The Higher-Order, Call-by-Value Applied Pi-Calculus*

Nobuyuki Sato and Eijiro Sumii
Tohoku University
{nsato,sumii}@kb.ecei.tohoku.ac.jp

Abstract. We define a higher-order process calculus with algebraic operations such as encryption and decryption, and develop a bisimulation proof method for behavioral equivalence in this calculus. Such development has been notoriously difficult because of the subtle interactions among generative names, processes as data, and the algebraic operations. We handle them by carefully defining the calculus and adopting Sumii et al.'s environmental bisimulation, and thereby give (to our knowledge) the first "useful" proof method in this setting. We demonstrate the utility of our method through examples involving both higher-order processes and asymmetric cryptography.

1 Introduction

Higher-order communication and encryption. The combination of cryptographic operations and higher-order, concurrent programs is ubiquitous in modern computer systems. For instance, software distribution systems (such as Windows Update) usually employ some digital signature scheme to verify the authenticity of the downloaded programs before installing them. For another example, Web-based e-mail user agents (such as Gmail) often distribute complex code (typically in HTML and JavaScript) interpreted at the client side, where the code itself is transferred through a secure channel, as well as the messages sent and received by the code. Guaranteeing the security of such systems is even more important than in first-order programs, because of the higher chance of "accidentally" executing arbitrary, malicious code.

Process calculi such as CCS and π-calculus have been useful for the verification of concurrent systems in general. In particular, spi-calculus [2] and applied π-calculus [1] are equipped with cryptographic operations such as encryption and decryption, and can be used for formal reasoning about cryptographic protocols. On the other hand, higher-order π-calculus [7] allows communication of processes themselves, and is able to model systems that transfer programs.

To our knowledge, however, there has been little research[1] on process calculus with *both* higher-order communication and cryptographic operations, probably because their combination is highly non-trivial. For instance, consider a process $P = \overline{c}\langle Q\rangle$ that sends another process $Q = \overline{c}\langle encrypt(m, k)\rangle$ to a public communication channel c. The process Q itself, when executed, sends message m encrypted under a secret key k. Now, is it possible for an observer on c to obtain m by intercepting the communications? One might say no, because k is secret. Another might disagree, because the observer can analyze the program text of Q and extract k from it. Yet another one might argue that such an analysis is impossible, because m is encrypted *before* Q is published on the

* Janury 23, 2009. Last revised on October 14, 2009. Detailed proofs are available online [12].

[1] An exception is a type system for higher-order spi-calculus [6], but it does not consider general algebra, decomposition, behavioral equivalence, nor bisimulations.

network. But what if $Q = c(x).\bar{c}\langle encrypt(x,k)\rangle$ instead? How about $Q = c(x).\bar{c}\langle encrypt(m,k)\rangle$ when m is independent of x?

The above gedankenexperiment leads us to our first observation that, unlike in applied π-calculus, the *values* of function applications must be explicitly distinguished from the function applications themselves in this setting. Thus, let us write $\hat{f}(V_1,\ldots,V_l)$ for the values of function applications $f(V_1,\ldots,V_l)$. In the last example, for instance, k (and m) can be extracted if $Q = c(x).\bar{c}\langle encrypt(m,k)\rangle$, but they cannot if $Q = c(x).\bar{c}\langle \widehat{encrypt}(m,k)\rangle$

Accordingly, we need to provide a construct to decompose the syntax of communicated terms (but not values) including communicated processes (but not *running* processes), so that an observer can analyze them. For this purpose we introduce operations of the form $match\ M\ as\ x\ in\ R$, which decompose the syntax (not value) of M and bind x to the tuple of the decomposed elements. The point is that, if M is already a value, like $\widehat{encrypt}(m,k)$, then it cannot be decomposed any further.

To make our theory realistic, we require that first-order terms are evaluated before they are sent to the network. Our calculus is thus "call-by-value." As usual, however, call-by-name computation can easily be encoded by means of thunks (which are straightforward to implement as processes).

Behavioural equivalence and bisimulations. The distinction between already computed values and yet-to-be-computed terms is crucial but not sufficient for our development. Specifically, we need a method for proving properties of processes. Traditionally, *behavioral equivalence* and *bisimulations* have been known to be useful for specifying and proving many interesting properties of concurrent systems, including security properties such as secrecy and authenticity.

However, traditional bisimulation proof methods for π-calculi are not of help here. Context bisimulation [7] is not useful by itself as a practical proof technique, because of the universal quantification over all receiver (and sender) contexts. Normal bisimulation [7] essentially encodes higher-order processes into the first order by passing pointers only, and therefore would not be sound under the presence of decomposition operation like ours.[2] Environment-sensitive bisimulations in spi-calculus (see [5] for example) are not applicable in our higher-order language, because the environment itself would include processes.

For these reasons, we adapt more recent work on *environmental bisimulation* [9, 14, 15] and extend it to account for the decomposition operation as well as the algebraic operations (which generalize various cryptographic operations, as in applied π-calculus). Although environmental bisimulations have previously been applied to λ-calculus with encryption [14] and to higher-order π-calculus [9], our extension is far from trivial: to formalize decomposition, we need to introduce quotations (as in Lisp) for terms as well

[2] In general, fully abstract (i.e., equivalence-preserving) encoding of our calculus into another would be extremely non-trivial. This includes an "obvious" translation from higher-order processes into the first order, where one communicates first-order terms *representing* the syntax of processes and runs a process to *interpret* them. To prove it correct, one must anyway define a higher-order calculus and then prove the translation to be fully abstract, which is more indirect *and* requires more work than the present approach.

as for processes, which requires careful definition of several kinds of contexts and context closure operations. Specification of the algebra also requires careful generalization of the conditions on terms in previous environmental bisimulations.

Our contributions in the present paper are thus twofold: the definition of the calculus itself, and the environmental bisimulation proof method for this calculus.

Overview of the environmental bisimulation. Our environmental bisimulation $\mathcal{X}$ is a set of triples of the form $(\mathcal{E}, P, Q)$, where P and Q are the tested processes and $\mathcal{E}$ is the environment, i.e., a binary relation on terms, representing the observer's knowledge. The membership $(\mathcal{E}, P, Q) \in \mathcal{X}$, which is often written $P\mathcal{X}_{\mathcal{E}}Q$ for readability, means that processes P and Q are bisimilar under environment $\mathcal{E}$. There are several conditions on $\mathcal{X}$, each corresponding to a change of the state of the observer and the processes. For instance, as in traditional (weak) bisimulations, if either P or Q makes an internal transition, then the other should make 0 or more internal transitions, and the resulting processes should also be bisimilar (under the same environment $\mathcal{E}$, because the observer's state has not changed). For output actions, if P sends a value V and becomes P', then Q should also send some value W and become Q', with the requirement that P' and Q' are bisimilar under the environment $\mathcal{E} \cup \{(V, W)\}$, which is extended with the values the observer has learned.

For input, we must consider any pair of values that can be synthesized by the attacker from its knowledge $\mathcal{E}$. We use $(\hat{\mathcal{E}})^*$ for the set of such value pairs, where $\hat{\mathcal{E}}$ is the set of pairs of values that can be obtained from $\mathcal{E}$ by first-order computation, and $(\hat{\mathcal{E}})^*$ is the context closure of $\hat{\mathcal{E}}$. Roughly, we define:

$$\begin{aligned}\hat{\mathcal{E}} &= \{\ (eval(D[\tilde{V}]),\ eval(D[\tilde{W}]))\ \mid\ \tilde{V}\mathcal{E}\tilde{W},\ \mathsf{fn}(D) = \emptyset,\ D \text{ is first-order}\ \}\\ \mathcal{E}^* &= \{\ (C[\tilde{V}],\ C[\tilde{W}])\ \mid\ \tilde{V}\mathcal{E}\tilde{W},\ \mathsf{fn}(C) = \emptyset\ \}\end{aligned}$$

(Here, $\tilde{V}$ denotes a sequence $V_1, \ldots, V_l$, and $\tilde{V}\mathcal{E}\tilde{W}$ denotes $V_i\mathcal{E}W_i$ for all i. We use similar notations for various kinds of meta-variables throughout the paper.) Recall that, unlike in previous environmental bisimulations with "built-in" conditions for some particular algebra (e.g., [14]), we need to consider general algebras. $\hat{\mathcal{E}}$ accounts for the synthesis of knowledge within such algebras.

For instance, let $decrypt(encrypt(x, y), y) = x$. If the ciphertexts $(encrypt(V, k),$ $encrypt(W, k))$ and the key pair (k, k) belong to $\mathcal{E}$, then the plaintexts (V, W) belong to $\hat{\mathcal{E}}$. This is because the first-order observer context $D = decrypt([]_1, []_2)$ can compute them by putting the ciphertexts into its first hole $[]_1$ and the key to $[]_2$, like:

$$\begin{aligned}D[encrypt(V, k), k] &= decrypt(encrypt(V, k), k) = V\\ D[encrypt(W, k), k] &= decrypt(encrypt(W, k), k) = W\end{aligned}$$

Thus, the bisimulation condition for input would be: for any $V(\hat{\mathcal{E}})^*W$, if P receives V and becomes P', then Q receives W and becomes Q', with P' and Q' bisimilar again under environment $\mathcal{E}$.

Furthermore, the observer can spawn arbitrary new processes from its knowledge $\mathcal{E}$. Thus, we also require $P|P'\mathcal{X}_{\mathcal{E}}Q|Q'$ for any $P\mathcal{X}_{\mathcal{E}}Q$ and $P'\hat{\mathcal{E}}Q'$. This may seem to be a heavy condition because of the universal quantification over processes P' and Q', drawn from $\hat{\mathcal{E}}$. However, we in fact work out an up-to context technique, where the requirement is weakened to $P|P'\mathcal{X}_{\mathcal{E}}^{(*)}Q|Q'$ for a certain form of context closure $\mathcal{X}_{\mathcal{E}}^{(*)}$

for $\mathcal{X}$. This essentially removes the universal quantification and significantly lightens the burden of a bisimulation proof in higher-order process calculus.[3] (Another subtle but important trick here is that, unlike for input, $\hat{\mathcal{E}}$ suffices in place of $(\hat{\mathcal{E}})^*$. Informally, this is because processes in $(\hat{\mathcal{E}})^*$ can only make the same observations as those in $\hat{\mathcal{E}}$.)

Finally, for decomposition of processes and terms, we require $P\mathcal{X}_{\mathcal{E}\cup\{(M',N')\}}Q$ for any $P\mathcal{X}_{\mathcal{E}}Q$ and $M\hat{\mathcal{E}}N$, where M' and N' are the result of decomposing M and N, respectively. (Obviously, this $\hat{\mathcal{E}}$ does not have to be $(\hat{\mathcal{E}})^*$, because there is no point in synthesizing a term and then decomposing it.) Again, this condition may seem heavy because, by repeatedly applying it, we need to transitively include all the subterms of M and N. As in the previous case, however, most of them can be removed by the up-to context technique.

Overview of the paper. The rest of this paper is structured as follows. Section 2 formally presents the syntax and labeled transition semantics of our calculus, which is (formally) parametrized by the semantics of terms. Sections 3 and 4 define the environmental bisimulation and the up-to context technique. Section 5 proves their soundness and completeness with respect to reduction-closed barbed equivalence. Section 6 gives examples and Section 7 concludes.

Throughout the paper, readers are assumed to be familiar with standard technical developments in the π-calculus [11] and be comfortable with basic mathematical notions such as inductive (and coinductive) definitions of sets (and relations).

2 The calculus

2.1 Syntax

As in applied π-calculus [1], our language consists of terms and processes. Terms represent channel names and communicated data. Processes represent running programs. The set of terms is defined as follows:

$$\begin{array}{rcll} M & ::= & x \quad \text{(variable)} \quad | \quad a \quad \text{(name)} \quad | \quad f \quad \text{(function)} \\ & | & \text{`}P \quad \text{(quoted process)} \quad | \quad \text{`}M \quad \text{(quoted term)} \\ & | & M(M_1, \ldots, M_l) \qquad \text{(uncomputed application)} \\ & | & \hat{f}(V_1, \ldots, V_l) \qquad \text{(computed application)} \end{array}$$

Meta-variables M, N range over terms, a, b, c, d, k, n over names, x, y over variables and f, g over functions. Term $M(M_1, \ldots, M_l)$ represents function application that is yet to be computed. Conversely, $\hat{f}(V_1, \ldots, V_l)$ represents function application that is already computed, where function f of arity l has been applied to values $V_1, \ldots, V_l$. Note that function symbols f are first-class but different from names (or variables) and therefore cannot be bound. Term $\text{`}P$ represents the syntax of processes, which allows us to communicate terms containing processes (i.e, higher-order terms). Although it has been written just P in previous work (e.g., [7, 9]) and in the introduction, we here put the quotation mark to clarify the distinction between communicated and running

[3] This was previously not possible and therefore is yet another technical contribution of the present work. See footnote 4 in the next section for details.

processes.[4] Term $`M$ represents the syntax of terms themselves. It is necessary for the decomposition operation explained below.

Simultaneously, we define a subset of terms as values, i.e., results of computation:

$$V ::= a \mid f \mid `P \mid `M \mid \hat{f}(V_1, \ldots, V_l)$$

Meta-variables V, W range over values. We write $\mathbf{Quo}$ for the set of values of the form $`P$ or $`M$.

The set of processes is defined by:

$$\begin{array}{rcll} P & ::= & 0 \quad \text{(nil)} \quad \mid \quad run(M) & \text{(execution)} \\ & \mid & M(x).P \quad \text{(input)} \quad \mid \quad \overline{M}\langle N\rangle.P & \text{(output)} \\ & \mid & !P \quad \text{(replication)} \quad \mid \quad \nu a.P & \text{(restriction)} \\ & \mid & (P|Q) & \text{(parallel composition)} \\ & \mid & \mathit{if}\ M = N\ \mathit{then}\ P\ \mathit{else}\ Q & \text{(conditional)} \\ & \mid & \mathit{match}\ M\ \mathit{as}\ x\ \mathit{in}\ P & \text{(decomposition)} \end{array}$$

P, Q, R range over processes. Their informal semantics is as follows. Process 0 does nothing. Process $run(M)$ executes quoted processes (i.e., $`P$). Parallel composition $P|Q$ represents concurrent execution of P and Q. Replication $!P$ executes as many copies of P as necessary in parallel. Restriction $\nu a.P$ creates a new name a and then becomes P. Conditional $\mathit{if}\ M = N\ \mathit{then}\ P\ \mathit{else}\ Q$ compares the values of M and N (up to α-equivalence, because they may contain processes), and executes either P or Q accordingly. Input $M(x).P$ receives a value and output $\overline{M}\langle N\rangle.P$ sends the value of N on channel M, before becoming P. Process $\mathit{match}\ M\ \mathit{as}\ x\ \mathit{in}\ P$ decomposes the value of M (which should be either $`P$ or $`N$), binds x to the decomposed elements, and executes P. Formal semantics of processes will be given in the next subsection.

As usual, we identify processes (and terms containing processes) up to α-conversion. We write $\mathsf{fn}(M)$ and $\mathsf{fn}(P)$ for the set of free names that appear in M and P, respectively. We often omit trailing 0.

Contexts and context closure. Because we have terms, values and processes in our language, we correspondingly define term contexts, value contexts and process contexts. They have multiple holes (indexed by positive integers $1, 2, \ldots$) for values.

$$\begin{array}{l} C_t ::= x \mid C_v \mid C_t(C_t, \ldots, C_t) \\ C_v ::= []_i \mid a \mid f \mid `C_p \mid `C_t \mid \hat{f}(C_v, \ldots, C_v) \\ C_p ::= 0 \mid run(C_t) \mid C_t(x).C_p \mid \overline{C_t}\langle C_t\rangle.C_p \mid !C_p \mid \nu a.C_p \mid (C_p|C_p) \mid \\ \qquad \mathit{if}\ C_t = C_t\ \mathit{then}\ C_p\ \mathit{else}\ C_p \mid \mathit{match}\ C_t\ \mathit{as}\ x\ \mathit{in}\ C_p \end{array}$$

We write C for any of the contexts above, and $\mathsf{bn}(C)$ for the set of names bound in C. As usual, contexts (unlike processes) are *not* identified by α-conversion in general, e.g., $\nu m.\overline{a}\langle []_1\rangle.0 \neq \nu n.\overline{a}\langle []_1\rangle.0$ so $\mathsf{bn}(\nu m.\overline{a}\langle []_1\rangle.0) = \{m\} \neq \{n\} = \mathsf{bn}(\nu n.\overline{a}\langle []_1\rangle.0)$.

[4] This distinction permits a more convenient up-to context technique (clause 6 in Definition 3) when the observer spawns new processes synthesized from its knowledge, because (unlike in traditional higher-order π-calculus [7]) the execution of a process now requires an internal transition step $run(`P) \xrightarrow{\tau} P$. This was not the case in previous work [9] on environmental bisimulation for higher-order π-calculus (with a limited version of up-to context [8, Definition E.1]), which often forced one to construct a significantly larger $\mathcal{X}$ than necessary in their bisimulation proof.

Since we are interested in behavioural equivalence of processes under contexts, we define context closure operations as follows. Let $\mathcal{E}$ be a (binary) relation on closed values. (As is often the case in π-calculi [11], "closed" in this paper means the lack of free *variables* only. Free *names* are still possible.) Relation $\mathcal{E}^*$ on closed terms is:

$$\{\ (C_t[\tilde{V}], C_t[\tilde{W}])\ \mid\ \tilde{V}\mathcal{E}\tilde{W},\ \mathsf{bn}(C_t) \cap \mathsf{fn}(\tilde{V}, \tilde{W}) = \mathsf{fn}(C_t) = \emptyset\ \}$$

We sometimes (ab)use $\mathcal{E}^*$ as a relation on closed processes, in which case it denotes:

$$\{\ (C_p[\tilde{V}], C_p[\tilde{W}])\ \mid\ \tilde{V}\mathcal{E}\tilde{W},\ \mathsf{bn}(C_p) \cap \mathsf{fn}(\tilde{V}, \tilde{W}) = \mathsf{fn}(C_p) = \emptyset\ \}$$

In the definitions above, $\mathsf{fn}(C_t)$ and $\mathsf{fn}(C_p)$ are required to be empty so that context cannot "guess" secret names just by chance. These conditions could be $\mathsf{fn}(C_t) \cap \mathsf{fn}(\tilde{V}, \tilde{W}) = \emptyset$ and $\mathsf{fn}(C_p) \cap \mathsf{fn}(\tilde{V}, \tilde{W}) = \emptyset$, instead of $\mathsf{fn}(C_t) = \emptyset$ and $\mathsf{fn}(C_p) = \emptyset$, but we preferred the latter for the sake of simplicity. This choice does *not* restrict observations made by contexts: one can put arbitrary free names into the holes of the contexts by including them in $\mathcal{E}$ whenever necessary. Note also that contexts can create as many fresh names as needed for observations, because $\mathsf{bn}(C_t)$ and $\mathsf{bn}(C_p)$ are *not* required to be empty, though they should again be distinct from other free names as usual.

As already stated, our calculus is parametrized by the semantics of terms. To formalize our assumptions on these semantics, we define *first-order* contexts, i.e., contexts with no quotation (and no names).

$$D_t ::= D_v \mid D_t(D_t, \ldots, D_t) \qquad D_v ::= []_i \mid f \mid \hat{f}(D_v, \ldots, D_v)$$

By using first-order contexts, we define another kind of context closure $\hat{\mathcal{E}}$ as follows. Let $\mathcal{E}$ be a relation on closed values. Then, relation $\hat{\mathcal{E}}$ is defined to be:

$$\{\ (eval(D_t[\tilde{V}]), eval(D_t[\tilde{W}]))\ \mid\ \tilde{V}\mathcal{E}\tilde{W}\ \}$$

The function $eval$ will be defined in the next subsection. Intuitively, $\hat{\mathcal{E}}$ is the set of (pairs of) values that can be computed from $\mathcal{E}$ only at the first order, i.e., without using quotation or processes. Note that $\mathsf{bn}(D_t) = \mathsf{fn}(D_t) = \emptyset$ by definition.

2.2 Semantics

Semantics of terms. We require that the meaning of terms is formally defined by a rewriting system [3] (cf. [4, Section 5], though their formulation is slighly different from ours) on closed terms, and that the system is confluent and strongly normalizing for ground terms. An example representing asymmetric cryptography is given in Section 6. Readers are referred to a standard textbook [3] for basic definitions in term rewriting.

In the system, we also assume tuples (and projection operations for them) and constant (i.e., nullary function) symbols `name`, `fun`, ... (and equality tests on them) to represent the syntax of processes. Recall that (function and) constant symbols are different from names.

The partial function $eval$ returns the value of a given term. It is undefined if the normal form of the term does not belong to the set of values defined in the previous subsection. For example, $eval(\#_1(a, b)) = a$ and $eval(\#_2(c))$ is undefined.

Finally, we require that $M(\hat{\mathcal{E}})^*N$ implies $eval(M)(\hat{\mathcal{E}})^* eval(N)$. This requirement is critical (and sufficient) throughout our developments. It means that the *values* of (pairs of) terms synthesized from $\hat{\mathcal{E}}$ can be synthesized from $\hat{\mathcal{E}}$ itself. That is, $eval$ does not introduce any new names or higher-order values. Recall that $\hat{\mathcal{E}}$ is a closure (and evaluation) under nameless and first-order contexts only.

Semantics of processes. We define the semantics of processes by a labeled transition system. The labels have three forms: τ, $a(V)$, and $\nu\tilde{c}.\overline{a}\langle V\rangle$, representing the silent action, an input action, and an output action, respectively. Metavariable α ranges over labels. $\mathsf{bn}(\alpha)$ is defined as $\mathsf{bn}(\nu\tilde{c}.\overline{a}\langle V\rangle) = \{\tilde{c}\}$ and $\mathsf{bn}(\tau) = \mathsf{bn}(a(V)) = \emptyset$. The transitions are defined by the rules below, with symmetric rules (Par-R) and (Tau-R) omitted. We write $\Rightarrow$ for the reflexive and transitive closure of $\xrightarrow{\tau}$.

$$\frac{eval(M) = a}{M(x).P \xrightarrow{a(V)} \{^{V}/_{x}\}P}\ \text{(In)} \qquad \frac{eval(M) = a}{\overline{M}\langle N\rangle.P \xrightarrow{\overline{a}\langle eval(N)\rangle} P}\ \text{(Out)}$$

$$\frac{P \xrightarrow{\alpha} P' \quad \mathsf{bn}(\alpha) \cap \mathsf{fn}(Q) = \emptyset}{P|Q \xrightarrow{\alpha} P'|Q}\ \text{(Par-L)}$$

$$\frac{P \xrightarrow{\nu\tilde{b}.\overline{a}\langle V\rangle} P' \quad Q \xrightarrow{a(V)} Q' \quad \{\tilde{b}\} \cap \mathsf{fn}(Q) = \emptyset}{P|Q \xrightarrow{\tau} \nu\tilde{b}.(P'|Q')}\ \text{(Tau-L)}$$

$$\frac{P|!P \xrightarrow{\alpha} Q}{!P \xrightarrow{\alpha} Q}\ \text{(Rep)} \qquad \frac{P \xrightarrow{\alpha} P' \quad a \notin \mathsf{bn}(\alpha) \cup \mathsf{fn}(\alpha)}{\nu a.P \xrightarrow{\alpha} \nu a.P'}\ \text{(Scope)}$$

$$\frac{P \xrightarrow{\nu\tilde{b}.\overline{a}\langle V\rangle} P' \quad c \neq a \quad c \in \mathsf{fn}(V) \setminus \{\tilde{b}\}}{\nu c.P \xrightarrow{\nu\tilde{b},c.\overline{a}\langle V\rangle} P'}\ \text{(Open)} \qquad \frac{eval(M) = {}`P}{run(M) \xrightarrow{\tau} P}\ \text{(Run)}$$

$$\frac{eval(M) = eval(N)}{\mathit{if}\ M = N\ \mathit{then}\ P\ \mathit{else}\ Q \xrightarrow{\tau} P}\ \text{(IfTrue)} \qquad \frac{eval(M) \neq eval(N)}{\mathit{if}\ M = N\ \mathit{then}\ P\ \mathit{else}\ Q \xrightarrow{\tau} Q}\ \text{(IfFalse)}$$

$$\frac{eval(M) = V \quad V \in \mathbf{Quo} \quad n \notin \mathsf{fn}(V, P)}{\mathit{match}\ M\ \mathit{as}\ x\ \mathit{in}\ P \xrightarrow{\tau} \nu n.\{^{reify_n(V)}/_{x}\}P}\ \text{(Match)}$$

Most of the rules are straightforward adaptation of standard labelled transition in the π-calculus [11]. As usual in untyped small-step operational semantics, transition gets stuck if the assumptions are not satisfied, e.g., if $eval(M)$ is not a name in rules (In) and (Out). In rule (Match), the operator $reify_n$ takes a quoted process or a quoted term and decomposes it into a tuple. (The name n is used for substituting a bound name or a bound variable, if there is any, in the reified process.) Formally, it is defined as:

$$\begin{array}{llll}
reify_n({`}0) & = (\widehat{\texttt{zero}}) & reify_n({`}run(M)) & = (\widehat{\texttt{exe}}, {`}M) \\
reify_n({`}(M(x).P)) & = (\widehat{\texttt{in}}, {`}M, n, {`}\{^{n}/_{x}\}P) & reify_n({`}(\overline{M_1}\langle M_2\rangle.P)) & = (\widehat{\texttt{out}}, {`}M_1, {`}M_2, {`}P) \\
reify_n({`}!P) & = (\widehat{\texttt{rep}}, {`}P) & reify_n({`}\nu c.P) & = (\widehat{\texttt{new}}, n, {`}\{^{n}/_{c}\}P) \\
reify_n({`}(P_1|P_2)) & = (\widehat{\texttt{par}}, {`}P_1, {`}P_2) & & \\
\multicolumn{4}{l}{reify_n({`}\mathit{if}\ M_1 = M_2\ \mathit{then}\ P_1\ \mathit{else}\ P_2) = (\widehat{\texttt{cond}}, {`}M_1, {`}M_2, {`}P_1, {`}P_2)} \\
\multicolumn{4}{l}{reify_n({`}\mathit{match}\ M\ \mathit{as}\ x\ \mathit{in}\ P) = (\widehat{\texttt{mtch}}, {`}M, n, {`}\{^{n}/_{x}\}P)} \\
reify_n({`}a) & = (\widehat{\texttt{name}}, a) & reify_n({`}f) & = (\widehat{\texttt{fun}}, f) \\
reify_n(\text{“}P) & = (\widehat{\texttt{pquo}}, {`}P) & reify_n(\text{“}M) & = (\widehat{\texttt{tquo}}, {`}M) \\
\multicolumn{4}{l}{reify_n({`}(M(M_1, \ldots, M_l))) = (\widehat{\texttt{uapp}}, {`}M, {`}M_1, \ldots, {`}M_l)} \\
\multicolumn{4}{l}{reify_n({`}\hat{f}(V_1, \ldots, V_l)) = (\widehat{\texttt{capp}}, \hat{f}(V_1, \ldots, V_l))}
\end{array}$$

Structural equivalence. Define evaluation contexts by $C ::= [] \mid (C|P) \mid (P|C) \mid \nu c.C$. Structural equivalence $\equiv$ is the smallest equivalence relation on processes that is closed under evaluation contexts, with:

$$P \equiv P|0 \qquad P_1|(P_2|P_3) \equiv (P_1|P_2)|P_3 \qquad P_1|P_2 \equiv P_2|P_1 \qquad !P \equiv P|!P$$
$$\nu a.0 \equiv 0 \qquad \nu a.\nu b.P \equiv \nu b.\nu a.P \qquad P_1|(\nu a.P_2) \equiv \nu a.(P_1|P_2)\ \ (\mathit{if}\ a \notin \mathsf{fn}(P_1))$$

The next lemma is useful for proving the soundness of some up-to techniques.

Lemma 1 (reduction respects structural equivalence).

1. $P \equiv Q$ *and* $P \xrightarrow{\alpha} P'$ *imply* $Q \xrightarrow{\alpha} Q'$ *and* $P' \equiv Q'$
2. $P \equiv Q$ *and* $Q \xrightarrow{\alpha} Q'$ *imply* $P \xrightarrow{\alpha} P'$ *and* $P' \equiv Q'$.

Proof. By induction on the derivation of $P \equiv Q$.

3 Environmental bisimulation

As outlined in the introduction, an environmental relation is a set of elements of the form $(\mathcal{E}, P, Q)$, where P, Q are closed processes and $\mathcal{E}$ is a binary relation on closed values. Intuitively, P and Q are the tested processes and $\mathcal{E}$ is the environment, i.e., the knowledge of the observer. We write $P\mathcal{X}_{\mathcal{E}}Q$ for $(\mathcal{E}, P, Q) \in \mathcal{X}$.

Definition 1 (environmental bisimulation). *Environmental relation $\mathcal{X}$ is an environmental bisimulation if $P\mathcal{X}_{\mathcal{E}}Q$ implies:*

1. $P \xrightarrow{\tau} P'$ *implies* $Q \Rightarrow Q'$ *and* $P'\mathcal{X}_{\mathcal{E}}Q'$
2. $P \xrightarrow{a(V)} P'$ *with* $a\hat{\mathcal{E}}b$ *and* $V(\hat{\mathcal{E}})^*W$, *implies* $Q \Rightarrow \xrightarrow{b(W)} \Rightarrow Q'$ *and* $P'\mathcal{X}_{\mathcal{E}}Q'$
3. $P \xrightarrow{\nu\tilde{c}.\overline{a}\langle V\rangle} P'$ *with* $a\hat{\mathcal{E}}b$ *and* $\tilde{c} \notin \mathsf{fn}(\#_1(\mathcal{E}))$, *implies* $\exists \tilde{d} \notin \mathsf{fn}(\#_2(\mathcal{E}))$. $Q \Rightarrow \xrightarrow{\nu\tilde{d}.\overline{b}\langle W\rangle} \Rightarrow Q'$ *and* $P'\mathcal{X}_{\mathcal{E}\cup\{(V,W)\}}Q'$
4. *the converse of (1-3) on* Q
5. $V_1\hat{\mathcal{E}}W_1$ *and* $V_2\hat{\mathcal{E}}W_2$ *imply* $V_1 = V_2 \iff W_1 = W_2$
6. $`(P')\hat{\mathcal{E}}`(Q')$ *implies* $P|P'\mathcal{X}_{\mathcal{E}}Q|Q'$
7. $P\mathcal{X}_{\mathcal{E}\cup\{(a,b)\}}Q$ *for any* $a \notin \mathsf{fn}(P, \#_1(\mathcal{E}))$ *and* $b \notin \mathsf{fn}(Q, \#_2(\mathcal{E}))$
8. $V\hat{\mathcal{E}}W$ *implies:*
   (a) $V = a$ *implies* $W = b$ *(i.e., if V is a name, then W is also a name)*
   (b) $V = f$ *implies* $W = f$
   (c) $V = \hat{f}(V_1, \ldots, V_l)$ *implies* $W = \hat{g}(W_1, \ldots, W_m)$
   (d) $V \in \mathbf{Quo}$ *implies* $\exists b \notin \mathsf{fn}(\mathcal{E}, P, Q)$. $P\mathcal{X}_{\mathcal{E}\cup\{(reify_b(V), reify_b(W))\}}Q$
9. *the converse of 8 on* W

Modulo symmetry, Definition 1 has 7 clauses. Clause 1 is the usual one for τ-transitions. Clause 2 is the input case. The channel names a and b are related by the observer's knowledge $\hat{\mathcal{E}}$. The input values V and W are synthesized from $\hat{\mathcal{E}}$, as discussed in the introduction. Clause 3 is the output case. Again, a and b are related by $\hat{\mathcal{E}}$. The environment is extended with the output values, again as discussed in the introduction. Clause 5 accounts for conditional contexts $if\ []_1 = []_2\ then\ P\ else\ Q$. Clause 6 allows the observer to run processes from the environment at any time. Clause 7 allows creation of fresh names by the observer. Clause 8 accounts for decomposition, with 8a–8c for contexts of the form $match\ `[]_1\ as\ x\ in\ P$ (which analyze the shape of the related values) and 8d for $match\ []_1\ as\ x\ in\ P$.

Environmental bisimilarity $\sim$ is the union of all environmental bisimulations, which exists because the union of all environmental bisimulations is an environmental bisimulation (all the conditions above are monotone on $\mathcal{X}$). Therefore, $P \sim_{\mathcal{E}} Q$ if $P\mathcal{X}_{\mathcal{E}}Q$ for some environmental bisimulation $\mathcal{X}$. The most important case is when $\mathcal{E} = \{(a, a) \mid a \in \mathsf{fn}(P, Q)\}$. We write $P \simeq Q$ for $P \sim_{\mathcal{E}} Q$ in this case. It asserts the equivalence between two processes when the observer knows all of their free names.

4 Up-to context technique

Up-to techniques are enhancements of the bisimulation proof method (see, e.g., [10]). "Bisimulations up-to" have weaker conditions than the original bisimulation clauses, and are therefore easier to use, but yet are included in the bisimilarity (provided that they are sound). We here present one of the most useful up-to techniques for our bisimulation.

We first define context closure for environmental bisimulations.

Definition 2. *For an environmental relation $\mathcal{X}$, we write $P\mathcal{X}_{\mathcal{E}}^{(*)}Q$ if $P \equiv \nu\tilde{c}.(P_0|P_1)$ and $Q \equiv \nu\tilde{d}.(Q_0|Q_1)$ where $P_0\mathcal{X}_{\mathcal{E}'}Q_0$ and $P_1(\hat{\mathcal{E}}')^*Q_1$, and if*

$$\hat{\mathcal{E}} \subseteq \{ (V,W) \mid V(\hat{\mathcal{E}}')^*W,\ \mathsf{fn}(V)\cap\{\tilde{c}\} = \mathsf{fn}(W)\cap\{\tilde{d}\} = \emptyset \}.$$

Intuitively, it is an extension of context closure for terms, where the observer's processes P_1, Q_1 are running in parallel with the tested processes P_0, Q_0, and fresh names $\tilde{c}$, $\tilde{d}$ have been generated but not exported yet.

Now we define the up-to technique. Essentially, this definition is obtained by replacing $\mathcal{X}$ with $\mathcal{X}^{(*)}$ in each clause of Definition 1.

Definition 3 (environmental bisimulation up-to context). *Environmental relation $\mathcal{X}$ is an environmental bisimulation up-to context*[5] *if $P\mathcal{X}_{\mathcal{E}}Q$ implies:*

1. *$P \xrightarrow{\tau} P'$ implies $Q \Rightarrow Q'$ and $P'\mathcal{X}_{\mathcal{E}}^{(*)}Q'$*
2. *$P \xrightarrow{a(V)} P'$ with $a\hat{\mathcal{E}}b$ and $V(\hat{\mathcal{E}})^*W$, implies $Q \Rightarrow \xrightarrow{b(W)} \Rightarrow Q'$ and $P'\mathcal{X}_{\mathcal{E}}^{(*)}Q'$*
3. *$P \xrightarrow{\nu\tilde{c}.\overline{a}\langle V\rangle} P'$ with $a\hat{\mathcal{E}}b$ and $\tilde{c} \notin \mathsf{fn}(\#_1(\mathcal{E}))$, implies $\exists\tilde{d} \notin \mathsf{fn}(\#_2(\mathcal{E}))$. $Q \Rightarrow \xrightarrow{\nu\tilde{d}.\overline{b}\langle W\rangle} \Rightarrow Q'$ and $P'\mathcal{X}_{\mathcal{E}\cup\{(V,W)\}}^{(*)}Q'$*
4. *the converse of (1-3) on Q*
5. *$V_1\hat{\mathcal{E}}W_1$ and $V_2\hat{\mathcal{E}}W_2$ imply $V_1 = V_2 \iff W_1 = W_2$*
6. *$`(P')\hat{\mathcal{E}}`(Q')$ implies $P|P'\mathcal{X}_{\mathcal{E}}^{(*)}Q|Q'$*
7. *$P\mathcal{X}_{\mathcal{E}\cup\{(a,b)\}}Q$ for any $a \notin \mathsf{fn}(P, \#_1(\mathcal{E}))$ and $b \notin \mathsf{fn}(Q, \#_2(\mathcal{E}))$*
8. *$V\hat{\mathcal{E}}W$ implies:*
   (a) $V = a$ implies $W = b$ (i.e., if V is a name, then W is also a name)
   (b) $V = f$ implies $W = f$
   (c) $V = \hat{f}(V_1, \ldots, V_l)$ implies $W = \hat{g}(W_1, \ldots, W_m)$
   (d) $V \in \mathbf{Quo}$ implies $\exists b \notin \mathsf{fn}(\mathcal{E}, P, Q)$. $P\mathcal{X}_{\mathcal{E}\cup\{(reify_b(V), reify_b(W))\}}^{(*)}Q$
9. *the converse of 8 on W*

Environmental bisimulations up-to context require weaker conditions than environmental bisimulations. (Thus an environmental bisimulation is always an environmental bisimulation up-to context.) Specifically, in clauses 1 to 3, the processes after transitions are required to be bisimilar only "up to context," i.e., modulo context closure. Similarly, in clauses 6 and 8d, the resulting processes are required to be bisimilar only modulo the context. Note that clause 6 is not a tautology because it allows to extract

[5] In fact, this is also up-to environment and up-to structural equivalence because of the use of $\subseteq$ and $\equiv$ in Definition 2.

(and execute) the quoted processes P' and Q', while the context closure $\mathcal{X}_{\mathcal{E}}^{(*)}$ does not (see Definition 2).

Soundness of the up-to technique is guaranteed by the fact that an environmental relation satisfying all the conditions above is a subset of $\sim$.

Theorem 1 (soundness of environmental bisimulation up-to context). *Let $\mathcal{Y}$ be the environmental bisimilarity up-to context. Then $\mathcal{X} = \{(\mathcal{E}, P, Q) \mid P\mathcal{Y}_{\mathcal{E}}^{(*)}Q\}$ is an environmental bisimulation.*

Proof. By checking each clause of environmental bisimulation against $\mathcal{X}$. The non-trivial cases are clauses 1, 2 and 3, which follow from the lemmas below (and their symmetric versions).

Lemma 2 (input transition). *Let $P_1\mathcal{E}^*Q_1$ and $a\mathcal{E}b$. Suppose that $\hat{\mathcal{E}}$ respects equality of names on the left hand side, i.e., for any a, there exists some b such that, for any W_1, $a\hat{\mathcal{E}}W_1$ implies $W_1 = b$. If $P_1 \xrightarrow{a(V)} P_1'$, then for any W, there exists some Q_1' such that $Q_1 \xrightarrow{b(W)} Q_1'$ with $P_1'(\mathcal{E} \cup \{(V, W)\})^*Q_1'$.*

Proof. By induction on the derivation of $P_1 \xrightarrow{a(V)} P_1'$.

Lemma 3 (output transition). *Let $P_1\mathcal{E}^*Q_1$ and $a\mathcal{E}b$. Suppose $\hat{\mathcal{E}}$ respects equality of names on the left hand side (see above for definition). If $P_1 \xrightarrow{\nu\tilde{c}.\overline{a}\langle V\rangle} P_1'$ with $\tilde{c} \notin \mathsf{fn}(\#_1(\mathcal{E}))$, then there exist some Q_1', W and $\tilde{d}$ with $V(\mathcal{E} \cup \widehat{\{(\tilde{c}, \tilde{d})\}})^*W$ such that $Q_1 \xrightarrow{\nu\tilde{d}.\overline{b}\langle W\rangle} Q_1'$ with $\tilde{d} \notin \mathsf{fn}(\#_2(\mathcal{E}))$ and $P_1'(\mathcal{E} \cup \{(\tilde{c}, \tilde{d})\})^*Q_1'$.*

Proof. By induction on the derivation of $P_1 \xrightarrow{\nu\tilde{c}.\overline{a}\langle V\rangle} P_1'$.

Note that, in the two lemmas above, no other assumption is necessary for $\mathcal{E}$.

Lemma 4 (τ transition). *Suppose $P_1(\hat{\mathcal{E}})^*Q_1$ and $P_0\mathcal{Y}_{\mathcal{E}}Q_0$ for an environmental bisimulation $\mathcal{Y}$ up-to context. If $P_1 \xrightarrow{\tau} P_1'$, then there exists some Q_1' such that $Q_1 \xrightarrow{\tau} Q_1'$ with $P_0|P_1'\mathcal{Y}_{\mathcal{E}}^{(*)}Q_0|Q_1'$.*

Proof. By induction on the derivation of $P_1 \xrightarrow{\tau} P_1'$, using Lemma 2 and 3.

Full details of the above proofs are available online [12].

While the up-to technique is useful for a bisimulation proof in general, we also use Theorem 1 to prove the soundness of the environmental bisimulation itself in the next section.

5 Soundness and completeness of environmental bisimilarity

We first define our criterion of observational equivalence, i.e., reduction-closed barbed equivalence. In this definition, meta-variable μ ranges over names and co-names ($\bar{a}$ etc.), $P \downarrow_a$ and $P \downarrow_{\bar{a}}$ mean that P can make an input and output transition on a, and $P \Downarrow_\mu$ is an abbreviation of $P \Rightarrow\downarrow_\mu$.

Definition 4 (reduction-closed barbed equivalence). *Reduction-closed barbed equivalence is the largest binary relation $\approx$ on closed processes such that $P \approx Q$ implies:*

1. *$P \xrightarrow{\tau} P'$ implies $Q \Rightarrow Q'$ and $P' \approx Q'$*
2. *$P \downarrow_\mu$ implies $Q \Downarrow_\mu$*
3. *the converse of 1 and 2 on Q*
4. *$P|R \approx Q|R$ for all processes R*

Theorem 2 (soundness and completeness of environmental bisimulation). *If $P \simeq Q$, then $P \approx Q$ and vice versa.*

Proof. For soundness (the forward implication), we check each clause in Definition 4 against $\simeq$. The non-trivial case is clause 4. Suppose $P \simeq Q$, i.e., $P \sim_{\mathcal{E}} Q$ for $\mathcal{E} = \{(a, a) \mid a \in \mathsf{fn}(P, Q)\}$. Let $\mathcal{E}' = \{(b, b) \mid b \in \mathsf{fn}(R)\}$. By clause 7 of environmental bisimulation, $P \sim_{\mathcal{E}\cup\mathcal{E}'} Q$. Since $R(\mathcal{E} \cup \mathcal{E}')^* R$, we have $P|R \sim^{(*)}_{\mathcal{E}\cup\mathcal{E}'} Q|R$ by Definition 2. Since $\sim$ is an environmental bisimulation up-to context, $P|R \sim_{\mathcal{E}\cup\mathcal{E}'} Q|R$ by Theorem 1. Hence $P|R \simeq Q|R$. For completeness (the backward implication), we take an environmental relation $\mathcal{X}$ that subsumes reduction-closed barbed equivalence, and prove it to be an environmental bisimulation. Again, see the online material [12] for details.

Note that reduction-closed barbed *congruence* is uninteresting in our calculus, since it almost coincides with α-equivalence (modulo possible differences between computed applications) because of quotation and decomposition, i.e., contexts like $match\ `[]_1\ as\ x\ in\ P$. (It is not interesting either to consider only contexts with no decomposition, because such contexts are *too* restricted, missing the whole point of our work.) In addition, it is anyway easy to (state and) prove the congruence of P and Q just by considering the equivalence of $\overline{a}\langle `P\rangle$ and $\overline{a}\langle `Q\rangle$ instead, because an evaluation context can receive $`P$ or $`Q$ from a and use them in arbitrary manners.

6 Examples

In the examples below, we use the following rewriting rules for terms, representing asymmetric cryptography.

$$\begin{array}{ll} pk(V) \to \widehat{pk}(V) & sk(V) \to \widehat{sk}(V) \\ f(V, \widehat{pk}(W)) \to \widehat{f}(V, \widehat{pk}(W)) & f^{-1}(V, \widehat{sk}(W)) \to \widehat{f^{-1}}(V, \widehat{sk}(W)) \\ f^{-1}(\widehat{f}(V, \widehat{pk}(W)), \widehat{sk}(W)) \to V & f(\widehat{f^{-1}}(V, \widehat{sk}(W)), \widehat{pk}(W)) \to V \end{array}$$

Functions pk and sk compute public and secret keys, respectively, from its argument. Functions f and f^{-1} denote encryption (or verification) and decryption (or signing). See e.g. [13] for more information on public-key encryption and digital signature.

The point of the examples is to show how to model and reason about higher-order communication systems involving (public-key) encryption by using our approach. It may also be possible to implement first-order variants of the systems, but they do not devalue our examples (just as the existence of first-order programs such as `mail(1)` does not devalue higher-order systems such as Gmail).

6.1 Software distribution with digital signature

The following system P consists of a server and clients. The server distributes a program R, which is then executed by the clients. For comparison, another system Q is defined where the clients "somehow" know R in the first place.

$$
\begin{array}{ll}
P = \nu k.(Server_k | Client_k) & Q = \nu k.(Server_k | Client'_k) \\
Client_k = !a(x).run(f(x, pk(k))) & Client'_k = !a(x).\nu c.(\overline{c}\langle f(x, pk(k))\rangle | c(y).R) \\
Server_k = !\overline{a}\langle pk(k)\rangle | !\overline{a}\langle f^{-1}(\text{`}R, sk(k))\rangle &
\end{array}
$$

We assume $k, c \notin \mathsf{fn}(R)$. $Client_k$ receives a quoted process R signed under the secret key $\widehat{sk}(k)$, and then verifies and executes it. By contrast, $Client'_k$ receives the same process but discards it, and then executes R. Equivalence of the two systems P and Q means that the clients can only execute R, not any Trojan horses. To prove this, we give an environmental relation $\mathcal{X}$ such that $P \mathcal{X}^{(*)}_{\mathcal{E}} Q$ for $\mathcal{E} = \{(b, b) \mid b \in \mathsf{fn}(P, Q)\}$.

Proposition 1. *The $\mathcal{X}$ below is an environmental bisimulation up-to context.*

$$
\begin{array}{ll}
\mathcal{X} = \{(\mathcal{E}_0, P_0, Q_0) \mid & P_0 = Server_{k'} | Client_{k'} | P_1 | \ldots | P_l, \\
& Q_0 = Server_{k''} | Client'_{k''} | Q_1 | \ldots | Q_l, \\
& l \geq 0, \\
& P_i = run(f(V_i, pk(k'))) \text{ for } i \geq 1, \\
& Q_i = \nu c.(\overline{c}\langle f(W_i, pk(k''))\rangle | c(y).R) \text{ with } c \notin \mathsf{fn}(W_i), \text{ for } i \geq 1, \\
& \tilde{V}(\hat{\mathcal{E}}_0)^* \tilde{W}, \\
& \mathcal{E}_0 = \mathcal{E}_1 \cup \mathcal{E}_2, \\
& \mathcal{E}_1 = \{(\widehat{pk}(k'), \widehat{pk}(k'')), (\widehat{f^{-1}}(\text{`}R, \widehat{sk}(k')), \widehat{f^{-1}}(\text{`}R, \widehat{sk}(k')))\}, \\
& \mathcal{E}_2 \supseteq \{(b, b) \mid b \in \mathsf{fn}(R, a)\}, \\
& \mathcal{E}_2 \text{ is a finite bijection on names}, \\
& k' \notin \mathsf{fn}(\#_1(\mathcal{E}_2)) \text{ and } k'' \notin \mathsf{fn}(\#_2(\mathcal{E}_2))\}
\end{array}
$$

Proof. By checking the conditions of environmental bisimulation up-to context, which follow from the construction of $\mathcal{X}$. Note, in particular, that we do *not* have to put (any number of) R in parallel with P_0 and Q_0, thanks to the up-to context technique.

First, observe that $P \mathcal{X}^{(*)}_{\mathcal{E}} Q$ by the definition of $\mathcal{X}$ (with $l = 0$) and by Definition 2 (context closure for environmental bisimulations). Hence $P \simeq Q$ by Theorem 1 (soundness of environmental bisimulation up-to context) if $\mathcal{X}$ is an environmental bisimulation up-to context.

To check $\mathcal{X}$ against the conditions of environmental bisimulation up-to context (Definition 3), consider first the transitions from P_0 (Conditions 1, 2 and 3).

The output of $pk(k')$ to a by $Server_{k'}$ on the left hand side can be matched by that of $pk(k'')$ to a by $Server_{k''}$ on the right hand side. In these transitions, neither the knowledge increases, nor the processes change (up-to structural equivalence). Ditto for the output of $f^{-1}(\text{`}R, sk(k'))$ and $f^{-1}(\text{`}R, sk(k''))$.

The input of V_i from a by $Client_{k'}$ spawns a new P_i, which can be matched by that of W_i from a by $Client'_{k''}$, spawning a new Q_i. Ditto for the internal communication from $Server_{k'}$ to $Client_{k'}$ (and from $Server_{k''}$ to $Client'_{k''}$) over a.

The internal transition by the process execution $run(f(V_i, pk(k')))$ in P_i succeeds only if the verification succeeds, i.e., only if V_i is of the form $\widehat{f^{-1}}(\text{`}R', \widehat{sk}(k'))$ for some

R'. Since $k' \notin \mathsf{fn}(\#_1(\mathcal{E}_2))$, this is possible only if $V_i = \widehat{f^{-1}}(\text{`}R, \widehat{sk}(k'))$, in which case $W_i = \widehat{f^{-1}}(\text{`}R, \widehat{sk}(k''))$. Then R is spawned both on the left hand side and on the right, which is cancelled out by up-to context.

The transitions from Q_0 (Condition 4) are similar. New processes spawned by the context (Condition 6) are also cancelled out by up-to context. This is straightforward because there are no quoted processes other than R in $\hat{\mathcal{E}}_0$. Conditions 5, 8 and 9 follow by straightforward induction on the first-order context D_t in the definition of $\hat{\mathcal{E}}_0$ (see Section 2.1). Finally, fresh names generated by the context (Condition 7) are immediately subsumed by the sub-environment $\mathcal{E}_2$.

We therefore have $P \approx Q$ from the soundness of environmental bisimulation (up-to context).

6.2 Secure mail user agent

Consider a client-server system where the user downloads (from the server) an e-mail user agent (MUA) to send an encrypted message.

$$
\begin{array}{ll}
P = \nu k_1.(Server_{k_1} | Client_{p,k_1}) & Q = \nu k_1.(Server_{k_1} | Client_{q,k_1}) \\
Client_{x,k_1} = \nu r.\overline{d}\langle f(r, pk(k_1))\rangle.r(y).run(\#_1(y))|\overline{\#_2(y)}\langle x\rangle & \\
Server_{k_1} = !\overline{c}\langle pk(k_1)\rangle \mid !d(x).\nu k_2.\nu m.\overline{f^{-1}(x, sk(k_1))}\langle \widehat{Pack}_{m,k_2}\rangle & \\
\widehat{Pack}_{m,k_2} = (\text{`}MUA_{m,k_2}, m) & MUA_{m,k_2} = m(y).\overline{c}\langle f(y, \widehat{pk}(k_2))\rangle
\end{array}
$$

In this example, c and d are public channels. The client first sends a request $f(r, pk(k_1))$ to download the MUA, and waits for a reply on channel r. The server then sends the MUA back to the client, with a fresh channel m for accepting the message y, and a fresh secret key k_2 for encrypting y. (We are using the private channel r only for the sake of simplicity. It could be implemented over a public network just as in the previous example.) Finally, the client sends its message x through m. Secrecy of the message x can be formalized by a standard non-interference property, i.e., that the system is equivalent regardless of the value of x. We here use two fresh, public names p and q for the values of x.

Again, to prove this equivalence, we give an environmental relation $\mathcal{X}$ s.t. $P\mathcal{X}_{\mathcal{E}}^{(*)}Q$ for $\mathcal{E} = \{(b, b) \mid b \in \mathsf{fn}(P, Q)\}$.

Proposition 2. *The $\mathcal{X}$ in Figure 1 is an environmental bisimulation up-to context.*

Proof. By checking each condition of environmental bisimulation up-to context. Again, this is easy thanks to the construction of $\mathcal{X}$ and to the up-to context technique.

As in the case of Proposition 1, we have $P\mathcal{X}_{\mathcal{E}}^{(*)}Q$ (by taking $l = 0$ in $\mathcal{X}_1$) and therefore $P \simeq Q$, provided that $\mathcal{X}$ is an environmental bisimulation up-to context.

Let us first consider the transitions from $\{^{k_1', \tilde{V}}/_{k_1, \tilde{z}}\}P_1$ in $\mathcal{X}_1$. The output of $pk(k_1')$ to c by $Server_{k_1'}$ on the left hand side is matched by that of $pk(k_1'')$ to c by $Server_{k_1''}$ on the right, with no increase of knowledge and no change of processes (up-to structural equivalence). The input of V_i from d by $Server_{k_1'}$ is matched by that of W_i from d by $Server_{k_1''}$, just incrementing the number l of processes in $R_{k_1'}$ and $R_{k_1''}$, respectively.

$$
\begin{aligned}
\mathcal{X} &= \mathcal{X}_1 \cup \mathcal{X}_2 \cup \mathcal{X}_3 \\
\mathcal{X}_1 &= \{(\mathcal{E}_0, \{^{k_1', \tilde{V}}/_{k_1,\tilde{z}}\}P_1, \{^{q}/_{p}\}\{^{k_1'', \tilde{W}}/_{k_1,\tilde{z}}\}P_1) \mid \\
&\quad P_1 = Server_{k_1} | R_{k_1} | Client_{p,k_1}, \\
&\quad R_{k_1} = \nu k_2.\nu m.\overline{f^{-1}(z_1, sk(k_1))}\langle \widehat{Pack}_{m,k_2}\rangle | \ldots | \nu k_2.\nu m.\overline{f^{-1}(z_l, sk(k_1))}\langle \widehat{Pack}_{m,k_2}\rangle, \\
&\quad l \geq 0, \\
&\quad \tilde{V}(\hat{\mathcal{E}}_0)^* \tilde{W}, \\
&\quad \mathcal{E}_0 = \mathcal{E}_1 \cup \mathcal{E}_2, \\
&\quad \mathcal{E}_1 = \{\ (\widehat{pk}(k_1'), \widehat{pk}(k_1''))\ \}, \\
&\quad \mathcal{E}_2 \supseteq \{(d,d),(c,c),(p,p),(q,q)\}, \\
&\quad \mathcal{E}_2 \text{ is a finite bijection on names}, \\
&\quad k_1' \notin \mathsf{fn}(\#_1(\mathcal{E}_2)) \text{ and } k_1'' \notin \mathsf{fn}(\#_2(\mathcal{E}_2))\} \\
\mathcal{X}_2 &= \{(\mathcal{E}_0, \{^{k_1', \tilde{V}, r'}/_{k_1,\tilde{z},r}\}P_2, \{^{q}/_{p}\}\{^{k_1'', \tilde{W}, r''}/_{k_1,\tilde{z},r}\}P_2), \\
&\quad (\mathcal{E}_0, \{^{k_1', \tilde{V}, r'}/_{k_1,\tilde{z},r}\}P_3, \{^{q}/_{p}\}\{^{k_1'', \tilde{W}, r''}/_{k_1,\tilde{z},r}\}P_3), \\
&\quad (\mathcal{E}_0, \{^{k_1', \tilde{V}, r'}/_{k_1,\tilde{z},r}\}P_4, \{^{q}/_{p}\}\{^{k_1'', \tilde{W}, r''}/_{k_1,\tilde{z},r}\}P_4), \\
&\quad (\mathcal{E}_0, \{^{k_1', \tilde{V}, r'}/_{k_1,\tilde{z},r}\}P_5, \{^{q}/_{p}\}\{^{k_1'', \tilde{W}, r''}/_{k_1,\tilde{z},r}\}P_5) \mid \\
&\quad P_2 = Server_{k_1} | R_{k_1} | r(y).(run(\#_1(y)) | \overline{\#_2(y)}\langle p\rangle), \\
&\quad P_3 = Server_{k_1} | R_{k_1} | \nu k_2.\nu m.(run(\#_1(\widehat{Pack}_{m,k_2})) | \overline{\#_2(\widehat{Pack}_{m,k_2})}\langle p\rangle), \\
&\quad P_4 = Server_{k_1} | R_{k_1} | \nu k_2.\nu m.(MUA_{m,k_2} | \overline{\#_2(\widehat{Pack}_{m,k_2})}\langle p\rangle), \\
&\quad P_5 = Server_{k_1} | R_{k_1} | \nu k_2.\overline{c}\langle f(p, \widehat{pk}(k_2))\rangle, \\
&\quad R_{k_1} = \nu k_2.\nu m.\overline{f^{-1}(z_1, sk(k_1))}\langle \widehat{Pack}_{m,k_2}\rangle | \ldots | \nu k_2.\nu m.\overline{f^{-1}(z_l, sk(k_1))}\langle \widehat{Pack}_{m,k_2}\rangle, \\
&\quad l \geq 0, \\
&\quad \tilde{V}(\hat{\mathcal{E}}_0)^* \tilde{W}, \\
&\quad \mathcal{E}_0 = \mathcal{E}_1 \cup \mathcal{E}_2, \\
&\quad \mathcal{E}_1 = \{\ (\widehat{pk}(k_1'), \widehat{pk}(k_1'')), \\
&\qquad\quad (\widehat{f}(r', \widehat{pk}(k_1')), \widehat{f}(r'', \widehat{pk}(k_1''))\ \}, \\
&\quad \mathcal{E}_2 \supseteq \{(d,d),(c,c),(p,p),(q,q)\}, \\
&\quad \mathcal{E}_2 \text{ is a finite bijection on names}, \\
&\quad k_1', r' \notin \mathsf{fn}(\#_1(\mathcal{E}_2)) \text{ and } k_1'', r'' \notin \mathsf{fn}(\#_2(\mathcal{E}_2))\} \\
\mathcal{X}_3 &= \{(\mathcal{E}_0, \{^{k_1', \tilde{V}, r'}/_{k_1,\tilde{z},r}\}P_6, \{^{q}/_{p}\}\{^{k_1'', \tilde{W}, r''}/_{k_1,\tilde{z},r}\}P_6) \mid \\
&\quad P_6 = Server_{k_1} | R_{k_1}, \\
&\quad R_{k_1} = \nu k_2.\nu m.\overline{f^{-1}(z_1, sk(k_1))}\langle \widehat{Pack}_{m,k_2}\rangle | \ldots | \nu k_2.\nu m.\overline{f^{-1}(z_l, sk(k_1))}\langle \widehat{Pack}_{m,k_2}\rangle, \\
&\quad l \geq 0, \\
&\quad \tilde{V}(\hat{\mathcal{E}}_0)^* \tilde{W}, \\
&\quad \mathcal{E}_0 = \mathcal{E}_1 \cup \mathcal{E}_2, \\
&\quad \mathcal{E}_1 = \{\ (\widehat{pk}(k_1'), \widehat{pk}(k_1'')), \\
&\qquad\quad (\widehat{f}(r', \widehat{pk}(k_1')), \widehat{f}(r'', \widehat{pk}(k_1''))), \\
&\qquad\quad (\widehat{f}(p, \widehat{pk}(k_2')), \widehat{f}(q, \widehat{pk}(k_2''))\ \}, \\
&\quad \mathcal{E}_2 \supseteq \{(d,d),(c,c),(p,p),(q,q)\}, \\
&\quad \mathcal{E}_2 \text{ is a finite bijection on names}, \\
&\quad k_1', r', k_2' \notin \mathsf{fn}(\#_1(\mathcal{E}_2)) \text{ and } k_1'', r'', k_2'' \notin \mathsf{fn}(\#_2(\mathcal{E}_2))\}
\end{aligned}
$$

Fig. 1. Environmental relation for the secure mail user agent

The possible output of $\widehat{Pack}_{m',k_2'}$ (with m' and k_2' fresh) by $R_{k_1'}$ is matched by that of $\widehat{Pack}_{m'',k_2''}$ (with m'' and k_2'' fresh) by $R_{k_1''}$. This increase of knowledge can then be cancelled out by up-to context with (m', m'') and (k_2', k_2'') added in $\mathcal{E}_2$.

The output of $f(r', pk(k_1'))$ (with r' fresh) to d by $Client_{p,k_1'}$ is matched by that of $f(r'', pk(k_1''))$ (with r'' fresh) to d by $Client_{q,k_1''}$. The results are included in $\mathcal{X}_2$.

Consider the transitions from $\{^{k_1',\tilde{V},r'}/_{k_1,\tilde{z},r}\}P_2$ in $\mathcal{X}_2$. The input and output by $Server_{k_1'}$ and $R_{k_1'}$ are the same as in the case of $\mathcal{X}_1$ (see above). The internal communication between $\nu k_2.\nu m.\overline{f^{-1}(V_i, sk(k_1'))}\langle\widehat{Pack}_{m,k_2}\rangle$ (in $R_{k_1'}$, with $V_i = \widehat{f}(r', \widehat{pk}(k_1'))$) and $r'(y).(run(\#_1(y))|\overline{\#_2(y)}\langle p\rangle)$ is matched by that between $\nu k_2.\nu m.\overline{f^{-1}(W_i, sk(k_1''))}$ $\langle\widehat{Pack}_{m,k_2}\rangle$ (in $R_{k_1''}$, with $W_i = \widehat{f}(r'', \widehat{pk}(k_1''))$) and $r''(y).(run(\#_1(y))|\overline{\#_2(y)}\langle q\rangle)$. The processes then become $\{^{k_1',\tilde{V},r'}/_{k_1,\tilde{z},r}\}P_3$ and $\{^{q}/_{p}\}\{^{k_1'',\tilde{W},r''}/_{k_1,\tilde{z},r}\}P_3$, respectively.

These processes make an internal transition by process execution $run(\#_1(\widehat{Pack}_{m,k_2}))$, becoming $\{^{k_1',\tilde{V},r'}/_{k_1,\tilde{z},r}\}P_4$ and $\{^{q}/_{p}\}\{^{k_1'',\tilde{W},r''}/_{k_1,\tilde{z},r}\}P_4$. They then make an internal communication over m and become $\{^{k_1',\tilde{V},r'}/_{k_1,\tilde{z},r}\}P_5$ and $\{^{q}/_{p}\}\{^{k_1'',\tilde{W},r''}/_{k_1,\tilde{z},r}\}P_5$, which send $\widehat{f}(p, \widehat{pk}(k_2'))$ and $\widehat{f}(q, \widehat{pk}(k_2''))$ (with k_2' and k_2'' fresh) to c. The results are included in $\mathcal{X}_3$. The other transitions are the same as in the case of $\{^{k_1',\tilde{V},r'}/_{k_1,\tilde{z},r}\}P_2$ (see above).

The transitions from $\{^{k_1',\tilde{V},r'}/_{k_1,\tilde{z},r}\}P_6$ in $\mathcal{X}_3$ are subsumed by the previous cases. Transitions from the right hand side are symmetric. Conditions on the environments follow again by straightforward induction on the first-order context D_t in the definition of $\hat{\mathcal{E}}_0$ (Section 2.1). Again as in the case of Proposition 1, processes spawned by the context are cancelled out by up-to context and fresh names generated by the context are subsumed by $\mathcal{E}_2$.

To repeat, the point of these examples is to illustrate our reasoning method for higher-order cryptographic processes ("Gmail"), even if it is possible to define first-order systems ("`mail(1)`") with a similar functionality.

7 Conclusion

We defined a higher-order process calculus parametrized by general algebra (which, for example, includes asymmetric cryptography), and developed a bisimulation proof method for proving behavioral equivalence in this language. We gave examples involving the security of higher-order systems with public-key encryption and digital signing.

As is the case with any bisimulation technique (or any "proof method" in general), it is always possible in hindsight to prove the same results as ours without explicitly using bisimulations, just by inlining (thereby duplicating) their soundness proof everywhere. In our case, doing so amounts to a brute-force proof based on the definition of reduction-closed barbed equivalence only. We emphasize that it is way too heavy to repeat such a proof in *every* instance of equivalence, so it pays to extract the proof pattern as a separate technique like ours. As in the present work, such development gives an essential insight—based on environments—for the (otherwise sightless) proof.

Acknowledgements. We thank the members of Kobayashi Laboratory in Tohoku University for their comments and helps. This work was partially supported by KAKENHI 18680003, 20240001, CASIO Science Promotion Foundation, and the Nakajima Foundation.

References

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 104–115, 2001.

[2] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999. Preliminary version appeared in *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pp. 36–47, 1997.

[3] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1999.

[4] B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. In *20th Annual IEEE Symposium on Logic in Computer Science*, pages 331–340, 2005.

[5] J. Borgström and U. Nestmann. On bisimulations for the spi calculus. In *9th International Conference on Algebraic Methodology and Software Technology*, volume 2422 of *Lecture Notes in Computer Science*, pages 287–303. Springer-Verlag, 2002.

[6] S. Maffeis, M. Abadi, C. Fournet, and A. D. Gordon. Code-carrying authorization. In *Computer Security - ESORICS 2008*, volume 5283 of *Lecture Notes in Computer Science*, pages 563–579. Springer-Verlag.

[7] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigm*. PhD thesis, University of Edinburgh, 1992.

[8] D. Sangiorgi, N. Kobayashi, and E. Sumii. Appendices to "environmental bisimulations for higher-order languages". `http://www.cs.unibo.it/~sangio/DOC_public/appLICS07.pdf`.

[9] D. Sangiorgi, N. Kobayashi, and E. Sumii. Environmental bisimulations for higher-order languages. In *Twenty-Second Annual IEEE Symposium on Logic in Computer Science*, pages 293–302, 2007.

[10] D. Sangiorgi and R. Milner. The problem of "weak bisimulation up to". In *CONCUR '92*, volume 630 of *Lecture Notes in Computer Science*, pages 32–46. Springer-Verlag, 1992.

[11] D. Sangiorgi and D. Walker. *The Pi Calculus – A Theory of Mobile Processes*. Cambridge University Press, 2001.

[12] N. Sato and E. Sumii. Proofs for "the higher-order, call-by-value applied pi-calculus". `http://www.kb.ecei.tohoku.ac.jp/~nsato/hoapp.pdf`.

[13] B. Schneier. *Applied Cryptography*. John Wiley & Sons, Inc., 1996.

[14] E. Sumii and B. C. Pierce. A bisimulation for dynamic sealing. *Theoretical Computer Science*, 375(1–3):169–192, 2007. Extended abstract appeared in *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 161–172, 2004.

[15] E. Sumii and B. C. Pierce. A bisimulation for type abstraction and recursion. *Journal of the ACM*, 54(5-26):1–43, 2007. Extended abstract appeared in *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 63–74, 2005.