with fresh symbolic bodies can be added using $\mathsf{fun}\ f@p = \phi$ transitions since the primitive pointcuts $\vec{p}$ are public, so with $\vec{A}_5 = (\mathsf{fun}\ \vec{f}@\vec{q} = \vec{\phi})$:

$$\Gamma, \vec{\alpha}, \vec{\phi}; \Delta, \vec{A}_1, \vec{A}_4, \vec{A}_5 \vdash \cdot/\cdot/M/\vec{U}, \vec{f} \sim \cdot/\cdot/N/\vec{V}, \vec{f}$$

Again the configurations are compatible. Next, we know there is no symbolic advice in M, $\vec{U}$, N, $\vec{V}$, $\vec{f}$, so for each $\mathsf{adv}\ r = W'' \in \vec{A}_3$ we successively apply Lemma 52 to the rightmost (necessarily symbolic) advice declaration for r in either Δ (if $\mathsf{pcd}\ r \in \Delta$) or $\vec{A}_4$ (if $\mathsf{pcd}\ r \in \vec{A}_1$). With variable renaming and declaration reordering, this yields fresh $\vec{\beta}$ and symbolic advice declarations $\vec{A}_6 = (\mathsf{adv}\ \vec{r} = \vec{\beta})$ such that:

$$\Gamma, \vec{\alpha}, \vec{\phi}, \vec{\beta}; \Delta, \vec{A}_1, \vec{A}_4, \vec{A}_5, \vec{A}_6 \vdash \cdot/\cdot/M/\vec{U}, \vec{f} \sim \cdot/\cdot/N/\vec{V}, \vec{f}$$

Since the free names of $\vec{\mathscr{E}}$, $\vec{W}$, $\vec{W}'$, $\vec{W}''$ are bound in the context, we can use Corollary 51 to add values and evaluation contexts, yielding:

$$\Gamma, \vec{\alpha}, \vec{\phi}, \vec{\beta}; \Delta, \vec{A}_1, \vec{A}_4, \vec{A}_5, \vec{A}_6 \vdash \cdot/\vec{\mathscr{E}}/M/\vec{U}, \vec{f}, \vec{W}, \vec{W}', \vec{W}'' \sim \cdot/\vec{\mathscr{E}}/N/\vec{V}, \vec{f}, \vec{W}, \vec{W}', \vec{W}''$$

The symbolic function declarations $\vec{A}_5$ and advice declarations $\vec{A}_6$ can be moved to the private declaration lists, simultaneously removing the $\vec{f}$ value lists, by a straightforward bisimulation proof to give the compatible, bisimilar configurations:

$$\Gamma, \vec{\alpha}, \vec{\phi}, \vec{\beta}; \Delta, \vec{A}_1, \vec{A}_4 \vdash \vec{A}_5, \vec{A}_6/\vec{\mathscr{E}}/M/\vec{U}, \vec{W}, \vec{W}', \vec{W}'' \sim \vec{A}_5, \vec{A}_6/\vec{\mathscr{E}}/N/\vec{V}, \vec{W}, \vec{W}', \vec{W}''$$

Then the real function and advice bodies $\vec{W}'$ and $\vec{W}''$ can be substituted for $\vec{\phi}$ and $\vec{\beta}$ respectively by Theorem 49 to recover $\vec{A}_2$ and $\vec{A}_3$:

$$\Gamma, \vec{\alpha}; \Delta, \vec{A}_1, \vec{A}_4 \vdash \vec{A}_2, \vec{A}_3/\vec{\mathscr{E}}/M/\vec{U}, \vec{W} \sim \vec{A}_2, \vec{A}_3/\vec{\mathscr{E}}/N/\vec{V}, \vec{W}$$

Now $\vec{A}_1$, $\vec{A}_4$ can be moved to the private declaration lists by a bisimulation proof:

$$\Gamma, \vec{\alpha}; \Delta \vdash \vec{A}_1, \vec{A}_4, \vec{A}_2, \vec{A}_3/\vec{\mathscr{E}}/M/\vec{U}, \vec{W} \sim \vec{A}_1, \vec{A}_4, \vec{A}_2, \vec{A}_3/\vec{\mathscr{E}}/N/\vec{V}, \vec{W}$$

Using Corollary 51 to add $|\vec{A}_4|$ copies of $\lambda z.z$ to both value lists, and Theorem 49 to substitute those advice bodies for $\vec{\alpha}$, we have:

$$\Gamma; \Delta \vdash \vec{A}_1, \mathsf{adv}\ \vec{p} = \lambda z.z, \vec{A}_2, \vec{A}_3/\vec{\mathscr{E}}/M/\vec{U}, \vec{W} \sim \vec{A}_1, \mathsf{adv}\ \vec{p} = \lambda z.z, \vec{A}_2, \vec{A}_3/\vec{\mathscr{E}}/N/\vec{V}, \vec{W}$$

However, it can be shown that the presence of advice declarations of the form $\mathsf{adv}\ p_i = \lambda z.z$ does not alter the result of advice lookup, so a bisimulation proof eliminates them to give:

$$\Gamma; \Delta \vdash \vec{A}_1, \vec{A}_2, \vec{A}_3/\vec{\mathscr{E}}/M/\vec{U}, \vec{W} \sim \vec{A}_1, \vec{A}_2, \vec{A}_3/\vec{\mathscr{E}}/N/\vec{V}, \vec{W}$$

This completes the proof of Lemma 35.

D Bisimulation is a congruence

This section contains the proof of Theorem 36 and shows that bisimulation is a congruence.

Application. Let $U_1 \sim U_1'$ and $U_2 \sim U_2'$. We need to show that $U_1\ U_2 \sim U_1'\ U_2'$. From:

$$\Gamma;\Delta \vdash \cdot/\cdot/U_2/\vec{g} \sim \cdot/\cdot/U_2'/\vec{g}$$

where $\vec{g}$ consists of the function names declared in Δ, and $fn(U_1\ U_2) \cup fn(U_1'\ U_2') \subseteq \Gamma \cup dn(\Delta)$, we deduce from put transitions that:

$$\Gamma;\Delta \vdash \cdot/\cdot/U_2/\vec{g},U_2 \sim \cdot/\cdot/U_2'/\vec{g},U_2'$$

From this, we deduce:

$$\Gamma,x_1,x_2;\Delta \vdash \cdot/\cdot/U_2/\vec{g},U_2 \sim \cdot/\cdot/U_2'/\vec{g},U_2'$$

and using Lemma 34:

$$\Gamma,x_1,x_2;\Delta \vdash \cdot/\cdot/x_1\ x_2/\vec{g},U_2 \sim \cdot/\cdot/x_1\ x_2/\vec{g},U_2'$$

Now, using Corollary 51 with a simple bisimulation proof to reorder value lists yields:

$$\Gamma,x_1,x_2;\Delta \vdash \cdot/\cdot/x_1\ x_2/\vec{g},U_1',U_2 \sim \cdot/\cdot/x_1\ x_2/\vec{g},U_1',U_2'$$

Similarly, from $\Gamma;\Delta \vdash \cdot/\cdot/U_1/ \sim \cdot/\cdot/U_1'/\cdot$ we get:

$$\Gamma,x_1,x_2;\Delta \vdash \cdot/\cdot/x_1\ x_2/\vec{g},U_1,U_2 \sim \cdot/\cdot/x_1\ x_2/\vec{g},U_1',U_2$$

Combining with transitivity:

$$\Gamma,x_1,x_2;\Delta \vdash \cdot/\cdot/x_1\ x_2/\vec{g},U_1,U_2 \sim \cdot/\cdot/x_1\ x_2/\vec{g},U_1',U_2'$$

Consider the substitution σ with stack $((),0)$ and partial function given by $\{x_i \mapsto |\vec{g}| + i\}$. Theorem 49 yields:

$$\Gamma;\Delta \vdash [\cdot/\cdot/x_1\ x_2/\vec{g},U_1,U_2]_\sigma \sim [\cdot/\cdot/x_1\ x_2/\vec{g},U_1',U_2']_\sigma$$

and finishes the proof.

Function declaration. Let $U \sim U'$ and $M \sim M'$, where f occurs in neither U nor U'. We need to show that $\mathsf{fun}\ f\,@\,p = U\ ;\ M \sim \mathsf{fun}\ f\,@\,p = U'\ ;\ M'$.

We start with compatible LTS configurations:

$$\Gamma,f;\Delta \vdash \cdot/\cdot/U/\vec{g} \sim \cdot/\cdot/U'/\vec{g} \tag{1}$$

$$\Gamma,f;\Delta \vdash \cdot/\cdot/M/\vec{g} \sim \cdot/\cdot/M'/\vec{g} \tag{2}$$

Using put transitions with Equation 1, we derive:

$$\Gamma,f;\Delta \vdash \cdot/\cdot/U/\vec{g},U \sim \cdot/\cdot/U'/\vec{g},U'$$

Apply Lemma 34 with term M to get:

$$\Gamma,f;\Delta \vdash \cdot/\cdot/M/\vec{g},U \sim \cdot/\cdot/M/\vec{g},U'$$

Next, adding the value U' to both sides of Equation 2 using Corollary 51 yields:

$$\Gamma,f;\Delta \vdash \cdot/\cdot/M/\vec{g},U' \sim \cdot/\cdot/M'/\vec{g},U'$$

By transitivity, we have:

$$\Gamma,f;\Delta \vdash \cdot/\cdot/M/\vec{g},U \sim \cdot/\cdot/M'/\vec{g},U'$$

We can add a fresh variable ϕ:

$$\Gamma,f,\phi;\Delta \vdash \cdot/\cdot/M/\vec{g},U \sim \cdot/\cdot/M'/\vec{g},U'$$

Moreover, M, M', U, U' contain no symbolic advice by hypothesis, so Lemma 35 can be used to add the function declaration $\mathsf{fun}\ f'@p = \phi$ (where f' is fresh) and the value f' to both sides:

$$\Gamma,f,\phi;\Delta \vdash \mathsf{fun}\ f'@p=\phi/\cdot/M/\vec{g},U,f' \sim \mathsf{fun}\ f'@p=\phi/\cdot/M'/\vec{g},U',f'$$

Using Theorem 49 with substitution σ with stack $((),0)$ and partial function given by $\{\phi \mapsto |\vec{g}|+1, f \mapsto |\vec{g}|+2\}$ (using the fact that there are no occurrences of f in U, U') gives:

$$\Gamma;\Delta \vdash \mathsf{fun}\ f'@p=U/\cdot/(M[f:=f'])/\vec{g} \sim \mathsf{fun}\ f'@p=U'/\cdot/(M'[f:=f'])/\vec{g}$$

Since f' was fresh, there are no occurrences of f' in M or M'. Therefore, renaming the bound variable f' to f yields:

$$\Gamma;\Delta \vdash \mathsf{fun}\ f@p=U/\cdot/M/\vec{g} \sim \mathsf{fun}\ f@p=U'/\cdot/M'/\vec{g}$$

Now the function declarations can be moved into the term positions with a simple bisimulation:

$$\Gamma;\Delta \vdash \cdot/\cdot/\mathsf{fun}\ f@p=U\,;M/\vec{g} \sim \cdot/\cdot/\mathsf{fun}\ f@p=U'\,;M'/\vec{g}$$

Thus we have:

$$\mathsf{fun}\ f@p=U\,;M \sim \mathsf{fun}\ f@p=U'\,;M'$$

Advice declaration. Let $U \sim U'$ and $M \sim M'$. We need to show that $\mathsf{adv}\ p=U\,;M \sim \mathsf{adv}\ p=U'\,;M'$.

We start with compatible LTS configurations:

$$\Gamma;\Delta \vdash \cdot/\cdot/U/\vec{g} \sim \cdot/\cdot/U'/\vec{g}$$

$$\Gamma;\Delta \vdash \cdot/\cdot/M/\vec{g} \sim \cdot/\cdot/M'/\vec{g}$$

where Δ contains the declaration $\mathsf{pcd}\ p$. As in the function declaration case, we derive:

$$\Gamma;\Delta \vdash \cdot/\cdot/M/\vec{g},U \sim \cdot/\cdot/M'/\vec{g},U'$$

Moreover, M, M', U, U' contain no symbolic advice by hypothesis, so a fresh α can be added to the context and Lemma 35 used to add the symbolic advice declaration $\mathsf{adv}\ p = \alpha$ to both sides:

$$\Gamma,\alpha;\Delta \vdash \mathsf{adv}\ p=\alpha/\cdot/M/\vec{g},U \sim \mathsf{adv}\ p=\alpha/\cdot/M'/\vec{g},U'$$

A simple bisimulation shows that the advice declarations can be moved into the terms:

$$\Gamma,\alpha;\Delta \vdash \cdot/\cdot/\mathsf{adv}\ p=\alpha\,;M/\vec{g},U \sim \cdot/\cdot/\mathsf{adv}\ p=\alpha\,;M'/\vec{g},U'$$

Finally, Theorem 49 with substitution σ with stack $((),0)$ and partial function given by $\{\alpha \mapsto |\vec{g}|+1\}$ gives:

$$\Gamma;\Delta \vdash \cdot/\cdot/\mathsf{adv}\ p=U\,;M/\vec{g} \sim \cdot/\cdot/\mathsf{adv}\ p=U'\,;M'/\vec{g}$$

Hence:

$$\mathsf{adv}\ p=U\,;M \sim \mathsf{adv}\ p=U'\,;M'$$

Lambda abstraction. Given $L \sim L'$, we wish to establish $\lambda x.L \sim \lambda x.L'$. The proof that the latter terms are bisimilar proceeds by a direct bisimulation argument.

Define a relation R between pairs of pairs of terms by:

$$\begin{array}{lll} R((M',N'),(M,N)) & \Leftrightarrow & (M'=M \wedge N'=N) \vee \\ & & (M \text{ and } N \text{ are values} \wedge M'=\lambda x.L \wedge N'=\lambda x.L') \vee \end{array}$$

And define the bisimulation candidate $\mathscr{R}$ by:

$$\Gamma;\Delta \vdash (\vec{A}/\vec{\mathscr{E}}/M'/\vec{U}')\ \mathscr{R}\ (\vec{B}/\vec{\mathscr{F}}/N'/\vec{V}')$$

whenever:

1. $(fn(L) \cup fn(L')) \setminus \{x\} \subseteq \Gamma \cup dn(\Delta)$.
2. $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$
3. $\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/M/\vec{U}$ and $\Gamma;\Delta \vdash \vec{B}/\vec{\mathscr{F}}/N/\vec{V}$ are compatible.
4. $R((M',N'),(M,N))$ and $R((U_i',V_i'),(U_i,V_i))$, for all $1 \le i \le |\vec{U}| = |\vec{V}| = |\vec{U}'| = |\vec{V}'|$.

It can be verified that $\mathscr{R}$ is a bisimulation. The key case is for $\mathsf{app}\ \phi$ transitions from:

$$\Gamma;\Delta \vdash (\vec{A}/\vec{\mathscr{E}}/\lambda x.L/\vec{U}')\ \mathscr{R}\ (\vec{B}/\vec{\mathscr{F}}/\lambda x.L'/\vec{V}')$$

where we know that there exist values W, W' with compatible, bisimilar configurations:

$$\Gamma;\Delta \vdash \vec{A}/\vec{\mathscr{E}}/W/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/W'/\vec{V}$$

such that $R((U_i', V_i'), (U_i, V_i))$, for all $1 \leq i \leq |\vec{U}| = |\vec{V}| = |\vec{U}'| = |\vec{V}'|$. We have to show that:

$$\Gamma; \Delta \vdash (\vec{A}/\vec{\mathscr{E}}/(\lambda x.L)\ \phi/\vec{U}')\ \mathscr{R}\ (\vec{B}/\vec{\mathscr{F}}/(\lambda x.L')\ \phi/\vec{V}')$$

Applying Lemma 34 to $\Gamma; \Delta \vdash \vec{A}/\vec{\mathscr{E}}/W/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/W'/\vec{V}$ and $L[x := \phi]$, we get:

$$\Gamma; \Delta \vdash \vec{A}/\vec{\mathscr{E}}/L[x := \phi]/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/L[x := \phi]/\vec{V} \tag{3}$$

In addition, by hypothesis and renaming variables, we have: $L[x := \phi] \sim L'[x := \phi]$. With Γ and Δ as above, the definition of $\sim$ implies, for function names $\vec{g}$ defined in Δ:

$$\Gamma; \Delta \vdash \cdot/\cdot/L[x := \phi]/\vec{g} \sim \cdot/\cdot/L'[x := \phi]/\vec{g}$$

Applying Lemma 35, to these compatible, bisimilar configurations and the well-formed LTS configuration $\Gamma; \Delta \vdash \vec{B}/\vec{\mathscr{F}}/()/\vec{V}$, together with a simple bisimulation to remove duplicated copies of $\vec{g}$, we have:

$$\Gamma; \Delta \vdash \vec{B}/\vec{\mathscr{F}}/L[x := \phi]/\vec{V} \sim \vec{B}/\vec{\mathscr{F}}/L'[x := \phi]/\vec{V} \tag{4}$$

Combining Equation 3 and Equation 4 by transitivity yields:

$$\Gamma; \Delta \vdash \vec{A}/\vec{\mathscr{E}}/L[x := \phi]/\vec{U} \sim \vec{B}/\vec{\mathscr{F}}/L'[x := \phi]/\vec{V}$$

Hence:

$$\Gamma; \Delta \vdash (\vec{A}/\vec{\mathscr{E}}/L[x := \phi]/\vec{U}')\ \mathscr{R}\ (\vec{B}/\vec{\mathscr{F}}/L'[x := \phi]/\vec{V}')$$

Finally, we have the reductions $\vec{A}/(\lambda x.L)\ \phi \to \vec{A}/L[x := \phi]$ and $\vec{A}/(\lambda x.L')\ \phi \to \vec{A}/L'[x := \phi]$, and reduction is deterministic, so:

$$\Gamma; \Delta \vdash (\vec{A}/\vec{\mathscr{E}}/(\lambda x.L)\ \phi/\vec{U}')\ \mathscr{R}\ (\vec{B}/\vec{\mathscr{F}}/(\lambda x.L')\ \phi/\vec{V}')$$

Thus $\mathscr{R}$ is a bisimulation. It follows immediately that $\lambda x.L \sim \lambda x.L'$.

Let. Given $L_1 \sim L_1'$ and $L_2 \sim L_2'$ we must show that $\mathsf{let}\ x = L_1\ ;\ L_2 \sim \mathsf{let}\ x = L_1'\ ;\ L_2'$. We do this in two stages and then use the transitivity of $\sim$:

$$\mathsf{let}\ x = L_1\ ;\ L_2 \sim \mathsf{let}\ x = L_1'\ ;\ L_2$$
$$\mathsf{let}\ x = L_1'\ ;\ L_2 \sim \mathsf{let}\ x = L_1'\ ;\ L_2'$$

For the first stage, we use the bisimulation constructed in the Let case of the proof of Lemma 34 to deduce $\mathsf{let}\ x = L_1\ ;\ L_2 \sim \mathsf{let}\ x = L_1'\ ;\ L_2$ from $L_1 \sim L_1'$.

For the second stage, a bisimulation proof establishes:

$$\mathsf{let}\ x = L_1'\ ;\ L_2 \sim \mathsf{let}\ x = L_1'\ ;\ (\lambda x.L_2)\ x$$
$$\mathsf{let}\ x = L_1'\ ;\ L_2' \sim \mathsf{let}\ x = L_1'\ ;\ (\lambda x.L_2')\ x$$

Now we know from the previous case that $L_2 \sim L_2'$ implies $\lambda x.L_2 \sim \lambda x.L_2'$. After saving those values into the value lists, use Lemma 34 to add $\mathsf{let}\ x = L_1'\ ;\ y\ x$ in the term position in both configurations. Substitution (Theorem 49) establishes the relationship between configurations with $(\mathsf{let}\ x = L_1'\ ;\ y\ x)[y := \lambda x.L_2]$ and $(\mathsf{let}\ x = L_1'\ ;\ y\ x)[y := \lambda x.L_2']$ in the term positions.

Primitive pointcuts declaration. Given $L \sim L'$ we must show that $\mathsf{pcd}\ p\,;L \sim \mathsf{pcd}\ p\,;L'$. From $L \sim L'$ we know:

$$\Gamma,\alpha;\Delta,\mathsf{pcd}\ p,\mathsf{adv}\ p=\alpha \vdash \cdot/\cdot/L/\cdot \sim \cdot/\cdot/L'/\cdot$$

A straightforward bisimulation shows that the declarations $\mathsf{pcd}\ p$ and $\mathsf{adv}\ p=\alpha$ can be moved to the front of the terms L and L'. Then the advice $\mathsf{adv}\ p=\alpha$ can be eliminated by substitution of $\lambda z.\lambda x.z\,x$ for α and a bisimulation proof, to leave:

$$\Gamma;\Delta \vdash \cdot/\cdot/\mathsf{pcd}\ p\,;L/\cdot \sim \cdot/\cdot/\mathsf{pcd}\ p\,;L'/\cdot$$

E Completeness

We show completeness ($M \equiv N$ implies $M \sim N$) by demonstrating the contrapositive ($M \not\sim N$ implies $M \not\equiv N$). The proof is a definability argument: we show that for every distinguishing trace we can construct a context that witnesses the trace. This construction proceeds via an analysis of normal forms for such traces.

We first define *normal* traces and demonstrate that they are sufficient to distinguish non-bisimilar terms. (Because the language is deterministic, bisimilarity coincides with trace equivalence, which simplifies the argument.)

Let s,t range over *traces* of visible labels $s,t ::= \kappa_1,\ldots,\kappa_n$, with empty trace ε.

Definition 53. A *complete normal trace* is a trace that is generable by the following grammar over labels:

$$\begin{array}{rcl}
\textsc{start} & ::= & \textsc{term}^*,\mathsf{put},\textsc{ctxt}^* \\
\textsc{term} & ::= & \mathsf{fcall}\ \phi,\mathsf{put},\textsc{ctxt}^*,\mathsf{ret}\ \psi \mid \mathsf{acall}\ \alpha,\mathsf{put},\textsc{ctxt}^*,\mathsf{ret}\ \psi \\
\textsc{ctxt} & ::= & \mathsf{get}\ i,\mathsf{app}\ \phi,\textsc{term}^*,\mathsf{put} \mid \mathsf{fun}\ f\,@p=\phi \mid \mathsf{adv}\ p=\alpha
\end{array}$$

A *normal trace* is a prefix of a complete normal trace. □

Proposition 54. *If* $\Gamma;\Delta \vdash \mathbf{M} \lesssim \mathbf{N}$ *and* $\Gamma;\Delta \vdash \mathbf{M} \xrightarrow{s}\!\!\!\!\rightarrow$ *then* $\Gamma;\Delta \vdash \mathbf{N} \xrightarrow{s}\!\!\!\!\rightarrow$.

PROOF. By induction on the length of the trace. □

Proposition 55. *If* $\Gamma;\Delta \vdash \mathbf{M} \not\lesssim \mathbf{N}$ *then for some normal trace* s *and label* $\kappa \in \{\mathsf{fcall},\mathsf{acall}\}$*:* $\Gamma;\Delta \vdash \mathbf{M} \xrightarrow{s}\!\!\!\!\rightarrow \xrightarrow{\kappa}$ *and* $\Gamma;\Delta \vdash \mathbf{N} \xrightarrow{s}\!\!\!\!\rightarrow \not\xrightarrow{\kappa}$.

PROOF. A rather tedious analysis of the commutativity of various labels and the resulting configurations. The essential observation is that the following categories of LTS states are disjoint:

- Write $\Gamma;\Delta \vdash \mathbf{M} \Uparrow$ if $\Gamma;\Delta \vdash \mathbf{M} \rightarrow^{\omega}$.
- Write $\Gamma;\Delta \vdash \mathbf{M} \Downarrow \textsc{term}$ if $\Gamma;\Delta \vdash \mathbf{M} \xrightarrow{\kappa}$ for $\kappa \in \{\mathsf{fcall},\mathsf{acall}\}$.
- Write $\Gamma;\Delta \vdash \mathbf{M} \Downarrow \textsc{ctxt}$ if $\Gamma;\Delta \vdash \mathbf{M} \xrightarrow{\kappa}$ for $\kappa \in \{\mathsf{put},\mathsf{get},\mathsf{ret},\mathsf{app},\mathsf{fun},\mathsf{adv}\}$. □

Completeness indicates that bisimilarity is not too fine a relation. If two terms are not bisimilar, completeness requires that there be some context that distinguishes them. Following Definition 5, the context must *signal* in one case, but not the other (by calling the distinguished function signal). Recall, that we write $M\!\downarrow$ if $M \twoheadrightarrow \mathscr{E}[\mathsf{signal}\ U]$ for some evaluation context $\mathscr{E}$ and value U.

Given a distinguishing trace t, we show how to create a context $\mathbb{C}_t[\text{–}]$, such that $\mathbb{C}_t[\Gamma;\Delta \vdash \mathbf{M}]$ will signal exactly when $\Gamma;\Delta \vdash \mathbf{M}$ can perform trace t. The inductive argument requires that we define contexts of the form $\mathbb{C}^s_t[\text{–}]$, where actions s are completed, and t have yet to be performed. We define the contexts to have the following properties.

Proposition 56. *Let s, κ, t be a normal trace. If $\Gamma;\Delta \vdash \mathbf{M} \xrightarrow{\kappa\!\!\!\!\!\!\rightarrow} \Gamma';\Delta' \vdash \mathbf{M}'$ then $\mathbb{C}^s_{\kappa,t}[\Gamma;\Delta \vdash \mathbf{M}] \twoheadrightarrow \mathbb{C}^{s,\kappa}_t[\Gamma';\Delta' \vdash \mathbf{M}']$.*

Proposition 57. *Let s, κ be a normal trace, where $\kappa \in \{\mathsf{fcall}, \mathsf{acall}\}$, and let $\Gamma;\Delta \vdash \mathbf{M}$ be an LTS state in which* signal *does not occur. (a) If $\Gamma;\Delta \vdash \mathbf{M} \xrightarrow{\kappa\!\!\!\!\!\!\rightarrow}$ then $\mathbb{C}^s_\kappa[\Gamma;\Delta \vdash \mathbf{M}]\!\downarrow$. (b) If $\Gamma;\Delta \vdash \mathbf{M} \xrightarrow{\kappa\!\!\!\!\!\!\rightarrow}$ then $\neg(\mathbb{C}^s_\kappa[\Gamma;\Delta \vdash \mathbf{M}]\!\downarrow)$.*

Starting from Definition 20, completeness follows by induction on the length of trace s from Proposition 55, using Propositions 56 and 57.

In rest of this appendix we describe the strategy for building contexts to satisfy the requirements of Propositions 56 and 57 (up to a structural equivalence that allows reordering of unrelated declarations). Since we are concerned only with normal traces, we adopt the following abbreviations,

$$\begin{aligned} \mathsf{getapp}\ i\ \phi &\triangleq \mathsf{get}\ i, \mathsf{app}\ \phi \\ \mathsf{fcallput}\ \phi &\triangleq \mathsf{fcall}\ \phi, \mathsf{put} \\ \mathsf{acallput}\ \alpha &\triangleq \mathsf{acall}\ \alpha, \mathsf{put} \end{aligned}$$

with completed normal traces formed by the following grammar.

$$\begin{aligned} \text{START} &::= \text{TERM}^*, \mathsf{put}, \text{CTXT}^* \\ \text{TERM} &::= \mathsf{fcallput}\ \phi, \text{CTXT}^*, \mathsf{ret}\ \psi \mid \mathsf{acallput}\ \alpha, \text{CTXT}^*, \mathsf{ret}\ \psi \\ \text{CTXT} &::= \mathsf{getapp}\ i\ \phi, \text{TERM}^*, \mathsf{put} \mid \mathsf{fun}\ f @ p = \phi \mid \mathsf{adv}\ p = \alpha \end{aligned}$$

We divide labels into three groups: *return labels* (ret), *call labels* (fcallput and acallput), and *context labels* (getapp, put, fun and adv). A call label is *unreturned* in a normal trace s if the matching ret is not included in s (due to truncation); similarly, a getapp label is *unreturned* if the matching put is missing. A return label is *uncalled* in a suffix of a normal trace if the matching call label is not included.

Recalling Definition 38, $Z(\vec{\mathscr{E}})[M]$ is the term created by iteratively expanding the context stack $\vec{\mathscr{E}}$ and filling the hole of the resulting context with M.

Fix s, t, Γ, Δ and $\mathbf{M} = \vec{A}/\vec{\mathscr{E}}/M/\vec{U}$. We show how to build term

$$\mathbb{C}^s_t[\Gamma;\Delta \vdash \mathbf{M}] = \mathscr{C}[Z(\vec{\mathscr{H}})[M]].$$

We refer to as $\mathscr{C}$ as *the ambient context*, which is defined below. We refer to the list of contexts $\vec{\mathscr{H}}$ as *the stack*. The stack $\vec{\mathscr{H}}$ alternates contexts from the *term stack* $\vec{\mathscr{E}}$, which

is given by the state of the LTS, with those from a *context stack* $\vec{\mathscr{F}}$, which is defined below.

We need only consider the case where $|\vec{\mathscr{E}}|$ is greater than the number of unreturned call labels in s which have returns in t; otherwise the final return in t could not occur (by the definition of the LTS). By construction, $|\vec{\mathscr{F}}|$ is the number of unreturned calls in s.

We now define the ambient context $\mathscr{C}$. The ambient context includes function declarations for each name in $\Gamma \cup dn(s,t)$, as well as the declarations Δ and $\vec{A}$. The ambient context also includes the following mutable structures.

- The vector values keeps track of the stored values in a configuration. $put(\mathsf{values}, V)$ pushes V onto the end of the vector; $get(\mathsf{values}, i)$ returns the i^{th} value from the vector; these functions have standard encodings in the lambda calculus with references. In $\mathbb{C}^{\_}_{\_}[\_/\_/\_/U_1,\ldots,U_n]$, $get(\mathsf{values}, i)$ returns U_i.
- The reference callcount holds the number of call labels occurring in t; thus !callcount is 0 in $\mathbb{C}^{s}_{\varepsilon}[\_]$.

The functions for $\Gamma \cup dn(s,t)$ are unadvisable, i.e., declared at a fresh primitive pointcut. (Symbolic advice and functions are treated similarly; to simplify the presentation, we abuse notation to allow function declarations at symbolic advice names.) The basic structure of a function body is a case structure on callcount.

$$\begin{aligned}\mathsf{fun}\ \phi = \lambda \mathsf{x}.\mathsf{callcount}\text{--}\,; put(\mathsf{values},\mathsf{x})\,;\\ \mathsf{case}\ !\mathsf{callcount}\ \mathsf{of}\ \cdots\ \mathsf{default} \Rightarrow \Omega\\ \mathsf{fun}\ \alpha = \lambda \mathsf{z}.\lambda \mathsf{x}.\mathsf{callcount}\text{--}\,; put(\mathsf{values},\mathsf{z})\,; put(\mathsf{values},\mathsf{x})\,;\\ \mathsf{case}\ !\mathsf{callcount}\ \mathsf{of}\ \cdots\ \mathsf{default} \Rightarrow \Omega\end{aligned}$$

We generate additional cases for these function bodies by working through the trace s,t. The context stack $\vec{\mathscr{F}}$, mentioned above, contains "suffixes" of these function bodies that have been called (in s) but have not yet returned (reading s forward); the context stack includes the actions yet to be performed by these functions (generated by analyzing t). The term stack $\vec{\mathscr{E}}$ includes the suffixes of functions interrupted by a call label; the context stack $\vec{\mathscr{F}}$ includes the suffixes of functions interrupted by a getapp. Call labels that do not have matching returns in s,t will end in Ω, both in the function declaration and in the context stack.

The last element of the trace is treated specially, as initialization. From Proposition 57, we can assume that the last element is a call label. Suppose it is fcallput ϕ (acallput is similar). Then we add case "$0 \Rightarrow$ signal ()" to the definition of ϕ, and the definition becomes

$$\begin{aligned}\mathsf{fun}\ \phi = \lambda x.\mathsf{callcount}\text{--}\,; put(\mathsf{values},x)\,;\\ \mathsf{case}\ !\mathsf{callcount}\ \mathsf{of}\ \cdots\ 0 \Rightarrow \mathsf{signal}\ ()\,;\ \mathsf{default} \Rightarrow \Omega.\end{aligned}$$

Now that we have initialized the function declaration, we can begin to generate new cases by working *backwards* through s,t. We generate these using a stack of contexts, called the *generating stack*. When we reach the beginning of t (before getting to the end of s), we record the generating stack, which becomes the context stack $\vec{\mathscr{F}}$. We continue the backwards processing of s to generate the remaining function body cases;

this continued processing is performed only to guarantee that function bodies do not change from $\mathbb{C}^{s}_{\kappa,t}[-]$ to $\mathbb{C}^{s,\kappa}_{t}[-]$.

Initially the generating stack contains a context "$[-]\,;\,\Omega$" for every unreturned call label in s,t. Labels are processed as follows:

Return Labels: We push a new context "$[-]\,;\,\psi$" onto the generating stack for every label $\mathsf{ret}\ \psi$ that we process.

Call Labels: We pop a context $\mathscr{G}$ and add a case "$\Rightarrow \mathscr{G}[\,()\,]$" to ϕ for every label $\mathsf{fcallput}\ \phi$, and similarly for $\mathsf{acallput}\ \alpha$. The guard on the case is derived by counting the number of call labels that have been processed.

Context Labels: As we process context labels, we replace the top context of the generating stack $\mathscr{G}$ with a new one, as dictated by the following table. (We use the name y for the variable holding the return value from all calls to term functions; using a single variable name simplifies code generation.)

$\mathsf{getapp}\ i\ \phi$	$\mathsf{let\ y} = (\mathit{get}(\mathsf{values}, i))\ \phi\,;\mathscr{G}$
put	$\mathit{put}(\mathsf{values}, \mathsf{y})\,;\mathscr{G}$
$\mathsf{fun}\ f\,@\,p = \phi$	$\mathsf{fun}\ f\,@\,p = \phi\,;\mathscr{G}$
$\mathsf{adv}\ p = \alpha$	$\mathsf{adv}\ p = \alpha\,;\mathscr{G}$

This strategy generates contexts that satisfy the requirements. We elide further details.