Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 242 (2009) 3–20
www.elsevier.com/locate/entcs

A Complete Symbolic Bisimilarity for an Extended Spi Calculus

Johannes Borgström[1]

Department of Software Engineering and Theoretical Computer Science, Technische Universität Berlin, Germany

Abstract

Several symbolic notions of bisimilarity have been defined for the spi calculus and the applied pi calculus. In this paper, we treat a spi calculus with a general constructor-destructor message algebra, and define a symbolic bisimilarity that is both sound and complete with respect to its concrete counterpart.

Keywords: Cryptographic Protocols, Formal Verification, Bisimulation, Symbolic Techniques

The spi calculus, proposed by Abadi and Gordon [4] for the modelling and formal verification of cryptographic protocols, is an extension of the pi calculus [18] with cryptographic operators and operations. In this paper, we work in an extended spi calculus where the message algebra permits arbitrary constructors, and destructors with unique applicability.

As seen in for instance [4,13], many correctness properties for cryptographic protocols are naturally expressed through equivalences between certain process terms. To verify security properties expressed in this style, we need to choose a notion of equivalence. *Contextual* equivalences—two terms are related if they behave in the same way *in all contexts*—are attractive because the quantification over all contexts directly captures the intuition of an unknown attacker expressible within the spi calculus.

Direct proofs of contextual equivalences are notoriously hard [4] due to the requirement of infinitary quantifications (usually quantifications over infinitely many process contexts). Unfortunately, labelled bisimilarity is too strong a notion of equivalence for spi processes: It intuitively renders encryption ($\mathrm{E}_{(\cdot)}$) useless, by distinguishing between the (barbed equivalent) processes $(\nu k)\,\overline{a}\langle \mathrm{E}_k(M)\rangle$ and $(\nu k)\,\overline{a}\langle \mathrm{E}_k(N)\rangle$ whenever $M \neq N$. This problem was adressed [3,9] by explicitly

[1] Email: borgstrom@acm.org

doi:10.1016/j.entcs.2009.07.078

taking into account the *knowledge* of an environment about a process. Hedged bisimularity [12], defined along the same lines, is the starting point for this paper.

There is an inherent problem with the operational semantics of message-passing process calculi: The possibility to receive arbitrary messages gives rise to an infinite number of "concrete" transitions. Using a symbolic semantics, the substitution of received messages for input variables never takes place. Instead, an input prefix produces a single "symbolic" transition, where the input variable is only indirectly instantiated by means of constraints.

In [11], we proposed a symbolic structural operational semantics and a symbolic bisimulation for the spi calculus. In this paper, we define *decompositions* [8,15] of symbolic environments and update symbolic bisimulation to account for this, yielding both soundness and completeness with respect to concrete bisimilarities. Compared to work on symbolic (bisimulation) semantics [14,17] for the applied pi calculus [2], we give a complete proof method in a setting where the operational semantics are finitely branching.

1 The Spi calculus

The pi calculus [18] is a small language for modelling communicating and distributed systems, where communication channels can be generated and passed around. In contrast to the pi calculus, the spi calculus offers next to mere names another kind of transmissible messages, namely *ciphertexts*, which are provided by the addition of primitive constructs to encrypt ($\mathrm{E}_k(M)$) and decrypt ($\mathrm{D}_k(M)$) data using shared-key cryptography. In this paper, we generalize the message language further, permitting arbitrary constructors with corresponding destructors, but not more general equations. Apart from the extended message language, we use the same syntax and semantics as in our original paper [11] on symbolic bisimilarity in the spi calculus.

We use the lower case letters a, b, c, k, l, m, n to range over the infinite set $\mathcal{N}$ of names. Names are untyped, meaning that the same name can be used as a channel, a key or the clear-text of a message. We use x, y, z to range over the infinite set $\mathcal{V}$ of variables, and let u, v, w range over $\mathcal{N} \cup \mathcal{V}$. When $s_1, \ldots, s_{k-1}$ and s_k are terms (where k may be 0), we write "$\widetilde{s}$" as a shorthand for "$s_1, \ldots, s_k$".

We assume a fixed finite signature Σ, containing constructors f and destructors g. While expressions $F \in \mathcal{E}$ are formed arbitrarily using both constructors and destructors, messages $M \in \mathcal{M}$ are the expressions not containing destructor symbols. There is exactly one rewrite rule for every destructor g, that is of the form $g(f(\widetilde{M}), \widetilde{N}) \to M'$ where $\widetilde{M}, \widetilde{N}$ don't contain any names and $M' \in \{\widetilde{M}, \widetilde{N}\}$. We let $\to^{\mathrm{H}}$ be rewriting at the top level of the term, and write $G \prec F$ iff G is a subterm of F. In keeping with the operational flavor of this constructor-destructor language, we define term evaluation as the partial function $\mathbf{e}(F) := F{\downarrow}$ whenever $G{\downarrow} \in \mathcal{M}$ for all $G \prec F$, i.e., we require all destructors in F to succeed.

Logical formulae ϕ generalize the usual matching operator of the pi calculus by conjunction and negation. The predicate $[\,F : \mathcal{N}\,]$ tests if F evaluates to a name, so that it can be used as a channel. The semantics $[\![\cdot]\!]$ of the base predicates are as

follows: $[\![[\, F = G \,]]\!]$ is true iff $\mathbf{e}(F) = \mathbf{e}(G) \neq \bot$ and $[\![[\, F : \mathcal{N} \,]]\!]$ is true iff $\mathbf{e}(F) \in \mathcal{N}$. Conjunction and negation have their usual meaning.

Processes $\mathcal{P}$ are composed of the halted process $\mathbf{0}$, the input $F(x).P$, output $\overline{F}\langle F \rangle.P$ and replicated input $!F(x).P$ prefixes, choice $P + P$, parallel composition $P \,|\, P$, restriction $(\nu a)\, P$ and boolean guard ϕP.

$$\begin{array}{l} F, G ::= u \mid \quad f(\widetilde{F}) \quad \mid g(\widetilde{F}) \\ \phi, \psi \ ::= tt \mid [\, F = F \,] \mid [\, F : \mathcal{N} \,] \mid \phi \wedge \phi \mid \neg\phi \\ P, Q ::= \mathbf{0} \mid \ F(x).P \ \mid \overline{F}\langle F \rangle.P \mid !\, F(x).P \mid P + P \mid \ P \,|\, P \ \mid (\nu a)\, P \mid \phi P \end{array}$$

The names $\mathrm{n}(\cdot)$ resp. variables $\mathrm{v}(\cdot)$ of a term are the names resp. variables occuring in the term. Free and bound names and variables of process terms are inductively defined as expected: the name a is bound in "$(\nu a)\, P$" and the variable x is bound in "$F(x).P$" and "$!F(x).P$". Two processes are *α-equivalent* if they can be made equal by conflict-free renaming of bound names and variables. We generally identify α-equivalent processes.

Substitutions are idempotent functions $\{^{F}/_{x}\}$ from variables x to expressions F, and are applied to processes, expressions and guards in the straightforward way, obeying the usual assumption that capture of bound names and variables is avoided through implicit α-conversion. For example, $P\{^{F}/_{x}\}$ replaces all free occurrences of x in P by F, renaming bound names and variables in P where needed. Below, we give some representative transition rules for the late input semantics of the spi calculus.

In Table 1, we give the transition rules for the late input semantics for closed processes ($\mathrm{fv}(P) = \emptyset$).

Constructor-Destructor Languages

Constructor-destructor languages, as defined above (cf. [7]), are subterm convergent [1]. As a comparison, the *data term languages* of [5] constrain rewrite rules to be of the form $g(\widetilde{M}) \to x$ (where $x \in \mathrm{v}(\widetilde{M})$), yielding a special case of (possibly non-convergent) subterm languages.

We chose the format and unicity of the destructor rules for constructor-destructor languages to ensure a well-defined (deterministic) notion of evaluation, a smooth extension of the notions of synthesis and analysis and a strong correspondence between the concrete and symbolic operational semantics.

Example 1.1 The nondeterministic choice rules $\mathtt{either}((x\ .\ y)) \to x$ and $\mathtt{either}((x\ .\ y)) \to y$ cannot both be present in a constructor-destructor language, but are permitted in a data term language. They also do not in general yield a convergent rewrite system.

On the other hand, the limited inverse rule $\mathtt{f}(\mathtt{g}(\mathtt{h}(x))) \to \mathtt{h}(x)$ can be part of a constructor destructor language (if `g` and `h` are constructors and `f` a destructor), but is not permitted in a data term language.

The idempotent rule $\mathtt{f}(\mathtt{f}(x)) \to \mathtt{f}(x)$ or the self-inverse rule $\mathtt{f}(\mathtt{f}(x)) \to x$ can be part of a subterm-convergent rewrite system, but are not permitted in a constructor-

$$\text{(OUT)}\ \frac{\mathbf{e}(G) = a \qquad \mathbf{e}(F) = M}{\overline{G}\langle F\rangle.P \xrightarrow{\overline{a}\langle M\rangle} P} \qquad \text{(INP)}\ \frac{\mathbf{e}(G) = a}{G(x).P \xrightarrow{a(x)} P}$$

$$\text{(COM-R)}\ \frac{P \xrightarrow{(\nu \tilde{b})\,\overline{a}\langle M\rangle} P' \qquad Q \xrightarrow{a(x)} Q'}{P \mid Q \xrightarrow{\tau} (\nu \tilde{b})\ \left(P' \mid Q'\left\{{}^{M}/_{x}\right\}\right)}\ \text{if}\ \{\tilde{b}\} \cap \mathrm{fn}(Q) = \emptyset$$

$$\text{(REP)}\ \frac{\mathbf{e}(G) = a}{!\,G(x).P \xrightarrow{a(x)} P \mid !\,G(x).P} \qquad \text{(GUARD)}\ \frac{P \xrightarrow{\mu} P'}{\phi P \xrightarrow{\mu} P'}\ \text{if}\ [\![\phi]\!]$$

$$\text{(PAR-R)}\ \frac{Q \xrightarrow{\mu} Q'}{P \mid Q \xrightarrow{\mu} P \mid Q'}\ \text{if}\ \mathrm{bn}(\mu) \cap \mathrm{fn}(P) = \emptyset \qquad \text{(SUM-R)}\ \frac{Q \xrightarrow{\mu} Q'}{P + Q \xrightarrow{\mu} Q'}$$

$$\text{(OPEN)}\ \frac{P \xrightarrow{(\nu \tilde{b})\,\overline{a}\langle M\rangle} P'}{(\nu c)\,P \xrightarrow{(\nu c \tilde{b})\,\overline{a}\langle M\rangle} P'}\ \text{if}\ \begin{array}{l} c \in \mathrm{n}(M) \\ c \notin \{a, \tilde{b}\} \end{array} \qquad \text{(RES)}\ \frac{P \xrightarrow{\mu} P'}{(\nu c)\,P \xrightarrow{\mu} (\nu c)\,P'}\ \text{if}\ c \notin \mathrm{n}(\mu)$$

Table 1
Operational Semantics

destructor language nor a data term language.

The parameterized choice rules $\mathtt{pick}((x\ .\ y), 1) \to x$ and $\mathtt{pick}((x\ .\ y), 2) \to y$ are permitted in a data term language and yield a convergent rewrite system, but are not permitted in a constructor-destructor language (but see the definition of π_1 and π_2 below).

Constructor-destructor languages can express standard formal cryptography.

Example 1.2 We let $\Sigma_{\mathrm{DY}} = (\{\mathrm{E}, \mathrm{E}^+, \mathrm{E}^-, \mathsf{H}, \mathsf{pub}, (\cdot\ .\ \cdot), \mathrm{D}, \mathrm{D}^+, \mathrm{D}^-, \pi_1, \pi_2\}, \mathrm{ar})$ where $1 = \mathrm{ar}(\mathsf{H}) = \mathrm{ar}(\mathsf{pub}) = \mathrm{ar}(\pi_1) = \mathrm{ar}(\pi_2)$ and $2 = \mathrm{ar}(\mathrm{E}) = \mathrm{ar}(\mathrm{E}^+) = \mathrm{ar}(\mathrm{E}^-) = \mathrm{ar}((\cdot\ .\ \cdot)) = \mathrm{ar}(\mathrm{D}) = \mathrm{ar}(\mathrm{D}^+) = \mathrm{ar}(\mathrm{D}^-)$. Here E^+ (resp. E^-) denotes public (private) key encryption, and D^+ (D^-) the corresponding decryption. The rewrite system is given by $\mathrm{D}^+_y(\mathrm{E}^+_{\mathsf{pub}(y)}(x)) \to x$, $\mathrm{D}^-_{\mathsf{pub}(y)}(\mathrm{E}^-_y(x)) \to x$, $\mathrm{D}_y(\mathrm{E}_y(x)) \to x$, $\pi_1(x\ .\ y) \to x$ and $\pi_2(x\ .\ y) \to y$.

2 Hedged Bisimilarity, Revisited

Hedged bisimilarity was introduced in [12] in order to clarify the differences between other notions of environment-sensitive bisimulation for the spi calculus. For the simpler message language used in the original paper, hedged bisimulation yielded a sound and complete (for structurally image-finite processes) approximation of

barbed equivalence. The basic data structure to represent the knowledge of an attacker is sets of pairs of messages, called *hedges*. Since we compare two processes, the message pairs in a hedge relate corresponding messages where the first message in a pair arises from interactions with the first process; the second message is related to the second process. An environment is *consistent* if there is no noticeable difference between the two messages of any message pair. Since we use a richer message language than in previous work, we will also need to extend the operations on hedges that were defined there. The set of messages that can be generated using the knowledge of a hedge is called its *synthesis* ($\mathcal{S}$, cf. [19]). The notion of *analysis* ($\mathcal{A}$) becomes slightly more complicated in the current setting, since we do not constrain the arguments of destructors ("keys") to be names. Here, the rule ANA attempts to apply g to both sides of a pair in the analysis, constructing "keys" from the material that already has been analyzed. As a compact representation targeted towards implementations, we work with *irreducible* hedges ($\mathcal{I}$), i.e., where no more mutual decryptions within projections of the hedge are possible.

Definition 2.1 A *hedge* is a subset of $\mathcal{E} \times \mathcal{E}$. The *synthesis* $\mathcal{S}(h)$ of a hedge h is the smallest hedge containing h and satisfying the rule

$$(\text{SYN})\ \frac{(F_j, G_j) \in \mathcal{S}(h) \text{ for } j \in \{1, \ldots, \text{ar}(f)\}}{(f(\widetilde{F}), f(\widetilde{G})) \in \mathcal{S}(h)}$$

Let $\mathcal{S}^+(h) := \{(f(\widetilde{F}), f(\widetilde{G})) \mid (F_j, G_j) \in \mathcal{S}(h) \text{ for } j \in \{1, \ldots, \text{ar}(f)\}\}$.

The *analysis* $\mathcal{A}(h)$ of a hedge h is defined by mutual induction with an auxiliary set $\mathcal{SA}(h)$ by the following rules.

$$(\text{ANA-KNOWN})\ \frac{(F, G) \in h}{(F, G) \in \mathcal{A}(h)} \qquad (\text{ANA-S-KNOWN})\ \frac{(F, G) \in \mathcal{A}(h)}{(F, G) \in \mathcal{SA}(h)}$$

$$(\text{ANA})\ \frac{\begin{array}{c}(f(\widetilde{F}), f(\widetilde{G})) \in \mathcal{A}(h) \\ (F'_l, G'_l) \in \mathcal{SA}(h) \text{ for } l \in \{1, \ldots \text{ar}(g) - 1\}\end{array}}{(F, G) \in \mathcal{A}(h)} \begin{array}{l}\text{IF} \quad g(f(\widetilde{F}), \widetilde{F'}) \to^{\text{H}} F \\ \text{AND} \quad g(f(\widetilde{G}), \widetilde{G'}) \to^{\text{H}} G\end{array}$$

$$(\text{ANA-S})\ \frac{(F_j, G_j) \in \mathcal{SA}(h) \text{ for } j \in \{1, \ldots, \text{ar}(f)\}}{(f(\widetilde{F}), f(\widetilde{G})) \in \mathcal{SA}(h)}$$

The *irreducibles* $\mathcal{I}(\cdot)$ of a hedge are defined as $\mathcal{I}(h) := \mathcal{A}(h) \setminus \mathcal{S}^+(\mathcal{A}(h))$.

If S is a set of expressions, we let $\text{Id}_S = \{(F, F) \mid F \in S\}$. We write $h \vdash F \leftrightarrow G$ for $(F, G) \in \mathcal{S}(h)$. If h is a hedge, we let $h^{\text{T}} := \{\, (G, F) \mid (F, G) \in h \,\}$ and $\pi_i(h) := \{\, F_i \mid (F_1, F_2) \in h \,\}$ when $i \in \{1, 2\}$. A hedge h is irreducible iff $h = \mathcal{I}(h)$.

The only purpose of the set $\mathcal{SA}$ is to ensure that $\mathcal{A}(h)$ is well-founded. If we replaced $\mathcal{SA}(h)$ by $\mathcal{S}(\mathcal{A}(h))$ in ANA the definition would no longer be inductive, since we would a priori need to argue about the presence of certain expression pairs in $\mathcal{A}(h)$ before applying the rule. Indeed, for all hedges h, $\mathcal{SA}(h) = \mathcal{S}(\mathcal{A}(h))$.

Example 2.2 We work with the constructor-destructor language Σ_{DY} and let $h = \{(\mathsf{pub}(k), \mathsf{pub}(k)), (\mathrm{E}_k^-((n\ .\ m)), \mathrm{E}_k^-((n\ .\ n))), (\mathrm{E}_k^-(n), \mathrm{E}_l^-(n))\}$.
Applying Definition 2.1 to h with this language, we get
$\mathcal{A}(h) = h \cup \{((n\ .\ m), (n\ .\ n)), (n, m), (n, n)\}$ and $\mathcal{I}(h) = h \cup \{(n, m), (n, n)\}$.

In order to define a notion of consistency for concrete hedges, we use the notion of a pattern for a rewrite rule, intuitively a more abstract version of the left-hand side of the rule. As an extension of patterns, σ-patterns also track the possibilities to generate subterms of messages in range(σ) (cf. [1]).

Definition 2.3 An expression $g(\widetilde{M})$ is a pattern if there is $\rho : \mathcal{V} \to (\mathcal{M} \setminus \mathcal{V})$ with $g(\widetilde{M})\rho = F$, where F is the left-hand side of the rewrite rule for g.
If $g(\widetilde{M})$ is a pattern and $\sigma, \rho : \mathcal{V} \rightharpoonup \mathcal{M}$, then $g(\widetilde{M\rho})$ is a σ-pattern whenever range(ρ) $\subseteq \{M \notin \mathcal{V} \mid \mathrm{n}(M) = \emptyset \wedge \mathrm{v}(M) \subseteq \mathrm{dom}(\sigma) \wedge \exists N \in \mathrm{range}(\sigma)\ M\sigma \prec N\}$.

Example 2.4 Modulo renaming of variables, the patterns for our example rewrite system are $\pi_1(x)$, $\pi_2(x)$, $\mathrm{D}_y(x)$, $\mathrm{D}_y^+(x)$, $\mathrm{D}_y^-(x)$, $\mathrm{D}_x^+(\mathrm{E}_z^+(y))$, $\mathrm{D}_x^-(\mathrm{E}_z^-(y))$ and $\mathrm{D}_{\mathsf{pub}(x)}^-(y)$.

A hedge is consistent if, inuitively, the same operations performed on both sides give indistinguishable results. Here, we give a more operational definition of this condition [2].

Definition 2.5 We denote by $\mathbf{H} = \mathcal{P}_{\mathrm{fin}}(\mathcal{M} \times \mathcal{M})$ the set of all finite concrete hedges. An irreducible hedge $h \in \mathbf{H}$ is left consistent iff

(i) if $(a, N) \in h$ with $a \in \mathcal{N}$ then $N \in \mathcal{N}$; and
(ii) if $(M, N), (M', N') \in h$ such that $M = M'$ then $N = N'$; and
(iii) if $(M, N) \in h$ there is no N' with $(M, N') \in \mathcal{S}^+(h)$; and
(iv) Take σ_1, σ_2 with $h = \{(\sigma_1(x), \sigma_2(x)) \mid x \in \mathrm{dom}(\sigma_1)\}$ and $\mathrm{dom}(\sigma_1) = \mathrm{dom}(\sigma_2)$. If $g(\widetilde{M})$ is a σ_1-pattern and there is N_1 such that $g(\widetilde{M})\sigma_1 \to N_1$ then there is N_2 such that $g(\widetilde{M})\sigma_2 \to N_2$.

h is consistent iff h and h^{T} are both left consistent.

Since there are only finitely many σ-patterns (up to renaming) for any given σ, consistency is decidable.

Example 2.6 Continuing Example 2.2, we let
$h = \{(\mathsf{pub}(k), \mathsf{pub}(k)), (\mathrm{E}_k^-((n\ .\ m)), \mathrm{E}_k^-((n\ .\ n))), (\mathrm{E}_k^-(n), \mathrm{E}_l^-(n))\}$,
$h' = \mathcal{I}(h) = h \cup \{(n, n), (n, m)\}$ and $h'' = \mathcal{I}(h' \cup \{(k, k)\})$.

Then h' violates condition 2 for consistency since $\{(n, n), (n, m)\} \subset h'$. h' also violates condition 4 for consistency since $(\mathrm{E}_k^-(n), \mathrm{E}_l^-(n)) \in h'$ and $\mathrm{E}_k^-(n)$, but not $\mathrm{E}_l^-(n)$, can be decrypted by $\mathsf{pub}(k)$. Moreover, h'' violates condition 3, since $(\mathrm{E}_k^-(n), \mathrm{E}_l^-(n)) \in h''$ and $(\mathrm{E}_k^-(n), \mathrm{E}_k^-(n)) \in \mathcal{S}^+(h'')$.

Now that the environment and notions of consistency are defined, the definition of hedged bisimulation is straightforward. A *hedged relation* $\mathcal{R}$ is a subset of $\mathbf{H} \times$

[2] For a logical characterization of hedge consistency, see Chapter 3 of [10].

$\mathcal{P} \times \mathcal{P}$, where we write $h \vdash P\ \mathcal{R}\ Q$ for $(h, P, Q) \in \mathcal{R}$. We say that $\mathcal{R}$ is *consistent* if $h \vdash P\ \mathcal{R}\ Q$ implies that h is consistent.

Definition 2.7 A consistent hedged relation $\mathcal{R}$ is a *hedged simulation* if whenever $h \vdash P\ \mathcal{R}\ Q$ we have that

(i) If $P \xrightarrow{\tau} P'$ then there exists Q' such that $Q \xrightarrow{\tau} Q'$ and $h \vdash P'\ \mathcal{R}\ Q'$.

(ii) If there are a, b, x, B, M, N, P' such that $P \xrightarrow{a(x)} P'$, $h \vdash a \leftrightarrow b$, $B \subset \mathcal{N}$ is finite, $B \cap (\mathrm{fn}(P, Q) \cup \mathrm{n}(h)) = \emptyset$, $M, N \in \mathcal{M}$, and $h \cup \mathrm{Id}_B \vdash M \leftrightarrow N$, then there exists Q' such that $Q \xrightarrow{b(x)} Q'$ and $h \cup \mathrm{Id}_B \vdash P'\{M/x\}\ \mathcal{R}\ Q'\{N/x\}$.

(iii) If there are $a, b, \widetilde{c}, M, P'$ such that $P \xrightarrow{(\nu \tilde{c})\,\overline{a}\langle M\rangle} P'$, $h \vdash a \leftrightarrow b$ and $\{\tilde{c}\} \cap (\mathrm{fn}(P) \cup \mathrm{n}(\pi_1(h))) = \emptyset$ there are $Q', N, \tilde{d}$ with $\{\tilde{d}\} \cap (\mathrm{fn}(Q) \cup \mathrm{n}(\pi_2(h))) = \emptyset$ such that $Q \xrightarrow{(\nu \tilde{d})\,\overline{b}\langle N\rangle} Q'$ and $\mathcal{I}(h \cup \{(M, N)\}) \vdash P'\ \mathcal{R}\ Q'$.

$\mathcal{R}$ is a *hedged bisimulation* if both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are hedged simulations. We write $\sim_{\mathrm{h}}$ for the union of all hedged bisimulations.

On process output we use $\mathcal{I}(\cdot)$ to construct the new hedge after the transition. This entails applying all decryptions that the environment can do, producing the minimal extension of the hedge h with (M, N). This extension may turn out to be inconsistent, signifying that the environment has detected a difference between the messages received from the process pair.

3 Symbolic Semantics

The idea behind the symbolic operational semantics, as previously described in [11], is to record the necessary conditions for a transition as it is derived. A symbolic transition is written $P \xrightarrow[\phi]{\mu_s} P'$, where $\mu_s \in \{(\nu \tilde{c})\,\tau, (\nu \tilde{c})\,G(x), (\nu \tilde{c})\,\overline{G}\langle F\rangle\}$ and ϕ is the accumulated conditions for the transition. We let $\mathrm{bv}(a(x)) := \{x\}$ and $\mathrm{bn}((\nu \tilde{c})\,\tau) := \mathrm{bn}((\nu \tilde{c})\,\overline{G}\langle F\rangle) := \mathrm{bn}((\nu \tilde{c})\,G(x)) := \{\tilde{c}\}$.

Due to the more general message language than in [11], we here introduce a two-stage semantics, where the second stage is responsible for closing the restrictions of names that will only be present in the transition guard. We begin by defining the first stage as a SOS (Table 2). Compared to the concrete semantics, we simply record the sideconditions for the transition in the rules (Sout) and (Sinp). We do not close the resulting process term after a communication in the rule (Scom-r), since the expression that is communicated may contain fresh names that are not extruded in any corresponding concrete transition (cf. Example 3.1).

We intend to use the symbolic semantics to verify if certain assignments to input variables, represented by a substitution σ, enable a concrete transition. We do this by comparing the effects of applying the substitution before and after a transition, both on the resulting processes and the transition constraints. However, the single-stage semantics are not sufficient for this purpose, as we see in the following examples.

$$(\textsc{Sout})\ \overline{G}\langle F\rangle.P \xrightarrow[[G:\mathcal{N}]\wedge[F:\mathcal{M}]]{\overline{G}\langle F\rangle} P \qquad (\textsc{Sinp})\ G(x).P \xrightarrow[[G:\mathcal{N}]]{G(x)} P$$

$$(\textsc{Srep})\ !\,G(x).P \xrightarrow[[G:\mathcal{N}]]{G(x)} P \mid !\,G(x).P$$

$$(\textsc{Scom-r})\ \frac{P \xrightarrow[\phi_1]{(\nu\tilde{b})\,\overline{G}\langle F\rangle} P' \qquad Q \xrightarrow[\phi_2]{(\nu\tilde{c})\,G'(x)} Q'}{P \mid Q \xrightarrow[\phi_1\wedge\phi_2\wedge[G=G']]{(\nu\tilde{b}\tilde{c})\ \tau} P' \mid Q'\{F/x\}} \begin{array}{ll} \text{if} & \{\tilde{c}\}\cap \mathrm{fn}(P)=\emptyset \\ \text{and} & \{\tilde{b}\}\cap \mathrm{fn}(Q)=\emptyset \\ \text{and} & \{\tilde{c}\}\cap\{\tilde{b}\}=\emptyset \end{array}$$

$$(\textsc{Spar-r})\ \frac{Q \xrightarrow[\phi]{\mu_s} Q'}{P \mid Q \xrightarrow[\phi]{\mu_s} P \mid Q'}\ \text{if } (\mathrm{bn}(\mu_s))\cap \mathrm{fn}(P)=\emptyset \qquad (\textsc{Ssum-r})\ \frac{Q \xrightarrow[\phi]{\mu_s} Q'}{P+Q \xrightarrow[\phi]{\mu_s} Q'}$$

$$(\textsc{Sres})\ \frac{P \xrightarrow[\phi]{\mu_s} P'}{(\nu a)\,P \xrightarrow[\phi]{\mu_s} (\nu a)\,P'}\ \text{if } a\notin \mathrm{n}(\mu_s)\cup \mathrm{n}(\phi) \qquad (\textsc{Sguard})\ \frac{P \xrightarrow[\phi]{\mu_s} P'}{\phi' P \xrightarrow[\phi\wedge\phi']{\mu_s} P'}$$

$$(\textsc{Sopen})\ \frac{P \xrightarrow[\phi]{\mu_s} P'}{(\nu a)\,P \xrightarrow[\phi]{(\nu a)\,\mu_s} P'}\ \text{if } (\mathrm{fn}(\mu_s)\cup \mathrm{n}(\phi))\ni a\notin \mathrm{bn}(\mu_s)$$

Table 2
Symbolic Operational Semantics

Firstly, the resulting processes after concrete resp. symbolic transitions differ in which names are restricted.

Example 3.1 Let $P := (\nu b)\,\overline{a}\langle\pi_1(a\,.\,b)\rangle.P'$ for some P'. Concretely, $P \xrightarrow{\overline{a}\langle a\rangle} (\nu b)\,P'$. Symbolically we have that $P \xrightarrow[[a:\mathcal{N}]\wedge[\pi_1(a\,.\,b):\mathcal{M}]]{(\nu b)\,\overline{a}\langle\pi_1(a\,.\,b)\rangle} P'$, where the processes after the step only differ in the restriction of the name b. Also note that the scope of the binder for b in the symbolic transition extends to both the transition constraint and the resulting process.

Secondly, the symbolic semantics allow the communication of non-message terms, which after substitution need to be evaluated to coincide with the messages that are communicated in the concrete semantics.

Example 3.2 Now consider $Q := \overline{a}\langle\pi_1(x)\rangle \mid a(y).\overline{a}\langle y\rangle$. We can derive $Q \xrightarrow[\phi]{\tau} \mathbf{0} \mid \overline{a}\langle\pi_1(x)\rangle =: Q'$ with $\phi := [\,a:\mathcal{N}\,]\wedge[\,\pi_1(x):\mathcal{M}\,]\wedge[\,a:\mathcal{N}\,]\wedge[\,a=a\,]$.
We do not have $[\![\phi]\!]$, but the substitution $\sigma := \{(a\,.\,a)/x\}$ enables the transition.

Concretely, $Q\{^{(a\,.\,a)}/_x\} \xrightarrow{\tau} \mathbf{0} \mid \overline{a}\langle a\rangle$, but $\mathbf{0} \mid \overline{a}\langle a\rangle \neq \mathbf{0} \mid \overline{a}\langle \pi_1(a\,.\,a)\rangle = Q'\sigma$.

As seen in Example 3.1, the symbolic semantics may extrude the scope of more names than the concrete semantics. However, when working with a constructor-destructor expression language, we can compute exactly which names would be extruded by the concrete semantics, using a notion of "abstract evaluation". This abstract evaluation, $\mathbf{e}_\mathrm{a} : \mathcal{E} \to \mathcal{E}$, intuitively reduces a term wherever possible, without checking that e.g. decryption and encryption keys correspond.

Definition 3.3 We define $\to_\mathrm{A}$ as follows: For each g, if $g(f(\widetilde{M}), \widetilde{N}) \to^H M'$, $\widetilde{x}, \widetilde{y}$ are pairwise different, $\sigma = \{^{\widetilde{M}}/_{\widetilde{x}}\}\{^{\widetilde{N}}/_{\widetilde{y}}\}$ and $M' = \sigma(z)$, then $g(f(\widetilde{x}), \widetilde{y}) \to_\mathrm{A} z$. We then let $\mathbf{e}_\mathrm{a}(F) \stackrel{\mathrm{def}}{=} F{\downarrow_A}$.

We let the extruded names of an expression $\mathrm{en}(F)$ be defined inductively by $\mathrm{en}(a) = \{a\}$, $\mathrm{en}(x) = \emptyset$, $\mathrm{en}(g(\widetilde{G})) = \emptyset$ and $\mathrm{en}(f_i(\widetilde{G})) = \cup_j \mathrm{en}(G_j)$.

Example 3.4 Let $F := \pi_1(x)$ and $\sigma := \{^{(a\,.\,a)}/_x\}$. We have $\mathbf{e}_\mathrm{a}(F) = \pi_1(x)$, $\mathbf{e}_\mathrm{a}(F)\sigma = \pi_1(a\,.\,a)$ and $\mathbf{e}_\mathrm{a}(F\sigma) = a$.

We then have $\mathbf{e}(F) = \mathbf{e}_\mathrm{a}(F)$ for all $F \in \mathrm{dom}(\mathbf{e})$, or in other words, $\mathbf{e}_\mathrm{a}$ extends $\mathbf{e}$ to the entire set of expressions. Moreover, abstract evaluation commutes with substitution (modulo concrete evaluation). Using abstract evaluation, we define a version of the symbolic transition system that adds back restrictions to the resulting process, yielding a stronger correspondence.

Definition 3.5 We define the transition relation $\xrightarrow[\phi]{\mu_s}_s$ by

$$\textsc{CDtau}\ \frac{P \xrightarrow[\phi]{(\nu\tilde{b})\,\tau} P'}{P \xrightarrow[\phi]{(\nu\tilde{b})\,\tau}_s (\nu\tilde{b})\,P'} \qquad \textsc{CDinp}\ \frac{P \xrightarrow[\phi]{(\nu\tilde{b})\,F(x)} P'}{P \xrightarrow[\phi]{(\nu\tilde{b})\,F(x)}_s (\nu\tilde{b})\,P'}\ \text{if } \{\tilde{b}\} \cap \mathrm{en}(\mathbf{e}_\mathrm{a}(F)) = \emptyset$$

$$\textsc{CDout}\ \frac{P \xrightarrow[\phi]{(\nu\tilde{c})\,\overline{F}\langle G\rangle} P'}{P \xrightarrow[\phi]{(\nu\tilde{c})\,\overline{F}\langle G\rangle}_s (\nu\tilde{b})\,P'}\ \begin{array}{ll} \text{if} & \tilde{b} \text{ are pair-wise different} \\ \text{and} & \{\tilde{b}\} = \{\tilde{c}\} \setminus \mathrm{en}(\mathbf{e}_\mathrm{a}(G)) \\ \text{and} & \{\tilde{c}\} \cap \mathrm{en}(\mathbf{e}_\mathrm{a}(F)) = \emptyset \end{array}$$

Note that all restrictions are put back at the top level. To cope with this, as well as with the problems outlined in Example 3.2, we introduce the partial order $>_\mathrm{a}$ ("more abstract than"), which would be a subset of structural equivalence in an applied pi-style semantics [2].

Definition 3.6 We let $>_\mathrm{a}$ be the least reflexive and transitive precongruence on expressions, guards and processes satisfying

(i) $F >_\mathrm{a} M$ whenever $\mathbf{e}(F) = M$; and

(ii) $(\nu a)\,(\nu b)\,P >_\mathrm{a} (\nu b)\,(\nu a)\,P$; and

(iii) $(\nu a)\,(P \mid Q) >_\mathrm{a} ((\nu a)\,P) \mid Q$ when $a \notin \mathrm{fn}(Q)$; and

(iv) $(\nu a)\,(P \mid Q) >_{\mathrm{a}} P \mid ((\nu a)\,Q)$ when $a \notin \mathrm{fn}(P)$.

Example 3.7 Relating the effects of substituting before and after the transition in Example 3.2, we have $Q'\sigma = (\mathbf{0} \mid \overline{a}\langle \pi_1((a\,.\,a))\rangle.\,\mathbf{0}) >_{\mathrm{a}} (\mathbf{0} \mid \overline{a}\langle a\rangle.\,\mathbf{0})$.

The relation $>_{\mathrm{a}}$ is a (concrete) labelled bisimulation.

Lemma 3.8 *If $P >_{\mathrm{a}} Q$ then*

(i) *If $P \xrightarrow{\mu} P'$ then there is Q' such that $Q \xrightarrow{\mu} Q'$ and $P' >_{\mathrm{a}} Q'$; and*

(ii) *if $Q \xrightarrow{\mu} Q'$ such that $\mathrm{bn}(\mu) \cap \mathrm{fn}(P) = \emptyset$ then there is P' such that $P \xrightarrow{\mu} P'$ and $P' >_{\mathrm{a}} Q'$; and*

(iii) *$P\sigma >_{\mathrm{a}} Q\sigma$ for all substitutions $\sigma : \mathcal{V} \rightharpoonup \mathcal{M}$.*

Theorem 3.9

(i) *If $P \xrightarrow[\phi]{\mu_s}_s P_1$ and σ is idempotent and satisfies $\mathrm{n}(\mathrm{range}(\sigma)) \cap \mathrm{bn}(\mu_s) = \emptyset$, $[\![\phi\sigma]\!]$ and $\mu := \mathbf{e}(\mu_s\sigma)$ is defined, then there is P_2 with $P\sigma \xrightarrow{\mu} P_2$ and $P_1\sigma >_{\mathrm{a}} P_2$.*

(ii) *If σ is idempotent and $P\sigma \xrightarrow{\mu} P_1$ with $\mathrm{n}(\mathrm{range}(\sigma)) \cap \mathrm{bn}(\mu) = \emptyset$ then there are ϕ, μ_s, P_2 such $\mu = \mathbf{e}(\mu_s\sigma)$, $[\![\phi\sigma]\!]$, $P \xrightarrow[\phi]{\mu_s}_s P_2$ and $P_2\sigma >_{\mathrm{a}} P_1$.*

We now have a symbolic operational semantics that is sound and complete with respect to the concrete one (modulo $>_{\mathrm{a}}$, which is a labelled bisimulation) and is finitely branching (modulo choices of bound names and variables).

3.1 Symbolic Environments

A symbolic environment is a concise description of a set of hedges, differing only in the instantiations of variables. Here, a *variable instantiation* is a pair of substitutions, that must *respect* the symbolic environment. The hedges that we get from instantiating variables in an environment-respecting way are called *concretizations*. The symbolic environments used in this paper are very similar to the ones in [11], with the only difference that we keep explicit track of fresh names. A symbolic environment consists of the following three elements.

(i) A timed hedge $th : \mathcal{E} \times \mathcal{E} \rightharpoonup \mathbb{N}$ containing pairs of messages considered equal by the environment and the time they were received.

(ii) A timed variable set $tv : \mathcal{V} \rightharpoonup \mathbb{N}^+$ containing input variables and the time they were input.

(iii) A pair of restricted formulae $((\nu C)\,\phi, (\nu D)\,\psi)$ representing the accumulated transition constraints and sets of fresh names.

As mentioned above, the original version of symbolic environments did not include C and D; they facilitate environment decomposition (Def. 3.15).

Definition 3.10 We write se for the environment $(th, tv, ((\nu C)\,\phi, (\nu D)\,\psi))$. By abuse of notation, we write ϕ for $(\nu\emptyset)\,\phi$ and $(\nu a)\,\phi$ for $(\nu\{a\})\,\phi$ in environments. We let $th^{\mathrm{T}} := \{(F, G) \mapsto th(G, F) \mid (G, F) \in \mathrm{dom}(th)\}$ in order to swap the sides of a

timed hedge. We let $\mathrm{n}_1(se) := \mathrm{n}(\pi_1(\mathrm{dom}(th)))\cup C\cup\mathrm{n}(\phi)$, $\mathrm{n}_2(se) := \mathrm{n}(\pi_2(\mathrm{dom}(th)))\cup D\cup\mathrm{n}(\psi)$ and $\mathrm{n}(se) := \mathrm{n}_1(se)\cup\mathrm{n}_2(se)$.

Intuitively, if the environment knows the pair (F, G) it must have learned about it with the help of the processes at time $th(F, G)$; if this pair contains an input variable x, then the process must have performed this input at the strictly earlier time $tv(x)$.

Definition 3.11 The environment se is *well-formed* if $\mathrm{dom}(th)$ is finite, $0 \in \mathrm{range}(th)$, $\mathrm{v}(\mathrm{range}(th), \phi, \psi) \subseteq \mathrm{dom}(tv)$, and whenever $(F, G) \in \mathrm{dom}(th)$ such that $x \in \mathrm{v}(F, G)$ then $tv(x) < th(F, G)$.

From here on we only consider well-formed symbolic environments, the set of which is denoted **SE**. By instantiating the input variables of the symbolic environment, we can get a concrete (non-timed) hedge. However, such an instantiation must be subject to several constraints, e.g., timing, guard satisfaction and freshness of invented names. For instance, an input performed at time t must be synthesizable from the knowledge of the environment at that time. Similarly to the symbolic early input semantics, we define environment respectfulness for substitutions. Naturally, with the bisimulation environments we need two (possibly different) substitutions, one for each process. We also create some fresh names B.

Definition 3.12 A substitution pair (σ, ρ) is se-respecting with $B \subseteq \mathcal{N}$, written $se \vdash \sigma \leftrightarrow_B \rho$ iff (i) to (iv) below hold.

(i) $\mathrm{dom}(\sigma) = \mathrm{dom}(\rho) = \mathrm{dom}(tv)$

(ii) $[\![\phi\sigma]\!]$ and $[\![\psi\rho]\!]$

(iii) if $tv(x) = t$ then $(\sigma(x), \rho(x)) \in \mathcal{S}(\mathcal{I}(\{(F\sigma\downarrow, G\rho\downarrow) \mid th(F, G) \leq t\} \cup \mathrm{Id}_B))$

(iv) B is fresh and minimal in the sense that $(\mathrm{n}(\mathrm{range}(th)) \cup C \cup D) \cap B = \emptyset$ and if $a \in B$ then $a \in \mathrm{n}(\mathrm{range}(\sigma))$ or $a \in \mathrm{n}(\mathrm{range}(\rho))$.

If $se \vdash \sigma \leftrightarrow_B \rho$ we can concretize the knowledge th of the symbolic environment se by letting $\mathbf{C}^B_{\sigma,\rho}(th) := \mathcal{I}(\{(\mathbf{e}(F\sigma), \mathbf{e}(G\rho)) \mid (F, G) \in \mathrm{dom}(th)\} \cup \mathrm{Id}_B)$.

In condition iii of the above definition we use $F\sigma\downarrow$ rather than $\mathbf{e}(F\sigma)$ since the latter may be undefined. Indeed, $\mathbf{C}^B_{\sigma,\rho}(th)$ may be undefined, signifying that a received message was in fact a non-message expression. This cannot happen when using the symbolic semantics, since the requirement for the transmitted expression to be a message is always added to the transition constraint. This yields a *concretizable* symbolic environment (defined below), that always has well defined concretizations.

Example 3.13 Let $th := \{(x, x) \mapsto 2\}$, $tv := \{x \mapsto 1\}$ and $\sigma := \rho := \left\{\mathrm{E}_a(a)/x\right\}$. Then we have $(th, tv, (tt, tt)) \vdash \sigma \leftrightarrow_{\{a\}} \rho$, and $\mathbf{C}^{\{a\}}_{\sigma,\rho}(th) = \{(a, a)\}$ is consistent. If the definition of $\mathbf{C}^{\cdot}_{\cdot,\cdot}(\cdot)$ did not use $\mathcal{I}(\cdot)$, then $\mathbf{C}^{\{a\}}_{\sigma,\rho}(th) = \{(\mathrm{E}_a(a), \mathrm{E}_a(a)), (a, a)\}$ would not be consistent.

Since the se-respecting substitution pairs describe all admissible (with respect to the knowledge and constraints of se) instantiations of input variables, it is interesting

to apply all of them to a pair of formulae (e.g., transition constraints) and study the results. If the formulae are only satisfied simultaneously, they are equivalent from the point of view of the environment. For an environment to be consistent, we require any concretization of its knowledge to be a consistent hedge. We also require that the accumulated constraints are satisfied simultaneously on both sides (the second condition below).

Definition 3.14 We write $se \vDash \phi' \Leftrightarrow \psi'$ if for all B, σ, ρ : $se \vdash \sigma \leftrightarrow_B \rho$ implies that $[\![\phi'\sigma]\!]$ iff $[\![\psi'\rho]\!]$. se is *concretizable* if when $(F, G) \in \mathrm{dom}(th)$ we have $se \vDash [F{:}\mathcal{M}] \Leftrightarrow tt$ and $se \vDash tt \Leftrightarrow [G{:}\mathcal{M}]$.

A concretizable symbolic environment se is *consistent* if $\mathbf{C}^B_{\sigma,\rho}(th)$ is consistent whenever $se \vdash \sigma \leftrightarrow_B \rho$, and $(th, tv, ((\nu C)\ tt, (\nu D)\ tt)) \vDash \phi \Leftrightarrow \psi$.

Note that if se is concretizable and $se \vdash \sigma \leftrightarrow_B \rho$ then $\mathbf{C}^B_{\sigma,\rho}(th)$ is always defined and σ and ρ are both idempotent.

When simulating a transition, we often need to consider different cases. In order to split a symbolic environment according to these cases, we may decompose the constraints [8,15]. Since we keep constraints for both sides of the environment we may require that the split is consistent, following [17].

Definition 3.15 Let $se = (th, tv, ((\nu C)\,\phi, (\nu D)\,\psi))$ be a concretizable symbolic environment. The set $\{se_i\}_{i\in I}$ is a decomposition of se if each se_i is of the form $(th, tv, ((\nu C)\,\phi_i, (\nu D)\,\psi_i))$, and whenever $se \vdash \sigma \leftrightarrow_B \rho$ there is $i \in I$ such that $se_i \vdash \sigma \leftrightarrow_B \rho$. A decomposition $\{se_i\}_{i\in I}$ is concretizable/consistent if each se_i is concretizable/consistent.

Example 3.16 Let $se_\phi := (\{(a,a) \mapsto 0\}, \{x \mapsto 1\}, (\phi, \phi))$. $\{se_{[x=a]}, se_{\neg[x=a]}\}$ is a decomposition of se_{tt}. Moreover, $\{se\}$ is a decomposition of any se.

We can *fully decompose* a consistent environment into an infinite set of environments with unique solutions as follows.

Lemma 3.17 *Let $se = (th, tv, ((\nu C)\,\phi, (\nu D)\,\psi))$ be a consistent environment and $I = \{(\sigma, \rho, B) \mid se \vdash \sigma \leftrightarrow_B \rho\}$. Then $\{se_{(\sigma,\rho,B)}\}_{(\sigma,\rho,B)\in I}$ where $\phi_{(\sigma,\rho,B)} = \bigwedge_{x\in\mathrm{dom}(tv)}[x = \sigma(x)]$ and $\psi_{(\sigma,\rho,B)} = \bigwedge_{x\in\mathrm{dom}(tv)}[x = \rho(x)]$ is a decomposition of se.*
Moreover, for each $(\sigma, \rho, B) \in I$, $se_{(\sigma,\rho,B)} \vdash \sigma' \leftrightarrow_{B'} \rho'$ iff $(\sigma', \rho', B') = (\sigma, \rho, B)$.

In the pi calculus, it is always sufficient to consider a finite number of cases in the decomposition [8]. However, in a spi calculus an infinite split may be needed when treating processes with replication.

Example 3.18 We take a simple expression language that allows us to encode integers. Let $\Sigma = (\{\mathtt{s}, \mathtt{p}\}, \{\mathtt{s} \mapsto 1, \mathtt{p} \mapsto 1\})$ with the rewrite rule $\mathtt{p}(\mathtt{s}(x)) \to x$. This language is a constructor-destructor language, and would also be admissible as a data term language. We write n_a for the message $\mathtt{s}^n(a)$.

We define processes P and Q with the same behavior (i.e., $P \sim Q$ where $\sim$ is strong labelled bisimulation, as commonly defined). Upon input of a message n_a,

P non-deterministically decides to diverge or to perform an output on a after n more steps. On the other hand, upon input of n_a Q non-deterministically decides to become either Q_1 or Q_2, where Q_1 performs an output on a after n steps if n is odd and diverges if n is even, while Q_2 performs an output on a after n steps if n is even and diverges if n is odd.

$$\begin{aligned}
P &= a(x).\Omega + a(x).(\nu c)\,(P'(x) \mid !c(y).P'(y)) \\
P'(x) &= \overline{x}\langle a\rangle + \overline{c}\langle \mathtt{p}(x)\rangle \\
Q &= (\nu c)\,((a(x).Q_1(x) + a(x).Q_2(x)) \mid !c(y).Q_2(y)) \\
Q_1(x) &= [\,x : \mathcal{N}\,]\Omega + \overline{c}\langle \mathtt{p}(x)\rangle \\
Q_2(x) &= \overline{x}\langle a\rangle + (\nu d)\,(\overline{d}\langle \mathtt{p}(x)\rangle \mid d(z).Q_1(z)) \\
\Omega &= (\nu c)\,(\overline{c}\langle c\rangle \mid !c(z).\overline{c}\langle c\rangle)
\end{aligned}$$

After the choice of the first process we need to make a choice in the second process, dependent on whether n is even or odd. Symbolically, in order to make the choice in the second process we need to describe the condition "n is even (odd)" using a disjunction of formulas. We conjecture that this cannot be done with a finite disjunction (of finite formulas) in this guard and expression language.

The question then arises if it would be possible to extend the logical language used in environments to always enable a finite decomposition. However, a more sophisticated version of this example would use that the (finite-control) spi calculus is Turing-complete [16]. We could then let P receive an encoding of a Turing machine and its input and choose between diverging or simulating the machine, signalling failure or success upon termination. Q would make an initial choice and simulate the machine in both cases, diverging on failure (resp. success) and signalling success (resp. failure). A finite decomposition would then require a finite disjunction representing the predicate
"$t \in \{(M\ .\ N)$ where M codes for a Turing machine that accepts (rejects) $N\}$".
This is clearly also an issue for automated verification. However, in our experiments with simple security protocols we have not had use for any decomposition, suggesting that the actual impact of this issue is highly domain-dependent.

3.2 Symbolic Bisimulation

In [11], we defined a notion of symbolic bisimulation that was sound with respect to hedged (concrete) bisimulation, and thus with respect to barbed equivalence. In this section, we extend this definition with environment decompositions, also yielding completeness. The main ingredient of this definition is the symbolic environments seen above, that keep track of the accumulated transition constraints and the time relationships between inputs and outputs in order to make a proper accounting of the knowledge of the adversary.

A *symbolic relation* $\mathcal{R}$ is a subset of $\mathbf{SE} \times \mathcal{P} \times \mathcal{P}$. We write $se \vdash P\ \mathcal{R}\ Q$ for $(se, P, Q) \in \mathcal{R}$. $\mathcal{R}$ is *symmetric* if whenever $se \vdash P\ \mathcal{R}\ Q$ we have that $(th^{\mathrm{T}}, tv^{\mathrm{T}}, ((\nu D)\,\psi, (\nu C)\,\phi)) \vdash Q\ \mathcal{R}\ P$. $\mathcal{R}$ is *consistent* if se is consistent and $\mathrm{fv}(P, Q) \subseteq \mathrm{dom}(tv)$ whenever $se \vdash P\ \mathcal{R}\ Q$.

Intuitively, for two processes to be bisimilar under a given environment every *possible* and *detectable* transition of one of the processes must yield a *decomposition* of the resulting environment such that every element in the decomposition has a simulating transition of the other process on a *corresponding* channel such that the *updated* environment is consistent. The consistency of the updated environment implies that the simulating transition is also possible and detectable.

Definition 3.19 A symmetric consistent symbolic relation $\mathcal{R}$ is a *symbolic bisimulation* if whenever $se \vdash P \ \mathcal{R} \ Q$ with $se = (th, tv, ((\nu C)\,\phi, (\nu D)\,\psi))$ and $t = \max(\mathrm{range}(th) \cup \mathrm{range}(tv))$ then

- If $P \xrightarrow[\phi']{(\nu\tilde{c})\,\tau}_s P'$ with $\{\tilde{c}\} \cap \mathrm{n}_1(se) = \emptyset$, and there are σ, ρ, B with $se \vdash \sigma \leftrightarrow_B \rho$, $[\![\phi'\sigma]\!]$ and $(\{\tilde{c}\} \cup \mathrm{fn}(P,Q)) \cap B = \emptyset$,
  then there is a decomposition $\{se_i\}_{i\in I}$ of $(th, tv, ((\nu C \cup \{\tilde{c}\})\,\phi \wedge \phi', (\nu D)\,\psi))$ such that for each $i \in I$, there are $\{\tilde{e}\}, \psi', Q'$ with $Q \xrightarrow[\psi']{(\nu\tilde{e})\,\tau}_s Q'$, $\{\tilde{e}\} \cap (\mathrm{n}_2(se) \cup B) = \emptyset$ and $(th, tv, ((\nu C \cup \{\tilde{c}\})\,\phi_i, (\nu D \cup \{\tilde{e}\})\,\psi_i)) \vdash tt \leftrightarrow \psi'$. Finally, we require $(th, tv, ((\nu C \cup \{\tilde{c}\})\,\phi_i, (\nu D \cup \{\tilde{e}\})\,\psi_i)) \vdash P' \ \mathcal{R} \ Q'$.
- If $P \xrightarrow[\phi']{(\nu\tilde{c})\,F(x)}_s P'$ with $\{\tilde{c}\} \cap \mathrm{n}_1(se) = \emptyset$ and $x \notin \mathrm{dom}(tv)$, and there are σ, ρ, B with $se \vdash \sigma \leftrightarrow_B \rho$, $[\![\phi'\sigma]\!]$, $\mathbf{e}(F\sigma) \in \pi_1(\mathbf{C}^B_{\sigma,\rho}(th))$ and $(\{\tilde{c}\} \cup \mathrm{fn}(P,Q)) \cap B = \emptyset$,
  then there are $y \notin (\mathrm{dom}(tv) \cup \{x\})$ and a decomposition $\{se_i\}_{i\in I}$ of $(th, tv', ((\nu C \cup \{\tilde{c}\})\,\phi \wedge \phi' \wedge [\,y = F\,], (\nu D)\,\psi))$ where $tv' = tv \cup \{x \mapsto t{+}1, y \mapsto t{+}1\}$
  such that for each $i \in I$, there are $\{\tilde{e}\}, \psi', Q', F'$ with $Q \xrightarrow[\psi']{(\nu\tilde{e})\,F'(x)}_s Q'$, $\{\tilde{e}\} \cap (\mathrm{n}_2(se) \cup B) = \emptyset$ and $(th, tv', ((\nu C \cup \{\tilde{c}\})\,\phi_i, (\nu D \cup \{\tilde{e}\})\,\psi_i)) \vdash tt \leftrightarrow \psi' \wedge [\,y = F'\,]$. Finally, we require $(th, tv', ((\nu C \cup \{\tilde{c}\})\,\phi_i, (\nu D \cup \{\tilde{e}\})\,\psi_i)) \vdash P' \ \mathcal{R} \ Q'$.
- If $P \xrightarrow[\phi']{(\nu\tilde{c})\,\overline{F}\langle G\rangle}_s P'$ with $\{\tilde{c}\} \cap \mathrm{n}_1(se) = \emptyset$, and
  there are σ, ρ, B with $se \vdash \sigma \leftrightarrow_B \rho$, $[\![\phi'\sigma]\!]$, $\mathbf{e}(F\sigma) \in \pi_1(\mathbf{C}^B_{\sigma,\rho}(th))$, $x \notin \mathrm{dom}(tv)$ and $(\{\tilde{c}\} \cup \mathrm{fn}(P,Q)) \cap B = \emptyset$,
  then there are $y \notin \mathrm{dom}(tv)$ and a decomposition $\{se_i\}_{i\in I}$ of $(th, tv', ((\nu C \cup \{\tilde{c}\})\,\phi \wedge \phi' \wedge [\,y = F\,], (\nu D)\,\psi))$ where $tv' = tv \cup \{y \mapsto t{+}1\}$
  such that for each $i \in I$, there are $\{\tilde{e}\}, \psi', Q', F', G'$ with $Q \xrightarrow[\psi']{(\nu\tilde{e})\,\overline{F'}\langle G\rangle'}_s Q'$, $\{\tilde{e}\} \cap (\mathrm{n}_2(se) \cup B) = \emptyset$ and $(th', tv', ((\nu C \cup \{\tilde{c}\})\,\phi_i, (\nu D \cup \{\tilde{e}\})\,\psi_i)) \vdash tt \leftrightarrow \psi' \wedge [\,y = F'\,]$ where $th' = th \cup \{(G, G') \mapsto i{+}1\}$ if $G, G' \notin \mathrm{dom}(th)$, $th' = th$ otherwise.
  Then $(th', tv', ((\nu C \cup \{\tilde{c}\})\,\phi_i, (\nu D \cup \{\tilde{e}\})\,\psi_i)) \vdash P' \ \mathcal{R} \ Q'$.

Symbolic bisimilarity, written $\sim_{\mathrm{s}}$, is the union of all symbolic bisimulations.

Symbolic bisimilarity is sound with respect to concrete bisimilarity. The structure of the soundness proof was described in [11], details can be found in [10].

Theorem 3.20 *For all processes P, Q, and symbolic environments se such that $se \vdash P \sim_{\mathrm{s}} Q$ we have that $\mathbf{C}^B_{\sigma,\rho}(se) \vdash P\sigma \sim_{\mathrm{h}} Q\rho$ for all $B \subset \mathcal{N}$ with $\mathrm{fn}(P,Q) \cap B = \emptyset$ and substitution pairs (σ, ρ) satisfying $se \vdash \sigma \leftrightarrow_B \rho$.*

By virtue of allowing decompositions, symbolic bisimilarity is complete with respect to concrete hedged bisimilarity.

Theorem 3.21 *Assume that se, P, Q are such that se is consistent and $\mathbf{C}^B_{\sigma,\rho}(se) \vdash P\sigma \sim_{\mathrm{h}} Q\rho$ whenever $se \vdash \sigma \leftrightarrow_B \rho$ with $B \cap \mathrm{fn}(P,Q) = \emptyset$. Then $se \vdash P \sim_{\mathrm{s}} Q$.*

Proof. The set $\mathcal{R} = \{(se, P, Q) \mid se$ is consistent and $\mathbf{C}^B_{\sigma,\rho}(se) \vdash P\sigma \sim_{\mathrm{h}} Q\rho$ whenever $se \vdash \sigma \leftrightarrow_B \rho$ with $B \cap \mathrm{fn}(P,Q) = \emptyset\}$ is a symbolic bisimulation. The proof uses Lemma 3.17 at every transition. □

4 Examples

The processes in the following examples are taken from [11], where they were given as examples of the incompleteness of the earlier version of symbolic bisimilarity (lacking distinctions) proposed in that paper. All these examples start from the same symbolic environment $se := (\{(a,a) \mapsto 0\}, \emptyset, (tt, tt))$. Since se has no variables, it has the unique solution $h := \mathbf{C}^{\emptyset}_{\epsilon,\epsilon}(\{(a,a) \mapsto 0\}) = \{(a,a)\}$. We assume that x, y, z, a, k, n are pair-wise different wherever they occur below. The first example shows how decompositions permit a simple case split.

Example 4.1 Let $P_1 := a(x).\overline{a}\langle a\rangle$ and $Q_1 := a(x).Q_1'$ with $Q_1' := ([\,x = a\,]\overline{a}\langle a\rangle + \neg[\,x = a\,]\overline{a}\langle a\rangle)$. Then $se \vdash P_1 \sim_{\mathrm{s}} Q_1$. Specifically, the symmetric closure of the set

$$\begin{aligned}
\mathcal{R} &:= \{(se, P_1, Q_1), (se_1, \overline{a}\langle a\rangle, Q_1'), (se_2, \mathbf{0}, \mathbf{0}), (se_3, \mathbf{0}, \mathbf{0}) \mid x, y, z \in \mathcal{V}\} \text{ where}\\
se_1 &:= (\{(a,a) \mapsto 0\}, \{x \mapsto 1, y \mapsto 1\}, ([\,y = a\,], [\,y = a\,]))\\
se_2 &:= (\{(a,a) \mapsto 0\}, \{x \mapsto 1, y \mapsto 1, z \mapsto 2\}, ([\,x = a\,] \wedge [\,y = a\,] \wedge [\,z = a\,],\\
&\qquad [\,x = a\,] \wedge [\,y = a\,] \wedge [\,z = a\,]))\\
se_3 &:= (\{(a,a) \mapsto 0\}, \{x \mapsto 1, y \mapsto 1, z \mapsto 2\}, ((\neg[\,x = a\,]) \wedge [\,y = a\,] \wedge [\,z = a\,],\\
&\qquad (\neg[\,x = a\,]) \wedge [\,y = a\,] \wedge [\,z = a\,]))
\end{aligned}$$

is a symbolic bisimulation. We consider $(se_1, \overline{a}\langle a\rangle, Q_1')$. The symbolic transition $P \xrightarrow[[\,a:\mathcal{N}\,]\wedge[\,a:\mathcal{M}\,]]{\overline{a}\langle a\rangle}_s \mathbf{0}$ is possible and detectable: Letting $\sigma = \{a/x\}\{a/y\}$ we have $se_1 \vdash \sigma \leftrightarrow_\emptyset \sigma$, $a \in \pi_1(\mathbf{C}^{\emptyset}_{\sigma,\sigma}(se_1)) = \{a\}$ and $[\![([\,a:\mathcal{N}\,] \wedge [\,a:\mathcal{M}\,])\sigma]\!]$.

We choose $\{se_2, se_3\}$ as a decomposition of $(\{(a,a) \mapsto 0\}, \{x \mapsto 1, y \mapsto 1, z \mapsto 2\}, ([\,y = a\,] \wedge [\,a:\mathcal{N}\,] \wedge [\,a:\mathcal{M}\,] \wedge [\,z = a\,], [\,y = a\,]))$: se_2 and se_3 are both consistent since they are symmetric, and for all $\rho : \{x, y, z\} \to \mathcal{M}$ we have either $[\![[\,x = a\,]\rho]\!]$ or $[\![\neg[\,x = a\,]\rho]\!]$.

Considering se_2, $Q_2' \xrightarrow[[\,a:\mathcal{N}\,]\wedge[\,a:\mathcal{M}\,]\wedge[\,x=a\,]]{\overline{a}\langle a\rangle}_s \mathbf{0}$ where trivially $se_2 \vdash tt \leftrightarrow [\,a:\mathcal{N}\,] \wedge [\,a:\mathcal{M}\,] \wedge [\,x = a\,] \wedge [\,z = a\,]$.

Similarly, $Q_2' \xrightarrow[[a:\mathcal{N}]\wedge[a:\mathcal{M}]\wedge\neg[x=a]]{\overline{a}\langle a\rangle}_s \mathbf{0}$ with
$se_3 \vdash tt \leftrightarrow [a:\mathcal{N}] \wedge [a:\mathcal{M}] \wedge (\neg[x=a]) \wedge [z=a]$.

In general, symbolic bisimulations let us postpone the "instantiation" of input variables until the moment they are actually used. In the following example, the variable x is instead constrained through use of decomposition.

Example 4.2 Let

$$\begin{aligned} P_2 &:= a(x).P_2' & P_2' &:= (\nu c)\,(\overline{c}\langle c\rangle \mid c(z) \mid c(z).[x=a]\overline{a}\langle a\rangle) \\ Q_2 &:= a(x).Q_2' & Q_2' &:= (\nu c)\,(\overline{c}\langle c\rangle \mid c(z) \mid [x=a]c(z).\overline{a}\langle a\rangle). \end{aligned}$$

Then $se \vdash P_2 \sim_{\mathrm{s}} Q_2$. Similarly to before, the symmetric closure of the set

$$\begin{aligned} \mathcal{R} := &\{(se, P_1, Q_1)\} \\ &\cup\{(se_1, \overline{a}\langle a\rangle, Q_1') \mid x, y \in \mathcal{V}\} \\ &\cup\{(se_2, \mathbf{0}, \mathbf{0} \mid \neg[x=a]\overline{a}\langle a\rangle) \mid x, y, z \in \mathcal{V}\} \\ &\cup\{(se_3, \mathbf{0}, [x=a]\overline{a}\langle a\rangle \mid \mathbf{0}) \mid x, y, z \in \mathcal{V}\} \end{aligned}$$

where

$$\begin{aligned} se_1 &:= (\{(a,a)\mapsto 0\}, \{x\mapsto 1, y\mapsto 2\}, ([y=a], [y=a])) \\ se_2 &:= (\{(a,a)\mapsto 0\}, \{x\mapsto 1, y\mapsto 2, z\mapsto 3\}, ([x=a]\wedge[y=a]\wedge[z=a], \\ &\qquad [x=a]\wedge[y=a]\wedge[z=a])) \\ se_3 &:= (\{(a,a)\mapsto 0\}, \{x\mapsto 1, y\mapsto 2, z\mapsto 3\}, ((\neg[x=a])\wedge[y=a]\wedge[z=a], \\ &\qquad (\neg[x=a])\wedge[y=a]\wedge[z=a])) \end{aligned}$$

is a symbolic bisimulation.

Orthogonally to the possibility to decompose, the symbolic bisimilarity now also imposes the necessary and sufficient constraints for the environment to detect the process action.

Example 4.3 Let

$$\begin{aligned} P_3 &:= a(x).(\nu k)\,\overline{a}\langle \mathrm{E}_k(x)\rangle.(\nu n)\,\overline{a}\langle \mathrm{E}_{\mathrm{E}_k(a)}(n)\rangle.\overline{n}\langle a\rangle \\ Q_3 &:= a(x).(\nu k)\,\overline{a}\langle \mathrm{E}_k(x)\rangle.(\nu n)\,\overline{a}\langle \mathrm{E}_{\mathrm{E}_k(a)}(n)\rangle.[x=a]\overline{n}\langle a\rangle. \end{aligned}$$

Then $se \vdash P_3 \sim_{\mathrm{s}} Q_3$: After the first three transitions we have the symbolically hedged process pair $(se', \overline{n}\langle a\rangle, [x=a]\overline{n}\langle a\rangle)$ where

$$\begin{aligned} se' &:= (th', tv', ((\nu\{k\})\,\phi', (\nu\{k\})\,\phi') \\ th' &:= (\{(a,a)\mapsto 0, (\mathrm{E}_k(x), \mathrm{E}_k(x))\mapsto 2, (\mathrm{E}_{\mathrm{E}_k(a)}(n), \mathrm{E}_{\mathrm{E}_k(a)}(n))\mapsto 3\} \\ tv' &:= \{x\mapsto 1, y_1\mapsto 1, y_2\mapsto 2, y_3\mapsto 3\} \\ \phi' &:= [y_1=a]\wedge[y_2=a]\wedge[y_3=a] \end{aligned}$$

The symbolic transitions of $\overline{n}\langle a\rangle$ and $[x=a]\overline{n}\langle a\rangle$ are

$$\overline{n}\langle a\rangle \xrightarrow[[n:\mathcal{N}]\wedge[a:\mathcal{M}]]{\overline{n}\langle a\rangle}_s \mathbf{0} \qquad\qquad [x=a]\overline{n}\langle a\rangle \xrightarrow[[n:\mathcal{N}]\wedge[a:\mathcal{M}]\wedge[x=a]]{\overline{n}\langle a\rangle}_s \mathbf{0}$$

Let $\sigma := \{^a/_x\}$. As $se' \vdash \sigma \leftrightarrow_\emptyset \sigma$ and $\mathbf{C}^\emptyset_{\sigma,\sigma}(th') = \{(a,a),(\mathrm{E}_k(a),\mathrm{E}_k(a)),(n,n)\}$, we have that $n \in \pi_1(\mathbf{C}^\emptyset_{\sigma,\sigma}(th'))$, so the transition of $\overline{n}\langle a\rangle$ must be simulated by $[\,x = a\,]\overline{n}\langle a\rangle$. The environment after the step is
$se'' := (th', tv' \cup \{z \mapsto 5\}, ((\nu\{k,n\})\,\phi' \wedge [\,z = n\,], (\nu\{k,n\})\,\phi' \wedge [\,z = n\,]))$.

We need to show that $se'' \vdash tt \leftrightarrow [\,n : \mathcal{N}\,] \wedge [\,a : \mathcal{M}\,] \wedge [\,x = a\,]$, i.e., that $\rho'(x) = a$ whenever $se'' \vdash \sigma' \leftrightarrow_B \rho'$. First note that $[\![(\phi' \wedge [\,z = n\,])\rho]\!]$ iff
$a = \rho(y_1) = \rho(y_2) = \rho(y_3)$ and $\rho(z) = n$; we let $\rho = \{^a/_{y_1}\}\{^a/_{y_2}\}\{^a/_{y_3}\}\{^n/_z\}$.

Assume that $\sigma' = \{^M/_x\}$ and $\rho' = \{^N/_x\}$ such that $se'' \vdash \rho\sigma' \leftrightarrow_B \rho\rho'$. We let $h' = \{(a,a),(\mathrm{E}_k(M),\mathrm{E}_k(N)),(\mathrm{E}_{\mathrm{E}_k(a)}(n),\mathrm{E}_{\mathrm{E}_k(a)}(n))\}$. In order to have $\rho(z) = n$ we must have $(n,n) \in \mathcal{S}(\mathbf{C}^B_{\rho\sigma',\rho\rho'}(th')) = \mathcal{S}(\mathcal{I}(h' \cup \mathrm{Id}_B))$. Since $\{k,n\}$ is restricted we cannot have $k, n \in B$.

Then the only way to derive $(n,n) \in \mathcal{A}(h' \cup \mathrm{Id}_B)$ is by generating $(\mathrm{E}_k(a),\mathrm{E}_k(a)) \in \mathcal{SA}(h' \cup \mathrm{Id}_B)$ to analyze $(\mathrm{E}_{\mathrm{E}_k(a)}(n),\mathrm{E}_{\mathrm{E}_k(a)}(n))$. Since we cannot derive $(k,k) \in \mathcal{SA}(h' \cup \mathrm{Id}_B)$ we must have $(\mathrm{E}_k(a),\mathrm{E}_k(a)) \in \mathcal{A}(h' \cup \mathrm{Id}_B)$. This is the case iff $M = a = N$, yielding $\sigma' = \{^a/_x\} = \rho'$.

Finally, se'' is concretizable since $\mathrm{dom}(th') \subset \mathcal{M} \times \mathcal{M}$ and consistent since it is symmetric.

5 Conclusions

We have given a smooth extension of the message algebra of the spi calculus, treating complex keys and public-key cryptography in a uniform fashion. We have also extended our pre-existing notion of symbolic bisimilarity for the spi calculus [11], making it sound and complete with respect to concrete hedged bisimilarity.

However, the issues of finding appropriate decompositions and deciding symbolic consistency still remain. A promising step in this direction is due to Baudet [6], who studied the symbolic consistency problem in the setting of subterm-convergent rewrite theories, giving an NP algorithm for symbolic consistency for the case of guards without disjunction and negation.

References

[1] Abadi, M. and V. Cortier, *Deciding knowledge in security protocols under equational theories*, Theoretical Computer Science **367** (2006), pp. 2–32.

[2] Abadi, M. and C. Fournet, *Mobile values, new names, and secure communication*, in: *Proceedings of POPL '01*, 2001, pp. 104–115.

[3] Abadi, M. and A. D. Gordon, *A bisimulation method for cryptographic protocols*, Nordic Journal of Computing **5** (1998), pp. 267–303.

[4] Abadi, M. and A. D. Gordon, *A calculus for cryptographic protocols: The Spi calculus*, Journal of Information and Computation **148** (1999), pp. 1–70.

[5] Baldamus, M., J. Parrow and B. Victor, *A fully abstract encoding of the pi-calculus with data terms*, in: *Proceedings of ICALP '05*, LNCS **3580** (2005), pp. 1202–1213.

[6] Baudet, M., "Sécurité des protocoles cryptographiques : aspects logiques et calculatoires," Ph.D. thesis, École Normale Supérieure de Cachan (2007).

[7] Blanchet, B., *An efficient cryptographic protocol verifier based on Prolog rules*, in: *Proceedings of CSFW'01* (2001), pp. 82–96.

[8] Boreale, M., "Process Algebraic Theories for Mobile Systems," Ph.D. thesis, Università degli Studi di Roma "La Sapienza" (1995).

[9] Boreale, M., R. De Nicola and R. Pugliese, *Proof techniques for cryptographic processes*, in: *Proceedings of LICS '99*, IEEE (1999), pp. 157–166.

[10] Borgström, J., "Equivalences and Calculi for Formal Verification of Cryptographic Protocols," Ph.D. thesis, EPFL (2008).

[11] Borgström, J., S. Briais and U. Nestmann, *Symbolic bisimulation in the spi calculus*, in: *Proceedings of CONCUR '04*, LNCS **3170** (2004), pp. 161–176.

[12] Borgström, J. and U. Nestmann, *On bisimulations for the spi calculus*, Mathematical Structures in Computer Science **15** (2005), pp. 487–552.

[13] Delaune, S., S. Kremer and M. D. Ryan, *Coercion-resistance and receipt-freeness in electronic voting*, in: *Proceedings of CSFW'06* (2006), pp. 28–39.

[14] Delaune, S., S. Kremer and M. D. Ryan, *Symbolic bisimulation for the applied pi-calculus*, in: *Proceedings of FSTTCS'07*, LNCS **4855** (2007), pp. 133–145.

[15] Hennessy, M. and H. Lin, *Symbolic bisimulations*, Theoretical Computer Science **138** (1995), pp. 353–389.

[16] Hüttel, H., *Deciding framed bisimilarity*, in: *Pre-proceedings of INFINITY '02*, 2002, pp. 1–20.

[17] Johansson, M. and B. Victor, *A fully abstract symbolic semantics for the applied pi-calculus* (2007), unpublished manuscript.

[18] Milner, R., J. Parrow and D. Walker, *A calculus of mobile processes, part I/II*, Journal of Information and Computation **100** (1992), pp. 1–77.

[19] Paulson, L. C., *The inductive approach to verifying cryptographic protocols*, Journal of Computer Security **6** (1998), pp. 85–128.