Symbolic Universal Composability

Florian Böhl
KIT

Dominique Unruh
University of Tartu

Abstract

We introduce a variant of the Universal Composability framework (UC; Canetti, FOCS 2001) that uses symbolic cryptography. Two salient properties of the UC framework are secure composition and the possibility of easily defining security by giving an ideal functionality as specification. These advantages are now also available in a symbolic modeling of cryptography, allowing for a modular analysis of complex protocols.

We furthermore introduce a new technique for modular design of protocols that uses UC but avoids the need for powerful cryptographic primitives that often comes with UC protocols; this "virtual primitives" approach is unique to the symbolic setting and has no counterpart in the original computational UC framework.

Contents

1 Introduction

In the analysis of cryptographic protocols, symbolic analysis techniques (going back to Dolev and Yao [DY81]) have shown to be very fruitful. Symbolic techniques allow for much better automation than techniques working in the computational model (wherein messages are bitstrings and adversaries are runtime-limited computations). In a symbolic model of cryptography, messages are typically modeled as terms in a certain algebra, and the capacities of the adversary are described by, e.g., certain deduction rules over these terms.

In this work, we show how to apply the idea of Universal Composability (UC) [Can01] to the setting of symbolic cryptography. (The independently developed Reactive Simulatability [BPW07] has the same idea. For simplicity, we only refer to UC in the following.) The Universal Composability framework is a framework for specifying security properties of cryptographic protocols that has the following two salient properties:

- *Specifying security properties via functionalities.* In the UC framework, the security goals of a protocol are specified by describing a so-called ideal functionality which is a hypothetical entity which, by construction, achieves all the desired security goals. For example, if we wish to ask whether a protocol is a secure communication protocol, we simply specify the secure channel functionality. This very simple functionality just takes a message from Alice, informs the adversary that Alice sent a message, and gives that message to Bob. From the description of the functionality, it is then obvious what properties we achieve: The adversary learns nothing except that a message is delivered (secrecy). The message Bob receives is the same as the one that Alice sent (integrity).

  Given the description of an ideal functionality, we then call a protocol secure if it "UC-emulates" that functionality. UC-emulation essentially means that the protocol is as secure as the functionality, i.e., that any security property satisfied by the functionality (secrecy and integrity in our example) is also satisfied by the protocol.

Using ideal functionalities to describe what security a protocol achieves is often simpler than explicitly describing all required properties one by one. For example, the security of the Direct Anonymous Attestation protocol [BCC04] is only specified by an ideal functionality.

Another view on this definition is one of security preserving refinement. The functionality is an abstract specification, and the protocol is a refinement that preserves security.[1]

Note that the fact that UC-emulation preserves security can be formalized: For a certain class of security properties we have that if the functionality has this property, so has any protocol that UC-emulates that functionality. (See Section 6).

- *Composition and modular design and analysis.* Security in the UC framework implies secure composition. That is, assume a secure protocol ρ that uses an ideal functionality $\mathcal{F}$ as a building block (e.g., ρ uses a secure channel $\mathcal{F}$). Then, if another protocol π UC emulates $\mathcal{F}$ (i.e., π is a message transmission protocol), we can replace $\mathcal{F}$ by π in ρ and again get a secure protocol.

  This composition operation enables the modular design and analysis of a protocol. For example, in Section 8, we show that a variant of the Needham-Schroeder-Lowe protocol NSL [Low95] UC-emulates the key exchange functionality $\mathcal{F}_{KE}$ which gives a secure key to two parties. Another protocol SC UC-emulates the secure channel functionality $\mathcal{F}_{SC}$. And finally, assume we had some complex protocol X implementing some complex functionality $\mathcal{F}_X$ (think, e.g., of some large e-commerce application), and that X uses secure channels. Then we can plug X, NSL, and SC together, and get a protocol X* that still UC-emulates $\mathcal{F}_X$. (And due to the composition theorem, we do not need to verify the composed protocol anew.) In contrast, without the composition theorem, we would have had to analyze X* in one go; that analysis being much more complex because the implementation of the secure channel would be intermixed with the complex protocol X.

  The composition theorem also has the implication that a protocol will keep its security when run in other, as yet unknown, contexts. This is a very important property, because on the Internet, a protocol will hardly run alone. (Cryptographers often call security definitions that do not have this property "stand-alone models".)

The UC framework has been defined in the context of computational cryptography. However, its two salient properties, security specification via functionalities and secure composition, are as useful in a context where cryptography is modeled symbolically. In particular, even though computer verification in the symbolic setting scales much better than the usually manual verification in the computational setting, most analysis

[1] Many other refinement notions do not preserve, e.g., anonymity. For example, imagine a protocol where user Alice sends A or B over the network (chosen non-deterministically). And Bob sends A or B. Then the adversary cannot distinguish Alice and Bob. A refinement might be that Alice sends A and Bob sends B. Obviously, the anonymity of Alice and Bob is now violated.

techniques still cannot deal with arbitrarily complex protocols.[2] So being able to design and verify a protocol modularly will allow us to analyze more complex protocols.

Our contribution. In this work, we show that the ideas of the UC framework carry over to the symbolic setting. We show that the composition theorem and the fact that security properties carry over still hold in the symbolic UC framework. (Concurrent composition turns out to be non-trivial because we need to encode a special variant of process replication in the applied pi calculus that provides session ids to replicated processes.) We present an example analysis of a key exchange using the Needham-Schroeder-Lowe protocol, and how to use it in a secure channel protocol via composition.

We show that impossibilities from the computational UC framework unfortunately still apply in the symbolic setting; in particular, implementing a commitment functionality without any trusted setup is impossible. On the positive side, we show that this impossibility can be circumvented to a large part by a trick that we call "virtual primitives"; here we perform the proof of security under the assumption that the cryptographic primitives have some exotic features, but in the end conclude security for the original cryptographic primitives without these exotic features. This "virtual primitives"-approach is unique to the symbolic setting, to the best of our knowledge no corresponding technique exists in the computational world.

We also show how to use Proverif as a helping tool for performing the observational equivalence proofs when showing security in our framework. For this we develop a set of lemmas that help in rewriting processes and allows us to use Proverif as a tool even for observational equivalence proofs that do not involve so-called biprocesses and are thus out of the scope of Proverif. (See Section 8.) We believe that this set of lemmas is useful also in other settings than that of our work.

Prior work. The problem of transporting the ideas of the UC framework into the symbolic setting has already been tackled by Delaune, Kremer, and Pereira [DKP09]. They do, however, differ from the original UC framework (and from our work) in one crucial point: In the original framework, the existence of a so-called simulator is required that makes two different protocol executions – the "real and ideal execution" – indistinguishable (this will become clearer later). Instead of indistinguishability, [DKP09] use an observational preorder. That is, everything that can happen in the real world can non-deterministically be matched by the ideal world, *but not necessarily vice-versa*. This was due to certain problems in constructing simulators when using observational equivalence instead. However, we show that using an observational preorder limits the strength of the security definition considerably. For example, if a functionality guarantees anonymity (e.g., an anonymous broadcast), a protocol that emulates that functionality will not necessarily satisfy anonymity. On the other hand, we show that using observational equivalence instead of an observational preorder gives a stronger definition that does, e.g., preserve anonymity properties. Furthermore, we show that, when designing

[2] Verification by type checking (e.g., [BBF+11]) being a notable exception; this approach usually scales very well. But annotating a protocol with types suitable for verification can be daunting.

the functionality according to a simple guideline, the problems with observational equivalence that [DKP09] observed vanish. (However, there are challenges when dealing with concurrent composition that apply only in our setting, and not when using the weaker definition based on observational preorders.) We explain the issues related to [DKP09] in more detail in Section 7.

On the computational side, relevant prior work is of course the UC framework [Can01] itself. Other models based on the same ideas are Reactive Simulatability (RSIM) [BPW07], SPPC [DKMR05], IITM [Küs06], Task-PIOA [CCK+06a, CCK+06b], and GNUC [HS11]. Some of our results are adaptations of existing computational soundness results: the impossibility of commitments [CF01] in Section 9.3 and the joint state technique [CR03] in Section 8. Finally, the symbolic setting is not the first example of the fact that the UC framework can easily adapted to other settings to get different or stronger security guarantees, e.g., GUC (UC with shared functionalities) [CDPW07], quantum-UC [Unr10, Unr11], UC with local adversaries [CV12], UC/c (incoercibility) [UMQ10], UC with everlasting security [MQU07]. Furthermore, links between UC and symbolic models occurred where UC-like models were used to establish computational soundness results [BPW03, CH11]. Furthermore, [PS04, BS05] present UC protocol constructions where impossibilities are circumvented by giving the simulator additional power (namely superpolynomial-time computation); this shows some parallels to our "virtual primitives"-approach, see the discussion on page 80.

Outlook. Further research might tackle the following points:

- Using our framework for analyzing the security of existing protocols. A particular interesting candidate is the Direct Anonymous Attestation protocol [BCC04] because its security is already formulated in a UC model.
- Although we partially used Proverif for some of the proof steps, the analysis of our example protocols still used a lot of manual work. Can the verification of symbolic UC security be automated?
- There are extensions of the UC framework. For example [UMQ10] provides an extension that captures incoercibility. That model could be translated to the symbolic setting and used for the analysis of voting protocols.
- In combination with computational soundness results (these are results that show that symbolic security in certain cases implies computational security), the virtual primitives approach could be a viable new technique for showing *computational* security: Design the protocol symbolically modularly using virtual primitives, and then carry the security over to the computational setting.

2 Review of the applied pi calculus

In this section we review the variant of the applied pi calculus from [BAF08] that we use in our paper. Below (Section 2.2) we list some non-standard definitions that we will use,

readers familiar with the applied pi calculus can directly skip to that section.

The process calculus presented in [BAF08] is a combination of the original applied pi calculus [AF01] and one of its dialects [Bla04].

We have a set of terms that is built upon three basic sets. The infinite set of *names* $\mathcal{N}$, the infinite set of *variables* $\mathcal{V}$ and the set of *function symbols* (called the *signature* Σ). Names describe all kinds of atomic data, i.e. are used as nonces or to represent messages. We distinguish two categories of function symbols: constructors, which are used to construct terms of higher order, and destructors. Let $\mathcal{T}(\Sigma)$ be the set of terms built from names in $\mathcal{N}$, variables in $\mathcal{V}$ and constructors in Σ.

A *substitution* is a function from variables to terms $\sigma : \mathcal{V} \to \mathcal{T}(\Sigma)$. For a term T $T\sigma$ denotes the substitution of every variable x in T by $\sigma(x)$ (all variables are replaced at once). We write $\{M_1/x_1, \ldots, M_n/x_n\}$ for a substitution σ s. t. $\sigma(x_i) = M_i$ and $\sigma(x) = x$ for all $x \in \mathcal{V} \setminus \{x_1, \ldots, x_n\}$.

Sometimes it is desirable to consider two terms, that were constructed differently, equivalent. Therefore we have a finite set E of equations (M, N) (for $M = N$) where M and N are terms that contain only variables and constructors. E is called *equational theory*.

The equivalence relation $=_{\mathrm{E}}$ on terms is defined as the reflexive, transitive and symmetric closure of E closed under the application of substitutions[3] and contexts (i.e. for all terms M, N and T $M =_{\mathrm{E}} N \Rightarrow T\{M/x\} =_{\mathrm{E}} T\{N/x\}$).

To define the semantics of a destructor d we introduce a finite set R of rewrite rules $d(M_1, \ldots, M_n) \to_P M$ where M and M_i, $i \in \{1, \ldots, n\}$ are terms that contain only variables and constructors and the variables in M must be a subset of the variables used in $M_1, \ldots, M_n$, and P is a predicate on n-tuples of terms invariant under $=_{\mathrm{E}}$[4]. (We write $\to$ instead of $\to_P$ when $P(\ldots) = \mathit{true}$ always.) Analogous to [BAF08] we introduce the rewrite rule $f(x_1, \ldots, x_n) \to f(x_1, \ldots, x_n)$ for each constructor $f \in \Sigma$.(Destructors with conditional rewrite rules have been introduced in Proverif 1.87, see also [CB13]. None of our results need this additional generality. However, we explain in Section 9.1.1 why such destructors can be useful in some cases.)

$D \Downarrow M$ denotes the evaluation of D to M where D is a *destructor term*, i.e., a term or the application of a function to destructor terms. For all terms M we define $M \Downarrow M$ (i.e. when evaluating a term we obtain the term itself). If we have $D = g(D_1, \ldots, D_n)$ for a function g where D_i are destructor terms we define $g(D_1, \ldots, D_n) \Downarrow M\sigma$ for substitution σ iff there is a rewrite rule $g(M_1, \ldots, M_n) \to_P M$ and terms $N_1, \ldots, N_n$ s.t. $D_i \Downarrow N_i$, $N_i =_{\mathrm{E}} M_i\sigma$, and $P(N_1, \ldots, N_n) = \mathit{true}$.

Definition 2.1 (Symbolic model) *By* symbolic model*, denoted* $\mathcal{M} = (\Sigma, \mathrm{E}, \mathrm{R})$*, we refer to the entity of a signature* Σ*, a finite set of equations* E *and a finite set of rewrite*

[3] I.e., for every substitution σ and $M =_{\mathrm{E}} N$ we have $M\sigma =_{\mathrm{E}} N\sigma$.

[4] I.e., "Invariant under $=_{\mathrm{E}}$" means that if $N_i =_{\mathrm{E}} N_i'$ for $i = 1, \ldots, n$, then $P(N_1, \ldots, N_n) = P(N_1', \ldots, N_n')$.

$$
\begin{aligned}
P ::= \; & \mathbf{0} \\
& P|Q \\
& !P \\
& M(x).P \\
& \overline{M}\langle N\rangle.P \\
& \text{let } x = D \text{ in } P \text{ else } Q \\
& \nu a.P
\end{aligned}
$$

Figure 1: Syntax of processes in the applied pi calculus

rules R.

Note that the infinite set of names and infinite set of variables are not explicitly part of the symbolic model since they are not specific for any concrete model in our setting. We refer to them globally as $\mathcal{N}$ and $\mathcal{V}$ respectively.

Except for Section 9, it will be clear from the context which symbolic model we use. In Section 9 we focus on the relation between different symbolic models. Only then we will introduce a notation that explicitly states the symbolic model underlying a property, e.g., observational equivalence of two processes.

We can describe processes in our process calculus using the inductively defined grammar from Figure 1. For a better understanding of the syntax we anticipate the following section about its semantics and give a quick overview of the intuition connected to the syntax. The **0**-process simply does nothing and terminates (and is therefore often omitted). Two processes, P and Q, can be executed in parallel (denoted $P|Q$) They may interact with each other or with the environment independently of each other. A replication ($!P$) behaves as an infinite number of copies (instances) of P running in parallel. The scope of a name n may be restricted to a process P ($\nu n.P$). $M(x).P$ allows P to receive a message (a term) T on a channel *identified* by the term M. The variable x is used in P as a reference to the input. The counterpart of $M(x)$ is $\overline{M}\langle T\rangle.P$ which sends a message (a term) T on M and then behaves like P.

In let $x = D$ in P else Q the symbol D stands for a term or a destructor term. If we have $D \Downarrow M$ for a term M the process behaves like $P\{M/x\}$ otherwise it behaves like Q.

Except for the let-statement and parallel execution, processes do have the structure $\mathsf{statement}.P$ and we say for P (or any part of P) that it is *under* the $\mathsf{statement}$ (e.g. we say that "P is under a bang" in $!P$ or that P is under an input in $c(x).\nu n.P$). We say that P is under a let if P occurs in one of the two branches of a let.

An occurrence of a name n in a process is *bound* if it is under a νn. An occurrence of a variable x is bound if it is under a $M(x)$ or in the P-branch of a let $x = D$ in P else Q. $bn(P)$ resp. $bv(P)$ denotes the set of names resp. variables with bound occurrences in P. If an occurrence is not bound, it is called *free* and $fn(P)$, $fv(P)$ denote the corresponding sets for names resp. variables. A process is *closed* if it has no free variables.

PAR-**0**	P	$\equiv$	$P \mid \mathbf{0}$
PAR-A	$P \mid (Q \mid R)$	$\equiv$	$(P \mid Q) \mid R$
PAR-C	$P \mid Q$	$\equiv$	$Q \mid P$
NEW-C	$\nu u.\nu v.P$	$\equiv$	$\nu v.\nu u.P$
NEW-PAR	$u \notin fn(P) \Rightarrow$		
	$P \mid \nu u.Q$	$\equiv$	$\nu u.(P \mid Q)$

Figure 2: Rules for structural equivalence

REPL		$!P \rightarrow P \mid !P$
COMM		$\overline{C}\langle T\rangle.P \mid C'(x).Q$
	$\rightarrow$	$P \mid Q\{T/x\}$ if $C =_{\mathrm{E}} C'$
LET-THEN		let $x = D$ in P else Q
	$\rightarrow$	$P\{M/x\}$ if $D \Downarrow M$
LET-ELSE		let $x = D$ in P else Q
	$\rightarrow$	Q if $\nexists M$ s.t. $D \Downarrow M$

Figure 3: Rules for internal reduction

A *context* $\mathcal{C}$ is a process where exactly one occurrence of $\mathbf{0}$ is replaced with $\Box$. $\mathcal{C}[P]$ denotes the process resulting from the replacement of $\Box$ with P in $\mathcal{C}$. An *evaluation context* is a closed context $\mathcal{C}$ built from $\Box$, $\mathcal{C}|P$, $P|\mathcal{C}$, and $\nu a.\mathcal{C}$. We call an occurrence of a term or process within a process *unprotected* if it is only below parallel compositions ($|$) and restrictions (ν).

Definition 2.2 (Structural equivalence ($\equiv$)) Structural equivalence, *denoted* $\equiv$, *is the smallest equivalence relation on processes that is closed under* α*-conversion*[5] *on names and variables, application of evaluation contexts and the rules from Figure 2.*[6]

Definition 2.3 (Internal reduction ($\rightarrow$)) Internal reduction, *denoted* $\rightarrow$, *is the smallest relation on closed processes closed under structural equivalence and application of evaluation contexts such that the rules from Figure 3 hold for any closed processes* P *and* Q. $\rightarrow^*$ *denotes the reflexive, transitive closure of* $\rightarrow$.

[5] An α-conversion is a renaming process that doesn't change the meaning of a term. E.g. renaming b to c in $\nu a.\nu b.\overline{net}\langle a\rangle.\overline{net}\langle b\rangle$ is a valid α-conversion (and thus we have that $\nu a.\nu b.\overline{net}\langle a\rangle.\overline{net}\langle b\rangle \equiv \nu a.\nu c.\overline{net}\langle a\rangle.\overline{net}\langle c\rangle$), renaming b to a is not.

[6] We differ from [BAF08] by defining $\equiv$ also for non-closed processes. But on closed processes, our definition coincides with that from [BAF08].

A closed process P *emits* on M (denoted $P \downarrow_M$) if $P \equiv \mathcal{C}[\overline{M'}\langle N\rangle.Q]$ for some evaluation context $\mathcal{C}$ that does not bind $fn(M)$ and $M =_{\mathrm{E}} M'$.[7] Analogously it *reads* on M (denoted $P \uparrow_M$) if $P \equiv \mathcal{C}[M'(N).Q]$. We say that P *communicates* on M (denoted $P \updownarrow_M$) if $P \downarrow_M$ or $P \uparrow_M$.

Definition 2.4 *A* simulation $\mathcal{R}$ *is a relation on closed processes such that* $(P, Q) \in \mathcal{R}$ *implies*

(i) if $P \downarrow_M$ *then for some* Q' *we have that* $Q \to^* Q'$ *and* $Q' \downarrow_M$

(ii) if $P \to P'$ *then for some* Q' *we have that* $Q \to^* Q'$ *and* $(P', Q') \in \mathcal{R}$

(iii) $(\mathcal{C}[P], \mathcal{C}[Q]) \in \mathcal{R}$ *for all evaluation contexts* $\mathcal{C}$.

A relation $\mathcal{R}$ *is a* bisimulation *if both* $\mathcal{R}$ *and* $\mathcal{R}^{-1}$ *are a simulation.*

Observational equivalence *(*$\approx$*) is the largest bisimulation.*

It is easy to check that the transitive hull of $\approx$ satisfies the conditions (i), (ii) and (iii) from above. Hence $\approx$ contains its own transitive hull and thus is indeed an equivalence relation.

Substitutions on processes work like substitutions on terms but must additionally respect the scopes of names and variables (bound or free). Since renaming of bound names and variables doesn't change the structural equivalence class of a process we assume w.l.o.g. from now on that for $P\sigma$ we have $\sigma(x) = x$ for all $x \in bv(P)$ and $\sigma(x)$ does not contain names $n \in bn(P)$ for all $x \in fv(P)$.

2.1 Syntactic sugar

We introduce if $D = D'$ then P else Q as syntactic sugar for let $x = equals(D, D')$ in P else Q where x must not occur in P or Q and D, D' are destructor terms. Note that we assume the existence of an *equals* destructor with the rewrite rule $equals(x, x) \to x$ throughout this paper (see Definition 2.5 (iii)). Furthermore, we write $C().P$ for $C(x).P$ where x is a fresh variable, and $\overline{C}\langle\rangle.P$ for $\overline{C}\langle empty\rangle$ assuming a nullary constructor *empty* (see Definition 2.5 (i)).

Later, when dealing with Proverif processes, e.g., in Definition 8.3, we use the Proverif syntax for pattern matching in inputs and lets: E.g., (let $(=n, x) = D$ in P else Q) executes $P\{T/x\}$ if $D \Downarrow (n, T)$ (i.e., D has to evaluate to a pair with n beeing the first value while x is used as a reference for the arbitrary second value T) and Q otherwise. Inputs of type $C((x, \_))$ expect a pair as input where the first value is referenced by x while the second value is dropped (i.e., when receiving an input (T, T') on C, $C((x, \_)).P$ continues to run as $P\{T/x\}$. For more details see the Proverif manual [Bla12b] We stress

[7] It is indeed intentional that the definition requires $\mathcal{C}$ not to bind $fn(M)$ (as opposed to $fn(M')$) even though we consider the process $\mathcal{C}[\overline{M'}\langle N\rangle.Q]$. This way the definition is equivalent to the following: $P \downarrow_M$ iff $P \equiv_{\mathrm{E}} \mathcal{C}[\overline{M}\langle N\rangle.Q]$ for some evaluation context $\mathcal{C}$ not binding $fn(M)$, and some process Q [Bla12a]. Here $\equiv_{\mathrm{E}}$ is structural equivalence modulo replacing terms by equivalent ones, see Definition 2.6.

that these constructions are just syntactic sugar and can be replaced by statements accoding to the grammar of the pi calculus we described above.

2.2 Additional concepts used in this work

In this section, we describe several nonstandard concepts related to the applied pi calculus that we use in this work.

Miscellaneous. A context always contains a single occurrence of the hole. Sometimes we need a context which may or may not contain a hole: A *0-1-context* is defined like a context, except that there may be zero or one occurrences of the hole.

We refer to occurrences of terms that identify channels in a process as *channel identifiers*. E.g., in $\overline{M}\langle T\rangle\ M$ is a channel identifier and T is not – even if M and T were the same term (because M and T are different occurrences).

We allow destructors with conditional rewrite rules following [CB13], see page 6. None of our results actually requires these conditional destructors, though. The reader may safely assume the usual, unconditional definition of constructors.

Natural symbolic models. A number of lemmas in this paper only hold when the symbolic model we use satisfies certain natural conditions. Instead of stating these explicitly each time, we collect all these conditions in the following definition:

Definition 2.5 (Natural symbolic model) *We say a symbolic model is* natural *if it satisfies the following conditions:*

(i) *there is a constructor* $empty/0 \in \Sigma$,
(ii) *a constructor for pairings, denoted* $(\Box,\Box)$, *is part of the signature* Σ,
(iii) *there is a destructor* $equals/2 \in \Sigma$ *with rewrite rule* $equals(x,x) \to x$ *and no further rewrite rules that contain equals,*
(iv) *there are destructors* $fst/1, snd/1 \in \Sigma$ *with rewrite rules* $fst((x,y)) \to x$ *and* $snd((x,y)) \to y$,
(v) *for all terms* T, T_1 *with* $fst(T) \Downarrow T_1$ *there exists a term* T_2 *with* $snd(T) \Downarrow T_2$ *and furthermore* $(T_1,T_2) =_{\mathrm{E}} T$ *for all such* T_2 *and vice versa,*
(vi) *for arbitrary terms* T_1, T_2, T_1', T_2' *we require that* $(T_1,T_2) =_{\mathrm{E}} (T_1',T_2')$ *entails* $T_1 =_{\mathrm{E}} T_1'$ *and* $T_2 =_{\mathrm{E}} T_2'$,
(vii) *for any destructor term* D *and any name* $n \notin fn(D)$ *we require that* $D \Downarrow T$ *for a term* T *entails the existence of a term* T' *with* $D \Downarrow T'$, $n \notin fn(T')$ *and* $T =_{\mathrm{E}} T'$,
(viii) *there are terms* T, T' *with* $T \neq_{\mathrm{E}} T'$.

In the following, we will always assume that the symbolic model is natural in the sense of Definition 2.5.

Equivalence of processes modulo rewriting. Structural equivalence $\equiv$ does not allow us to replace a term M by another term $M' =_{\mathrm{E}} M$. In some places, we will

therefore need to apply $=_E$ to processes, and we will also use an extension $\equiv_E$ of $\equiv$ that allows us to replace terms:

Definition 2.6 *We extend $=_E$ to destructor terms and processes as follows:*

Given two destructor terms D, D', we have $D =_E D'$ iff D can be rewritten into D' by replacing subterms by $=_E$-equivalent subterms. (But replacing destructors is not allowed. E.g., if d is a destructor and f, g are constructors, and $f(x) =_E g(x)$ is in the equational theory, we have $d(f(a)) =_E d(g(a))$ but not $f(d(a)) =_E g(d(a))$. Formally, $=_E$ is the smallest equivalence relation on destructor terms such that $D\{M/x\} =_E D\{M'/x\}$ for destructor terms D and terms $M =_E M'$.

Given two processes P, P', we have $P =_E P'$ iff P can be rewritten into P' by α-conversion and by replacing terms and destructor terms by $=_E$-equivalent ones. Formally, $=_E$ is the smallest equivalence relation closed under α-renaming such that $P\{M/x\} =_E P\{M'/x\}$ for processes P and terms $M =_E M'$.

Given two processes P, P', we have $P \equiv_E P'$ iff P can be rewritten into P' by $=_E$ and $\equiv$. Formally, $\equiv_E \;:=\; (=_E \cup \equiv)^*$.

Full observational equivalence. A substitution σ is a *closing substitution* if $P\sigma$ is closed. We call two (not necessarily closed) processes P and Q *fully observationally equivalent* (denoted $P \approx\!\!\!\!\approx Q$) iff $P\sigma \approx Q\sigma$ for all closing substitutions σ (where we implicitly assume that the bound names in P, Q are renamed so that they are distinct from the free names of σ). Since $\approx$ is closed under $\equiv$ it follows in a straightforward way that $\approx\!\!\!\!\approx$ is closed under $\equiv$.

The motivation behind the definition of $\approx\!\!\!\!\approx$ is the following lemma which allows us to replace fully observationally equivalent subprocesses by each other.

Lemma 2.7 *Let P and Q be processes and $P \approx\!\!\!\!\approx Q$. Then $\mathcal{C}[P] \approx\!\!\!\!\approx \mathcal{C}[Q]$ for every context $\mathcal{C}$.*

To show this lemma, we first prove the following lemma:

Lemma 2.8 *Let P and Q be closed processes. We have $P \approx Q \Rightarrow\; !P \approx\; !Q$.*

Proof. We define a relation $\mathcal{R} := \approx \cup \{(\nu \underline{n}.(IP|!P), \nu \underline{n}.(IQ|!Q)) : IP, IQ$ closed processes with $IP \approx IQ$ and $\underline{n}$ a vector of names $\}$ closed under structural equivalence. Intuitively, IP and IQ represent the running instances of P resp. Q. For $(A, B) \in \mathcal{R}$ we show the three points of observational equivalence.

If $(A, B) \in \approx$ there is nothing to show. Otherwise $(A, B) = (\nu \underline{n}.(IP|!P), \nu \underline{n}.(IQ|!Q))$.

- If $\nu \underline{n}.(IP|!P) \downarrow_M$ we have $\nu \underline{n}.IP \downarrow_M$ and, since $IQ \approx IQ$, $\nu \underline{n}.IQ \downarrow_M$. Therefore $\nu \underline{n}.(IQ|!Q) \downarrow_M$.
- For internal reductions $\rightarrow$ in $\nu \underline{n}.(IP|!P)$ we distinguish two cases:

- A new instance of P spawns, i.e., $\nu\underline{n}.(IP|!P) \to \nu\underline{n}.(IP|P|!P)$. We define $IP' := IP|P$ and IQ' analogously. Then there is a corresponding internal reduction (following the REPL rule) for the Q-side $\nu\underline{n}.(IQ|!Q) \to \nu\underline{n}.(IQ'|!Q)$ and therefore $(\nu\underline{n}.(IP'|!P), \nu\underline{n}.(IQ'|!Q) \in \mathcal{R}$ (note that $IP' \approx IQ'$ since $IP \approx IQ$ and $P \approx Q$).
- The reduction $\to$ only affects $!P$ structurally. That is, we basically have $\nu\underline{n}.(IP|!P) \to \nu\underline{n}.(IP'|!P)$. Since $IP \approx IQ$ we find IQ' s.t. $IQ \to^* IQ'$ and $IP' \approx IQ'$. Hence $(\nu\underline{n}.(IP'|!P), \nu\underline{n}.(IQ'|!Q)) \in \mathcal{R}$.

- For any evaluation context $\mathcal{C}$ we have $\mathcal{C}[\nu\underline{n}.(IP|!P)] \equiv \nu\underline{n}'.(\mathcal{C}'[IP]|!P)$ where $\mathcal{C}'$ is $\mathcal{C}$ with all restrictions moved into $\underline{n}'$. Analogously we have $\mathcal{C}[\nu\underline{n}.(IQ|!Q)] \equiv \nu\underline{n}'.(\mathcal{C}'[IQ]|!Q)$ with the same $\mathcal{C}'$, $\underline{n}'$. Since $\mathcal{C}'$ is an evaluation context, $\mathcal{C}'[IP] \approx \mathcal{C}'[IQ]$. Altogether we have $(\nu\underline{n}'.(\mathcal{C}'[IP]|!P), \nu\underline{n}'.(\mathcal{C}'[IQ]|!Q)) \in \mathcal{R}$.

This concludes our proof since the definition of $\mathcal{R}$ is symmetric. □

We can now show Lemma 2.7:

Proof of Lemma 2.7. First consider the case that $\mathcal{C}$ is an evaluation context which is allowed to have free variables here. For all closing substitutions σ we have $P\sigma \approx Q\sigma$ and hence $\mathcal{C}\sigma[P\sigma] \approx \mathcal{C}\sigma[Q\sigma]$. Therefore $\mathcal{C}[P]\sigma \approx \mathcal{C}[Q]\sigma$ which entails $\mathcal{C}[P] \approxeq \mathcal{C}[Q]$.

To expand the proof from evaluation contexts to general contexts $\mathcal{C}$ we show the following properties for $\approxeq$ from which the Lemma immediately follows by induction:

1. If $P \approxeq Q$ then $\overline{M}\langle T\rangle.P \approxeq \overline{M}\langle T\rangle.Q$ for arbitrary terms M and T:
   Let σ be a closing substitution for $\overline{M}\langle T\rangle.P$ and $\overline{M}\langle T\rangle.Q$. We define the relation $\mathcal{R} := \approx \cup \{(\mathcal{C}[(\overline{M}\langle T\rangle.P)\sigma], \mathcal{C}[(\overline{M}\langle T\rangle.Q)\sigma]) : \mathcal{C}$ closed evaluation context$\}$ closed under structural equivalence. We show that $\mathcal{R}$ satisfies the three points of observational equivalence. Let $(A, B) \in \mathcal{R}$. For $(A, B) \in \approx$ there is nothing to do. Otherwise $(A, B) = (\mathcal{C}[(\overline{M}\langle T\rangle.P)\sigma], \mathcal{C}[(\overline{M}\langle T\rangle.Q)\sigma])$ for some closed evaluation context $\mathcal{C}$.

   - $A \downarrow_N$: If $\mathcal{C}[\mathbf{0}] \downarrow_N$ obviously $B \downarrow_N$ as well. Otherwise $(\overline{M}\langle T\rangle.P)\sigma \downarrow_N$ where the free names of N are not bound by $\mathcal{C}$ which requires $N =_{\mathrm{E}} M$ and hence leads to $(\overline{M}\langle T\rangle.Q)\sigma \downarrow_N \Rightarrow B \downarrow_N$.
   - For internal reductions in A we distinguish two cases:
     - $\to$ is the COMM reduction $\mathcal{C}[(\overline{M}\langle T\rangle.P)\sigma] \to \mathcal{C}'[P\sigma]$ (up to structural equivalence). In the same way we can reduce $\mathcal{C}[(\overline{M}\langle T\rangle.Q)\sigma] \to \mathcal{C}'[Q\sigma]$. Since $P\sigma \approx Q\sigma$ and $\mathcal{C}'$ is closed we have $(\mathcal{C}'[P\sigma], \mathcal{C}'[Q\sigma]) \in \approx \subseteq \mathcal{R}$.
     - The reduction $\to$ affects $(\overline{M}\langle T\rangle.P)\sigma$ only structurally. That is, we basically have $\mathcal{C}[\mathbf{0}] \to \mathcal{C}'[\mathbf{0}]$. In this case we apply the same reduction in effect to B and have $(\mathcal{C}'[(\overline{M}\langle T\rangle.P)\sigma], \mathcal{C}'[(\overline{M}\langle T\rangle.Q)\sigma]) \in \mathcal{R}$.
   - Obviously, $\mathcal{R}$ is closed under the application of closed evaluation contexts.

   This concludes our proof since the definition of $\mathcal{R}$ is symmetric.

2. If $P \approxeq Q$ then $M(x).P \approxeq M(x).Q$ for an arbitrary term M:
   We prove this statement analogously to the previous one: It only differs in the direction of message flow on M. In the corresponding branch of the proof an input of N on M results in $P\{N/x\}$ resp. $Q\{N/x\}$ (note that $\mathcal{C}$ is closed and hence N is closed). Since we have $P\sigma \approx Q\sigma$ in particular for every closing σ with $\sigma(x) = N$ we have that $P\{N/x\} \approxeq Q\{N/x\}$ holds.

3. If $P \approxeq Q$ then $!P \approxeq !Q$:
   A closing substitution σ with $P\sigma \approx Q\sigma$ but $!P\sigma \not\approx !Q\sigma$ contradicts Lemma 2.8.

4. If $P_1 \approxeq Q_1$ and $P_2 \approxeq Q_2$ then (let $x = D$ in P_1 else P_2) $\approxeq$ (let $x = D$ in Q_1 else Q_2) for an arbitrary destructor term D:
   Again, the complete proof is analogous to the one in case 2. Hence we only discuss the reduction of the let-statement here: For all closing substitutions σ for let $x = D$ in P_1 else P_2 and let $x = D$ in Q_1 else Q_2 we have that $D\sigma$ is closed. If we have $D\sigma \Downarrow M$ for a (closed!) term M the let-statement reduces to $P_1\{M/x\}\sigma \approx Q_1\{M/x\}\sigma$ (note that $\sigma(x) = x$ since x is a bound variable) which holds since $P_1 \approxeq Q_1$. Otherwise it reduces to $P_2\sigma \approx Q_2\sigma$ which holds since $P_2 \approxeq Q_2$. □

Product processes. In order to argue about concurrent composition, as a technical tool, we will need an extension of the applied pi calculus that supports infinite parallel compositions of processes which are tagged with distinct terms.

Intuitively, the *indexed replication* $\prod_{x\in S} P$ stands for $P\{s_1/x\}|P\{s_2/x\}|\dots$ when $S = \{s_1, s_2, \dots\}$. (Like $!P$ stands for $P|P|\dots$.) We call processes from this extended calculus *product processes*. Note that our main definitions and results are still stated with respect to the original calculus from [BAF08]; we only use product processes in some specific situations.

Definition 2.9 (Product processes) Product processes *are defined by the grammar in Figure 1 with the additional construct* $\prod_{x\in S} P$ *where* x *is a variable,* S *a (possibly infinite) set of terms, and* P *a product process. (We call* $\prod_{x\in S} P$ *an* indexed replication.*)*

(Note that we consider $\prod_{x\in S}$ *to be a binder. I.e., in* $\prod_{x\in S} P$*, we consider* x *a bound variable.)*

Structural equivalence ($\equiv$) *on product processes is defined using the same rules as on processes (see Figure 2).*

The reduction relation $\to$ *on product processes is defined using the same rules as on processes (see Figure 3), with the following additional rule (IREPL): If* $M \in S$*, then* $\prod_{x\in S} P \to \left(\prod_{x\in S'} P\right) \mid P\{M/x\}$ *with* $S' := S \setminus \{M' : M =_{\mathrm{E}} M'\}$*. (Essentially* S *is treated as a set of session ids which contains each sid at most once modulo* $=_{\mathrm{E}}$*.)*

Observational equivalence ($\approx$) *on product processes is defined like observational equivalence on processes (Definition 2.4). In particular, as in Definition 2.4, in rule* (iii) *we quantify over evaluation contexts that* do not *contain indexed replications.*

Notice that processes are also product processes, and that on processes, the new definitions of $\equiv$, $\rightarrow$, and $\approx$ from Definition 2.9 coincide with the original definitions.

3 Useful properties of the pi calculus

In this section, we introduce a number of useful lemmas for the applied pi calculus. These lemmas are useful to derive observational equivalences of processes by step by step rewriting (and for using Proverif as a tool in deriving equivalences that Proverif cannot handle). We believe that they may be useful in other similar situations, too.

Lemma 3.1 *For natural symbolic models, the following hold:*
(i) If $n \notin fn(M)$, then $n \neq_E M$.
(ii) $n \neq_E m$ for names $n \neq m$.
(iii) $(n, M') \neq_E M$ for all terms M, M' and names $n \notin fn(M)$.

Proof. We show (i):

Fix a term M with $n \notin fn(M)$. Assume for contradiction $n =_E M$. Fix a renaming α such that $\alpha(n) =: n^* \neq n$ and $\alpha(m) = m$ for all $m \in fn(M)$. (This is possible since $n \notin fn(M)$.) Hence $n =_E M = M\alpha =_E n\alpha = n^*$ (since the rules defining $=_E$ are closed under renaming). Thus $n =_E n^* \neq n$. Intuitively, this means that all names are equivalent under $=_E$.

By Definition 2.5 (viii) (natural symbolic model) there are terms T, T with $T \neq_E T'$. Since the equations in E contain by definition only variables and constructors, all rules defining $=_E$ are closed under substitutions of names by terms. Hence $n =_E n^*$ implies $T =_E T'$.

We have a contradiction, hence (i) follows.

(ii) follows from (i) with $M := m$.

We show (iii):

Assume $(n, M') =_E M$ towards contradiction. Since M does not contain n, $M = M\sigma$ for $\sigma := (n \mapsto n', n' \mapsto n)$ and any $n' \notin fn(M)$. Then $(n', M'\sigma) = (n, M')\sigma =_E M\sigma = M =_E (n, M')$. (Here we use that $=_E$ is closed under renaming which follows from the fact that equations and reduction rules in the symbolic model do not contain names.) By Definition 2.5 (vi) (natural symbolic model), this implies $n' =_E n$ which contradicts (ii). Thus, the assumption that $(n, M') =_E M$ was wrong. (iii) follows. □

Lemma 3.2 *Let P, P' be processes. Let D, D' be destructor terms. Let M, M' be terms.*
(i) If $a \notin fn(P)$, then $P \approx\!\!\approx \nu a.P$.
(ii) If $a \notin fn(M)$, then $\nu a.M(x).P \approx\!\!\approx M(x).\nu a.P$.
(iii) Assume P is closed and that P does not contain unprotected inputs or outputs. Assume $P \rightarrow P'$, and that for all P'' with $P \rightarrow P''$ we have $P' \approx P''$. Then $P \approx P'$.

(iv) *If M, M' are terms with $M =_{\mathrm{E}} M'$, then $P\{M/x\} \lessapprox P\{M'/x\}$.*
(v) *If for all substitutions σ that close D, M we have $D\sigma \Downarrow M\sigma$, and for all M' with $D\sigma \Downarrow M'\sigma$ we have $M\sigma =_{\mathrm{E}} M'\sigma$, then $(\mathit{let}\ x = D\ \mathit{in}\ P\ \mathit{else}\ P') \lessapprox P\{M/x\}$.*
(vi) *If D is closed and there is no M with $D \Downarrow M$, then $(\mathit{let}\ x = D\ \mathit{in}\ P\ \mathit{else}\ P') \lessapprox P'$.*
(vii) *If for all substitution σ that close D, D' there exist M, M' with $D\sigma \Downarrow M\sigma$, $D'\sigma \Downarrow M'\sigma$ and $M\sigma =_{\mathrm{E}} M'\sigma$ then $(\mathit{if}\ D = D'\ \mathit{then}\ P\ \mathit{else}\ P') \lessapprox P$*
(viii) *We have $!P \approx P\,|!P$.*
(ix) *$\prod_{x\in SID} P \approx \prod_{x\in SID\setminus\{t_1,\ldots,t_n\}} P|P\{\frac{t_1}{x}\}|\ldots|P\{\frac{t_n}{x}\}$ for $t_1, \ldots, t_n \in SID$.*

Proof. We show (i): Let $\mathcal{R} := \{(Q, \nu a.Q) : Q$ *a closed process*, $a \notin fn(Q)$ a name$\}$ up to structural equivalence. It is easy to see that $\mathcal{R}$ is a bisimulation. Thus $Q \approx \nu a.Q$ for any closed process. This implies that $P\sigma \approx \nu a.(P\sigma) \equiv (\nu a.P)\sigma$ for any closing σ. Hence $P \lessapprox \nu a.P$.

We show (ii): Let $\mathcal{R} := \{(E[\nu a.M(x).Q], E[M(x).\nu a.Q])\} \cup \approx$ up to structural equivalence where E ranges over all evaluation contexts, Q over closed processes, a over names, and M over terms with $a \notin fn(M)$. One can check that $\mathcal{R}$ satisfies the conditions for a bisimulation. To show $\nu a.M(x).P \lessapprox M(x).\nu a.P$, fix a closing substitution σ. Then $\big((\nu a.M(x).P)\sigma, (M(x).\nu a.P)\sigma\big) \in \mathcal{R}$, thus $(\nu a.M(x).P)\sigma \approx (M(x).\nu a.P)\sigma$. Since this holds for any closing σ, we have $\nu a.M(x).P \lessapprox M(x).\nu a.P$ and (ii) follows.

We show (iii): Let $\mathcal{R} := \{(E[P], E[P']) : E$ evaluation context$\} \cup \approx$. (Here P, P' refer to the processes from the statement of the lemma.) We check that $\mathcal{R}$ is a bisimulation. In all the following cases, if $A \approx B$, the statement is immediate. Thus we assume $A \equiv E[P]$, $B \equiv E[P']$ in each case.

- If $(A, B) \in \mathcal{R}$ and $A \downarrow_M$ then there exists a B' with $B \to^* B'$ and $B' \downarrow_M$: If $A \approx B$, then this is immediate. Thus assume $A \equiv E[P]$, $B \equiv E[P']$. Since P does not contain unprotected outputs, we have that the output on M is in E. Hence $B \equiv E[P'] \downarrow_M$.

- If $(A, B) \in \mathcal{R}$ and $B \downarrow_M$ then there exists an A' with $A \to^* A'$ and $A' \downarrow_M$: If $A \approx B$, then this is immediate. Thus assume $A \equiv E[P]$, $B \equiv E[P']$. Since $P \to P'$ we have $A \to A' := E[P'] \equiv B$. Since $B \downarrow_M$, also $A' \downarrow_M$.

- If $(A, B) \in \mathcal{R}$ and $A \to A'$ then there exists a B' with $B \to^* B'$ and $(A', B') \in \mathcal{R}$: If $A \approx B$, then this is immediate. Thus assume $A \equiv E[P]$, $B \equiv E[P']$. Since P does not contain unprotected inputs or outputs, $A' \equiv E'[P]$ for some evaluation context E or $A' \equiv E[P'']$ for some P'' with $P \to P''$. In the first case, $B \to B' := E'[P']$ and hence $(A', B') \in \mathcal{R}$. In the second case, $P'' \approx P'$ and thus $A' \approx E[P'] \equiv B =: B'$. Thus $B \to^* B'$ and $(A', B') \in \mathcal{R}$.

- If $(A, B) \in \mathcal{R}$ and $B \to B'$ then there exists a A' with $A \to^* A'$ and $(A', B') \in \mathcal{R}$: If $A \approx B$, then this is immediate. Thus assume $A \equiv E[P]$, $B \equiv E[P']$. Since $P \to P'$, we have $A \to A'' := E[P'] \equiv B$. Since $B \to B'$, we have $A \to A'' \to A' := B'$. Hence $A \to^* A'$ and $(A', B') \in \mathcal{R}$.

- $\mathcal{R}$ is closed under application of evaluation contexts by construction.

We show (iv): Let $(A, B) \in \mathcal{R}$ iff A results from B by replacing terms M by terms M' with $M =_{\mathrm{E}} M'$. It is easy to check that $\mathcal{R}$ is a bisimulation. Fix a process P, terms M, M' with $M =_{\mathrm{E}} M'$, and σ a substitution mapping variables to ground terms that closes $P\{M/x\}$ and $P\{M'/x\}$. Then $P\{M/x\}\sigma$ results from $P\{M'/x\}\sigma$ by replacing some occurrences of $M'\sigma$ by $M\sigma$. Since $M =_{\mathrm{E}} M'$, we have $M\sigma =_{\mathrm{E}} M'\sigma$. Thus $(P\{M/x\}\sigma, P\{M'/x\}\sigma) \in \mathcal{R}$, hence $P\{M/x\}\sigma \approx P\{M'/x\}\sigma$. Since this holds for any closing σ, $P\{M/x\} \mathrel{\approx\!\!\!\!\approx} P\{M'/x\}$.

We show (v): First, assume that $A := (\text{let } x = D \text{ in } P \text{ else } P')$ is closed. We have that if $A \to A'$, then $A' \equiv P\{M'/x\}$ for some M' with $D \Downarrow M'$. By (iv) and using that $M =_{\mathrm{E}} M'$ for all M' with $D \Downarrow M'$, this implies $A' \approx P\{M/x\}$. Furthermore A does not contain unprotected inputs or outputs. Thus by (iii), we have $A \approx P\{M/x\}$. From this follows that $(\text{let } x = D \text{ in } P \text{ else } P') \mathrel{\approx\!\!\!\!\approx} P\{M/x\}$ even if $(\text{let } x = D \text{ in } P \text{ else } P')$ is not closed, analogously to (i).

We show (vi): First, assume that $A := (\text{let } x = D \text{ in } P \text{ else } P')$ is closed. We have that if $A \to A'$, then $A' \equiv P'$. Furthermore A does not contain unprotected inputs or outputs. Thus by (iii), we have $A \approx P'$. From this follows that $(\text{let } x = D \text{ in } P \text{ else } P') \mathrel{\approx\!\!\!\!\approx} P'$ even if $(\text{let } x = D \text{ in } P \text{ else } P')$ is not closed, analogously to (i).

We show (vii): First, assume that $A := (\text{if } D = D' \text{ then } P \text{ else } P')$ is closed. We resolve the syntactic sugar for "if" and have $A = (\text{let } x = \mathit{equals}(D, D') \text{ in } P \text{ else } P')$. If $A \to A'$, then $A' \equiv P$ $(x \notin \mathit{fv}(P))$. Thus by (iii), we have $A \approx P'$. From this follows that $(\text{let } x = D \text{ in } P \text{ else } P') \mathrel{\approx\!\!\!\!\approx} P'$ even if $(\text{let } x = D \text{ in } P \text{ else } P')$ is not closed, analogously to (i).

We show (viii): If $!P \to P''$, then $P'' \equiv P|!P$ by definition of $\to$. By (iii) this implies $!P \approx P|!P$.

We show (ix): Given a set $A = \{t_1, \ldots, t_k\} \subseteq \mathit{SID}$, we write $\sum_{x \in A} P$ for $P\{t_1/x\}| \ldots |P\{t_k/x\}$. Let

$$\mathcal{R} := \{(E[\prod_{x \in \mathit{SID} \setminus A \setminus D} P | \sum_{x \in A} P], E[\prod_{x \in \mathit{SID} \setminus B \setminus D} P | \sum_{x \in B} P])\}$$

up to structural equivalence where E ranges over evaluation contexts and A, B, D range over subsets of SID with D disjoint of $A \cup B$. One can check that $\mathcal{R}$ satisfies all conditions for being a bisimulation. Since $(\prod_{x \in \mathit{SID}} P, \prod_{x \in \mathit{SID} \setminus \{t_1, \ldots, t_n\}} |P\{t_1/x\}| \ldots |P\{t_n/x\}) \in \mathcal{R}$, (ix) follows. □

Lemma 3.3 *Let C be a 0-1-context whose hole is not under a bang and such that n does not occur in C, Q, or t. Assume that C does not bind any of $\mathit{fv}(Q) \setminus \{x\}$ or $\mathit{fn}(Q)$ over its hole. Then $\nu n.C[\overline{n}\langle t\rangle]|n(x).Q \mathrel{\approx\!\!\!\!\approx} C[Q\{t/x\}]$*

Proof. We show the lemma for $\approx$ instead of $\lessapprox$, and assuming that $\nu n.C[\overline{n}\langle t\rangle]|n(x).Q$ and $C[Q\{t/x\}]$ are closed and that $fn(Q) \subseteq \{x\}$. The general case then follows by definition of $\lessapprox$. We define the relation $\mathcal{R}$: $(A, B) \in \mathcal{R}$ iff $A \approx B$ or there is a name n, a list of names $\tilde{a}$, a term t, a variable x, an integer k, a 0-1-context C not containing n and not having its hole under a bang and not binding $fn(Q)$ over its hole, such that the following holds:

$$A \equiv \nu n\tilde{a}.C[\overline{n}\langle t\rangle]|n(x).Q, \qquad B \equiv \nu n\tilde{a}.C[Q\{t/x\}] \tag{1}$$

We check the three conditions for bisimulations (in both directions).

- If $(A, B) \in \mathcal{R}$ and $A \downarrow_M$, then $B \downarrow_M$:

  The case $A \approx B$ is trivial. We thus assume that A, B are as in (1).

  If $\nu n\tilde{a}.C[\overline{n}\langle t\rangle]|n(x).Q \downarrow_M$, then the output on M is in C. ($\overline{n}\langle t\rangle$ cannot be that output, because n is bound.) Hence $\nu n\tilde{a}.C[Q\{t/x\}] \downarrow_M$.

- If $(A, B) \in \mathcal{R}$ and $B \downarrow_M$, then there exists an A' with $A \rightarrow^* A'$ and $A' \downarrow_M$:

  The case $A \approx B$ is trivial. We thus assume that A, B are as in (1).

  If $\nu n\tilde{a}.C[Q\{t/x\}] \downarrow_M$, we distinguish two cases. If the output on M is in C, then $\nu n\tilde{a}.C[\overline{n}\langle t\rangle]|n(x).Q \downarrow_M$. Consider the case that the output on M is in $Q\{t/x\}$. Without loss of generality, we can assume that no name in t is bound in C (otherwise we could move the corresponding restrictions from C into $\nu\tilde{a}$ since C does not bind $fn(Q)$ over its hole). Since the output on M is in $Q\{t/x\}$, C is an evaluation context and thus $\nu n\tilde{a}.C[\overline{n}\langle t\rangle]|n(x).Q \rightarrow \nu n\tilde{a}.C[0]|Q\{t/x\} \downarrow_M$.

- If $(A, B) \in \mathcal{R}$ and $A \rightarrow A'$, then there is a B' with $B \rightarrow^* B'$ and $(A', B') \in \mathcal{R}$:

  The case $A \approx B$ is trivial. We thus assume that A, B are as in (1).

  We distinguish the following cases:

  If the reduction $A \rightarrow A'$ involves only C, then $A' \equiv \nu n\tilde{a}.\tilde{C}[\overline{n}\langle t\rangle\sigma]|n(x).Q$ for some 0-1-context $\tilde{C}$. Here the substitution σ represents possible variable assignments performed over the hole of C (e.g., if $C = \overline{a}\langle T\rangle \mid a(y).\Box$, then $\sigma = \{T/y\}$).

  Then $B \rightarrow B' := \nu n\tilde{a}.\tilde{C}[Q\{t/x\}\sigma] = \nu n\tilde{a}.\tilde{C}[Q\{t\sigma/x\}]$ where the last equality uses that $fn(Q) \subseteq x$. Also, $\tilde{C}$ does not have more that one hole (in which case $\tilde{C}$ would not be a zero-or-one-hole context) because the hole in C does not occur under a bang.

  Thus we have $(A', B') \in \mathcal{R}$.

  If the reduction involves $\overline{n}\langle t\rangle$ or $n(x).Q$, then the hole of C is only under restrictions and parallel compositions. We assume without loss of generality that the hole in C is not under any restriction (otherwise we could move the corresponding restrictions into $\nu\tilde{a}$ since C does not bind $fn(Q)$ over its hole). Then $A' \equiv \nu n\tilde{a}.C[0]|Q\{t/x\} \equiv \nu n\tilde{a}.C[Q\{t/x\}] =: B' \equiv B$. Thus $B \rightarrow^* B'$ and $(A', B') \in \mathcal{R}$ (since $A' \approx B'$).

- If $(A,B) \in \mathcal{R}$ and $B \to B'$, then there is an A' with $A \to^* A'$ and $(A',B') \in \mathcal{R}$:

  The case $A \approx B$ is trivial. We thus assume that A, B are as in (1).

  If the reduction $B \to B'$ involves only C, then $B' \equiv \nu n\tilde{a}.\tilde{C}[Q\{t/x\}\sigma] \stackrel{(*)}{=} \nu n\tilde{a}.\tilde{C}[Q\sigma\{t/x\}]$ for some zero-or-one-hole context $\tilde{C}$. Here the substitution σ represents possible variable assignments performed over the hole of C (e.g., if $C = \overline{a}\langle T\rangle \mid a(y).\Box$, then $\sigma = \{T/y\}$). And the equality $(*)$ uses that $fn(Q) \subseteq x$.

  Then $A \to A' := \nu n\tilde{a}.\tilde{C}[\overline{n}\langle t\rangle\sigma] | n(x).Q$. Also, $\tilde{C}$ does not have more that one hole (in which case $\tilde{C}$ would not be a context) because the hole in C does not occur under a bang.

  Thus we have $(A',B') \in \mathcal{R}$.

  If the reduction $B \to B'$ involves $Q\{t/x\}$, then the hole of C is only under restrictions and parallel compositions. We assume without loss of generality that the hole in C is not under any restriction (otherwise we could move the corresponding restrictions into $\nu\tilde{a}$ since C does not bind $fn(Q)$ over its hole). Then $A \to \nu n\tilde{a}.C[0]|Q\{t/x\} \equiv \nu n\tilde{a}.C[Q\{t/x\}] \equiv B \to B' =: A'$. Thus trivially $(A',B') \in \mathcal{R}$ (since $A' = B'$ and thus $A' \approx B'$), and $A \to^* A'$.

- If E is an evaluation context, and $(A,B) \in \mathcal{R}$, then $(E[A], E[B]) \in \mathcal{R}$:

  The case $A \approx B$ is trivial. We thus assume that A, B are as in (1). Then $E[A] \equiv E[\nu n\tilde{a}.C[\overline{n}\langle t\rangle]|n(x).Q] \equiv \nu n\tilde{a}.C[\overline{n}\langle t\rangle]|P|n(x).Q$ for some process P up to renaming of the names $n, \tilde{a}$. And $E[B] \equiv E[\nu n\tilde{a}.C[Q\{t/x\}]] \equiv \nu n\tilde{a}.C[Q\{t/x\}]|P$. Thus (using the context $C|P$ instead of C), we have $(E[A], E[B]) \in \mathcal{R}$.

Thus $\mathcal{R}$ is a bisimulation. Thus $\nu n.C[\overline{n}\langle t\rangle]|n(x).Q \approx \nu n.C[Q\{t/x\}]$ (where n, C, t, x refer to the values from the statement of the lemma). And since n does not occur in C, Q, t, we have $\nu n.C[Q\{t/x\}] \approx C[Q\{t/x\}]$ by Lemma 3.2 (i). Thus $\nu n.C[\overline{n}\langle t\rangle]|n(x).Q \approx C[Q\{t/x\}]$. □

Lemma 3.4 *Let C, D be contexts, Q a process, n, m names, t, t' terms, and x a variable. Assume that C, D have no bang over their holes. Assume that $n, m \notin fn(C, D, Q, t, t')$. Assume that C, D do not bind $n, m, fn(Q)$. Assume that $fv(Q) \subseteq \{x\}$.*

Then $\nu n.(C[!\overline{n}\langle t\rangle] \mid D[n(x).Q]) \approx \nu m.(C[m().Q\{t/x\}] \mid D[\overline{m}\langle t'\rangle])$.

Proof. We define the relation $\mathcal{R}$ as follows: We have $(A,B) \in \mathcal{R}$ iff $A \approx B$ or there exist 0-1-contexts C, D without a bang over their holes and not binding $n, fn(Q)$, terms t, t', a name $n \notin fn(C, D, Q, t, t')$, a list of names $\tilde{a}$ not containing n, and an integer $i \geq 0$ such that

$$\begin{aligned} A &\equiv \nu n\tilde{a}.(C[\overline{n}\langle t\rangle^i \mid !\overline{n}\langle t\rangle] \mid D[n(x).Q]) \\ B &\equiv \nu n\tilde{a}.(C[n().Q\{t/x\}] \mid D[\overline{n}\langle t'\rangle]) \end{aligned} \tag{2}$$

Here $\overline{n}\langle t\rangle^i$ denotes $\overline{n}\langle t\rangle|\ldots|\overline{n}\langle t\rangle$ (i copies). Note: Q is the process from the statement of the lemma. (It is intentional that we use n in the definition of B, not m as in the statement of the lemma. We will rename n into m at the end of the proof.)

We show that $\mathcal{R}$ is a bisimulation. In all cases below, the case $A \approx B$ is trivial by the properties of $\approx$, so we assume in each case that A, B are as in (2).

- If $(A,B) \in \mathcal{R}$ and $A\downarrow_M$, then $B \to^*\downarrow_M$:

  Since n is bound, the output on M is not one of the $\overline{n}\langle t\rangle$ (here we use that $M \neq_{\mathrm{E}} n$ if $n \notin fn(M)$ by Lemma 3.1 (i)). Hence $C\downarrow_M$ or $D\downarrow_M$. Thus $B\downarrow_M$.

- If $(A,B) \in \mathcal{R}$ and $B\downarrow_M$, then $A \to^*\downarrow_M$:

  Since n is bound, the output on M is not $\overline{n}\langle t'\rangle$. Hence $C\downarrow_M$ or $D\downarrow_M$. Thus $A\downarrow_M$.

- If $(A,B) \in \mathcal{R}$ and $A \to A'$, then there is a B' such that $B \to^* B'$ and $(A',B') \in \mathcal{R}$:

  We distinguish the following cases:

  - $A \to A'$ is a reduction $!\overline{n}\langle t\rangle \to \overline{n}\langle t\rangle \mid !\overline{n}\langle t\rangle$: Then $A' \equiv \nu n\tilde{a}.(C[\overline{n}\langle t\rangle^{i+1} \mid !\overline{n}\langle t\rangle] \mid D[n(x).Q])$ and hence $(A',B') \in \mathcal{R}$ for $B' := B$.
  - $A \to A'$ is a reduction within C, within D, or a communication between C and D (in all cases not involving the argument of C,D): Then $A' \equiv \nu n\tilde{a}.(C'[\overline{n}\langle t\rangle^i \mid !\overline{n}\langle t\rangle] \mid D'[n(x).Q])$ for suitable contexts C', D' (satisfying all the conditions required for C, D in the definition of $\mathcal{R}$), and $B \to B' := \nu n\tilde{a}.(C'[n().Q\{t/x\}] \mid D'[\overline{n}\langle t'\rangle])$. (Note: This uses implicitly that Q has no free variables except x, otherwise Q might change in this reduction.)
  - $A \to A'$ is a communication between $\overline{n}\langle t\rangle$ and $n(x).Q$:
    Then C and D are evaluation contexts.
    Without loss of generality, we can assume that C, D do not bind any names over their holes: For this, we first rename the bound names in C, D such that they become distinct from all free names (possibly also renaming the names in t in the process, but not in Q since $fn(Q)$ are not bound), and then move the restrictions up into $\nu\tilde{a}$.
    Then $A' \equiv \nu n\tilde{a}.(C[\overline{n}\langle t\rangle^{i-1} \mid !\overline{n}\langle t\rangle] \mid D[Q\{t/x\}])$. Furthermore

    $$\begin{aligned} B' := B &\equiv \nu\ \tilde{a}.(C[0] \mid D[\nu n.(n().Q\{t/x\} \mid \overline{n}\langle t'\rangle)]) \\ &\overset{(*)}{\approx} \nu\tilde{a}.(C[0] \mid D[Q\{t/x\}]) \\ &\overset{(**)}{\approx} \nu\tilde{a}.(C[\nu n.(\overline{n}\langle t\rangle^{i-1} \mid !\overline{n}\langle t\rangle)] \mid D[Q\{t/x\}]) \equiv A' \end{aligned}$$

    Here $(*)$ follows from Lemma 3.3. And $(**)$ uses that $\nu n.(\overline{n}\langle t\rangle^{i-1} \mid !\overline{n}\langle t\rangle) \approx 0$, which can be seen by verifying that $\mathcal{R}' := \{(E[\nu n.(\overline{n}\langle t\rangle^{i-1} \mid !\overline{n}\langle t\rangle)], E[0]) : E \text{ evaluation context}\}$ is a bisimulation.
    Thus $A' \approx B'$ and hence $(A',B') \in \mathcal{R}$. And $B = B'$ implies $B \to^* B'$.
  - $A \to A'$ is a communication between C or D and $\overline{n}\langle t\rangle$ or $n(x).Q$: This case does not occur because $n \notin fn(C,D)$.

- If $(A,B) \in \mathcal{R}$ and $B \to B'$, then there is a A' such that $A \to^* A'$ and $(A',B') \in \mathcal{R}$:

  We distinguish the following cases:

  - $B \to B'$ is a reduction within C, within D, or a communication between C and D (in all cases not involving the argument of C,D): Then $B' = \nu n\tilde{a}.(C'[n().Q\{t/x\}] \mid D'[\overline{n}\langle t'\rangle])$ for suitable contexts C', D' (satisfying all the conditions required for C, D in the definition of $\mathcal{R}$), and $A \to A' \equiv \nu n\tilde{a}.(C'[\overline{n}\langle t\rangle^i \mid !\overline{n}\langle t\rangle] \mid D'[n(x).Q])$.
  - $B \to B'$ is a communication between $n().Q\{t/x\}$ and $\overline{n}\langle t'\rangle$:
    Then C, D are evaluation contexts.
    Without loss of generality, we can assume that C, D do not bind any names over their holes (analogous to the corresponding subcase of $A \to A'$ above).
    Then $B' \equiv \nu n\tilde{a}.(C[Q\{t/x\}] \mid D[0])$.
    Furthermore,

    $$\begin{aligned} A \to^* A' &:= \nu\tilde{a}.(C[\nu n.(\overline{n}\langle t\rangle^i \mid !\overline{n}\langle t\rangle)] \mid D[Q\{t/x\}]) \\ &\overset{(*)}{\approx} \nu\tilde{a}.(C[0] \mid D[Q\{t/x\}]) \equiv \nu\tilde{a}.(C[Q\{t/x\}] \mid D[0]) \overset{(**)}{\approx} B' \end{aligned}$$

    Here $(*)$ uses that $\nu n.(\overline{n}\langle t\rangle^i \mid !\overline{n}\langle t\rangle) \approx 0$ (see the corresponding subcase of $A \to A'$ above). And $(**)$ uses Lemma 3.2 (i). So $A' \approx B'$, hence $(A', B') \in \mathcal{R}$. Hence $A \to^* A'$ and $(A', B') \in \mathcal{R}$.
  - $B \to B'$ is a communication between C or D and $\overline{n}\langle t\rangle$ or $n(x).Q$: This case does not occur because $n \notin fn(C, D)$.

- If $(A,B) \in \mathcal{R}$ and E is an evaluation context, then $(E[A], E[B]) \in \mathcal{R}$:

  Then $E \equiv \nu\tilde{b}.(\Box \mid P)$ for some names $\tilde{b}$ and some process P.

  Without loss of generality, n does not occur in $\tilde{b}$ or $fn(P)$ (otherwise we rename n).

  Thus with $\tilde{a}' := \tilde{a}\tilde{b}$ and $C' := C \mid P$, we have

  $$\begin{aligned} E[A] &\equiv \nu n\tilde{a}'.(C'[\overline{n}\langle t\rangle^i \mid !\overline{n}\langle t\rangle] \mid D[n(x).Q]) \\ E[B] &\equiv \nu n\tilde{a}'.(C'[n().Q\{t/x\}] \mid D[\overline{n}\langle t'\rangle]) \end{aligned}$$

  Hence $(E[A], E[B]) \in \mathcal{R}$.

Under the conditions of the lemma, we have $(\nu n.C[!\overline{n}\langle t\rangle] \mid D[n(x).Q], \nu n.C[n().Q\{t/x\}] \mid D[\overline{n}\langle t'\rangle]) \in \mathcal{R}$ where C, D, Q, n, t, t', x are as in the statement of the lemma. Since $\mathcal{R}$ is a bisimulation, this implies

$$\nu n.C[!\overline{n}\langle t\rangle] \mid D[n(x).Q] \approx \nu n.C[n().Q\{t/x\}] \mid D[\overline{n}\langle t'\rangle] \equiv \nu m.C[m().Q\{t/x\}] \mid D[\overline{m}\langle t'\rangle])$$

□

Lemma 3.5 *Let A, B, C be closed processes. If $A \equiv_{\mathrm{E}} B \to C$, then there is a closed process B' such that $A \to B' \equiv_{\mathrm{E}} C$.*

Proof. It is easy to see that $\to$ is the smallest relation satisfying the following rules:

STREQ	If $P \equiv P' \to Q' \equiv Q$, then $P \to Q$
E-REPL	$E[!P] \to E[P \mid !P]$
E-COMM	$E[\overline{C}\langle T\rangle.P \mid C'(x).Q] \ \to\ E[P \mid Q\{T/x\}]$ if $C =_{\mathrm{E}} C'$
E-LET-THEN	$E[\text{let } x = D \text{ in } P \text{ else } Q] \ \to\ E[P\{M/x\}]$ if $D \Downarrow M$
E-LET-ELSE	$E[\text{let } x = D \text{ in } P \text{ else } Q] \ \to\ E[Q]$ if $\nexists M$ s.t. $D \Downarrow M$

Here in all rules E ranges over evaluation contexts with the following property: Let $E[R]$ denote the left hand side of the rule. Then all bound names in $E[R]$ are different from each other and from the free names in $E[R]$. (In a derivation of $\to$, we can always enforce this latter property by first using STREQ to alpha-rename the left hand side of the reduction.) We say $E[R]$ has no name conflicts.

For stating the next claim, we also need to introduce an asymmetric variant $\not\succeq$ of the structural equivalence $\equiv$. The difference is that in $\equiv$, we are allowed to apply the rule NEW-PAR in both directions, while in $\not\succeq$ we are only allowed to move restrictions up ($P \mid \nu u.Q \not\succeq \nu u.(P \mid Q)$), but not down (not: $\nu u.(P \mid Q) \not\succeq P \mid \nu u.Q$). More formally, $\not\succeq$ is the smallest transitive, reflexive (but not necessarily symmetric) relation closed under α-conversion, and closed under application of evaluation contexts, and satisfying the rules PAR-0, PAR-A, PAR-C, NEW-C, NEW-PAR from Figure 2 as well as the reversed rule PAR-0-rev (but not NEW-PAR-rev). (By reversed rule we mean the rules with left hand side and right hand side exchanged. E.g., PAR-0-rev says $P|0 \not\succeq P$. Note that PAR-C-rev and NEW-C-rev are not needed since PAR-C and NEW-C are symmetric. And PAR-A-rev follows from PAR-C and PAR-A via $(P|Q)|R \not\succeq R|(P|Q) \not\succeq (R|P)|Q \not\succeq Q|(R|P) \not\succeq (Q|R)|P \not\succeq P|(Q|R)$.)

Also, we define $\not\succeq_E$ analogously to $\equiv_{\mathrm{E}}$: $\not\succeq_E$ corresponds to a sequence of rewritings using $\not\succeq$ and $=_{\mathrm{E}}$, i.e., $\not\succeq_E := (\not\succeq \cup =_{\mathrm{E}})^*$.

Claim 1 *For closed processes A, B, C, if $A =_{\mathrm{E}} B \not\succeq C$, then there exists a closed process B' such that $A \not\succeq B' =_{\mathrm{E}} C$.*

We show this claim by induction over the derivation of $B \not\succeq C$. We distinguish the following cases:

- *α-conversion:* Then $B = C$ up to α-conversion. Hence $A =_{\mathrm{E}} B$ implies $A =_{\mathrm{E}} C$ since $=_{\mathrm{E}}$ allows α-conversions. Thus $A \not\succeq B^* =_{\mathrm{E}} C$ with $B^* := A$.
- *Closure under evaluation contexts:* Then $B = E[\tilde{B}]$ and $C = E[\tilde{C}]$ for processes $\tilde{B} \not\succeq \tilde{C}$ and an evaluation context E. And the induction hypothesis holds for $\tilde{B} \not\succeq \tilde{C}$. Since $A =_{\mathrm{E}} B = E[\tilde{B}]$, we have that $A = E^*[\tilde{B}^*\sigma]$ for some evaluation context $E^* =_{\mathrm{E}} E$, some process $\tilde{B}^* =_{\mathrm{E}} \tilde{B}$, and a renaming σ that corresponds to the alpha-renaming over the hole of E. Since $\tilde{B}^* =_{\mathrm{E}} \tilde{B}$, the induction hypothesis implies that $\tilde{B}^* \not\succeq \tilde{B}' =_{\mathrm{E}} \tilde{C}$ for some process $\tilde{B}'$. Hence

  $$A = E^*[\tilde{B}^*\sigma] \not\succeq E^*[\tilde{B}'\sigma] =_{\mathrm{E}} E[\tilde{B}'] =_{\mathrm{E}} E[\tilde{C}] = C.$$

  Thus $A \not\succeq B' =_{\mathrm{E}} C$ with $B' := E^*[\tilde{B}'\sigma]$.

- *Reflexivity:* Then $B = C$. Hence $A \not\cong B^* =_{\mathrm{E}} C$ with $B^* := A$.
- *Transitivity:* Then $B \not\cong S \not\cong C$ for some process S. And the induction hypothesis applies to $B \not\cong S$ and $S \not\cong C$. Since $A =_{\mathrm{E}} B \not\cong S$, by induction hypothesis, there is a process B' with $A \not\cong B' =_{\mathrm{E}} S$. Since $B' =_{\mathrm{E}} S \not\cong C$, by induction hypothesis there is a process S^* with $B' \not\cong S^* =_{\mathrm{E}} C$. Thus $A \not\cong S^* =_{\mathrm{E}} C$, and the claim follows with $B^* := S^*$.
- *PAR-0*: In this case, $C = B|0$ and $A =_{\mathrm{E}} B$. Hence $A \not\cong B^* =_{\mathrm{E}} C$ with $B^* := A|0$.
- *PAR-0-rev*: In this case, $B = C|0$ and $A =_{\mathrm{E}} B$. Hence $A = B^*|0$ for some process $B^* =_{\mathrm{E}} C$. Then $A \not\cong B^* =_{\mathrm{E}} C$.
- *PAR-A*: In this case, $B = B_1|(B_2|B_3)$ and $C = (B_1|B_2)|B_3$. Since $A =_{\mathrm{E}} B$, $A = A_1|(A_2|A_3)$ for some processes A_i with $A_i =_{\mathrm{E}} B_i$, $i = 1, 2, 3$. Then with $B^* := (A_1|A_2)|A_3$, we have $A \not\cong B^* =_{\mathrm{E}} C$.
- *PAR-C, PAR-C*: Analogous to PAR-A.
- *NEW-C*: In this case, $B = \nu nm.\hat{B}$ and $C = \nu mn.\hat{B}$ for some names n, m and a process $\hat{B}$. Since $A =_{\mathrm{E}} B$, we have that $A = \nu ab.\hat{A}$ for some names a, b and a process $\hat{A}$. (Not necessarily $ab = nm$, because $=_{\mathrm{E}}$ allows α-conversion.) Thus $\nu ab.\hat{A} =_{\mathrm{E}} \nu nm.\hat{B}$. This implies $\nu ba.\hat{A} =_{\mathrm{E}} \nu mn.\hat{B}$ (by induction over the derivation of $\nu ab.\hat{A} =_{\mathrm{E}} \nu nm.\hat{B}$). Hence with $B^* := \nu ba.\hat{A}$, we have that $A \not\cong B^* =_{\mathrm{E}} C$.
- *NEW-PAR*: Then $B = B_1|\nu n.B_2$ and $C = \nu n.(B_1|B_2)$ with $n \notin fn(B_1)$. Since $A =_{\mathrm{E}} B$, we have $A = A_1|\nu a.A_2$ for some name a and processes A_1, A_2 with $A_1 =_{\mathrm{E}} B_1$ and $\nu a.A_2 =_{\mathrm{E}} \nu n.B_2$. (Not necessarily $a = n$, because $=_{\mathrm{E}}$ allows α-conversion.) Let m be a fresh name, i.e., $m \notin fn(A_1, A_2, B_1, B_2)$. Let $B^* := \nu m.(A_1|A_2\{m/a\})$. Since $\nu n.B_2 =_{\mathrm{E}} \nu a.A_2$ and $m \notin fn(A_2, B_2)$, we have $\nu m.B_2\{m/n\} =_{\mathrm{E}} \nu m.A_2\{m/a\}$. Hence $\nu m.(A_1|B_2\{m/n\}) =_{\mathrm{E}} \nu m.(A_1|A_2\{m/a\})$. And using $A_1 =_{\mathrm{E}} B_1$, we get $\nu m.(B_1|B_2\{m/n\}) =_{\mathrm{E}} \nu m.(A_1|A_2\{m/a\}) = B^*$. Furthermore $C = \nu n.(B_1|B_2) =_{\mathrm{E}} \nu m.(B_1|B_2\{m/n\})$ since $n, m \notin fn(B_1)$, $m \notin fn(B_2)$. Thus $B^* =_{\mathrm{E}} C$. And $A = A_1|\nu a.A_2 \not\cong A_1|\nu m.A_2\{m/a\} \not\cong \nu m.(A_1|A_2\{m/a\}) = B^*$. Thus B^* is a process with $A \not\cong B^* =_{\mathrm{E}} C$.

This shows Claim 1.

Claim 2 *If $A \not\cong_E B$, then there exists an S such that $A \not\cong S =_{\mathrm{E}} B$.*

This follows directly from Claim 1.

Claim 3 *If B, C are closed processes and $B \to C$ (derived using the rules listed at the beginning of this proof), then for any closed A with $A \equiv_{\mathrm{E}} B$ there exists a closed B' with $A \to B' \equiv_{\mathrm{E}} C$.*

This claim will then immediately prove the lemma. We show the claim by induction over the derivation of $B \to C$. We distinguish the following rule applications:

- STREQ: Then $B \equiv \tilde{B} \to \tilde{C} \equiv C$ for some $\tilde{B}, \tilde{C}$, and the induction hypothesis holds for $\tilde{B} \to \tilde{C}$. Since $A \equiv_{\mathrm{E}} B \equiv \tilde{B}$, the induction hypothesis implies that $A \to B' \equiv_{\mathrm{E}} \tilde{C}$ for some closed B'. Since $\tilde{C} \equiv C$, we have $A \to B' \equiv_{\mathrm{E}} C$.

- E-REPL: Then $B = E[!\tilde{B}]$ and $C = E[\tilde{B} \mid !\tilde{B}]$ where E is an evaluation context and $E[!\tilde{B}]$ has no name conflicts. We have $A \equiv_{\mathrm{E}} E[!\tilde{B}]$. From this it follows that $A \not\cong_E E'[!\tilde{B}]$ where E' results from E by moving all unprotected restrictions to the top (no names in $\tilde{B}$ need to be renamed because $E[!\tilde{B}]$ has no name conflicts). By Claim 2, this implies that $A \not\cong S =_{\mathrm{E}} E'[!\tilde{B}]$ for some S. Hence $S = E''[!\tilde{B}'\sigma]$ where $E'' =_{\mathrm{E}} E'$ and $\tilde{B}' =_{\mathrm{E}} \tilde{B}$ and where σ is a renaming that corresponds to the alpha-conversions between E' and E'' over the hole. Thus $A \not\cong S \to E''[(\tilde{B}|!\tilde{B})\sigma] =_{\mathrm{E}} E'[\tilde{B} \mid !\tilde{B}] \equiv E[\tilde{B} \mid !\tilde{B}] = C$ and hence $A \to B' \equiv_{\mathrm{E}} C$ with $B' := E''[(\tilde{B}|!\tilde{B})\sigma]$.

- E-COMM: Then $B = E[\overline{M}\langle T\rangle.P \mid N(x).Q]$ and $C = E[P \mid Q\{T/x\}]$ where E is an evaluation context, $M =_{\mathrm{E}} N$, and B has no name conflicts. As in the E-REPL case, we have $A \not\cong_E E'[\overline{M}\langle T\rangle.P \mid N(x).Q]$ where E' results from E by moving all unprotected restrictions to the top. By Claim 2, this implies that $A \not\cong S =_{\mathrm{E}} E'[\overline{M}\langle T\rangle.P \mid N(x).Q]$ for some S. Hence $S = E''[(\overline{M'}\langle T'\rangle.P' \mid N'(x).Q')\sigma]$ where $E'' =_{\mathrm{E}} E'$, $M' =_{\mathrm{E}} M$, $T' =_{\mathrm{E}} T$, $P' =_{\mathrm{E}} P$, $N' =_{\mathrm{E}} N$, $Q' =_{\mathrm{E}} Q$, and σ is as in the case of E-REPL. Then

  $$A \not\cong S \to E''[P' \mid Q'\{T'/x\}\sigma] =_{\mathrm{E}} E'[P \mid Q\{T'/x\}] \stackrel{(*)}{=_{\mathrm{E}}} E'[P \mid Q\{T/x\}] \equiv E[P \mid Q\{T/x\}] = C.$$

  (Note that $(*)$ also uses the fact that $=_{\mathrm{E}}$ may also rewrite terms that are subterms of destructor terms; this is needed if x occurs in a destructor term in Q.)

  Hence $A \to B' \equiv_{\mathrm{E}} C$ for $B' := E''[P' \mid Q'\{T'/x\}\sigma]$.

- E-LET-THEN: Then $B = E[\text{let } x = D \text{ in } P \text{ else } Q]$ and $C = E[P\{M/x\}]$ where E is an evaluation context, $D \Downarrow M$, and B has no name conflicts. As in the E-REPL case, we have $A \not\cong_E E'[\text{let } x = D \text{ in } P \text{ else } Q]$ where E' results from E by moving all unrestricted restrictions to the top. By Claim 2, this implies that $A \not\cong S =_{\mathrm{E}} E'[\text{let } x = D \text{ in } P \text{ else } Q]$ for some S. Hence $S = E''[(\text{let } x = D' \text{ in } P' \text{ else } Q')\sigma]$ where $E'' =_{\mathrm{E}} E'$, $D' =_{\mathrm{E}} D$, $P' =_{\mathrm{E}} P$, $Q' =_{\mathrm{E}} Q$, and σ is as in the case of E-REPL. Then $D' =_{\mathrm{E}} D$ and $DM \Downarrow$ imply $D'M \Downarrow'$ for some $M' =_{\mathrm{E}} M$. Hence $(\text{let } x = D' \text{ in } P' \text{ else } Q') \to P'\{M'/x\}$. Then

  $$A \not\cong S \to E''[P'\{M'/x\}\sigma] =_{\mathrm{E}} E'[P\{M'/x\}] \stackrel{(*)}{=_{\mathrm{E}}} E'[P\{M/x\}] \equiv E[P\{M/x\}] = C.$$

  (Here $(*)$ again uses that $=_{\mathrm{E}}$ rewrites destructor terms, see the case E-COMM.)

  Hence $A \to B' \equiv_{\mathrm{E}} C$ for $B' := E''[P'\{M'/x\}\sigma]$.

- E-LET-ELSE: Then $B = E[\text{let } x = D \text{ in } P \text{ else } Q]$ and $C = E[Q]$ where E is an evaluation context, $\forall M.\ D \not\Downarrow M$, and B has no name conflicts. As in the E-REPL case, we have $A \not\cong_E E'[\text{let } x = D \text{ in } P \text{ else } Q]$ where E' results from E by moving all unrestricted restrictions to the top. By Claim 2, this implies that $A \not\cong S =_{\mathrm{E}} E'[\text{let } x = D \text{ in } P \text{ else } Q]$ for some S. Hence $S = E''[(\text{let } x = D' \text{ in } P' \text{ else } Q')\sigma]$ where $E'' =_{\mathrm{E}} E'$, $D' =_{\mathrm{E}} D$, $P' =_{\mathrm{E}} P$, $Q' =_{\mathrm{E}} Q$, and σ is as in the case of E-REPL. Since $D' =_{\mathrm{E}} D$ and $\forall M.\ D \not\Downarrow M$, we have $\forall M.\ D' \not\Downarrow M$. Hence $(\text{let } x = D' \text{ in } P' \text{ else } Q') \to Q'$. Then

  $$A \not\cong S \to E''[Q'\sigma] =_{\mathrm{E}} E'[Q] \equiv E[Q] = C.$$

Hence $A \to B' \equiv_{\mathrm{E}} C$ for $B' := E''[Q'\sigma]$.

This shows Claim 3. And from that claim the lemma follows. □

3.1 Relating events and observational equivalence

For stating Lemma 3.7 below, we will need processes containing events. The variant of the applied pi calculus presented in Section 2 (which is used by Proverif for observational equivalence proofs) does not support events. When using Proverif for showing trace properties defined in terms of events, a different variant of the applied pi calculus is used [Bla09]. We will call processes in that calculus *event processes*. Syntactically, event processes differ from processes as in Figure 1 only by an additional construct *event* $f(t_1, \ldots, t_n).P$ which means that the event f is raised, with arguments $t_1, \ldots, t_n$ (these are normal terms), and then the event process P is executed.

The semantics of event processes are formulated in [Bla09] in a different way from the semantics used here. Fortunately, we will be able to encapsulate everything that we need to know about that semantics in Lemma 3.6 below, so we do not need to repeat those semantics here.

Instead, we extend the definition of the internal reduction relation $\to$ to event processes. $\to$ is defined as in Definition 2.3, except that we add the following rule:

$$\text{EVENT:} \quad \mathit{event}\ f(t_1, \ldots, t_n).P \to P$$

The semantics defined by $\to$ will be related to those from [Bla09] by Lemma 3.6 below.

Finally, [Bla09] defines the concept of a *trace property*. We will only need trace properties of a specific form, namely

$$\mathit{end}(x) \Rightarrow \mathit{start}(x) \vee x = t_1 \vee \cdots \vee x = t_n$$

Intuitively, an event process P *satisfies a trace property* $\mathit{end}(x) \Rightarrow \mathit{start}(x) \vee x = t_1 \vee \cdots \vee x = t_n$ if in any execution $P|R \to P_1 \to \ldots \to P_n$, we have that if one of the transitions raises the event $\mathit{end}(t)$, then $t \in \{t_1, \ldots, t_n\}$ and in the same trace, the event $\mathit{start}(t)$ is also raised (for any adversarial R not containing events).

Formally, satisfying a trace property is defined with respect to the semantics from [Bla09].[8] Instead of giving those semantics here, we present the following lemma which summarizes seven fact about that definition. We will not use any other facts. The facts can be verified by inspecting the semantics and definitions from [Bla09].

[8] Strictly speaking, the semantics described in [Bla09] does not allow expressions of the form $x = t_i$ in trace properties. Such expressions are, however, supported by Proverif. Also, [Bla09, footnote 3 in the full version] explains how to encode such equality tests in the trace properties supported by [Bla09]. In their notation, our trace property becomes the somewhat less readable trace property: $\mathit{end}(x) \Rightarrow (\mathit{end}(x) \rightsquigarrow \mathit{start}(x)) \vee (\mathit{end}(t_1) \rightsquigarrow \mathit{true}) \vee \cdots \vee (\mathit{end}(t_n) \rightsquigarrow \mathit{true})$.

Also, the semantics described [Bla09] do not support equations (i.e., $t =_{\mathrm{E}} t'$ iff $t = t'$ in their semantics). However, Proverif supports these, so we assume the intended semantics of Proverif is that of [Bla09] with the natural extension of equality tests to equality modulo $=_{\mathrm{E}}$.

Lemma 3.6 *Let $t_1, \ldots, t_n$ be terms. Let $\wp$ stand for the trace property $start(x) \Rightarrow end(x) \vee x = t_1 \vee \cdots \vee x = t_n$. Let P be an event process.*

(i) If $P \equiv P'$ and P satisfies $\wp$, then P' satisfies $\wp$.

(ii) Assume $P \to P'$ and P satisfies $\wp$ and the reduction $P \to P'$ does not use the EVENT rule. Then P' satisfies $\wp$.

(iii) Let t be a closed term. Assume $P = C[event\ start(t).Q]$ where C is an event context not binding $fn(t)$ over its hole. Assume that P satisfies $\wp$. Then $P' := C[Q]$ satisfies $\wp \vee x = t$.

(iv) Assume $P = C[event\ end(t).Q]$ where C is an event context. Assume that P satisfies $\wp$. Then $P' := C[Q]$ satisfies $\wp$.

(v) Assume P satisfies $\wp$ and E is an evaluation context (not containing events) and E does not bind $fn(t_1, \ldots, t_n)$ over its hole. Then $E[P]$ satisfies $\wp$.

(vi) Assume E is an evaluation event context that does not bind any names over its hole. Assume $P = E[event\ end(t).Q]$. Assume that P satisfies $\wp$. Then $t =_{\mathrm{E}} t_i$ for some i.

(vii) If $\nu a.P$ satisfies $\wp$, then P satisfies $\wp$.

We explain the intuitive reason for each fact:

(i) Structurally equivalent processes behave identically and thus raise the same events.

(ii) If $P \to P'$ without raising an event, then for any event trace that P' may produce, P may produce the same by first reducing to P'.

(iii) P' has the same event traces as P, except that some $start(t)$-events are removed. If P' does not satisfy $\wp \vee x = t$, then there must be an event $end(t')$ with $t \neq t'$ that is not preceded by a $start(t')$-event. But then also in a trace of P, there would be an $end(t')$-event not preceded by $start(t')$ (since the traces only differ in their $start(t)$-events and $start(t) \neq start(t')$).

(iv) P' has the same event traces as P, except that various $end(\cdot)$-events are removed. (Since t is not necessarily closed, $end(t)$ may be instantiated to different $end(\cdot)$-events.) If a trace of P' does not satisfy $\wp$, this means there was an $end(t')$-event not preceded by a $start(t')$ event. Then also in P the corresponding $end(t')$-event is not preceded by a $start(t')$-event, as P has the same $start(\cdot)$-events, and more $end(\cdot)$-events.

(v) The semantics of satisfying trace properties are defined with respect to P running in parallel with an adversary R not containing events. Thus the case of an evaluation context running with P is already covered. (It is important that E does not bind $fn(t_1, \ldots, t_n)$ because otherwise the terms $t_1, \ldots, t_n$ occurring in the process would be considered different from those in $\wp$.)

(vi) There is a trace of P that consists only of an $end(t)$-event. That trace does not satisfy $end(t) \Rightarrow start(t)$. Thus it satisfies $\wp$ only if $\wp$ contains $x = t$ as one of its clauses.

(vii) $\nu a.P$ has the same traces as P, except that occurrences of a in the P-traces are replaced by a fresh restricted name a'. Thus, if P does not satisfy $\wp$, then there is a trace containing an $end(t)$-event without preceding $start(t)$-event such that $t \notin \{t_1, \ldots, t_n\}$. In the corresponding $\nu a.P$-trace, we have an $end(t\{a'/a\})$-event without preceding $start(t\{a'/a\})$-event. Since $t \notin \{t_1, \ldots, t_n\}$ and a is fresh, also $t\{a'/a\} \notin \{t_1, \ldots, t_n\}$. Hence the $\nu a.P$-trace does not satisfy $\wp$, either.

Lemma 3.7 *Let s be a name. Let P be a process containing s only in constructs of the form $(!\overline{(s,t)}\langle t'\rangle)|P'$ and $(s,t)().P'$ (for arbitrary and possibly different t, t', P').*

Let $plain^s(P)$ denote the process resulting from P by replacing all occurrences $!\overline{(s,t)}\langle t'\rangle|P'$ and $(s,t)().P'$ by P'.

Let $ev^s(P)$ denote the process resulting from P by replacing all occurrences $!\overline{(s,t)}\langle t'\rangle|P'$ by event $start(t).P'$ and $(s,t)().P'$ by event $end(t).P'$.

Assume that $ev^s(P)$ satisfies the trace property $end(x) \Rightarrow start(x)$.

Then $plain^s(P) \approx \nu s.P$.

Proof. We call a process P s-well-formed if it contains s only in constructs of the form $!\overline{(s,t)}\langle t'\rangle|P'$ and $(s,t)().P'$ (for arbitrary and possibly different t, t', P'). Given a multiset $T = \{t_1 \mapsto t'_1, \ldots, t_n \mapsto t'_n\}$ with t_i, t'_i terms, we call an event-process P T-good if P satisfies the trace property $end(x) \Rightarrow start(x) \vee x = t_1 \vee \cdots \vee x = t_n$.

For example, the process P from the statement of the lemma is s-well-formed, and $ev^s(P)$ is $\varnothing$-good.

We define the following relation $\mathcal{R}$ (up to structural equivalence):

$$\mathcal{R} := \Big\{ \big(\nu \underline{a}.plain^s(P), \quad \nu \underline{a} s.(P \mid !\overline{(s,t_1)}\langle t'_1\rangle \mid \cdots \mid !\overline{(s,t_n)}\langle t'_n\rangle \mid \overline{(s,u_1)}\langle u'_1\rangle \mid \cdots \mid \overline{(s,u_m)}\langle u'_m\rangle\big)$$
$$P \text{ } s\text{-well-formed}, \quad s, \underline{a} \text{ distinct names}, \quad ev^s(P) \text{ is } \{t_1, \ldots, t_n\}\text{-good} \Big\}$$

Here $P, n, m, t_i, t'_i, u_i, u'_i, s, \underline{a}$ refer to arbitrary values, not only to the values P, s from the statement of the lemma.

We write short $syncout^s(\{t_1 \mapsto t'_1, \ldots, t_n \mapsto t'_n\}; \{u_1 \mapsto u'_1, \ldots, u_n \mapsto u'_n\})$ for $!\overline{(s,t_1)}\langle t'_1\rangle \mid \cdots \mid !\overline{(s,t_n)}\langle t'_n\rangle \mid \overline{(s,u_1)}\langle u'_1\rangle \mid \cdots \mid \overline{(s,u_m)}\langle u'_m\rangle$.

We now show that $\mathcal{R}$ is a bisimulation:

- If $(A,B) \in \mathcal{R}$, and $A \downarrow_M$, then $B \downarrow_M$:

  Then $A = \nu \underline{a}.plain^s(P)$. Hence $plain^s(P) \downarrow_M$ and $\underline{a} \notin fn(M)$. Also, $s \notin fn(plain^s(P))$, so $s \notin fn(M)$. By definition of $plain^s(\cdot)$, $plain^s(P) \downarrow_M$ implies $P \downarrow_M$. Since $\underline{a}, s \notin fn(M)$, it follows $B = \nu \underline{a} s.(P|\ldots) \downarrow_M$.

- If $(A,B) \in \mathcal{R}$, and $B \downarrow_M$, then $A \downarrow_M$:

  Then $B = \nu \underline{a} s.(P|syncout^s(T;U))$. Thus $\underline{a}, s \notin fn(M)$ and $P|syncout^s(T;U) \downarrow_M$. Since all channels in $syncout^s(T;U)$ are of the form $(s,\cdot)$, we have $syncout^s(T;U) \not\downarrow_M$.[9] Hence $P \downarrow_M$. By definition of $plain^s(P)$ and since M does not contain s, this implies $plain^s(P) \downarrow_M$. Hence $A = \nu \underline{a}.plain^s(P) \downarrow_M$.

[9] Here we implicitly use the fact that $(s,\cdot) \neq_{\mathrm{E}} M$ for any M not containing s (Lemma 3.1 (iii)).

- If $(A, B) \in \mathcal{R}$, and $A \to A'$, then there exists a B' with $B \to^* B'$ and $(A', B') \in \mathcal{R}$:

  Then $A \equiv \nu \underline{a}.\mathit{plain}^s(P)$ and $B \equiv \nu \underline{a} s.(P | \mathit{syncout}^s(T; U))$. We call an event process *name-reduced*, if it does not contain unprotected restrictions.

  Without loss of generality, assume that P (and hence also $ev^s(P)$) is name-reduced (otherwise we could move the superfluous restrictions into the $\nu \underline{a}$).

  Let $\underline{a}_0 := \underline{a}$ and $P_0 := P$ and $T_0 := T$. We first construct a sequence $P_1, \ldots, P_k$ of processes and a sequence of lists of names $\underline{a}_1, \ldots, \underline{a}_k$, and a sequence of sets $T_1, \ldots, T_k$ such that P_k does not contain unprotected inputs $(s, \cdot)().Q$ or unprotected outputs $!\overline{(s, \cdot)}\langle\cdot\rangle$, and for all $i = 0, \ldots, k$ we have:

  (a) $\nu s.(P | \mathit{syncout}^s(T; U)) \to^* \nu \underline{a}_i s.(P_i | \mathit{syncout}^s(T_i; U))$, and
  (b) $ev^s(P_i)$ is T_i-good, and
  (c) $\mathit{plain}^s(P) \equiv \nu \underline{a}_i.\mathit{plain}^s(P_i)$.
  (d) P_i is s-well-formed.

  For $i = 0$, these conditions are trivially satisfied. When constructing P_i for $i > 0$, we already have a process P_{i-1} satisfying these conditions. We distinguish three cases:

  - If P_{i-1} does not contain unprotected inputs $(s, \cdot)()$, we are done ($k := i - 1$).
  - If P_{i-1} does contain an unprotected input $(s, t)()$ that is not part of a subterm of the form $!\overline{(s, \cdot)}\langle\cdot\rangle | Q$, then we can write P_{i-1} as $P_{i-1} = \nu \underline{b}.E[(s, t)().P']$ for some names $\underline{b}$ and some evaluation context E that has no restrictions over its hole. Since $(s, t)()$ is not part of a subterm of the form $!\overline{(s, \cdot)}\langle\cdot\rangle | Q$, $ev^s(E)$ is an evaluation context ($!\overline{(s, \cdot)}\langle\cdot\rangle | Q$ would have translated to *event start*$(\cdot).Q$). Without loss of generality, $\underline{b} \cap \mathit{fn}(T_{i-1}, U) = \varnothing$.

    Since $ev^s(P_{i-1}) \equiv \nu \underline{b}.ev^s(E)[\mathit{event}\ \mathit{end}(t).ev^s(P')]$ is T_{i-1}-good by (b), Lemma 3.6 (vii) implies that $ev^s(E)[\mathit{event}\ \mathit{end}(t).ev^s(P')]$ is T_{i-1}-good. Since E does not bind any names over its hole, Lemma 3.6 (vi) implies that $t =_{\mathrm{E}} t^*$ for some $t^* \in T_{i-1}$. Thus $P_{i-1} | \mathit{syncout}^s(T_{i-1}; U) \equiv (\nu \underline{b}.E[(s, t)().P']) | \mathit{syncout}(T_{i-1}; U) \to^* (\nu \underline{b}.E[P']) | \mathit{syncout}(T_{i-1}; U)$. Since without loss of generality, $\underline{b} \cap \mathit{fn}(T_{i-1}, U) = \varnothing$, $(\nu \underline{b}.E[P']) | \mathit{syncout}(T_{i-1}; U) \equiv \nu \underline{b}.P_i | \mathit{syncout}(T_{i-1}; U)$ with $P_i := E[P']$. Hence $\nu s.P | \mathit{syncout}^s(T; U) \overset{(a)}{\to}{}^* \nu \underline{a}_{i-1} s.(P_{i-1} | \mathit{syncout}^s(T_{i-1}; U)) \to^* \nu \underline{a}_{i-1} \underline{b} s.P_i | \mathit{syncout}^s(T_{i-1}; U) \equiv \nu \underline{a}_i s.P_i | \mathit{syncout}^s(T_i; U)$ with $T_i := T_{i-1}$ and $\underline{a}_i := \underline{a}_{i-1} \underline{b}$. Thus (a) is satisfied by $P_i, \underline{a}_i, T_i$.

    Since $ev^s(P_{i-1}) \equiv \nu \underline{b}.ev^s(E)[\mathit{event}\ \mathit{end}(t).ev^s(P')]$ is T_{i-1}-good by (b) and thus T_i-good, we have by Lemma 3.6 (vii) that $ev^s(E)[\mathit{event}\ \mathit{end}(t).ev^s(P')]$ is T_i-good. Since E does not bind names over its hole, neither does $ev^s(E)$. Thus by Lemma 3.6 (iv), $ev^s(E)[ev^s(P')] = ev^s(P_i)$ is T_i-good. Thus (b) is satisfied by $P_i, \underline{a}_i, T_i$.

    Since $P_{i-1} = \nu \underline{b}.E[(s, t)().P']$ is s-well-formed by (d), so is $P_i = E[P']$. Thus (d) is satisfied by $P_i, \underline{a}_i, T_i$.

Finally, $plain^s(P_{i-1}) = \nu\underline{b}.plain^s(E)[plain^s(P')] = \nu\underline{b}.plain^s(P_i)$. Since by (c) we have that $plain^s(P) \equiv \nu\underline{a}_{i-1}.plain^s(P_{i-1})$, we have $plain^s(P) \equiv \nu\underline{a}_i.plain^s(P_i)$. Thus (c) is satisfied by $P_i, \underline{a}_i, T_i$.

– If P_{i-1} contains an unprotected output $!\overline{(s,t)}\langle t'\rangle$ that is not part of a subterm of the form $!\overline{(s,\cdot)}\langle\cdot\rangle|Q$, then we can write P_{i-1} as $P_{i-1} = \nu\underline{b}.E[\overline{(s,t)}\langle t'\rangle|P']$ for some names $\underline{b}$ and some evaluation context E that has no restrictions over its hole. Since $\overline{(s,t)}\langle t'\rangle$ is not part of a subterm of the form $!\overline{(s,\cdot)}\langle\cdot\rangle|Q$, $ev^s(E)$ is an evaluation context ($!\overline{(s,\cdot)}\langle\cdot\rangle|Q$ would have translated to $event\ start(\cdot).Q$). Without loss of generality, $\underline{b} \cap fn(T_{i-1}, U) = \varnothing$.

  We have $P_{i-1}|syncout^s(T_{i-1};U) \equiv (\nu\underline{b}.E[!\overline{(s,t)}\langle t'\rangle|P'])|syncout(T_{i-1};U) \overset{(*)}{\equiv} \nu\underline{b}.(E[!\overline{(s,t)}\langle t'\rangle|P']|syncout(T_{i-1};U)) \equiv \nu\underline{b}.(E[P']|syncout(T_i;U))$ with $T_i := T_{i-1} \cup \{t \mapsto t'\}$. Here $(*)$ uses that $\underline{b} \cap fn(T_{i-1}, U) = \varnothing$. Hence $\nu s.P|syncout^s(T;U) \overset{(a)}{\to}{}^* \nu\underline{a}_{i-1}s.(P_{i-1}|syncout^s(T_{i-1};U)) \to^* \nu\underline{a}_{i-1}s\underline{b}.(E[P']|syncout^s(T_i;U)) \equiv \nu\underline{a}_i s.(P_i|syncout^s(T_i;U))$ with $P_i := E[P']$ and $\underline{a}_i := \underline{a}_{i-1}\underline{b}$ (remember that $T_i = T_{i-1} \cup \{t \mapsto t'\}$. Thus (a) is satisfied by $P_i, \underline{a}_i, T_i$.

  Since $ev^s(P_{i-1}) \equiv \nu\underline{b}.ev^s(E)[event\ start(t).ev^s(P')]$ is T_{i-1}-good by (b), we have by Lemma 3.6 (vii) that $ev^s(E)[event\ start(t).ev^s(P')]$ is T_{i-1}-good. Since E does not bind names over its hole, neither does $ev^s(E)$. Thus by Lemma 3.6 (iii), $ev^s(E)[ev^s(P')] = ev^s(P_i)$ is T_i-good. Thus (b) is satisfied by $P_i, \underline{a}_i, T_i$.

  Since $P_{i-1} = \nu\underline{b}.E[\overline{(s,t)}\langle t'\rangle.P']$ is s-well-formed by (d), so is $P_i = E[P']$. Thus (d) is satisfied by $P_i, \underline{a}_i, T_i$.

  That (c) is satisfied by $P_i, \underline{a}_i, T_i$ is shown as in the previous case.

Note that in the last two cases, the size of P_i is smaller than that of P_{i-1}, so we eventually reach the first case. Hence the construction terminates and we get a process P_k that satisfies (a)–(d) and that does not contain unprotected inputs $(s,\cdot)()$ or unprotected outputs $!\overline{(s,\cdot)}\langle\cdot\rangle$. We have $A \equiv \nu\underline{a}.plain^s(P) \overset{(c)}{\equiv} \nu\underline{a}\underline{a}_k.plain^s(P_k)$. Thus $A \to A'$ implies that $\nu\underline{a}\underline{a}_k.plain^s(P_k) \to A'$ and and thus $plain^s(P_k) \to A''$ where A'' is A' with the restrictions $\nu\underline{a}\underline{a}_k$ removed. (I.e. $A' \equiv \nu\underline{a}\underline{a}_k.A''$.) Since P_k is s-well-formed by (d) and does not contain unprotected inputs $(s,\cdot)()$ or unprotected outputs $!\overline{(s,\cdot)}\langle\cdot\rangle$, by inspection of the definition of $plain^s$, ev^s, and $\to$, it follows that $P_k \to P'$ and $ev^s(P_k) \to ev^s(P')$ for some s-well-formed P' with $plain^s(P') \equiv A''$. The reduction $ev^s(P_k) \to ev^s(P')$ does not use the EVENT rule. Since $ev^s(P_k)$ is T_k-good by (b), from Lemma 3.6 (ii) we have that $ev^s(P')$ is T_k-good. Let $B' := \nu\underline{a}\underline{a}_k s.(P'|syncout^s(T_k;U))$. Then $(A', B') \equiv (\nu\underline{a}\underline{a}_k.plain^s(P'), B') \in \mathcal{R}$. Finally, $B = \nu\underline{a}s.(P|syncout^s(T;U)) \overset{(a)}{\to}{}^* \nu\underline{a}\underline{a}_k s.(P_k|syncout^s(T_k;U)) \to \nu\underline{a}\underline{a}_k s.(P'|syncout^s(T_k;U)) = B'$.

- If $(A, B) \in \mathcal{R}$, and $B \to B'$, then there exists an A' with $A \to^* A'$ and $(A', B') \in \mathcal{R}$:

We have $A \equiv \nu \underline{a}.plain^s(P)$ and $B \equiv \nu \underline{a}s.(P|syncout^s(T;U))$ for some s-well-formed P and T-good $ev^s(P)$.

We distinguish three cases for $B \to B'$:

- $B \to B'$ is a reduction within $syncout^s(T;U)$:
  In this case, the reduction of the form $E[!\overline{(s,t)}\langle t'\rangle] \to E[\overline{(s,t)}\langle t'\rangle|!\overline{(s,t)}\langle t'\rangle]$ for some t, t'. Thus $B' \equiv \nu \underline{a}s.(P|syncout^s(T; U \cup \{t \mapsto t'\}))$. Then $A = A' := \nu \underline{a}.plain^s(P)$ and $ev^s(P)$ is T-good. Hence $A \to^* A'$ and $(A', B') \in \mathcal{R}$.
- $B \to B'$ is a COMM reduction between P and $syncout^s(T;U)$:
  Then for some terms t, t', some process Q, and some evaluation context E, we have $P \equiv E[(s,t)().Q]$ for some t, t', and $B' \equiv \nu \underline{a}s.(P'|syncout^s(T;U'))$ with $P' := E[Q]$ and U' with $U = U' \cup \{t \mapsto t'\}$. Since $plain^s((s,t)().Q) = plain^s(Q)$, we have $A \equiv A' := \nu \underline{a}.plain^s(P')$. Furthermore, $ev^s(P) = ev^s(E)[event\ end(t).ev^s(Q)]$ and $ev^s(P') = ev^s(E)[ev^s(Q)]$. Thus by Lemma 3.6 (iv), the fact that $ev^s(P)$ is T-good implies that $ev^s(P')$ is T-good. Hence $A \to^* A'$ and $(A', B') \in \mathcal{R}$.
- $B \to B'$ is a reduction within P.
  Thus $P \to P'$ for some P', and $B' \equiv \nu \underline{a}s.(P'|syncout^s(T;U))$. Since P is s-well-formed, we have $P \equiv E[Q] \to E[Q'] \equiv P'$ for some evaluation context E and process Q, such that Q is of the form $!\overline{(s,t)}\langle t'\rangle|Q_1$, or Q is a redex not of the form $!\overline{(s,\cdot)}\langle\cdot\rangle$, or $Q = \overline{M}\langle N\rangle.Q_1|M'(x).Q_2$ with $M \neq_E (s,\cdot)$. (We cannot have a reduction on a channel $(s,\cdot)$, since s-well-formed terms have outputs on such channels only below bangs.) Without loss of generality, we can assume that all unprotected occurrences of $!\overline{(s,t)}\langle t'\rangle$ in E are not below a restriction (otherwise we could move these restrictions from E to $\nu \underline{a}$).
  Let E^* be E with all unprotected occurrences of $!\overline{(s,t)}\langle t'\rangle$ removed (for arbitrary t, t'). Let T^* be the multiset of the pairs $(t \mapsto t')$ from these occurrences. Then $E[Q] \equiv E^*[Q]|syncout^s(T^*;\varnothing)$. Since $ev^s(P) = ev^s(E[Q])$ is T-good, and since $ev^s(E^*[Q])$ results from $ev^s(P)$ by removing $event\ start(t)$ for all $(t \mapsto \cdot) \in T^*$, by Lemma 3.6 (iii) we have that $ev^s(E^*[Q])$ is $T \cup T^*$-good.
  We now distinguish on the form of Q:
  - If $Q = !\overline{(s,t)}\langle t'\rangle|Q_1$:
    Then $B' \equiv \nu \underline{a}s.(E^*[Q_1]|syncout^s(T';U'))$ for $T' := T \cup T^* \cup \{t \mapsto t'\}$ and $U' := U \cup \{t \mapsto t'\}$, and $A' := \nu \underline{a}s.plain(E^*[Q_1]) = \nu \underline{a}s.plain(E^*[!\overline{(s,t)}\langle t'\rangle|Q_1]) \equiv A$. And since $ev^s(E^*[Q]) = ev^s(E^*)[event\ start(t).ev^s(Q_1)]$ is $T \cup T^*$-good, we have that $ev^s(E^*[Q]) \equiv ev^s(E^*)[ev^s(Q_1)]$ is T'-good by Lemma 3.6 (iii). Thus $A \to^* A'$ and $(A', B') \in \mathcal{R}$.
  - If Q is a redex, or $Q = \overline{M}\langle N\rangle.Q_1|M'(x).Q_2$ with $M =_{\mathrm{E}} M'$ and $M \neq_E (s,\cdot)$:
    Then $B' \equiv \nu \underline{a}s.(P'|syncout^s(T';U))$ with $P' = E^*[Q']$ and $Q \mapsto Q'$ and $T' := T \cup T^*$. And $A \to A' := \nu \underline{a}.plain(P')$. And $ev^s(Q) \to ev^s(Q')$.

Since E^* is an evaluation context and does not contain unprotected $!\overline{(s,t)}\langle t'\rangle$, we have that $ev^s(E^*)$ is an event evaluation context. Hence $ev^s(E^*[Q]) = ev^s(E^*)[ev^s(Q)] \to ev^s(E^*)[ev^s(Q')] = ev^s(P')$, not using the EVENT rule. By Lemma 3.6 (ii) and using that $ev^s(E^*[Q])$ is T'-good, this implies that $ev^s(P')$ is T'-good, too. Thus $A \to^* A'$ and $(A', B') \in \mathcal{R}$.

- If $(A, B) \in \mathcal{R}$, and E is an evaluation context, then $(E[A], E[B]) \in \mathcal{R}$:

  We have $A \equiv \nu \underline{a}.plain^s(P)$ for some s-well-formed P. And $B \equiv \nu \underline{a}s.(P \mid syncout^s(T;U))$ for some sets T, U. And $ev^s(P)$ is T-good. Without loss of generality, $\underline{a}, s$ do not occur in E (neither bound nor free). Let $\nu \underline{b}.E'$ be E with all restrictions over the hole moved up into $\underline{b}$. Then $E[A] \equiv \nu \underline{b}.E'[A]$ and $E[B] \equiv \nu \underline{b}.E'[B]$.

  Since P is s-well-formed, and E and hence E' does not contain s, $E'[P]$ is s-well-formed.

  Since E does not contain $\underline{a}, s$, we have that $\underline{ab}s$ are distinct names.

  Since $ev^s(P)$ is T-good, by Lemma 3.6 (v) we have $ev^s(E'[P])) = E'[ev^s(P)]$ is T-good. (We use the fact that E' does not bind the $fn(T)$ as they have been moved into $\nu \underline{b}$.)

  Thus $(\nu \underline{ab}.plain^s(E'[P]), \nu \underline{ab}s.(E'[P] | syncout^s(T;U))) \in \mathcal{R}$ with $E'[P]$ instead of P and $\underline{ab}$ instead of $\underline{a}$.

  By definition of $plain^s(\cdot)$, $E[A] \equiv \nu \underline{b}.E'[A] \equiv \nu \underline{b}.E'[\nu \underline{a}.plain^s(P)] = \nu \underline{ab}.plain^s(E'[P])$. And $E[B] \equiv \nu \underline{b}.E'[B] \equiv \nu \underline{b}.E[\nu \underline{a}s.(P | syncout^s(T;U))] \equiv \nu \underline{ab}s.(E'[P] | syncout^s(T;U))$.

  Since $\mathcal{R}$ is closed under structural equivalence, this implies that $(E[A], E[B]) \in \mathcal{R}$.

Since $\mathcal{R}$ is a bisimulation, and $(plain^s(P), \nu s.P) \in \mathcal{R}$ (using P, s as in the statement of the lemma), we have $plain^s(P) \approx \nu s.P$. □

3.2 Unpredictability of nonces

Lemma 3.8 (Unpredictability of nonces) *Let C be a context not binding the variable x and let P, Q be processes. Then $\nu r.C[\mathit{if}\ x = r\ \mathit{then}\ P\ \mathit{else}\ Q] \approx\!\!\!\approx \nu r.C[Q]$.*

Proof. In the following, a *multi-hole context* is a context C with zero, one, or more holes. $C[P]$ means C with every occurrence of the hole replaced by the *same* process P.

We define the following relation $\mathcal{R}$:

$$\mathcal{R} := \Big\{ (\nu r.C[\text{if } T = r \text{ then } P \text{ else } Q],\ \nu r.C[Q]) \Big\}$$

up to structural equivalence. Here C ranges over multi-hole contexts, T over terms, $r \notin fv(T)$ over names, and P, Q over processes.

We show that $\mathcal{R}$ is a bisimulation:

- If $(A, B) \in \mathcal{R}$ and $A \downarrow_M$, then $B \to^* \downarrow_M$:

  Immediate since "if $T = r$ then P else Q" does not have unprotected outputs.

- If $(A, B) \in \mathcal{R}$ and $B \downarrow_M$, then $A \to^* \downarrow_M$:

  If the output on M is in C, $A \downarrow_M$. Otherwise the output is in an unprotected instance of Q in $\nu r.C[Q] \equiv B$. Since $r \notin fn(T)$, we have $T \neq_{\mathrm{E}} r$ by Lemma 3.1 (i) and hence (if $T = r$ then P else Q) $\to Q$. Then $A \to A'$ where A' results from replacing one instance of "if $T = r$ then P else Q" by Q. Then $A' \downarrow_M$.

- If $(A, B) \in \mathcal{R}$ and $A \to A'$ then there is a B' with $B \to^* B'$ and $(A', B') \in \mathcal{R}$:

  Then $A \equiv \nu r.C[\text{if } T = r \text{ then } P \text{ else } Q]$ and $B \equiv \nu r.C[Q]$. If the reduction $A \to A'$ takes place in C, then there is a corresponding reduction $B \to B'$ and $(A', B') \in \mathcal{R}$.

  Thus we can assume that one of the "if $T = r$ then P else Q" is being reduced in A. Since $T \neq_{\mathrm{E}} r$ by Lemma 3.1 (i), that subprocess reduces to Q. Thus $A' \equiv \nu r.C'[\text{if } T = r \text{ then } P \text{ else } Q]$ where C' is C with one of the holes replaced by Q. Then $B' := B \equiv \nu r.C[Q] = \nu r.C'[Q]$. Hence $B \to^* B'$ and $(A', B') \in \mathcal{R}$.

- If $(A, B) \in \mathcal{R}$ and $B \to B'$ then there is an A' with $A \to^* A'$ and $(A', B') \in \mathcal{R}$:

  Then $A \equiv \nu r.C[\text{if } T = r \text{ then } P \text{ else } Q]$ and $B \equiv \nu r.C[Q]$. As before, we have (if $T = r$ then P else Q) $\to Q$. The reduction $B \to B'$ may involve C and up to two instances of Q. We can thus write B as $B \equiv C''[Q]$ where C'' results from replacing in C the holes corresponding to these instances of Q. These instances of Q are not protected, so the holes we have replaced by Q are not protected, either. Thus $A \to^* C''[\text{if } T = r \text{ then } P \text{ else } Q] =: A''$. Then the reduction $B \equiv C''[Q] \to B'$ involves only C''. Hence $B' \equiv C'[Q]$ for some C', and $A'' \to C'[\text{if } T = r \text{ then } P \text{ else } Q] =: A'$. Thus $A \to^* A'$ and $(A', B') \in \mathcal{R}$.

- If $(A, B) \in \mathcal{R}$ and E is an evaluation context, then $(E[A], E[B]) \in \mathcal{R}$:

  Then $A \equiv \nu r.C[\text{if } T = r \text{ then } P \text{ else } Q]$ and $B \equiv \nu r.C[Q]$. Without loss of generality, $r \notin fn(E), bn(E)$. Hence $E[A] \equiv \nu r.E[C[\text{if } T = r \text{ then } P \text{ else } Q]]$ and $E[B] \equiv \nu r.E[C[Q]]$. Hence $(E[A], E[B]) \in \mathcal{R}$ (with $E[C]$ instead of C).

We can now show the lemma. Let C, P, Q, r be as in the lemma. Let σ be a substitution closing $\nu r.C[\text{if } x = r \text{ then } P \text{ else } Q]$ and $\nu r.C[Q]$. Without loss of generality, $r \notin fn(\sigma)$ (otherwise we rename r and change C, P, Q accordingly). In particular, $\sigma(x)$ will be some closed term T with $r \notin fn(T)$. Then $C[\text{if } x = r \text{ then } P \text{ else } Q]\sigma = C'[\text{if } T = r \text{ then } P' \text{ else } Q']$ and $C[Q]\sigma = C'[Q']$ where C', P', Q' are the result of applying σ to C, P, Q. (In the case of P, Q, restricted to those variables not bound by C.) And $(C'[\text{if } T = r \text{ then } P' \text{ else } Q'], C'[Q']) \in \mathcal{R}$. Thus $C'[\text{if } T = r \text{ then } P' \text{ else } Q'] \approx C'[Q']$. Since this holds for any closing σ, we have $C[\text{if } x = r \text{ then } P \text{ else } Q] \approxeq C[Q]$. □

4 Symbolic UC

Intuition. We start by presenting the intuition that underlies the original UC framework [Can01] and thus also our work. The basic idea is to define security of a protocol π by comparing it to a so-called ideal functionality $\mathcal{F}$. The ideal functionality is a machine that by definition does what the protocol should achieve. For example, if the task of the protocol is to transmit a message m securely from Alice to Bob, then the functionality is a trusted machine that expects a message m from Alice over a secure channel, sends to the adversary that such a message was received (but does not send the message itself), and then after the adversary allows delivery, forwards the message to Bob. (In the applied pi calculus, this functionality would be $net_{scstart}().io_A(x).(\overline{net_{notify}}\langle\rangle \mid net_{deliver}().\overline{io_B}\langle x\rangle)$ where the $net_{...}$-channels belong to the adversary; see Definition 6.2 below.) In a sense, the functionality is an abstract specification of the protocol behavior, and the protocol is supposed to be a concrete instantiation of that specification using crypto, in a way that preserves the security properties of the specification.

So how to model that a protocol π is as secure as a functionality $\mathcal{F}$? The basic idea is to ensure that any attack on π is also possible on $\mathcal{F}$. Since by assumption $\mathcal{F}$ does not allow any attacks, this implies that π does not allow any attacks either, so π is secure. To model that any attack on π is possible on $\mathcal{F}$, we require that for any adversary attacking π, there is a corresponding adversary (the "simulator") attacking $\mathcal{F}$ that performs an equivalent attack. And what do we mean by equivalent? Any "environment" that can observe the overall protocol outcome (inputs and outputs), and that can talk to the adversary (i.e., it learns what secret information the adversary might have obtained), cannot distinguish between the two attacks. In other words, for any adversary A, there is a simulator S such that for all environments Z, we have that $\pi + A + Z$ (the protocol running with A and Z) and $\mathcal{F} + S + Z$ are indistinguishable from Z's point of view. Notice that we do not wish to allow Z to observe the internal protocol communication – doing so would require that π and $\mathcal{F}$ work the same way internally, but we only want that the two have the same "observable effects", we do not care about their inner workings. Due to this, in a formal definition, we need to distinguish between the protocol-internal communication channels (net-channels), and the protocol's interface (io-channels). Only the latter is accessible to the environment.

Formal definition. To formalize the above intuition in the applied pi calculus, we first formalize the distinction between channels that make up the protocol's input/output interface, and those that make up the protocol's internal channels. We partition the set of all names into two sets IO and NET (both infinite). We will then require adversaries and simulators to only communicate on NET channels. (We do not forbid the environment to access NET channels. But we will give the adversary/simulator the ability to rename and hide NET channels, and thus effectively protect the protocol's NET channels from the environment.)

In order to keep the distinction between NET-channels and IO-channels, we also want to avoid that NET-channels are transmitted to the environment (we use this in a few

places in our proofs):

Definition 4.1 *We call a process P* NET-stable *if every name $n \in \text{NET} \cap fn(P)$ in P occurs only in channel identifiers (i.e., in particular, P does not send n to the environment).*

Note that there is no restrictions on the bound names. Thus a NET-stable adversary is free to share arbitrary fresh names with the environment and to use them as channels.

We now define the concept of an adversary. Essentially, an adversary is just a process A that is intended to interact with the protocol (or functionality). Since the adversary connects to the protocol over some NET-names, the specification of the adversary additionally includes a list of NET-names $\underline{n}$ of the protocol that will be accessed by A (and are thus private between A and the protocol). Finally, an adversary/simulator sometimes needs to rename NET-channels of the protocol/functionality to avoid name clashes. Since NET-channels are protocol internal and not part of the externally visible interface, it should not matter whether the same name is used in protocol and functionality or not. This is achieved by letting the adversary rename NET-names, we model this by specifying a renaming φ as part of the adversary.

Definition 4.2 *An* adversary *is a triple $(A, \varphi, \underline{n})$ where A is a closed* NET*-stable process with $\text{IO} \cap fn(A) = \emptyset$, $\varphi : \text{NET} \to \text{NET}$ a bijection and $\underline{n}$ a list of names $\underline{n} \subseteq \text{NET}$.*

We can now state our security definition. Both protocol and functionality are modeled by processes P and Q, respectively. An adversary $(A, \varphi_A, \underline{n}_A)$ connecting to P is modeled as $\nu\underline{n}_A.(P\varphi_A|A)$, as we would expect from the meaning of φ and $\underline{n}$ explained above. To model that P emulates Q, we would require that $\nu\underline{n}_A.(P\varphi_A|A)$ and $\nu\underline{n}_S.(Q\varphi_S|S)$ are indistinguishable for any environment for a suitable simulator $(S, \varphi_S, \underline{n}_S)$. We do not need to specify the environment explicitly because we have the notion of observational equivalence: $\nu\underline{n}_A.(P\varphi_A|A) \approx \nu\underline{n}_S.(Q\varphi_S|S)$ means that no context can distinguish the left and right hand side. The following definition captures this, except that we make one simplification: Instead of quantifying over all adversaries $(A, \varphi_A, \underline{n}_A)$, we fix $A := 0$, φ_A the identity, and $\underline{n}_A$ the empty list, so that $\nu\underline{n}_A.(P\varphi_A|A) = P$. (Such an adversary, that essentially just leaves all NET-channels accessible to the environment, is usually called a *dummy adversary*.) This definition is often technically much simpler to handle, and Lemma 4.4 below guarantees that it is equivalent to the more natural definition that quantifies over all adversaries.

Definition 4.3 *Let P and Q be processes. We say P* emulates *Q (written $P \leq Q$) iff there exists an adversary $(S, \varphi, \underline{n})$ such that $P \lessapprox \nu\underline{n}.(Q\varphi|S)$. $(S, \varphi, \underline{n})$ will often be called* simulator.

We use $\lessapprox$ instead of $\approx$ to get a more general definition, allowing non-closed P, Q. For the applications presented in this paper, the special case using $\approx$ (which is equivalent to our definition restricted to closed processes) is sufficient. (Note however that we would still use $\lessapprox$ to state various technical lemmas more conveniently.)

Note that there is no formal distinction between protocols and functionalities. Indeed, it can sometimes be convenient to compare two protocols P, Q. Furthermore, note that $\leq$ is weaker than $\approx$: $P \approx Q$ entails $P \leq Q$ (and $Q \leq P$) with the simulator $(\mathbf{0}, id, \emptyset)$.

As observed in [KDMR08] there are several approaches to define simulation based security. The following Lemma shows that our definition (resembling *strong simulatability*) is equivalent to the two alternatives: *black-box simulatability* and *universally-composable simulatability* (the latter being the definition that corresponds directly to the intuition given at the beginning of this section).

Lemma 4.4 *For processes P, Q we have that the following are equivalent:*

(i) strong simulatability: $P \leq Q$

(ii) black-box simulatability: $\exists(S, \varphi_S, \underline{n}_S) \quad \forall(A, \varphi_A, \underline{n}_A) \quad \nu\underline{n}_A.(P\varphi_A|A) \approx \nu\underline{n}_A.((\nu\underline{n}_S.(Q\varphi_S|S))\varphi_A|A)$

(iii) universally-composable simulatability: $\forall(A, \varphi_A, \underline{n}_A)\ \exists(S, \varphi_S, \underline{n}_S)\ \nu\underline{n}_A.(P\varphi_A|A) \approx \nu\underline{n}_S.(Q\varphi_S|S)$

where all triples are adversaries according to Definition 4.2.

Proof.

- (i) $\Rightarrow$ (ii):

$$\begin{aligned} P \leq Q \Rightarrow \exists(S, \varphi_S, \underline{n}_S) \quad & P \approx \nu\underline{n}_S.(Q\varphi_S|S) \\ \overset{(*)}{\Rightarrow} \forall \text{ bijections } \varphi_A \quad & P\varphi_A \approx (\nu\underline{n}_S.(Q\varphi_S|S))\varphi_A \\ \overset{(**)}{\Rightarrow} \forall(A, \varphi_A, \underline{n}_A) \quad & \nu\underline{n}_A.(P\varphi_A|A) \approx \nu\underline{n}_A.((\nu\underline{n}_S.(Q\varphi_S|S))\varphi_A|A) \end{aligned}$$

  $(*)$ since $\approx$ is closed under renaming and $(**)$ since $\approx$ is closed under the application of evaluation contexts.

- (ii) $\Rightarrow$ (iii): Let $(S, \varphi_S, \underline{n}_S)$ be the simulator from (ii), $(A, \varphi_A, \underline{n}_A)$ be an adversary and φ a bijection on names such that $\underline{n}_S(\varphi \circ \varphi_A) \cap fn(A) = \varnothing$ and φ is the identity on the free names of $Q(\varphi_A \circ \varphi_S)$ and $S\varphi_A$ (this φ can be used as α-conversion in step three below). We observe

$$\begin{aligned} & \nu\underline{n}_A.((\nu\underline{n}_S.(Q\varphi_S|S))\varphi_A|A) \\ \equiv\ & \nu\underline{n}_A.(\nu\underline{n}_S\varphi_A.(Q(\varphi_A \circ \varphi_S)|S\varphi_A)|A) \\ \equiv\ & \nu\underline{n}_A.(\nu\underline{n}_S(\varphi \circ \varphi_A).(Q(\varphi \circ \varphi_A \circ \varphi_S)|S(\varphi \circ \varphi_A))|A) \\ \equiv\ & \nu\underline{n}_A.\nu\underline{n}_S(\varphi \circ \varphi_A).(Q(\varphi \circ \varphi_A \circ \varphi_S)|S(\varphi \circ \varphi_A)|A) \end{aligned}$$

  and thus $(S_A, \underline{n}_{S_A}, \varphi_{S_A}) := (S(\varphi \circ \varphi_A)|A, \underline{n}_A \cup \underline{n}_S(\varphi \circ \varphi_A), (\varphi \circ \varphi_A \circ \varphi_S))$ is an adversary such that

$$\nu\underline{n}_A.(P\varphi_A|A) \approx \nu\underline{n}_{S_A}.(Q\varphi_{S_A}|S_A)$$

- (iii) $\Rightarrow$ (i) We construct the simulator from the last step for the adversary $(\mathbf{0}, \varnothing, id)$ and have (i).

□

Lemma 4.5 (Reflexivity, transitivity) *Let P, Q, R be processes. Then $P \leq P$. And if $P \leq Q$ and $Q \leq R$, then $P \leq R$.*

Proof. $P \leq P$ follows directly from Definition 4.3 by setting $S := \mathbf{0}$, φ as the identity, and $\underline{n} := \varnothing$.

Assume now that $P \leq Q$ and $Q \leq R$. Then there are processes S_1, S_2 with $\mathrm{IO} \cap fn(S_1) = \mathrm{IO} \cap fn(S_2) = \varnothing$, bijections $\varphi_1, \varphi_2 : \mathrm{NET} \to \mathrm{NET}$, and lists of names $\underline{n}_1, \underline{n}_2 \subseteq \mathrm{NET}$ such that $P \approx \nu \underline{n}_1.(Q\varphi_1|S_1)$ and $Q \approx \nu \underline{n}_2.(R\varphi_2|S_2)$. Without loss of generality we can choose $\underline{n}_2$ such that $\underline{n}_2\varphi_1 \cap fn(S_1) = \varnothing$. It follows

$$\begin{aligned} P &\approx \nu \underline{n}_1.(Q\varphi_1|S_1) \\ &\overset{(*)}{\approx} \nu \underline{n}_1.((\nu \underline{n}_2.(R\varphi_2|S_2))\varphi_1|S_1)] \\ &\overset{(**)}{\equiv} \nu \underline{n}_1.((\nu \underline{n}_2\varphi_1.(R(\varphi_1 \circ \varphi_2)|S_2\varphi_1))|S_1) \\ &\overset{(***)}{\equiv} \nu \underline{n}_1.\nu \underline{n}_2\varphi_1.(R(\varphi_1 \circ \varphi_2)|S_2\varphi_1|S_1) \end{aligned}$$

Here $(*)$ follows since $\approx$ is closed under the application of evaluation contexts and under renaming of free names.

And $(**)$ follows since for any process R, we have $(\nu \underline{n}_2.R)\varphi_1 \equiv \nu \underline{n}_2\varphi_1.(R\varphi_1)$.

And $(***)$ follows since $\underline{n}_2\varphi_1 \cap fn(S_1) = \varnothing$.

Thus, choosing $\underline{n} := \underline{n}_1 \cup \underline{n}_2\varphi_1$, $\varphi := \varphi_1 \circ \varphi_2$, and $S := S_2\varphi_1|S_1$, we get $P \approx \nu \underline{n}.(R\varphi|S)$. Hence $P \leq R$. □

Corruption. So far, we have not yet modeled the ability of the adversary to corrupt parties. There are two main variants of corruption: static and adaptive corruption. In the case of static corruption, it is determined in the beginning of the protocol who is corrupted. For adaptive corruption, corruption may occur during the protocol and depend on protocol messages. Modeling static corruption is quite easy in our model: When a party X is corrupted, we simply remove the subprocess P_X corresponding to that party from the protocol P, make all NET-names occurring in P_X public, and – in the case of a functionality – additionally rename all IO-names of P_X into NET-names. For example, if $P = \nu net_1 net_2.(P_A|P_B|\mathcal{F})$ where net_1 occurs in P_A and P_B and net_2 only in P_B, and $\mathcal{F}$ has IO-names io_{FA}, io_{FB} then corrupting A leads to $P' = \nu net_2.(P_B|\mathcal{F}\{net_{FA}/io_{FA}\})$. And a functionality $\mathcal{G}$ with IO-names io_A, io_B becomes $\mathcal{G}\{net_A/io_A\}$.

So, if we want to verify that a P emulates $\mathcal{G}$ for any corruption, we need to check:

- Uncorrupted: $P \leq \mathcal{G}$.
- Alice corrupted: $\nu net_2.(P_B|\mathcal{F}\{net_{FA}/io_{FA}\}) \leq \mathcal{G}\{net_A/io_A\}$.
- Bob corrupted: $P_A|\mathcal{F}\{net_{FB}/io_{FB}\} \leq \mathcal{G}\{net_B/io_B\}$.

An example is given in Section 9.1 in the case of UC secure commitments.

Modeling adaptive corruptions is more complex. For this one would need to introduce special parties that react to a special signal from the environment and then switch into a corrupted mode. We do not follow that approach here.

5 Composition

One of the salient properties of the UC framework is composition. Assume a protocol π UC-emulates a functionality $\mathcal{F}$. And ρ is a protocol using $\mathcal{F}$. Then $\rho^{\pi/\mathcal{F}}$ (which is ρ with $\mathcal{F}$ replaced by π) UC-emulates ρ. And hence, by transitivity, if ρ emulates some functionality $\mathcal{G}$, $\rho^{\pi/\mathcal{F}}$ UC-emulates $\mathcal{G}$.

In our context, ideally we would like a composition theorem such as $P \leq Q \Longrightarrow C[P] \leq C[Q]$ for arbitrary contexts C. Unfortunately, the situation is not as simple. A simple observation is that if C may contain NET-names, then composition will not work: For example, assume $P \leq Q$, and P is a protocol using some NET-channel *net* to implement an ideal functionality Q (which does not use *net*). And $C = \Box | R$ receives on a NET-channel *net* and outputs the received messages on an IO-channel *io*. Then $C[P]$ will output protocol-internal messages on *io* (observable to the environment), while $C[Q]$ will not (since the functionality Q will not use the channel *net*). Hence $C[P] \not\leq C[Q]$. (We give a formal analysis of the various cases in which the composition theorem does not hold in Appendix A.)

Thus a first condition on C is that it may not use the same NET-names. In fact, we show below (Theorem 5.37) that if C is an evaluation context binding only IO-names and not using any of the NET-names of P, Q, then $P \leq Q \Longrightarrow C[P] \leq C[Q]$ holds.

This already allows for a large range of composition operations. (In particular, we can connect different protocols through their interfaces securely by composing them in parallel, and restricting the IO-channels through which they are connected.) But one important operation is missing, namely concurrent composition. Concurrent composition means that if $P \leq Q$, then $P' \leq Q'$ where P' consists of many instances of P and Q' analogously. Such a result is important in many cases, e.g., if P is a single session key-exchange, but an embedding protocol needs a large number of keys. The most obvious way to model this in our setting would be a theorem stating $P \leq Q \Longrightarrow !P \leq !Q$.

Unfortunately, such a theorem cannot hold, either. The intuitive reason is as follows: When trying to construct a simulator for $!Q$, then this simulator will not be able to distinguish messages from different instances of Q. The simulator will then be unable to even decide whether he talks to a single instance or several. For example:

$$\begin{aligned} P := \nu n\, m.\big(\ & \overline{io_1}\langle n\rangle \mid io_2(x).\text{if } x = n \text{ then } \overline{net_2}\langle m\rangle \\ & \mid io_3(x).\text{if } x = n \text{ then } \overline{net_3}\langle m\rangle\big) \\ Q := \nu n.\big(\ & \overline{io_1}\langle n\rangle \mid io_2(x).\text{if } x = n \text{ then } \overline{net_2}\langle \mathit{empty}\rangle \\ & \mid io_3(x).\text{if } x = n \text{ then } \overline{net_3}\langle \mathit{empty}\rangle\big) \end{aligned}$$

Here we have $P \leq Q$ because a simulator receiving *empty* on net_2 or net_3 just has to replace it by some fresh name m. However, we do not have $!P \leq !Q$. Depending on

the messages the environment sends on io_2, $!P$ will output either the same name m on net_2, net_3, or different names m, m'. However, a simulator interacting with $!Q$ in both cases gets $empty, empty$ on net_2, net_3. The simulator then does not know whether he should change this into m, m or m, m' for fresh m, m'. Thus the simulator fails. (The formal argument is in Appendix A.)

So we cannot have a theorem stating $P \leq Q \Longrightarrow !P \leq !Q$. Does this mean concurrent composition is not possible? No, just that ! is not the right operator to model it. In the computational UC framework, composition also does not involve a number of indistinguishable instances. Instead, each instance of P and Q is given a unique session id, and all communication is tagged with that session id so that it can be routed to the right instance. In our setting, one possibility to achieve this is to define an operator !! [Che66] such that $!!P$ behaves like an unlimited number of instances of P, where each instance is tagged with a unique session id *sid*. I.e., each channel C in P is replaced by (sid, C).[10]

The question is how to define $!!P$. The applied pi calculus does not have any construct that conveniently allows to perform infinite branching with different ids. Thus, we have to work around this restriction by introducing a more elaborate construction. As a first step, we define the tagged version $P(\!(M)\!)$ of the process P:

Definition 5.1 *Let P be a process, and let M be a term. We write $P(\!(M)\!)$ for P with every occurrence of $C(x)$ replaced by $(M, C)(x)$ and every occurrence of $\overline{C}\langle T\rangle$ replaced by $\overline{(M, C)}\langle T\rangle$. (If M contains bound variables or bound names from P, we assume that these bound variables/names are first renamed in P.)*

Now we have to somehow define $!!P$ as $P(\!(s_1)\!)|P(\!(s_2)\!)|\ldots$ where $s_1, s_2, \ldots$ range over some infinite set *SID* of session ids. Using product processes (see Section 2.2) this is easy: $!!P := \prod_{x \in SID} P(\!(x)\!)$ does the job. However, product processes are a nonstandard extension of the applied pi calculus, but we wish to stay compatible with existing variants (in particular, to be able to use Proverif for verification). Thus, instead of using $\prod_{x \in SID} P(\!(x)\!)$, we define a suitable context $\mathcal{C}$ such that $\mathcal{C}[P(\!(x)\!)]$ behaves like $\prod_{x \in SID} P(\!(x)\!)$. Then we can define $!!P := \mathcal{C}[P(\!(x)\!)]$. Of course, depending on the particular set *SID* we choose, a different context $\mathcal{C}$ will be needed. Instead of fixing a particular one, we thus give a general definition what contexts are suitable for a given set *SID*, and from then on, just assume an arbitrary such context.

Definition 5.2 (Indexing context) *Given a set S of terms, a variable x (will be used for tagging), and names $\underline{n}$, we call a closed context $\mathcal{C}_{x,\underline{n}}$ with $bn(\mathcal{C}_{x,\underline{n}}) = \underline{n}$ and $fn(\mathcal{C}_{x,\underline{n}}) = \emptyset$ (not containing indexed replications) an S-*indexing context *iff for all processes P with*

[10] One might instead consider tagging the messages sent over the channel with *sid*. This, however, does not work as well: One would need a specific multiplexer process that given a message (sid, T) discovers the corresponding instance of P and delivers to it. This might be possible, but is probably considerably more complicated than the approach we take below.

$x \notin bv(P)$ [11] *and* $\underline{n} \cap fn(P) = \emptyset$ *we have*

$$\mathcal{C}_{x,\underline{n}}[P(\!(x)\!)] \approxeq \prod_{x \in S} P(\!(x)\!)$$

In the following, we fix a set *SID* of terms containing no names or variables. The set *SID* will represent the set of all session IDs. We assume that $id =_{\mathrm{E}} id'$ entails $id = id'$ for $id, id' \in SID$ (different IDs are never equivalent by the equational theory).

Note that not for every set *SID* a *SID*-indexing context exists. For example, if *SID* is not semi-decidable (but the equational theory is), then there is no *SID*-indexing context. One might be concerned that our definition of *SID*-indexing contexts cannot be fulfilled. The following definition shows that this is not the case, at least if we use suitably encoded bitstrings as SIDs.

Definition 5.3 *Assume that a nullary constructor nil and unary constructors zero and one are part of our symbolic model. Let* SID_{bits} *be the set of all terms built from nil, zero and one. Assume furthermore that for* $id, id' \in SID_{bits}$ *in our symbolic model* $id =_{\mathrm{E}} id'$ *entails* $id = id'$*. Let*

$$\mathcal{C}_{x,a}^{SID_{bits}} := \nu a.(\overline{a}\langle nil\rangle | !a(x).(\overline{a}\langle zero(x)\rangle | \overline{a}\langle one(x)\rangle | \Box))$$

Intuitively, $\mathcal{C}_{x,a}^{SID_{bits}}$ is a factory with parameters x and a for tagged instances of P that realizes the abstract construction of $\prod_{x \in SID_{bits}} P(\!(x)\!)$. We now show that $\mathcal{C}_{x,a}^{SID_{bits}}$ actually is an SID_{bits}-indexing context. Towards this goal we first define an intermediate representation of $\mathcal{C}_{x,a}^{SID_{bits}}$.

Definition 5.4 *Let* P *be a process. We write* P^n *for* n *parallel instances of* P $(P|\ldots|P)$*. We define the following functions on the set of processes:*

$$\begin{aligned} G_{x,a}(P) :=& a(x).(\overline{a}\langle zero(x)\rangle \mid \overline{a}\langle one(x)\rangle \mid P) \\ \mathcal{G}_{x,a}^{n}(P) :=& (G_{x,a}(P))^n \mid !G_{x,a}(P) \\ \mathcal{C}_{x,a}^{(sID,gID,n)}(P) :=& \Sigma_{x \in sID} P \mid \nu a.(\Sigma_{x \in gID} \overline{a}\langle x\rangle \mid \mathcal{G}_{x,a}^{n}(P)) \end{aligned}$$

where $\Sigma_{x \in \mathcal{T}} P$ *for a finite set of terms* $\mathcal{T} = \{T_1, \ldots, T_l\}$ *is syntactic sugar for* $P\{T_1/x\}|\ldots|P\{T_l/x\}$ *(this is only well-defined up to structural equivalence),* $sID \subseteq SID$*,* $gID \subseteq SID$ *and* $n \in \mathbb{N}$*.*

Intuitively, *sID* (spawned IDs) contains the ids for all instances of P, that have already been tagged but are still formally a part of $\mathcal{C}_{x,a}^{SID_{bits}}$ (i.e., "are still in the factory"). *gID* is the foundation for the ids yet to be generated. These ids are the elements of the *span* of *gID* which we will introduce in the following definition. The last parameter n exists mainly for technical reasons and counts the number of currently active generator instances $G^{x,a}(P)$.

[11] P may have $x \in fv(P)$ but we forbid $x \in bv(P)$ to avoid technicalities in the definition of $P(\!(x)\!)$ due to the shadowed x.

Definition 5.5 (Span) *Let $S \subseteq SID_{bits}$ be a set of IDs. We call $\langle S\rangle := S \cup \{c_n(\dots c_2(c_1(s))\dots) : s \in S, c_i \in \{zero, one\}\}$ the span of S (note that $\langle S\rangle \subseteq SID_{bits}$).*

The following definition bridges the gap between $\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!))$ and $\prod_{x\in S} P(\!(x)\!)$. Have in mind that S denotes the set of ids that are yet to be used by the product process for tagging and we have $S = SID_{bits}$ at the beginning.

Definition 5.6 (S-valid) *Let $sID \subseteq SID_{bits}$, $gID \subseteq SID_{bits}$ and $S \subseteq SID_{bits}$ be sets of ids and sID and gID be finite. We call $\mathcal{C}_{x,a}^{(sID,gID,n)}$* S-valid *if $sID = \varnothing$ and $gID = \{nil\}$ or if*

(i) $sID \subseteq S$

(ii) $gID = \{f(x) : x \in G, f(x) \notin G, f \in \{zero, one\}\}$ *where $G := (SID_{bits} \setminus S) \cup sID$ (intuitively, G is the set of ids already generated)*

(iii) $\langle gID\rangle = S \setminus sID$

Lemma 5.7 *Let $S \subseteq SID_{bits}$ and $\mathcal{C}_{x,a}^{(sID,gID,n)}$ be S-valid where $n \geq 1$. Then for any $id \in gID$ we have that $\mathcal{C}_{x,a}^{(sID',gID',n-1)}$ is S-valid where $sID' := sID \cup \{id\}$ and $gID' := gID \setminus \{id\} \cup \{zero(id), one(id)\}$.*

Proof. Due to Definition 5.6 point iii we have that $gID \cap sID = \varnothing$ and $gID \subseteq S$. We check the three points of Definition 5.6 for sID' and gID':

(i) $id \in gID \subseteq S$ and $sID \subseteq S$ entail $sID' = (sID \cup \{id\}) \subseteq S$

(ii) For a set $G \subseteq SID_{bits}$ we define $M(G) := \{f(x) : x \in G, f(x) \notin G, f \in \{zero, one\}\}$. By assumption we have $gID = \{nil\}$ or $gID = M(G)$ for $G := (SID_{bits} \setminus S) \cup sID$. The first case leads to $sID' = \{nil\}$ and $gID' = \{zero(nil), one(nil)\}$ for which this point can easily be verified. For the second case we define $G' := G \cup \{id\}$. $id \notin M(G')$ since $id \in G'$. $f(id) \in M(G')$ for $f \in \{zero, one\}$ iff $f(id) \notin G'$. We assume towards contradiction that $f(id) \in G'$. Then $f(id) \in G$ and by definition of G $f(id) \in (SID_{bits} \setminus S) \cup sID$. However

- $f(id) \in (SID_{bits} \setminus S)$ entails $f(id) \notin S$ and thus $f(id) \notin \langle gID\rangle$. This contradicts $f(id) \in \langle gID\rangle$ (which holds since $id \in gID$).
- $f(id) \in sID$ entails $f(id) \notin \langle gID\rangle$ and leads to a contradiction analogously.

All together we have $f(id) \notin G'$ and hence $M(G') = M(G) \setminus \{id\} \cup \{zero(id), one(id)\} = gID'$.

(iii) $\langle gID'\rangle = \langle gID\setminus\{id\}\cup\{zero(id), one(id)\}\rangle = \langle gID\rangle\setminus\{id\} = S\setminus sID\setminus\{id\} = S\setminus gID'$.

□

To show that $\mathcal{C}_{x,a}^{SID_{bits}}$ is a SID_{bits}-indexing context (see Lemma 5.10) we first show $\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!)) \approxeq \nu a.\prod_{x\in S} P(\!(x)\!)$ for every S-valid $\mathcal{C}_{x,a}^{(sID,gID,n)}$.

Lemma 5.8 *Let P be a process and M be a term. If $\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!)) \updownarrow_M$ there is exactly one $id \in sID$ such that $P(\!(id)\!) \updownarrow_M$.*

Proof. It is easy to see that $\mathcal{C}_{x,a}^{(sID,gID,n)}(\mathbf{0})$ never communicates on a channel (note that a is bound). Hence for $\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!)) \updownarrow_M$ we need one of the tagged instances of P in $\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!))$ to communicate on M, i.e., $P(\!(id)\!) \updownarrow_M$ for some $id \in sID$ requiring $M =_{\mathrm{E}} (id, \square)$. Analogously, for any $id' \in sID$ with $P(\!(id')\!) \updownarrow_M$ we have $M =_{\mathrm{E}} (id', \square)$. Due to Definition 2.5 (vi) (natural symbolic model) this entails $id =_{\mathrm{E}} id'$ which leads to $id = id'$ by definition of SID_{bits} ($sID \subseteq SID_{bits}$). Thus, the ID id with $P(\!(id)\!) \updownarrow_M$ is unique. □

Lemma 5.9 *Let P be a process with at most one free variable, which we call x if existent, and $x \notin bv(P)$. Let $a \notin fn(P)$ be a name. Then*

$$\mathcal{C}_{x,a}^{(\emptyset,\{nil\},0)}(P(\!(x)\!)) \approx \prod_{x \in SID_{bits}} P(\!(x)\!)$$

Proof. We define the relation

$$\mathcal{R} := \approx \cup \{(\mathcal{E}[\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!))], \mathcal{E}[\prod_{x \in S} P(\!(x)\!)]) : \text{ for any } n \geq 0,\ S \subseteq SID_{bits},$$
$$\text{evaluation context } \mathcal{E}, \text{ process } P \text{ and } \mathcal{C}_{x,a}^{(sID,gID,n)}\ S\text{-valid}\}$$

closed under structural equivalence. Then we show that $\mathcal{R} \subseteq \approx$. Towards this goal we show that $\mathcal{R}$ and $\mathcal{R}^{-1}$ are simulations. We start with $\mathcal{R}$:

- $\mathcal{E}[\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!))] \downarrow_M$: If $\mathcal{E}[\mathbf{0}] \downarrow_M$ we obviously have $\mathcal{E}[\prod_{x \in S} P(\!(x)\!)] \downarrow_M$. Otherwise $\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!)) \downarrow_M$. In this case, according to Lemma 5.8, there is a distinct $id \in sID$ such that $P(\!(id)\!) \downarrow_M$ and, since $\mathcal{E}[\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!))] \downarrow_M$, $\mathcal{E}[P(\!(id)\!)] \downarrow_M$. On the other hand, due to the S-validity of $\mathcal{C}_{x,a}^{(sID,gID,n)}$, $sID \subseteq S$. With $id \in S$ we have $\prod_{x \in S} P(\!(x)\!) \rightarrow P(\!(id)\!) | \prod_{x \in S \setminus \{id\}} P(\!(x)\!)$ and hence $\mathcal{E}[\prod_{x \in S} P(\!(x)\!)] \rightarrow\downarrow_M$.

- $\mathcal{E}[\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!))] \rightarrow (\mathcal{E}[\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!))])'$: We distinguish three cases

  1. $\rightarrow$ does only affect $\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!))$ up to structural equivalence. In this case we have $\mathcal{E}[\mathbf{0}] \rightarrow \mathcal{E}'[\mathbf{0}]$, $\mathcal{E}[\prod_{x \in S} P(\!(x)\!)] \rightarrow \mathcal{E}'[\prod_{x \in S} P(\!(x)\!)]$ and $(\mathcal{E}'[\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!))], \mathcal{E}'[\prod_{x \in S} P(\!(x)\!)]) \in \mathcal{R}$.
  2. $\rightarrow$ is a COMM reduction that interferes with $\mathcal{E}$ and $\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!))$. Due to Lemma 5.8 we find a distinct $id \in sID$ such that

$$\mathcal{E}[\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!))] \rightarrow \mathcal{E}'[P(\!(id)\!)' | \mathcal{C}_{x,a}^{(sID \setminus \{id\},gID,n)}(P(\!(x)\!))]$$

Analogously to the case for $\mathcal{E}[\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!))] \downarrow_M$ we spawn a properly tagged instance of P from $\prod_{x\in S} P(\!(x)\!)$. With $\widetilde{\mathcal{E}}[\Box] := \mathcal{E}'[P(\!(id)\!)'|\Box]$ we have

$$(\widetilde{\mathcal{E}}[\mathcal{C}_{x,a}^{(sID\setminus\{id\},gID,n)}(P(\!(x)\!))], \widetilde{\mathcal{E}}[\prod_{x\in S\setminus\{id\}} P(\!(x)\!)]) \in \mathcal{R}$$

since $\mathcal{C}_{x,a}^{(sID\setminus\{id\},gID,n)}$ is $(S\setminus\{id\})$-valid.

3. $\rightarrow$ does only affect $\mathcal{E}$ up to structural equivalence. In this case we have $\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!)) \rightarrow \mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!))'$. We distinguish three cases:
   - $\rightarrow$ is a REPL reduction and spawns a new instance of $G_{x,a}$ (see Definition 5.4). In this case $\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!)) \rightarrow \mathcal{C}_{x,a}^{(sID,gID,n+1)}(P(\!(x)\!))$ and $(\mathcal{E}[\mathcal{C}_{x,a}^{(sID,gID,n+1)}(P(\!(x)\!))], \mathcal{E}[\prod_{x\in S} P(\!(x)\!)]) \in \mathcal{R}$.
   - $\rightarrow$ is a COMM reduction on channel a ($\overline{a}\langle id\rangle$) (note that this requires $n \geq 1$). In this case $id \in gID \subseteq S$ and $\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!)) \rightarrow \mathcal{C}_{x,a}^{(sID',gID',n-1)}(P(\!(x)\!))$ where $sID' := sID \cup \{id\}$ and $gID' := gID \setminus \{id\} \cup \{zero(id), one(id)\}$. By Lemma 5.7 we see that $\mathcal{C}_{x,a}^{(sID',gID',n-1)}(P(\!(x)\!))$ is still S-valid. Hence $(\mathcal{E}[\mathcal{C}_{x,a}^{(sID',gID',n-1)}(P(\!(x)\!))], \mathcal{E}[\prod_{x\in S} P(\!(x)\!)]) \in \mathcal{R}$.
   - $\rightarrow$ is a reduction of one of the P-instances $P(\!(id)\!)$ ($id \in sID$) (note that due to Lemma 5.8 and $a \notin fn(P)$ only one instance can be affected). In this case we proceed analogously to case 2.

- Obviously $\mathcal{R}$ is closed under the application of evaluation contexts.

We continue by showing the three points of observational equivalence for $\mathcal{R}^{-1}$:

- $\mathcal{E}[\prod_{x\in S} P(\!(x)\!)] \downarrow_M$ iff $\mathcal{E}[\mathbf{0}] \downarrow_M$. Therefore $\mathcal{E}[\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!))] \downarrow_M$.
- $\mathcal{E}[\prod_{x\in S} P(\!(x)\!)] \rightarrow \mathcal{E}[\prod_{x\in S} P(\!(x)\!)]'$: If we have $\mathcal{E}[\prod_{x\in S} P(\!(x)\!)] \rightarrow \mathcal{E}'[\prod_{x\in S} P(\!(x)\!)]$ we have $(\mathcal{E}'[\prod_{x\in S} P(\!(x)\!)], \mathcal{E}'[\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!))]) \in \mathcal{R}^{-1}$. Otherwise $\rightarrow$ is an IREPL reduction: $\prod_{x\in S} P(\!(x)\!) \rightarrow P(\!(id)\!)|\prod_{x\in S\setminus\{id\}} P(\!(x)\!)$ with $id \in S$. On the right side of the relation we have $\mathcal{E}[\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!))]$. Since $\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!))$ is S-valid, we have that $id \in sID$ or $id \in \langle gID\rangle$.

  If $id \notin sID$, i.e., $id \in \langle gID\rangle$, id is of the form $id = c_l(\dots c_1(id_0)\dots)$ for some $id_0 \in gID$, some $l \in \mathbb{N}$ and $c_i \in \{zero, one\}$ for $i \in \{1,\dots,l\}$. We write id_i for $c_i(\dots c_1(id_0)\dots)$ for $i \in \{1,\dots,l\}$, $\overline{c_i} := zero$ if $c_i = one$, $\overline{c_i} := one$ otherwise and $\overline{id_i}$ for $\overline{c_i}(c_{i-1}(\dots c_1(id_0)\dots))$. The reduction $\xrightarrow{\overline{a}\langle id_i\rangle}$ denotes a REPL reduction that spawns an instance of $G_{x,a}$ (see Definition 5.4) and a following COMM reduction on channel a with message $id_i \in gID$. The application of the sequence $\xrightarrow{\overline{a}\langle id_0\rangle}$ $\dots$ $\xrightarrow{\overline{a}\langle id_k\rangle}$ to $\mathcal{E}[\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!))]$ for some $0 \leq k \leq l$ yields a process that is structurally equivalent to $\mathcal{E}[\mathcal{C}_{x,a}^{(sID_k,gID_k,n)}(P(\!(x)\!))]$ with $sID_k := sID \cup \{id_0,\dots,id_k\}$ and $gID_k := gID \setminus \{id_0\} \cup \{\overline{id_1},\dots,\overline{id_{k-1}}\} \cup \{zero(id_k), one(id_k)\}$. For each step

$k \rightsquigarrow k+1$ the S-validity of $\mathcal{C}_{x,a}^{(sID_k,gID_k,n)}$ is guaranteed by Lemma 5.7. We define $sID' := sID_l$ and $gID' := gID_l$ and have that $id \in sID'$.

Otherwise, if $id \in sID$, we define $sID' := sID$ and $gID' := gID$.

With $id \in sID'$ and $\mathcal{E}'[\Box] := \mathcal{E}[P(\!(id)\!)|\Box]$ we have that

$$(\mathcal{E}'[\prod_{x \in S \setminus \{id\}} P(\!(x)\!)], \mathcal{E}'[\mathcal{C}_{x,a}^{(sID' \setminus \{id\},gID',n)}(P(\!(x)\!))]) \in \mathcal{R}^{-1}$$

since $\mathcal{C}_{x,a}^{(sID' \setminus \{id\},gID',n)}$ is $(S \setminus \{id\})$-valid.

- Obviously $\mathcal{R}^{-1}$ is closed under the application of evaluation contexts.

Since $\mathcal{C}_{x,a}^{(\emptyset,\{nil\},0)}$ is SID_{bits}-valid the Lemma holds. □

Lemma 5.10 *$\mathcal{C}_{x,a}^{SID_{bits}}$ is an SID_{bits}-indexing context.*

Proof. Let, according to Definition 5.2, P be a process and x be a variable with $x \notin bv(P)$. We pick a name a with $a \notin fn(P)$. We claim

$$\mathcal{C}_{x,a}^{SID_{bits}} \approxeq \prod_{x \in SID_{bits}} P(\!(x)\!)$$

We have to show $\mathcal{C}_{x,a}^{SID_{bits}}[P(\!(x)\!)]\sigma \approx (\prod_{x \in SID_{bits}} P(\!(x)\!))\sigma$ for all closing substitutions σ. W.l.o.g. $a \notin \sigma$ and $\sigma(x) = x$ and thus it suffices to show

$$\mathcal{C}_{x,a}^{SID_{bits}}[P(\!(x)\!)\sigma] \approx \prod_{x \in SID_{bits}} (P(\!(x)\!)\sigma) \tag{3}$$

Note that $P\sigma$ is a process with at most one free variable, denoted x. Furthermore $x \notin bv(P\sigma)$, $a \notin fn(P\sigma)$ and $\mathcal{C}_{x,a}^{SID_{bits}}[P(\!(x)\!)\sigma] = \mathcal{C}_{x,a}^{(\emptyset,\{nil\},0)}(P(\!(x)\!)\sigma)$ by Definition 5.4. By Lemma 5.9 we have (3) which concludes our proof. □

We stress that $\mathcal{C}_{x,a}^{SID_{bits}}$ is just one example of an indexing context. From now on SID is an arbitrary but fixed set of indexes and $\mathcal{C}_{x,\underline{n}}^{SID}$ an arbitrary but fixed SID-indexing context according to Definition 5.2. All our results then hold independently of the particular choice of SID.

We can now finally define $!!P$:

Definition 5.11 (Indexed replication) *Let P be a process. We define $!!_x P := \mathcal{C}_{x,\underline{n}}^{SID}[P(\!(x)\!)]$ for some arbitrary $\underline{n}$ with $\underline{n} \cap fn(P) = \emptyset$. We write $!!P$ for $!!_x P$ with $x \notin (fv(P) \cup bv(P))$.*

Notice that our definition is a bit more general, we can even write $!!_x P$, in this case P will have access to the sid via the variable x. We need this added flexibility in Section 8.3 for the protocol KE^*.

Note that since $\mathcal{C}_{x,\underline{n}}^{SID}[P(\!(x)\!)] \approxeq \prod_{x \in S} P(\!(x)\!)$ by definition, we can actually think of $!!_x P$ as being defined as $\prod_{x \in S} P(\!(x)\!)$. Our definition, however, has the advantage that $!!_x P$ is actually a process in the original calculus, the concept of product processes was only used as a tool for defining $!!$.

On real-life implementations of !!. When implementing a process $!!P$ in real life (i.e., in software for actual deployed protocols), a process such as $!!c(x).P'$ is probably best implemented by a process that listens on c for messages of the form (sid, m). Whenever such a message is received, a new instance of P' with session id sid is spawned, and all further messages with that sid are routed to that instance of P'. On the other hand, a process such as $!!\overline{c}\langle M\rangle.P'$ cannot be implemented, because such a process would non-deterministically send (sid, M) for all possible sid. A process $!!(A|B)$, where A and B correspond to processes run on different computers, does not immediately make sense, because if, e.g., A receives a message that spawns a new instance, B would have to spawn a new instance, too, without communication between A and B. Fortunately, we show in Lemma 5.36 below that $!!(A|B) \lessapprox !!A \mid !!B$; then A and B can spawn instances independently.

Properties of !!. The following four lemmas state several important properties of !!. We will need these to prove the composition theorem below. Lemmas 5.12, 5.13, and 5.36 also hold for ! instead of !!. But Lemma 5.35 is specific to !!, and is crucial for enabling the composition theorem.

Lemma 5.12 *Let P be a process and $\varphi : \mathcal{N} \to \mathcal{N}$ be a permutation on names. Then $(!!_x P)\varphi \equiv !!_x(P\varphi)$ for all variables $x \notin bv(P)$.*

Proof. Pick names $\underline{n}$ with $\underline{n} \cap fn(P) = \emptyset$ and $\varphi(\underline{n}) \cap fn(P) = \emptyset$. Note that $(!!_x P)\varphi \equiv \mathcal{C}^{SID}_{x,\underline{n}}[P(\!(x)\!)]\varphi$. Therefore $(!!_x P)\varphi \equiv \mathcal{C}^{SID}_{x,\underline{n}}[P(\!(x)\!)]\varphi = C^{SID}_{x,\varphi(\underline{n})}[P(\!(x)\!)\varphi] \equiv !!_x(P\varphi)$ since $\varphi(\underline{n}) \cap fn(P) = \emptyset$. □

Lemma 5.13 *Let P, Q be processes. Then $P \lessapprox Q \Rightarrow !!_x P \lessapprox !!_x Q$ for all variables $x \notin bv(P) \cup bv(Q)$.*

This lemma was surprisingly hard to prove. Before we proceed to the proof (for which we have to develop a number of auxiliary concepts and definitions first) We very roughly sketch the proof idea here: The main thing to show is that $P \approx Q \Longrightarrow P(\!(M)\!) \approx Q(\!(M)\!)$ for arbitrary fixed M. To show this, we define an operation *untag* that maps $P(\!(M)\!)$ to P, i.e., removes the tag M from all channels. Then we wish to prove that the following relation is a bisimulation: $\sim_{\mathcal{S}_{sid}} := \{(P, Q) : untag(P) \approx untag(Q)\}$. Once we have that, we see that $P(\!(M)\!) \sim_{\mathcal{S}_{sid}} Q(\!(M)\!)$ and hence $P(\!(M)\!) \approx Q(\!(M)\!)$. Unfortunately, $\sim_{\mathcal{S}_{sid}}$ is not really a bisimulation. A bisimulation must be closed under evaluation contexts, even under contexts in which not all channels are tagged with M. To solve this problem, we tweak *untag* in such a way that non-tagged channels C are mapped to specially marked channels (using a special name n_{sid})which can then be mapped back to C when tagging again. And we need to tweak the notion of a bisimulation slightly, so that $\sim_{\mathcal{S}_{sid}}$ only needs to be closed under evaluation contexts on which our operation *untag* works properly. These tweaks lead to an unexpectedly complex proof of Lemma 5.13.

Before we prove Lemma 5.13 (on page 56), we will need to develop a number of tools and lemmas.

Definition 5.14 *A set $\mathcal{S}$ of closed processes is* n-complete *for a name n iff for any closed process P with $n \notin fn(P) \cup bn(P)$, there is a closed process $S \in \mathcal{S}$ such that $P \approx S$.*

Definition 5.15 ($\mathcal{S}$-n-observational equivalence) *Let $\mathcal{S}$ be a set of closed processes and n be a name. An* $\mathcal{S}$-n-simulation *$\mathcal{R}$ is a relation on closed processes P, Q with $n \notin (fn(P) \cup fn(Q) \cup bn(P) \cup bn(Q))$ such that $(P, Q) \in \mathcal{R}$ implies*

(i) if $P \downarrow_M$ then $Q \rightarrow^* \downarrow_M$

(ii) if $P \rightarrow P'$ with $n \notin fn(P') \cup bn(P')$ then $Q \rightarrow^* Q'$ and $(P', Q') \in \mathcal{R}$ for some Q'

(iii) $(\nu \underline{s}.(S|P), \nu \underline{s}.(S|Q)) \in \mathcal{R}$ for all closed $S \in \mathcal{S}$ and names $\underline{s} \subseteq \mathcal{N}$ with $n \notin (fn(S) \cup bn(S) \cup \underline{s})$.

A relation $\mathcal{R}$ is an $\mathcal{S}$-n-bisimulation *if both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are $\mathcal{S}$-n-simulations.* $\mathcal{S}$-n-observational equivalence *($\approx^n_{\mathcal{S}}$) is the largest $\mathcal{S}$-n-bisimulation.*

Intuitively $\approx^n_{\mathcal{S}}$ is like observational equivalence on processes that do not contain n where the environment is restricted to be a process from $\mathcal{S}$. It is easy to check that the transitive hull of $\approx^n_{\mathcal{S}}$ satisfies the conditions (i), (ii) and (iii) from above. Hence $\approx^n_{\mathcal{S}}$ contains its own transitive hull and thus is indeed an equivalence relation.

Lemma 5.16 *If a set of processes $\mathcal{S}$ is n-complete and $n \notin (fn(S) \cup bn(S))$ for all $S \in \mathcal{S}$, then $P \approx^n_{\mathcal{S}} Q \Leftrightarrow P \approx Q$ for all closed processes P, Q with $n \notin (fn(P) \cup fn(Q) \cup bn(P) \cup bn(Q))$.*

Proof.

Let $P, Q \in \{(P, Q) : P, Q \text{ closed processes with } n \notin (fn(P) \cup fn(Q) \cup bn(P) \cup bn(Q)\}$.

$P \approx Q \Rightarrow P \approx^n_{\mathcal{S}} Q$. We show that observational equivalence restricted to processes that do not contain n is an $\mathcal{S}$-n-bisimulation. Points (i) and (iii) of Definition 5.15 follow directly from points (i) and (iii) of observational equivalence (see Definition 2.4). It remains to show that for $P \rightarrow P'$ with $n \notin fn(P') \cup bn(P')$ we can find a sequence of corresponding internal reductions for Q. Since $P \approx Q$ we find a sequence $Q =: Q_1 \rightarrow \ldots \rightarrow Q_\ell =: Q'$ with $P' \approx Q'$. However, we do not necessarily have $n \notin fn(Q') \cup bn(Q')$ since this is not a requirement for observational equivalence. Fortunately, we we will see that we can find a process $\hat{Q}'$ with $Q \rightarrow^* \hat{Q}'$, $P' \approx \hat{Q}'$ and $n \notin fn(\hat{Q}') \cup bn(\hat{Q}')$. For this, we transform the sequence $Q_1 \rightarrow \ldots \rightarrow Q_\ell$ to a sequence $\hat{Q}_1 \rightarrow \ldots \rightarrow \hat{Q}_\ell$ with $Q_i \equiv_{\mathrm{E}} \hat{Q}_i$ and $n \notin fn(\hat{Q}_i) \cup bn(\hat{Q}_i)$ for $i \in \{1, \ldots, \ell\}$: First, we set $\hat{Q}_1 := Q_1$ and in particular have $Q_1 \equiv_{\mathrm{E}} \hat{Q}_1$ and $n \notin fn(\hat{Q}_1) \cup bn(\hat{Q}_1)$. For $i \in \{2, \ldots, \ell\}$ we define $\hat{Q}_i$ as follows: By Lemma 3.5, since $\hat{Q}_{i-1} \equiv_{\mathrm{E}} Q_{i-1} \rightarrow Q_i$, we find $\tilde{Q}$ with $\hat{Q}_{i-1} \rightarrow \tilde{Q} \equiv_{\mathrm{E}} Q_i$. W.l.o.g. we can assume $n \notin bn(\tilde{Q})$ since $\rightarrow$ and $\equiv_{\mathrm{E}}$ allow for renaming of bound names. We distinguish two cases:

- $n \notin fn(\tilde{Q})$: Then $\hat{Q}_i := \tilde{Q}$ meets our requirements.
- $n \in fn(\tilde{Q})$: Since $\hat{Q}_{i-1} \to \tilde{Q}$ and $n \notin fn(\hat{Q}_{i-1})$, the free occurrences of n can only be the result of a destructor evaluation (LET-THEN, Figure 3). Let D denote the corresponding destructor term with $D \Downarrow T$. By Definition 2.5 (vii) (natural symbolic model) and since $n \notin fn(D)$ we find a term T' with $n \notin fn(T')$ such that $D \Downarrow T'$ and $T =_{\mathrm{E}} T'$. Then $\hat{Q}_i := \tilde{Q}\{T/T'\}$ meets our requirements.

Finally, $\hat{Q}_\ell$ does not contain n and $Q = \hat{Q}_1 \to^* \hat{Q}_\ell \equiv_{\mathrm{E}} Q_\ell = Q' \approx P'$. Hence $(P', \hat{Q}_\ell) \in \approx \cap \{(P, Q) : P, Q \text{ closed processes with } n \notin (fn(P) \cup fn(Q) \cup bn(P) \cup bn(Q)\}$ and thus observational equivalence restricted to processes that do not contain n fulfills Definition 5.15 (ii).

$P \approx^n_{\mathcal{S}} Q \Rightarrow P \approx Q$. We first introduce a bisimulation $\approx_\varphi$ and then show $P \approx^n_{\mathcal{S}} Q \Rightarrow P \approx_\varphi Q \Rightarrow P \approx Q$: Let $\varphi : \mathcal{N} \to \mathcal{N} \setminus \{n\}$ be a bijection on names. We define

$$\approx_\varphi := \{(P, Q) : P\varphi \approx^n_{\mathcal{S}} Q\varphi\}$$

We claim that $\approx_\varphi$ is a bisimulation: It is easy to verify that $\approx_\varphi$ satisfies points (i) and (ii) of Definition 2.4 (both follow straightforwardly by Definition 5.15). For point (iii) we have to show $\mathcal{C}[P] \approx_\varphi \mathcal{C}[Q]$, i.e., $\mathcal{C}[P]\varphi \approx^n_{\mathcal{S}} \mathcal{C}[Q]\varphi$, for all evaluation contexts $\mathcal{C}$ and $P \approx_\varphi Q$, i.e., $P\varphi \approx^n_{\mathcal{S}} Q\varphi$. For any evaluation context $\mathcal{C}$ we have $\mathcal{C}[\Box] \equiv \nu \underline{n}.(C|\Box)$ for some process C and names $\underline{n} \subseteq \mathcal{N}$. Due to the completeness of $\mathcal{S}$ we find an evaluation context $\widetilde{\mathcal{C}}[\Box] := \nu \underline{n}\varphi.(\widetilde{C}|\Box)$ such that $C\varphi \approx \widetilde{C}$ with $\widetilde{C} \in \mathcal{S}$. Since n is not in the range of φ and $n \notin (fn(\widetilde{C}) \cup bn(\widetilde{C}))$ for $\widetilde{C} \in \mathcal{S}$ we have $\widetilde{\mathcal{C}}[P\varphi] \approx^n_{\mathcal{S}} \widetilde{\mathcal{C}}[Q\varphi]$. Furthermore $\widetilde{\mathcal{C}}[P\varphi] \approx \mathcal{C}[P]\varphi$ and hence (both sides do not contain n) $\widetilde{\mathcal{C}}[P\varphi] \approx^n_{\mathcal{S}} \mathcal{C}[P]\varphi$ (analogously for Q). Altogether we have $\mathcal{C}[P]\varphi \approx^n_{\mathcal{S}} \widetilde{\mathcal{C}}[P\varphi] \approx^n_{\mathcal{S}} \widetilde{\mathcal{C}}[Q\varphi] \approx^n_{\mathcal{S}} \mathcal{C}[Q]\varphi$. Since $\approx_\varphi$ is symmetric by definition this closes the proof of our claim that $\approx_\varphi$ is a bisimulation.

We have that $P \approx^n_{\mathcal{S}} Q$ entails $P \approx_\varphi Q$ by definition of $\approx_\varphi$. Furthermore $P \approx_\varphi Q$ entails $P \approx Q$ since $\approx$ is the largest bisimulation. Hence $P \approx^n_{\mathcal{S}} Q$ entails $P \approx Q$. This closes the second part of our proof. □

In the following we fix a name n_{sid} and closed term M_{sid} with $n_{sid} \notin fn(M_{sid})$.

Definition 5.17 (Sid-sensitive processes) *$\mathcal{S}_{sid}$, the set of* sid-sensitive processes, *is the set of processes following the grammar from Figure 4.*

Definition 5.18 ($\mathcal{S}_{sid}$-transformation) *We define the function $\Phi : P \mapsto \Phi(P) = S$, which maps a closed process P with $n_{sid} \notin P$ to a sid-sensitive process $S \in \mathcal{S}_{sid}$, as follows:*

1. *For each protected occurrence of an input $C(x).P'$ in P we replace $C(x).P'$ by*

$$\text{if } M_{sid} = fst(C) \text{ then } (\text{let } y = snd(C) \text{ in } (M_{sid}, y)(x).P') \text{ else } C(x).P'$$

2. *For each occurrence of an output in P we proceed analogously.*

$$
\begin{aligned}
P, Q ::= \; & \mathbf{0} \\
& (M_{sid}, C)(x).P \\
& \overline{(M_{sid}, C)}\langle T\rangle.P \\
& C^*(x).P \\
& \overline{C^*}\langle T\rangle.P \\
& \text{if } M_{sid} = \mathit{fst}(C) \text{ then } P \text{ else } C(x).Q \\
& \text{if } M_{sid} = \mathit{fst}(C) \text{ then } P \text{ else } \overline{C}\langle T\rangle.Q \\
& P \mid Q \\
& !P \\
& \nu a.P \\
& \text{let } x = D \text{ in } P \text{ else } Q
\end{aligned}
$$

Figure 4: Syntax of sid-sensitive processes. M_{sid} is the fixed term. C, T range over all terms with $n_{sid} \notin \mathit{fn}(C)$ and $n_{sid} \notin \mathit{fn}(T)$, C^* over all terms with $n_{sid} \notin \mathit{fn}(C^*)$ such that there is no substitution σ with $C^*\sigma =_{\mathrm{E}} (M_{sid}, \Box)$ for some term $\Box$. D is a destructor term with $n_{sid} \notin \mathit{fn}(D)$ and $a \neq n_{sid}$ is a name. Note that in the if-constructions both occurrences of C stand for the same term.

Lemma 5.19 *$\mathcal{S}_{sid}$ is n_{sid}-complete.*

Proof.

- Claim 1: For all processes P we have

$$\text{if } M_{sid} = \mathit{fst}(C) \text{ then } (\text{let } y = \mathit{snd}(C) \text{ in } (M_{sid}, y)(x).P) \text{ else } C(x).P \approx\!\!\!\approx C(x).P \tag{4}$$

  (analogously for outputs). Proof: Let σ be a closing substitution for Equation 4. We remember that

$$\text{if } M_{sid} = \mathit{fst}(C) \text{ then } (\text{let } y = \mathit{snd}(C) \text{ in } (M_{sid}, y)(x).P) \text{ else } C(x).P$$

  is just syntactic sugar for

$$\text{let } z = \mathit{equals}(M_{sid}, \mathit{fst}(C)) \text{ in } (\text{let } y = \mathit{snd}(C) \text{ in } (M_{sid}, y)(x).P) \text{ else } C(x).P$$

  By definition of *equals* we have $\mathit{equals}(M_{sid}, \mathit{fst}(C))\sigma \Downarrow M_{sid}$ iff $\mathit{fst}(C)\sigma \Downarrow M_{sid}$. We distinguish two cases:

  - If $\mathit{fst}(C\sigma) \Downarrow M_{sid}$, then by Definition 2.5 (v) (natural symbolic model) we have

that $(M_{sid}, C_2) =_{\mathrm{E}} C\sigma$ for all C_2 with $snd(C\sigma) \Downarrow C_2$. Hence

$$\begin{aligned}
&\text{if } M_{sid} = fst(C\sigma) \text{ then } (\text{let } y = snd(C\sigma) \text{ in } (M_{sid}, y)(x).P\sigma) \text{ else } C\sigma(x).P\sigma \\
\overset{(*)}{\approx}\ &\text{if } M_{sid} = fst((M_{sid}, C_2)) \text{ then} \\
&\quad \text{let } y = snd((M_{sid}, C_2)) \text{ in } (M_{sid}, y)(x).P\sigma \\
&\text{else} \\
&\quad (M_{sid}, C_2)(x).P\sigma \\
\overset{(**)}{\approx}\ &\text{let } y = snd((M_{sid}, C_2)) \text{ in } (M_{sid}, y)(x).P\sigma \\
\overset{(**)}{\approx}\ &(M_{sid}, C_2)(x).P\sigma \\
\overset{(*)}{\approx}\ &C(x).P
\end{aligned}$$

$(*)$ by Lemma 3.2 (iv) and $(**)$ by Lemma 3.2 (vii).

– If $fst(C)\sigma \not\Downarrow M_{sid}$, then the claim follows by Lemma 3.2 (vi).

- Claim 2: $P \lessapprox \Phi(P)$. We prove this by structural induction on P. Since Φ does only affect in- and outputs we can focus on those: If $P = C(x).P'$ then

$$\begin{aligned}
P &= C(x).P' \\
&\overset{(*)}{\lessapprox} C(x).\Phi(P') \\
&\overset{(**)}{\lessapprox} \text{if } M_{sid} = fst(C) \text{ then } (\text{let } y = snd(C) \text{ in } (M_{sid}, y)(x).P') \text{ else } C(x).P' \\
&= \Phi(P)
\end{aligned}$$

where $(*)$ holds by the induction hypothesis and $(**)$ by Claim 1.

For any closed P we have $P \lessapprox \Phi(P)$ by Claim 2. $\Phi(P)$ is closed since P is closed and hence $P \approx \Phi(P)$. For P with $n_{sid} \notin (fn(P) \cup bn(P))$ we have $\Phi(P) \in \mathcal{S}_{sid}$. Thus $\mathcal{S}_{sid}$ is n_{sid}-complete. □

Lemma 5.20 *For closed $S \in \mathcal{S}_{sid}$ and $S \to S'$ with $n_{sid} \notin fn(S') \cup bn(S')$ we have $S' \in \mathcal{S}_{sid}$.*

Proof. First, we observe that all processes not containing n_{sid} and being structurally equivalent to a sid-sensitive process are sid-sensitive as well. Furthermore $\mathcal{C}[P]$, where $\mathcal{C}$ is an evaluation context and P a process, is sid-sensitive iff $\mathcal{C}[\mathbf{0}]$ and P are sid-sensitive. In all cases w.l.o.g. $n_{sid} \notin fn(\mathcal{C}) \cup bn(\mathcal{C})$ because $\equiv$ does not introduce free names and bound names are w.l.o.g. not n_{sid}. We have the following cases:

- REPL: $S \equiv \mathcal{C}[!P] \to \mathcal{C}[P|!P] \equiv S'$. $!P$ is sid-sensitive, hence P and $P|!P$ are.
- COMM: $S \equiv \mathcal{C}[\overline{C}\langle T\rangle.P|\tilde{C}(x).Q] \to \mathcal{C}[P|Q\{T/x\}] \equiv S'$. Q is sid-sensitive and $n_{sid} \notin fn(T)$ since $n_{sid} \notin fn(S) \cup bn(\mathcal{C})$. We can easily check the grammar of sid-sensitive processes from Figure 4 to see that a substitution $\{T/x\}$ with $n_{sid} \notin T$ applied to a sid-sensitive process yields a sid-sensitive process. Therefore $Q\{T/x\}$ and $P|Q\{T/x\}$ are sid-sensitive.

$$
\begin{aligned}
P, Q ::= {} & \mathbf{0} \\
& C(x).P \\
& \overline{C}\langle T\rangle.P \\
& (n_{sid}, C^*)(x).P \\
& \overline{(n_{sid}, C^*)}\langle T\rangle.P \\
& \text{if } M_{sid} = \mathit{fst}(C) \text{ then } P \text{ else } (n_{sid}, C)(x).Q \\
& \text{if } M_{sid} = \mathit{fst}(C) \text{ then } P \text{ else } \overline{(n_{sid}, C)}\langle T\rangle.Q \\
& P \mid Q \\
& !P \\
& \nu a.P \\
& \text{let } x = D \text{ in } P \text{ else } Q
\end{aligned}
$$

Figure 5: Syntax of n_{sid}-good processes. M_{sid} is the fixed term. C, T range over all terms with $n_{sid} \notin \mathit{fn}(C)$, $n_{sid} \notin \mathit{fn}(T)$. C^* ranges over all terms with $n_{sid} \notin \mathit{fn}(C^*)$ such that there is no substitution σ with $C^*\sigma =_{\mathrm{E}} (M_{sid}, T)$ for some term T. D is a destructor term with $n_{sid} \notin \mathit{fn}(D)$ and $a \neq n_{sid}$ is a name. Note that in the if-constructions both occurrences of C stand for the same term.

- LET-THEN: $S \equiv \mathcal{C}[\text{let } x = D \text{ in } P \text{ else } Q] \to \mathcal{C}[P\{T/x\}] \equiv S'$ for some term T with $D \Downarrow T$ and $n_{sid} \notin \mathit{fn}(T)$ since $n_{sid} \notin \mathit{fn}(S') \cup \mathit{bn}(\mathcal{C})$. Analogously to the argument in the COMM case, $P\{T/x\}$ is sid-sensitive.
- LET-ELSE: Here, according to the grammar of sid-sensitive processes from Figure 4, we distinguish three cases:
  - $S \equiv \mathcal{C}[\text{if } M_{sid} = \mathit{fst}(C) \text{ then } P \text{ else } C(x).Q] \to \mathcal{C}[C(x).Q] \equiv S'$. C is closed since S is closed. $M_{sid} = \mathit{fst}(C)$ is false, i.e., there is no term M such that $\mathit{equals}(M_{sid}, \mathit{fst}(C)) \Downarrow M$. Therefore $\mathit{fst}(C) \not\Downarrow =_{\mathrm{E}} M_{sid}$. This implies $C \neq_{\mathrm{E}} (M_{sid}, X)$ for all terms X by Definition 2.5 (v) (natural symbolic model). Hence $C(x).Q$ is sid-sensitive (matching the $C^*(x).P$ rule).
  - $S \equiv \mathcal{C}[\text{if } M_{sid} = \mathit{fst}(C) \text{ then } P \text{ else } \overline{C}\langle T\rangle.Q] \to \mathcal{C}[\overline{C}\langle T\rangle.Q] \equiv S'$. Analogously to the previous case.
  - $S \equiv \mathcal{C}[\text{let } x = D \text{ in } P \text{ else } Q] \to \mathcal{C}[Q] \equiv S'$. Q is sid-sensitive by definition.

This concludes our proof. □

Definition 5.21 (n_{sid}-good) *A process P is n_{sid}-good if it follows the grammar from Figure 5.*

Definition 5.22 (tag) *We define the function tag on terms:*

$$\begin{aligned} tag((n_{sid}, C)) :=& C \\ tag(C) :=& (M_{sid}, C) \text{ otherwise} \end{aligned}$$

Let P be an n_{sid}-good process. Then we write $tag(P)$ for the process that results from replacing any channel identifier C by $tag(C)$ in P.

The function *tag* adds a tag M_{sid} to all channel identifiers in a process. We will see that *tag* returns a sid-sensitive process. We will need that *tag* is a bijective mapping between n_{sid}-good processes and sid-sensitive processes. The special name n_{sid} is needed to cover the corner cases when constructing that bijection.

Lemma 5.23 *Let P be an n_{sid}-good process. Then $tag(P) \in \mathcal{S}_{sid}$.*

Proof. We do a structural induction over the grammar of n_{sid}-good processes from Figure 5. Assume that $tag(P')$ and $tag(Q')$ are in $\mathcal{S}_{sid}$.

- For the communication on a channel C with $n_{sid} \notin fn(C)$ we have $tag(C(x).P') = (M_{sid}, C)(x).tag(P')$ which is obviously in $\mathcal{S}_{sid}$. $tag(\overline{C}\langle T\rangle.P')$ analogous.
- For the communication on a channel $C = (n_{sid}, C^*)$ we have $tag((n_{sid}, C^*)(x).P') = C^*(x).tag(P')$. $C^*(x).tag(P')$ is in $\mathcal{S}_{sid}$ since, by definition of n_{sid}-good, there is no substitution σ with $C^*\sigma =_{\mathrm{E}} (M_{sid}, T)$ for some term T. $\overline{(n_{sid}, C^*)}\langle T\rangle.P'$ analogous.
- For the first pair of if statements we have that

$$\begin{aligned} &tag(\text{if } M_{sid} = fst(C) \text{ then } P' \text{ else } (n_{sid}, C)(x).Q') \\ =\ &(\text{if } M_{sid} = fst(C) \text{ then } tag(P') \text{ else } C(x).tag(Q')) \end{aligned}$$

  is in $\mathcal{S}_{sid}$ since $n_{sid} \notin fn(C)$. Analogous for $\overline{(n_{sid}, C)}\langle T\rangle.Q'$ in the ELSE branch.

Checking the remaining rules from Figure 5 is a straightforward task. □

Definition 5.24 (untag) *We define the function untag on terms:*

$$\begin{aligned} untag((M_{sid}, C)) :=& C \\ untag(C) :=& (n_{sid}, C) \text{ otherwise} \end{aligned}$$

Let P be a sid-sensitive process. Then we write $untag(P)$ for the process that results from replacing any channel identifier C by $untag(C)$.

Lemma 5.25 *Let $P \in \mathcal{S}_{sid}$ be a sid-sensitive process. Then $untag(P)$ is n_{sid}-good.*

Proof. Analogous to the proof of Lemma 5.23 a straightforward structural induction shows this Lemma. We quickly sketch the interesting cases:

- $untag((M_{sid}, C)(x).P') = C(x).untag(P')$ matches rule $C(x).P$ from Figure 5 (note that $n_{sid} \notin fn(C)$). $\overline{(M_{sid}, C)}\langle T\rangle.P'$ analogous.
- $untag(C^*(x).P') = (n_{sid}, C^*)(x).untag(P')$: $untag(C^*) = (n_{sid}, C^*)$ since there is no substitution σ with $C^*\sigma =_{\mathrm{E}} (M_{sid}, \Box)$ for some term $\Box$. The expression matches rule $(n_{sid}, C^*)(x).P$ from Figure 5. $\overline{C^*}\langle T\rangle.P$ analogous.
- For the first if-rule we distinguish two cases:
  - $C \neq (M_{sid}, \Box)$. Then

    $$\begin{aligned} &untag(\text{if } M_{sid} = fst(C) \text{ then } P' \text{ else } C(x).Q') \\ &= (\text{if } M_{sid} = fst(C) \text{ then } untag(P') \text{ else } (n_{sid}, C)(x).untag(Q')) \end{aligned}$$

    matches rule (if $M_{sid} = fst(C)$ then P else $(n_{sid}, C)(x).Q$) from Figure 5.
  - $C = (M_{sid}, C')$. Then

    $$\begin{aligned} &untag(\text{if } M_{sid} = fst(C) \text{ then } P' \text{ else } C(x).Q') \\ &= (\text{if } M_{sid} = fst((M_{sid}, C')) \text{ then } untag(P') \text{ else } C'(x).untag(Q')) \\ &= (\text{let } y = equals(M_{sid}, fst((M_{sid}, C'))) \text{ in } untag(P') \text{ else } C'(x).untag(Q')) \end{aligned}$$

    $n_{sid} \notin fn(C')$ since $n_{sid} \notin fn(C)$. Hence $C'(x).untag(Q')$ is n_{sid}-good. The process (let $y = equals(M_{sid}, fst((M_{sid}, C')))$ in $untag(P')$ else $C'(x).untag(Q')$) matches rule (let $x = D$ in P else Q) from Figure 5 with $D = equals(M_{sid}, fst((M_{sid}, C')))$. Analogous for $\overline{C}\langle T\rangle.Q'$ in the ELSE branch.

□

Lemma 5.26 *Let P be an n_{sid}-good process. Then $untag(tag(P)) \approxeq P$.*

Proof.

We prove this lemma by structural induction over P according to the grammar from Figure 5.

- $P = C(x).P'$ where C is a channel identifier with $n_{sid} \notin C$: Then $C \neq (n_{sid}, C')$ for some term C' and thus $tag(C) = (M_{sid}, C)$. Hence $untag(tag(C)) = C$ and $untag(tag(P)) = untag(tag(C(x).P')) = C(x).untag(tag(P')) \approxeq C(x).P' = P$ by the induction hypothesis and since $\approxeq$ is closed under the application of contexts (Lemma 2.7). $P = \overline{C}\langle T\rangle.P'$ analogously.

- $P = (n_{sid}, C^*)(x).P'$ for some term C^* with $n_{sid} \notin fn(C^*)$ and $C^*\sigma \neq_{\mathrm{E}} (M_{sid}, \tilde{C}^*)$ for all substitutions σ and terms $\tilde{C}^*$. Certainly $tag((n_{sid}, C^*)) = C^*$. By assumption $C^* \neq (M_{sid}, \tilde{C}^*)$ and thus $untag(tag((n_{sid}, C^*))) = untag(C^*) = (n_{sid}, C^*)$. The rest of this case, as well as the case for $P = \overline{(n_{sid}, C^*)}\langle T\rangle.P'$, is analogous to the previous case.

- $P =$ if $M_{sid} = fst(C)$ then P' else $(n_{sid}, C)(x).Q'$ where $n_{sid} \notin fn(C)$: Clearly $tag((n_{sid}, C)) = C$. We now distinguish two cases for C:

– $C = (M_{sid}, C')$ for some term C'. Then $untag(C) = untag((M_{sid}, C')) = C' \neq C$. This is the reason why we cannot have $untag(tag(P)) = P$ in general. However,

$$\begin{aligned}
&untag(tag(P))\\
&= untag(tag(\text{if } M_{sid} = fst(C) \text{ then } P' \text{ else } (n_{sid}, C)(x).Q'))\\
&= \text{if } M_{sid} = fst((M_{sid}, C')) \text{ then } untag(tag(P')) \text{ else } untag(tag((n_{sid}, C)(x).Q'))\\
&\overset{(*)}{\approx} untag(tag(P')) \overset{(**)}{\approx} P'\\
&\overset{(*)}{\approx} \text{if } M_{sid} = fst((M_{sid}, C')) \text{ then } P' \text{ else } (n_{sid}, C)(x).Q'\\
&= \text{if } M_{sid} = fst(C) \text{ then } P' \text{ else } (n_{sid}, C)(x).Q' = P
\end{aligned}$$

In both cases $(*)$ holds by Lemma 3.2 (vii) and Definition 2.5 (iv) (natural symbolic model). $(**)$ holds by the induction hypothesis.

– Otherwise $untag(C) = (n_{sid}, C)$ and it is easy to see that $untag(tag(P)) = P$.

$P = \text{if } M_{sid} = fst(C) \text{ then } P' \text{ else } \overline{(n_{sid}, C)}\langle T\rangle.Q'$ analogously.

The missing cases for parallel composition, bang, name restriction and let-statement all work straightforwardly.

□

Lemma 5.27 *Let P be a sid-sensitive process. Then $tag(untag(P)) = P$.*

Proof. Since *tag* and *untag* do only modify channel identifiers we show $tag(untag(C)) = C$ for the different kinds of channel identifiers that are allowed in an sid-sensitive process by Figure 4:

- C is a channel identifier with $C = (M_{sid}, C')$ for some term C' with $n_{sid} \notin fn(C')$: Then $untag(C) = C'$ and $tag(C') = (M_{sid}, C') = C$ since $n_{sid} \notin fn(C)$. Hence $untag(tag(C)) = C$.
- C is a channel identifier C^* with $n_{sid} \notin fn(C^*)$ and $C^*\sigma \neq_{\mathrm{E}} (M_{sid}, \tilde{C}^*)$ for all substitutions σ and terms $\tilde{C}^*$. Then $tag(untag(C)) = tag((n_{sid}, C^*)) = C^* = C$.
- C is a channel identifier with $n_{sid} \notin fn(C)$ in the ELSE-branch of (if $tag = fst(C)$). We distinguish two cases:
  – $C = (M_{sid}, C')$ for some term C'. Then $untag(C) = C'$ and $tag(C') = (M_{sid}, C')$ since $n_{sid} \notin fn(C') \subseteq fn(C)$.
  – Otherwise $untag(C) = (n_{sid}, C)$ and $tag((n_{sid}, C)) = C$.

  In both cases we have $untag(tag(C)) = C$.

□

Definition 5.28 *We define a relation* $\sim_{\mathcal{S}_{sid}} := \{(P, Q) : P, Q \in \mathcal{S}_{sid}, untag(P) \approx untag(Q)\}$.

Lemma 5.29 *Assume that* $\sim_{\mathcal{S}_{sid}}$ *is an* $\mathcal{S}_{sid}$*-bisimulation and* $P \approx Q$ *for closed* n_{sid}*-good processes* P *and* Q*. Then* $tag(P) \approx tag(Q)$.

Proof. Note that $tag(P)$ and $tag(Q)$ are sid-sensitive processes by Lemma 5.23 and thus do not contain n_{sid}. We have

$$\begin{aligned}
P \approx Q \Rightarrow & untag(tag(P)) \approx P \approx Q \approx untag(tag(Q)) \text{ (by Lemma 5.26)} \\
\Rightarrow & tag(P) \sim_{\mathcal{S}_{sid}} tag(Q) \\
\Rightarrow & tag(P) \approx^{n_{sid}}_{\mathcal{S}_{sid}} tag(Q) \text{ (since } \approx^{n_{sid}}_{\mathcal{S}_{sid}} \text{ is the largest } \mathcal{S}_{sid}\text{-bisimulation by Definition 5.15)} \\
\Rightarrow & tag(P) \approx tag(Q) \text{ (by Lemmas 5.16, 5.19)}
\end{aligned}$$

□

Lemma 5.30 *Let* P *be a closed* n_{sid}*-good process with* $P \equiv_{\mathrm{E}} Q \to Q'$ *for some closed processes* Q*,* Q'*. Then there is a closed* n_{sid}*-good process* P' *such that* $P \to P' \equiv_{\mathrm{E}} Q'$ *and* $tag(P) \to tag(P')$.

Proof. According to Lemma 3.5 we find a closed process $\tilde{P}'$ such that $P \to \tilde{P}' \equiv_{\mathrm{E}} Q'$ (this holds for any P, not just for n_{sid}-good ones). Now we show that if P is additionally n_{sid}-good, there is a closed n_{sid}-good process P' with $P \to P' \equiv_{\mathrm{E}} \tilde{P}'$ and $tag(P) \to tag(P')$ which proves the Lemma.

First, we make some general observations: For $P \to \tilde{P}'$ we find an evaluation context $\mathcal{C}$ and processes R, R' such that $P \equiv \mathcal{C}[R] \to \mathcal{C}[R'] \equiv \tilde{P}'$ and $R \to R'$ is a direct application of one of the rules for internal reductions from Figure 3. Furthermore, it is easy to verify that any process A with $P \equiv A$ and $n_{sid} \notin bn(A)$ is also n_{sid}-good and $tag(P) \equiv tag(A)$. Additionally, $\mathcal{C}[R]$ is n_{sid}-good iff $\mathcal{C}[\mathbf{0}]$ and R are n_{sid}-good. Hence, w.l.o.g. (since $\equiv$ allows for renaming of bound names), we can assume $\mathcal{C}[\mathbf{0}]$ and R to be n_{sid}-good. Since $tag(\mathcal{C}[R]) = tag(\mathcal{C})[tag(R)]$, it remains to show that R' is n_{sid}-good and that $tag(R) \to tag(R')$. We will be able to show this for the REPL, the COMM and the THEN-ELSE rules and have that $P' := \mathcal{C}[R'] \equiv \tilde{P}' \equiv Q'$ in these cases. In the LET-THEN case however, the destructor evaluation might introduce a term T containing a free occurrence of n_{sid}. Fortunately, replacing T with an equivalent term T' will solve the problem and we have that $P' := \mathcal{C}[R'\{T/T'\}] \equiv_{\mathrm{E}} \tilde{P}' \equiv Q'$ for $R'\{T/T'\}$ being n_{sid}-good. In detail:

- REPL: $!R \to \mathcal{C}[R|!R] \equiv \tilde{P}'$ where w.l.o.g. $\mathcal{C}[!R]$ and therefore $\mathcal{C}[R|!R]$ are n_{sid}-good. We set $P' := \mathcal{C}[R|!R]$ and have $tag(P) \equiv tag(\mathcal{C}[!R]) = tag(\mathcal{C})[!tag(R)] \overset{(*)}{\to} tag(\mathcal{C})[tag(R)|!tag(R)] = tag(\mathcal{C}[R|!R]) = tag(P')$. $(*)$ by the REPL rule.
- COMM: Analogously to REPL $P \equiv \mathcal{C}[\overline{C}\langle T\rangle.R|\tilde{C}(x).\tilde{R}] \to \mathcal{C}[R|\tilde{R}\{T/x\}] \equiv \tilde{P}'$ where $C =_{\mathrm{E}} \tilde{C}$ and w.l.o.g. $\mathcal{C}[\overline{C}\langle T\rangle.R|\tilde{C}(x).\tilde{R}]$ and $\mathcal{C}[R|\tilde{R}\{T/x\}]$ are n_{sid}-good. We observe

$$tag(\overline{C}\langle T\rangle.R) = \overline{tag(C)}\langle T\rangle.tag(R) \text{ and } tag(\tilde{C}(x).\tilde{R}) = tag(\tilde{C})(x).tag(\tilde{R})$$

by Definition 5.22. Analogously to REPL we have to show

$$tag(\mathcal{C})[\overline{tag(C)}\langle T\rangle.tag(R)|tag(\tilde{C})(x).tag(\tilde{R})] \to tag(\mathcal{C})[tag(R)|tag(\tilde{R})\{T/x\}]$$

Note that $tag(\tilde{R})\{T/x\} = tag(\tilde{R}\{T/x\})$ since $n_{sid} \notin fn(T)$. Hence it is necessary and sufficient to show $tag(C) =_{\mathrm{E}} tag(\tilde{C})$. Now we distinguish two cases to show $tag(C) =_{\mathrm{E}} tag(\tilde{C})$:

- $C = (n_{sid}, C')$ for some term C'. By assumption we have $C =_{\mathrm{E}} \tilde{C}$ and hence $\tilde{C} =_{\mathrm{E}} (n_{sid}, C')$. By the grammar of n_{sid}-good processes (Figure 5) we have $n_{sid} \notin fn(\tilde{C})$ or $\tilde{C} = (n_{sid}, C^*)$ for some C^*. Lemma 3.1 (iii) above excludes the first case and leaves us with $\tilde{C} = (n_{sid}, C^*)$. By Definition 2.5 (vi) (natural symbolic model) we have $C' =_{\mathrm{E}} C^*$ and hence $tag(\tilde{C}) = C^* =_{\mathrm{E}} C' = tag(C)$.
- $C \neq (n_{sid}, C')$ for any term C'. By the grammar of n_{sid}-good processes (Figure 5) we then have $n_{sid} \notin fn(C)$. $\tilde{C} = (n_{sid}, C')$ for some term C' leads to $C =_{\mathrm{E}} (n_{sid}, C')$ which contradicts Lemma 3.1 (iii). Hence (again by the grammar of n_{sid}-good processes) $n_{sid} \notin fn(\tilde{C})$. Thus $tag(C) = (M_{sid}, C) =_{\mathrm{E}} (M_{sid}, \tilde{C}) = tag(\tilde{C})$.

- LET-THEN: $P \equiv \mathcal{C}[\text{let } x = D \text{ in } R \text{ else } \tilde{R}] \to \mathcal{C}[R\{T/x\}] \equiv \tilde{P}'$ with $D \Downarrow T$. By Definition 2.5 (vii) (natural symbolic model) we find T' with $n_{sid} \notin T'$, $D \Downarrow T'$ and $T' =_{\mathrm{E}} T$. Hence we have

$$P \equiv \mathcal{C}[\text{let } x = D \text{ in } R \text{ else } \tilde{R}] \to \mathcal{C}[R\{T'/x\}] =: P'$$

and $P' \equiv_{\mathrm{E}} \tilde{P}' \equiv Q'$. Altogether

$$\begin{aligned} tag(P) &\equiv tag(\mathcal{C}[\text{let } x = D \text{ in } R \text{ else } \tilde{R}]) \\ &= tag(\mathcal{C})[\text{let } x = D \text{ in } tag(R) \text{ else } tag(\tilde{R})] \\ &\to tag(\mathcal{C})[tag(R)\{T'/x\}] \\ &\overset{(*)}{=} tag(\mathcal{C})[tag(R\{T'/x\})] \\ &\equiv tag(P') \end{aligned}$$

($*$) since $n_{sid} \notin fn(T')$.

- LET-ELSE is not affected by *tag* and the proof is analogous to that for the REPL rule.

□

Lemma 5.31 *Let P be a closed sid-sensitive process and P' be a closed process with $n_{sid} \notin fn(P')$. Then there is a process P^* with $untag(P) \to P^*$ and $P^* \approx untag(P')$.*

Proof. The rest of this proof is partially analogous to that of Lemma 5.30. Similarly, we can focus on the rules from Figure 3 directly. The main difference is that, for some sid-sensitive process R and term T with $n_{sid} \notin fn(T)$, $untag(R)\{T/x\} \neq untag(R\{T/x\})$. Instead, we only have $untag(R)\{T/x\} \approxeq untag(R\{T/x\})$ (we are going to prove that first). Therefore the COMM rule and the LET-THEN rule, where substitutions occur, have to be handled differently. The arguments for the REPL rule and the LET-ELSE rule are analogous.

Claim: If R is a sid-sensitive process, $\mathit{untag}(R)\{T/x\} \approx\!\!\!\!\approx \mathit{untag}(R\{T/x\})$ for all T with $n_{sid} \notin \mathit{fn}(T)$. For all channel identifiers $C = (M_{sid}, C')$ and $C = C^*$ according Figure 4 we obviously have $\mathit{untag}(C)\{T/x\} = \mathit{untag}(C\{T/x\})$ for all substitutions $\{T/x\}$. However, in the ELSE-branch of (if $M_{sid} = \mathit{fst}(C)$), C can be an arbitrary term with $n_{sid} \notin \mathit{fn}(C)$. If $C = (M_{sid}, C')$ for some term C', $\mathit{untag}(C)\{T/x\} = \mathit{untag}(C\{T/x\})$ holds. Otherwise, for a substitution $\{T/x\}$, we distinguish two cases:

- $C\{T/x\} \neq (M_{sid}, C')$ for all terms C'. Then $\mathit{untag}(C)\{T/x\} = (n_{sid}, C\{T/x\}) = \mathit{untag}(C\{T/x\})$.
- Otherwise $C\{T/x\} = (M_{sid}, C')$ for some term C'. Then $\mathit{untag}(C)\{T/x\} = (n_{sid}, C\{T/x\}) \neq C' = \mathit{untag}(C\{T/x\})$. Since $\mathit{fst}(C\{T/x\}) \Downarrow M_{sid}$ the ELSE-branch of R will never be executed and we, analogously to the proof of Lemma 5.26, replace $(n_{sid}, C\{T/x\})$ by C' to have $\mathit{untag}(R)\{T/x\} \approx\!\!\!\!\approx \mathit{untag}(R\{T/x\})$.

Note that P' is sid-sensitive by Lemma 5.20.

We now handle the COMM rule and the LET-THEN rule:

- COMM: Analogously to Lemma 5.30 we have to prove $\mathit{untag}(C) =_{\mathrm{E}} \mathit{untag}(\tilde{C})$ where C and $\tilde{C}$ are the channel identifiers used for communication. By the grammar of sid-sensitive processes from Figure 4 all channel identifiers which occur unrestricted are either of the form (a) (M_{sid}, C') for some term C' or (b) C^* such that $C^*\sigma \neq_{\mathrm{E}} (M_{sid}, C')$ for all substitutions σ and all terms C'. We distinguish two cases
  - $C = (M_{sid}, C')$. $\tilde{C}$ cannot be of form (b) since $C =_{\mathrm{E}} \tilde{C}$. Hence $\tilde{C} = (M_{sid}, \tilde{C}')$ and $C' =_{\mathrm{E}} \tilde{C}$ by Definition 2.5 (vi) (natural symbolic model). Therefore $\mathit{untag}(C) = C' =_{\mathrm{E}} \tilde{C}' = \mathit{untag}(\tilde{C})$.
  - Otherwise, C is of form (b). Then $\tilde{C}$ cannot be of form (a) since $C =_{\mathrm{E}} \tilde{C}$. We thus have $\mathit{untag}(C) = (n_{sid}, C) =_{\mathrm{E}} (n_{sid}, \tilde{C}) = \mathit{untag}(\tilde{C})$.

  We find

  $$\begin{aligned} & P \equiv \mathcal{C}[\overline{C}\langle T\rangle.R | \tilde{C}(x).\tilde{R}] \to \mathcal{C}[R|\tilde{R}\{T/x\}] \equiv P' \\ \Rightarrow & \mathit{untag}(P) \equiv \mathit{untag}(\mathcal{C})[\overline{\mathit{untag}(C)}\langle T\rangle.\mathit{untag}(R) | \mathit{untag}(\tilde{C})(x).\mathit{untag}(\tilde{R})] \\ & \overset{(*)}{\to} \mathit{untag}(\mathcal{C})[\mathit{untag}(R) | \mathit{untag}(\tilde{R})\{T/x\}] =: P^* \end{aligned}$$

  $(*)$ since $\mathit{untag}(C) =_{\mathrm{E}} \mathit{untag}(\tilde{C})$. Due to the claim above $P^* \approx \mathit{untag}(P')$ which proves the COMM case.

- LET-THEN: We have $P \equiv \mathcal{C}[\text{let } x = D \text{ in } R \text{ else } \tilde{R}] \to \mathcal{C}[R\{T/x\}] \equiv P'$. In contrast to Lemma 5.30 the evaluation of the destructor may not lead to a term T with $n_{sid} \in \mathit{fn}(T)$ here if $x \in \mathit{fv}(R)$ since we required P' to be sid-sensitive. (Otherwise, if $x \notin \mathit{fv}(R)$, we obviously have $\mathit{untag}(R)\{T/x\} = \mathit{untag}(R\{T/x\})$.)

Thus

$$\begin{aligned} \mathit{untag}(P) &\equiv \mathit{untag}(\mathcal{C})[\text{let } x = D \text{ in } \mathit{untag}(R) \text{ else } \mathit{untag}(\tilde{R})] \\ &\to \mathit{untag}(\mathcal{C})[\mathit{untag}(R)\{T/x\}] =: P^* \overset{(*)}{\approx} \mathit{untag}(\mathcal{C}[R\{T/x\}]) = \mathit{untag}(P') \end{aligned}$$

(*) due to the claim above. This proves the LET-THEN case.

Since *untag* dos not affect the REPL and LET-ELSE cases these can be handled exactly like the REPL case in the proof of Lemma 5.30. □

Lemma 5.32 *$\sim_{\mathcal{S}_{sid}}$ is an $\mathcal{S}_{sid}$-n_{sid}-bisimulation*

Proof.
Let $(P, Q) \in \sim_{\mathcal{S}_{sid}}$. We show the three points of an $\mathcal{S}_{sid}$-n_{sid}-simulation.

- $P \downarrow_C$: We have $P \downarrow_C$ iff $P \downarrow_{\hat{C}}$ for a channel identifier $\hat{C} =_{\mathrm{E}} C$ which occurs in P and thus follows the grammar from Figure 4. Since $P \sim_{\mathcal{S}_{sid}} Q$: $\mathit{untag}(P) \approx \mathit{untag}(Q)$ holds by definition. Since $P \downarrow_{\hat{C}}$ we have $\mathit{untag}(P) \downarrow_{\mathit{untag}(\hat{C})}$ and thus $\mathit{untag}(Q) =: \hat{Q}_1 \to \ldots \to \hat{Q}_n \downarrow_{\mathit{untag}(\hat{C})}$ for some $n \in \mathbb{N}$ and processes Q_i, $i \in \{1, \ldots, n\}$. By Lemma 5.25 $\hat{Q}_1 = \mathit{untag}(Q)$ is n_{sid}-good. By Lemma 5.30 we get a sequence of n_{sid}-good processes $\hat{Q}'_1 \to \ldots \to \hat{Q}'_n$ with $\hat{Q}'_1 = \hat{Q}_1$, $\hat{Q}'_i \equiv_{\mathrm{E}} \hat{Q}_i$ and $\mathit{tag}(\hat{Q}'_1) \to \ldots \to \mathit{tag}(\hat{Q}'_n)$. Since $\hat{Q}'_1 = \hat{Q}_1 = \mathit{untag}(Q)$ we have $\mathit{tag}(\hat{Q}'_1) = Q$ by Lemma 5.27. Furthermore, $\hat{Q}'_n \equiv_{\mathrm{E}} \hat{Q}_n \downarrow_{\mathit{untag}(\hat{C})}$ implies $\hat{Q}'_n \downarrow_{\mathit{untag}(\hat{C})}$ (see Footnote 7) and $\mathit{tag}(\hat{Q}'_n) \downarrow_{\mathit{tag}(\mathit{untag}(\hat{C}))}$. Since $\hat{C}$ is a term according to Figure 4 we have $\mathit{tag}(\mathit{untag}(\hat{C})) = \hat{C}$ $(=_{\mathrm{E}} C)$ (see Lemma 5.27). Hence $Q = \mathit{tag}(\hat{Q}'_1) \to^* \mathit{tag}(\hat{Q}'_n) \downarrow_C$.

- $P \to P'$ with $n_{sid} \notin \mathit{fn}(P') \cup \mathit{bn}(P')$: According to Lemma 5.31 we find P^* such that $\mathit{untag}(P) \to P^* \approx \mathit{untag}(P')$. Since $P \sim_{\mathcal{S}_{sid}} Q$ we also have $\mathit{untag}(Q) =: \hat{Q}_1 \to \ldots \to \hat{Q}_n \approx P^*$. Analogously to the previous part we find some n_{sid}-good $\hat{Q}'_n$ such that $Q \to^* \mathit{tag}(\hat{Q}'_n)$ and $\hat{Q}'_n \equiv_{\mathrm{E}} \hat{Q}_n$. By Lemma 5.26 we have $\mathit{untag}(\mathit{tag}(\hat{Q}'_n)) \approx \hat{Q}'_n$ ($\hat{Q}'_n$ is closed). Thus $\mathit{untag}(\mathit{tag}(\hat{Q}'_n)) \approx \hat{Q}'_n \equiv_{\mathrm{E}} \hat{Q}_n \approx P^* \approx \mathit{untag}(P')$ which implies $\mathit{untag}(\mathit{tag}(\hat{Q}'_n)) \approx \mathit{untag}(P')$ since $\equiv_{\mathrm{E}}$ entails $\approx$ by Lemma 3.2 (iv). Hence $Q \to^* \mathit{tag}(\hat{Q}'_n)$ and $P' \sim_{\mathcal{S}_{sid}} \mathit{tag}(\hat{Q}'_n)$.

- Assume $P \sim_{\mathcal{S}_{sid}} Q$ and let $R \in \mathcal{S}_{sid}$ be a process and $\underline{a}$ names. We have $\mathit{untag}(P) \approx \mathit{untag}(Q)$ by definition of $\sim_{\mathcal{S}_{sid}}$ and $\approx$ is closed under the application of evaluation contexts. Hence $\mathit{untag}(\nu\underline{a}.(P \mid R)) = \nu\underline{a}.(\mathit{untag}(P)|\mathit{untag}(R)) \approx \nu\underline{a}.(\mathit{untag}(Q)|\mathit{untag}(R)) = \mathit{untag}(\nu\underline{a}.(Q|R))$. Thus, by definition of $\sim_{\mathcal{S}_{sid}}$, $\nu\underline{a}.(P|R) \sim_{\mathcal{S}_{sid}} \nu\underline{a}.(Q|R)$.

Since $\sim_{\mathcal{S}_{sid}}$ is symmetric it is an $\mathcal{S}_{sid}$-n_{sid}-bisimulation.

□

Lemma 5.33 *Let P and Q be closed processes and M be an arbitrary closed term. Then $P \approx Q \Rightarrow P(\!(M)\!) \approx Q(\!(M)\!)$.*

Proof. Fix a name $n_{sid} \notin (fn(M) \cup fn(P) \cup bn(P) \cup fn(Q) \cup bn(Q))$and $M_{sid} := M$. Remember that all lemmas in this section were proven for an arbitrary fixed M_{sid} with $n_{sid} \notin fn(M_{sid})$. Now P, Q are n_{sid}-good and $P(\!(M_{sid})\!) = tag(P)$ and $Q(\!(M_{sid})\!) = tag(Q)$. By Lemmas 5.29,5.32: $tag(P) \approx tag(Q)$. Hence $P(\!(M)\!) = P(\!(M_{sid})\!) \approx Q(\!(M_{sid})\!) = Q(\!(M)\!)$. □

Lemma 5.34 *Let P and Q be processes and M be a term with $fv(M) \cap (bv(P) \cup bv(Q)) = \varnothing$. Then $P \approx Q \Rightarrow P(\!(M)\!) \approx Q(\!(M)\!)$.*

Proof. For all closing substitutions σ we have $P \approx Q \Rightarrow P\sigma \approx Q\sigma$. By Lemma 5.33 we have $P\sigma(\!(M\sigma)\!) \approx Q\sigma(\!(M\sigma)\!)$ for the closed processes $P\sigma$ and $Q\sigma$ and the closed term $M\sigma$. This entails $P(\!(M)\!)\sigma \approx Q(\!(M)\!)\sigma$ since $fv(M) \cap (bv(P) \cup bv(Q)) = \varnothing$. Therefore $P(\!(M)\!) \approx Q(\!(M)\!)$. □

Proof of Lemma 5.13. By Lemma 5.34 $P(\!(x)\!) \approx Q(\!(x)\!)$. According to Definition 5.11 $!!_x P = C^{x,\underline{np}}_{SID}[P(\!(x)\!)]$ for some names $\underline{np} \cap fn(P) = \varnothing$ and $!!_x Q = C^{x,\underline{nq}}_{SID}[Q(\!(x)\!)]$ for some names $\underline{nq} \cap fn(Q) = \varnothing$. Let $\underline{n}$ be names such that $\underline{n} \cap (fn(P) \cup fn(Q)) = \varnothing$ and $|\underline{n}| \geq \max(|\underline{np}|, |\underline{nq}|)$. We have

$$C^{x,\underline{np}}_{SID}[P(\!(x)\!)] \equiv C^{x,\underline{n}}_{SID}[P(\!(x)\!)] \overset{(*)}{\approx} C^{x,\underline{n}}_{SID}[Q(\!(x)\!)] \equiv C^{x,\underline{nq}}_{SID}[Q(\!(x)\!)]$$

(*) since $P(\!(x)\!) \approx Q(\!(x)\!)$ and $\approx$ is closed under the application of contexts (Lemma 2.7). Therefore $!!_x P \approx !!_x Q$. □

Note that Lemma 5.13 also implies $P \approx Q \Rightarrow !!P \approx !!Q$.

Lemma 5.35 *Let P be a process and n be a name that occurs only in channel identifiers in P. Then $\nu n.!!_x P \approx !!_x \nu n.P$ for all variables $x \notin bv(P)$.*

Proof. First, we observe that instances of P with distinct tags cannot communicate with each other. This can be formalized by the following

Claim. Let $id, id' \in SID$ be distinct IDs and P, Q arbitrary processes. Then $P(\!(id)\!) \rightarrow^* \updownarrow_C$ and $Q(\!(id')\!) \rightarrow^* \updownarrow_{C'}$ for terms C, C' implies $C \neq_{\mathrm{E}} C'$. Proof: By Definition 5.1 every channel identifier in $P(\!(id)\!)$ is of the form (id, X) for some term X. Analogously, every channel identifier in $Q(\!(id')\!)$ is of the form (id', Y). Towards contradiction we assume $C = (id, X) =_{\mathrm{E}} (id', Y) = C'$. Then, by Definition 2.5 (vi) (natural symbolic model), we have $id =_{\mathrm{E}} id'$. However, $id \neq_{\mathrm{E}} id'$ is required for all pairs of distinct IDs $id, id' \in SID$. This proves the claim.

It is now easy to check that

$$\mathcal{R} := \{(\mathcal{C}[\nu n.(P_1(\!(id_1)\!)|\ldots|P_\ell(\!(id_\ell)\!)|\prod_{x\in S} P(\!(x)\!))], \mathcal{C}[\nu n.P_1(\!(id_1)\!)|\ldots|\nu n.P_\ell(\!(id_\ell)\!)|\prod_{x\in S}\nu n.P(\!(x)\!)]) :$$
$$P_1,\ldots,P_\ell \text{ processes where } n \text{ occurs only in channel identifiers,}$$
$$id_1,\ldots,id_l \subseteq \mathit{SID}\setminus S \text{ are distinct, } S\subseteq \mathit{SID} \text{ and } \mathcal{C} \text{ evaluation context}\}$$

is a bisimulation and thereby prove the lemma. Although the P_i in $\mathcal{R}$ are formally arbitrary processes that contain n only in channel identifiers, they intuitively allow to represent the running instances of P. Note that the claim above holds for any pair $P_i(\!(id_i)\!)$, $P_j(\!(id_j)\!)$ with $i \neq j$. Intuitively, since n occurs only in channel identifiers and thus is never transmitted, no context can tell the difference between a private n that is shared among all instances and an n individual n for each instance. □

Lemma 5.36 *Let P and Q be processes. Then $!!_x(P|Q) \approx\!\!\!\approx !!_x P|!!_x Q$ for all variables $x \notin bv(P)\cup bv(Q)$.*

Proof. We use the semantics of product processes (see Definition 2.9) for this proof. By Definition 5.2 and Definition 5.11 we have $!!_x R \approx\!\!\!\approx \prod_{x\in \mathit{SID}} R(\!(x)\!)$ for any process R. Let σ be a closing substitution for $!!_x P$ and $!!_x Q$ (i.e., $fv(P(\!(x)\!)\sigma), fv(Q(\!(x)\!)\sigma) \subseteq \{x\}$). We set $\Pi_P(X) := \prod_{x\in X} P(\!(x)\!)\sigma$ for arbitrary $X \subseteq \mathit{SID}$ and $\sum_P(X) := \sum_{x\in X} P(\!(x)\!)\sigma = P(\!(x_1)\!)\sigma|\ldots|P(\!(x_\ell)\!)\sigma$ for finite $X = \{x_1,\ldots,x_\ell\} \subseteq \mathit{SID}$. Analogously $\Pi_Q(X)$, $\sum_Q(X)$ and $\Pi_{PQ}(X) := \prod_{x\in X}(P(\!(x)\!)\sigma|Q(\!(x)\!)\sigma)$. We then define the relation $\mathcal{R}$:

$$\mathcal{R} := \{(\mathcal{C}[\sum_P(S_P) \mid \sum_Q(S_Q) \mid \Pi_{PQ}(S_{PQ})],\ \mathcal{C}[\Pi_P(S_{PQ}\cup S_P) \mid \Pi_Q(S_{PQ}\cup S_Q)]) :$$
$$\mathcal{C} \text{ evaluation context}, S_P, S_Q, S_{PQ} \subseteq \mathit{SID}, S_P\cap S_{PQ} = \emptyset, S_Q \cap S_{PQ} = \emptyset\}$$

closed under structural equivalence. Note that

$$\left(\prod_{x\in \mathit{SID}} (P(\!(x)\!)\sigma|Q(\!(x)\!)\sigma),\ \prod_{x\in \mathit{SID}} P(\!(x)\!)\sigma \mid \prod_{x\in \mathit{SID}} Q(\!(x)\!)\sigma\right) \in \mathcal{R}$$

for $S_P := \emptyset$, $S_Q := \emptyset$ and $S_{PQ} := \mathit{SID}$ which proves this lemma if $\mathcal{R} \subseteq \approx$. Therefore, we show the three points of a simulation for $\mathcal{R}$ and $\mathcal{R}^{-1}$ respectively. First, we show that $\mathcal{R}$ is a simulation. For $(A,B) \in \mathcal{R}$:

1. $A\downarrow_C$: Product processes do not emit on channels. Three cases remain:
   (a) If $\mathcal{C}[\mathbf{0}]\downarrow_C$, then $B\downarrow_C$.
   (b) If $P(\!(id)\!)\sigma\downarrow_C$ for some $id \in S_P$, then B can spawn the instance $P(\!(id)\!)\sigma$ from $\Pi_P(S_{PQ}\cup S_P)$ and then emit on C. Hence $B\rightarrow\downarrow_C$.
   (c) Analogously for $Q(\!(id)\!)\sigma\downarrow_C$ for some $id \in S_Q$.
   Hence $A\downarrow_C$ entails $B\rightarrow^*\downarrow_C$.
2. $A \rightarrow A'$: We distinguish two cases:

(a) $\rightarrow$ follows the IREPL rule: Then $\rightarrow$ spawns a new instance with id id from $\Pi_{PQ}(S_{PQ})$: We set $\mathcal{C}'[\Box] := \mathcal{C}[P(\!(id)\!)\sigma \mid Q(\!(id)\!)\sigma \mid \Box]$ and $S'_{PQ} := S_{PQ} \setminus \{id\}$. Hence we have $A \rightarrow \mathcal{C}'[\sum_P(S_P) \mid \sum_Q(S_Q) \mid \Pi_{PQ}(S'_{PQ})]$. Additionally, we observe $B \equiv \mathcal{C}[\Pi_P(S_{PQ} \cup S_P) \mid \Pi_Q(S_{PQ} \cup S_Q)] \rightarrow\!\!\rightarrow \mathcal{C}'[\Pi_P(S'_{PQ} \cup S_P) \mid \Pi_Q(S'_{PQ} \cup S_Q)]$ by spawning $P(\!(id)\!)\sigma$ from $\Pi_P(S_{PQ} \cup S_P)$ and $Q(\!(id)\!)\sigma$ from $\Pi_Q(S_{PQ} \cup S_Q)$. We have $(\mathcal{C}'[\sum_P(S_P) \mid \sum_Q(S_Q) \mid \Pi_{PQ}(S'_{PQ})], \mathcal{C}'[\Pi_P(S'_{PQ} \cup S_P) \mid \Pi_Q(S'_{PQ} \cup S_Q)]) \in \mathcal{R}$.

(b) $\rightarrow$ follows a rule from Figure 3: Then we distinguish two cases:

i. If we have $\mathcal{C}[\mathbf{0}] \rightarrow \mathcal{C}'[\mathbf{0}]$, $\rightarrow$ translates canonically to $\mathcal{C}$ in $B \rightarrow B'$ such that $(A', B') \in \mathcal{R}$.

ii. Otherwise, $\rightarrow$ affects instances from $\sum_P(S_P) \mid \sum_Q(S_Q)$. We remove the ids of the affected instances from S_P and S_Q yielding sets S'_P and S'_Q and define a context $\mathcal{C}'$ (including the affected instances) such that $A \rightarrow \mathcal{C}'[\sum_P(S'_P) \mid \sum_Q(S'_Q) \mid \Pi_{PQ}(S_{PQ})]$. We now spawn the corresponding instances in B first and then mimic $\rightarrow$ exactly yielding $B \rightarrow^* \mathcal{C}'[\Pi_P(S_{PQ} \cup S'_P) \mid \Pi_Q(S_{PQ} \cup S'_Q)]$. We have $(\mathcal{C}'[\sum_P(S'_P) \mid \sum_Q(S'_Q) \mid \Pi_{PQ}(S_{PQ})], \mathcal{C}'[\Pi_P(S_{PQ} \cup S'_P) \mid \Pi_Q(S_{PQ} \cup S'_Q)]) \in \mathcal{R}$.

3. By definition $\mathcal{R}$ is closed under the application of evaluation contexts.

Now we show that $\mathcal{R}^{-1}$ is a simulation. For $(A, B) \in \mathcal{R}^{-1}$:

1. $A \downarrow_C$: Since product processes do not emit on channels we have $\mathcal{C}[\mathbf{0}] \downarrow_C$ and thus $B \downarrow_C$.
2. $A \rightarrow A'$: We distinguish two cases:
   (a) $\rightarrow$ follows the IREPL rule: We distinguish four cases:
      i. A new instance $P(\!(id)\!)\sigma$ is spawned from $\Pi_P(S_{PQ} \cup S_P)$ with $id \in S_P$: We define the context $\mathcal{C}'[\Box] := \mathcal{C}[P(\!(id)\!)\sigma \mid \Box]$, $S'_P := S_P \setminus \{id\}$ and have $A \rightarrow \mathcal{C}'[\Pi_P(S_{PQ} \cup S'_P) \mid \Pi_Q(S_{PQ} \cup S_Q)]$ and $B \equiv \mathcal{C}'[\sum_P(S'_P) \mid \sum_Q(S_Q) \mid \Pi_{PQ}(S_{PQ})]$. Hence $(\mathcal{C}'[\Pi_P(S_{PQ} \cup S'_P) \mid \Pi_Q(S_{PQ} \cup S_Q)], \mathcal{C}'[\sum_P(S'_P) \mid \sum_Q(S_Q) \mid \Pi_{PQ}(S_{PQ})] \in \mathcal{R}^{-1}$.
      ii. A new instance $Q(\!(id)\!)\sigma$ is spawned from $\Pi_Q(S_{PQ} \cup S_Q)$ with $id \in S_Q$: Analogous to the previous case.
      iii. A new instance $P(\!(id)\!)\sigma$ is spawned from $\Pi_P(S_{PQ} \cup S_P)$ with $id \in S_{PQ}$: We define the context $\mathcal{C}'[\Box] := \mathcal{C}[P(\!(id)\!)\sigma \mid \Box]$, $S'_{PQ} := S_{PQ} \setminus \{id\}$, $S'_Q := S_Q \cup \{id\}$ and have $A \rightarrow \mathcal{C}'[\Pi_P(S'_{PQ} \cup S_P) \mid \Pi_Q(S'_{PQ} \cup S'_Q)]$. Note that $S_{PQ} \cup S_Q = S'_{PQ} \cup S'_Q$. In B we spawn $P(\!(id)\!)\sigma \mid Q(\!(id)\!)\sigma$ from $\Pi_{PQ}(S_{PQ})$ and have $B \rightarrow \mathcal{C}'[\sum_P(S_P) \mid \sum_Q(S'_Q) \mid \Pi_{PQ}(S'_{PQ})]$. Hence $(\mathcal{C}'[\Pi_P(S'_{PQ} \cup S_P) \mid \Pi_Q(S'_{PQ} \cup S'_Q)], \mathcal{C}'[\sum_P(S_P) \mid \sum_Q(S'_Q) \mid \Pi_{PQ}(S'_{PQ})] \in \mathcal{R}^{-1}$.
      iv. A new instance $Q(\!(id)\!)\sigma$ is spawned from $\Pi_Q(S_{PQ} \cup S_Q)$ with $id \in S_{PQ}$: Analogous to the previous case.
   (b) $\rightarrow$ follows a rule from Figure 3: Then we basically have $\mathcal{C}[\mathbf{0}] \rightarrow \mathcal{C}'[\mathbf{0}]$ which translates canonically to $\mathcal{C}$ in $B \rightarrow B'$ such that $(A', B') \in \mathcal{R}$.
3. By definition $\mathcal{R}$ is closed under the application of evaluation contexts.

This shows that $\mathcal{R}$ is a bisimulation and hence $\mathcal{R} \subseteq \approx$. $\square$

Alternative definitions of !!. Of course, our definition of $!!P$ is not the only possible definition of a replication with session ids. For example, one might try to define $!!P$ in such a way that an instance of P is spawned for arbitrary terms as sessions id, not only terms in some fixed set *SID*. In particular, a fresh name could then be used as session id which is not possible with our modeling. (Then, of course, the set of processes to which !! may be applied should be restricted to processes which wait for an input before doing anything. Otherwise processes could spawn spontaneously that use some other process' fresh names as session ids.)

Any definition of !! that satisfies Lemmas 5.12, 5.13, 5.35, and 5.36 would lead to the same composition theorem. (If that definition !! is applicable only to a certain set $\mathbf{P}$ of processes, we additionally need that $\mathbf{P}$ is closed under parallel composition, restrictions, renaming, and !!, and that the definition of $\leq$ (Definition 4.3) is with respect to simulators in $\mathbf{P}$.)

The composition theorem. We can now state and prove the composition theorem. It says that if $P \leq Q$, we can restrict the IO-names, compose in parallel with processes that have disjoint NET-names, rename names (as long as NET- and IO-names are not interchanged), and perform concurrent composition.

Theorem 5.37 (Composition Theorem) *Let P, Q be processes with $P \leq Q$. Then*

(i) For any list of names $\underline{io} \subseteq \mathrm{IO}$ we have $\nu \underline{io}.P \leq \nu \underline{io}.Q$.
(ii) For any process R with $(fn(R) \cap (fn(P) \cup fn(Q))) \subseteq \mathrm{IO}$ we have $P|R \leq Q|R$.
(iii) For any permutation $\psi : \mathrm{NET} \to \mathrm{NET}$ we have $P\psi \leq Q$ and $P \leq Q\psi$.
(iv) For any permutation $\psi : \mathrm{IO} \to \mathrm{IO}$ we have $P\psi \leq Q\psi$.
(v) If Q is a NET*-stable process, $!!_x P \leq !!_x Q$ for all variables $x \notin bv(P) \cup bv(Q)$.*

Proof. In the following, let $(S, \varphi, \underline{n})$ be as in Definition 4.3. (They exist because $P \leq Q$.)

(i) $P \approx\!\!\!\approx \nu \underline{n}.(Q\varphi|S) \overset{(*)}{\Rightarrow} \nu \underline{io}.P \approx\!\!\!\approx \nu \underline{io}.\nu \underline{n}.(Q\varphi|S) \overset{(**)}{\approx\!\!\!\approx} \nu \underline{n}.((\nu \underline{io}.Q)\varphi|S)$
($*$) since $\approx\!\!\!\approx$ is closed under the application of evaluation contexts.
($**$) since neither S nor φ contain names from IO

(ii) W.l.o.g. we can assume $fn(R) \cap \underline{n} = \emptyset$ and that φ is the identity on $(fn(R) \cup bn(R)) \cap \mathrm{NET}$. These assumptions guarantee $(*)$ in the upcoming equations. $P \approx\!\!\!\approx \nu \underline{n}.(Q\varphi|S) \Rightarrow P|R \approx\!\!\!\approx \nu \underline{n}.(Q\varphi|S)|R \overset{(*)}{\approx\!\!\!\approx} \nu \underline{n}.((Q|R)\varphi|S)$

(iii) $P \approx\!\!\!\approx \nu \underline{n}.(Q\varphi|S) \Rightarrow P\psi \approx\!\!\!\approx (\nu \underline{n}.(Q\varphi|S))\psi \equiv \nu \underline{n}\psi.(Q(\psi \circ \varphi)|S\psi)$. Therefore, with $(S\psi, \psi \circ \varphi, \underline{n}\psi)$ as simulator, we have $P\psi \leq Q$. With $(S, \varphi \circ \psi^{-1}, \underline{n})$ we have $P \leq Q\psi$.

(iv) $P \approx\!\!\!\approx \nu \underline{n}.(Q\varphi|S) \Rightarrow P\psi \approx\!\!\!\approx (\nu \underline{n}.(Q\varphi|S))\psi \equiv \nu \underline{n}.(Q(\varphi \circ \psi)|S)$ since S, φ and $\underline{n}$ do not contain IO names and thus are not affected by $\psi : \mathrm{IO} \to \mathrm{IO}$.

(v) Note that $Q\varphi$ is NET-stable since Q is NET-stable. Then $P \approx\!\!\!\approx \nu\underline{n}.(Q\varphi|S)$ entails

$$\begin{aligned}
!!_x P &\approx\!\!\!\approx !!_x \nu\underline{n}.(Q\varphi|S) && \text{(by Lemma 5.13)}\\
&\approx\!\!\!\approx \nu\underline{n}.!!_x(Q\varphi|S) && \text{(by Lemma 5.35 since } Q\varphi|S \text{ NET-stable)}\\
&\approx\!\!\!\approx \nu\underline{n}.(!!_x(Q\varphi)|!!_x S) && \text{(by Lemma 5.36)}\\
&\equiv \nu\underline{n}.((!!_x Q)\varphi|!!_x S) && \text{(by Lemma 5.12)}
\end{aligned}$$

Thus $(!!_x S, \varphi, \underline{n})$ is a proper simulator for $!!_x P \leq !!_x Q$. □

6 Property preservation

Besides secure composition, the second salient property of the UC framework is the fact that security properties of the ideal functionality $\mathcal{F}$ automatically carry over to any protocol emulating $\mathcal{F}$. For example, a secure channel functionality that takes a message m from Alice and gives it directly to Bob will obviously have the property that m stays secret. Then, if π UC-emulates $\mathcal{F}$, any message given to π will also stay secret. A similar property preservation law holds in our case, the following theorem formalizes it:

Theorem 6.1 (Property preservation) *Let P, Q be NET-stable processes with $P \leq Q$. Let E_1 and E_2 be contexts whose holes are protected only by parallel compositions ($|$), restrictions (ν), and indexed replications ($!!_x$). Assume that E_1, E_2 do not contain NET-names (neither bound nor free). Assume that the number of $!!_x$ (possibly with different x) over the hole is the same in E_1 and E_2.*

If $E_1[Q] \approx\!\!\!\approx E_2[Q]$, then $E_1[P] \approx\!\!\!\approx E_2[P]$.

Proof. Let b denote the number of $!!_x$ over the hole of E_1, E_2. We write $!!^b S$ for $b \geq 0$ applications of $!!$ to S.

Since $P \leq Q$, there are $S, \varphi, \underline{n}$ with $P \approx\!\!\!\approx \nu\underline{n}.(Q\varphi|S)$ and S closed and NET-stable, and $\mathrm{IO} \cap fn(A) = \emptyset$, $\varphi : \mathrm{NET} \to \mathrm{NET}$ a bijection and $\underline{n}$ a list of names $\underline{n} \subseteq \mathrm{NET}$. Without loss of generality, we can assume that $\underline{n} \cap fn(E_1, E_2) = \underline{n} \cap bn(E_1, E_2) = \emptyset$. For $i = 1, 2$, we have

$$\begin{aligned}
E_i[P] &\overset{\text{(i)}}{\approx\!\!\!\approx} E_i[\nu\underline{n}.(Q\varphi|S)]\\
&\overset{\text{(ii)}}{\approx\!\!\!\approx} \nu\underline{n}.E_i[(Q\varphi|S)]\\
&\overset{\text{(iii)}}{\approx\!\!\!\approx} \nu\underline{n}.(E_i[Q\varphi]|!!^b S)\\
&\overset{\text{(iv)}}{=} \nu\underline{n}.(E_i[Q]\varphi|!!^b S).
\end{aligned}$$

Here (i) uses Lemma 2.7.

And (ii) uses that the names $\underline{n}$ do not occur in E_i, the rules NEW-C and NEW-PAR from Figure 2, and Lemma 5.35 for swapping $!!_x$ in E_i and the names $\underline{n}$ (the preconditions of Lemma 5.35 are fulfilled because $\underline{n}$ are NET-names and thus do not occur in E_i).

```
free netscstart, netnotify, netdeliver, n1, n2.
fun empty/0.

let FSC = in(netstart,y); in(ioA,x);
          ( out(netnotify,empty) | in(netdeliver,z); out(ioB,x) ).

process new ioA; new ioB; out(ioA,choice[n1,n2]) | in(ioB,z) | FSC
```

Figure 6: Proverif code for showing $E_1[\mathcal{F}_{SC}] \approx E_2[\mathcal{F}_{SC}]$ in Lemma 6.3 (`prop-pres.pv`, see [BU13]).

And (iii) uses that the names in E_i (IO-names only) and the names in S (NET-names only) are disjoint, as well as Lemma 5.36 for moving S over a !! in E_i. (Lemma 5.36 guarantees $!!_x(R|S) \approx !!_x R|!!_x S$, this is why S accumulates on $!!_x$ for each $!!_x$ over the hole of E_i. Since S is closed, we can drop the x from $!!_x$.)

And (iv) uses that E_i does not contain NET-names (bound or free) while φ is a substitution on NET-names.

Furthermore, since $\approx\!\!\!\!\approx$ is closed under renaming of free names, and under application of contexts (Lemma 2.7), from $E_1[Q] \approx\!\!\!\!\approx E_2[Q]$ it follows that $\nu\underline{n}.(E_1[Q]\varphi|!!^b S) \approx\!\!\!\!\approx \nu\underline{n}.(E_2[Q]\varphi|!!^b S)$ and hence $E_1[P] \approx\!\!\!\!\approx E_2[P]$. □

Thus, any security property that can be expressed by an indistinguishability game of the form "$E_1[P] \approx\!\!\!\!\approx E_2[P]$" with E_1, E_2 as in the theorem will carry over from the ideal functionality Q to the protocol P, given $P \leq Q$. Note that even many trace based properties can be expressed in such a way. E.g., if we want to say that $E_1[P]$ does not raise an event *bad* (modeled by emitting on a special channel), we just define E_2 to be like E_1, but without the event. Then $E_1[P] \approx\!\!\!\!\approx E_2[P]$ implies that $E_1[P]$ does not raise the event.

Example: Strong secrecy. We illustrate the use of this theorem with an example. Consider the secure channel functionality:

Definition 6.2 (Secure channel)
$\mathcal{F}_{SC} := net_{scstart}().io_A(x).(\overline{net_{notify}}\langle\rangle \mid net_{deliver}().\overline{io_B}\langle x\rangle)$

We want to show:

Lemma 6.3 *If $P \leq \mathcal{F}_{SC}$, then P has strong secrecy in the following sense: We have $P_1 \approx P_2$ where $P_i := \nu io_A io_B.\overline{io_A}\langle n_i\rangle|io_B()|P$.*

Proof. Let $E_i := \nu io_A io_B.\overline{io_A}\langle n_i\rangle|io_B()|\square$. We use Proverif to show that $E_1[\mathcal{F}_{SC}] \approx E_2[\mathcal{F}_{SC}]$. The Proverif code is given in Figure 6. By Theorem 6.1 (and using that $\approx$ and $\approx\!\!\!\!\approx$ coincide for closed processes), we have $P_1 = E_1[\mathcal{F}_{SC}] \approx E_2[\mathcal{F}_{SC}] = P_2$. □

Anonymity properties are modeled very similarly, except that instead of different payloads n_1, n_2, different user ids are provided to the two processes.

Example: Unlinkability. The next example is strong unlinkability [ACRR10]. This property requires that the adversary cannot distinguish whether every user runs only one session of a protocol, or whether every user runs many sessions. Formally: $!\nu id.!\nu sid.P \approx !\nu id.\nu sid.P$ if we assume that P contains free names id, sid for the user id and the session id. At a first glance, such a property seems to be excluded by the restriction of Theorem 6.1 that E_1, E_2 may not have a ! over their hole. This is, however, not the case if protocol P (and the functionality Q) are modeled suitably, namely if P is already a multi-session protocol. For example, if P expects a pair of user id and session id on an IO-channel *init* for each session to be run, then strong unlinkability can be expressed as follows:

Definition 6.4 *A protocol P has* strong unlinkability *iff*

$$\nu init.(P|!\nu id.!\nu sid.\overline{init}\langle(id, sid)\rangle) \approx \nu init.(P|!\nu id.\nu sid.\overline{init}\langle(id, sid)\rangle).$$

Then Theorem 6.1 guarantees that if Q has strong unlinkability and $P \leq Q$, then P has strong unlinkability.

Notice that if we model a different session id mechanism, we also need a different definition. For example, if P and Q are constructed using the !! operator, session ids will be part of the channel name (we would have channels such as $(sid, (id, init))$). The variant described above seems most realistic for unlinkability, though, because !! includes session ids in the clear in all network-messages, so constructing unlinkable protocols by concurrent composition of individual sessions using !! does not seem to work well.

In Appendix A we show that the various restrictions in Theorem 6.1 are necessary. In particular, property preservation for contexts E_1, E_2 having a ! over their hole (instead of a !!) does not hold. The reasons are similar to those that forbid ! in the composition theorem (cf. Section 5). This is another indication that an operator like !! is more natural in this context.

7 Relation to Delaune-Kremer-Pereira

DKP-security. As mentioned in the introduction, Delaune, Kremer, and Pereira [DKP09] have already presented a variant of the UC model in the applied pi calculus. In this section, we describe the differences between their and our model, and why these differences are necessary to achieve stronger security results.

In [DKP09], security is defined as follows:

Definition 7.1 (DKP-security) *Let $\preceq$ (*observational preorder*) be the largest simulation (not bisimulation).*

Let P, Q be processes. Then $P \leq^{\mathsf{SS}} Q$ iff there exists a simulator S (a context) such that $P \preceq S[Q]$.

Here a simulator S is an evaluation context subject to certain conditions, see [DKP09], notably that it only binds NET-names.

Notice that in this definition, the main difference to our definition is that P and $S[Q]$ do not have to be observationally equivalent, but only observationally preordered. (Also, the notion of the simulator S is somewhat different from ours, but not in essence.) The effect of this is that the simulator may introduce additional non-determinism. For example, in our model, if the protocol P can take one out of two actions, the simulator needs to simulate the appropriate action, he thus needs to figure out which of the two actions is taken. With respect to DKP-security, the simulator can just non-deterministically choose which action to take; the observational preorder takes care that the right action is taken in the right situation. This makes simulators for DKP-security much easier to construct and DKP-security into a considerably weaker notion.

DKP-security satisfies similar laws as our notion. In particular, $\leq^{\mathsf{SS}}$ is reflexive and transitive and it satisfies a composition theorem (which differs from ours mainly in that $P \leq^{\mathsf{SS}} Q \Longrightarrow !P \leq^{\mathsf{SS}} !Q$ holds, no need to introduce !!). They do not state a property preservation theorem. We believe, though, that their DPK-security supports property preservation for certain kinds of trace properties.[12]

The problem with observational preorder. We explain why we believe that a definition based on observational preorder instead of equivalence does not give sufficient security guarantees. We illustrate this by the following example. Consider a simple functionality that is supposed to model an insecure but anonymous channel:

$$\mathcal{F}_{anon} := io_A(x).\overline{net}\langle x\rangle | io_B(x).\overline{net}\langle x\rangle$$

Obviously, this functionality preserves anonymity about whether Alice or Bob sends a message (i.e., whether an input on io_A or io_B occurs). Formally: $\nu io_A io_B.(\overline{io_A}\langle T\rangle|\mathcal{F}_{anon}) \approx \nu io_A io_B.(\overline{io_B}\langle T\rangle|\mathcal{F}_{anon})$. (In fact, we even have $\equiv$.) Now consider a naive protocol in which Alice and Bob send their message over distinct channels net_A, net_B. Formally:

$$P := io_A(x).\overline{net_A}\langle x\rangle | io_B(x).\overline{net_B}\langle x\rangle$$

Obviously, P does not provide anonymity, it is easy to see that $\nu io_A io_B.(\overline{io_A}\langle T\rangle|P) \not\approx \nu io_A io_B.(\overline{io_B}\langle T\rangle|P)$. Consequently (Theorem 6.1), we have $P \not\leq \mathcal{F}_{anon}$ as we would expect since P gives less security than $\mathcal{F}_{anon}$.

On the other hand, with respect to DKP-security, P is considered as secure as $\mathcal{F}_{anon}$, i.e., $P \leq^{\mathsf{SS}} \mathcal{F}_{anon}$. We use the following simulator: $S := net(x).\overline{net_A}\langle x\rangle \mid net(x).\overline{net_B}\langle x\rangle \mid \Box$. Then $P \preceq S[\mathcal{F}_{anon}]$ because S relays messages sent on net onto net_A or net_B, and the definition of $\preceq$ makes sure that the message is non-deterministically delivered on the right channel net_A or net_B. Hence $P \leq^{\mathsf{SS}} \mathcal{F}_{anon}$.

Lemma 7.2 (with non-rigorous proof) $P \leq^{\mathsf{SS}} \mathcal{F}_{anon}$.

[12] Probably a law of the following kind holds: Assume $P \leq^{\mathsf{SS}} Q$. Let $c \notin fv(P,Q)$, and E be a context satisfying certain properties. Then $E[Q] \not\downarrow_c \Longrightarrow E[P] \not\downarrow_c$. Compare with Theorem 6.1 which can deal with indistinguishability properties.

Proof. In this proof, we assume that Lemma 3.3 also holds for the calculus from [DKP09]. Since that calculus is somewhat different from ours, this makes the present proof non-rigorous. (However, probably the proof of Lemma 3.3 can be easily adapted to the calculus of [DKP09].)

Then we have

$$\begin{aligned}
P &\overset{(*)}{\approx} \nu net.(io_A(x).\overline{net}\langle x\rangle | net(x).\overline{net_A}\langle x\rangle) \\
&\quad | \nu net.(io_B(x).\overline{net}\langle x\rangle | net(x).\overline{net_B}\langle x\rangle) \\
&\overset{(**)}{\preceq} io_A(x).\overline{net}\langle x\rangle | net(x).\overline{net_A}\langle x\rangle \\
&\quad | io_B(x).\overline{net}\langle x\rangle | net(x).\overline{net_B}\langle x\rangle \\
&\equiv S[\mathcal{F}_{anon}] \text{ with } S := net(x).\overline{net_A}\langle x\rangle \mid net(x).\overline{net_B}\langle x\rangle \mid \Box.
\end{aligned}$$

Here $(*)$ uses two applications of Lemma 3.3 (in the reverse direction), the first with $n := net$, $t := x$, $x := x$, and $Q := \overline{net_A}\langle x\rangle$, and the second with $n := net$, $t := x$, $x := x$, and $Q := \overline{net_B}\langle x\rangle$. And $(**)$ uses that $\nu c.P \preceq P$ ([DKP, Lemma 8]).

Since $\approx$ implies $\preceq$ and $\preceq$ is transitive, we have $P \preceq S[\mathcal{F}_{anon}]$. Furthermore, S is a valid simulator for DKP-security. Thus $P \leq^{\mathsf{SS}} \mathcal{F}_{anon}$. □

Thus, the security of a protocol in the sense of [DKP09] does not imply that the protocol has the same anonymity properties as the ideal functionality. The same probably holds for other equivalence properties such as strong secrecy etc. We consider this a strong restriction of the notion and thus believe that a symbolic analogue to UC security should use observational equivalence or a similar notion of equivalence.

Why observational preorder? The reader may wonder why [DKP09] use observational preorder instead of observational equivalence, in particular since observational equivalence is the more direct analogue to the indistinguishability in the computational UC framework [Can01]. We explain the reasons as we understand them (this is based both on explanations in [DKP09] and on our own insights while working on the current result), and due to what definitional decisions we managed to get around those reasons:

- It is not possible to model "relays". That is, if we have a process P that outputs on a channel c, then as a technical tool we might wish to construct a process R (a relay) that forwards all message on c to another channel c', i.e., we want $\nu c.(P|R) \approx P\{c'/c\}$. Unfortunately, such a relay does not seem to exist in the applied pi calculus. $R :=!c(x).\overline{c'}\langle x\rangle$ does not work. Consider, e.g., $P := \overline{c}\langle n\rangle.\overline{a}\langle n\rangle$. Then $\nu c.(P|R) \downarrow_a$ but $P\{c'/c\} \not\downarrow_a$. With respect to $\preceq$, however, we can have relays $(P\{c'/c\} \preceq \nu c.(P|R))$.

  Why are relays important? One reason is whether a dummy adversary exists. Such a dummy adversary is an adversary that forwards all messages on NET-channels from the protocol to the environment and vice versa. (So, essentially, a relay.) The existence of the dummy adversary is used implicitly or explicitly in most structural theorems (reflexivity, transitivity, concurrent composition). In fact, it seems

that when using observational equivalence in [DKP09], one would not even have reflexivity.

We get around this problem by using a slightly different definition of adversaries/simulators (Definition 4.2). In our setting, a dummy can be trivially constructed as $(0, \varphi, \varnothing)$ where φ just renames the protocol's NET-channels to the NET-channels that the environment expects the messages on. This simple trick obviates the need for using relays in the construction of the dummy adversary.

- The second problem is that one does not get a composition theorem that guarantees $P \leq^{\mathsf{SS}} Q \Longrightarrow !P \leq^{\mathsf{SS}} !Q$ when using observational equivalence. However, we believe that this is a natural limitation because we can show that property preservation does not even hold for equivalence-based security properties that replicate the protocol. Thus we cannot expect to get such a composition theorem and simultaneously have property preservation for equivalence properties. We get around this problem by defining a different notion of concurrent composition, using the !! operator (see Section 5).

- Finally, the non-existence of relays is a problem when proving the security of concrete protocols $P \leq \mathcal{F}$: A typical thing a simulator has to do is to take a message m on a NET-channel and somehow rewrite it (e.g., to $enc(k, m)$) before sending it on to the environment. This, of course, is a generalization of the concept of a relay. Thus, if relays are impossible, we can hardly expect to construct sensible simulators. This, however, is not true if we pay some attention in the definition of the functionality and obey the following guideline:

  > **Guideline:** When designing a functionality, use different names for all NET-channels and, whenever sending something on a NET-channel C, use $\overline{C}\langle T\rangle|P'$ instead of $\overline{C}\langle T\rangle.R$.

  In these cases, $R :=!c(x).\overline{c'}\langle x\rangle$ will usually work as a relay (e.g., $\nu c.(P|R) \approx P\{c'/c\}$ for $P := \overline{c}\langle n\rangle|\overline{a}\langle n\rangle$).

8 Example: Secure channels

In this section we apply symbolic UC hands on. We illustrate how our results from Section 5 can be usefully applied in practice to construct a secure channel from the widely known NSL protocol and a PKI. Furthermore, when extending the secure channel to multiple sessions, we present an example for a joint state, i.e., multiple instances of one protocol that jointly use one instance of another functionality. While the original UC model of Canetti [Can01] requires an additional theorem to handle joint states [CR03], we can directly use !! in our case. We used Proverif[13] for our proofs as much as possible to show how it helps with the verification of various properties in the context of symbolic UC.

[13] Version 1.86pl4

```
fun senc/3. (* senc(key,msg,rand) *)
reduc sdec(k,senc(k,m,r)) = m.
fun empty/0.
fun hash/1.
fun pk/1.
fun sk/1.
fun penc/3. (* penc(pk,msg,rand) *)
reduc pdec(sk(k),penc(pk(k),m,r)) = m.
reduc pkofsk(sk(k)) = pk(k).
reduc pkofenc(penc(p,m,r)) = p.
```

Figure 7: Key-exchange example: Proverif code for the symbolic model (`secchan-model.pv`, see [BU13])

In this section, we only consider an example where we assume all parties to be honest (as the goal of secure channels is to protect from an outside adversary). For an example with corruption, see Section 9.

We first define the symbolic model used in this section. The constructors are: $penc/3$, $pk/1$, $sk/1$, $senc/3$, $(\cdot,\cdot)$, $hash/1$, and $empty/0$, representing public-key encryption, public and secret keys, symmetric encryption, pairs, hashing, and empty messages, respectively. Encryption has a third argument modeling randomness used for encrypting. More specifically, $penc(pk(k), m, r)$ models a public key encryption using key $pk(k)$, plaintext m, and randomness r, and $senc(k, m, r)$ a symmetric encryption using key k, plaintext m, and randomness r. We believe that *senc* without the additional randomness argument r would also work in our setting. However, we introduce this additional nonce to help Proverif, which can then better distinguish ciphertexts (e.g., the proof of `secchan-sc2.pv` fails without r due to Proverif's overapproximation technique). We have no equations in our theory.

Furthermore we have the destructors $pdec/2$, $sdec/2$, $pkofsk/1$, and $pkofenc/1$, modeling public-key decryption, symmetric decryption, extraction of a public key from a secret key, and extraction of a public key from a ciphertext. (The latter two are not needed in our protocols, but we provide them to make the adversary more realistic.) The behavior of the destructors is specified by the following rewrite rules:

$$\begin{aligned} pdec(sk(x), penc(pk(x), y, z)) &\to y \\ sdec(x, senc(x, y, z)) &\to y \\ pkofsk(sk(x)) &\to pk(x) \\ pkofenc(penc(x, y, z)) &\to x \end{aligned}$$

The Proverif code for this symbolic model is given in Figure 7.

8.1 Key exchange using NSL

With the symbolic model set up we next show how to tailor a UC-secure key exchange from NSL using a PKI functionality $\mathcal{F}_{PKI}$. Towards this goal we model the ideal key exchange functionality $\mathcal{F}_{KE}$, the PKI $\mathcal{F}_{PKI}$ and the NSL protocol based on $\mathcal{F}_{PKI}$ as follows:

Definition 8.1 (Key exchange functionality)

$$\mathcal{F}_{KE} := \nu k.net_{delA}().\overline{io_{ka}}\langle k\rangle \mid net_{delB}().\overline{io_{kb}}\langle k\rangle$$

Definition 8.2 (Public key infrastructure functionality)

$$\begin{aligned}\mathcal{F}_{PKI} := \nu k_a k_b.&\overline{io_{pkeA}}\langle(sk(k_a), pk(k_a), pk(k_b))\rangle\\ &\mid \overline{io_{pkeB}}\langle(sk(k_b), pk(k_a), pk(k_b))\rangle\\ &\mid \overline{net_{pke}}\langle(pk(k_a), pk(k_b))\rangle\end{aligned}$$

Definition 8.3 (Needham-Schroeder-Lowe)

$$\begin{aligned}\mathsf{NSL}_A := {} & io_{pkeA}((x_{sk}, \_, x_{pk_B})).\nu n_a.\nu r_1.\\ & \overline{net_{nslA}}\langle penc(x_{pk_B}, n_a, r_1)\rangle.net_{nslA}(x_c).\\ & let\ (=n_a, x_{n_b}, =B) = pdec(x_{sk}, x_c)\ in\\ & \quad \nu r_2.\overline{net_{nslA}}\langle penc(x_{pk_B}, x_{n_b}, r_2)\rangle.\\ & \quad \overline{io_{ka}}\langle hash((n_a, x_{n_b}))\rangle\\ \mathsf{NSL}_B := {} & io_{pkeB}((x_{sk}, x_{pk_A}, \_)).net_{nslB}(x_c).\\ & let\ x_{n_a} = pdec(x_{sk}, x_c)\ in\\ & \quad \nu n_b.\nu r.\overline{net_{nslB}}\langle penc(x_{pk_A}, (x_{n_a}, n_b, B), r)\rangle.\\ & \quad net_{nslB}(x'_c).if\ n_b = pdec(x_{sk}, x'_c)\ then\\ & \quad\ \overline{io_{kb}}\langle hash((x_{n_a}, n_b))\rangle\\ \mathsf{NSL} := {} & \nu io_{pkeA} io_{pkeB}.(\mathsf{NSL}_A \mid \mathsf{NSL}_B \mid \mathcal{F}_{PKI})\end{aligned}$$

The differences to the original NSL protocol [Low95] are: The original protocol includes A's identity in the first message, and the original protocol does not specify what to do with the nonces n_a, n_b, while we use them to derive a key $hash((n_a, n_b))$. Also, [Low95] also presents an extended version of the protocol that explicitly communicates with a server S for getting the keys for Alice and Bob. We could get this extended protocol by proving that this retrieval protocol implements $\mathcal{F}_{PKI}$, and then composing our NSL protocol with the retrieval protocol.

We can now state the first result of this section, namely that the NSL is a UC-secure realization of $\mathcal{F}_{KE}$.

Lemma 8.4 $\mathsf{NSL} \leq \mathcal{F}_{KE}$.

Proof. Let NSL'_A be NSL_A without the initial $io_{pkeA}((x_{sk}, \_, x_{pk_B}))$. NSL'_B analogously. And $\mathsf{NSL}''_A := \mathsf{NSL}'_A\{net_{delA}/io_{ka}, sk(k_a)/x_{sk}, pk(k_b)/x_{pk_B}\}$ and $\mathsf{NSL}''_B := \mathsf{NSL}'_B\{net_{delB}/io_{kb}, sk(k_b)/x_{sk}, pk(k_a)/x_{pk_A}\}$.

We have

$$\begin{aligned}
\mathsf{NSL} &\equiv \nu io_{pkeA} io_{pkeB} k_a k_b.\big(io_{pkeA}((x_{sk}, _, x_{pk_B})).\mathsf{NSL}'_A \mid io_{pkeA}((x_{sk}, x_{pk_A}, _)).\mathsf{NSL}'_B \\
&\qquad \mid \overline{io_{pkeA}}\langle(sk(k_a), pk(k_a), pk(k_b))\rangle \mid \overline{io_{pkeB}}\langle(sk(k_b), pk(k_a), pk(k_b))\rangle \mid \overline{net_{pke}}\langle(pk(k_a), pk(k_b))\rangle\big) \\
&\overset{(v)}{\approx} \nu k_a k_b.\big(\text{let } (x_{sk}, _, x_{pk_B}) = (sk(k_a), pk(k_a), pk(k_b)) \text{ in } \mathsf{NSL}'_A \\
&\qquad \mid \text{let } (x_{sk}, x_{pk_A}, _) = (sk(k_b), pk(k_a), pk(k_b)) \text{ in } \mathsf{NSL}'_B \mid \overline{net_{pke}}\langle(pk(k_a), pk(k_b))\rangle\big) \\
&\overset{(vi)}{\approx} \nu k_a k_b.\big(\mathsf{NSL}'_A\{sk(k_a)/x_{sk}, pk(k_b)/x_{pk_B}\} \mid \mathsf{NSL}'_B\{sk(k_b)/x_{sk}, pk(k_a)/x_{pk_A}\} \\
&\qquad \mid \overline{net_{pke}}\langle(pk(k_a), pk(k_b))\rangle\big) \\
&\overset{(vii)}{\approx} \nu net_{delA} net_{delB} k_a k_b.\big(\mathsf{NSL}''_A \mid \mathsf{NSL}''_B \mid \overline{net_{pke}}\langle(pk(k_a), pk(k_b))\rangle \\
&\qquad \mid net_{delA}(x).\overline{io_{ka}}\langle x\rangle \mid net_{delB}(x).\overline{io_{kb}}\langle x\rangle\big) \\
&\overset{(viii)}{\approx} \nu net_{delA} net_{delB} k_a k_b.\big(\mathsf{NSL}''_A \mid \mathsf{NSL}''_B \mid \overline{net_{pke}}\langle(pk(k_a), pk(k_b))\rangle \\
&\qquad \mid \nu k.(net_{delA}(x).\overline{io_{ka}}\langle x\rangle \mid net_{delB}(x).\overline{io_{kb}}\langle x\rangle)\big) =: \mathsf{NSL}_1
\end{aligned}$$

Here (v) uses two consecutive applications of Lemma 3.3, the first with $n := io_{pkeA}$ and $C := \Box$ and $t := (sk(k_a), pk(k_a), pk(k_b))$, and the second with $n := io_{pkeB}$ and $C := \Box$ and $t := (sk(k_b), pk(k_a), pk(k_b))$. Remember also that $io_{pkeA}((x_{sk}, \_, x_{pk_B}))$ is syntactic sugar for $io_{pkeA}(x).\text{let } (x_{sk}, \_, x_{pk_B}) = x$.

And (vi) uses two consecutive applications of Lemma 3.2 (v) and the fact that $\approx$ is closed under evaluation contexts.

And (vii) uses two applications of Lemma 3.3 (both in the opposite direction), the first with $n := net_{delA}$, $Q := \overline{io_{ka}}\langle x\rangle$, and $t := H((n_a, x_{n_b}))$, and the second with $n := net_{delB}$, $Q := \overline{io_{kb}}\langle x\rangle$, and $t := H((x_{n_a}, n_b))$.

And (viii) uses Lemma 3.2 (i) to add νk.

Using Proverif, we can show the following observational equivalence:

$$\begin{aligned}
\mathsf{NSL}_1 &\overset{(*)}{\approx} \nu net_{delA} net_{delB} k_a k_b.(\mathsf{NSL}''_A \mid \mathsf{NSL}''_B \mid \overline{net_{pke}}\langle(pk(k_a), pk(k_b))\rangle \mid \mathcal{F}_{KE}) \\
&\equiv \nu net_{delA} net_{delB}.(\mathcal{F}_{KE} | S)
\end{aligned}$$

for $S := \nu k_a k_b.(\mathsf{NSL}''_A | \mathsf{NSL}''_B | \overline{net_{pke}}\langle(pk(k_a), pk(k_b))\rangle)$. The Proverif code for checking $(*)$ is given in Figure 8.

Hence $\mathsf{NSL} \leq \mathcal{F}_{KE}$. □

8.2 Secure channel from key exchange.

Next, we realize a secure channel. Since we already have a realization of a secure key exchange at hand, we realize the secure channel SC from the idealized key exchange $\mathcal{F}_{KE}$. Later we replace $\mathcal{F}_{KE}$ by NSL. We model $\mathcal{F}_{SC}$ and SC based on $\mathcal{F}_{KE}$ as follows:

```
free B, netnsla, netnslb, netpke.
free ioka, iokb.

let A =
    new na;
    new r1;
    out(netnsla,penc(pk(kb),na,r1));
    in(netnsla,xc);
    let (=na,xnb,=B) = pdec(sk(ka),xc) in
    new r2;
    out(netnsla,penc(pk(kb),xnb,r2));
    out(netdela,hash((na,xnb))).

let B =
    in(netnslb,xc);
    let xna = pdec(sk(kb),xc) in
    new nb;
    new r;
    out(netnslb,penc(pk(ka),(xna,nb,B),r));
    in(netnslb,xc2);
    if nb = pdec(sk(kb),xc2) then
    out(netdelb,hash((xna,nb))).

let KE =
    new k;
    (in(netdela,x);out(ioka,choice[x,k])) |
    (in(netdelb,x);out(iokb,choice[x,k])).

process
    new netdela; new netdelb;
    new ka; new kb; (A | B | out(netpke,(pk(ka),pk(kb))) | KE)
```

Figure 8: Key-exchange example: Proverif code for analyzing NSL (`secchan-nsl.pv`, see [BU13]). (Has to be prefixed with the code from Figure 7.)

Definition 8.5 (Secure channel) [14]

$$\mathcal{F}_{SC} := net_{scstart}().io_A(x).(\overline{net_{notify}}\langle\rangle \mid net_{deliver}().\overline{io_B}\langle x\rangle)$$

Definition 8.6 (Secure channel protocol)

$$\begin{aligned}\mathsf{SC}_A &:= io_{ka}(x_k).io_A(x_m).\nu r.\overline{net_A}\langle senc(x_k, x_m, r)\rangle\\ \mathsf{SC}_B &:= io_{kb}(x_k).net_B(x_c).let\ x_m = sdec(x_k, x_c)\ in\ \overline{io_B}\langle x_m\rangle\\ \mathsf{SC} &:= \nu io_{ka} io_{kb}.(\mathsf{SC}_A|\mathsf{SC}_B|\mathcal{F}_{KE})\end{aligned}$$

Lemma 8.7 $\mathsf{SC} \leq \mathcal{F}_{SC}$.

Proof. We have:

$$\begin{aligned}\mathsf{SC} \equiv\ & \nu io_{ka} io_{kb} k.\big(io_{ka}(x_k).io_A(x_m).\nu r.\overline{net_A}\langle senc(x_k, x_m, r)\rangle \mid io_{kb}(x_k).net_B(x_c).\\ & \quad \text{let } x_m = sdec(x_k, x_c) \text{ in } \overline{io_B}\langle x_m\rangle \mid net_{delA}().\overline{io_{ka}}\langle k\rangle \mid net_{delB}().\overline{io_{kb}}\langle k\rangle\big)\\ \overset{(*)}{\approx}\ & \nu k.\big(net_{delA}().io_A(x_m).\nu r.\overline{net_A}\langle senc(k, x_m, r)\rangle \mid net_{delB}().net_B(x_c).\\ & \quad \text{let } x_m = sdec(k, x_c) \text{ in } \overline{io_B}\langle x_m\rangle\big) =: \mathsf{SC}_1\end{aligned}$$

Here $(*)$ uses two consecutive applications of Lemma 3.3, the first with $n := io_{ka}$ and $C := net_{delA}().\Box$ and $t := k$, and the second with $n := io_{kb}$ and $C := net_{delB}().\Box$ and $t := k$. (And it uses Lemma 2.7, so that we can apply Lemma 3.3 to a subprocess instead of the whole process.)

We show next:

$$\begin{aligned}\mathsf{SC}_1 \approx\ & \nu s\, k.\big(net_{delA}().io_A(x_m).\nu r.(!\overline{(s, senc(k, x_m, r))}\langle x_m\rangle \mid \overline{net_A}\langle senc(k, x_m, r)\rangle) \mid\\ & \quad net_{delB}().net_B(x_c).\text{let } x_m = sdec(k, x_c) \text{ in } (s, x_c)(x'_m).\overline{io_B}\langle x_m\rangle\big) =: \mathsf{SC}_2\end{aligned}$$

By Lemma 3.7, to show the above it is sufficient to show that the trace property $end() \Rightarrow start()$ holds in the following event process:

$$\begin{aligned}&\nu k.\big(net_{delA}().io_A(x_m).\nu r.event\ start(senc(k, x_m, r)).\overline{net_A}\langle senc(k, x_m, r)\rangle \mid\\ &\quad net_{delB}().net_B(x_c).\text{let } x_m = sdec(k, x_c) \text{ in } event\ end(x_c).\overline{io_B}\langle x_m\rangle\big).\end{aligned}$$

We show this trace property using Proverif, the required code is given in Figure 9.

Note: We could also have shown an analogous observational equivalence with s instead of $(s, senc(k, x_m, r))$. Then, however, Proverif fails on the code given in Figure 10 because it does not see there is only one message x_m sent over the channel. Thus, it believes that different x_m could be confused. Adding x_c to the channel name helps Proverif to see that x_m is unique (since x_c already determines x_m).

Since we send the message x_m directly to Bob via the channel $(s, \cdot)$ (who receives it as x'_m), we can let Bob output the message x'_m received over that channel instead of

[14] This definition was already given in Section 6 (Definition 6.2) and is repeated here for convenience.

```
free ioa. (* A-input of F_SC *)
free iob. (* B-output of F_SC *)
free neta. (* A-end of insecure channel in P_SC *)
free netb. (* B-end of insecure channel in P_SC *)
free netdela, netdelb.

query ev:end(x) ==> ev:start(x).

let PA =
    in(netdela,x);
    in(ioa,xm);
    new r;
    event start(senc(k,xm,r));
    out(neta,senc(k,xm,r)).

let PB =
    in(netdelb,x);
    in(netb,xc);
    let xm=sdec(k,xc) in
    event end(xc);
    out(iob,xm).

process
    new k;
    PA | PB
```

Figure 9: Key-exchange example: Proverif code for analyzing the trace property of SC (`secchan-sc1.pv`, see [BU13]). (Has to be prefixed with the code from Figure 7.)

```
free ioa. (* A-input of F_SC *)
free iob. (* B-output of F_SC *)
free neta. (* A-end of insecure channel in P_SC *)
free netb. (* B-end of insecure channel in P_SC *)
free netdela, netdelb.

let PA =
    in(netdela,x);
    in(ioa,xm);
    new r;
    (!out((s,senc(k,choice[xm,empty],r)),xm)) |
    out(neta,senc(k,choice[xm,empty],r)).

let PB =
    in(netdelb,x);
    in(netb,xc);
    let xm=sdec(k,xc) in
    in((s,xc),xm2);
    out(iob,choice[xm,xm2]).

process
    new s;
    new k;
    PA | PB
```

Figure 10: Key-exchange example: Proverif code for analyzing the observation equivalence in SC (`secchan-sc2.pv`, see [BU13]). (Has to be prefixed with the code from Figure 7.)

using the decrypted value x_m. Since then the plaintext of the ciphertext x_c is then not used any more, we can encrypt *empty* instead of x_m (as the adversary cannot tell the difference). Formally, we show the following observational equivalence:

$$\mathsf{SC}_2 \approx \nu s\, k.(net_{delA}().io_A(x_m).\nu r.(!\overline{(s, senc(k, empty, r))}\langle x_m\rangle \mid \overline{net_A}\langle senc(k, empty, r)\rangle) \mid net_{delB}().net_B(x_c).\text{let } x_m = sdec(k, x_c) \text{ in } (s, x_c)(x'_m).\overline{io_B}\langle x'_m\rangle) =: \mathsf{SC}_3.$$

We show this observational equivalence using Proverif, the required code is given in Figure 10.

Then we move the restriction νr to the top and replace the channel $(s, senc(k, empty, r))$ by s:

$$\mathsf{SC}_3 \overset{(*)}{\approx} \nu s\, k\, r.(net_{delA}().io_A(x_m).(!\overline{(s, senc(k, empty, r))}\langle x_m\rangle \mid \overline{net_A}\langle senc(k, empty, r)\rangle) \mid net_{delB}().net_B(x_c).\text{let } x_m = sdec(k, x_c) \text{ in } (s, x_c)(x'_m).\overline{io_B}\langle x'_m\rangle)$$

```
free ioa. (* A-input of F_SC *)
free iob. (* B-output of F_SC *)
free neta. (* A-end of insecure channel in P_SC *)
free netb. (* B-end of insecure channel in P_SC *)
free netdela, netdelb.

let PA =
    in(netdela,x);
    in(ioa,xm);
    (!out(choice[(s,senc(k,empty,r)),s],xm)) |
    out(neta,senc(k,empty,r)).

let PB =
    in(netdelb,x);
    in(netb,xc);
    let xm=sdec(k,xc) in
    in(choice[(s,xc),s],xm2);
    out(iob,xm2).

process
    new s;
    new k;
    new r;
    PA | PB
```

Figure 11: Key-exchange example: Proverif code for analyzing the second observation equivalence in SC (`secchan-sc3.pv`, see [BU13]). (Has to be prefixed with the code from Figure 7.)

$$\overset{(**)}{\approx} \nu s\, k\, r.(net_{delA}().io_A(x_m).(!\overline{s}\langle x_m\rangle \mid \overline{net_A}\langle senc(k, empty, r)\rangle) \mid$$
$$net_{delB}().net_B(x_c).\text{let } x_m = sdec(k, x_c) \text{ in } s(x'_m).\overline{io_B}\langle x'_m\rangle) =: \mathsf{SC}_4$$

Here $(*)$ follows from Lemma 3.2 (ii), and $(**)$ is proven using Proverif. The required code is given in Figure 11.

We continue:

$$\mathsf{SC}_4 \overset{(*)}{\approx} \nu net_{deliver}\, k\, r.(net_{delA}().io_A(x_m).(net_{deliver}().\overline{io_B}\langle x_m\rangle \mid \overline{net_A}\langle senc(k, empty, r)\rangle) \mid$$
$$net_{delB}().net_B(x_c).\text{let } x_m = sdec(k, x_c) \text{ in } \overline{net_{deliver}}\langle\rangle)$$
$$\overset{(**)}{\approx} \nu net_{deliver}\, k\, r\, net_{notify}.(net_{delA}().io_A(x_m).(net_{deliver}().\overline{io_B}\langle x_m\rangle \mid \overline{net_{notify}}\langle\rangle) \mid$$
$$net_{delB}().net_B(x_c).\text{let } x_m = sdec(k, x_c) \text{ in } \overline{net_{deliver}}\langle\rangle \mid net_{notify}().\overline{net_A}\langle senc(k, empty, r)\rangle)$$
$$\equiv \nu net_{deliver}\, net_{notify}.(\mathcal{F}_{SC}\{net_{delA}/net_{scstart}\}|S)$$
$$\text{with } S := \nu k r.net_{delB}().net_B(x_c).\text{let } x_m = sdec(k, x_c) \text{ in } \overline{net_{deliver}}\langle\rangle$$

$$\mid net_{notify}().\overline{net_A}\langle senc(k, empty, r)\rangle$$

Here $(*)$ uses Lemma 3.4 with $Q := \overline{io_B}\langle x'_m\rangle$, $x := x'_m$, $n := s$, and $m := net_{deliver}$.

And $(**)$ uses Lemma 3.3 with $Q := \overline{net_A}\langle senc(k, empty, r)\rangle$, $n := net_{notify}$, $t := empty$.

So $\mathsf{SC} \approx \nu\underline{n}.(\mathcal{F}_{SC}\sigma|S)$ for $\sigma := \{net_{delA}/net_{scstart}\}$ and $\underline{n} := net_{deliver}\, net_{notify}$. Hence $\mathsf{SC} \leq \mathcal{F}_{SC}$. □

With $\mathsf{NSL} \leq \mathcal{F}_{KE}$ (Lemma 8.4) and $\mathsf{SC} \leq \mathcal{F}_{SC}$ (Lemma 8.7) at hand we can now use the compositional capabilities of UC: We define an evaluation context $\mathcal{C}[\Box] := \nu io_{ka} io_{kb}.(\mathsf{SC}_A|\mathsf{SC}_B|\Box)$ where SC_A and SC_B are the processes from Definition 8.6. Since $\mathcal{C}$ meets the requirements of Theorem 5.37 $\mathsf{NSL} \leq \mathcal{F}_{KE}$ implies $\mathcal{C}[\mathsf{NSL}] \leq \mathcal{C}[\mathcal{F}_{KE}]$. Since $\mathcal{C}[\mathcal{F}_{KE}] = \mathsf{SC}$ and $\mathsf{SC} \leq \mathcal{F}_{SC}$ we have, by transitivity of $\leq$ (Lemma 4.5), $\mathcal{C}[\mathsf{NSL}] \leq \mathcal{F}_{SC}$.

We did construct a secure channel from a PKI using the NSL protocol. More interesting than this result is the way we achieved it: We did not have to analyze the complete system $\mathcal{C}[\mathsf{NSL}]$ in one piece but could replace the NSL protocol with an idealized functionality. This illustrates two striking advantages of the UC approach:

- The fact that NSL realizes an ideal key exchange can be re-used for security proofs of further systems.
- We cannot only plug NSL into $\mathcal{C}$ but any protocol that realizes a secure key exchange (e.g., if no PKI is available and thus NSL is not an option).

Instead of one monolithic security proof for $\mathcal{C}[\mathsf{NSL}]$ we end up with smaller proofs and results which can be used flexibly. Furthermore, to split the security analysis of a complex system into smaller parts might be the only feasible option to tackle it at all.

8.3 Generating many keys from one

While the example until now illustrates composition and the power of UC, $\mathcal{C}[\mathsf{NSL}]$ only realizes a single-use secure channel. To transfer multiple messages, we could just use concurrent composition to have $!!\mathcal{C}[\mathsf{NSL}] \leq !!\mathcal{F}_{SC}$. However, the resulting protocol uses one instance of NSL per message, and – since NSL contains $\mathcal{F}_{PKI}$, *another* PKI for each message that is sent. This is clearly unrealistic. To get rid of this overhead we want to have all the instances of SC to jointly use just one key exchange $\mathcal{F}_{KE}$, i.e., we want to use the previously mentions joined state technique here. Towards this goal we model a wrapper protocol KE^* which uses one key exchange to emulate multiple key exchanges (from a key k it derives session keys $hash((sid, k))$ where sid is the session id). Formally, we define KE^* as follows and then show $\mathsf{KE}^* \leq !!\mathcal{F}_{KE}$.

Definition 8.8

$$\begin{aligned}
\mathsf{KE}^*_A &:= io'_{ka}(x_k).!!_{x_{sid}}\overline{io_{ka}}\langle hash((x_{sid}, x_k))\rangle\\
\mathsf{KE}^*_B &:= io'_{kb}(x_k).!!_{x_{sid}}\overline{io_{kb}}\langle hash((x_{sid}, x_k))\rangle\\
\mathsf{KE}^* &:= \nu io'_{ka} io'_{kb}.(\mathsf{KE}^*_A \mid \mathsf{KE}^*_B \mid \mathcal{F}'_{KE})
\end{aligned}$$

where $\mathcal{F}'_{KE} := \mathcal{F}_{KE}\{io'_{ka}/io_{ka}, io'_{kb}/io_{kb}\}$.

Lemma 8.9 $\mathsf{KE}^* \leq !!\mathcal{F}_{KE}$.

Proof. Let $S := net_{delA}().!!\overline{net'_{delA}}\langle\rangle \mid net_{delB}().!!\overline{net'_{delB}}\langle\rangle$. Here we use the shorthand $\overline{t}\langle\rangle$ for $\overline{t}\langle empty\rangle$. Let $\underline{n} := net'_{delA}net'_{delB}$. Let $\sigma := \{net'_{delA}/net_{delA}, net'_{delB}/net_{delB}\}$.

We have

$$\begin{aligned}
\mathsf{KE}^* &\overset{(i)}{\approx} \nu k.net_{delA}().!!_{x_{sid}}\overline{io_{ka}}\langle hash((x_{sid},k))\rangle \mid net_{delB}().!!_{x_{sid}}\overline{io_{kb}}\langle hash((x_{sid},k))\rangle \\
&\overset{(ii)}{\approx} \nu k.net_{delA}().!!_{x_{sid}}\nu net'_{delA}.(\overline{net'_{delA}}\langle\rangle \mid net'_{delA}().\overline{io_{ka}}\langle hash((x_{sid},k))\rangle) \\
&\quad \mid net_{delB}().!!_{x_{sid}}\nu net'_{delB}.(\overline{net'_{delB}}\langle\rangle \mid net'_{delB}().\overline{io_{kb}}\langle hash((x_{sid},k))\rangle) \\
&\overset{(iii)}{\approx} \nu k.\nu net'_{delA}.net_{delA}().(!!_{x_{sid}}\overline{net'_{delA}}\langle\rangle \mid !!_{x_{sid}}net'_{delA}().\overline{io_{ka}}\langle hash((x_{sid},k))\rangle) \\
&\quad \mid \nu net'_{delB}.net_{delB}().(!!_{x_{sid}}\overline{net'_{delB}}\langle\rangle \mid !!_{x_{sid}}net'_{delB}().\overline{io_{kb}}\langle hash((x_{sid},k))\rangle) \\
&\overset{(iv)}{\approx} \nu k.\nu net'_{delA}.(net_{delA}().!!_{x_{sid}}\overline{net'_{delA}}\langle\rangle \mid !!_{x_{sid}}net'_{delA}().\overline{io_{ka}}\langle hash((x_{sid},k))\rangle) \\
&\quad \mid \nu net'_{delB}.(net_{delB}().!!_{x_{sid}}\overline{net'_{delB}}\langle\rangle \mid !!_{x_{sid}}net'_{delB}().\overline{io_{kb}}\langle hash((x_{sid},k))\rangle) \\
&\overset{(v)}{\approx} \nu\underline{n}.\big(\nu k.!!_{x_{sid}}\big(net'_{delA}().\overline{io_{ka}}\langle hash((x_{sid},k))\rangle \mid net'_{delB}().\overline{io_{kb}}\langle hash((x_{sid},k))\rangle\big) \mid S\big) \\
&\overset{(vi)}{\approx} \nu\underline{n}.(!!_{x_{sid}}\nu k.(net'_{delA}().\overline{io_{ka}}\langle k\rangle \mid net'_{delB}().\overline{io_{kb}}\langle k\rangle) \mid S) \\
&= \nu\underline{n}.(!!\mathcal{F}_{KE}\,\sigma \mid S)
\end{aligned}$$

Here (i) uses two application of Lemma 3.3, the first with $C := net_{delA}().\Box$, $n := io'_{ka}$, and $t := k$, the second with $C := net_{delB}().\Box$, $n := io'_{kb}$, and $t := k$. (And it uses Lemma 2.7, so that we can apply Lemma 3.3 to a subprocess instead of the whole process.)

And (ii) uses Lemma 3.3 with $C := \Box$ to show $\overline{io_{ka}}\langle hash((x_{sid},k))\rangle \approx\!\!\!\approx \nu net'_{delA}.(\overline{net'_{delA}}\langle\rangle \mid net'_{delA}().\overline{io_{ka}}\langle hash((x_{sid},k))\rangle$ and $\overline{io_{kb}}\langle hash((x_{sid},k))\rangle \approx\!\!\!\approx \nu net'_{delB}.(\overline{net'_{delB}}\langle\rangle \mid net'_{delB}().\overline{io_{kb}}\langle hash((x_{sid},k))\rangle$.

And (iii) uses Lemma 3.2 (ii) and Lemma 5.36 and Lemma 5.35.

And (iv) uses the following claim (proven below) twice. First with $n := net'_{delA}$, $m := net_{delA}$, $Q := \overline{io_{ka}}\langle hash((x_{sid},k))\rangle$. Then with $n := net'_{delB}$, $m := net_{delB}$, $Q := \overline{io_{kb}}\langle hash((x_{sid},k))\rangle$.

Claim 4 *For names n, m, and for any process Q, we have $\nu n.m().(!!_x\overline{n}\langle\rangle \mid !!_x n().Q) \approx \nu n.((m().!!_x\overline{n}\langle\rangle) \mid !!_x n().Q)$.*

(Intuitively, this claim holds because $!!_x n().Q$ cannot perform any observable actions until $!!_x\overline{n}\langle\rangle$ is executed. So it makes no difference whether both $!!_x n().Q$ and $!!_x\overline{n}\langle\rangle$ wait for the input on m to occur, or whether only $!!_x n().Q$ waits for it.)

And (v) follows from the definition of $\equiv$ and Lemma 5.36.

Finally, (vi) follows from the following claim (proven below):

Claim 5 *For any process P, we have $\nu k.!!_x P\{hash((x,k))/k\} \approx !!_x \nu k.P$.*

Thus we have derived $\mathsf{KE}^* \approx \nu \underline{n}.(!!\mathcal{F}_{KE}\,\sigma \mid S)$. This shows $\mathsf{KE}^* \leq !!\mathcal{F}_{KE}$. It remains to show the two claims.

To show Claim 4, consider the following relation:

$$\mathcal{R} := \Big\{ E[\nu n.m().(\prod_{x\in SID} \overline{n}\langle\rangle \mid \prod_{x\in SID} n().Q(\!(x)\!))], \\ E[\nu n.(m().\prod_{x\in SID} \overline{n}\langle\rangle \mid \prod_{x\in SID\setminus S} n().Q(\!(x)\!) \mid \sum_{x\in S} n().Q(\!(x)\!))] \Big\} \cup \approx$$

up to structural equivalence. Here E ranges over evaluation contexts, and S over finite subsets of SID. n, m, Q are from the statement of the lemma. $\sum_{x\in S} P$ stands short for $P\{s_1/x\}|\ldots|P\{s_k/x\}$ with $S =: \{s_1, \ldots, s_k\}$. I.e., $\sum_{x\in S}$ is almost the same as $\prod_{x\in S}$, except that $\sum_{x\in S}$ is syntactic sugar (and only makes sense for finite S) while $\prod_{x\in S}$ is a proper construct in the syntax of product processes.

We show that $\mathcal{R}$ is a bisimulation:

- If $(A, B) \in \mathcal{R}$ and $A \downarrow_M$, then $B \downarrow_M$:

  In the case $A \approx B$, the statement is immediate. We can thus assume $A \equiv E[\nu n.m().(\prod_{x\in SID} \overline{n}\langle\rangle \mid \prod_{x\in SID} n().Q(\!(x)\!))]$ and $B \equiv E[\nu n.(m().\prod_{x\in SID} \overline{n}\langle\rangle \mid \prod_{x\in SID\setminus S} n().Q(\!(x)\!) \mid \sum_{x\in S} n().Q(\!(x)\!))]$.

  In the argument to E, there are no unprotected outputs. Thus the output on M is in E and thus $B \downarrow_M$ trivially follows.

- If $(A, B) \in \mathcal{R}$ and $B \downarrow_M$, then $A \downarrow_M$: Analogous to the previous case.

- If $(A, B) \in \mathcal{R}$ and $A \to A'$, then there is a B' with $B \to^* B'$ and $(A', B') \in \mathcal{R}$:

  In the case $A \approx B$, the statement is immediate. We can thus assume $A \equiv E[\nu n.m().(\prod_{x\in SID} \overline{n}\langle\rangle \mid \prod_{x\in SID} n().Q(\!(x)\!))]$ and $B \equiv E[\nu n.(m().\prod_{x\in SID} \overline{n}\langle\rangle \mid \prod_{x\in SID\setminus S} n().Q(\!(x)\!) \mid \sum_{x\in S} n().Q(\!(x)\!))]$.

  If $A \to A'$ is a reduction within E, then let $B \to B'$ be the corresponding reduction, and then $(A', B') \in \mathcal{R}$.

  Otherwise, $A \to A'$ is a communication on m between E and the input $m()$ in its argument, hence $A' \equiv E'[\nu n.(\prod_{x\in SID} \overline{n}\langle\rangle \mid \prod_{x\in SID} n().Q(\!(x)\!))]$. And $B \to B' := E'[\nu n.(\prod_{x\in SID} \overline{n}\langle\rangle \mid \prod_{x\in SID\setminus S} n().Q(\!(x)\!) \mid \sum_{x\in S} n().Q(\!(x)\!))]$.

  From Lemma 3.2 (ix), we have $A' \approx B'$, hence $(A', B') \in \mathcal{R}$.

- If $(A, B) \in \mathcal{R}$ and $B \to B'$, then there is a A' with $A \to^* A'$ and $(A', B') \in \mathcal{R}$:

  In the case $A \approx B$, the statement is immediate. We can thus assume $A \equiv E[\nu n.m().(\prod_{x\in SID} \overline{n}\langle\rangle \mid \prod_{x\in SID} n().Q(\!(x)\!))]$ and $B \equiv E[\nu n.(m().\prod_{x\in SID} \overline{n}\langle\rangle \mid \prod_{x\in SID\setminus S} n().Q(\!(x)\!) \mid \sum_{x\in S} n().Q(\!(x)\!))]$.

  If $B \to B'$ is a reduction within E, or if $B \to B'$ is a communication on m between E and $m()$ in its argument, then the reasoning is as in the previous case.

Otherwise, we have that $B \to B'$ is a reduction of the second product, i.e. $B' \equiv E[\nu n.(m().\prod_{x\in SID}\overline{n}\langle\rangle \mid \prod_{x\in SID\setminus S'} n().Q(\!(x)\!) \mid \sum_{x\in S'} n().Q(\!(x)\!))]$ with $S' := S \setminus \{t\}$ for some $t \in SID \setminus S$. Then $(A', B') \in \mathcal{R}$ with $A' := A$.

- If $(A, B) \in \mathcal{R}$, then $(E[A], E[B]) \in \mathcal{R}$:

  Immediate from the definition of $\mathcal{R}$.

The statement of the claim is equivalent to

$$P_1 := \nu n.m().(\prod_{x\in SID}\overline{n}\langle\rangle \mid \prod_{x\in SID} n().Q(\!(x)\!)) \approx \nu n.((m().\prod_{x\in SID}\overline{n}\langle\rangle) \mid \prod_{x\in SID} n().Q(\!(x)\!) =: P_2.$$

And this follows from the fact that $\mathcal{R}$ is a bisimulation since $(P_1, P_2) \in \mathcal{R}$. Thus Claim 4 is shown.

To show Claim 5, consider the following relation:

$$\mathcal{R} := \left\{ \left(\nu \underline{n} k.Q\sigma \mid \prod_{x\in S} P\{hash((x,k))/k\}, \quad \nu \underline{n}\, \underline{k}_\sigma.Q \mid \prod_{x\in S} \nu k.P\right)\right\}$$

up to structural equivalence. Here $k \notin fn(S)$ is an arbitrary name, $S \subseteq SID$ is a set of terms, σ is a (finite) substitution mapping names to distinct (with respect to $=_{\mathrm{E}}$) terms $hash((t,k))$ with $t \in SID \setminus S$, $\underline{k}_\sigma = \operatorname{dom}\sigma$, $\underline{k}_\sigma \cap fn(P,S) = \varnothing$, $\underline{n}$ is a list of names, and Q is an arbitrary process with $k \notin fn(Q)$.

We show that $\mathcal{R}$ is a bisimulation:

- If $(A, B) \in \mathcal{R}$ and $A \downarrow_M$ then $B \downarrow_M$:

  Since k and $\underline{k}_\sigma$ are bound names, we have that M does not contain either of them. But only terms containing k or k_S are different in A and B. Thus $B \downarrow_M$.

- If $(A, B) \in \mathcal{R}$ and $B \downarrow_M$ then $A \downarrow_M$:

  Analogous.

- If $(A, B) \in \mathcal{R}$ and $A \to A'$, then there is a B' with $B \to^* B'$ and $(A', B') \in \mathcal{R}$:

  If the reduction is $\prod_{x\in S} P\{hash((x,k))/k\} \to P\{hash((t,k))/k, t/x\} \mid \prod_{x\in S'} P\{hash((x,k))/k\}$ with $S' := S \setminus \{t\}$, then we have $B \to^* B'$ and $(A', B') \in \mathcal{R}$ with $B' := \nu \underline{n} \underline{k}_{\sigma'}.Q \mid P\{k_t/k, t/x\} \mid \prod_{x\in S'} \nu k.P$ and $\sigma' := \sigma \cup \{k_t \mapsto H((t,k))\}$ for some fresh name k_t. Notice that the terms in the range of σ' are still distinct because $S \subseteq SID$ contains only distinct terms, and $t \in SID \setminus S$.

  If the reduction is a reduction of $Q\sigma \to Q'$, then it is easy to see (by checking, in particular, for all destructors that $f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma)$) that $Q \to Q'\sigma^{-1}$. From this it follows that $B \to^* B'$ and $(A, B) \in \mathcal{R}$ with $B' := \nu \underline{n}\, \underline{k}_\sigma.Q'\sigma^{-1} \mid \prod_{x\in S} \nu k.P$.

- If $(A,B) \in \mathcal{R}$ and $B \to B'$, then there is a A' with $A \to^* A'$ and $(A',B') \in \mathcal{R}$:

  If the reduction is $\prod_{x\in S} \nu k.P \to \nu k.P\{t/x\} \mid \prod_{x\in S'} \nu P$ with $S' := S \setminus \{t\}$, then we have $(A',B') \in \mathcal{R}$ with $A' := \nu \underline{n} k.(Q \mid P\{H((t,k))/k\})\sigma \mid \prod_{x\in S'} P\{H((x,k))/k\}$ and $B' \equiv \nu \underline{n} \underline{k}_{\sigma'}.Q \mid P\{k_t/k\} \mid \prod_{x\in S'} \nu k.P$ and $\sigma' := \sigma \cup \{k_t \mapsto H((t,k))\}$ and some fresh name k_t. Notice that the terms in the range of σ' are still distinct because $S \subseteq SID$ contains only distinct terms, and $t \in SID \setminus S$.

  If the reduction is a reduction of $Q \to Q'$, then it is easy to see (by checking, in particular, for all destructors that $f(t_1,\ldots,t_n)\sigma = f(t_1\sigma,\ldots,t_n\sigma)$) that $Q\sigma \to Q'\sigma$. From this it follows that $(A,B) \in \mathcal{R}$ with $A' := \nu \underline{n} k.Q'\sigma \mid \prod_{x\in S} P\{hash((x,k))/k\}$.

- If $(A,B) \in \mathcal{R}$ and E is an evaluation context, then $(E[A],E[B]) \in \mathcal{R}$:

  Then $A = \nu \underline{n} k.Q\sigma \mid \prod_{x\in S} P\{hash((x,k))/k\}$ and $B = \nu \underline{n}\, \underline{k}_\sigma.Q \mid \prod_{x\in S} \nu k.P$. Without loss of generality, $k, \underline{k}_\sigma \notin fn(E) \cup fn(E)$. (Otherwise we could replace $k, \underline{k}_\sigma$ by other names in A,B.) There is a process Q' and a list of names $\underline{n}'$ such that $E[P] \equiv \nu \underline{n}'.(P|Q')$ for all P. Then

$$(E[A],E[B]) \equiv \Big(\nu \underline{n}'\, \underline{n} k.(Q|Q')\sigma \mid \prod_{x\in S} P\{hash((x,k))/k\},\quad \nu \underline{n}'\, \underline{n}\, \underline{k}_\sigma.(Q|Q') \mid \prod_{x\in S} \nu k.P\Big) \in \mathcal{R}.$$

Since $(\nu k.\prod_{x\in SID} P\{hash((x,k))/k\}, \prod_{x\in SID} \nu k.P) \in \mathcal{R}$, we have $\nu k.!!_x P\{hash((x,k))/k\} \approx \nu k.\prod_{x\in SID} P(\!(x)\!)\{hash((x,k))/k\} \approx \prod_{x\in SID} \nu k.P(\!(x)\!) \approx !!\nu k.P$. This shows Claim 5. □

Analogously to the single session case we define a suitable context $\mathcal{C}^*$ by replacing $\mathcal{F}'_{KE}$ in KE^* with $\Box$ and have

$$\mathcal{C}^*[\mathsf{NSL}] \le \mathcal{C}^*[\mathcal{F}'_{KE}] = \mathsf{KE}^* \le !!\mathcal{F}_{KE}$$

Furthermore, $!!\mathsf{SC} \approx \nu io_{ka} io_{kb}.(!!\mathsf{SC}_A|!!\mathsf{SC}_B|!!\mathcal{F}_{KE})$ (by Lemmas 5.35,5.36). Hence

$$\begin{aligned} &\nu io_{ka} io_{kb}.(!!\mathsf{SC}_A|!!\mathsf{SC}_B|\mathcal{C}^*[\mathsf{NSL}]) \\ \le\ &\nu io_{ka} io_{kb}.(!!\mathsf{SC}_A|!!\mathsf{SC}_B|!!\mathcal{F}_{KE}) \\ \le\ &!!\mathsf{SC} \le !!\mathcal{F}_{SC}. \end{aligned}$$

Finally, we have a protocol which realizes multiple secure channels while invoking the NSL protocol and using only one PKI.

9 Virtual primitives

In this section, we present a technique for deriving security of protocols in the symbolic UC model that is specific to the symbolic model. No analogue in the computational world seems to exist. The idea is the following: When constructing UC secure protocols, it is often necessary to include specific "trapdoors" that allow the simulator to extract or modify certain information. For example, when constructing a simulator for

a commitment scheme, we need to include in the protocol some way for the simulator to extract the value of the commitment when given a commitment by the environment (*extractability*), or to change the content of a commitment when producing a commitment for the environment (*equivocality*), see [CF01]. These additional trapdoors often make the protocols more complex, and they often also need more complex cryptographic primitives. A simple commitment protocol in which the committer just sends $hash(m, r)$ for message m and randomness r is not UC secure because the simulator cannot extract or equivoke. Instead, one would need to assume a special hash function that takes an additional parameter *crs* (the common reference string) $hash(crs, m, r)$ in such a way that given a suitably chosen "fake" *crs*, one can find collisions in *hash* or extract m from $hash(crs, m, r)$. With such a hash function, one can construct a UC secure commitment relatively easily (see Definition 9.3 below). However, now our protocol uses a considerably more complex primitive than a simple hash function. And certainly common hash functions such as SHA-3 do not have these properties.

This leads to a strange situation: We have a protocol that we can only prove secure using a hash function that has additional weaknesses (namely that given a "bad" crs, one can cheat). One might be tempted to state that if the protocol is secure for such weak hash functions, it should in particular be secure for good hash functions. Unfortunately, such reasoning does not work in the computational setting: We cannot just remove the existence of trapdoors from the hash function – if we do so, we have a completely different hash function and our security proof makes no claims about that function.

In the symbolic world, things are different. Here it turns out that we can indeed first analyze a protocol using a hash function with trapdoors, and then remove these trapdoors in a later step, still preserving security. We call this approach the "virtual primitives" approach, because we use primitives (in this example a hash function with trapdoors) that do not need to actually exist, and that are removed in the final protocol.

In a nutshell, the virtual primitives approach when trying to realize a functionality $\mathcal{F}$ (e.g., a commitment) works as follows:

- First, identify a symbolic model $\mathcal{M}_{real}$ containing cryptographic primitives (e.g. a hash function) that should be used in the final protocol.
- Extend $\mathcal{M}_{real}$ by additional constructors, destructors, or equality rules, call the resulting model $\mathcal{M}_{virt}$. The extension $\mathcal{M}_{virt}$ should be "safe" in the sense that in $\mathcal{M}_{virt}$ an adversary will have at least as much power as in $\mathcal{M}_{real}$ (this will be made formal in Section 9.2).
- Design a protocol P. Show that P emulates $\mathcal{F}$ with respect to $\mathcal{M}_{virt}$.
- Compose P with other protocols, leading to a complex protocol $C[P] \leq C[\mathcal{F}] \leq \mathcal{G}$ (with respect to $\mathcal{M}_{virt}$) where $\mathcal{G}$ is some desired final goal, e.g., some crypto-heavy voting protocol.
- Property preservation guarantees that any property $\wp$ that holds for $\mathcal{G}$ also holds for $C[P]$ (with respect to $\mathcal{M}_{virt}$). Since $\mathcal{M}_{virt}$ only makes adversaries stronger, $\wp$ also holds for $C[P]$ with respect to $\mathcal{M}_{real}$.
- Summarizing, we have constructed a protocol $C[P]$ in a modular way such that $C[P]$ uses the symbolic model $\mathcal{M}_{real}$ (without any trapdoors) and has all the security

properties of the functionality $\mathcal{G}$.

The virtual primitive approach is not limited to commitments. But in the following sections, we illustrate it in the case of a commitment protocol. Note however, that the main theorem that allows us to conclude that $\mathcal{M}_{virt}$-security implies $\mathcal{M}_{real}$-security is formulated for general safe extensions.

A few words are in order why the virtual primitives approach works in the symbolic setting. What is the specific property of the symbolic model – in contrast to the computational one – that makes it possible? In our interpretation, this is due to the fact that a primitive (like hashes) in the symbolic world is a concrete object (i.e., a particular constructor with certain reduction rules and equalities) while in the computational world it is a class of objects (hash functions) that are described by some negative properties ("functions such that the adversary cannot..."). Therefore in the symbolic world, it is possible to formally compare executions using different kinds of a primitive (e.g., hashes with and without trapdoors); executions in one setting can be mapped into executions in the other setting by rewriting the terms sent around. In contrast, in the computational setting, this is not possible: a security result for hash functions with trapdoors has no implications for hash functions without trapdoors – these two are completely different mathematical functions on bitstrings, and it is not possible to rewrite an execution that uses one hash function into an execution using another (in particular if the adversary makes his actions depend on individual bits of the hashes). This difference between the symbolic and the computational setting seems to be the reason why virtual primitives work in the symbolic setting.

Related approaches in the computational model. Although virtual primitives as described above are restricted to the symbolic setting, somewhat related techniques do exist in the computational model. [PS04, BS05] show how to circumvent UC impossibility results (such as the impossibility of OT, commitment, or general multi-party computation without trusted setup) by giving the simulator additional power. Namely the simulator is allowed to run in (slightly) superpolynomial time. This is in some sense similar to giving the simulator access to additional constructors/destructors for extraction/equivocation as we do. Yet, there are three crucial differences to our setting: First, they can only use primitives that can actually exist computationally. For example, even a superpolynomial-time simulator cannot invert a fixed-length hash function, as part of the input is information-theoretically lost. In contrast, we can add arbitrary properties to, e.g., hash functions by introducing new equations in the symbolic model. Second, their final protocols have to use whatever primitives have been introduced for proof purposes; it is not possible to remove additional properties in the end as done in our approach. Third, their protocols involve advanced cryptographic techniques which makes the protocols considerably more involved and, consequently, inefficient. On the other hand, of course, protocols designed with our techniques are only proven secure in the symbolic model but lack a proof in the computational model – we believe therefore that our and their approaches are incomparable with respect to their advantages and disadvantages.

```
fun hash/2.
fun empty/0.
fun fake/3.
fun fakeH/2.
fun crseqv/1.
fun crsext/1.
equation hash(crseqv(n),(m,fake(n,m,r))) = fakeH(n,r).
reduc extract(n,hash(crsext(n),(m,r))) = m.
```

Figure 12: Virtual primitives example: Proverif code for the symbolic model (`virtprim-model.pv`, see [BU13])

9.1 Realizing commitments

For simplicity, we formulate a commitment functionality where the adversary is not informed that a commitment takes place (when both Alice and Bob are honest). Of course, such a functionality can only be realized if we assume perfectly secure channels between Alice and Bob that do not even allow the adversary to notice or block messages. If our protocols were to use secure channels where the adversary can notice and block communication, we would instead realize a somewhat weaker functionality which notifies the adversary[15] (the resulting changes in the proof are orthogonal to the issues of this chapter).

Definition 9.1 (Commitment) $\mathcal{F}_{COM} := io_{coma}(x_m).(\overline{io_{comb}}\langle\rangle|io_{opena}().\overline{io_{openb}}\langle x_m\rangle)$.

Symbolic model. The symbolic model $\mathcal{M}_{real}$ has constructors $hash/2$, $empty/0$, and $(\cdot,\cdot)$ (pairs) – f/n means f has arity n –, has destructors fst, snd, has no equalities, and has the rewrite rules for $fst, snd, equals$ prescribed by Definition 2.5. This model $\mathcal{M}_{real}$ is quite standard and does not use any cryptography except hash functions ($hash$ is binary for convenience only).

As explained above, to construct UC-secure commitments, we need additional "trapdoors" in our equational theory. Let $\mathcal{M}_{virt}$ be the symbolic model $\mathcal{M}_{real}$ with the following additions: Constructors $fake/3$, $fakeH/2$, $crseqv/1$, $crsext/1$, destructor $extract/2$, equation $hash(crseqv(x_n),(x_m,fake(x_n,x_m,x_r))) =_{\mathrm{E}} fakeH(x_n,x_r)$, and rewrite rule $extract(x_n,hash(crsext(x_n),(x_m,x_r))) \to x_m$.

The Proverif code for this symbolic model is given in Figure 12.

Notice that if we have a CRS $crseqv(n)$ and know n, we can open $fakeH(n,r)$ to arbitrary values. Similarly, if the CRS is $crsext(n)$ and we know n, we can extract m from $hash(crsext(n),(m,r))$. These two facts allow us to construct a simulator that does equivocation and extraction.

[15] Namely, $\mathcal{F}_{COM} := io_{coma}(x_m).(\overline{net_{coma}}\langle\rangle|net_{comb}().\overline{io_{comb}}\langle\rangle|io_{opena}().(\overline{net_{opena}}\langle\rangle|net_{openb}().\overline{io_{openb}}\langle x_m\rangle))$

Note that we introduced two different CRS-constructors for faking, *crsext* and *crseqv*. It would be tempting to use only one of them, i.e., use the equation $hash(fakecrs(x),(y,fake(x,y,z))) =_{\mathrm{E}} fakeH(x,z)$ and the reduction rule $extract(x,hash(fakecrs(x),(y,z))) \to y$. But then we would have for any terms k,m,r that $extract(k,fakeH(k,r)) =_{\mathrm{E}} extract(k,hash(fakecrs(k),(m,r))) \to m$, so by computing $extract(k,fake(k,r))$ the adversary can derive any term m, thus the adversary will know *all* secrets. This is clearly not a sensible symbolic model.

The commitment protocol. The protocol we construct uses a crs, so we first need to define the crs functionality $\mathcal{F}_{CRS}$ that gives a random non-secret value k to Alice, Bob, and the adversary.

Definition 9.2 (Common reference string) $\mathcal{F}_{CRS} := \nu k.\overline{io_{crsa}}\langle k\rangle \mid \overline{io_{crsb}}\langle k\rangle \mid \overline{net_{crs}}\langle k\rangle$.

Our protocol is then as expected. To commit to a message x_m, Alice fetches the crs x_{crs}, picks a random r, and sends $h := hash(x_{crs},(x_m,r))$ to Bob. To unveil, Alice sends (x_m,r), so that Bob can check whether h indeed contained these values. We call Alice's part of the protocol COM_A and Bob's part COM_B.

Definition 9.3 (Commitment protocol)

$$\begin{aligned}
\mathsf{COM}_A &:= io_{crsa}(x_{crs}).io_{coma}(x_m).\\
&\qquad \nu r.\big(\overline{net_1}\langle hash(x_{crs},(x_m,r))\rangle\\
&\qquad\quad |io_{opena}().\overline{net_2}\langle(x_m,r)\rangle\big)\\
\mathsf{COM}_B &:= io_{crsb}(x_{crs}).net_1(x_h).\big(\overline{io_{comb}}\langle\rangle|net_2((x_m,x_r)).\\
&\qquad \textit{if } x_h = hash(x_{crs},(x_m,x_r)) \textit{ then } \overline{io_{openb}}\langle x_m\rangle\big)\\
\mathsf{COM} &:= \nu io_{crsa} io_{crsb} net_1 net_2.(\mathsf{COM}_A|\mathsf{COM}_B|\mathcal{F}_{CRS})
\end{aligned}$$

To show that COM is a secure commitment protocol, we need to show the following lemma (cf. also the discussion on how to model corruptions in Section 4):

Lemma 9.4 *With respect to $\mathcal{M}_{virt}$, we have*

(i) Uncorrupted case: $\mathsf{COM} \le \mathcal{F}_{COM}$.

(ii) Alice corrupted: $\nu io_{crsb}.(\mathsf{COM}_B|\mathcal{F}_{CRS}\{\frac{net_{crsa}}{io_{crsa}}\}) \le \mathcal{F}_{COM}\{\frac{net_{coma}}{io_{coma}},\frac{net_{opena}}{io_{opena}}\}$

(iii) Bob corrupted: $\nu io_{crsa}.(\mathsf{COM}_A|\mathcal{F}_{CRS}\{\frac{net_{crsb}}{io_{crsb}}\}) \le \mathcal{F}_{COM}\{\frac{net_{comb}}{io_{comb}},\frac{net_{openb}}{io_{openb}}\}$.

In the proof, we show the various observational equivalences by a sequence of rewriting steps on the protocol, interspersed with automated Proverif proofs for the steps that actually involve the symbolic model (i.e., we do not have to manually deal with the complex symbolic model $\mathcal{M}_{virt}$).

We split this lemma into the following three lemmas:

Lemma 9.5 (Commitment – uncorrupted case) $\mathsf{COM} \le \mathcal{F}_{COM}$.

Proof.

$$
\begin{aligned}
\mathsf{COM} &\equiv \nu io_{crsa} io_{crsb} net_1 net_2\, k\, r.\ \overline{io_{crsa}}\langle k\rangle \mid \overline{io_{crsb}}\langle k\rangle \mid \overline{net_{crs}}\langle k\rangle \\
&\quad \mid io_{crsa}(x_{crs}).io_{coma}(x_m).\big(\overline{net_1}\langle hash(x_{crs},(x_m,r))\rangle \mid io_{opena}().\overline{net_2}\langle (x_m,r)\rangle\big) \\
&\quad \mid io_{crsb}(x_{crs}).net_1(x_h).\big(\overline{io_{comb}}\langle\rangle | net_2((x'_m,x_r)).\\
&\qquad\qquad \text{if } x_h = hash(x_{crs},(x'_m,x_r)) \text{ then } \overline{io_{openb}}\langle x'_m\rangle\big) \\
&\overset{\text{(i)}}{\approx} \nu\ net_1 net_2\, k\, r.\ \overline{net_{crs}}\langle k\rangle \\
&\quad \mid io_{coma}(x_m).\big(\overline{net_1}\langle hash(k,(x_m,r))\rangle \mid io_{opena}().\overline{net_2}\langle (x_m,r)\rangle\big) \\
&\quad \mid net_1(x_h).\big(\overline{io_{comb}}\langle\rangle | net_2((x'_m,x_r)).\\
&\qquad\qquad \text{if } x_h = hash(k,(x'_m,x_r)) \text{ then } \overline{io_{openb}}\langle x'_m\rangle\big) \\
&\overset{\text{(ii)}}{\approx} \nu\ net_2\, k\, r.\ \overline{net_{crs}}\langle k\rangle \\
&\quad \mid io_{coma}(x_m).\big(\overline{io_{comb}}\langle\rangle \mid net_2((x'_m,x_r)).\\
&\qquad\qquad \text{if } hash(k,(x_m,r)) = hash(k,(x'_m,x_r)) \text{ then } \overline{io_{openb}}\langle x'_m\rangle \\
&\qquad\qquad \mid io_{opena}().\overline{net_2}\langle (x_m,r)\rangle\big) \\
&\overset{\text{(iii)}}{=} \nu net_2\, k\, r.\ \overline{net_{crs}}\langle k\rangle \\
&\quad \mid io_{coma}(x_m).\big(\overline{io_{comb}}\langle\rangle \mid net_2(x_{tmp}).\text{let } (x'_m,x_r) = z \text{ in} \\
&\qquad\qquad \text{if } hash(k,(x_m,r)) = hash(k,(x'_m,x_r)) \text{ then } \overline{io_{openb}}\langle x'_m\rangle \\
&\qquad\qquad \mid io_{opena}().\overline{net_2}\langle (x_m,r)\rangle\big) \\
&\overset{\text{(iv)}}{\approx} \nu\ k\, r.\ \overline{net_{crs}}\langle k\rangle \\
&\quad \mid io_{coma}(x_m).\big(\overline{io_{comb}}\langle\rangle \mid io_{opena}().\text{let } (x'_m,x_r) = (x_m,r) \text{ in} \\
&\qquad\qquad \text{if } hash(k,(x_m,r)) = hash(k,(x'_m,x_r)) \text{ then } \overline{io_{openb}}\langle x'_m\rangle\ \big) \\
&\overset{\text{(v)}}{\approx} \nu k\, r.\ \overline{net_{crs}}\langle k\rangle \mid io_{coma}(x_m).\big(\overline{io_{comb}}\langle\rangle \mid io_{opena}().\ \overline{io_{openb}}\langle x_m\rangle\big) \\
&\equiv \mathcal{F}_{COM} \mid S \quad \text{with} \quad S := \nu k\, r.\overline{net_{crs}}\langle k\rangle
\end{aligned}
$$

Here (i) uses two invocations of Lemma 3.3, one with $n := io_{crsa}$, $t := k$, and $x := x_{crs}$, and one with $n := io_{crsb}$, $t := k$, and $x := x_{crs}$.

And (ii) uses one invocation of Lemma 3.3 with $n := net_1$, $x := x_h$, and $t := hash(k,(x_m,r))$.

And (iii) uses the fact that $t(p).P$ is syntactic sugar for $t(z).\text{let } p = z \text{ in } P$ for a pattern p and a fresh variable z.

And (iv) uses one invocation of Lemma 3.3 with $n := net_2$, $x := x_{tmp}$, and $t := (x_m,r)$. (And it uses Lemma 2.7, so that we can apply Lemma 3.3 to a subprocess instead of the whole process.)

And (v) uses several invocations of Lemma 3.2 (v) to evaluate the let- and the if-statement.

So $\mathsf{COM} \approx \mathcal{F}_{COM} \mid S$ for some S with $\mathrm{IO} \cap fn(S) = \varnothing$. Hence $\mathsf{COM} \leq \mathcal{F}_{COM}$. □

Lemma 9.6 (Commitment – Alice corrupted)
$\nu io_{crsb}.(\mathsf{COM}_B | \mathcal{F}_{CRS}\{\frac{net_{crsa}}{io_{crsa}}\}) \leq \mathcal{F}_{COM}\{\frac{net_{coma}}{io_{coma}}, \frac{net_{opena}}{io_{opena}}\}$

```
free netcrs,netcrsa,net1,net2,iocomb,ioopenb.

process
new k;
out(netcrsa,choice[k,crsext(k)]) |
out(netcrs,choice[k,crsext(k)]) |
in(net1,xh);
out(iocomb,empty) |
in(net2,(xm,xr));
if xh = hash(choice[k,crsext(k)],(xm,xr)) then
out(ioopenb,choice[xm,extract(k,xh)])
```

Figure 13: Virtual primitives example: Proverif code for corrupted Alice (`virtprim-acorr.pv`, see [BU13]). (Has to be prefixed with the code from Figure 12.)

Proof. We have

$\nu io_{crsb}.(\mathsf{COM}_B|\mathcal{F}_{CRS}\{\frac{net_{crsa}}{io_{crsa}}\})$

$\overset{\text{(i)}}{\approx} \nu k.\overline{net_{crsa}}\langle k\rangle \mid \overline{net_{crs}}\langle k\rangle \mid net_1(x_h).\big(\overline{io_{comb}}\langle\rangle|$
$\quad net_2((x_m,x_r)).\text{if } x_h = hash(k,(x_m,x_r)) \text{ then } \overline{io_{openb}}\langle x_m\rangle\big)$

$\overset{\text{(ii)}}{\approx} \nu k.\overline{net_{crsa}}\langle crsext(k)\rangle \mid \overline{net_{crs}}\langle crsext(k)\rangle \mid net_1(x_h).\big(\overline{io_{comb}}\langle\rangle|$
$\quad net_2((x_m,x_r)).\text{if } x_h = hash(crsext(k),(x_m,x_r)) \text{ then } \overline{io_{openb}}\langle extract(k,x_h)\rangle\big)$

$\overset{\text{(iii)}}{\approx} \nu k.\overline{net_{crsa}}\langle crsext(k)\rangle \mid \overline{net_{crs}}\langle crsext(k)\rangle \mid net_1(x_h).\nu net_{opena}.\big(\overline{io_{comb}}\langle\rangle|net_{opena}().\overline{io_{openb}}\langle extract(k,x_h)\rangle|$
$\quad net_2((x_m,x_r)).\text{if } x_h = hash(crsext(k),(x_m,x_r)) \text{ then } \overline{net_{opena}}\langle\rangle\big)$

$\overset{\text{(iv)}}{\approx} \nu net_{opena}\, k.\overline{net_{crsa}}\langle crsext(k)\rangle \mid \overline{net_{crs}}\langle crsext(k)\rangle \mid net_1(x_h).\big(\overline{io_{comb}}\langle\rangle|net_{opena}().\overline{io_{openb}}\langle extract(k,x_h)\rangle$
$\quad net_2((x_m,x_r)).\text{if } x_h = hash(crsext(k),(x_m,x_r)) \text{ then } \overline{net_{opena}}\langle\rangle\big)$

$\overset{\text{(v)}}{\approx} \nu net_{coma}\, net_{opena} k.\overline{net_{crsa}}\langle crsext(k)\rangle \mid \overline{net_{crs}}\langle crsext(k)\rangle \mid net_1(x_h).\big(\overline{net_{coma}}\langle extract(k,x_h)\rangle|$
$\quad net_2((x_m,x_r)).\text{if } x_h = hash(crsext(k),(x_m,x_r)) \text{ then } \overline{net_{opena}}\langle\rangle\big)|$
$\quad net_{coma}(x'_m).\big(\overline{io_{comb}}\langle\rangle|net_{opena}().\overline{io_{openb}}\langle x'_m\rangle\big)$

$\equiv \nu net_{coma}\, net_{opena}.(\mathcal{F}_{COM}\{\frac{net_{coma}}{io_{coma}}, \frac{net_{opena}}{io_{opena}}\}|S)$ for some S with $\mathrm{IO} \cap fn(S) = \varnothing$.

Here (i) uses Lemma 3.3 with $n := io_{crsb}$, $C := \nu k.\overline{net_{crsa}}\langle k\rangle \mid \overline{net_{crs}}\langle k\rangle \mid \Box$, $x := x_{crs}$, and $t := k$.

And (ii) is shown using Proverif, the required code is given in Figure 13. Note that in the rhs of (ii), we have replaced all occurrences of the CRS k by $crsext(k)$, and instead of outputting x_m in the end, we output $extract(k, x_h)$.

And (iii) uses Lemma 3.3 (in the opposite direction) with $n := net_{opena}$, $Q := \overline{io_{openb}}\langle extract(k,x_h)\rangle$, and $C := \overline{io_{comb}}\langle\rangle|net_2((x_m,x_r))$.

if $x_h = hash(crsext(k), (x_m, x_r))$ then $\Box$. (And it uses Lemma 2.7, so that we can apply Lemma 3.3 to a subprocess instead of the whole process.)

And (iv) uses Lemma 3.2 (ii) to swap νnet_{opena} and $net_1(x_h)$. (And Lemma 2.7 to apply Lemma 3.2 (ii) to a subprocess.)

And (v) uses Lemma 3.3 (in the opposite direction) with $n := net_{coma}$, $x := x'_m$, $t := extract(k, x_h)$, and $Q := \overline{io_{comb}}\langle\rangle | net_{opena}().\overline{io_{openb}}\langle x'_m\rangle\big)$.

So we have $\nu io_{crsb}.(\mathsf{COM}_B|\mathcal{F}_{CRS}\{\frac{net_{crsa}}{io_{crsa}}\}) \approx \nu net_{coma}\, net_{opena}.(\mathcal{F}_{COM}\{\frac{net_{coma}}{io_{coma}}, \frac{net_{opena}}{io_{opena}}\}|S)$ for some S with $\mathrm{IO} \cap fn(S) = \varnothing$. Hence $\nu io_{crsb}.(\mathsf{COM}_B|\mathcal{F}_{CRS}\{\frac{net_{crsa}}{io_{crsa}}\}) \leq \mathcal{F}_{COM}\{\frac{net_{coma}}{io_{coma}}, \frac{net_{opena}}{io_{opena}}\}$. $\Box$

Lemma 9.7 (Commitment – Bob corrupted)
$\nu io_{crsa}.(\mathsf{COM}_A|\mathcal{F}_{CRS}\{\frac{net_{crsb}}{io_{crsb}}\}) \leq \mathcal{F}_{COM}\{\frac{net_{comb}}{io_{comb}}, \frac{net_{openb}}{io_{openb}}\}$.

Proof. We have

$\nu io_{crsa}.(\mathsf{COM}_A|\mathcal{F}_{CRS}\{\frac{net_{crsb}}{io_{crsb}}\})$

$\overset{\text{(i)}}{\approx} \nu k.\overline{net_{crsb}}\langle k\rangle \mid \overline{net_{crs}}\langle k\rangle \mid io_{coma}(x_m).\nu r.\big(\overline{net_1}\langle hash(k,(x_m,r))\rangle | io_{opena}().\overline{net_2}\langle(x_m,r)\rangle\big)$

$\overset{\text{(ii)}}{\approx} \nu k.\overline{net_{crsb}}\langle crseqv(k)\rangle \mid \overline{net_{crs}}\langle crseqv(k)\rangle \mid io_{coma}(x_m).\nu r.$
$\big(\overline{net_1}\langle fakeH(k,r)\rangle | io_{opena}().\overline{net_2}\langle(x_m, fake(k,x_m,r))\rangle\big)$

$\overset{\text{(iii)}}{\approx} \nu k.\overline{net_{crsb}}\langle crseqv(k)\rangle \mid \overline{net_{crs}}\langle crseqv(k)\rangle \mid io_{coma}(x_m).\nu r.$
$\nu net_{openb}.\big(\overline{net_1}\langle fakeH(k,r)\rangle | io_{opena}().\overline{net_{openb}}\langle x_m\rangle | net_{openb}(x'_m).\overline{net_2}\langle(x'_m, fake(k,x'_m,r))\rangle\big)$

$\overset{\text{(iv)}}{\approx} \nu net_{openb}\, k\, r.\overline{net_{crsb}}\langle crseqv(k)\rangle \mid \overline{net_{crs}}\langle crseqv(k)\rangle \mid io_{coma}(x_m).$
$\big(\overline{net_1}\langle fakeH(k,r)\rangle | io_{opena}().\overline{net_{openb}}\langle x_m\rangle | net_{openb}(x'_m).\overline{net_2}\langle(x'_m, fake(k,x'_m,r))\rangle\big)$

$\overset{\text{(v)}}{\approx} \nu net_{comb}\, net_{openb}\, k\, r.\overline{net_{crsb}}\langle crseqv(k)\rangle \mid \overline{net_{crs}}\langle crseqv(k)\rangle \mid io_{coma}(x_m).$
$\big(io_{opena}().\overline{net_{openb}}\langle x_m\rangle | \overline{net_{comb}}\langle\rangle\big) \mid$
$net_{comb}().\big(\overline{net_1}\langle fakeH(k,r)\rangle | net_{openb}(x'_m).\overline{net_2}\langle(x'_m, fake(k,x'_m,r))\rangle\big)$

$\equiv \nu net_{comb}\, net_{openb}.(\mathcal{F}_{COM}\{\frac{net_{comb}}{io_{comb}}, \frac{net_{openb}}{io_{openb}}\}|S)$ for some S with $\mathrm{IO} \cap fn(S) = \varnothing$.

Here (i) uses Lemma 3.3 with $n := io_{crsa}$, $C := \nu k.\overline{net_{crsb}}\langle k\rangle \mid \overline{net_{crs}}\langle k\rangle \mid \Box$, $x := x_{crs}$, and $t := k$.

And (ii) is shown using Proverif, the required code is given in Figure 14. Note that in the rhs of (ii), we have replaced all occurrences of the CRS k by $crseqv(k)$, and instead of sending the hash value $hash(k,(x_m,r))$ we send $fakeH(k,r)$ which does not depend on x_m, and in the end, instead of sending the randomness r, we send $fake(k,x_m,r)$. Intuitively, this replacement is indistinguishable because our symbolic model contains the equation $hash(crseqv(k),(m,fake(k,m,r))) =_{\mathrm{E}} fakeH(k,r)$.

And (iii) uses Lemma 3.3 (in the opposite direction) with $n := net_{openb}$, $x := x'_m$, $t := x_m$, $Q := \overline{net_2}\langle(x'_m, fake(k,x'_m,r))\rangle$, and $C := \overline{net_1}\langle fakeH(k,r)\rangle \mid io_{opena}().\Box$. (And

```
free netcrs,netcrsb,net1,net2,iocoma,ioopena.

process
new k;
out(netcrs,choice[k,crseqv(k)]) |
out(netcrsb,choice[k,crseqv(k)]) |
in(iocoma,xm);
new r;
out(net1,choice[hash(k,(xm,r)),fakeH(k,r)]) |
in(ioopena,x);
out(net2,(xm,choice[r,fake(k,xm,r)]))
```

Figure 14: Virtual primitives example: Proverif code for corrupted Bob (`virtprim-bcorr.pv`, see [BU13]). (Has to be prefixed with the code from Figure 12.)

it uses Lemma 2.7, so that we can apply Lemma 3.3 to a subprocess instead of the whole process.)

And (iv) uses Lemma 3.2 (ii) to swap νr and νnet_{openb} with $io_{coma}(x_m)$. (And Lemma 2.7 to apply Lemma 3.2 (ii) to a subprocess.)

And (v) uses Lemma 3.3 (in the opposite direction) with $n := net_{comb}$, $t := empty$, and $Q := \overline{net_1}\langle fakeH(k,r)\rangle \mid net_{openb}(x'_m).\overline{net_2}\langle(x'_m, fake(k,x'_m,r))\rangle$.

So we have $\nu io_{crsa}.(\mathsf{COM}_A|\mathcal{F}_{CRS}\{\frac{net_{crsb}}{io_{crsb}}\}) \approx \nu net_{comb}\, net_{openb}.(\mathcal{F}_{COM}\{\frac{net_{comb}}{io_{comb}}, \frac{net_{openb}}{io_{openb}}\}|S)$ for some S with $\mathrm{IO} \cap fn(S) = \varnothing$. Hence $\nu io_{crsa}.(\mathsf{COM}_A|\mathcal{F}_{CRS}\{\frac{net_{crsb}}{io_{crsb}}\}) \leq \mathcal{F}_{COM}\{\frac{net_{comb}}{io_{comb}}, \frac{net_{openb}}{io_{openb}}\}$. □

9.1.1 A note on adaptive corruption

We have only modeled static corruption in our examples, i.e., it is fixed in the beginning of the execution which parties are corrupted. If we were to model adaptive corruption where parties may be corrupted during the protocol execution, we would face an additional challenge (besides the fact that the descriptions of the processes would be much more complex): Since the simulator may have to provide the CRS before he knows whether Alice or Bob will be corrupted, he will not know whether he should use $crseqv(k)$ or $crsext(k)$ as CRS. And on page 82 we explained why we cannot just replace both *crseqv* and *crseqv* by a single constructor *fakecrs* because then the adversary would be able to deduce any term. However, this problem can be solved using the conditional destructors supported by Proverif 1.87: we can make sure that the rewrite rule $extract(x_n, hash(crsext(x_n),(x_m,x_r))) \to x_m$ only triggers if $hash(crsext(x_n),(x_m,x_r)) \neq_{\mathrm{E}} fakeH(M,M')$ for all M, M'. The resulting symbolic model is shown in Figure 15. We can show Lemmas 9.5, 9.6, and 9.7 also using this symbolic model by replacing all occurrences of *crseqv* and *crsext* in the simulators by *fakecrs*. Proverif still shows all the necessary equivalences. Although this does not show adaptive

```
(* Needs proverif1.87 beta *)

fun hash(bitstring,bitstring):bitstring.
const empty:bitstring.
fun fake(bitstring,bitstring,bitstring):bitstring.
fun fakeH(bitstring,bitstring):bitstring.
fun fakecrs(bitstring):bitstring.

equation forall n:bitstring,m:bitstring,r:bitstring;
  hash(fakecrs(n),(m,fake(n,m,r))) = fakeH(n,r).

fun extract(bitstring,bitstring):bitstring
reduc forall n:bitstring,r:bitstring;
    extract(n,fakeH(n,r)) = empty
otherwise forall n:bitstring,m:bitstring,r:bitstring;
    extract(n,hash(fakecrs(n),(m,r))) = m.
```

Figure 15: Virtual primitives example: Proverif code for the symbolic model when using *fakecrs* constructor (`virtprim-model-x.pv`, see [BU13]) Note that we use the typed Proverif syntax here because Proverif 1.87 does not support conditional destructors in the untyped syntax.

security, it shows that the simulator does not need to choose the CRS depending on who is corrupted, giving hope for the adaptive case. We leave that case for future work.

9.2 Removing the virtual primitives

In this section, we will consider different symbolic models. Since the relation symbols $\rightarrow, \Downarrow, \approx, \downarrow, =_{\mathrm{E}}$ etc. do not explicitly specify the symbolic model, we use the following convention: When referring to a symbolic model $\mathcal{M}_i$, we write $\rightarrow_i, \Downarrow_i, \approx_i, \downarrow^i, =_{E_i}$ etc. We say a term (or destructor term) is an $\mathcal{M}$-*term* (or $\mathcal{M}$-destructor term) if it contains only constructors (and destructors) from $\mathcal{M}$. We call a process an $\mathcal{M}$-*process* if it contains only $\mathcal{M}$-terms and $\mathcal{M}$-destructor terms.

We have now shown that COM is a secure commitment protocol with respect to $\mathcal{M}_{virt}$. However, we would like to deduce security of protocols using COM with respect to $\mathcal{M}_{real}$. For this, we first need to formalize what it means that $\mathcal{M}_{virt}$ is a safe extension of $\mathcal{M}_{real}$:

Definition 9.8 (Safe extension) *We call a symbolic model $\mathcal{M}_1 = (\Sigma_1, \mathrm{E}_1, \mathrm{R}_1)$ a* safe extension *of a symbolic model $\mathcal{M}_2 = (\Sigma_2, \mathrm{E}_2, \mathrm{R}_2)$ iff the following holds:*

(i) $\Sigma_1 \supseteq \Sigma_2$.

(ii) If D is an $\mathcal{M}_2$-destructor term, and M is an $\mathcal{M}_1$-term, and $D \Downarrow_1 M$, then there exists an $\mathcal{M}_2$-term $M' =_{E_1} M$ with $D \Downarrow_2 M'$.

(iii) For all $\mathcal{M}_2$-destructor terms D and $\mathcal{M}_2$-terms M, we have $D \Downarrow_2 M \Rightarrow D \Downarrow_1 M$.

(iv) For all $\mathcal{M}_2$-terms M, M' we have $M =_{E_1} M' \Leftrightarrow M =_{E_2} M'$.

The following lemma is relatively easy to show:

Lemma 9.9 *$\mathcal{M}_{virt}$ is a safe extension of $\mathcal{M}_{real}$.*

Proof. Obviously, $\Sigma_{virt} \supseteq \Sigma_{real}$. So Definition 9.8 (i) is satisfied.

We show that Definition 9.8 (ii) is satisfied: Let D be an $\mathcal{M}_{real}$-destructor term and M be an $\mathcal{M}_{virt}$-term. Since $\mathcal{M}_{real}$ contains no destructors, D is an $\mathcal{M}_{real}$-term. Thus $D \Downarrow_{virt} M$ implies $D = M$. This implies that $M' := M$ is an $\mathcal{M}_{real}$-term and $D \Downarrow_{real} M'$.

We show that Definition 9.8 (iii) is satisfied: Let D be an $\mathcal{M}_{real}$-destructor term and M be an $\mathcal{M}_{real}$-term. Since $\mathcal{M}_{real}$ contains no destructors, D is an $\mathcal{M}_{real}$-term. Thus $D \Downarrow_{virt} M$ implies $D = M$ which implies $D \Downarrow_{real} M$.

We show that Definition 9.8 (iv) is satisfied: For $\mathcal{M}_{real}$-terms M, M', obviously $M =_{E_{real}} M'$ implies $M =_{E_{virt}} M'$. We show the opposite direction: The only equation in E_{virt} (namely $hash(crseqv(k), (m, fake(k, m, r))) =_{\mathrm{E}} fakeH(k, r)$) only allows us to rewrite terms containing *crseqv* or *fakeH*. Since M, M' are $\mathcal{M}_{real}$-terms, they do not contain these constructors. Hence $M =_{E_{virt}} M'$ only if $M = M'$. So $M =_{E_{virt}} M'$ implies $M =_{E_{real}} M'$. □

The following theorem justifies the above definition of safe extensions:

Theorem 9.10 *Assume that $\mathcal{M}_1$ is a safe extension of $\mathcal{M}_2$. Then for all $\mathcal{M}_2$-processes P, P' we have $P \approx_1 P' \Rightarrow P \approx_2 P'$.*

Proof. We first show some auxiliary claims:

Claim 1 *For all $\mathcal{M}_2$-processes P, P', we have $P \to_2 P' \Rightarrow P \to_1 P'$.*

We show this claim by induction over the derivation of $P \to_2 P'$. We distinguish the following cases:

- *Closure under structural equivalence:* In this case $P \to_2 P'$ has been derived from $P \equiv \hat{P} \to_2 \hat{P}' \equiv P'$ for $\mathcal{M}_2$-processes $\hat{P}, \hat{P}'$, and the induction hypothesis implies $\hat{P} \to_1 \hat{P}'$. Thus $P \equiv \hat{P} \to_1 \hat{P}' \equiv P'$ which implies $P \to_1 P'$. The claim follows.
- *Closure under evaluation contexts:* In this case $P \to_2 P'$ has been derived from $P = E[\hat{P}]$, $P' = E[\hat{P}']$, and $\hat{P} \to_2 \hat{P}'$ for some $\mathcal{M}_2$-processes $\hat{P}, \hat{P}'$ and some $\mathcal{M}_2$-evaluation context E. The induction hypothesis implies $\hat{P} \to_1 \hat{P}'$. Hence $P = E[\hat{P}] \to_1 E[\hat{P}'] = P'$.
- *REPL:* In this case $P = !\hat{P}$ and $P' = \hat{P}|!\hat{P}$. Hence $P \to_1 P'$.
- *COMM:* In this case $P = \overline{C}\langle T\rangle.\hat{P} \mid C'(x).\hat{Q}$ and $P' = \hat{P} \mid Q\{T/x\}$ and $C =_{E_2} C'$. Since P is an $\mathcal{M}_2$-process, C, C' are $\mathcal{M}_2$-terms. Since $\mathcal{M}_1$ is a safe extension of $\mathcal{M}_2$, $C =_{E_2} C'$ implies $C =_{E_1} C'$. Thus $P \to_1 P'$. The claim follows.
- *LET-THEN:* In this case $P = (\text{let } x = D \text{ in } \hat{P} \text{ else } \hat{Q})$ and $P' = \hat{P}\{M/x\}$ for some $\mathcal{M}_2$-processes $\hat{P}, \hat{Q}$, and some $\mathcal{M}_2$-destructor term D and $\mathcal{M}_2$-term M with $D \Downarrow_2 M$. Since P is an $\mathcal{M}_2$-process, D is an $\mathcal{M}_2$-destructor term. Since $\mathcal{M}_1$ is a safe extension of $\mathcal{M}_2$, $D \Downarrow_2 M$ implies that $D \Downarrow_1 M$. Thus $P \to_1 P'$. The claim follows.

- *LET-ELSE:* In this case $P = (\text{let } x = D \text{ in } \hat{P} \text{ else } \hat{Q})$ and $P' = \hat{Q}$ and for all $\mathcal{M}_2$-terms M we have $D \not\Downarrow_2 M$. Since P is an $\mathcal{M}_2$-process, D is an $\mathcal{M}_2$-destructor term. If we had $D \Downarrow_1 M$ for some $\mathcal{M}_1$-term M, we would have $D \Downarrow_2 M'$ for some $\mathcal{M}_2$-term M' since $\mathcal{M}_1$ is a safe extension of $\mathcal{M}_2$. This contradicts $D \not\Downarrow_2 M$ for all $\mathcal{M}_2$-terms M. Thus $D \not\Downarrow_1 M$ for all $\mathcal{M}_1$-terms M. Hence $P \rightarrow_1 P'$. The claim follows.

Claim 2 *For all $\mathcal{M}_2$-processes P, and all $\mathcal{M}_1$-processes P'' with $P \rightarrow_1 P''$, there exists an $\mathcal{M}_2$-process P' such that $P \rightarrow_2 P' \equiv_{E_1} P''$.*

We show this claim by induction over the derivation of $P \rightarrow_1 P''$. We distinguish the following cases:

- *Closure under structural equivalence:* In this case $P \rightarrow_1 P''$ has been derived from $P \equiv \hat{P} \rightarrow_1 \hat{P}'' \equiv P''$ for $\mathcal{M}_1$-processes $\hat{P}, \hat{P}''$, and the induction hypothesis (Claim 2) holds for $\hat{P} \rightarrow_1 \hat{P}''$. Since structural equivalence does not rewrite terms, the fact that P is an $\mathcal{M}_2$-process implies that $\hat{P}$ is an $\mathcal{M}_2$-process. Thus $\hat{P} \rightarrow_1 \hat{P}''$ implies together with the induction hypothesis that $\hat{P} \rightarrow_2 P' \equiv_{E_1} \hat{P}''$ for some $\mathcal{M}_2$-process P'. Thus $P \equiv \hat{P} \rightarrow_2 P'$ which implies $P \rightarrow_2 P'$ and we have $P' \equiv_{E_1} \hat{P}'' \equiv P''$ which implies $P' \equiv_{E_1} P''$. The claim follows.
- *Closure under evaluation contexts:* In this case $P \rightarrow_1 P''$ has been derived from $P = E[\hat{P}]$, $P'' = E[\hat{P}'']$, and $\hat{P} \rightarrow_1 \hat{P}''$ for some $\mathcal{M}_1$-processes $\hat{P}, \hat{P}''$ and some $\mathcal{M}_1$-evaluation context E. And the induction hypothesis holds for $\hat{P} \rightarrow_1 \hat{P}''$. Since P is an $\mathcal{M}_2$-process and $P = E[\hat{P}]$, we have that $\hat{P}$ is an $\mathcal{M}_2$-process and E and $\mathcal{M}_2$-evaluation context. Thus by induction hypothesis, there exists an $\mathcal{M}_2$-process $\hat{P}'$ such that $\hat{P} \rightarrow_2 \hat{P}' \equiv_{E_1} \hat{P}''$. Let $P' := E[\hat{P}']$. Obviously P' is an $\mathcal{M}_2$-process. And $P = E[\hat{P}] \rightarrow_2 E[\hat{P}'] = P'$ and $P'' = E[\hat{P}''] \equiv_{E_1} E[\hat{P}'] = P'$. The claim follows.
- *REPL:* In this case $P = !\hat{P}$ and $P'' = \hat{P}|!\hat{P}$. Since P is an $\mathcal{M}_2$-process, so is $\hat{P}$, and hence also $P' := P''$ is an $\mathcal{M}_2$-process. Then $P \rightarrow_2 P'$ and $P'' \equiv_{E_1} P'$ and the claim follows.
- *COMM:* In this case $P = \overline{C}\langle T\rangle.\hat{P} \mid C'(x).\hat{Q}$ and $P'' = \hat{P} \mid \hat{Q}\{T/x\}$ and $C =_{E_1} C'$. Since P is an $\mathcal{M}_2$-process, C, C' are $\mathcal{M}_2$-terms and $\hat{P}, \hat{Q}$ are $\mathcal{M}_2$-processes. Since $\mathcal{M}_1$ is a safe extension of $\mathcal{M}_2$, $C =_{E_1} C'$ implies $C =_{E_2} C'$. Thus $P \rightarrow_2 P''$. With $P' := P''$, the claim follows.
- *LET-THEN:* In this case $P = (\text{let } x = D \text{ in } \hat{P} \text{ else } \hat{Q})$ and $P'' = \hat{P}\{M/x\}$ for some $\mathcal{M}_1$-processes $\hat{P}, \hat{Q}$, and some $\mathcal{M}_1$-destructor term D and $\mathcal{M}_1$-term M with $D \Downarrow_1 M$. Since P is an $\mathcal{M}_2$-process, $\hat{P}, \hat{Q}$ are $\mathcal{M}_2$-processes and D is an $\mathcal{M}_2$-destructor term. Since $\mathcal{M}_1$ is a safe extension of $\mathcal{M}_2$, $D \Downarrow_1 M$ implies that $D \Downarrow_2 M'$ for some $\mathcal{M}_2$-term $M' =_{E_1} M$. Let $P' := \hat{P}\{M/x\}$. Then $P'' = \hat{P}\{M/x\} \equiv_{E_1} \hat{P}\{M'/x\} = P'$ and $P \rightarrow_2 P'$. The claim follows.
- *LET-ELSE:* In this case $P = (\text{let } x = D \text{ in } \hat{P} \text{ else } \hat{Q})$ and $P'' = \hat{Q}$ and for all $\mathcal{M}_1$-terms M we have $D \not\Downarrow_1 M$. Since P is an $\mathcal{M}_2$-process, $\hat{P}, \hat{Q}$ are $\mathcal{M}_2$-processes and D is an $\mathcal{M}_2$-destructor term. Since $\mathcal{M}_1$ is a safe extension of $\mathcal{M}_2$, for all $\mathcal{M}_2$-terms

M, $D \not\Downarrow_1 M$ implies that $D \not\Downarrow_2 M$. With $P' := \hat{Q} = P''$, we thus have $P'' \equiv_{E_1} P'$ and $P \to_2 P'$. The claim follows.

Claim 3 *For all $\mathcal{M}_2$-processes P, and all $\mathcal{M}_1$-processes P'' with $P \to_1^* P''$, there exists an $\mathcal{M}_2$-process P' such that $P \to_2^* P' \equiv_{E_1} P''$.*

Proof. To show this claim, we show that for all $n \geq 0$, all $\mathcal{M}_2$-processes P, and all $\mathcal{M}_1$-processes P'' with $P \to_1^n P''$, there exists an $\mathcal{M}_2$-process P' such that $P \to_2^* P' \equiv_{E_1} P''$. Here $\to_1^n$ means exactly n applications of $\to$. We show this by induction over n. For $n = 0$, the statement is trivial. Assume the statement holds for n, we show it for $n+1$: We have $P \to_1^{n+1} P''$ hence $P \to_1^n \hat{P}'' \to_1 P''$ for some $\mathcal{M}_1$-process $\hat{P}''$. By induction hypothesis there exists an $\mathcal{M}_2$-process $\hat{P}'$ with $P \to_2^* \hat{P}' \equiv_{E_1} \hat{P}''$. Since $\hat{P}' \equiv_{E_1} \hat{P}'' \to_1 P''$, by Lemma 3.5, we have $\hat{P}' \to_1 P_2 \equiv_{E_1} P''$ for some $\mathcal{M}_1$-process P_2. Since $\hat{P}'$ is an $\mathcal{M}_2$-process and $\hat{P}' \to_1 P_2$, by Claim 2, there is an $\mathcal{M}_2$-process P' such that $\hat{P}' \to_2 P' \equiv_{E_1} P_2$. Combining all this, we have

$$P \to_2^* \hat{P}' \to_2 P' \equiv_{E_1} P_2 \equiv_{E_1} P''.$$

Thus $P \to_2^* P' \equiv_{E_1} P''$. □

We are now ready to show Theorem 9.10. Let $\mathcal{R} := \{(P, Q) : P, Q \ \mathcal{M}_2\text{-processes}, P \approx_1 Q\}$. We show that $\mathcal{R}$ is an $\mathcal{M}_2$-simulation (and due to its symmetry also an $\mathcal{M}_2$-bisimulation):

- If $(P, Q) \in \mathcal{R}$ and $P \downarrow_M^2$ for some $\mathcal{M}_2$-term M, then $Q \to_2^* Q' \downarrow_M^2$ for some $\mathcal{M}_2$-process Q'.

  $P \downarrow_M^2$ implies (see Footnote 7) $P \equiv_{E_2} E[\overline{M}\langle T\rangle.P']$ for some evaluation context E not binding $fn(M)$. This implies $P \equiv_{E_1} E[\overline{M}\langle T\rangle.P']$ (since $M_1 =_{E_2} M_2$ implies $M_1 =_{E_1} M_2$ for $\mathcal{M}_2$-terms M_1, M_2). Thus $P \downarrow_M^1$. Since $(P, Q) \in \mathcal{R}$, we have that $P \approx_1 Q$ and thus $Q \to_1^* Q'' \downarrow_M^1$ for some $\mathcal{M}_1$-process Q''. By Claim 3, this implies that $Q \to_2^* Q' \equiv_{E_1} Q''$ for some $\mathcal{M}_2$-process Q'. Since $Q'' \equiv_{E_1} Q'' \downarrow_M^1$, we have $Q' \downarrow_M^1$ (this follows immediately using the characterization from Footnote 7). Since $Q' \downarrow_M^1$, by definition of $\downarrow$, we have $Q' \equiv E[\overline{M'}\langle T'\rangle.\tilde{Q}]$ for some $\mathcal{M}_1$-terms M', T' with $M' =_{E_1} M$ and $\mathcal{M}_1$-process $\tilde{Q}$, and some evaluation context not binding $fn(M)$. Since Q' is an $\mathcal{M}_2$-process, $E[\overline{M'}\langle T'\rangle.\tilde{Q}]$ is an $\mathcal{M}_2$-process, hence M' is an $\mathcal{M}_2$-term. Thus M, M' are $\mathcal{M}_2$-terms, and $M' =_{E_1} M$. Since $\mathcal{M}_1$ is a safe extension of $\mathcal{M}_2$, this implies $M' =_{E_2} M$. Thus $Q' \equiv E[\overline{M'}\langle T'\rangle.\tilde{Q}]$ implies $Q' \downarrow_M^2$. So we have $Q \to_2^* Q' \downarrow_M^2$ and Q' is an $\mathcal{M}_2$-process.

- If $(P, Q) \in \mathcal{R}$ and $P \to_2 P'$ for an $\mathcal{M}_2$-process P', then there exists an $\mathcal{M}_2$-process Q' with $(P', Q') \in \mathcal{R}$ and $Q \to_2^* Q'$:

  Since P, P' are $\mathcal{M}_2$-processes, and $P \to_2 P'$, by Claim 1 we have $P \to_1 P'$. Since $(P, Q) \in \mathcal{R}$, we have $P \approx_1 Q$ and thus $Q \to_1^* Q''$ for some $\mathcal{M}_1$-process $Q'' \approx_1 P'$. By Claim 3, there is an $\mathcal{M}_2$-process Q' such that $Q'' \equiv_{E_1} Q'$ and $Q \to_2^* Q'$.

Furthermore, by Lemma 3.2 (iv), we have $=_{E_1} \subseteq \approx_1$ and trivially $\equiv \subseteq \approx_1$, hence $\equiv_{E_1} \subseteq \approx_1$. Thus $Q'' \equiv_{E_1} Q'$ implies $Q'' \approx_1 Q'$. Together with $Q'' \approx_1 P'$, we have $P' \approx_1 Q'$ and thus $(P', Q') \in \mathcal{R}$.

- If $(P, Q) \in \mathcal{R}$ and E is an $\mathcal{M}_2$-evaluation context, then $(E[P], E[Q]) \in \mathcal{R}$.

  Since $(P, Q) \in \mathcal{R}$, we have $P \approx_1 Q$. Furthermore, since E is an $\mathcal{M}_2$-evaluation context, E is also an $\mathcal{M}_1$-evaluation context. Hence $E[P] \approx_1 E[Q]$ and thus $(E[P], E[Q]) \in \mathcal{R}$.

Since $\mathcal{R}$ is a $\mathcal{M}_2$-bisimulation, $\mathcal{R} \subseteq \approx_2$. Thus for $\mathcal{M}_2$-terms P, P' we have $P \approx_1 P' \Rightarrow (P, P') \in \mathcal{R} \Rightarrow P \approx_2 P'$. Theorem 9.10 follows. □

Now we can finally state the following result that derives security of COM with respect to $\mathcal{M}_{real}$ in any context (we state it generally, though):

Lemma 9.11 *Let $P, \mathcal{F}$ be $\mathcal{M}_{real}$-processes (representing a protocol and an ideal functionality, e.g., $P = \mathsf{COM}$ and $\mathcal{F} = \mathcal{F}_{COM}$). Let $\mathcal{M}_{virt}$ be a safe extension of $\mathcal{M}_{real}$. Assume that $P \leq_{virt} \mathcal{F}$.*

Let C be an $\mathcal{M}_{real}$-context whose hole is protected only by νio for IO-names io, by parallel compositions, and by $!$, and that does not contain any NET-names in $fn(P, \mathcal{F})$. Assume that $C[\mathcal{F}] \leq_{virt} \mathcal{G}$ for some $\mathcal{M}_{real}$-process $\mathcal{G}$.

Let E_1, E_2 be $\mathcal{M}_{real}$-contexts satisfying the conditions of Theorem 6.1 (property preservation).

If $E_1[\mathcal{G}] \approx_{virt} E_2[\mathcal{G}]$ then $E_1[C[P]] \approx_{real} E_2[C[P]]$.

Proof. By the composition theorem (Theorem 5.37), $P \leq_{virt} \mathcal{F}$ implies $C[P] \leq_{virt} C[\mathcal{F}]$. With transitivity and $C[\mathcal{F}] \leq_{virt} \mathcal{G}$, this implies $C[P] \leq_{virt} \mathcal{G}$. Then by the property preservation theorem (Theorem 6.1), $E_1[\mathcal{G}] \approx_{virt} E_2[\mathcal{G}]$ implies $E_1[C[P]] \approx_{virt} E_2[C[P]]$. Since $\mathcal{M}_{virt}$ is a safe extension of $\mathcal{M}_{real}$, this implies $E_1[C[P]] \approx_{real} E_2[C[P]]$ by Theorem 9.10. □

9.3 On removing the CRS

Using virtual primitives, we have managed to get rid of the need for trapdoors in our commitment protocol. However, we still use a common reference string. This leads to the question whether the CRS can also be removed from the protocol. We do not answer that question here, but we give some indications as to how it might be possible to remove the CRS, also.

First, the question is whether we can construct a UC secure commitment protocol without using a CRS in the first place (i.e., instead of the protocol from Section 9.1). We know that this is impossible in the computational UC setting (no matter what primitives we use) [CF01]. Unfortunately, their impossibility result carries over to the symbolic setting:

Lemma 9.12 *There are no closed processes A, B and NET-names $\underline{net}$ with the following three properties:*

(i) $\nu\underline{net}.(A|B) \leq \mathcal{F}_{COM}$. *(Uncorrupted case.)*
(ii) $A \leq \mathcal{F}_{COM}\{\frac{net_{comb}}{io_{comb}}, \frac{net_{openb}}{io_{openb}}\}$. *(Bob corrupted.)*
(iii) $B \leq \mathcal{F}_{COM}\{\frac{net_{coma}}{io_{coma}}, \frac{net_{opena}}{io_{opena}}\}$. *(Alice corrupted.)*

Thus, a UC secure commitment protocol has to be of the form $\nu\underline{net}.(A|B|\mathcal{F})$ for some functionality $\mathcal{F}$, e.g., $\mathcal{F}_{CRS}$.

Proof. Assume that there are such processes A, B and NET-names $\underline{net}$.

Then there are simulators $(S_0, \varphi_0, \underline{n}_0)$, $(S_A, \varphi_A, \underline{n}_A)$, and $(S_B, \varphi_B, \underline{n}_B)$ such that

$$\nu\underline{net}.(A|B) \approx \nu\underline{n}_0.(\mathcal{F}_{COM}\varphi_0|S_0) = \nu\underline{n}_0.(\mathcal{F}_{COM}|S_0) \tag{5}$$

$$A \approx \nu\underline{n}_A.(\mathcal{F}_{COM}\{\tfrac{net_{comb}}{io_{comb}}, \tfrac{net_{openb}}{io_{openb}}\}\varphi_A|S_A) = \nu\underline{n}_A.(\mathcal{F}_{COM}\{\tfrac{net'_{comb}}{io_{comb}}, \tfrac{net'_{openb}}{io_{openb}}\}|S_A) \tag{6}$$

$$B \approx \nu\underline{n}_B.(\mathcal{F}_{COM}\{\tfrac{net_{coma}}{io_{coma}}, \tfrac{net_{opena}}{io_{opena}}\}\varphi_B|S_B) = \nu\underline{n}_B.(\mathcal{F}_{COM}\{\tfrac{net'_{coma}}{io_{coma}}, \tfrac{net'_{opena}}{io_{opena}}\}|S_B) \tag{7}$$

for suitable names $net'_{coma}, net'_{opena}, net'_{comb}, net'_{openb}$. The equalities use the fact that $\mathcal{F}_{COM}$ does not contain any NET-names.

Let

$$E := \nu io_{coma} io_{comb} io_{opena} io_{openb}.\\ \Big(\Big(\nu r.\big(\overline{io_{coma}}\langle r\rangle|io_{comb}().(\overline{io_{opena}}\langle\rangle|io_{openb}(x).\text{if } x = r \text{ then } \overline{c}\langle\rangle)\big)\Big)|\Box\Big)$$

where c is a fresh name. Intuitively, this context commits to a fresh nonce r, waits until the commit succeeds, then opens the commitment and checks whether the unveiled value is indeed r. For a "good" commitment scheme, this should always be the case. Indeed: By definition of $\mathcal{F}_{COM}$ (and using that $\underline{n}_0$ does not contain IO-names), we have that $E[\nu\underline{n}_0.(\mathcal{F}_{COM}|S_0)] \to^*\downarrow_c$. By (5) we have $E[\nu\underline{net}.(A|B)] \approx E[\nu\underline{n}_0.(\mathcal{F}_{COM}|S_0)]$ and thus $E[\nu\underline{net}.(A|B)] \to^*\downarrow_c$.

We now use (6) and (7) to transform $E[\nu\underline{net}.(A|B)]$ into a process that does not use the commitment protocol $A|B$ any more, but instead uses *two* instances of $\mathcal{F}_{COM}$:

$$E[\nu\underline{net}.(A|B)] \overset{(6,7)}{\approx} E[\nu\underline{net}.(\nu\underline{n}_A.(\mathcal{F}_{COM}\{\tfrac{net'_{comb}}{io_{comb}}, \tfrac{net'_{openb}}{io_{openb}}\}|S_A)|\nu\underline{n}_B.(\mathcal{F}_{COM}\{\tfrac{net'_{coma}}{io_{coma}}, \tfrac{net'_{opena}}{io_{opena}}\}|S_B))]$$

By moving all restrictions up (and potentially renaming names to avoid clashes of bound variables), we get:

$$E[\nu\underline{net}.(A|B)] \approx \nu\underline{net'}.E[\mathcal{F}_{COM}\{\tfrac{net''_{comb}}{io_{comb}}, \tfrac{net''_{openb}}{io_{openb}}\}|\mathcal{F}_{COM}\{\tfrac{net''_{coma}}{io_{coma}}, \tfrac{net''_{opena}}{io_{opena}}\}|S_{AB}] =: P$$

Here $\underline{net'}$ is the list of all names that were moved up. net''_{coma} etc are potentially renamed names, and $S_{AB} := S_A|S_B$ potentially up to renamings. Note that S_{AB} does not contain IO-names.

We now use several application of Lemma 3.3 to simplify P. Each of the following observational equivalences corresponds to one application of Lemma 3.3.

$$
\begin{aligned}
P \equiv\ & \nu \underline{net}\, io_{coma} io_{comb} io_{opena} io_{openb} r. \\
& \overline{io_{coma}}\langle r\rangle \mid io_{comb}().(\overline{io_{opena}}\langle\rangle \mid io_{openb}(x).\text{if } x = r \text{ then } \overline{c}\langle\rangle) \\
& \mid \mathcal{F}_{COM}\{\tfrac{net''_{coma}}{io_{coma}}, \tfrac{net''_{opena}}{io_{opena}}\} \mid \mathcal{F}_{COM}\{\tfrac{net''_{comb}}{io_{comb}}, \tfrac{net''_{openb}}{io_{openb}}\} \mid S_{AB} \\
= \ & \nu \underline{net}\, io_{coma} io_{comb} io_{opena} io_{openb} r. \\
& \overline{io_{coma}}\langle r\rangle \mid io_{comb}().(\overline{io_{opena}}\langle\rangle \mid io_{openb}(x).\text{if } x = r \text{ then } \overline{c}\langle\rangle) \\
& \mid \mathcal{F}_{COM}\{\tfrac{net''_{coma}}{io_{coma}}, \tfrac{net''_{opena}}{io_{opena}}\} \mid io_{coma}(x_m).(\overline{net''_{comb}}\langle\rangle \mid io_{opena}().\overline{net''_{openb}}\langle x_m\rangle) \mid S_{AB} \\
\overset{(i)}{\approx}\ & \nu \underline{net}\, io_{comb} io_{opena} io_{openb} r. \\
& \overline{net''_{comb}}\langle\rangle \mid io_{opena}().\overline{net''_{openb}}\langle r\rangle \mid io_{comb}().(\overline{io_{opena}}\langle\rangle \mid io_{openb}(x).\text{if } x = r \text{ then } \overline{c}\langle\rangle) \\
& \mid \mathcal{F}_{COM}\{\tfrac{net''_{coma}}{io_{coma}}, \tfrac{net''_{opena}}{io_{opena}}\} \mid S_{AB} \\
\overset{(ii)}{\approx}\ & \nu \underline{net}\, io_{comb} io_{openb} r. \\
& \overline{net''_{comb}}\langle\rangle \mid io_{comb}().(\overline{net''_{openb}}\langle r\rangle \mid io_{openb}(x).\text{if } x = r \text{ then } \overline{c}\langle\rangle) \\
& \mid \mathcal{F}_{COM}\{\tfrac{net''_{coma}}{io_{coma}}, \tfrac{net''_{opena}}{io_{opena}}\} \mid S_{AB} \\
= \ & \nu \underline{net}\, io_{comb} io_{openb} r. \\
& \overline{net''_{comb}}\langle\rangle \mid io_{comb}().(\overline{net''_{openb}}\langle r\rangle \mid io_{openb}(x).\text{if } x = r \text{ then } \overline{c}\langle\rangle) \\
& \mid net''_{coma}(x_m).(\overline{io_{comb}}\langle\rangle \mid net''_{opena}().\overline{io_{openb}}\langle x_m\rangle) \mid S_{AB} \\
\overset{(iii)}{\approx}\ & \nu \underline{net}\, io_{openb} r.\ \overline{net''_{comb}}\langle\rangle \\
& \mid net''_{coma}(x_m).(\overline{net''_{openb}}\langle r\rangle \mid io_{openb}(x).\text{if } x = r \text{ then } \overline{c}\langle\rangle \mid net''_{opena}().\overline{io_{openb}}\langle x_m\rangle) \mid S_{AB} \\
\overset{(iv)}{\approx}\ & \nu \underline{net}\, r.\ \overline{net''_{comb}}\langle\rangle \\
& \mid net''_{coma}(x_m).(\overline{net''_{openb}}\langle r\rangle \mid net''_{opena}().\text{if } x_m = r \text{ then } \overline{c}\langle\rangle) \mid S_{AB}
\end{aligned}
$$

Here (i) uses Lemma 3.3 with $n := io_{coma}$, $t := r$, and $x := x_m$.

And (ii) uses Lemma 3.3 with $n := io_{opena}$.

And (iii) uses Lemma 3.3 with $n := io_{comb}$.

And (iv) uses Lemma 3.3 with $n := io_{openb}$, $t := x_m$, and $x := x$ (and Lemma 3.2 (ii) to move the νio_{openb} below the $net''_{coma}(x_m)$ first, and Lemma 2.7, so that we can apply Lemma 3.3 to a subprocess instead of the whole process.)

Thus we have

$$
\begin{aligned}
& E[\nu \underline{net}.(A|B)] \approx P \approx \\
& \nu \underline{net}\, r.\ \overline{net''_{comb}}\langle\rangle \mid net''_{coma}(x_m).(\overline{net''_{openb}}\langle r\rangle \mid net''_{opena}().\text{if } x_m = r \text{ then } \overline{c}\langle\rangle) \mid S_{AB} =: P_2
\end{aligned}
$$

Note that in P_2, x_m is received before the fresh nonce r is revealed. Thus we expect

that the comparison $x_m = r$ will always fail. Indeed:

$$P_2 \overset{(*)}{\equiv} \nu \underline{net}\ .\overline{net''_{comb}}\langle\rangle \mid net''_{coma}(x_m).\nu r.(\overline{net''_{openb}}\langle r\rangle \mid net''_{opena}().\text{if } x_m = r \text{ then } \overline{c}\langle\rangle) \mid S_{AB}$$
$$\overset{(**)}{\approx} \nu \underline{net}.\overline{net''_{comb}}\langle\rangle \mid net''_{coma}(x_m).\nu r.(\overline{net''_{openb}}\langle r\rangle \mid net''_{opena}().0) \mid S_{AB} =: P_3$$

Here $(*)$ uses Lemma 3.2 (ii) with $x := x_m$ to move the restriction νr down, and $(**)$ uses Lemma 3.8 to replace the if-statement by its else-branch (which is 0).

Thus we have that $E[\nu \underline{net}.(A|B)] \approx P_2 \approx P_3$. Furthermore, we showed above that $E[\nu \underline{net}.(A|B)] \to^* \downarrow_c$. But since c does not occur in P_3 (we chose it as a fresh name, thus it also does not occur in S_{AB}), we have that $P_3 \to^* \downarrow_c$ cannot hold. This is a contradiction to the observational equivalence $E[\nu \underline{net}.(A|B)] \approx P_3$. Thus our assumption was wrong that processes A, B and NET-names $\underline{net}$ as in the statement of the lemma exist. □

However, Lemma 9.12 does not exclude that an approach similar to the virtual primitives approach might work: We first construct a UC secure commitment protocol (again, commitments are just one example), build a complex protocol from it using the composition theorem, and then show that security of the complex protocol implies (non-UC) security of a modification that does not use the CRS. It is likely that this works as the CRS returned by the CRS functionality is just a fresh public name, so instead of the CRS we should be able to just use some fresh (non-restricted) name a.

There is one subtlety, though: When composing the commitment protocol P, we end up with a complex protocol $C[P]$ that may use multiple instances of $\mathcal{F}_{CRS}$. In particular, if $C[P]$ contains $!!P$, then $C[P]$ will contain an unbounded number of $\mathcal{F}_{CRS}$-instances. So we cannot replace $\mathcal{F}_{CRS}$ just by a single name, we will need a way to generate an arbitrary number of fresh values. The obvious way for this is to use something like $hash(a, sid)$ instead of the CRS that we get from the $\mathcal{F}_{CRS}$-instance with session-id sid (here a is a fresh name).

A lemma roughly like the following conjecture should therefore lead to a method for removing the CRS from a protocol that was produced by UC composition:

Conjecture 9.13 *Let hash be a free constructor (i.e., not occurring in any equations or rewrite rules in the symbolic models). Let P be a process. Let E_1, E_2 be contexts. Assume that hash does not occur in E_1, E_2, P. Let $a \notin fn(E_1, E_2, P) \cup bn(E_1, E_2, P)$.*

(i) Let P' result from P by replacing all subterms "$net_{crsa}(x).Q$" by "let $x = a$ in Q". Then $E_1[\nu net_{crsa}.(P|\mathcal{F}_{CRS})] \approx\!\!\approx E_2[\nu net_{crsa}.(P|\mathcal{F}_{CRS})]$ implies $E_1[\nu net_{crsa}.(P')] \approx\!\!\approx E_2[\nu net_{crsa}.(P')]$.

(ii) Let P' result from P by replacing all subterms "$(M_{sid}, net_{crsa})(x).Q$" by "let $x = hash(a, M_{sid})$ in Q". Then $E_1[\nu net_{crsa}.(P|!!\mathcal{F}_{CRS})] \approx\!\!\approx E_2[\nu net_{crsa}.(P|!!\mathcal{F}_{CRS})]$ implies $E_1[\nu net_{crsa}.(P')] \approx\!\!\approx E_2[\nu net_{crsa}.(P')]$.

Proving (i) is probably considerably simpler than proving (ii). An alternative to proving (ii) could be to make sure that $C[P]$ does not contain $\mathcal{F}_{CRS}$ under a $!!$. This could be achieved if we design a commitment protocol P that does not implement $\mathcal{F}_{COM}$, but $!!\mathcal{F}_{COM}$ (compare with Section 8.3). Then a single copy of P would be sufficient in $C[P]$.

We leave further exploration of approaches to get rid of the CRS to future research.

```
fun empty/0.

free net2, net3.

let Q = new n; out(io1,n) |
    (in(io2,x); if x=n then out(net2,empty)) |
    (in(io3,x); if x=n then out(net3,empty)).

process new io1; new io2; new io3; in(io1,x1); in(io1,x2);
                                   out(io2,x1) | out(io3,choice[x1,x2]) | !Q
```

Figure 16: Proverif code for showing $E_1[Q] \approx E_2[Q]$ in Lemma A.1 (`prop-pres-bang1.pv`, see [BU13]).

A Limits for composition and property preservation

In this section, we show that the restrictions of the composition theorem are necessary. More precisely, we show that if $P \leq Q$, then not necessarily $!P \leq !Q$ or $io(x).P \leq io(x).Q$ or $\overline{io}\langle t\rangle.P \leq \overline{io}\langle t\rangle.Q$ or $\nu net.P \leq \nu net.Q$ or $P|R \leq Q|R$ (for R that has NET-names in common with P, Q). We show that this is not just a limitation of the composition theorem, we show that similar limitations also apply to property preservation. More precisely, property preservation $P \leq Q, E_1[Q] \approx E_2[Q] \implies E_1[P] \approx E_2[P]$ does not necessarily hold if E_1, E_2 contain a bang (!) over their hole, or an input/output over their hole, or an if/let over their hole, or a different number of !!'s over their respective holes, or restrict NET-names over their holes, or use NET-names.

Example A.1

$$P := \nu n\, m.\ \overline{io_1}\langle n\rangle \mid io_2(x).\mathit{if}\ x = n\ \mathit{then}\ \overline{net_2}\langle m\rangle \mid io_3(x).\mathit{if}\ x = n\ \mathit{then}\ \overline{net_3}\langle m\rangle$$
$$Q := \nu n\ .\ \overline{io_1}\langle n\rangle \mid io_2(x).\mathit{if}\ x = n\ \mathit{then}\ \overline{net_2}\langle \mathit{empty}\rangle \mid io_3(x).\mathit{if}\ x = n\ \mathit{then}\ \overline{net_3}\langle \mathit{empty}\rangle$$
$$E_1 := \nu io_1\, io_2\, io_3.\ io_1(x_1).io_1(x_2).(\overline{io_2}\langle x_1\rangle \mid \overline{io_3}\langle x_1\rangle) \mid !\Box$$
$$E_2 := \nu io_1\, io_2\, io_3.\ io_1(x_1).io_1(x_2).(\overline{io_2}\langle x_1\rangle \mid \overline{io_3}\langle x_2\rangle) \mid !\Box$$

Lemma A.1 *Using the notation from Example A.1, we have $P \leq Q$, and $E_1[Q] \approx E_2[Q]$, but $E_1[P] \not\approx E_2[P]$.*

Proof. We show $P \leq Q$: We have $P \approx \nu net_2' net_3'.(Q\{\frac{net_2'}{net_2}, \frac{net_3'}{net_3}\}|S)$ for $S := \nu m.(net_2'(x).\overline{net_2}\langle m\rangle | net_3'(x).\overline{net_3}\langle m\rangle)$ by two invocations of Lemma 3.3 (first with $n := net_2'$, $x := x$, and $t := \mathit{empty}$, second with $n := net_3'$, $x := x$, and $t := \mathit{empty}$). Hence $P \leq Q$.

The claim $E_1[Q] \approx E_2[Q]$ is shown using Proverif. The Proverif code is given in Figure 16

```
fun empty/0.

free net2, net3.
private free c.

query mess:c,c.

let P = new n; new m; out(io1,n) |
    (in(io2,x); if x=n then out(net2,m)) | (in(io3,x); if x=n then out(net3,m)).

let E2P = new io1; new io2; new io3; in(io1,x1); in(io1,x2);
                                     out(io2,x1) | out(io3,x2) | !P.

let D = in(net2,y1); in(net3,y2); if y1=y2 then out(c,empty).

process D | E2P
```

Figure 17: Proverif code for showing that $D|E_2[P] \rightarrow^*\downarrow_c$ does not hold in the proof of Lemma A.1 (`prop-pres-bang2.pv`, see [BU13]).

We now show $E_1[P] \not\approx E_2[P]$. Let $D := net_2(y_1).net_3(y_2).\text{if } y_1 = y_2 \text{ then } \overline{c}\langle empty\rangle$. Then $D \mid E_1[P] \rightarrow^* D \mid \cdots \mid \nu m.(\overline{net_2}\langle m\rangle \mid \overline{net_3}\langle m\rangle) \rightarrow^* \nu m.(\cdots \mid \text{if } m = m \text{ then } \overline{c}\langle\rangle) \rightarrow^*\downarrow_c$. Using Proverif, we show that $D \mid E_2[P] \rightarrow^*\downarrow_c$ does not hold (for any context D not containing c). The Proverif code is given in Figure 17. $E_1[P] \approx E_2[P]$ would imply $D \mid E_1[P] \approx D \mid E_2[P]$ which together with $D \mid E_1[P] \rightarrow^*\downarrow_c$ would imply the wrong fact $D \mid E_2[P] \rightarrow^*\downarrow_c$. Thus $E_1[P] \not\approx E_2[P]$.

□

Lemma A.2 *Using the notation from Example A.1, we have $P \leq Q$ but not $!P \leq !Q$.*

Proof. From Lemma A.1 we have $P \leq Q$ and $E_1[Q] \approx E_2[Q]$. Assume $!P \leq !Q$. We can write $E_1 = E_1'[!\Box]$ and $E_2 = E_2'[!\Box]$ for NET-free evaluation contexts E_1, E_2. Then $E_1'[!Q] = E_1[Q] \approx E_2[Q] = E_2'[!Q]$ and thus by Theorem 6.1, we have $E_1[P] = E_1'[!P] \approx E_2'[!P] = E_2[P]$. This is a contradiction to Lemma A.1. Thus the assumption $!P \leq !Q$ was wrong. □

Example A.2

$$\begin{aligned} P &:= \overline{net}\langle empty\rangle \\ Q &:= 0 \\ E_1 &:= \nu io.\ (io().\Box \mid \overline{io}\langle empty\rangle) \\ E_2 &:= \nu io.\ (io().\Box) \end{aligned}$$

Lemma A.3 *Using the notation from Example A.2, we have $P \leq Q$, and $E_1[Q] \approx E_2[Q]$, but $E_1[P] \not\approx E_2[P]$.*

Proof. Obviously, $P \approx Q|S$ with $S := \overline{net}\langle empty\rangle$. Hence $P \leq S$.

We show $E_1[Q] \approx E_2[Q]$: We have $E_1[Q] = \nu io.\ (io().0 \mid \overline{io}\langle empty\rangle) \approx 0$ by Lemma 3.3 with $n := io$ and $C := \Box$. And $E_2[Q] = \nu io.io().0 \approx 0$ by Lemma 3.3 with $n := io$ and $C := 0$. Hence $E_1[Q] \approx E_2[Q]$.

We show $E_1[P] \not\approx E_2[P]$: We have $E_1[P] \to^* \nu io.\overline{net}\langle empty\rangle \downarrow_{net}$. But $E_2[P] \not\downarrow_{net}$, and $E_2[P]$ does not reduce. Thus there is no successor of $E_2[P]$ that emits on net. This contradicts $E_1[P] \approx E_2[P]$ by definition of observational equivalence. □

Lemma A.4 *Using the notation from Example A.2, we have $P \leq Q$ but not $io().P \leq io().Q$.*

Proof. From Lemma A.3 we have $P \leq Q$ and $E_1[Q] \approx E_2[Q]$. Assume $io().P \leq io().Q$. We can write $E_1 = E_1'[io().\Box]$ and $E_2 = E_2'[io().\Box]$ for NET-free evaluation contexts E_1, E_2. Then $E_1'[io().Q] = E_1[Q] \approx E_2[Q] = E_2'[io().Q]$ and thus by Theorem 6.1, we have $E_1[P] = E_1'[io().P] \approx E_2'[io().P] = E_2[P]$. This is a contradiction to Lemma A.3. Thus the assumption $io().P \leq io().Q$ was wrong. □

Example A.3 *Let P, Q be as in Example A.2.*

$$E_1 := \nu io.\ (\overline{io}\langle empty\rangle.\Box \mid io())$$
$$E_2 := \nu io.\ (\overline{io}\langle empty\rangle.\Box)$$

Lemma A.5 *Using the notation from Example A.3, we have $P \leq Q$, and $E_1[Q] \approx E_2[Q]$, but $E_1[P] \not\approx E_2[P]$.*

Lemma A.6 *Using the notation from Example A.3, we have $P \leq Q$ but not $\overline{io}\langle empty\rangle.P \leq \overline{io}\langle empty\rangle.Q$.*

The proofs of Lemmas A.5 and A.6 are identical to those of Lemmas A.5 and A.6, except that $io()$ and $\overline{io}\langle empty\rangle$ are exchanged.

Example A.4 *Let P, Q be as in Example A.2.*

$$E_1 := \mathit{if\ true\ then}\ \Box$$
$$E_2 := \mathit{if\ false\ then}\ \Box$$

Here true is an equality $t = t$ for an arbitrary closed t (e.g., $empty = empty$), and false is an equality $t = t'$ for arbitrary closed t, t' with $t \neq_{\mathrm{E}} t'$ (e.g., $empty = (empty, empty)$).

Remember that if $x = y$ is syntactic sugar for let $z = equals(x, y)$. So this example is a counterexample for let-statements.

Lemma A.7 *Using the notation from Example A.4, we have $P \leq Q$, and $E_1[Q] \approx E_2[Q]$, but $E_1[P] \not\approx E_2[P]$.*

Proof. $P \leq Q$ was already shown in Lemma A.3. By Lemma 3.2 (v) we have that $E_1[P] \approx P$ and $E_1[Q] \approx Q = 0$ and by Lemma 3.2 (v) we have that $E_1[P] \approx 0$ and $E_2[Q] \approx 0$. Obviously, $P \not\approx 0$. $E_1[P] \not\approx E_2[P]$, but $E_1[Q] \approx E_2[Q]$. □

Example A.5 *Let P, Q be as in Example A.2.*

$$E_1 := !!\Box$$
$$E_2 := \Box$$

Lemma A.8 *Using the notation from Example A.5, we have $P \leq Q$, and $E_1[Q] \approx E_2[Q]$, but $E_1[P] \not\approx E_2[P]$.*

Proof. $P \leq Q$ was already shown in Lemma A.3. Let $t \in SID$ be arbitrary. We have $E_1[P] \approx \prod_{x \in SID} \overline{(x, net)}\langle empty \rangle \rightarrow^* \downarrow_{(t,net)}$. But no successor of $E_2[P] = \overline{net}\langle empty \rangle$ emits on $(t, net) \neq_{\mathrm{E}} net$. Thus $E_1[P] \not\approx E_2[P]$.

It is easy to see that $0 \approx \prod_{x \in SID} 0$ (by showing that $\mathcal{R} := \{(R, R | \prod_{x \in SID \setminus S} 0)\}$ up to structural equivalence is a bisimulation). Thus

$$E_1[Q] = !!0 \approx \prod_{x \in SID} 0 \approx 0 = E_2[Q].$$

□

Example A.6

$$P := net().io().\overline{io'}\langle\rangle$$
$$Q := net'().io().\overline{io'}\langle\rangle$$
$$E_1 := \nu io.(\overline{io}\langle\rangle \mid \nu net'.\Box)$$
$$E_2 := \nu io.(\nu net'.\Box)$$

Lemma A.9 *Using the notation from Example A.6, we have $P \leq Q$, and $E_1[Q] \approx E_2[Q]$, but $E_1[P] \not\approx E_2[P]$.*

Proof. $P \leq Q$ holds with simulator $S := 0$, $\varphi := (net' \mapsto net)$, $\underline{n} := \varnothing$.

It is easy to see that $\nu net'.Q \approx 0$. Hence $E_1[Q] \approx \nu io.\overline{io}\langle\rangle$ and $E_2[Q] \approx \nu io.0$. Thus $E_1[Q] \approx E_2[Q]$.

But $E_1[P] \rightarrow^* \downarrow_{io'}$ and $E_2[P] \not\rightarrow^* \downarrow_{io'}$. Hence $E_1[P] \not\approx E_2[P]$. □

Lemma A.10 *Using the notation from Example A.1, we have $P \leq Q$ but not $\nu net'.P \leq \nu net'.Q$.*

Proof. From Lemma A.9 we have $P \leq Q$ and $E_1[Q] \approx E_2[Q]$. Assume $\nu net'.P \leq \nu net'.Q$. We can write $E_1 = E_1'[\nu net'.\Box]$ and $E_2 = E_2'[\nu net'.\Box]$ for NET-free evaluation contexts E_1, E_2. Then $E_1'[\nu net'.Q] = E_1[Q] \approx E_2[Q] = E_2'[\nu net'.Q]$ and thus by Theorem 6.1, we have $E_1[P] = E_1'[\nu net'.P] \approx E_2'[\nu net'.P] = E_2[P]$. This is a contradiction to Lemma A.9. Thus the assumption $\nu net'.P \leq \nu net'.Q$ was wrong. □

Example A.7

$$\begin{aligned} P &:= io().\overline{net}\langle\rangle \\ Q &:= io().\overline{net'}\langle\rangle \\ E_1 &:= \nu io.(\overline{io}\langle\rangle \mid \Box \mid !\overline{net'}\langle\rangle) \\ E_2 &:= (\nu io.\Box \mid !\overline{net'}\langle\rangle) \end{aligned}$$

Lemma A.11 *Using the notation from Example A.6, we have $P \leq Q$, and $E_1[Q] \approx E_2[Q]$, but $E_1[P] \not\approx E_2[P]$.*

Proof. $P \leq Q$ holds with simulator $S := 0$, $\varphi := (net' \mapsto net)$, $\underline{n} := \varnothing$.

By Lemma 3.3, we have $E_1[Q] \approx \overline{net'}\langle\rangle \mid !\overline{net'}\langle\rangle$. And by Lemma 3.2 (viii), $\overline{net'}\langle\rangle \mid !\overline{net'}\langle\rangle \approx !\overline{net'}\langle\rangle$. Finally $E_2[Q] \approx 0 \mid !\overline{net'}\langle\rangle$. Hence $E_1[Q] \approx E_2[Q]$.

But $E_1[P] \rightarrow^* \downarrow_{net}$ and $E_2[P] \not\rightarrow^* \downarrow_{net}$. Hence $E_1[P] \not\approx E_2[P]$. □

Lemma A.12 *Using the notation from Example A.1, we have $P \leq Q$ but not $P \mid !\overline{net'}\langle\rangle \leq Q \mid !\overline{net'}\langle\rangle$.*

Proof. From Lemma A.11 we have $P \leq Q$ and $E_1[Q] \approx E_2[Q]$. Assume $P \mid !\overline{net'}\langle\rangle \leq Q \mid !\overline{net'}\langle\rangle$. We can write $E_1 = E_1'[\Box \mid !\overline{net'}\langle\rangle]$ and $E_2 = E_2'[\Box \mid !\overline{net'}\langle\rangle]$ for NET-free evaluation contexts E_1, E_2. Then $E_1'[Q \mid !\overline{net'}\langle\rangle] = E_1[Q] \approx E_2[Q] = E_2'[Q \mid !\overline{net'}\langle\rangle]$ and thus by Theorem 6.1, we have $E_1[P] = E_1'[P \mid !\overline{net'}\langle\rangle] \approx E_2'[P \mid !\overline{net'}\langle\rangle] = E_2[P]$. This is a contradiction to Lemma A.11. Thus the assumption $P \mid !\overline{net'}\langle\rangle \leq Q \mid !\overline{net'}\langle\rangle$ was wrong. □

Acknowledgement. Florian Böhl was supported by MWK grant "MoSeS". Dominique Unruh was supported by the European Union through the European Regional Development Fund through the sub-measure "Supporting the development of R&D of info and communication technology", by the European Social Fund's Doctoral Studies and Internationalisation Programme DoRa, and by the Estonian Centre of Excellence in Computer Science, EXCS.

References

[ACRR10] Myrto Arapinis, Tom Chothia, Eike Ritter, and Mark Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *CSF 2010, 23rd IEEE Computer Security Foundations Symposium*, pages 107–121. IEEE, 2010.

[AF01] Martin Abadi and Cedric Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 104–115. ACM New York, NY, USA, 2001.

[BAF08] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 75:3–51, 2008. Online available at `http://www.di.ens.fr/~blanchet/publications/BlanchetAbadiFournetJLAP07.pdf`.

[BBF+11] Jesper Bengtson, Karthikeyan Bhargavan, Cédric Fournet, Andrew D. Gordon, and Sergio Maffeis. Refinement types for secure implementations. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 33:8:1–8:45, 2011.

[BCC04] Ernest F. Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *Proc. 11th ACM Conference on Computer and Communications Security*, pages 132–145. ACM Press, 2004.

[Bla04] Bruno Blanchet. Automatic Proof of Strong Secrecy for Security Protocols. Technical Report MPI-I-2004-NWG1-001, Max-Planck-Institut für Informatik, Saarbrücken, Germany, July 2004.

[Bla09] Bruno Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4):363–434, 2009. Preprint available as arXiv:0802.3444v1 [cs.CR].

[Bla12a] Bruno Blanchet. Personal communication, August 2012.

[Bla12b] Bruno Blanchet. Proverif 1.86pl4: Automatic cryptographic protocol verifier - user manual and tutorial. `http://prosecco.gforge.inria.fr/personal/bblanche/proverif/manual.pdf`, 2012.

[BPW03] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A composable cryptographic library with nested operations. In *Proc. 10th ACM CCS*, pages 220–230, 2003.

[BPW07] Michael Backes, Birgit Pfitzmann, and Michael Waidner. The reactive simulatability (RSIM) framework for asynchronous systems. *Information and Computation*, 205(12):1685–1720, 2007.

[BS05] Boaz Barak and Amit Sahai. How to play almost any mental game over the net — concurrent composition via super-polynomial simulation. In *Proc. 46th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 543–552, 2005.

[BU13] Florian Böhl and Dominique Unruh, 2013. Proverif examples from the present paper: `http://boehl.name/publications/symbolic-uc/proverif-files.zip`.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001. Extended version in Cryptology ePrint Archive, Report 2000/67, `http://eprint.iacr.org/`.

[CB13] Vincent Cheval and Bruno Blanchet. Proving more observational equivalences with proverif. In David Basin and John Mitchell, editors, *POST 2013*, volume 7796 of *LNCS*, pages 226–246. Springer, 2013.

[CCK+06a] Ran Canetti, Ling Cheung, Dilsun Kaynar, Moses Liskov, Nancy Lynch, Olivier Pereira, and Roberto Segala. Task-structured probabilistic I/O automata. Technical Report MIT-CSAIL-TR-2006-060, MIT CSAIL, September 2006. Online available at `http://dspace.mit.edu/handle/1721.1/33964`.

[CCK+06b] Ran Canetti, Ling Cheung, Dilsun Kirli Kaynar, Moses Liskov, Nancy A. Lynch, Olivier Pereira, and Roberto Segala. Time-bounded task-PIOAs: A framework for analyzing security protocols. In *DISC*, pages 238–253, 2006.

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In *Proc. 4th Theory of Cryptography Conference (TCC)*, pages 61–85, 2007.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *Advances in Cryptology, Proceedings of CRYPTO 2001*, number 2139 in Lecture Notes in Computer Science, pages 19–40. Springer-Verlag, 2001. Full version online available at `http://eprint.iacr.org/2001/055.ps`.

[CH11] Ran Canetti and Jonathan Herzog. Universally composable symbolic security analysis. *J Cryptology*, 24(1):83–147, January 2011.

[Che66] Cher. Bang bang (my baby shot me down). 7" single, 1966. Written by Sonny Bono, sound sample: `http://en.wikipedia.org/wiki/File:Cher-_Bang_Bang_My_Baby_Shot_Me_Down.ogg`.

[CR03] Ran Canetti and Tal Rabin. Universal composition with joint state. In *Proc. CRYPTO 2003*, volume 2729 of *LNCS*, pages 265–281. Springer, 2003.

[CV12] Ran Canetti and Margarita Vald. Universally composable security with local adversaries. In Ivan Visconti and Roberto De Prisco, editors, *SCN 2012*, volume 7485 of *Lecture Notes in Computer Science*, pages 281–301. Springer, 2012.

[DKMR05] Anupam Datta, Ralf Küsters, John C. Mitchell, and Ajith Ramanathan. On the relationships between notions of simulation-based security. In Joe Kilian, editor, *Theory of Cryptography, Proceedings of TCC 2005*, Lecture Notes in Computer Science, pages 476–494. Springer-Verlag, 2005.

[DKP] Stephanie Delaune, Steve Kremer, and Olivier Pereira. Simulation based security in the applied pi calculus. IACR ePrint 2009/267, version 5 June 2009. full version of [DKP09].

[DKP09] Stephanie Delaune, Steve Kremer, and Olivier Pereira. Simulation based security in the applied pi calculus. In Ravi Kannan and K. Narayan Kumar, editors, *FSTTCS*, volume 4 of *LIPIcs*, pages 169–180. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2009.

[DY81] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols (extended abstract). In *FOCS*, pages 350–357. IEEE, 1981.

[HS11] Dennis Hofheinz and Victor Shoup. GNUC: A new universal composability framework. IACR ePrint 2011/303, 2011.

[KDMR08] Ralf Küsters, Anupam Datta, John C. Mitchell, and Ajith Ramanathan. On the relationships between notions of simulation-based security. *Journal of Cryptology*, 2008. To appear. Electronic publication http://dx.doi.org/10.1007/s00145-008-9019-9.

[Küs06] Ralf Küsters. Simulation-based security with inexhaustible interactive Turing machines. In *CSFW 2006, Computer Security Foundations Workshop*, pages 309–320. IEEE Computer Society, 2006. Long version available as IACR eprint 2006/151.

[Low95] Gavin Lowe. An attack on the needham-schroeder public-key authentication protocol. *Information Processing Letters*, 56:131–133, November 1995.

[MQU07] Jörn Müller-Quade and Dominique Unruh. Long-term security and universal composability. In *Theory of Cryptography, Proceedings of TCC 2007*, volume 4392 of *Lecture Notes in Computer Science*, pages 41–60. Springer-Verlag, March 2007. Preprint on IACR ePrint 2006/422, superseeded by [MQU07].

[PS04] Manoj Prabhakaran and Amit Sahai. New notions of security: Achieving universal composability without trusted setup. In *Proc. 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 242–251, 2004.

[UMQ10] Dominique Unruh and Jörn Müller-Quade. Universally composable incoercibility. In *Crypto 2010*, volume 6223 of *LNCS*, pages 411–428. Springer, August 2010. Preprint on IACR ePrint 2009/520.

[Unr10] Dominique Unruh. Universally composable quantum multi-party computation. In *Eurocrypt 2010*, LNCS, pages 486–505. Springer, 2010. Preprint on arXiv:0910.2912 [quant-ph].

[Unr11] Dominique Unruh. Concurrent composition in the bounded quantum storage model. In *Eurocrypt 2011*, volume 6632 of *LNCS*, pages 467–486. Springer, May 2011. Preprint on IACR ePrint 2010/229.

Symbol index

$C().P$	Syntactic sugar for $C(x).P$ with fresh variable x	9
$\overline{C}\langle\rangle.P$	Syntactic sugar for $C(\mathit{empty}).P$	9
equals	Destructor equals	10
fst	Destructor: Extracts the first component of a tag	10
snd	Destructor: Extracts the second component of a tag	10
$\equiv_{\mathrm{E}}$	Structural equivalence modulo equational theory E	11
$P \approx Q$	Full observational equivalence of the non-closed processes P and Q	11
$\prod_{x \in S} P$	Indexed replication of the process P	13
$\{a/b\}$	Substitution replacing b with a	14
$\not\cong$	Asymmetric variant of structural equivalence	21
$\not\cong_E$	$\not\cong$ modulo equational theory	21
event $f(t)$	Raise event $f(t)$	24
$\mathit{plain}^s(P)$	P with synchronization channel s removed	26
$\mathit{ev}^s(P)$	P with synchronization channel s replaced by events	26
$\mathit{syncout}^s(t_1 \mapsto t_1', \ldots; u_1 \mapsto u_1', \ldots)$	Outputs on synchronization channel s	26
IO	Set of all I/O names	32
NET	Set of all network names	32
$P \leq Q$	P emulates Q	33
$P(\!(M)\!)$	Process P with session-id M	37
SID	Set of all session IDs	38
$\mathcal{C}^{SID}_{x,\underline{n}}$	An arbitrary but fixed *SID*-indexing context	38
nil	Constructor denoting the empty bitstring	38
zero	Constructor prefixing a bitstring with 0	38
one	Constructor prefixing a bitstring with 1	38
SID_{bits}	Concrete set of session IDs built from bitstrings	38
$\mathcal{C}^{\mathit{SID}_{bits}}_{x,a}$	A concrete fixed SID_{bits}-indexing context	38
$\mathcal{G}^n_{x,a}$	Auxiliary definition in analysis of $\mathcal{C}^{\mathit{SID}_{bits}}_{x,a}$	38
$\mathcal{C}^{(sID,gID,n)}_{x,a}$	Auxiliary definition in analysis of $\mathcal{C}^{\mathit{SID}_{bits}}_{x,a}$	38
sID	Auxiliary definition in analysis of $\mathcal{C}^{\mathit{SID}_{bits}}_{x,a}$ – set of spawned IDs	38
gID	Auxiliary definition in analysis of $\mathcal{C}^{\mathit{SID}_{bits}}_{x,a}$ – set of generator IDs	38
$\Sigma_{x \in S} P$	Short for $P\{s_1/x\} \mid P\{s_2/x\} \mid \ldots$ for $S = \{s_1, s_2, \ldots\}$	38
$\langle\blacksquare\rangle$	Span of a set of IDs	39
$!!P$	Concurrent composition of P with session ids	42
$\approx^n_{\mathcal{S}}$	Observational equivalence restricted to processes that do not contain n and contexts build from $\mathcal{S}$	44
n_{sid}	Fixed name for sid-sensitive processes	45

M_{sid}	Fixed term for sid-sensitive processes	45
$\mathcal{S}_{sid}$	The set of sid-sensitive processes	45
Φ	Transformation of a generic plain process into a sid-sensitive process	45
tag	Tag channel identifiers	48
untag	Untag channel identifiers	49
$\sim_{\mathcal{S}_{sid}}$	An $\mathcal{S}_{sid}$-observational equivalence relation	52
$\mathcal{F}_{SC}$	Secure channel functionality	61
$\preceq$	Observational preorder	62
$P \leq^{\mathsf{SS}} Q$	P emulates Q in the sense of Delaune et al. [DKP09].	62
$\mathcal{F}_{anon}$	Insecure but anonymous channel functionality	63
penc	Constructor: public key encryption	66
pk	Constructor: public key	66
sk	Constructor: secret key	66
senc	Constructor: symmetric encryption	66
hash	Constructor: hash function	66
empty	Constructor: empty message	66
pdec	Destructor: public key decryption	66
sdec	Destructor: symmetric decryption	66
pkofsk	Destructor extracting secret from public key	66
pkofenc	Destructor extracting public key from ciphertext	66
$\mathcal{F}_{KE}$	Key exchange functionality	67
$\mathcal{F}_{PKI}$	Public key infrastructure functionality	67
NSL	Needham-Schroeder-Lowe protocol	67
SC	Secure channel protocol	70
KE^*	Protocol for generating many keys	74
$\mathcal{F}_{COM}$	Commitment functionality	81
$\mathcal{M}_{virt}$	Symbolic model with virtual primitives	81
$\mathcal{M}_{real}$	Symbolic model without virtual primitives	81
crsext	Constructor: CRS for extraction	81
crseqv	Constructor: CRS for equivocation	81
fakeH	Constructor: Fake (equivocal) hash	81
fake	Constructor: Randomness for fake hash	81
extract	Destructor: Extracting from a hash	81
$\mathcal{F}_{CRS}$	Common reference string functionality	82
COM	Commitment protocol	82

Index