Environmental Bisimulations for Delimited-Control Operators with Dynamic Prompt Generation

Andrés Aristizábal, Dariusz Biernacki, Sergueï Lenglet, Piotr Polesiuk

Environmental Bisimulations for Delimited-Control Operators with Dynamic Prompt Generation*

Andrés Aristizábal[1], Dariusz Biernacki[2], Sergueï Lenglet[3], and Piotr Polesiuk[4]

1 **Pontificia Universidad Javeriana Cali, Cali, Columbia**
aaaristizabal@javerianacali.edu.co
2 **University of Wrocław, Wrocław, Poland**
dabi@cs.uni.wroc.pl
3 **Université de Lorraine, Nancy, France**
serguei.lenglet@univ-lorraine.fr
4 **University of Wrocław, Wrocław, Poland**
ppolesiuk@cs.uni.wroc.pl

Abstract

We present sound and complete environmental bisimilarities for a variant of Dybvig et al.'s calculus of multi-prompted delimited-control operators with dynamic prompt generation. The reasoning principles that we obtain generalize and advance the existing techniques for establishing program equivalence in calculi with single-prompted delimited control.

The basic theory that we develop is presented using Madiot et al.'s framework that allows for smooth integration and composition of up-to techniques facilitating bisimulation proofs. We also generalize the framework in order to express environmental bisimulations that support equivalence proofs of evaluation contexts representing continuations. This change leads to a novel and powerful up-to technique enhancing bisimulation proofs in the presence of control operators.

1998 ACM Subject Classification D.3.3 Language Constructs and Features, F.3.2 Semantics of Programming Languages

Keywords and phrases delimited continuation, dynamic prompt generation, contextual equivalence, environmental bisimulation, up-to technique

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.9

1 Introduction

Control operators for delimited continuations, introduced independently by Felleisen [12] and by Danvy and Filinski [9], allow the programmer to delimit the current context of computation and to abstract such a delimited context as a first-class value. It has been shown that all computational effects are expressible in terms of delimited continuations [13], and so there exists a large body of work devoted to this canonical control structure, including our work on a theory of program equivalence for the operators shift and reset [5, 6, 7].

In their paper on type-directed partial evaluation for typed λ-calculus with sums, Balat et al. [2] have demonstrated that Gunter et al.'s delimited-control operators set and cupto [15], that support multiple prompts along with dynamic prompt generation, can have a practical advantage over single-prompted operators such as shift and reset. Delimited-control operators

* This work was partially supported by PHC Polonium and by National Science Centre, Poland, grant no. 2014/15/B/ST6/00619.

with dynamically-generated prompts are now available in several production programming languages such as OCaml [19] and Racket [14], and they have been given formal semantic treatment in the literature. In particular, Dybvig et al. [11] have proposed a calculus that extends the call-by-value λ-calculus with several primitives that allow for: fresh-prompt generation, delimiting computations with a prompt, abstracting control up to the corresponding prompt, and composing captured continuations. Dybvig et al.'s building blocks were shown to be able to naturally express most of other existing control operators and as such they form a general framework for studying delimited continuations. Reasoning about program equivalence in Dybvig et al.'s calculus is considerably more challenging than in single-prompted calculi: one needs to reconcile control effects with the intricacies introduced by fresh-prompt generation and local visibility of prompts.

In this article we investigate the behavioral theory of a slightly modified version of Dybvig et al.'s calculus that we call the $\lambda_{\mathcal{G}\#}$-calculus. One of the most natural notions of program equivalence in languages based on the λ-calculus is *contextual equivalence*: two terms are contextually equivalent if we cannot distinguish them when evaluated within any context. The quantification over contexts makes this relation hard to use in practice, so it is common to characterize it using simpler relations, like coinductively defined *bisimilarities*. As pointed out in [21], among the existing notions of bisimilarities, *environmental bisimilarity* [29] is the most appropriate candidate to characterize contextual equivalence in a calculus with generated resources, such as prompts in $\lambda_{\mathcal{G}\#}$. Indeed, this bisimilarity features an environment which accumulates knowledge about the terms we compare. This is crucial in our case to remember the relationships between the prompts generated by the compared programs. We therefore define environmental bisimilarities for $\lambda_{\mathcal{G}\#}$, as well as *up-to techniques*, which are used to simplify the equivalence proof of two given programs. We do so using the recently developed framework of Madiot et al. [25, 24], where it is simpler to prove that a bisimilarity and its up-to techniques are *sound* (i.e., imply contextual equivalence).

After presenting the syntax, semantics, and contextual equivalence of the calculus in Section 2, in Section 3 we define a sound and complete environmental bisimilarity and its corresponding up-to techniques. In particular, we define a bisimulation *up to context*, which allows to forget about a common context when comparing two terms in a bisimulation proof. The bisimilarity we define is useful enough to prove, e.g., the folklore theorem about delimited control [4] expressing that the static delimited-control operators shift and reset [9] can be simulated by the dynamic control operators control and prompt [12]. The technique, however, in general requires a cumbersome analysis of terms of the form $E[e]$, where E is a captured evaluation context and e is any expression (not necessarily a value). We therefore define in Section 4 a refined bisimilarity, called $\star$-bisimilarity, and a more expressive bisimulation up to context, which allows to factor out a context built with captured continuations. Proving the soundness of these two relations requires us to extend Madiot et al.'s framework. These results non-trivially generalize and considerably improve the existing techniques [7]. Finally, we discuss related work and conclude in Section 5. An accompanying research report [1] contains the proofs.

2 The Calculus $\lambda_{\mathcal{G}\#}$

The calculus we consider, called $\lambda_{\mathcal{G}\#}$, extends the call-by-value λ-calculus with four building blocks for constructing delimited-control operators as first proposed by Dybvig et al. [11]. [1]

[1] Dybvig et al.'s control operators slightly differ from their counterparts considered in this work, but they can be straightforwardly macro-expressed in the $\lambda_{\mathcal{G}\#}$-calculus.

Syntax. We assume we have a countably infinite set of term variables, ranged over by x, y, z, and k, as well as a countably infinite set of prompts, ranged over by p, q. Given an entity denoted by a meta-variable m, we write $\overrightarrow{m}$ for a (possibly empty) sequence of such entities. Expressions (e), values (v), and evaluation contexts (E) are defined as follows:

$$\begin{array}{lcll} e & ::= & v \mid e\ e \mid \mathcal{P}x.e \mid \#_v e \mid \mathcal{G}_v x.e \mid v \triangleleft e & \text{(expressions)} \\ v & ::= & x \mid \lambda x.e \mid p \mid \ulcorner E \urcorner & \text{(values)} \\ E & ::= & \Box \mid E\ e \mid v\ E \mid \#_p E & \text{(evaluation contexts)} \end{array}$$

Values include captured evaluation contexts $\ulcorner E \urcorner$, representing delimited continuations, as well as generated prompts p. Expressions include the four building blocks for delimited control: $\mathcal{P}x.e$ is a prompt-generating construct, where x represents a fresh prompt locally visible in e, $\#_v e$ is a control delimiter for e, $\mathcal{G}_v x.e$ is a continuation grabbing or capturing construct, and $v \triangleleft e$ is a throw construct.

Evaluation contexts, in addition to the standard call-by-value contexts, include delimited contexts of the form $\#_p E$, and they are interpreted outside-in. We use the standard notation $E[e]$ ($E[E']$) for plugging a context E with an expression e (with a context E'). Evaluation contexts are a special case of (general) contexts, understood as a term with a hole and ranged over by C.

The expressions $\lambda x.e$, $\mathcal{P}x.e$, and $\mathcal{G}_v x.e$ bind x; we adopt the standard conventions concerning α-equivalence. If x does not occur in e, we write $\lambda\_.e$, $\mathcal{P}\_.e$, and $\mathcal{G}_v\_.e$. The set of free variables of e is written $\mathsf{fv}(e)$; a term e is called closed if $\mathsf{fv}(e) = \emptyset$. We extend these notions to evaluation contexts. We write $\#(e)$ (or $\#(E)$) for the set of all prompts that occur in e (or E respectively). The set $\mathsf{sp}(E)$ of surrounding prompts in E is the set of all prompts guarding the hole in E, defined as $\{p \mid \exists E_1, E_2, E = E_1[\#_p E_2]\}$.

Reduction semantics. The reduction semantics of $\lambda_{\mathcal{G}\#}$ is given by the following rules:

$$\begin{array}{rcll} (\lambda x.e)\ v & \to & e\{v/x\} & \\ \#_p v & \to & v & \\ \#_p E[\mathcal{G}_p x.e] & \to & e\{\ulcorner E \urcorner / x\} & p \notin \mathsf{sp}(E) \\ \ulcorner E \urcorner \triangleleft e & \to & E[e] & \\ \mathcal{P}x.e & \to & e\{p/x\} & p \notin \#(e) \end{array}$$

$$\textsc{Compatibility}\quad \frac{e_1 \to e_2 \qquad \mathsf{fresh}(e_2, e_1, E)}{E[e_1] \to E[e_2]}$$

The first rule is the standard β_v-reduction. The second rule signals that a computation has been completed for a given prompt. The third rule abstracts the evaluation context up to the dynamically nearest control delimiter matching the prompt of the grab operator. In the fourth rule, an expression is thrown (plugged, really) to the captured context. Note that, like in Dybvig et al.'s calculus, the expression e is not evaluated before the throw operation takes place. In the last rule, a prompt p is generated under the condition that it is fresh for e.

The compatibility rule needs a side condition, simply because a prompt that is fresh for e may not be fresh for a surrounding evaluation context. Given three entities m_1, m_2, m_3 for which $\#$ is defined, we write $\mathsf{fresh}(m_1, m_2, m_3)$ for the condition $(\#(m_1) \backslash \#(m_2)) \cap \#(m_3) = \varnothing$, so the side condition states that E must not mention prompts generated in the reduction step $e_1 \to e_2$. This approach differs from the previous work on bisimulations for resource-generating constructs [23, 22, 30, 31, 32, 3, 26], where configurations of the operational semantics contain explicit information about the resources, typically represented by a set. We find our way of proceeding less invasive to the semantics of the calculus.

When reasoning about reductions in the $\lambda_{\mathcal{G}\#}$-calculus, we rely on the notion of *permutation* (a bijection on prompts), ranged over by σ, which allows to reshuffle the prompts of an expression to avoid potential collisions (e with prompts permuted by σ is written $e\sigma$). E.g., we can use the first item of the following lemma before applying the compatibility rule, to be sure that any prompt generated by $e_1 \to e_2$ is not in $\#(E)$.

▶ **Lemma 1.** *Let σ be a permutation.*
- *If $e_1 \to e_2$ then $e_1\sigma \to e_2\sigma$.*
- *For any entities m_1, m_2, m_3, we have* $\mathsf{fresh}(m_1, m_2, m_3)$ *iff* $\mathsf{fresh}(m_1\sigma, m_2\sigma, m_3\sigma)$.

A closed term e either uniquely, up to permutation of prompts, reduces to a term e', or it is a normal form (i.e., there is no e'' such that $e \to e''$). In the latter case, we distinguish values, control-stuck terms $E[\mathcal{G}_p k.e]$ where $p \notin \mathsf{sp}(E)$, and the remaining expressions that we call errors (e.g., $E[p\, v]$ or $E[\mathcal{G}_{\lambda x.e} k.e']$). We write $e_1 \to^* e_2$ if e_1 reduces to e_2 in many (possibly 0) steps, and we write $e \not\to$ when a term e diverges (i.e., there exists an infinite sequence of reductions starting with e) or when it reduces (in many steps) to an error.

When writing examples, we use the fixed-point operator fix, let-construct, conditional if along with boolean values true and false, and sequencing ";", all defined as in the call-by-value λ-calculus. We also use the diverging term $\Omega \stackrel{\mathsf{def}}{=} (\lambda x.x\, x)\,(\lambda x.x\, x)$, and we define an operator $\stackrel{?}{=}$ to test the equality between prompts, as follows:

$$e_1 \stackrel{?}{=} e_2 \quad \stackrel{\mathsf{def}}{=} \quad \mathsf{let}\ x = e_1\ \mathsf{in}\ \mathsf{let}\ y = e_2\ \mathsf{in}\ \#_x((\#_y \mathcal{G}_{x\_}.\mathsf{false});\mathsf{true})$$

If e_1 and e_2 evaluate to different prompts, then the grab operator captures the context up to the outermost prompt to throw it away, and false is returned; otherwise, true is returned.

Contextual equivalence. We now define formally what it takes for two terms to be considered equivalent in the $\lambda_{\mathcal{G}\#}$-calculus. First, we characterize when two closed expressions have equivalent observable actions in the calculus, by defining the following relation $\sim$.

▶ **Definition 2.** We say that e_1 and e_2 have equivalent observable actions, noted $e_1 \sim e_2$, if
1. $e_1 \to^* v_1$ iff $e_2 \to^* v_2$,
2. $e_1 \to^* E_1[\mathcal{G}_{p_1} x.e_1']$ iff $e_2 \to^* E_2[\mathcal{G}_{p_2} x.e_2']$, where $p_1 \notin \mathsf{sp}(E_1)$ and $p_2 \notin \mathsf{sp}(E_2)$,
3. $e_1 \not\to$ iff $e_2 \not\to$.

We can see that errors and divergence are treated as equivalent, which is standard.

Based on $\sim$, we define *contextual equivalence* as follows.

▶ **Definition 3** (Contextual equivalence)**.** Two closed expressions e_1 and e_2 are contextually equivalent, written, $e_1 \equiv_E e_2$, if for all E such that $\#(E) = \varnothing$, we have $E[e_1] \sim E[e_2]$.

Contextual equivalence can be extended to open terms in a standard way: if $\mathsf{fv}(e_1) \cup \mathsf{fv}(e_2) \subseteq \overrightarrow{x}$, then $e_1 \equiv_E e_2$ if $\lambda\overrightarrow{x}.e_1 \equiv_E \lambda\overrightarrow{x}.e_2$. We test terms using only promptless contexts, because the testing context should not use prompts that are private for the tested expressions. For example, the expressions $\lambda f.f\ p\ q$ and $\lambda f.f\ q\ p$ should be considered equivalent if nothing is known from the outside about p and q. As common in calculi with resource generation [31, 30, 29], testing with evaluation contexts (as in $\equiv_E$) is not the same as testing with all contexts: we have $\mathcal{P}x.x \equiv_E p$, but these terms can be distinguished by

$$\mathsf{let}\ f = \lambda x.\Box\ \mathsf{in}\ \mathsf{if}\ f\ \lambda x.x \stackrel{?}{=} f\ \lambda x.x\ \mathsf{then}\ \Omega\ \mathsf{else}\ \lambda x.x,$$

In the rest of the article, we show how to characterize $\equiv_E$ with environmental bisimilarities.[2]

[2] If $\equiv_C$ is the contextual equivalence testing with all contexts, then we can prove that $e_1 \equiv_C e_2$ iff $\lambda x.e_1 \equiv_E \lambda x.e_2$, where x is any variable. We therefore obtain a proof method for $\equiv_C$ as well.

3 Environmental Bisimilarity

In this section, we propose a first characterization of $\equiv_E$ using an environmental bisimilarity. We express the bisimilarity in the style of [25], using a so called first-order labeled transition system (LTS), to factorize the soundness proofs of the bisimilarity and its up-to techniques. We start by defining the LTS and its corresponding bisimilarity.

3.1 Labeled Transition System and Bisimilarity

In the original formulation of environmental bisimulation [29], two expressions e_1 and e_2 are compared under some environment $\mathcal{E}$, which represents the knowledge of an external observer about e_1 and e_2. The definition of the bisimulation enforces some conditions on e_1 and e_2 as well as on $\mathcal{E}$. In Madiot et al.'s framework [25, 24], the conditions on e_1, e_2, and $\mathcal{E}$ are expressed using a LTS between *states* of the form (Γ, e_1) (and (Δ, e_2)), where Γ (and Δ) is a finite sequence of values corresponding to the first (and second) projection of the environment $\mathcal{E}$. Note that in (Γ, e_1), e_1 may be a value, and therefore a state can be simply of the form Γ. Transitions from states of the form (Γ, e_1) (where e_1 is not a value) express conditions on e_1, while transitions from states of the form Γ explain how we compare environments. In the rest of the paper we use Γ, Δ to range over finite sequences of values, and we write Γ_i, Δ_i for the i^{th} element of the sequence. We use Σ, Θ to range over states.

Figure 1 presents the LTS $\xrightarrow{\alpha}$, where α ranges over all the labels. We define $\#(\Gamma)$ as $\bigcup_i \#(\Gamma_i)$. The transition $\xrightarrow{\mathbb{E}}$ uses a relation $e \xrightarrow{=} e'$, defined as follows: if $e \to e'$, then $e \xrightarrow{=} e'$, and if e is a normal form, then $e \xrightarrow{=} e$.[3] To build expressions out of sequences of values, we use different kinds of *multi-hole contexts* defined as follows.

$$
\begin{array}{rcll}
\mathbb{C} & ::= & \mathbb{C}_{\mathsf{v}} \mid \mathbb{C}\,\mathbb{C} \mid \mathcal{P}x.\mathbb{C} \mid \#_{\mathbb{C}_{\mathsf{v}}}\mathbb{C} \mid \mathcal{G}_{\mathbb{C}_{\mathsf{v}}}x.\mathbb{C} \mid \mathbb{C}_{\mathsf{v}} \triangleleft \mathbb{C} & \text{(contexts)} \\
\mathbb{C}_{\mathsf{v}} & ::= & x \mid \lambda x.\mathbb{C} \mid \ulcorner \mathbb{E} \urcorner \mid \Box_i & \text{(value contexts)} \\
\mathbb{E} & ::= & \Box \mid \mathbb{E}\,\mathbb{C} \mid \mathbb{C}_{\mathsf{v}}\,\mathbb{E} \mid \#_{\Box_i}\mathbb{E} & \text{(evaluation contexts)}
\end{array}
$$

The holes of a multi-hole context are indexed, except for the special hole $\Box$ of an evaluation context $\mathbb{E}$, which is in evaluation position (that is, filling the other holes of $\mathbb{E}$ with values gives a regular evaluation context E). We write $\mathbb{C}[\Gamma]$ (respectively $\mathbb{C}_{\mathsf{v}}[\Gamma]$ and $\mathbb{E}[\Gamma]$) for the application of a context $\mathbb{C}$ (respectively $\mathbb{C}_{\mathsf{v}}$ and $\mathbb{E}$) to a sequence Γ of values, which consists in replacing $\Box_i$ with Γ_i; we assume that this application produce an expression (or an evaluation context in the case of $\mathbb{E}$), i.e., each hole index in the context is smaller or equal than the size of Γ, and for each $\#_{\Box_i}\mathbb{E}$ construct, Γ_i is a prompt. We write $\mathbb{E}[e, \Gamma]$ for the same operation with evaluation contexts, where we assume that e is put in $\Box$ (note that e may also be a value). Notice that prompts are not part of the syntax of $\mathbb{C}_{\mathsf{v}}$, therefore a multi-hole context does not contain any prompt: if $\mathbb{C}[\Gamma]$, $\mathbb{C}_{\mathsf{v}}[\Gamma]$, or $\mathbb{E}[e, \Gamma]$ contains a prompt, then it comes from Γ or e. Our multi-hole contexts are promptless because $\equiv_E$ also tests with promptless contexts.

We now detail the rules of Figure 1, starting with the transitions that one can find in any call-by-value λ-calculus [25]. An internal action $(\Gamma, e_1) \xrightarrow{\tau} \Sigma$ corresponds to a reduction step, except we ensure that any generated prompt is fresh w.r.t. Γ. The transition $\Gamma \xrightarrow{\lambda, i, \mathbb{C}_{\mathsf{v}}} \Sigma$ signals that Γ_i is a λ-abstraction, which can be tested by passing it an argument built from Γ with the context $\mathbb{C}_{\mathsf{v}}$. The transition $\xrightarrow{\ulcorner.\urcorner, i, \mathbb{C}}$ for testing continuations is built the same way,

[3] The relation $\xrightarrow{=}$ is not exactly the reflexive closure of $\to$, since an expression which is not a normal form *has* to reduce.

$$\frac{e_1 \to e_2 \qquad \mathsf{fresh}(e_2, e_1, \Gamma)}{(\Gamma, e_1) \xrightarrow{\tau} (\Gamma, e_2)} \qquad \frac{\Gamma_i = \lambda x.e}{\Gamma \xrightarrow{\lambda, i, \mathbb{C}_\mathsf{v}} (\Gamma, e\{\mathbb{C}_\mathsf{v}[\Gamma]/x\})} \qquad \frac{}{\Gamma \xrightarrow{\mathsf{v}} \Gamma}$$

$$\frac{\Gamma_i = \ulcorner E \urcorner}{\Gamma \xrightarrow{\ulcorner \cdot \urcorner, i, \mathbb{C}} (\Gamma, E[\mathbb{C}[\Gamma]])} \qquad \frac{\Gamma_i = p \qquad \Gamma_j = p}{\Gamma \xrightarrow{\#, i, j} \Gamma} \qquad \frac{p \notin \#(\Gamma)}{\Gamma \xrightarrow{\#} (\Gamma, p)}$$

$$\frac{p \notin \mathsf{sp}(E) \qquad \mathbb{E}[E[\mathcal{G}_p x.e], \Gamma] \xrightarrow{=} e'}{(\Gamma, E[\mathcal{G}_p x.e]) \xrightarrow{\mathbb{E}} (\Gamma, e')}$$

Figure 1 Labeled Transition System for $\lambda_{\mathcal{G}\#}$.

except we use a context $\mathbb{C}$, because any expression can be thrown to a captured context. Finally, the transition $\Gamma \xrightarrow{\mathsf{v}} \Gamma$ means that the state Γ is composed only of values; it does not test anything on Γ, but this transition is useful for the soundness proofs of Section 3.2. When we have $\Gamma \mathcal{R} (\Delta, e)$ (where $\mathcal{R}$ is, e.g., a bisimulation), then (Δ, e) has to match with $(\Delta, e) \xrightarrow{\tau}{}^{*} \xrightarrow{\mathsf{v}} (\Delta, v)$ so that (Δ, v) is related to Γ. We can then continue the proofs with two related sequences of values. Such a transition has been suggested in [24, Remark 5.3.6] to simplify the proofs for a non-deterministic language, like $\lambda_{\mathcal{G}\#}$.

We now explain the rules involving prompts. When comparing two terms generating prompts, one can produce p and the other a different q, so we remember in Γ, Δ that p corresponds to q. But an observer can compare prompts using $\overset{?}{=}$, so p has to be related *only* to q. We check it with $\xrightarrow{\#,i,j}$: if $\Gamma \xrightarrow{\#,i,j} \Gamma$, then Δ has to match, meaning that $\Delta_i = \Delta_j$, and doing so for all j such that $\Gamma_i = \Gamma_j$ ensures that all copies of Γ_i are related only to Δ_i. The transition $\xrightarrow{\#,i,i}$ also signals that Γ_i is a prompt and should be related to a prompt. The other transition involving prompts is $\Gamma \xrightarrow{\#} (\Gamma, p)$, which encodes the possibility for an outside observer to generate fresh prompts to compare terms. If Γ is related to Δ, then Δ has to match by generating a prompt q, and we remember that p is related to q. For this rule to be automatically verified, we define the *prompt checking* rule for a relation $\mathcal{R}$ as follows:

$$\frac{\Gamma \mathcal{R} \Delta \qquad p \notin \#(\Gamma) \qquad q \notin \#(\Delta)}{(\Gamma, p) \mathcal{R} (\Delta, q)} \ (\#\text{-check})$$

Henceforth, when we construct a bisimulation $\mathcal{R}$ by giving a set of rules, we always include the (#-check) rule so that the $\xrightarrow{\#}$ transition is always verified.

Finally, the transition $\xrightarrow{\mathbb{E}}$ deals with stuck terms. An expression $E[\mathcal{G}_p x.e]$ is able to reduce if the surrounding context is able to provide a delimiter $\#_p$. However, it is possible only if p is available for the outside, and therefore is in Γ. If $p \notin \mathsf{sp}(\mathbb{E}[\Gamma])$, then $\mathbb{E}[E[\mathcal{G}_p x.e], \Gamma]$ remains stuck, and we have $\mathbb{E}[E[\mathcal{G}_p x.e], \Gamma] \xrightarrow{=} \mathbb{E}[E[\mathcal{G}_p x.e], \Gamma]$. Otherwise, it can reduce and we have $\mathbb{E}[E[\mathcal{G}_p x.e], \Gamma] \xrightarrow{=} e'$, where e' is the result after the capture. The rule for $\xrightarrow{\mathbb{E}}$ may seem demanding, as it tests stuck terms with all contexts $\mathbb{E}$, but up-to techniques will alleviate this issue (see Example 8).

For weak transitions, we define $\Rightarrow$ as $\xrightarrow{\tau}{}^{*}$, $\overset{\alpha}{\Rightarrow}$ as $\Rightarrow$ if $\alpha = \tau$ and as $\Rightarrow\xrightarrow{\alpha}\Rightarrow$ otherwise. We define bisimulation and bisimilarity using a more general notion of *progress*. Henceforth, we let $\mathcal{R}$, $\mathcal{S}$ range over relations on states.

▶ **Definition 4.** A relation $\mathcal{R}$ progresses to $\mathcal{S}$, written $\mathcal{R} \rightarrowtail \mathcal{S}$, if $\mathcal{R} \subseteq \mathcal{S}$ and $\Sigma\ \mathcal{R}\ \Theta$ implies
- if $\Sigma \xrightarrow{\alpha} \Sigma'$, then there exists Θ' such that $\Theta \stackrel{\alpha}{\Rightarrow} \Theta'$ and $\Sigma'\ \mathcal{S}\ \Theta'$;
- the converse of the above condition on Θ.

A *bisimulation* is a relation $\mathcal{R}$ such that $\mathcal{R} \rightarrowtail \mathcal{R}$, and *bisimilarity* $\approx$ is the union of all bisimulations.

3.2 Up-to Techniques, Soundness, and Completeness

Before defining the up-to techniques for $\lambda_{\mathcal{G}\#}$, we briefly recall the main definitions and results we use from [28, 25, 24]; see these works for more details and proofs. We use f, g to range over functions on relations on states. An *up-to technique* is a function f such that $\mathcal{R} \rightarrowtail f(\mathcal{R})$ implies $\mathcal{R} \subseteq \approx$. However, this definition can be difficult to use to prove that a given f is an up-to technique, so we rely on *compatibility* instead. A function f is monotone if $\mathcal{R} \subseteq \mathcal{S}$ implies $f(\mathcal{R}) \subseteq f(\mathcal{S})$. Given a set F of functions, we also write F for the function defined as $\bigcup_{f_i \in F} f_i$ (where $f \cup g$ is defined argument-wise, i.e., $(f \cup g)(R) = f(R) \cup g(R)$). Given a function f, f^ω is defined as $\bigcup_{n \in \mathbb{N}} f^n$.

▶ **Definition 5.** A function f *evolves* to g, written $f \rightsquigarrow g$, if for all $\mathcal{R} \rightarrowtail \mathcal{S}$, we have $f(\mathcal{R}) \rightarrowtail g(\mathcal{S})$. A set F of monotone functions is compatible if for all $f \in F$, $f \rightsquigarrow F^\omega$.

▶ **Lemma 6.** *Let F be a compatible set, and $f \in F$; f is an up-to technique, and $f(\approx) \subseteq \approx$.*

Proving that f is in a compatible set F is easier than proving it is an up-to technique, because we just have to prove that it evolves towards a combination of functions in F. Besides, the second property of Lemma 6 can be used to prove that $\approx$ is a congruence just by showing that bisimulation up to context is compatible.

The first technique we define allows to forget about prompt names; in a bisimulation relating (Γ, e_1) and (Δ, e_2), we remember that $\Gamma_i = p$ is related to $\Delta_i = q$ by their position i, not by their names. Consequently, we can apply different permutations to the two states to rename the prompts without harm, and bisimulation *up to permutations*[4] allows us to do so. Given a relation $\mathcal{R}$, we define $\mathsf{perm}(\mathcal{R})$ as $\Sigma\sigma_1\ \mathsf{perm}(\mathcal{R})\ \Theta\sigma_2$, assuming $\Sigma\ \mathcal{R}\ \Theta$ and σ_1, σ_2 are any permutations.

We then allow to remove or add values from the states with, respectively, bisimulation *up to weakening* weak and bisimulation *up to strengthening* str, defined as follows

$$\frac{(\overrightarrow{v}, \Gamma, e_1)\ \mathcal{R}\ (\overrightarrow{w}, \Delta, e_2)}{(\Gamma, e_1)\ \mathsf{weak}(\mathcal{R})\ (\Delta, e_2)} \qquad \frac{(\Gamma, e_1)\ \mathcal{R}\ (\Delta, e_2)}{(\Gamma, \mathbb{C}_\mathsf{v}[\Gamma], e_1)\ \mathsf{str}(\mathcal{R})\ (\Delta, \mathbb{C}_\mathsf{v}[\Delta], e_2)}$$

Bisimulation up to weakening diminishes the testing power of states, since less values means less arguments to build from for the transitions $\xrightarrow{\lambda, i, \mathbb{C}_\mathsf{v}}$, $\xrightarrow{\ulcorner \cdot \urcorner, i, \mathbb{C}}$, and $\xrightarrow{\mathbb{E}}$. This up-to technique is usual for environmental bisimulations, and is called "up to environment" in [29]. In contrast, str adds values to the state, but without affecting the testing power, since the added values are built from the ones already in Γ, Δ.

Finally, we define the well-known bisimulation up to context, which allows to factor out a common context when comparing terms. As usual for environmental bisimulations [29], we define two kinds of bisimulation up to context, depending whether we operate on values or

[4] Madiot defines a bisimulation "up to permutation" in [24] which reorders values in a state. Our bisimulation up to permutations operates on prompts.

any expressions. For values, we can factor out any common context $\mathbb{C}$, but for expressions that are not values, we can factor out only an evaluation context $\mathbb{E}$, since factoring out any context in that case would lead to an unsound up-to technique [24]. We define up to context for values ctx and for any expression ectx as follows:

$$\frac{\Gamma\ \mathcal{R}\ \Delta}{(\Gamma, \mathbb{C}[\Gamma])\ \mathsf{ctx}(\mathcal{R})\ (\Delta, \mathbb{C}[\Delta])} \qquad \frac{(\Gamma, e_1)\ \mathcal{R}\ (\Delta, e_2)}{(\Gamma, \mathbb{E}[e_1, \Gamma])\ \mathsf{ectx}(\mathcal{R})\ (\Delta, \mathbb{E}[e_2, \Delta])}$$

▶ **Lemma 7.** *The set* $\{\mathsf{perm}, \mathsf{weak}, \mathsf{str}, \mathsf{ctx}, \mathsf{ectx}\}$ *is compatible.*

The function ectx is particularly helpful in dealing with stuck terms, as we can see below.

▶ **Example 8.** Let $\Sigma \stackrel{\mathsf{def}}{=} (\Gamma, \mathcal{G}_p x.e_1)$ and $\Theta \stackrel{\mathsf{def}}{=} (\Delta, \mathcal{G}_q x.e_2)$ (for some e_1, e_2), so that $\Sigma\ \mathcal{R}\ \Theta$. If p and q are not in Γ, Δ, then the two expressions remain stuck, as we have $\Sigma \xrightarrow{\mathbb{E}} (\Gamma, \mathbb{E}[\mathcal{G}_p x.e_1, \Gamma])$ and similarly for Θ. We have directly $(\Gamma, \mathbb{E}[\mathcal{G}_p x.e_1, \Gamma])\ \mathsf{ectx}(\mathcal{R})\ (\Delta, \mathbb{E}[\mathcal{G}_q x.e_2, \Delta])$. Otherwise, the capture can be triggered with a context $\mathbb{E}$ of the form $\mathbb{E}_1[\#_{\square_i}\mathbb{E}_2]$, giving $\Sigma \xrightarrow{\mathbb{E}} (\Gamma, \mathbb{E}_1[e_1\{\ulcorner\mathbb{E}_2[\Gamma]\urcorner/x\}, \Gamma])$ and $\Theta \xrightarrow{\mathbb{E}} (\Delta, \mathbb{E}_1[e_2\{\ulcorner\mathbb{E}_2[\Delta]\urcorner/x\}, \Delta])$. Thanks to ectx, we can forget about $\mathbb{E}_1$ which does not play any role, and continue the bisimulation proof by focusing only on $(\Gamma, e_1\{\ulcorner\mathbb{E}_2[\Gamma]\urcorner/x\})$ and $(\Delta, e_2\{\ulcorner\mathbb{E}_2[\Delta]\urcorner/x\})$.

Because bisimulation up to context is compatible, Lemma 6 ensures that $\approx$ is a congruence w.r.t. all contexts for values, and w.r.t. evaluation contexts for all expressions. As a corollary, we can deduce that $\approx$ is sound w.r.t. $\equiv_E$; we can also prove that it is complete w.r.t. $\equiv_E$, leading to the following full characterization result.

▶ **Theorem 9.** $e_1 \equiv_E e_2$ *iff* $(\emptyset, e_1) \approx (\emptyset, e_2)$.

For completeness, we prove that $\{(\Gamma, e_1), (\Delta, e_2) \mid \forall \mathbb{E}, \mathbb{E}[e_1, \Gamma] \sim \mathbb{E}[e_2, \Delta]\}$ is a bisimulation up to permutation; the proof is in [1, Appendix A.1].

3.3 Example

As an example, we show a folklore theorem about delimited control [4], stating that the static operators shift and reset can be simulated by the dynamic operators $\mathsf{control}$ and prompt. In fact, what we prove is a more general and stronger result than the original theorem, since we demonstrate that this simulation still holds when multiple prompts are around.

▶ **Example 10** (Folklore theorem)**.** We encode shift, reset, $\mathsf{control}$, and prompt as follows

$$\begin{array}{llll} \mathsf{shift}_p & \stackrel{\mathsf{def}}{=} & \lambda f.\mathcal{G}_p k.\#_p f(\lambda y.\#_p k \triangleleft y) & \qquad \mathsf{control}_p \stackrel{\mathsf{def}}{=} \lambda f.\mathcal{G}_p k.\#_p f(\lambda y.k \triangleleft y) \\ \mathsf{reset}_p & \stackrel{\mathsf{def}}{=} & \ulcorner\#_p\square\urcorner & \qquad \mathsf{prompt}_p \stackrel{\mathsf{def}}{=} \ulcorner\#_p\square\urcorner \end{array}$$

Let $\mathsf{shift'}_p \stackrel{\mathsf{def}}{=} \lambda f.\mathsf{control}_p\ (\lambda l.f\ (\lambda z.\mathsf{prompt}_p \triangleleft l\ z))$; we prove that the pair $(\mathsf{shift}_p, \mathsf{reset}_p)$ (encoded as $\lambda f.f\,\mathsf{shift}_p\,\mathsf{reset}_p$) is bisimilar to $(\mathsf{shift'}_p, \mathsf{prompt}_p)$ (encoded as $\lambda f.f\,\mathsf{shift'}_p\,\mathsf{prompt}_p$).

Proof. We iteratively build a relation $\mathcal{R}$ closed under (#-check) such that $\mathcal{R}$ is a bisimulation up to context, starting with $(p, \mathsf{shift}_p)\ \mathcal{R}\ (p, \mathsf{shift'}_p)$. The transition $\xrightarrow{\#,1,1}$ is easy to check. For $\xrightarrow{\lambda,2,\mathbb{C}_\mathsf{v}}$, we obtain states of the form $(p, \mathsf{shift}_p, e_1)$, $(p, \mathsf{shift'}_p, e_2)$ that we add to $\mathcal{R}$, where e_1 and e_2 are the terms below

$$\frac{\Gamma\ \mathcal{R}\ \Delta}{(\Gamma, \mathcal{G}_p k.\#_p \mathbb{C}_\mathsf{v}[\Gamma]\ (\lambda y.\#_p k \triangleleft y))\ \mathcal{R}\ (\Delta, \mathcal{G}_p k.\#_p(\lambda l.\mathbb{C}_\mathsf{v}[\Delta]\ (\lambda z.\mathsf{prompt}_p \triangleleft l\ z))\ (\lambda y.k \triangleleft y))}$$

We use an inductive, more general rule, because we want $\xrightarrow{\lambda,2,\mathbb{C}_\mathsf{v}}$ to be still verified after we extend (p, shift_p) and (p, shift'_p). The terms e_1 and e_2 are stuck, so we test them with $\xrightarrow{\mathbb{E}}$. If $\mathbb{E}$ does not trigger the capture, we obtain $\mathbb{E}[e_1, \Gamma]$ and $\mathbb{E}[e_2, \Delta]$, and we can use ectx to conclude. Otherwise, $\mathbb{E} = \mathbb{E}'[\#_{\Box_1}\mathbb{E}'']$ (where $\#_{\Box_1}$ does not surround $\Box$ in $\mathbb{E}''$), and we get

$$\mathbb{E}'[\#_p\mathbb{C}_\mathsf{v}[\Gamma]\ (\lambda y.\#_p\ulcorner\mathbb{E}''[\Gamma]\urcorner \triangleleft y), \Gamma] \text{ and } \mathbb{E}'[\#_p\mathbb{C}_\mathsf{v}[\Delta]\ (\lambda z.\mathsf{prompt}_p \triangleleft (\lambda y.\ulcorner\mathbb{E}''[\Delta]\urcorner \triangleleft y)\ z), \Delta]$$

We want to use ctx to remove the common context $\mathbb{E}'[\#_{\Box_1}\mathbb{C}_\mathsf{v}\ \Box_i]$, which means that we have to add the following states in the definition of $\mathcal{R}$ (again, inductively):

$$\frac{\Gamma\ \mathcal{R}\ \Delta}{(\Gamma, \lambda y.\#_p\ulcorner\mathbb{E}''[\Gamma]\urcorner \triangleleft y)\ \mathcal{R}\ (\Delta, \lambda z.\mathsf{prompt}_p \triangleleft (\lambda y.\ulcorner\mathbb{E}''[\Delta]\urcorner \triangleleft y)\ z)}$$

Testing these functions with $\xrightarrow{\lambda,i,\mathbb{C}_\mathsf{v}}$ gives on both sides states where $\#_{\Box_1}\mathbb{E}''[\mathbb{C}_\mathsf{v}]$ can be removed with ctx. Because $(\varnothing, \lambda f.f\ \mathsf{shift}_p\ \mathsf{reset}_p)\ \mathsf{weak}(\mathsf{ctx}(\mathcal{R}))\ (\varnothing, \lambda f.f\ \mathsf{shift}'_p\ \mathsf{prompt}_p)$, it is enough to conclude. Indeed, $\mathcal{R}$ is a bisimulation up to context, so $\mathcal{R} \subseteq \approx$, which implies $\mathsf{weak}(\mathsf{ctx}(\mathcal{R})) \subseteq \mathsf{weak}(\mathsf{ctx}(\approx))$ (because weak and ctx are monotone), and $\mathsf{weak}(\mathsf{ctx}(\approx)) \subseteq \approx$ (by Lemma 6). Note that this reasoning works for any combination of monotone up-to techniques and any bisimulation (up-to). ◀

What makes the proof of Example 10 quite simple is that we relate (p, shift_p) and (p, shift'_p), meaning that p can be used by an outside observer. But the control operators $(\mathsf{shift}_p, \mathsf{reset}_p)$ and $(\mathsf{shift}'_p, \mathsf{prompt}_p)$ should be the only terms available for the outside, since p is used only to implement them. If we try to prove equivalent these two pairs directly, i.e., while keeping p private, then testing reset_p and prompt_p with $\xrightarrow{\ulcorner\cdot\urcorner,2,\mathbb{C}}$ requires a cumbersome analysis of the behaviors of $\#_p\mathbb{C}[\Gamma]$ and $\#_p\mathbb{C}[\Delta]$. In the next section, we define a new kind of bisimilarity with a powerful up-to technique to make such proofs more tractable.

4 The ⋆-Bisimilarity

4.1 Motivation

Testing continuations. In Section 3, a continuation $\Gamma_i = \ulcorner E\urcorner$ is tested with $\Gamma \xrightarrow{\ulcorner\cdot\urcorner,i,\mathbb{C}} (\Gamma, E[\mathbb{C}[\Gamma]])$. We then have to study the behavior of $E[\mathbb{C}[\Gamma]]$, which depends primarily on how $\mathbb{C}[\Gamma]$ reduces; e.g., if $\mathbb{C}[\Gamma]$ diverges, then E does not play any role. Consequently, the transition $\xrightarrow{\ulcorner\cdot\urcorner,i,\mathbb{C}}$ does not really test the continuation directly, since we have to reduce $\mathbb{C}[\Gamma]$ first. To really exhibit the behavior of the continuation, we change the transition so that it uses a value context instead of a general one. We then have $\Gamma \xrightarrow{\ulcorner\cdot\urcorner,i,\mathbb{C}_\mathsf{v}} (\Gamma, E[\mathbb{C}_\mathsf{v}[\Gamma]])$, and the behavior of the term we obtain depends primarily on E. However, this is not equivalent to testing with $\mathbb{C}$, since $\mathbb{C}[\Gamma]$ may interact in other ways with E if $\mathbb{C}[\Gamma]$ is a stuck term. If E is of the form $E'[\#_pE'']$, and p is in Γ, then $\mathbb{C}$ may capture E'', since p can be used to build an expression of the form $\mathcal{G}_px.e$. To take into account this possibility, we introduce a new transition $\Gamma \xrightarrow{\ulcorner\cdot\urcorner,i,j} (\Gamma, \ulcorner E'\urcorner, \ulcorner E''\urcorner)$, which decomposes $\Gamma_i = E'[\#_pE'']$ into $\ulcorner E'\urcorner$ and $\ulcorner E''\urcorner$, provided $\Gamma_j = p$. The stuck term $\mathbb{C}[\Gamma]$ may also capture E entirely, as part of a bigger context of the form $\mathbb{E}_1[E[\mathbb{E}_2]]$. To take this into account, we introduce a way to build such contexts using captured continuations. This is also useful to make bisimulation up to context more expressive, as we explain in the next paragraph.

A more expressive bisimulation up to context. As we already pointed out in [7], bisimulation up to context is not very helpful in the presence of control operators. For example, suppose we prove the β_Ω axiom of [18], i.e., $(\lambda x.E[x])\ e$ is equivalent to $E[e]$ if $x \notin \mathsf{fv}(E)$ and $\mathsf{sp}(E) = \emptyset$. If e is a stuck term $\mathcal{G}_p y.e_1$, we have to compare $e_1\{\ulcorner E_1[(\lambda x.E[x])\ \Box]\urcorner/y\}$ and $e_1\{\ulcorner E_1[E]\urcorner/y\}$ for some E_1. If e_1 is of the form $y \triangleleft (y \triangleleft e_2)$, then we get respectively $E_1[(\lambda x.E[x])\ E_1[(\lambda x.E[x])\ e_2]]$ and $E_1[E[E_1[E[e_2]]]]$. We can see that the two resulting expressions have the same shape, and yet we can only remove the outermost occurrence of E_1 with ectx. The problem is that bisimulation up to context can factor out only a *common* context. We want an up-to technique able to identify *related* contexts, i.e., contexts built out of related continuations. To do so, we modify the multi-hole contexts to include a construct $\star_i[\mathbb{C}]$ with a special hole $\star_i$, which can be filled only with $\ulcorner E\urcorner$ to produce a context $E[\mathbb{C}]$. As a result, if $\Gamma = (\ulcorner(\lambda x.E[x])\ \Box\urcorner)$ and $\Delta = (\ulcorner E\urcorner)$, then $E_1[(\lambda x.E[x])\ E_1[(\lambda x.E[x])\ \Box]]$ and $E_1[E[E_1[E[\Box]]]]$ can be written $\mathbb{E}[\Gamma]$, $\mathbb{E}[\Delta]$ with $\mathbb{E} = E_1[\star_1[E_1[\star_1[\Box]]]]$. We can then focus only on testing Γ and Δ.

However, such a bisimulation up to related context would be unsound if not restricted in some way. Indeed, let $\ulcorner E_1\urcorner$, $\ulcorner E_2\urcorner$ be any continuations, and let $\Gamma = (\ulcorner E_1\urcorner)$, $\Delta = (\ulcorner E_2\urcorner)$. Then the transitions $\Gamma \xrightarrow{\ulcorner.\urcorner,1,\mathbb{C}_\mathsf{v}} (\Gamma, E_1[\mathbb{C}_\mathsf{v}[\Gamma]])$ and $\Delta \xrightarrow{\ulcorner.\urcorner,1,\mathbb{C}_\mathsf{v}} (\Delta, E_2[\mathbb{C}_\mathsf{v}[\Delta]])$ produce states of the form $(\Gamma, \mathbb{C}[\Gamma])$, $(\Delta, \mathbb{C}[\Delta])$ with $\mathbb{C} = \star_1[\mathbb{C}_\mathsf{v}]$. If bisimulation up to related context was sound in that case, it would mean that $\ulcorner E_1\urcorner$ and $\ulcorner E_2\urcorner$ would be bisimilar for all E_1 and E_2, which, of course, is wrong.[5] To prevent this, we distinguish *passive* transitions (such as $\xrightarrow{\ulcorner.\urcorner,i,\mathbb{C}_\mathsf{v}}$) from the other ones (called *active*), so that only selected up-to techniques (referred to as *strong*) can be used after a passive transition. In contrast, any up-to technique (including this new bisimulation up to related context) can be used after an active transition. To formalize this idea, we have to extend Madiot et al.'s framework to allow such distinctions between transitions and between up-to techniques.

4.2 Labeled Transition System and Bisimilarity

First, we explain how we alter the LTS of Section 3.1 to implement the changes we sketched in Section 4.1. We extend the grammar of multi-hole contexts $\mathbb{C}$ (resp. $\mathbb{E}$) by adding the production $\star_i[\mathbb{C}]$ (resp. $\star_i[\mathbb{E}]$), where the hole $\star_i$ can be filled only with a continuation (the grammar of value contexts $\mathbb{C}_\mathsf{v}$ is unchanged). When we write $(\star_i[\mathbb{C}])[\Gamma]$, we assume Γ_i is a continuation $\ulcorner E\urcorner$, and the result of the operation is $E[\mathbb{C}[\Gamma]]$ (and similarly for $\mathbb{E}$).

We also change the way we deal with captured contexts, by using the following rules:

$$\frac{\Gamma_i = \ulcorner E\urcorner}{\Gamma \xrightarrow{\ulcorner.\urcorner,i,\mathbb{C}_\mathsf{v}} (\Gamma, E[\mathbb{C}_\mathsf{v}[\Gamma]])} \qquad \frac{\Gamma_i = \ulcorner E_1[\#_p E_2]\urcorner \qquad \Gamma_j = p \qquad p \notin \mathsf{sp}(E_2)}{\Gamma \xrightarrow{\ulcorner.\urcorner,i,j} (\Gamma, \ulcorner E_1\urcorner, \ulcorner E_2\urcorner)}$$

The transition $\xrightarrow{\ulcorner.\urcorner,i,\mathbb{C}_\mathsf{v}}$ is the same as in Section 3, except that it tests with an argument built with a value context $\mathbb{C}_\mathsf{v}$ instead of a regular context $\mathbb{C}$. We also introduce the transition $\xrightarrow{\ulcorner.\urcorner,i,j}$, which decomposes a captured context $\ulcorner E_1[\#_p E_2]\urcorner$ into sub-contexts $\ulcorner E_1\urcorner$, $\ulcorner E_2\urcorner$, provided that p is in Γ. This transition is necessary to take into account the possibility for an external observer to capture a part of a context, scenario which can no longer be tested with $\xrightarrow{\ulcorner.\urcorner,i,\mathbb{C}_\mathsf{v}}$, as explained in Section 4.1, and as illustrated with the next example.

[5] The problem is similar if we test continuations using contexts $\mathbb{C}$ (as in Section 3) instead of $\mathbb{C}_\mathsf{v}$.

▶ **Example 11.** Let $\Gamma = (p, \ulcorner \#_p \Box \urcorner)$, $\Delta = (q, \ulcorner \Box \urcorner)$; then $\Gamma \xrightarrow{\ulcorner . \urcorner, 2, \mathbb{C}_\mathsf{v}} (\Gamma, \#_p \mathbb{C}_\mathsf{v}[\Gamma]) \xrightarrow{\tau} (\Gamma, \mathbb{C}_\mathsf{v}[\Gamma])$ and $\Delta \xrightarrow{\ulcorner . \urcorner, 2, \mathbb{C}_\mathsf{v}} (\Delta, \mathbb{C}_\mathsf{v}[\Delta])$. Without the $\xrightarrow{\ulcorner . \urcorner, i, j}$ transition, Γ and Δ would be bisimilar, which would not be sound (they are distinguished by the context $\Box_2 \triangleleft \mathcal{G}_{\Box_1} x.\Omega$).

The other rules are not modified, but their meaning is still affected by the change in the contexts grammars: the transitions $\xrightarrow{\lambda, i, \mathbb{C}_\mathsf{v}}$ and $\xrightarrow{\mathbb{E}}$ can now test with more arguments. This is a consequence of the fact that an observer can build a bigger continuation from a captured context. For instance, if $\Gamma = (p, \ulcorner E \urcorner, \lambda x.x \triangleleft v)$, then with the LTS of Section 3, we have $\Gamma \xrightarrow{\ulcorner . \urcorner, 2, \mathbb{E}_1[\mathcal{G}_{\Box_1} x.x]} \xrightarrow{\#_{\Box_1} \mathbb{E}_2} \xrightarrow{\lambda, 3, \Box_4} (\Gamma, \ulcorner \mathbb{E}_1[E[\mathbb{E}_2[\Gamma]], \Gamma] \urcorner, \ulcorner \mathbb{E}_1[E[\mathbb{E}_2[\Gamma]], \Gamma] \urcorner \triangleleft v)$. In the new LTS, the first transition is no longer possible, but we can still test the λ-abstraction with the same argument using $\Gamma \xrightarrow{\lambda, 3, \mathbb{E}_1[\star_2[\mathbb{E}_2]]} (\Gamma, \ulcorner \mathbb{E}_1[E[\mathbb{E}_2[\Gamma]], \Gamma] \urcorner \triangleleft v)$.

As explained in Section 4.1, we want to prevent the use of some up-to techniques (like the bisimulation up to related context we introduce in Section 4.3) after some transitions, especially $\xrightarrow{\ulcorner . \urcorner, i, \mathbb{C}_\mathsf{v}}$. To do so, we distinguish the *passive* transitions $\xrightarrow{\ulcorner . \urcorner, i, \mathbb{C}_\mathsf{v}}$, $\xrightarrow{\mathsf{v}}$ from the other ones, called *active*. In a LTS, a visible action $\xrightarrow{\alpha}$ (where $\alpha \neq \tau$) usually corresponds to an interaction with an external observer. The transition $\xrightarrow{\mathsf{v}}$ does not fit that principle; similarly, $\xrightarrow{\ulcorner . \urcorner, i, \mathbb{C}_\mathsf{v}}$ does not correspond exactly to an observer interacting with a continuation, since we throw a value, and not any expression. In contrast, $\xrightarrow{\lambda, i, \mathbb{C}_\mathsf{v}}$ corresponds to function application, $\xrightarrow{\mathbb{E}}$ to context capture, $\xrightarrow{\ulcorner . \urcorner, i, j}$ to continuation decomposition, and $\xrightarrow{\#, i, j}$ to testing prompts equality. This is how we roughly distinguish the former transitions as passive, and the latter ones as active. We then change the definition of progress, to allow a relation $\mathcal{R}$ to progress towards different relations after passive and active transitions.

▶ **Definition 12.** A relation $\mathcal{R}$ diacritically progresses to $\mathcal{S}, \mathcal{T}$ written $\mathcal{R} \rightarrowtail\!\!\!\!\twoheadrightarrow \mathcal{S}, \mathcal{T}$, if $\mathcal{R} \subseteq \mathcal{S}$, $\mathcal{R} \subseteq \mathcal{T}$, and $\Sigma \mathcal{R} \Theta$ implies that

- if $\Sigma \xrightarrow{\alpha} \Sigma'$ and $\xrightarrow{\alpha}$ is passive, then there exists Θ' such that $\Theta \overset{\alpha}{\Rightarrow} \Theta'$ and $\Sigma' \mathcal{S} \Theta'$;
- if $\Sigma \xrightarrow{\alpha} \Sigma'$ and $\xrightarrow{\alpha}$ is active, then there exists Θ' such that $\Theta \overset{\alpha}{\Rightarrow} \Theta'$ and $\Sigma' \mathcal{T} \Theta'$;
- the converse of the above conditions on Θ.

A $\star$-bisimulation is a relation $\mathcal{R}$ such that $\mathcal{R} \rightarrowtail\!\!\!\!\twoheadrightarrow \mathcal{R}, \mathcal{R}$, and $\star$-bisimilarity $\overset{\star}{\approx}$ is the union of all $\star$-bisimulations.

Note that with the same LTS, $\rightarrowtail$ and $\rightarrowtail\!\!\!\!\twoheadrightarrow$ entail the same notions of bisimulation and bisimilarity (but we use a different LTS in this section).

4.3 Up-to Techniques, Soundness, and Completeness

We now discriminate up-to techniques, so that regular up-to techniques cannot be used after passive transitions, while *strong* ones can. An up-to technique (resp. *strong* up-to technique) is a function f such that $\mathcal{R} \rightarrowtail\!\!\!\!\twoheadrightarrow \mathcal{R}, f(\mathcal{R})$ (resp. $\mathcal{R} \rightarrowtail\!\!\!\!\twoheadrightarrow f(\mathcal{R}), f(\mathcal{R})$) implies $\mathcal{R} \subseteq \overset{\star}{\approx}$. We also adapt the notions of evolution and compatibility.

▶ **Definition 13.** A function f evolves to g, h, written $f \rightsquigarrow g, h$, if for all $\mathcal{R} \rightarrowtail\!\!\!\!\twoheadrightarrow \mathcal{R}, \mathcal{T}$, we have $f(\mathcal{R}) \rightarrowtail\!\!\!\!\twoheadrightarrow g(\mathcal{R}), h(\mathcal{T})$.

A function f *strongly* evolves to g, h, written $f \rightsquigarrow_\mathsf{s} g, h$, if for all $\mathcal{R} \rightarrowtail\!\!\!\!\twoheadrightarrow \mathcal{S}, \mathcal{T}$, we have $f(\mathcal{R}) \rightarrowtail\!\!\!\!\twoheadrightarrow g(\mathcal{S}), h(\mathcal{T})$.

Strong evolution is very general, as it uses any relation $\mathcal{R}$, while regular evolution is more restricted, as it relies on relations $\mathcal{R}$ such that $\mathcal{R} \rightarrowtail\!\!\!\!\twoheadrightarrow \mathcal{R}, \mathcal{T}$. But the definition of *diacritical*

compatibility below still allows to use any combinations of strong up-to techniques after a passive transition, even for functions which are not themselves strong. In contrast, regular functions can only be used once after a passive transition of an other regular function.

▶ **Definition 14.** A set F of monotone functions is *diacritically compatible* if there exists $S \subseteq F$ such that

- for all $f \in S$, we have $f \rightsquigarrow_{\mathsf{s}} S^\omega, F^\omega$;
- for all $f \in F$, we have $f \rightsquigarrow S^\omega \circ F \circ S^\omega, F^\omega$.

If S_1 and S_2 are subsets of F which verify the conditions of the definition, then $S_1 \cup S_2$ also does, so there exists the largest subset of F which satisfies the conditions, written $\mathsf{strong}(F)$. This (possibly empty) subset of F corresponds to the strong up-to techniques of F.

▶ **Lemma 15.** *Let F be a compatible set.*

- *If $\mathcal{R} \rightarrowtail\!\!\!\!\twoheadrightarrow \mathsf{strong}(F)^\omega(\mathcal{R}), F^\omega(\mathcal{R})$, then $F^\omega(\mathcal{R})$ is a bisimulation.*
- *If $f \in F$, then f is an up-to technique. If $f \in \mathsf{strong}(F)$, then f is a strong up-to technique.*
- *For all $f \in F$, we have $f(\approx) \subseteq \approx$.*

We now use this framework to define up-to techniques for the $\star$-bisimulation. The definitions of perm and weak are unchanged. We define bisimulation up to related contexts for values rctx and for any term rectx as follows:

$$\frac{\Gamma \mathrel{\mathcal{R}} \Delta}{(\Gamma, \overrightarrow{\mathbb{C}_{\mathsf{v}}}[\Gamma], \mathbb{C}[\Gamma]) \mathrel{\mathsf{rctx}(\mathcal{R})} (\Delta, \overrightarrow{\mathbb{C}_{\mathsf{v}}}[\Delta], \mathbb{C}[\Delta])} \qquad \frac{(\Gamma, e_1) \mathrel{\mathcal{R}} (\Delta, e_2)}{(\Gamma, \overrightarrow{\mathbb{C}_{\mathsf{v}}}[\Gamma], \mathbb{E}[e_1, \Gamma]) \mathrel{\mathsf{rectx}(\mathcal{R})} (\Delta, \overrightarrow{\mathbb{C}_{\mathsf{v}}}[\Delta], \mathbb{E}[e_2, \Delta])}$$

The definitions look similar to the ones of ctx and ectx, but the grammar of multi-hole contexts now include $\star_i$. Besides, we inline strengthening in the definitions of rctx and rectx, allowing Γ, Δ to be extended. This is necessary because, e.g., str and rectx cannot be composed after a passive transition (they are both not strong), so rectx have to include str directly. Note that the behavior of str can be recovered from rectx by taking $\mathbb{E} = \square$.

▶ **Lemma 16.** $F \stackrel{\mathsf{def}}{=} \{\mathsf{perm}, \mathsf{weak}, \mathsf{rctx}, \mathsf{rectx}\}$ *is compatible, with* $\mathsf{strong}(F) = \{\mathsf{perm}, \mathsf{weak}\}$.

As a result, these functions are up-to techniques, and weak and perm can be used after a passive transition. Because of the last item of Lemma 15, $\stackrel{\star}{\approx}$ is also a congruence w.r.t. evaluation contexts, which means that $\stackrel{\star}{\approx}$ is sound w.r.t. $\equiv_E$. We can also prove it is complete the same way as for Theorem 9, leading again to full characterization.

▶ **Theorem 17.** $e_1 \equiv_E e_2$ *iff* $(\emptyset, e_1) \stackrel{\star}{\approx} (\emptyset, e_2)$.

4.4 Examples

We illustrate the use of $\stackrel{\star}{\approx}$, rctx, and rectx with two examples that would be much harder to prove with the techniques of Section 3.

▶ **Example 18** (β_Ω axiom)**.** We prove $(\lambda x.E[x])\ e \stackrel{\star}{\approx} E[e]$ if $x \notin \mathsf{fv}(E)$ and $\mathsf{sp}(E) = \emptyset$. Define $\mathcal{R}$ starting with $(\ulcorner(\lambda x.E[x])\ \square\urcorner) \mathrel{\mathcal{R}} (\ulcorner E\urcorner)$, and closing it under the (#-check) and the following rule:

$$\frac{\Gamma \mathrel{\mathcal{R}} \Delta}{(\Gamma, (\lambda x.E[x])\ \mathbb{C}_{\mathsf{v}}[\Gamma]) \mathrel{\mathcal{R}} (\Delta, E[\mathbb{C}_{\mathsf{v}}[\Delta]])}$$

Then $(\emptyset, (\lambda x.E[x])\ e)$ $\mathsf{weak}(\mathsf{rctx}(\mathcal{R}))$ $(\emptyset, E[e])$ and $\mathcal{R}$ is a bisimulation up to context, since the sequence $\Gamma \xrightarrow{\ulcorner.\urcorner,1,\mathbb{C}_\mathsf{v}} (\Gamma, (\lambda x.E[x])\ \mathbb{C}_\mathsf{v}[\Gamma]) \xrightarrow{\tau} (\Gamma, E[\mathbb{C}_\mathsf{v}[\Gamma]])$ fits $\Delta \xrightarrow{\ulcorner.\urcorner,1,\mathbb{C}_\mathsf{v}} (\Delta, E[\mathbb{C}_\mathsf{v}[\Delta]]) \overset{\tau}{\Rightarrow} (\Delta, E[\mathbb{C}_\mathsf{v}[\Delta]])$, where the final states are in rctx. Notice we use rctx after $\xrightarrow{\tau}$, and not after the passive $\xrightarrow{\ulcorner.\urcorner,1,\mathbb{C}_\mathsf{v}}$ transition.

▶ **Example 19** (Exceptions)**.** A possible way of extending a calculus with exception handling is to add a construct $\mathsf{try}_r\ e\ \mathsf{with}\ v$, which evaluates e with a function raising an exception stored under the variable r. When e calls the function in r with some argument v', even inside another try block, then the computation of e is aborted and replaced by vv'. We can implement this behavior directly in $\lambda_{\mathcal{G}\#}$; more precisely, we write $\mathsf{try}_r\ e\ \mathsf{with}\ v$ as $\mathsf{handle}\ (\lambda r.e)\ v$, where handle is a function expressed in the calculus. One possible implementation of handle in $\lambda_{\mathcal{G}\#}$ is very natural and heavily relies on fresh-prompt generation:

$$\mathsf{handle} \overset{\mathsf{def}}{=} \lambda f.\lambda h.\mathcal{P}x.\#_x f\ (\lambda z.\mathcal{G}_{x\_}.h\ z)$$

The idea is to raise an exception by aborting the current continuation up to the corresponding prompt. The same function can be implemented using any comparable-resource generation and only one prompt p:

$$\mathsf{handle}_p \overset{\mathsf{def}}{=} \lambda f.\lambda h.\mathcal{P}x.(\#_p \mathsf{let}\ r = f\ \mathsf{raise}_{p,x}\ \mathsf{in}\ \lambda\_.\lambda\_.r)\ x\ h$$
$$\mathsf{raise}_{p,x} \overset{\mathsf{def}}{=} \mathsf{fix}\ r(z).\mathcal{G}_{p\_}.\lambda y.\lambda h.\mathsf{if}\ x \overset{?}{=} y\ \mathsf{then}\ h\ z\ \mathsf{else}\ r\ z$$

Here the idea is to keep a freshly generated name x and a handler function h with the prompt corresponding to each call of handle_p. The exception-raising function $\mathsf{raise}_{p,x}$ iteratively aborts the current delimited continuation up to the nearest call of handle_p and checks the name stored there in order to find the corresponding handler. Note that this implementation also uses prompt generation, since it is the only comparable resource that can be dynamically generated in $\lambda_{\mathcal{G}\#}$, but the implementation can be easily translated to, e.g., a calculus with single-prompted delimited-control operators and first-order store.

Proof. We prove that both versions of handle are bisimilar. As in Example 10 we iteratively build a relation $\mathcal{R}$ closed under the ($\#$-check) rule, so that $\mathcal{R}$ is a bisimulation up to context. We start with (handle) $\mathcal{R}$ (handle_p); to match the $\xrightarrow{\lambda,1,\mathbb{C}_\mathsf{v}}$ transition, we extend $\mathcal{R}$ as follows:

$$\frac{\Gamma\ \mathcal{R}\ \Delta}{(\Gamma, \lambda h.\mathcal{P}x.\#_x\mathbb{C}_\mathsf{v}[\Gamma]\ (\lambda z.\mathcal{G}_{x\_}.h\ z))\ \mathcal{R}\ (\Delta, \lambda h.\mathcal{P}x.(\#_p \mathsf{let}\ r = \mathbb{C}_\mathsf{v}[\Delta]\ \mathsf{raise}_{p,x}\ \mathsf{in}\ \lambda\_.\lambda\_.r)\ x\ h)}$$

We obtain two functions which are in turn tested with $\xrightarrow{\lambda,n+1,\mathbb{C}'_\mathsf{v}}$, and we obtain the states

$$(\Gamma, \#_{p_1}\mathbb{C}_\mathsf{v}[\Gamma]\ (\lambda z.\mathcal{G}_{p_1\_}.\mathbb{C}'_\mathsf{v}[\Gamma]\ z)) \text{ and } (\Delta, (\#_p \mathsf{let}\ r = \mathbb{C}_\mathsf{v}[\Delta]\ \mathsf{raise}_{p,p_2}\ \mathsf{in}\ \lambda\_.\lambda\_.r)\ p_2\ \mathbb{C}'_\mathsf{v}[\Delta]).$$

Instead of adding them to $\mathcal{R}$ directly, we decompose them into corresponding parts using up to context (with $\mathbb{C} = \star_{n+1}[\mathbb{C}_\mathsf{v}\ \Box_{n+2}]$), and we add these subterms to $\mathcal{R}$:

$$\frac{\Gamma\ \mathcal{R}\ \Delta \qquad p_1 \notin \#(\Gamma) \qquad p_2 \notin \#(\Delta)}{(\Gamma, \ulcorner \#_{p_1}\Box \urcorner, \lambda z.\mathcal{G}_{p_1\_}.\mathbb{C}'_\mathsf{v}[\Gamma]\ z)\ \mathcal{R}\ (\Delta, \ulcorner (\#_p \mathsf{let}\ r = \Box\ \mathsf{in}\ \lambda\_.\lambda\_.r)\ p_2\ \mathbb{C}'_\mathsf{v}[\Delta] \urcorner, \mathsf{raise}_{p,p_2})} \quad (*)$$

Testing the two captured contexts with $\xrightarrow{\ulcorner.\urcorner,n+1,\mathbb{C}''_\mathsf{v}}$ is easy, because they both evaluate to the thrown value. We now consider $\lambda z.\mathcal{G}_{p_1\_}.\mathbb{C}'_\mathsf{v}[\Gamma]\ z$ and raise_{p,p_2}; after the transition $\xrightarrow{\lambda,n+2,\mathbb{C}_\mathsf{v}}$ we get the two control stuck terms

$$\mathcal{G}_{p_1\_}.\mathbb{C}'_\mathsf{v}[\Gamma]\ \mathbb{C}_\mathsf{v}[\Gamma] \quad \text{and} \quad \mathcal{G}_{p\_}.\lambda y.\lambda h.\mathsf{if}\ p_2 \overset{?}{=} y\ \mathsf{then}\ h\ \mathbb{C}_\mathsf{v}[\Delta]\ \mathsf{else}\ \mathsf{raise}_{p,p_2}\ \mathbb{C}_\mathsf{v}[\Delta]$$

Adding such terms to the relation will not be enough. The first one can be unstuck only using the corresponding context $\ulcorner \#_{p_1} \square \urcorner$, but the second one can be unstuck using any context added by rule $(*)$, even for a different p_2. In such a case, it will consume a part of the context and evaluate to itself. To be more general we add the following rule:

$$\frac{\Gamma \mathrel{\mathcal{R}} \Delta \qquad \mathbb{E}[\mathcal{G}_{p_1 \_}.\mathbb{C}'_{\mathsf{v}}[\Gamma]\ \mathbb{C}_{\mathsf{v}}[\Gamma], \Gamma] \text{ is control-stuck}}{(\Gamma, \mathbb{E}[\mathcal{G}_{p_1 \_}.\mathbb{C}'_{\mathsf{v}}[\Gamma]\ \mathbb{C}_{\mathsf{v}}[\Gamma], \Gamma]) \mathrel{\mathcal{R}} (\Delta, \mathcal{G}_{p \_}.\lambda y.\lambda h.\mathsf{if}\ p_2 \stackrel{?}{=} y\ \mathsf{then}\ h\ \mathbb{C}_{\mathsf{v}}[\Delta]\ \mathsf{else}\ \mathsf{raise}_{p,p_2}\ \mathbb{C}_{\mathsf{v}}[\Delta])}$$

The newly introduced stuck terms are tested with $\xrightarrow{\mathbb{E}'}$; if $\mathbb{E}'$ does not have $\star_i$ surrounding $\square$, they are still stuck, and we can use up to evaluation context to conclude. Assume $\mathbb{E}' = \mathbb{E}_1[\star_i[\mathbb{E}_2]]$ where $\mathbb{E}_2$ has not $\star_j$ around $\square$. If i points to the evaluation context added by $(*)$ for the same p_2, then they both evaluate to terms of the same shape, so we use up to context with $\mathbb{C} = \mathbb{E}_1[\mathbb{C}'_{\mathsf{v}}\ \mathbb{C}_{\mathsf{v}}]$. Otherwise, we know the second program compares two different prompts, so it evaluates to $\mathbb{E}_1[\mathcal{G}_{p \_}.\lambda y.\lambda h.\mathsf{if}\ p_2 \stackrel{?}{=} y\ \mathsf{then}\ h\ \mathbb{C}_{\mathsf{v}}[\Delta]\ \mathsf{else}\ \mathsf{raise}_{p,p_2}\ \mathbb{C}_{\mathsf{v}}[\Delta], \Delta]$ and we use rectx with the last rule. ◀

5 Related Work and Conclusion

Related work. In previous works [5, 6, 7], we defined several bisimilarities for a calculus (called $\lambda_{\mathcal{S}}$) with the (less expressive) delimited-control operators shift and reset. The bisimilarity of Section 3 and the corresponding up-to techniques are close to the ones of [7, Section 3], except that in [7], we do not compare stuck terms using *all* evaluation contexts. However, there is no equivalent of bisimulation up to related contexts in [7], which makes the proof of the β_Ω axiom very difficult in that paper. The proof in Example 18 is as easy as the proof of the β_Ω axiom in [6], but the bisimilarity of [6] is not complete, unlike $\overset{\star}{\approx}$. As a matter of fact, following the developments of Section 4, we believe it is possible to define environmental bisimulations up to related contexts for the $\lambda_{\mathcal{S}}$-calculus.

Environmental bisimilarity has been defined in several calculi with dynamic resource generation, like stores and references [23, 22, 30], information hiding constructs [31, 32], or name creation [3, 26]. In these works, an expression is paired with its generated resources, and behavioral equivalences are defined on these pairs. Our approach is different since we do not carry sets of generated prompts when manipulating expressions (e.g., in the semantic rules of Section 2); instead, we rely on side-conditions and permutations to avoid collisions between prompts. This is possible because all we need to know is if a prompt is known to an outside observer or not, and the correspondences between the public prompts of two related expressions; this can be done through the environment of the bisimilarity. This approach cannot be adapted to more complex generated resources, which are represented by a mapping (e.g., for stores or existential types), but we believe it can be used for name creation in π-calculus [26].

A line of work on program equivalence for which relating evaluation contexts is crucial, as in our work, are logical relations based on the notion of biorthogonality [27]. In particular, this concept has been successfully used to develop techniques for establishing program equivalence in ML-like languages with call/cc [10], and for proving the coherence of control-effect subtyping [8]. Hur et al. combine logical relations and behavioral equivalences in the definition of *parametric bisimulation* [16], where terms are reduced to normal forms that are then decomposed into subterms related by logical relations. This framework has been extended to abortive control in [17], where *stuttering* is used to allow terms not to reduce for a finite amount of time when comparing them in a bisimulation proof. This is reminiscent

of our distinction between active and passive transitions, as passive transitions can be seen as "not reducing", but there is still some testing involved in these transitions. Besides, the concern is different, since the active/passive distinction prevents the use of up-to techniques, while stuttering has been proposed to improve plain parametric bisimulations.

Conclusion and future work. We have developed a behavioral theory for Dybvig et al.'s calculus of multi-prompted delimited control, where the enabling technology for proving program equivalence are environmental bisimulations, presented in Madiot's style. The obtained results generalize our previous work in that they account for multiple prompts and local visibility of dynamically generated prompts. Moreover, the results of Section 4 considerably enhance reasoning about captured contexts by treating them as first-class objects at the level of bisimulation proofs (thanks to the construct $\star_i$) and not only at the level of terms. The resulting notion of bisimulation up to related contexts improves on the existing bisimulation up to context in presence of control operators, as we can see when comparing Example 18 to the proof of the same result in [7]. We believe bisimulation up to related contexts could be useful for constructs akin to control operators, like passivation in π-calculus [26]. The soundness of this up-to technique has been proved in an extension of Madiot's framework; we plan to investigate further this extension, to see how useful it could be in defining up-to techniques for other languages. Finally, it may be possible to apply the tools developed in this paper to [20], where a single-prompted calculus is translated into a multi-prompted one, but no operational correspondence is given to guarantee the soundness of the translation.

Acknowledgements. We would like to thank Jean-Marie Madiot for the insightful discussions about his work, and Małgorzata Biernacka, Klara Zielińska, and the anonymous reviewers for the helpful comments on the presentation of this work.

References

1 Andrés Aristizábal, Dariusz Biernacki, Sergueï Lenglet, and Piotr Polesiuk. Environmental bisimulations for delimited-control operators with dynamic prompt generation. Research Report RR-8905, Inria, Nancy, France, April 2016. Available at `http://hal.inria.fr/hal-01305137`.

2 Vincent Balat, Roberto Di Cosmo, and Marcelo P. Fiore. Extensional normalisation and type-directed partial evaluation for typed lambda calculus with sums. In Xavier Leroy, editor, *Proceedings of the Thirty-First Annual ACM Symposium on Principles of Programming Languages*, pages 64–76, Venice, Italy, January 2004. ACM Press.

3 Nick Benton and Vasileios Koutavas. A mechanized bisimulation for the nu-calculus. Technical Report MSR-TR-2008-129, Microsoft Research, September 2008.

4 Dariusz Biernacki and Olivier Danvy. A simple proof of a folklore theorem about delimited control. *Journal of Functional Programming*, 16(3):269–280, 2006.

5 Dariusz Biernacki and Sergueï Lenglet. Applicative bisimulations for delimited-control operators. In Lars Birkedal, editor, *Foundations of Software Science and Computation Structures, 15th International Conference (FOSSACS'12)*, volume 7213 of *Lecture Notes in Computer Science*, pages 119–134, Tallinn, Estonia, March 2012. Springer.

6 Dariusz Biernacki and Sergueï Lenglet. Normal form bisimulations for delimited-control operators. In Tom Schrijvers and Peter Thiemann, editors, *Functional and Logic Programming, 13th International Symposium (FLOPS'12)*, volume 7294 of *Lecture Notes in Computer Science*, pages 47–61, Kobe, Japan, May 2012. Springer.

7 Dariusz Biernacki and Sergueï Lenglet. Environmental bisimulations for delimited-control operators. In Chung-chieh Shan, editor, *Proceedings of the 11th Asian Symposium on Programming Languages and Systems (APLAS'13)*, volume 8301 of *Lecture Notes in Computer Science*, pages 333–348, Melbourne, VIC, Australia, December 2013. Springer.

8 Dariusz Biernacki and Piotr Polesiuk. Logical relations for coherence of effect subtyping. In Thorsten Altenkirch, editor, *Typed Lambda Calculi and Applications, 13th International Conference, TLCA 2015*, volume 38 of *Leibniz International Proceedings in Informatics*, pages 107–122, Warsaw, Poland, July 2015. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

9 Olivier Danvy and Andrzej Filinski. Abstracting control. In Mitchell Wand, editor, *Proceedings of the 1990 ACM Conference on Lisp and Functional Programming*, pages 151–160, Nice, France, June 1990. ACM Press.

10 Derek Dreyer, Georg Neis, and Lars Birkedal. The impact of higher-order state and control effects on local relational reasoning. *Journal of Functional Programming*, 22(4-5):477–528, 2012.

11 R. Kent Dybvig, Simon Peyton-Jones, and Amr Sabry. A monadic framework for delimited continuations. *Journal of Functional Programming*, 17(6):687–730, 2007.

12 Matthias Felleisen. The theory and practice of first-class prompts. In Jeanne Ferrante and Peter Mager, editors, *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Programming Languages*, pages 180–190, San Diego, California, January 1988. ACM Press.

13 Andrzej Filinski. Representing monads. In Hans-J. Boehm, editor, *Proceedings of the Twenty-First Annual ACM Symposium on Principles of Programming Languages*, pages 446–457, Portland, Oregon, January 1994. ACM Press.

14 Matthew Flatt, Gang Yu, Robert Bruce Findler, and Matthias Felleisen. Adding delimited and composable control to a production programming environment. In Norman Ramsey, editor, *Proceedings of the 2007 ACM SIGPLAN International Conference on Functional Programming (ICFP'07)*, pages 165–176, Freiburg, Germany, September 2007. ACM Press.

15 Carl Gunter, Didier Rémy, and Jon G. Riecke. A generalization of exceptions and control in ML-like languages. In Simon Peyton Jones, editor, *Proceedings of the Seventh ACM Conference on Functional Programming and Computer Architecture*, pages 12–23, La Jolla, California, June 1995. ACM Press.

16 Chung-Kil Hur, Derek Dreyer, Georg Neis, and Viktor Vafeiadis. The marriage of bisimulations and Kripke logical relations. In John Field and Michael Hicks, editors, *Proceedings of the Thirty-Ninth Annual ACM Symposium on Principles of Programming Languages*, pages 59–72, Philadelphia, PA, USA, January 2012. ACM Press.

17 Chung-Kil Hur, Georg Neis, Derek Dreyer, and Viktor Vafeiadis. A logical step forward in parametric bisimulations. Technical Report MPI-SWS-2014-003, Max Planck Institute for Software Systems (MPI-SWS), Saarbrücken, Germany, January 2014.

18 Yukiyoshi Kameyama and Masahito Hasegawa. A sound and complete axiomatization of delimited continuations. In Olin Shivers, editor, *Proceedings of the 2003 ACM SIGPLAN International Conference on Functional Programming (ICFP'03)*, pages 177–188, Uppsala, Sweden, August 2003. ACM Press.

19 Oleg Kiselyov. Delimited control in OCaml, abstractly and concretely: System description. In Matthias Blume and German Vidal, editors, *Functional and Logic Programming, 10th International Symposium, FLOPS 2010*, volume 6009 of *Lecture Notes in Computer Science*, pages 304–320, Sendai, Japan, April 2010. Springer.

20 Ikuo Kobori, Yukiyoshi Kameyama, and Oleg Kiselyov. ATM without tears: prompt-passing style transformation for typed delimited-control operators. In Olivier Danvy, editor, *2015 Workshop on Continuations: Pre-proceedings*, London, UK, April 2015.

21 Vasileios Koutavas, Paul Blain Levy, and Eijiro Sumii. From applicative to environmental bisimulation. In Michael Mislove and Joël Ouaknine, editors, *Proceedings of the 27th Annual Conference on Mathematical Foundations of Programming Semantics (MFPS XXVII)*, volume 276 of *Electronic Notes in Theoretical Computer Science*, pages 215–235, Pittsburgh, PA, USA, May 2011.

22 Vasileios Koutavas and Mitchell Wand. Bisimulations for untyped imperative objects. In Peter Sestoft, editor, *15th European Symposium on Programming, ESOP 2006*, volume 3924 of *Lecture Notes in Computer Science*, pages 146–161, Vienna, Austria, March 2006. Springer.

23 Vasileios Koutavas and Mitchell Wand. Small bisimulations for reasoning about higher-order imperative programs. In J. Gregory Morrisett and Simon L. Peyton Jones, editors, *Proceedings of the 33rd Annual ACM Symposium on Principles of Programming Languages*, pages 141–152, Charleston, SC, USA, January 2006. ACM Press.

24 Jean-Marie Madiot. *Higher-Order Languages: Dualities and Bisimulation Enhancements*. PhD thesis, Université de Lyon and Università di Bologna, 2015.

25 Jean-Marie Madiot, Damien Pous, and Davide Sangiorgi. Bisimulations up-to: Beyond first-order transition systems. In Paolo Baldan and Daniele Gorla, editors, *25th International Conference on Concurrency Theory*, volume 8704 of *Lecture Notes in Computer Science*, pages 93–108, Rome, Italy, September 2014. Springer.

26 Adrien Piérard and Eijiro Sumii. A higher-order distributed calculus with name creation. In *Proceedings of the 27th IEEE Symposium on Logic in Computer Science (LICS 2012)*, pages 531–540, Dubrovnik, Croatia, June 2012. IEEE Computer Society Press.

27 Andrew Pitts and Ian Stark. Operational reasoning for functions with local state. In Andrew Gordon and Andrew Pitts, editors, *Higher Order Operational Techniques in Semantics*, pages 227–273. Publications of the Newton Institute, Cambridge University Press, 1998.

28 Damien Pous and Davide Sangiorgi. Enhancements of the bisimulation proof method. In Davide Sangiorgi and Jan Rutten, editors, *Advanced Topics in Bisimulation and Coinduction*, chapter 6, pages 233–289. Cambridge University Press, 2011.

29 Davide Sangiorgi, Naoki Kobayashi, and Eijiro Sumii. Environmental bisimulations for higher-order languages. *ACM Transactions on Programming Languages and Systems*, 33(1):1–69, January 2011.

30 Eijiro Sumii. A complete characterization of observational equivalence in polymorphic lambda-calculus with general references. In Erich Grädel and Reinhard Kahle, editors, *Computer Science Logic, CSL'09*, volume 5771 of *Lecture Notes in Computer Science*, pages 455–469, Coimbra, Portugal, September 2009. Springer.

31 Eijiro Sumii and Benjamin C. Pierce. A bisimulation for dynamic sealing. *Theoretical Computer Science*, 375(1-3):169–192, 2007.

32 Eijiro Sumii and Benjamin C. Pierce. A bisimulation for type abstraction and recursion. *Journal of the ACM*, 54(5), 2007.