A Testing Theory for a Higher-Order Cryptographic Language[⋆]
(Extended Abstract)

Vasileios Koutavas and Matthew Hennessy

Trinity College Dublin
{Vasileios.Koutavas, Matthew.Hennessy}@scss.tcd.ie

Abstract. We study a higher-order concurrent language with cryptographic primitives, for which we develop a sound and complete, first-order testing theory for the preservation of safety properties. Our theory is based on co-inductive set simulations over transitions in a first-order Labelled Transition System. This keeps track of the knowledge of the observer, and treats transmitted higher-order values in a symbolic manner, thus obviating the quantification over functional contexts. Our characterisation provides an attractive proof technique, and we illustrate its usefulness in proofs of equivalence, including cases where bisimulation theory does not apply.

1 Introduction

The verification of higher-order distributed systems that employ security protocols is now more than ever relevant to software development. Extensions of the π-calculus [18] with cryptographic primitives, such as the spi-calculus [3] and the applied π-calculus [1], have provided an effective framework for modelling security and authentication protocols of first-order systems [3, 1, 4, 8, 2, 5, 9, 6, 11, 10]. It is only natural that similar extensions to the *higher-order π-calculus* (HOπ) [19] would be equally effective for modelling and verifying security and authentication protocols of higher-order systems, such as distributed systems in which code is communicated between principals over public channels [17, 25].

This paper is inspired by the work of Maffeis et al. [17], where the focus is in the safety of higher-order authentication protocols. The authors use a higher-order version of the spi-calculus, augmented with a combination of static and dynamic typing that enables the use of untrusted code in the dynamic verification of authentication policies. They call this "Code-Carrying Authorisation".

Our goal is to develop behavioural theories for such a language, and in this paper we make the first step by studying safety in the language HOspi, a version of HOπ augmented with the symmetric cryptographic primitives of the spi-calculus and an extremely simple type system. To the extent of our knowledge, this is the first theory of safety for such a language.

[⋆] This research was supported by SFI project SFI 06 IN.1 1898.

Even in HOspi there are examples of systems that use Code-Carrying Authorisation similar to those in [17]. Consider a rather simple part of a conference server that expects the submission of a review for a particular paper by a particular reviewer Rev_1 or any legitimate delegate reviewer:

$$\begin{aligned}\mathsf{Conf} = \mathtt{inp}\, \mathit{subm}(x_{subm}).\, \mathtt{dec}\, x_{subm}\, \mathtt{as}\, \{\!|\, (x_r, f_{prf})\,|\!\}_{s_p} \\ \mathtt{in}\, \nu a.\, \mathtt{fork}\, f_{prf}\,(\{\!|\, a\,|\!\}_{s_{rev_1}}).\, \mathtt{inp}\, a().\, \mathtt{outp}\, \mathit{ok}(x_r).\, \mathbb{0}\end{aligned}$$

The paper has been associated by the server to key s_p and the reviewer to key s_{rev_1}. After receiving a possible submission on *subm*, the server verifies that the sender had access to the paper by successfully decrypting the input message with s_p. The message contains the review (bound to x_r) and a function (bound to f_{prf}) which can be used to prove that the review came from Rev_1 or by a delegate reviewer. To do this, the server generates a fresh channel a, encrypts it with the key s_{rev_1}, and gives it as an argument to the function. The legitimacy of the review is verified by an input on a, which can only be matched by an output in the function bound to f_{prf}, after it successfully decrypts its argument.

Reviewer Rev_1 can submit a review by the following code:

$$\begin{aligned}\mathsf{Rev}_1 &= \mathtt{outp}\, \mathit{subm}(\{\!|\, r, \mathsf{F}_{prf_1}\,|\!\}_{s_p}).\, \mathbb{0} \\ \text{where }\ \mathsf{F}_{prf_1} &= \lambda x.\, \mathtt{dec}\, x\, \mathtt{as}\, \{\!|\, z\,|\!\}_{s_{rev_1}}\, \mathtt{in}\, \mathtt{outp}\, z().\, \mathbb{0}\end{aligned}$$

Alternatively, Rev_1 can delegate the review of the paper to Rev_2 by giving to the latter access to the paper (i.e. the key s_p) and a function that can be used to prove the delegation:

$$\begin{aligned}\mathsf{Rev}_1' &= \mathtt{outp}\, \mathit{deleg}(\{\!|\, s_p, \mathsf{F}_{del_{12}}\,|\!\}_{s_{rev_2}}).\, \mathbb{0} \\ \mathsf{F}_{del_{12}} &= \lambda(x,y).\, \mathtt{dec}\, x\, \mathtt{as}\, \{\!|\, z\,|\!\}_{s_{rev_1}}\, \mathtt{in}\, \mathtt{outp}\, y(\{\!|\, z\,|\!\}_{s_{rev_2}}).\, \mathbb{0}\end{aligned}$$

When this function is applied to a challenge $\{\!|\, a\,|\!\}_{s_{rev_1}}$ bound to x and a continuation channel bound to y, it sends the challenge $\{\!|\, a\,|\!\}_{s_{rev_2}}$ on y. Hence, Rev_2 can submit a review using the following code:

$$\begin{aligned}\mathsf{Rev}_2 &= \mathtt{inp}\, \mathit{deleg}(x_{deleg}).\, \mathtt{dec}\, x_{deleg}\, \mathtt{as}\, \{\!|\, x_p, f_{del_{12}}\,|\!\}_{s_{rev_2}} \\ &\qquad\qquad \mathtt{in}\, \mathtt{outp}\, \mathit{subm}(\{\!|\, r, \mathsf{F}_{prf_2}\,|\!\}_{x_p}).\, \mathbb{0} \\ \mathsf{F}_{prf_2} &= \lambda x.\, \nu b.\, \mathtt{fork}\, f_{del_{12}}(x,b).\, \mathtt{inp}\, b(y).\, \mathtt{dec}\, y\, \mathtt{as}\, \{\!|\, z\,|\!\}_{s_{rev_2}}\, \mathtt{in}\, \mathtt{outp}\, z().\, \mathbb{0}\end{aligned}$$

One would want to prove that the system where Rev_1 submits a review is equivalent to the system where the review is delegated to Rev_2. This can be done by proving that the system

$$\mathsf{Sys}_1 = \nu s_p, s_{rev_1}, s_{rev_2}.\, (\mathsf{Conf} \mid \mathsf{Rev}_1 \mid \mathsf{D})$$

is observationally equivalent to

$$\mathsf{Sys}_2 = \nu s_p, s_{rev_1}, s_{rev_2}.\, (\mathsf{Conf} \mid \mathsf{Rev}_1' \mid \mathsf{Rev}_2)$$

where D generates dummy traffic on channel *deleg*.

It is important therefore to develop techniques for proving the equivalence of higher-order concurrent systems with cryptographic primitives. These techniques depend on the particular choice of semantic equivalence, ranging from branching-time equivalences, such as *reduction barbed congruence* [25, 1, 4, 13] and *bisimulation equivalence* [2], to linear-time equivalences, such as *may-testing equivalence* [3, 4, 12] and must-testing equivalence [12]. We aim at developing proof techniques that are adaptable to many semantic equivalences;[1] here we focus on *safe equivalence*, which is closely related to may-testing equivalence [7].

There are two main previous approaches that could potentially be used for proving safe equivalence for HOspi. The first would be to adapt Sangiorgi's well-known translation from the HOπ with finite types into the standard π-calculus [20], based on *triggers*, thereby in principle allowing first-order proof principles to be applied to HOspi. However it is unclear how the translation could be adapted to HOspi in a way that will be fully abstract and useful in proofs of equivalence. The second approach would be to make use of *environmental bisimulations* [23], which has been shown to provide a sound, complete and useful proof technique for reduction barbed congruence in Applied HOπ, a higher-order concurrent language with cryptographic primitives [25]. However, environmental bisimulations do not provide a complete proof principle for linear-time semantic equivalences, such as safe equivalence.

Here we pursue an alternative strategy similar to that of Jeffrey and Rathke [14], developing a proof technique which works at the level of a Labelled Transition System (LTS) for HOspi. The success of LTS-based reasoning for first-order calculi makes us confident that this approach is adaptable to semantic equivalences other than safe equivalence, and to higher-order concurrent languages with richer type-systems, as that by Maffeis et al. [17].

Configurations in our LTS record the interaction of the observer with the process, as well as the knowledge of the observer at every step of this interaction. This is similar in spirit to the environments used in environmental bisimulations [23, 25] and in proof techniques for first-order cryptographic calculi [2, 4]. Our LTS is first-order because it employs a *symbolic treatment* of higher-order values generated by the context, which obviates the need for quantification over functional contexts. Our approach is informed by the translation to triggers but avoids the complexities and incompleteness issues of a potential translation to a first-order language. We believe this to be the first first-order semantics for a higher-order language with cryptographic primitives.

We develop a sound and complete first-order co-inductive proof principle for safe equivalence for HOspi processes in terms of novel *set simulations.* These are essentially simulations over our first-order LTS between configurations and sets of configurations.

The symbolic treatment of functions in the LTS is a sound approach for HOspi because the language allows attackers to only apply functions or test them for equality. A variation of the language and theory that can express more

[1] For example semantic equivalences that can be used to prove security properties such as safety and non-interference.

Syntax

$$\begin{array}{rcll}
x, y, f & \in \mathsf{Var} & a, b, c, n, k, r \in \mathsf{Name} \qquad u, v \in \mathsf{Var} \cup \mathsf{Name} & \\
T & ::= & \mathsf{Nm} \mid \mathsf{Pr} \mid T \times T \mid \mathsf{Enc}(T) & \mathsf{Type} \\
P, Q & ::= & \mathbb{0} \mid \mathtt{outp}\, u(V{:}T).\, P \mid \mathtt{inp}\, u(x{:}T).\, P \mid (P \mid P) \mid \nu n.\, P \mid \mathtt{fork}\, V(u).\, P & \mathsf{Proc} \\
 & \mid & \mathtt{if}\, V = V\, \mathtt{then}\, P\, \mathtt{else}\, P \mid \mathtt{let}\, f = \lambda x.\, P\, \mathtt{in}\, P & \\
 & \mid & \mathtt{match}\, V\, \mathtt{as}\, (x, y)\, \mathtt{in}\, P \mid \mathtt{decr}\, V\, \mathtt{as}\, \{\!|\, x \,|\!\}_u\, \mathtt{in}\, P\, \mathtt{else}\, P & \\
U, V, W & ::= & x \mid n \mid (V, V) \mid \lambda_r x.\, P \mid \{\!|\, V \,|\!\}_k & \mathsf{Val}
\end{array}$$

Reduction Semantics

$$\begin{array}{llll}
\mathtt{outp}\, c(V{:}T).\, P \mid \mathtt{inp}\, c(x{:}T).\, Q & \to P \mid Q\{V/x\} & & (\textsc{Rcomm}) \\
\mathtt{decr}\, \{\!|\, V \,|\!\}_k\, \mathtt{as}\, \{\!|\, x \,|\!\}_k\, \mathtt{in}\, P\, \mathtt{else}\, Q & \to P\{V/x\} & & (\textsc{Rdec-S}) \\
\mathtt{decr}\, \{\!|\, V \,|\!\}_l\, \mathtt{as}\, \{\!|\, x \,|\!\}_k\, \mathtt{in}\, P\, \mathtt{else}\, Q & \to Q & \text{if } k \neq l & (\textsc{Rdec-F}) \\
\mathtt{match}\, (V, U)\, \mathtt{as}\, (x, y)\, \mathtt{in}\, P & \to P\{V/x, U/y\} & & (\textsc{Rmtch}) \\
\mathtt{let}\, f = \lambda x.\, P\, \mathtt{in}\, Q & \to \nu r.\, Q\{\lambda_r x.\, P/f\} & \text{if } r \notin fn(P, Q) & (\textsc{Rlet}) \\
\mathtt{fork}\, (\lambda_r x.\, P)(n).\, Q & \to P\{n/x\} \mid Q & & (\textsc{Rapp}) \\
\mathtt{if}\, U = V\, \mathtt{then}\, P\, \mathtt{else}\, Q & \to P & \text{if } \vdash equal(U, V) & (\textsc{Rif-T}) \\
\mathtt{if}\, U = V\, \mathtt{then}\, P\, \mathtt{else}\, Q & \to Q & \text{if } \nvdash equal(U, V) & (\textsc{Rif-F}) \\
P_1 \mid Q & \to P_2 \mid Q & \text{if } P_1 \to P_2 & (\textsc{Rpar}) \\
\nu a.\, P_1 & \to \nu a.\, P_2 & \text{if } P_1 \to P_2 & (\textsc{Rnu}) \\
P_1 & \to P_2 & \text{if } P_1 \equiv P_1' \to P_2' \equiv P_2 & (\textsc{Rcng})
\end{array}$$

Fig. 1. Syntax and reduction semantics of HOspi.

sophisticated attackers, for example attackers that can learn the complete source-level text of transmitted code, as in Applied HOπ [25], would also be possible, but not first-order, and it would be closer to the theory of Sato and Sumii [25].

The language and reduction semantics are given in Section 2 while the LTS of configurations is in Section 3. In Section 5 we explain set simulations and state their soundness and completeness. We illustrate the usefulness of set simulations in Section 6, by giving coinductive proofs relating higher-order systems, such as those discussed above. The paper concludes with discussion of related and future work in Section 7.

2 Semantics of HOspi

We study the language HOspi, a higher-order concurrent calculus with primitives for symmetric-key cryptography, similar to those of the spi-calculus [3]. Public-key cryptography can be added to the language without any significant change to the semantics and the theory of the following sections. The syntax and reduction semantics of HOspi are shown in Figure 1. We employ the standard π-calculus abbreviations; $(\equiv)$ is the standard π-calculus structural equivalence.

We use a lightweight dynamic type system for HOspi which helps streamline the presentation of our theory. *Closed values* in HOspi, ranged over by V and U, are either names of type Nm, process abstractions $(\lambda_r x.\, P)$ of type Pr, pairs of type $T \times T'$, and messages encrypted with a name ($\{\!|V|\!\}_k$) of type $\mathsf{Enc}(T)$. Abstractions, for simplicity, can be applied to names with the construct $\mathtt{fork}\,(\lambda_r x.\, P)(n).\, Q$ and fork the body P as a new process in parallel with the continuation Q (Rapp). We write $\mathtt{fork}\,(\lambda_r x.\, P)(n)$ when the continuation is $\mathbf{0}$.

Communication happens by synchronisation of an output ($\mathtt{outp}\, c(V{:}T).\, P$) and an input ($\mathtt{inp}\, c(x{:}T).\, Q$) over a common channel c (Rcomm). The dynamic type system guarantees that communication happens only when the transmitted and the expected value have the same type. Restriction ($\nu n.\, P$) encodes privacy of information, such as keys, which can be extruded by communication.

In a cryptographic calculus it is important for processes or attackers to be able to detect the retransmission of messages. This creates the requirement for an equality semantics defined at any type, including function type (Rif-T, Rif-F). We use an equality semantics, denoted by *equal* and present in languages like Scheme, which identifies only functional values representing the same object, identical names, and pairs and encrypted packages with equal components. Object identity for functional values is possible by requiring each such value to be generated by the construct $\mathtt{let}\, f = \lambda x.\, P\, \mathtt{in}\, Q$, which annotates the value with a fresh name r (Rlet). Equality at function type is thus reduced to equality of names. Such an equality construct is convenient for a symbolic treatment of functions in the following sections. We call a closed process with no functional values *source-level*; the reduction semantics of Figure 1 are defined only for processes derivable by a source-level process, which guarantees that functional values annotated with the same name were generated by a single `let` statement.

The remaining reduction rules are rather standard and deal with decryption of messages (Rdec-S, Rdec-F), the deconstruction of pairs (Rmtch), and π-calculus processes (Rpar, Rnu, Rcng). We omit the `else` branch of decryption if it is $\mathbf{0}$.

A safety property can be formulated as a safety test T^ω; a process which reports bad behaviour on a special channel ω. Let us write $R\downarrow_\omega$ whenever $R \equiv \nu\widetilde{n}.\, \mathtt{outp}\, \omega(V{:}T).\, R_1 \mid R_2$, and $\omega \notin \{\widetilde{n}\}$.

Definition 2.1 (Passing Safety Tests). *A process P passes a safety test T^ω, written P cannot T^ω, when $P \mid T^\omega \rightarrow^* R$ implies $R\not\downarrow_\omega$.*

Definition 2.2 (Safety Preservation). *Two source-level processes P and Q are related by $P \sqsubseteq_{\mathsf{safe}} Q$ when for all source-level tests T^ω, P cannot T^ω implies Q cannot T^ω. We use $P \simeq_{\mathsf{safe}} Q$ to denote the associated equivalence.*

The reader familiar with [7] will recognise this safe-preorder as the inverse of the well-known *may-testing* preorder. An important property of ($\sqsubseteq_{\mathsf{safe}}$), as with other contextual equivalences, is that it enables compositional reasoning.

Proposition 2.3 (Compositional Reasoning). *If $P \sqsubseteq_{\mathsf{safe}} Q$ then $P \mid R \sqsubseteq_{\mathsf{safe}} Q \mid R$ and $\nu n.\, P \sqsubseteq_{\mathsf{safe}} \nu n.\, Q$.*

3 First-Order Semantics

We describe the interaction between a process and an observer by transitions in an LTS. The LTS uses *configurations* that record the values transmitted from the process to the observer and vice versa, the knowledge of the observer, and the private knowledge of the process. The LTS is *first-order*: only the names that annotate abstractions are exchanged between the process and the observer, not the abstractions themselves. Below we explain the details of configurations and transitions in this LTS.

3.1 Configurations

An LTS configuration describes the state of a process interrogated by an observer and takes the form $\langle H \parallel K \parallel I \rangle \triangleright \mathcal{P}$ where

- K is the current knowledge of the observer, a finite set of names
- I is the current private knowledge of the process, the set of names known to the process $\mathcal{P}$ but not the observer
- H is the history of the interaction between the observer and the process, in which the messages exchanged are accessible via indices
- $\mathcal{P}$ is an extended process, which may contain pointers to values in the history.

The interrogation of the process by the observer proceeds by the exchange of messages between them, each message being recorded in H, and available for use in future interactions. We first explain how these interactions take place, and how they are recorded.

Example 3.1. First we consider the configuration

$$\langle H_0 \parallel K \parallel s, r \rangle \triangleright \mathtt{outp}\, c(\{\!|\, \lambda_r x.\, P_0 \,|\!\}_s).\, P_1$$

where the key s (and the annotation r) are private to the configuration. Suppose the channel name c is known to the observer; that is $c \in K$. Then the process can output the encrypted message on c, resulting in the configuration

$$\langle H_1 \parallel K \parallel s, r \rangle \triangleright P_1$$

where $H_1 = H_0, \kappa{\mapsto}\{\!|\, \lambda_r x.\, P_0 \,|\!\}_s$. The history is extended with a new entry, indexed by the fresh κ, which records the message received, $\{\!|\, \lambda_r x.\, P_0 \,|\!\}_s$; these indices κ of messages received from the process are taken from a distinct set of *output abstract names* (OAName) The observer now has access to this message via the index κ, but not to the contents since the key s is private to the process.

If $P_1 = \mathtt{outp}\, c(s).\, P_2$, it can send the key s to the observer and end up in the configuration

$$\langle H_1 \parallel K, s \parallel r \rangle \triangleright P_2$$

Here the history has not changed since it is only necessary to record the non-base values used in the interactions. But the knowledge of the observer has been extended with the key s. Now using this key the observer has access to the abstraction and may apply it to a name or use it in a message sent back to the process. However in the latter case, in order to maintain a clear interface between the observer and process, the abstraction itself is not used. For example if $P_2 = \mathtt{inp}\, c(x).\, P_3$, the above configuration can transition to

$$\langle H_2 \,\|\, K, s \,\|\, r\rangle \triangleright P_3\{\iota/x\}$$

where H_2 is the history $H_1, (\iota{\mapsto}\{\!|\, \kappa.\mathsf{decr}_s \,|\!\}_l)$, ι is a fresh index taken from a separate set of *input abstract names* (IAName) used for recording the messages sent from the observer to the process, and l is any key in the observers knowledge (K, s). But note that the process actually only receives the input index ι, and in the history the *observer pattern* $\kappa.\mathsf{decr}_s$ represents the actual abstraction $\lambda_r x.\, P_0$ previously received from the process. □

Thus, in a configuration the history H records the non-base messages exchanged, using $\kappa \in \mathsf{OAName}$ for those sent by the process, and $\iota \in \mathsf{IAName}$ for those received by the process. We use the following metavariables to range over these sets.

$$\begin{array}{ll} \iota \in \mathsf{IAName} & \alpha \in \mathsf{AName} = \mathsf{IAName} \cup \mathsf{OAName} \\ \kappa \in \mathsf{OAName} & \zeta \in \mathsf{AName} \cup \mathsf{Name} \end{array}$$

Moreover, as we have seen in the previous example, observers can access data bound to output names in the history using *observer patterns* in order to further interrogate the process. As we will see, the process can dually access data bound to input names in the history using *process patterns*. The structure of these patterns is:

$$\begin{array}{llll} \phi ::= \iota \mid \phi.\mathsf{op} & \mathsf{ProcPattern} & \mathsf{op} ::= 1 \mid 2 \mid \mathsf{decr}_s & \mathsf{Op} \\ \psi ::= \kappa \mid \psi.\mathsf{op} & \mathsf{ObsPattern} & \pi \;\in \mathsf{Pattern} = \mathsf{ProcPattern} \cup \mathsf{ObsPattern} & \end{array}$$

The process (observer) pattern ι (resp. κ) refers to the value in the history that is bound to that name; $\pi.i$ refers to the ith projection of the value referred to by π, and $\pi.\mathsf{decr}_s$ refers to the contents of a message encrypted with s and referred to by π.[2] Therefore we extend the syntax for values (AVal) and processes (AProc) to contain such patterns ($\mathcal{U}, \mathcal{V} ::= \ldots \mid \pi \quad \mathsf{AVal}$).

The meta-function $\mathit{fan}(\mathcal{P})$ gives the abstract names in $\mathcal{P}$, and $T(\alpha)$ gives the type of the abstract name α. Patterns are typed by the following rules.

$$\frac{}{\vdash \alpha : T(\alpha)} \qquad \frac{\vdash \pi : T_1 \times T_2}{\vdash \pi.i : T_i} \qquad \frac{\vdash \pi : \mathsf{Enc}(T)}{\vdash \pi.\mathsf{decr}_s : T}$$

The observer may use its accumulating knowledge K, together with the abstract values received from the process, occurring in the history H, to further the interrogation of the process. However, unlike [21, 23, 25], in order to keep

[2] The use of s in the pattern $\pi.\mathsf{decr}_s$ is only for quantifying over possible patterns, given a set of known keys.

the LTS first-order we severely restrict the ability of the observer to construct higher-order values. The soundness of our technique in Section 5 shows that this restriction does not compromise the possible observations made in the LTS.

Example 3.2. Consider the following configuration

$$\langle H_0 \parallel K \parallel \cdot\rangle \triangleright \mathtt{inp}\, c(x).\, \mathtt{inp}\, c(y).\, \mathtt{dec}\, x\, \mathtt{as}\, \{\!|\, z\, |\!\}_y\, \mathtt{in}\, \mathtt{fork}\, z(n)$$

where $c \in K$. The process is expecting an encrypted function on the channel c. In our LTS the observer can only supply a *symbolic* representation of an abstraction, denoted $\Diamond_r$, informally representing an arbitrary but unknown function annotated with the name r. So after the first input we get the configuration

$$\langle H_1 \parallel K, s, r \parallel \cdot\rangle \triangleright \mathtt{inp}\, c(y).\, \mathtt{dec}\, \iota_1\, \mathtt{as}\, \{\!|\, z\, |\!\}_y\, \mathtt{in}\, \mathtt{fork}\, z(n)$$

where $H_1 = H_0, (\iota_1 \mapsto \{\!|\, \Diamond_r\, |\!\}_s)$. The key s and annotation r are freshly generated by the observer and are added to the knowledge.

After the key s is passed to the process by the next input we get

$$\langle H_1 \parallel K, s, r \parallel \cdot\rangle \triangleright \mathtt{dec}\, \iota_1\, \mathtt{as}\, \{\!|\, z\, |\!\}_s\, \mathtt{in}\, \mathtt{fork}\, z(n)$$

The process can now use the key s to decrypt the message indexed by ι_1. However, in order to keep the process independent of the actual observer interrogating it (i.e. avoid the propagation of observer-generated non-base values in the process), our operational semantics will ensure that the successful decryption of pattern ι_1 will actually generate the process pattern $\iota_1.\mathsf{decr}_s$, and we obtain the configuration

$$\langle H_1 \parallel K, s, r \parallel \cdot\rangle \triangleright \mathtt{fork}\, (\iota_1.\mathsf{decr}_s)(n)$$

The pattern $\iota_1.\mathsf{decr}_s$ gives to the process access to the "function" $\Diamond_r$ in the history H_1. However, since this is purely symbolic its application to n cannot lead to any real computation; as we will see, this symbolic application is recorded by the LTS. □

Well-formed histories map abstract names to values of the same type. Moreover, indexed input values contain only observer patterns and indexed output values contain only process patterns, and all references to abstract names are to the left of each binding.

$$\begin{array}{l} \text{if } (\iota \mapsto \mathcal{V}) \in H \;\text{ then } \mathit{fan}(\mathcal{V}) \subset \mathsf{OAName} \\ \text{if } (\kappa \mapsto \mathcal{V}) \in H \;\text{ then } \mathit{fan}(\mathcal{V}) \subset \mathsf{IAName} \\ \text{if } H, \alpha \mapsto \mathcal{V}, H' \text{ then } \mathit{fan}(\mathcal{V}) \subseteq \mathit{dom}(H) \end{array}$$

Given a well-formed history H and a pattern π we define the partial dereferencing operation $H_K(\mathcal{V})$, relative to the knowledge K:

$$\begin{array}{ll} H_K(\mathcal{V}) = \mathcal{V} & \text{if } \; \mathcal{V} \notin \mathsf{Pattern} \\ H_K(\alpha) = \mathcal{V} & \text{if } \; (\alpha \mapsto \mathcal{V}') \in H \text{ and } H_K(\mathcal{V}') = \mathcal{V} \\ H_K(\pi.\mathsf{decr}_s) = \mathcal{V} & \text{if } \; H_K(\pi) = \{\!|\, \mathcal{V}'\, |\!\}_s,\; H_K(\mathcal{V}') = \mathcal{V}, \text{ and } s \in K \\ H_K(\pi.i) = H_K(\mathcal{V}_i) & \text{if } \; H_K(\pi) = (\mathcal{V}_1', \mathcal{V}_2') \text{ and } H_K(\mathcal{V}_i') = \mathcal{V}_i \end{array}$$

We write $H(\mathcal{V})$ when only the first two rules are used and $H_*(\mathcal{V})$ when the knowledge contains all names.

The closure of the observers knowledge K under decryption of messages in the history H is given by the construction $K^\star_H$.

Definition 3.3 (Knowledge closure). $n \in K^\star_H$ *if* $n \in K$ *or* $\exists\psi. H_{(K^\star_H)}(\psi) = n$.

In the rest of this paper will only consider *well-formed* configurations, and we will use $\mathcal{C}$ to range over them.

Definition 3.4. $\langle H \parallel K \parallel I\rangle \triangleright \mathcal{P}$ *is well-formed when:*

- *H is well-formed, and K is closed ($K^\star_H \subseteq K$)*
- Observer values use only K: $\forall(\iota\mapsto\mathcal{V}) \in H.\ fn(\mathcal{V}) \subseteq K$.
- Private and global names are distinct: $K \cap I = \emptyset$.
- Closed: $fv(codom(H), \mathcal{P}) = \emptyset$, $fn(codom(H), \mathcal{P}) \subseteq K \cup I$, *and* $fan(codom(H), P) \subseteq dom(H)$.
- Well-annotated: *if $\lambda_r x.\,\mathcal{P}$ is in the configuration then $r \in I$; if $\lambda_r x.\,\mathcal{P}'$ is also in the configuration then $\mathcal{P} = \mathcal{P}'$; if $\Diamond_r$ in the codomain of H then $r \in K$.*

3.2 Transitions

Our LTS defines labelled transitions between the configurations, briefly discussed informally in the examples of the previous section. The main transitions of the LTS are shown in Figure 2, subject to the well-formedness conditions for configurations.

Rule TOUT describes an output on channel c, labelled by $c!\zeta$. The channel name c must be in the knowledge of the observer and the effect of the action on the configuration is calculated using the auxiliary relation $outp(\mathcal{V} : T)$:

$$(n, \cdot, \{n\}) \in outp(n : \mathsf{Nm}) \qquad\qquad (\kappa, (\kappa\mapsto\mathcal{V}), \emptyset) \in outp(\mathcal{V} : T) \text{ if } T \neq \mathsf{Nm}$$

Thus, if the output value $\mathcal{V}$ is a name it is added directly to the observers knowledge; otherwise it is added to the history H via a new index. In both cases the output transition may allow the observer to discover previously private names known only to the process, by decrypting the current and previous messages. This knowledge extension of the observer is taken into account by the closure of the environment's knowledge $(\ )^\star_H$.

Rule TIN describes an input transition, labelled by $c?\zeta$; as for the output rule, the channel name c must be in the knowledge of the observer. The input value of type T is provided by the set $inp_{H,K,I}(T)$ as a tuple of an abstract or actual name ζ, a history extension H', and a knowledge extension K':

$$\begin{array}{lll}
(n,\ \cdot,\ \{n\}) & \in inp_{H,K,I}(\mathsf{Nm}) & \text{if } n \notin I \\
(\iota,\ (\iota\mapsto\Diamond_r),\ \{r\}) & \in inp_{H,K,I}(\mathsf{Pr}) & \\
\multicolumn{3}{l}{(\iota, (\iota\mapsto(H_1(\zeta_1), H_2(\zeta_2))), (K_1, K_2))} \\
 & \in inp_{H,K,I}(T_1 \times T_2) & \text{if } (\zeta_i, H_i, K_i) \in inp_{H,K,I}(T_i) \\
(\iota,\ (\iota\mapsto\{\!|\, H_1(\zeta)\, |\!\}_s),\ (K_1, K_2)) & \in inp_{H,K,I}(\mathsf{Enc}(T)) & \text{if } (\zeta, H_1, K_1) \in inp_{H,K,I}(T) \\
 & & \text{and } (s, \cdot, K_2) \in inp_{H,K,I}(\mathsf{Nm}) \\
(\iota,\ (\iota\mapsto\psi),\ \emptyset) & \in inp_{H,K,I}(T) & \text{if } \vdash H_K(\psi) : T \neq \mathsf{Nm}
\end{array}$$

$$\frac{(\zeta, H', K') \in \mathit{inp}_{H,K,I}(T) \qquad c \in K}{\langle H \parallel K \parallel I \rangle \rhd \mathtt{inp}\, c(x{:}T).\, \mathcal{P} \xrightarrow{c?\zeta} \langle H, H' \parallel K, K' \parallel I \rangle \rhd \mathcal{P}\{\zeta/x\}}\ \textsc{Tin}$$

$$\frac{(\zeta, H', K') \in \mathit{outp}(\mathcal{V}{:}T) \qquad I' = I \backslash (K, K')^{\star}_{H,H'} \qquad c \in K}{\langle H \parallel K \parallel I \rangle \rhd \mathtt{outp}\, c(\mathcal{V}{:}T).\, \mathcal{P} \xrightarrow{c!\zeta} \langle H, H' \parallel (K, K')^{\star}_{H,H'} \parallel I' \rangle \rhd \mathcal{P}}\ \textsc{Tout}$$

$$\frac{\begin{array}{l}\langle H \parallel K, I \parallel \cdot \rangle \rhd \mathcal{P} \xrightarrow{c!\zeta} \langle H, H' \parallel K' \parallel \cdot \rangle \rhd \mathcal{P}' \\ \langle H \parallel K, I \parallel \cdot \rangle \rhd \mathcal{Q} \xrightarrow{c?\zeta_{fr}} \langle H'' \parallel K'' \parallel \cdot \rangle \rhd \mathcal{Q}' \qquad \zeta_{fr} \notin \mathit{fan}(\mathcal{Q})\end{array}}{\langle H \parallel K \parallel I \rangle \rhd \mathcal{P} \mid \mathcal{Q} \xrightarrow{\tau} \langle H \parallel K \parallel I \rangle \rhd \mathcal{P}' \mid \mathcal{Q}'\{H'(\zeta)/\zeta_{fr}\}}\ \textsc{Tcomm}$$

$$\frac{H_K(\psi) = \lambda_r x.\, \mathcal{Q}}{\langle H \parallel K \parallel I \rangle \rhd \mathcal{P} \xrightarrow{\mathtt{app}(\psi,r,n)} \langle H \parallel K \parallel I \rangle \rhd \mathcal{P} \mid \mathcal{Q}\{n/x\}}\ \textsc{Tapp-Obs}$$

$$\frac{H_{K \cup I}(\phi) = \lozenge_r \qquad I' = I \backslash (K, n)^{\star}_{H}}{\langle H \parallel K \parallel I \rangle \rhd \mathtt{fork}\, \phi(n).\, \mathcal{P} \xrightarrow{\mathtt{sig}(r,n)} \langle H \parallel (K, n)^{\star}_{H} \parallel I' \rangle \rhd \mathcal{P}}\ \textsc{Tapp-}\lozenge$$

$$\frac{H_{K \cup I}(\mathcal{V}) = \lambda_r x.\, \mathcal{P}}{\langle H \parallel K \parallel I \rangle \rhd \mathtt{fork}\, \mathcal{V}(n).\, Q \xrightarrow{\tau} \langle H \parallel K \parallel I \rangle \rhd \mathcal{P}\{n/x\} \mid \mathcal{Q}}\ \textsc{Tapp-}\lambda$$

Fig. 2. Main LTS rules (omitting symmetric rules).

At type Nm, the input value is a name disjoint from the private names of the process I that can be either in K or fresh. At type Pr, the input can be the symbol $\lozenge_r$ and r is added in the knowledge K. The cases for $T_1 \times T_2$ and $\mathsf{Enc}(T)$ proceed by induction on the type, returning a singleton history extension and the union of the knowledge extension. At any non-base type, an output pattern can be used to reference a value that was previously sent to the observer.

Communication is achieved by rule Tcomm. Here, one of the processes inputs a fresh (actual or abstract) name, which is replaced by the value output by the other and may be indexed in the history. This is a τ-transition in which the observer does not participate; therefore the history H and knowledge K remain unchanged.

At any point in time the observer may choose to apply to a name n one of the abstractions reachable by an observer pattern ψ using keys from the knowledge K (Tapp-Obs). Intuitively this means that the abstraction in question $\lambda_r x.\, \mathcal{Q}$ was previously sent by the process to the observer, perhaps as the contents of an encrypted message, but which can be now decrypted with the current knowledge K. The resulting transition, labelled $\mathtt{app}(\psi, r, n)$, causes the application to run in parallel with the observed process.

Tapp-$\lozenge$ encodes the situation where the process applies an unknown abstraction that was created by the observer; such abstractions are only reachable by the process via a process pattern ϕ that uses keys from $K \cup I$. The effect of this rule is merely a signal $\mathtt{sig}(r, n)$ to the observer containing the annotation of the ab-

Tdecr-S

$$\frac{H \vdash \mathcal{V}.\mathsf{decr}_s \rightsquigarrow \mathcal{V}'}{\begin{array}{l}\langle H \parallel K \parallel I\rangle \rhd \mathtt{decr}\,\mathcal{V}\,\mathtt{as}\,\{\!|\,x\,|\!\}_S\,\mathtt{in}\,\mathcal{P}\,\mathtt{else}\,\mathcal{Q}\\ \quad\xrightarrow{\tau} \langle H \parallel K \parallel I\rangle \rhd \mathcal{P}\{\mathcal{V}'/x\}\end{array}}$$

Tdecr-F

$$\frac{H_{K\cup I}(\mathcal{V}) = \{\!|\,\mathcal{V}'\,|\!\}_l \qquad s \neq l}{\begin{array}{l}\langle H \parallel K \parallel I\rangle \rhd \mathtt{decr}\,\mathcal{V}\,\mathtt{as}\,\{\!|\,x\,|\!\}_S\,\mathtt{in}\,\mathcal{P}\,\mathtt{else}\,\mathcal{Q}\\ \quad\xrightarrow{\tau} \langle H \parallel K \parallel I\rangle \rhd \mathcal{Q}\end{array}}$$

Tcond-T

$$\frac{H \vdash \mathit{equal}(\mathcal{V}_1, \mathcal{V}_2)}{\begin{array}{l}\langle H \parallel K \parallel I\rangle \rhd \mathtt{if}\,\mathcal{V}_1 = \mathcal{V}_2\,\mathtt{then}\,\mathcal{P}\,\mathtt{else}\,\mathcal{Q}\\ \quad\xrightarrow{\tau} \langle H \parallel K \parallel I\rangle \rhd \mathcal{P}\end{array}}$$

Tcond-F

$$\frac{H \vdash \neg\mathit{equal}(\mathcal{V}_1, \mathcal{V}_2)}{\begin{array}{l}\langle H \parallel K \parallel I\rangle \rhd \mathtt{if}\,\mathcal{V}_1 = \mathcal{V}_2\,\mathtt{then}\,\mathcal{P}\,\mathtt{else}\,\mathcal{Q}\\ \quad\xrightarrow{\tau} \langle H \parallel K \parallel I\rangle \rhd \mathcal{Q}\end{array}}$$

Tmatch

$$\frac{H \vdash \mathcal{V}.1 \rightsquigarrow \mathcal{V}_1 \qquad H \vdash \mathcal{V}.2 \rightsquigarrow \mathcal{V}_2}{\begin{array}{l}\langle H \parallel K \parallel I\rangle \rhd \mathtt{match}\,\mathcal{V}\,\mathtt{as}\,(x_1, x_2)\,\mathtt{in}\,\mathcal{P}\\ \quad\xrightarrow{\tau} \langle H \parallel K \parallel I\rangle \rhd \mathcal{P}\{\mathcal{V}_1/x_1, \mathcal{V}_2/x_2\}\end{array}}$$

Tdef-λ

$$\frac{r \notin I \cup K}{\begin{array}{l}\langle H \parallel K \parallel I\rangle \rhd \mathtt{let}\,x = \lambda y.\,\mathcal{P}\,\mathtt{in}\,\mathcal{Q}\\ \quad\xrightarrow{\tau} \langle H \parallel K \parallel I, r\rangle \rhd \mathcal{Q}\{\lambda_r y.\,\mathcal{P}/x\}\end{array}}$$

Tpar-L

$$\frac{\langle H \parallel K \parallel I\rangle \rhd \mathcal{P} \xrightarrow{\mu} \langle H' \parallel K' \parallel I'\rangle \rhd \mathcal{P}'}{\begin{array}{l}\langle H \parallel K \parallel I\rangle \rhd \mathcal{P} \mid \mathcal{Q}\\ \quad\xrightarrow{\mu} \langle H' \parallel K' \parallel I'\rangle \rhd \mathcal{P}' \mid \mathcal{Q}\end{array}}$$

Tnu

$$\frac{a \notin I \cup K}{\langle H \parallel K \parallel I\rangle \rhd \nu a.\,\mathcal{P} \xrightarrow{\tau} \langle H \parallel K \parallel I, a\rangle \rhd \mathcal{P}}$$

Fig. 3. More LTS rules (omitting symmetric rules).

straction that was applied and the argument given. The resulting process is just the continuation of the application. As we will see, this transition is sufficient to reason about applications of input values and can be seen as a form of trigger semantics [19, 20, 14]. The process can also apply a process-generated abstraction, which is a τ-step in the LTS (Tapp-λ) since the application is unobservable.

The rest of the LTS rules are shown in Figure 3 and encode silent transitions and a congruence rule for parallel composition (Tpar-L). Rules Tdecr-S and Tdecr-F reduce a successful and an unsuccessful decryption using the following evaluation predicate:

$$\begin{array}{ll}H, K \vdash \{\!|\,\mathcal{V}\,|\!\}_S.\mathsf{decr}_s \rightsquigarrow \mathcal{V} & \\ H, K \vdash (\mathcal{V}_1, \mathcal{V}_2).i \rightsquigarrow \mathcal{V}_i & \\ H, K \vdash \phi.\mathsf{op} \rightsquigarrow \phi.\mathsf{op} & \text{if } \; H_K(\phi.\mathsf{op}) = \mathcal{V} \neq n\\ H, K \vdash \phi.\mathsf{op} \rightsquigarrow n & \text{if } \; H_K(\phi.\mathsf{op}) = n\end{array}$$

The important point in the rule Tdecr-S is that if the encrypted message $\mathcal{V}$ is a process pattern, that refers to some data in the history H, then the successful decryption returns not the actual contents of the message but rather another process pattern with which the contents can be extracted from the history.

Rule Tmatch reduces the decomposition of a pair using the same evaluation predicate, while rules Tcond-T and Tcond-F reduce conditionals using a standard equality predicate that dereferences all patterns:

$$\begin{array}{ll}
H \vdash \mathit{equal}(n, n) & \\
H \vdash \mathit{equal}(\Diamond_r, \Diamond_r) & \\
H \vdash \mathit{equal}(\lambda_r x.\, \mathcal{P}, \lambda_r x.\, \mathcal{Q}) & \\
H \vdash \mathit{equal}(\{\!|\, \mathcal{V} \,|\!\}_S, \{\!|\, \mathcal{V}' \,|\!\}_S) & \text{if } H \vdash \mathit{equal}(\mathcal{V}, \mathcal{V}') \\
H \vdash \mathit{equal}((\mathcal{V}_1, \mathcal{V}_2), (\mathcal{V}'_1, \mathcal{V}'_2)) & \text{if } H \vdash \mathit{equal}(\mathcal{V}_1, \mathcal{V}'_1) \text{ and } H \vdash \mathit{equal}(\mathcal{V}_2, \mathcal{V}'_2) \\
H \vdash \mathit{equal}(\pi, \mathcal{V}') & \text{if } H \vdash \mathit{equal}(H_*(\pi), \mathcal{V}') \\
H \vdash \mathit{equal}(\mathcal{V}, \pi) & \text{if } H \vdash \mathit{equal}(\mathcal{V}, H_*(\pi))
\end{array}$$

Bound names in the process are promoted to actual names at the level of the configuration by a τ-move (Tnu) that simplifies the handling of extrusion.

We write $\overset{\mu}{\Rightarrow}$ to mean the reflexive, transitive closure of $\xrightarrow{\tau}$, if $\mu = \tau$, and $\overset{\tau}{\Rightarrow}\xrightarrow{\mu}\overset{\tau}{\Rightarrow}$ otherwise. We call $\overset{t}{\Rightarrow}$ a *weak trace* of $\mathcal{C}$ if it is a sequence of non-τ actions $t = \mu_1 \ldots \mu_n$ and for some $\mathcal{C}'$, $\mathcal{C} \overset{\tau}{\Rightarrow}\xrightarrow{\mu_1}\overset{\tau}{\Rightarrow} \ldots \overset{\tau}{\Rightarrow}\xrightarrow{\mu_n}\overset{\tau}{\Rightarrow} \mathcal{C}'$.

The following propositions show that silent transitions do not change the history and knowledge of a configuration, weakly correspond to reduction steps of HOspi, and are invariant to transfer of knowledge between the process and the environment.

Proposition 3.5. *If* $\langle H_1 \,\|\, K_1 \,\|\, I_1 \rangle \rhd P \xrightarrow{\tau} \langle H_2 \,\|\, K_2 \,\|\, I_2 \rangle \rhd Q'$ *then* $H_1 = H_2$ *and* $K_1 = K_2$.

Proposition 3.6. *If* $P \to Q$ *then there exist* $\widetilde{a}$, $\widetilde{c}$, Q', *such that* $\mathit{fn}(P) \subseteq \widetilde{c}$, $Q \equiv \nu\widetilde{a}.\, Q'$ *and* $\langle \cdot \,\|\, \widetilde{c} \,\|\, \cdot \rangle \rhd P \xrightarrow{\tau}{}^{*} \langle \cdot \,\|\, \widetilde{c} \,\|\, \widetilde{a} \rangle \rhd Q'$.

Proposition 3.7. *If* $\langle \cdot \,\|\, \widetilde{c} \,\|\, \widetilde{a} \rangle \rhd P \xrightarrow{\tau} \langle \cdot \,\|\, \widetilde{c} \,\|\, \widetilde{b} \rangle \rhd Q$ *then* $\nu\widetilde{a}.\, P \to^* \nu\widetilde{b}.\, Q$.

Proposition 3.8 (Knowledge Transfer).

$$\begin{array}{ll}
\langle H_1 \,\|\, K_1 \,\|\, I_1, b \rangle \rhd P \xrightarrow{\tau} \langle H_2 \,\|\, K_2 \,\|\, I_2, b \rangle \rhd Q & \textit{iff} \\
\langle H_1 \,\|\, K_1, b \,\|\, I_1 \rangle \rhd P \xrightarrow{\tau} \langle H_2 \,\|\, K_2, b \,\|\, I_2 \rangle \rhd Q &
\end{array}$$

Finally, private names can be renamed to fresh names without affecting the transitions of the configuration.

Lemma 3.9. *If* $\langle H \,\|\, K \,\|\, I, a \rangle \rhd \mathcal{P} \xrightarrow{\mu} \langle H' \,\|\, K' \,\|\, I', a \rangle \rhd Q$ *and* $b \notin I' \cup K'$ *then* $\langle H\{b/a\} \,\|\, K \,\|\, I, b \rangle \rhd \mathcal{P}\{b/a\} \xrightarrow{\mu\{b/a\}} \langle H'\{b/a\} \,\|\, K' \,\|\, I', b \rangle \rhd Q\{b/a\}$.

4 History Equivalence

Given a history H of a configuration $\mathcal{C}$, an observer with knowledge K can make a number of tests on the values previously output by $\mathcal{C}$ and bound to output abstract names in H: it can attempt to decode encrypted values with keys from K, and compare values reachable with keys from K with other constructed values. The following equivalence states when two histories are indistinguishable by these tests for a given knowledge K.

Definition 4.1 (History Equivalence ($\simeq$)). *Two histories H, H' are equivalent under the knowledge K, and we write $K \vdash H \simeq H'$, if*

$$\begin{gathered}
dom(H) = dom(H') \qquad (\iota \mapsto \mathcal{V}) \in H \;\; \mathit{iff} \;\; (\iota \mapsto \mathcal{V}) \in H' \\
\forall \psi, \mathcal{V}. \quad fn(\psi, \mathcal{V}) \subseteq K \;\wedge\; fan(\psi, \mathcal{V}) \subseteq dom(H) \cap \mathsf{OAName} \\
\implies (H \vdash equal(\psi, \mathcal{V}) \;\; \mathit{iff} \;\; H' \vdash equal(\psi, \mathcal{V}))
\end{gathered}$$

Intuitively the first two conditions of the above definition require that equivalent histories describe the same sequence of inputs and outputs, and the observer-generated inputs are equal. The final condition compares values in the history, reachable by an observer pattern ψ using keys from K, with any other value that the observer can construct. These tests subsume the decryption tests of the observer.

Example 4.2. Consider an observer with knowledge K, where $k_1, k_2 \notin K$, and the histories $H = (\kappa_1 \mapsto \{\!|\, a \,|\!\}_{k_1})$ and $H' = (\kappa_1 \mapsto \{\!|\, a \,|\!\}_{k_2})$. The observer can test whether the message in each of the histories is encrypted with a known key $k \in K$ by attempting to decrypt it using k. This test, which fails in both histories, is encoded in the definition of ($\simeq$) by the equality: $equal(\kappa_1, \{\!|\, \kappa_1.\mathsf{decr}_k \,|\!\}_k)$.

In fact, under the given knowledge K, the two histories are indistinguishable by an observer and $K \vdash H \simeq H'$. If, after an output of the configuration, the knowledge K is extended with the key k_1 then the observer can distinguish the two histories by successfully decrypting the message in H and failing to do so in H'. The equality that distinguishes the histories in this case is $equal(\kappa_1, \{\!|\, \kappa_1.\mathsf{decr}_{k_1} \,|\!\}_{k_1})$, which is true under H and false under H'. $\square$

Example 4.3. Consider $a, k_2 \in K$, $k_1 \notin K$, and the histories $H = (\kappa_1 \mapsto \{\!|\, a \,|\!\}_{k_1})$ and $H' = (\kappa_1 \mapsto \{\!|\, b \,|\!\}_{k_1})$. The observer cannot distinguish the two histories as any decryption test will fail for both. If after a transition the histories become

$$H = (\kappa_1 \mapsto \{\!|\, a \,|\!\}_{k_1},\; \kappa_2 \mapsto \{\!|\, k_1 \,|\!\}_{k_2}) \qquad H' = (\kappa_1 \mapsto \{\!|\, b \,|\!\}_{k_1},\; \kappa_2 \mapsto \{\!|\, k_1 \,|\!\}_{k_2})$$

the observer can decrypt the second message and increase the knowledge K with the key k_1. With the new knowledge, the decryption of the first message reveals that its content is different in H and H'. This is captured in the definition of ($\simeq$) by the following equality that holds only under H: $equal(\kappa_1.\mathsf{decr}_{k_1}, a)$. $\square$

Example 4.4. The observer can also test two outputs of the configuration for equality. Thus, even with an empty knowledge, the histories

$$H = (\kappa_1 \mapsto \{\!|\, a \,|\!\}_k, \kappa_2 \mapsto \{\!|\, a \,|\!\}_k) \qquad H' = (\kappa_1 \mapsto \{\!|\, a \,|\!\}_k,\; \kappa_2 \mapsto \{\!|\, b \,|\!\}_k)$$

are distinguished. This is captured by the equality $equal(\kappa_1, \kappa_2)$. $\square$

The necessary equality tests to decide $K \vdash H \simeq H'$ are *finite*. If we ignore function types, we can easily verify this statement by considering that the length of a history, the type of each position in the history, and the values of each type using a finite knowledge K are finite. For function types, we observe that the histories of well-formed configurations (Definition 3.4) contain functions $\lambda_r x.\,\mathcal{P}$ with r in the private names of the configuration, and therefore $r \notin K$. This means that in a well-formed configuration any equality $H \vdash \mathit{equal}(\psi, \lambda_r x.\,\mathcal{P})$ with $r \in K$ is false—these tests are unnecessary.

However, $(\simeq)$ tests functions for equality via the use of patterns and $\Diamond$-values.

Example 4.5. Consider the knowledge $K = \{r\}$ and the histories $H = (\iota{\mapsto}\Diamond_r,\ \kappa{\mapsto}\iota)$ and $H' = (\iota{\mapsto}\Diamond_r,\ \kappa{\mapsto}\lambda_{r'} x.\,\mathtt{fork}\,\iota(x))$. The first history describes a process that outputs the same function that it received as an input, while the second its eta-expansion. These histories are distinguishable by an observer using the test $\mathit{equal}(\kappa, \Diamond_r)$, which is true under H but not under H'. $\square$

The finite-test equivalence $(\simeq)$ is not too permissive as it implies that the equivalence theory of values under related histories is the same. Moreover, we extend history equivalence to configurations.

Proposition 4.6. *If $K \vdash H \simeq H'$ then for all values $\mathcal{V}_1, \mathcal{V}_2$ with $\mathit{fn}(\mathcal{V}_1, \mathcal{V}_2) \subseteq K$ and $\mathit{fan}(\mathcal{V}_1, \mathcal{V}_2) \subseteq \mathit{dom}(H)$: $H \vdash \mathit{equal}(\mathcal{V}_1, \mathcal{V}_2)$ iff $H' \vdash \mathit{equal}(\mathcal{V}_1, \mathcal{V}_2)$.*

Definition 4.7. $\langle H \parallel K \parallel I\rangle \rhd \mathcal{P} \simeq \langle H' \parallel K' \parallel I'\rangle \rhd \mathcal{Q}$ *if* $K = K' \wedge K \vdash H \simeq H'$.

5 Set Simulations

Here we give a coinductive characterisation of safety preservation for HOspi based on *set simulations.* This gives a convenient, and complete, proof methodology which, moreover, is amenable to the usual up-to techniques [22]. Indeed our formulation is already up-to τ-moves. In this extended abstract we give a sketch of the characterisation. A *set simulation* is a relation between configurations and sets of configurations, ranged over by $\mathcal{S}$. For the definition we need to extend LTS transitions to sets.

Definition 5.1 (Set Transition). *If $\mathcal{S}$ is a set of configurations: $\mathcal{S} \xrightarrow{\mu} \mathcal{S}'$ when $\mathcal{S}'$ is non-empty and $\mathcal{S}' \subseteq \{\mathcal{C}' \mid \exists \mathcal{C} \in \mathcal{S}.\ \mathcal{C} \xrightarrow{\mu} \mathcal{C}'\}$; $\mathcal{S} \overset{\mu}{\Rightarrow} \mathcal{S}'$ when $\mathcal{S}'$ is non-empty and $\mathcal{S}' \subseteq \{\mathcal{C}' \mid \exists \mathcal{C} \in \mathcal{S}.\ \mathcal{C} \overset{\mu}{\Rightarrow} \mathcal{C}'\}$.*

Definition 5.2 (Set Simulation up-to τ). *A relation $\mathcal{X}$ between configurations and sets of configurations is a set simulation up-to τ if for all $\mathcal{C}\ \mathcal{X}\ \mathcal{S}$:*

1. *there exists $\mathcal{C}' \in \mathcal{S}$ such that $\mathcal{C} \simeq \mathcal{C}'$*
2. *if $\mathcal{C} \overset{\mu}{\Rightarrow} \mathcal{C}'$ and $\mu \neq \tau$ then there exists $\mathcal{S}'$ such that $\mathcal{S} \overset{\mu}{\Rightarrow} \mathcal{S}'$ and $\mathcal{C}' \overset{\tau}{\Rightarrow}\mathcal{X}\ \mathcal{S}'$.*

The definition for a set simulation up-to τ is monotone; therefore the largest set simulation up-to τ exists and we write it as $(\preceq)$. Set similarity up-to τ is sound and complete with respect to the safety preorder.

Theorem 5.3 (Soundness and Completeness of $(\preceq)$). *$P \sqsubseteq_{\mathsf{safe}} Q$ iff $\langle \cdot \parallel K \parallel \cdot\rangle \rhd Q \preceq \{\langle \cdot \parallel K \parallel \cdot\rangle \rhd P\}$.*

The main intuition of set simulations is that they are insensitive to the branching behaviour of processes.

Example 5.4. Consider the following example:

$$\begin{aligned} P &= \mathtt{outp}\, a().\, (\mathtt{outp}\, b().\, \mathbb{0} + \mathtt{outp}\, c().\, \mathbb{0}) \\ Q &= \mathtt{outp}\, a().\, \mathtt{outp}\, b().\, \mathbb{0} + \mathtt{outp}\, a().\, \mathtt{outp}\, c().\, \mathbb{0} \end{aligned}$$

These two terms are not bisimilar because when P takes an a-move, Q would need to match it, committing to one of the two branches. From a safety perspective, however, the two terms are equivalent since there is no safety test that distinguishes them. We can construct a set simulation that relates the two possible states of Q with a single state of P, which effectively delays the choice of a particular branch of Q; here we let $\mathcal{C}(R) = \langle \cdot \parallel \{a, b, c\} \parallel \cdot \rangle \rhd R$:

$$\{(\mathcal{C}(P), \{\mathcal{C}(Q)\}),\ (\mathcal{C}(\mathtt{outp}\, b().\, \mathbb{0} + \mathtt{outp}\, c().\, \mathbb{0}), \{\mathcal{C}(\mathtt{outp}\, b().\, \mathbb{0}), \mathcal{C}(\mathtt{outp}\, c().\, \mathbb{0})\})\}$$

□

6 Examples

6.1 Messaging Servers

As our first example we consider a specification of a messaging service with load balancing, which allows its clients to send a message once (using the state channel st), and distributes clients to a number of transmission servers, each listening to a separate channel in the set S. For simplicity the load balancing algorithm is abstracted away by an internal choice operator, denoted by Σ:

$$\begin{aligned} &\mathsf{MServer}_{req,res,p,S} \\ &\quad = \nu sk, st.\, \mathtt{inp}\, req(x).\, \mathtt{dec}\, x\, \mathtt{as}\, \{\!|\, y_{ck} \,|\!\}_p\, \mathtt{in} \\ &\qquad\qquad \sum_{s \in S} \big(\mathtt{outp}\, res(\{\!|\lambda_r x.\, \mathtt{inp}\, st().\, \mathtt{outp}\, s(\{\!|\, x \,|\!\}_{sk}).\, \mathbb{0} \,|\!\}_{y_{ck}}).\, \mathtt{outp}\, st().\, \mathbb{0}\big) \end{aligned}$$

Here the choice of server is decided before the client receives a response from the server. However, an implementation of this service may delegate this decision to the code sent to the client:

$$\begin{aligned} &\mathsf{MServer}'_{req,res,p,S} \\ &\quad = \nu sk, st.\, \mathtt{inp}\, req(x).\, \mathtt{dec}\, x\, \mathtt{as}\, \{\!|\, y_{ck} \,|\!\}_p\, \mathtt{in} \\ &\qquad\qquad \mathtt{outp}\, res(\{\!|\, \lambda_r x.\, \mathtt{inp}\, st().\, \sum_{s \in S} \mathtt{outp}\, s(\{\!|\, x \,|\!\}_{sk}).\, \mathbb{0} \,|\!\}_{y_{ck}}).\, \mathtt{outp}\, st().\, \mathbb{0} \end{aligned}$$

It is not possible to show that the safety properties of the specification $\mathsf{MSys} = \nu p.\, (\mathsf{Client} \mid \mathsf{MServer})$ are preserved by the implementation $\mathsf{MSys}' = \nu p.\, (\mathsf{Client} \mid \mathsf{MServer}')$ in a behavioural theory based on bisimulations. This is because the two systems are not bisimilar: the choice of the messaging channel from S happens before the message on res in MSys, and after that message in MSys'. However, using our linear theory of safety we can prove this equivalence. The interesting direction is proving $\mathsf{MSys}' \sqsubsim_{\mathsf{safe}} \mathsf{MSys}$.

We do this proof by reasoning *compositionally*. By Proposition 2.3, it suffices to show that $\mathsf{MServer}' \sqsubsetsim_{\mathsf{safe}} \mathsf{MServer}$. We consider the following continuations of $\mathsf{MServer}$ and $\mathsf{MServer}'$.

$$\begin{array}{ll|ll}
\mathsf{Srv}_1 & = \mathtt{inp}\, req(x).\ldots & \mathsf{Srv}'_1 & = \mathtt{inp}\, req(x).\ldots \\
\mathsf{Srv}_2(ck, s) & = \mathtt{outp}\, res(\{\!|\, \mathsf{F}(s) \,|\!\}_{ck}).\ldots & \mathsf{Srv}'_2(ck) & = \mathtt{outp}\, res(\{\!|\, \mathsf{F}' \,|\!\}_{ck}).\ldots \\
\mathsf{Srv}_3 & = \mathtt{outp}\, st().\,\mathbb{0} & \mathsf{Srv}'_3 & = \mathtt{outp}\, st().\,\mathbb{0} \\
\mathsf{Srv}_4(s) & = \mathtt{outp}\, s(\{\!|\, x \,|\!\}_{sk}).\,\mathbb{0} & \mathsf{Srv}'_4(s) & = \mathtt{outp}\, s(\{\!|\, x \,|\!\}_{sk}).\,\mathbb{0} \\
\mathsf{F}(s) & = \lambda_r x.\, \mathtt{inp}\, st(). & \mathsf{F}' & = \lambda_r x.\, \mathtt{inp}\, st(). \\
 & \qquad \mathtt{outp}\, s(\{\!|\, x \,|\!\}_{sk}).\,\mathbb{0} & & \qquad \sum_{s \in S} \mathtt{outp}\, s(\{\!|\, x \,|\!\}_{sk}).\,\mathbb{0}
\end{array}$$

We let $K_0 = \{req, res, p\} \cup S$ and $I_0 = \{sk, st\}$ and the configurations of $\mathsf{MServer}$:

$$\begin{array}{ll}
\mathcal{C}_1 & = \langle \cdot \parallel K_0 \parallel I_0 \rangle \rhd \mathsf{Srv}_1 \\
\mathcal{C}_2(H, K, ck, s) & = \langle H \parallel K_0 \uplus K, ck \parallel I_0, r \rangle \rhd \mathsf{Srv}_2(ck, s) \\
\mathcal{C}_3(H, K) & = \langle H \parallel K_0 \uplus K \parallel I_0, r \rangle \rhd \mathsf{Srv}_3 \\
\mathcal{C}_4(H, K, R, s) & = \langle H \parallel K_0 \uplus K \parallel I_0, r \rangle \rhd \mathsf{Srv}_4(\mathsf{s}) \mid R \\
\mathcal{C}_5(H, K, I, R) & = \langle H \parallel K_0 \uplus K \parallel I \rangle \rhd R
\end{array}$$

and the corresponding configurations of $\mathsf{MServer}'$ obtained by replacing Srv_1 to Srv_4 by Srv'_1 to Srv'_4, respectively.

As shown in Figure 4, the choice of the messaging channel in $\mathsf{MServer}$ happens before the response of the server, and after that in $\mathsf{MServer}'$. Hence, in our set simulation we relate each configuration $\mathcal{C}'_2(H', K, ck)$ and $\mathcal{C}'_3(H', K)$, where the value $\{\!|\, \mathsf{F}' \,|\!\}_{ck}$ has been indexed by some κ in H', with the set $S_2(H', K, ck)$ and $S_3(H', K, \kappa, ck)$, respectively:

$$\begin{array}{rl}
S_2(H, K, ck) = \{\mathcal{C}_2(H, K, ck, s) \mid & s \in S\} \\
S_3(H', K, \kappa, ck) = \{\mathcal{C}_3(H, K, ck, s) \mid & s \in S,\ H(\kappa) = \{\!|\, \mathsf{F}(s) \,|\!\}_{ck}, \\
 & (\forall \alpha \neq \kappa.\, H(\alpha) = H'(\alpha))\}
\end{array}$$

We construct the relation of well-formed configurations:

$$\begin{array}{rl}
\mathcal{X} = & \{(\langle \cdot \parallel K_0 \parallel \cdot \rangle \rhd \mathsf{MServer}, \{\langle \cdot \parallel K_0 \parallel \cdot \rangle \rhd \mathsf{MServer}'\}),\quad (\mathcal{C}'_1, \{\mathcal{C}_1\})\} \\
\cup & \{(\mathcal{C}'_2(H, K, ck), \mathcal{S}_2(H, K, ck)) \mid \exists \iota_1.\ \ H(\iota_1) = \{\!|\, ck \,|\!\}_p\} \\
\cup & \{(\mathcal{C}'_3(H', K), \mathcal{S}_3(H', K, \kappa, ck)\}) \mid\ \ H'(\kappa) = \{\!|\, \mathsf{F}' \,|\!\}_{ck},\ \ ck \in K\} \\
\cup & \{(\mathcal{C}'_4(H', K, R), \{\mathcal{C}_4(H, K, R)\}) \mid R \text{ deadlocked and } \Phi(H, H')\} \\
\cup & \{(\mathcal{C}'_5(H', K, I', R), \{\mathcal{C}_5(H, K, I, R)\}) \mid R \text{ deadlocked and} \\
 & \qquad\qquad\qquad\qquad r \in I, r \in I' \text{ and } \Phi(H, H')\}
\end{array}$$

$$\begin{array}{rl}
\text{where } \ \Phi(H, H') \overset{\text{def}}{=} \exists \kappa.\ H(\kappa) = \{\!|\, \mathsf{F}(s) \,|\!\}_{ck} \text{ and} \\
H'(\kappa) = \{\!|\, \mathsf{F}' \,|\!\}_{ck} \text{ and } (\forall \alpha \neq \kappa.\, H(\alpha) = H'(\alpha))
\end{array}$$

We prove that $\mathcal{X}$ is a set simulation up-to τ by considering the possible weak transitions from these configurations, the main of which are shown in Figure 4, and by showing that configurations related in $\mathcal{X}$ are also related in $(\simeq)$. The latter is easy since related histories contain either identical values, or abstractions with the same annotations ($\mathsf{F}(s)$ and F').

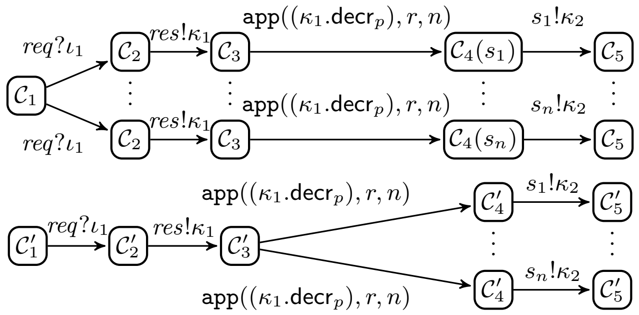

Fig. 4. Main transitions of $\mathsf{MServer}$ and $\mathsf{MServer}'$.

6.2 Example 2: Conference Servers

We now prove that the conference system Sys_2, shown in the introduction, preserves the safety properties of Sys_1. Here the function annotations are not important and we omit them. We also assume that functions can be applied to non-base values, which can be encoded by:

$$\texttt{fork}\,(\lambda x.\, P)(V) \overset{\text{def}}{=} \nu c.\, \texttt{fork}\,(\lambda y.\, \texttt{inp}\, y(x).\, P)(c).\, \texttt{outp}\, c(V).\, \mathbb{0}$$

We prove that $\mathsf{Sys}_1 \sqsubseteq_{\mathsf{safe}} \mathsf{Sys}_2$ by using Theorem 5.3 and giving a set simulation $\mathcal{X}$ such that $\mathsf{Sys}_2\ \mathcal{X}\ \{\mathsf{Sys}_1\}$. First we consider the dummy process D and the continuations of Conf and Rev_2:

$$\begin{array}{l|l} \mathsf{D}_1 = \mu X.\, \texttt{inp}\, del_{12}(x).\, X & \mathsf{D}_2 \quad\ = \mu X.\, \texttt{outp}\, del_{12}(\{\!|\, s, \lambda.\, \mathbb{0} \,|\!\}_s).\, X \\ \mathsf{D}\ \ = \mathsf{D}_1 \mid \nu s.\, \mathsf{D}_2 & \mathsf{Conf}_1 = \texttt{outp}\, ok(r).\, \mathbb{0} \end{array}$$
$$\mathsf{Rev}_{21} = \texttt{outp}\, subm(\{\!|\, r, \mathsf{F}_{prf_2}\{\mathsf{F}_{del_{12}}/f_{del_{12}}\} \,|\!\}_{s_p}).\, \mathbb{0}$$
$$\mathsf{Rev}_{22}(\iota) = \texttt{outp}\, subm(\{\!|\, r, \mathsf{F}_{prf_2}\{(\iota.\mathsf{decr}_{s_{rev_1}}.2)/f_{del_{12}}\} \,|\!\}_{s_p}).\, \mathbb{0}$$

We let $K_0 = \{subm, del_{12}, r\}$ and $I_0 = \{s_p, s_{rev_1}, s\}$, and consider the families of configurations for Sys_1:

$$\begin{array}{ll} \mathcal{C}_1 & = \langle \cdot \parallel K_0 \parallel I_0 \rangle \rhd \mathsf{Conf} \mid \mathsf{Rev}_1 \mid \mathsf{D}_1 \mid \mathsf{D}_2 \\ \mathcal{C}_2(H, K) & = \langle H \parallel K_0 \uplus K \parallel I_0 \rangle \rhd \mathsf{Conf} \mid \mathsf{D}_1 \mid \mathsf{D}_2 \\ \mathcal{C}_3(H, K) & = \langle H \parallel K_0 \uplus K \parallel I_0 \rangle \rhd \mathsf{Conf}_1 \mid \mathsf{D}_1 \mid \mathsf{D}_2 \\ \mathcal{C}_4(H, K) & = \langle H \parallel K_0 \uplus K \parallel I_0 \rangle \rhd \mathsf{D}_1 \mid \mathsf{D}_2 \\ \mathcal{C}_5(H, K) & = \langle H \parallel K_0 \uplus K \parallel I_0 \rangle \rhd \mathsf{Rev}_1 \mid \mathsf{D}_1 \mid \mathsf{D}_2 \end{array}$$

We also let $I'_0 = \{s_p, s_{rev_1}, s_{rev_2}\}$, and consider for Sys_2:

$$\begin{array}{ll} \mathcal{C}'_1 & = \langle \cdot \parallel K_0 \parallel I'_0 \rangle \rhd \mathsf{Conf} \mid \mathsf{Rev}'_1 \mid \mathsf{Rev}_2 \\ \mathcal{C}'_2(H, K) & = \langle H \parallel K_0 \uplus K \parallel I'_0 \rangle \rhd \mathsf{Conf} \mid \mathsf{Rev}_2 \\ \mathcal{C}'_{31}(H, K) & = \langle H \parallel K_0 \uplus K \parallel I'_0 \rangle \rhd \mathsf{Conf} \mid \mathsf{Rev}_{21} \\ \mathcal{C}'_{32}(H, K, \iota) & = \langle H \parallel K_0 \uplus K \parallel I'_0 \rangle \rhd \mathsf{Conf} \mid \mathsf{Rev}_{22}(\iota) \\ \mathcal{C}'_4(H, K) & = \langle H \parallel K_0 \uplus K \parallel I'_0 \rangle \rhd \mathsf{Conf} \\ \mathcal{C}'_5(H, K) & = \langle H \parallel K_0 \uplus K \parallel I'_0 \rangle \rhd \mathsf{Conf}_1 \\ \mathcal{C}'_6(H, K, R) & = \langle H \parallel K_0 \uplus K \parallel I'_0 \rangle \rhd R \end{array}$$

Here R will represent the process in states that do not lead to an output on ok. The proof is completed by showing that the following relation is a set simulation up-to τ. The proof is similar to the one in the previous example, but due to space constrains here we give only the relation.

$$\begin{aligned}
\mathcal{X} = \{ & (\langle \cdot \parallel K_0 \parallel \cdot \rangle \triangleright \mathsf{Sys}_2, \{\langle \cdot \parallel K_0 \parallel \cdot \rangle \triangleright \mathsf{Sys}_1\}), \quad (\mathcal{C}'_1, \{\mathcal{C}_1\}), \\
\cup & \{(\mathcal{C}'_2(H', K), \{\mathcal{C}_1(H, K)\}) \mid \exists \kappa.\ \Phi_1(H, H', \kappa)\} \\
\cup & \{(\mathcal{C}'_{31}(H, K), \{\mathcal{C}_1(H, K)\})\} \\
\cup & \{(\mathcal{C}'_{32}(H', K, \iota), \{\mathcal{C}_1(H, K)\}) \mid \exists \iota \in dom(H).\ \Phi_1(H, H', \kappa) \wedge H'(\iota) = \kappa\} \\
\cup & \{(\mathcal{C}'_4(H', K), \{\mathcal{C}_2(H, K)\}) \mid \Phi_2(H, H')\} \\
\cup & \{(\mathcal{C}'_5(H', K), \{\mathcal{C}_3(H, K)\}) \mid H = H' \vee \Phi_1(H, H') \vee \Phi_2(H, H')\} \\
\cup & \{(\mathcal{C}'_6(H', K, \mathbf{0}), \{\mathcal{C}_4(H, K)\}) \mid H_0 = H'_0 \vee \Phi_1(H_0, H'_0) \vee \Phi_2(H_0, H'_0)\} \\
\cup & \mathcal{Y}
\end{aligned}$$

where

$$\begin{aligned}
\Phi_1(H, H', \kappa) & \overset{\text{def}}{=} \exists H_0.\ H = H_0, (\kappa \mapsto \{\!| s, \lambda.\, \mathbf{0} |\!\}_s) \wedge H' = H_0, (\kappa \mapsto \{\!| s_p, \mathsf{F}_{del_{12}} |\!\}_{s_{rev_2}}) \\
\Phi_2(H, H') & \overset{\text{def}}{=} \exists H_0, H'_0, F.\ H = H_0, (\kappa \mapsto \{\!| r, \mathsf{F}_{prf_1} |\!\}_{s_p}) \wedge H' = H'_0, (\kappa \mapsto \{\!| r, F |\!\}_{s_p}) \\
& \qquad \wedge \big(\ (H_0 = H'_0 \wedge F = \mathsf{F}_{prf_2}\{\mathsf{F}_{del_{12}} / f_{del_{12}}\}) \vee \\
& \qquad\quad (\exists \iota, \kappa, .\ \Phi_1(H_0, H'_0, \kappa) \wedge H'_0(\iota) = \kappa \wedge \\
& \qquad\qquad F = \mathsf{F}_{prf_2}\{(\iota.\mathsf{decr}_{s_{rev_2}}.2) / f_{del_{12}}\})\ \big)
\end{aligned}$$

and $\mathcal{Y}$ relates the configurations that do not lead to a signal on ok:

$$\begin{aligned}
\mathcal{Y} = \{ & (\mathcal{C}'_6(H, K, (\mathsf{Conf} \mid \mathsf{Rev}'_1)), \{\mathcal{C}_2(H, K)\})\} \\
\cup & \{(\mathcal{C}'_6(H', K, \mathsf{Conf}), \{\mathcal{C}_2(H, K)\}) \mid \exists \kappa.\ \Phi_1(H, H', \kappa)\} \\
\cup & \{(\mathcal{C}'_6(H, K, R), \{\mathcal{C}_4(H, K)\}) \mid R \in \{\mathsf{Rev}'_1, (\mathsf{Rev}'_1 \mid \mathsf{Rev}_2), \mathsf{Rev}_{21}\}\} \\
\cup & \{(\mathcal{C}'_6(H', K, \mathsf{Rev}_2), \{\mathcal{C}_4(H, K)\}) \mid \exists \kappa.\ \Phi_1(H, H', \kappa)\} \\
\cup & \{(\mathcal{C}'_6(H', K, \mathsf{Rev}_{22}(\iota)), \{\mathcal{C}_4(H, K)\}) \mid \exists \kappa.\ \Phi_1(H, H', \kappa) \wedge H'(\iota) = \kappa\}
\end{aligned}$$

7 Related and Future Work

We have proposed a behavioural testing theory for safe equivalence for a higher-order version of the spi-calculus. We have given a characterisation of safety preservation in terms of novel *set simulations*, which provides a sound and complete proof methodology for process equivalence; the usefulness of the methodology has been demonstrated via simple illustrative examples. The LTS in our proof methodology makes extensive use of the *current knowledge* of observers, or adversaries, a generalisation of similar ideas for the first-order spi-calculus [2, 4], and environmental bisimulations [23, 25].

Sato and Sumii [25] defined a higher-order version of the applied π-calculus called the *Applied HOπ* calculus, developed a bisimulation method which is sound and complete with respect to reduction barbed congruence, and gave bisimulation proofs for secrecy properties of example higher-order systems. This work uses a strong assumption about the power of attackers: higher-order terms are sent over channels as decomposable syntax objects. The resulting proof methodology is significantly different from ours, employing *environmental bisimulations* and sophisticated *up-to-context* techniques [23]. However, bisimulations do not provide a complete proof technique for linear-time equivalences, such as safe equivalence. We believe that an LTS-based theory can be developed for safe equivalence for Applied HOπ, but that would not be first-order as for HOspi, where a weak assumption about attackers is used. We leave this to future work.

Sangiorgi has given a translation of HOπ with finite types to the π-calculus [20, 19] based on *triggers*, and a full-abstraction proof. This translation, generating a fresh trigger at every output, would be unsound for HOspi where transmitted functions can be tested for equality. An adaptation of the translation where a trigger is generated at every function definition [24, Sec. 13.2] would be sound but incomplete, at least without the use of a complex type system in the target language [24, pg. 402]. To the extent of our knowledge the details of such a translation have not been published. However, our LTS directly encodes the intuitions of the translation, avoiding the issues with full-abstraction.

Jeffrey and Rathke [14] used *symbolic triggers*, an encoding of triggers as extended processes in an LTS, to prove that bisimilarity in their LTS characterises reduction barbed congruence in the presence of recursive types, where Sangiorgi's fully-abstract translation does not apply. Our LTS takes this idea a step further by encoding a notion of triggers within the interaction history recorded in the configurations. This allows us to define a first-order theory for HOspi, a more intricate language that HOπ. Also we study a linear-time semantic equivalence, via set simulations, rather than reduction barbed congruence and bisimulation.

We believe that the approach to higher-order semantics we follow here, using an augmented LTS of configurations, is robust. In addition to safety for HOspi, we have followed this approach to characterise reduction barbed congruence for HOπ [15], and we believe that other equivalences, such as must-equivalence [12], can be similarly treated. Different cryptographic primitives, such as those used in the applied π-calculus, can also be easily accommodated. We believe that this approach can be applied to higher-order cryptographic languages with more complex type-systems such as the language in [17].

Symbolic techniques that reduce the quantification over first-order messages have been developed for the spi- and applied π-calculus [8, 5, 9]. Our symbolic treatment addresses the quantification over higher-order, rather than first-order, values. The use of simple types keeps the quantification over first-order messages finite up to fresh names.

Finally, we intend to investigate possible connections between our operational notion of histories and game semantics models for HOπ [16].

Acknowledgements: We are thankful to Eijiro Sumii and the anonymous referees for useful comments on this work.

References

1. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. *SIGPLAN Not.*, 36(3):104–115, 2001.

2. M. Abadi and A. D. Gordon. A bisimulation method for cryptographic protocols. *Nordic Journal of Computing*, 5:267–303, 1998.
3. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Inf. Comput.*, 148(1):1–70, 1999.
4. M. Boreale, R. De Nicola, and R. Pugliese. Proof techniques for cryptographic processes. *SIAM J. Comput.*, 31(3):947–986, 2001.
5. J. Borgström, S. Briais, and U. Nestmann. Symbolic bisimulation in the spi calculus. In *CONCUR*, volume 3170, pages 161–176. Springer, 2004.
6. J. Borgström and U. Nestmann. On bisimulations for the spi calculus. *Math. Structures in Comp. Sc.*, 15(3):487–552, 2005.
7. R. De Nicola and M. C. B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34(1–2):83–133, 1984.
8. S. Delaune, S. Kremer, and M. D. Ryan. Symbolic bisimulation for the applied pi calculus. *J. of Comp. Security*, 18(2):317–377, 2010.
9. L. Durante, R. Sisto, and A. Valenzano. Automatic testing equivalence verification of spi calculus specifications. *ACM Trans. Softw. Eng. Methodol.*, 12(2):222–284, 2003.
10. C. Fournet, A. D. Gordon, and S. Maffeis. A type discipline for authorization in distributed systems. In *CSF*, pages 31–48. IEEE Computer Society, 2007.
11. C. Fournet, A. D. Gordon, and S. Maffeis. A type discipline for authorization policies. *ACM Trans. Program. Lang. Syst.*, 29(5), 2007.
12. M. Hennessy. The security pi-calculus and non-interference. *J. Log. Algebr. Program.*, 63(1):3–34, 2005.
13. K. Honda and N. Yoshida. A uniform type structure for secure information flow. *ACM Trans. Program. Lang. Syst.*, 29(6), 2007.
14. A. Jeffrey and J. Rathke. Contextual equivalence for higher-order pi-calculus revisited. *LMCS*, 1(1:4), 2005.
15. V. Koutavas and M. Hennessy. First-order reasoning for higher-order concurrency (manuscript), Feb. 2010.
16. James Laird. Game semantics for higher-order concurrency. In *FSTTCS*, volume 4337 of *LNCS*, pages 417–428. Springer, 2006.
17. S. Maffeis, M. Abadi, C. Fournet, and A. D. Gordon. Code-carrying authorization. In *ESORICS*, pages 563–579. Springer-Verlag, 2008.
18. R. Milner. *Comunicating and Mobile Systems: the π-Calculus.* Cambridge University Press, 1999.
19. D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.* PhD thesis, Univ. of Edinburgh, 1992.
20. D. Sangiorgi. From pi-calculus to higher-order pi-calculus–and back. In *TAPSOFT*, volume 668 of *LNCS*, pages 151–166. Springer, 1993.
21. D. Sangiorgi. Bisimulation for higher-order process calculi. *Information and Computation*, 131(2):141–178, 1996.
22. D. Sangiorgi. On the bisimulation proof method. *Mathematical Structures in Comp. Sci.*, 8(5):447–479, 1998.
23. D. Sangiorgi, N. Kobayashi, and E. Sumii. Environmental bisimulations for higher-order languages. In *LICS*, 2007.
24. D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes.* Cambridge University Press, 2001.
25. N. Sato and E. Sumii. The higher-order, call-by-value applied pi-calculus. In *APLAS*, pages 311–326. Springer-Verlag, 2009.