A Computationally Sound Mechanized Prover for Security Protocols

Bruno Blanchet*†

June 16, 2012

Abstract

We present a new mechanized prover for secrecy properties of security protocols. In contrast to most previous provers, our tool does not rely on the Dolev-Yao model, but on the computational model. It produces proofs presented as sequences of games; these games are formalized in a probabilistic polynomial-time process calculus. Our tool provides a generic method for specifying security properties of the cryptographic primitives, which can handle shared-key and public-key encryption, signatures, message authentication codes, and hash functions. Our tool produces proofs valid for a number of sessions polynomial in the security parameter, in the presence of an active adversary. We have implemented our tool and tested it on a number of examples of protocols from the literature.

1 Introduction

There exist two main approaches for analyzing security protocols. In the computational model, messages are bitstrings, and the adversary is a probabilistic polynomial-time Turing machine. This model is close to the real execution of protocols, but the proofs are usually manual and informal. In contrast, in the formal, Dolev-Yao model, cryptographic primitives are considered as perfect blackboxes, modeled by function symbols in an algebra of terms, possibly with equations. The adversary can compute using these blackboxes. This abstract model makes it possible to build automatic verification tools, but the security proofs are in general not sound with respect to the computational model.

Since the seminal paper by Abadi and Rogaway [3], there has been much interest in relating both frameworks (see for example [1, 9, 12, 24, 28, 29, 38, 39]), to show the soundness of the Dolev-Yao model with respect to the computational model, and thus obtain automatic proofs of protocols in the computational model. However, this approach has limitations: since the computational and Dolev-Yao models do not correspond exactly, additional hypotheses are necessary in order to guarantee soundness. (For example, key cycles have to be excluded, or a specific security definition of encryption is needed [5].)

In this paper, we propose a different approach for automatically proving protocols in the computational model: we have built a mechanized prover that works directly in the computational model, without considering the Dolev-Yao model. Our tool produces proofs valid for a number of sessions polynomial in the security parameter, in the presence of an active adversary. These proofs are presented as sequences of games, as used by cryptographers [17,45,46]: the initial game represents the protocol to prove; the goal is to show that the probability of breaking a certain security property (secrecy in this paper) is negligible in this game; intermediate games are obtained each from the previous one by transformations such that the difference of probability between consecutive games is negligible; the final game is such that the desired probability is obviously negligible from the form of the game. The desired probability is then negligible in the initial game.

We represent games in a process calculus. This calculus is inspired by the pi-calculus and by the calculi of [34, 35, 40] and of [33]. In this calculus, messages are bitstrings, and cryptographic primitives are functions from bitstrings to bitstrings. The calculus has a probabilistic semantics, and all processes run in polynomial time. The main tool for specifying security properties is observational equivalence: Q is observationally equivalent to Q', $Q \approx Q'$, when the adversary has a negligible probability of distinguishing Q from Q'. With respect to previous calculi mentioned above, our calculus introduces an important novelty which is key for the automatic proof of security protocols: the values of all variables during the execution of a process are stored in arrays. For instance, $x[i]$ is the value of x in the i-th copy of the process that defines x. Arrays replace lists often used by cryptographers in their manual proofs of protocols. For example, consider the definition of security of a message authentication code (MAC). Informally, this definition says that the adversary has a negligible probability of forging a MAC, that is, that all correct MACs have been computed by calling the MAC oracle. So, in cryptographic proofs, one defines a list containing the arguments of calls to the MAC oracle, and when checking a MAC of a message m, one can additionally check that m is in this list, with a negligible change in probability. In our calculus, the arguments of the MAC oracle are stored in arrays, and we perform a lookup in these arrays in order to find the message m. Arrays make it easier to automate proofs since they are always present in the calculus: one does not need to add explicit instructions to insert values in them, in contrast to the lists used in manual proofs. Therefore, many trivially sound but difficult to automate syntactic transformations disappear. Furthermore, relations between elements of arrays can easily be expressed by equalities, possibly involving computations on array indices.

Our prover relies on a collection of game transformations, in order to transform the initial protocol into a game on which the desired security property is obvious. The most important kind of transformations exploits the definition of security of crypto-

*B. Blanchet is with CNRS, École Normale Supérieure, Paris, France E-mail: blanchet@di.ens.fr

†A short version of this paper appears at IEEE Symposium on Security and Privacy, Oakland, California, May 2006.

graphic primitives in order to obtain a simpler game. As described in Section 3.2, these transformations can be specified in a generic way: we represent the definition of security of each cryptographic primitive by an observational equivalence $L \approx R$, where the processes L and R encode functions: they input the arguments of the function and send its result back. Then, the prover can automatically transform a process Q that calls the functions of L (more precisely, contains as subterms terms that perform the same computations as functions of L) into a process Q' that calls the functions of R instead. We have used this technique to specify several variants of shared-key and public-key encryption, signature, message authentication codes, and hash functions, simply by giving the appropriate equivalence $L \approx R$ to the prover. Other game transformations are syntactic transformations, used in order to be able to apply the definition of cryptographic primitives, or to simplify the game obtained after applying these definitions.

In order to prove protocols, these game transformations are organized using a proof strategy based on advice: when a transformation fails, it suggests other transformations that should be applied before, in order to enable the desired transformation. Thanks to this strategy, protocols can often be proved in a fully automatic way. For delicate cases, our prover has an interactive mode, in which the user can manually specify the transformations to apply. It is usually sufficient to specify a few transformations coming from the security definitions of primitives, by indicating the concerned cryptographic primitive and the concerned secret key if any; the prover infers the intermediate syntactic transformations by the advice strategy. This mode is helpful for proving some public-key protocols, in which several security definitions of primitives can be applied, but only one leads to a proof of the protocol. Importantly, our prover is always sound: whatever indications the user gives, when the prover shows a security property of the protocol, the property indeed holds assuming the given hypotheses on the cryptographic primitives.

Our prover CryptoVerif has been implemented in Ocaml (17300 lines of code for version 1.03 of CryptoVerif) and is available at `http://www.di.ens.fr/~blanchet/cryptoc-eng.html`.

1.1 Outline

The next section presents our process calculus for representing games. Section 3 describes the game transformations that we use for proving protocols. Section 4 gives criteria for proving secrecy properties of protocols. Section 5 explains how the prover chooses which transformation to apply at each point. Section 6 presents our experimental results. Section 7 discusses related work and Section 8 concludes. The appendices contain additional formal details, proof sketches, and details on the modeling of some cryptographic primitives.

1.2 Notations

We recall the following standard notations. We denote by $\{M_1/x_1, \dots, M_m/x_m\}$ the substitution that replaces x_j with M_j for each $j \leq m$. The cardinal of a set or multiset S is denoted by $|S|$. If S is a finite set, $x \xleftarrow{R} S$ chooses a random element uniformly in S and assigns it to x. If $\mathcal{A}$ is a probabilistic algorithm, $x \leftarrow \mathcal{A}(x_1, \dots, x_m)$ denotes the experiment of choosing random coins r and assigning to x the result of running $\mathcal{A}(x_1, \dots, x_m)$ with coins r. Otherwise, $x \leftarrow M$ is a simple assignment statement.

$M, N ::=$	terms
$\quad i$	replication index
$\quad x[M_1, \dots, M_m]$	variable access
$\quad f(M_1, \dots, M_m)$	function application
$Q ::=$	input process
$\quad 0$	nil
$\quad Q \mid Q'$	parallel composition
$\quad !^{i \leq n} Q$	replication n times
$\quad \mathsf{newChannel}\ c; Q$	channel restriction
$\quad c[M_1, \dots, M_l](x_1[\widetilde{i}] : T_1, \dots, x_k[\widetilde{i}] : T_k); P$	input
$P ::=$	output process
$\quad \overline{c[M_1, \dots, M_l]}\langle N_1, \dots, N_k \rangle; Q$	output
$\quad \mathsf{new}\ x[i_1, \dots, i_m] : T; P$	random number
$\quad \mathsf{let}\ x[i_1, \dots, i_m] : T = M\ \mathsf{in}\ P$	assignment
$\quad \mathsf{if}\ \mathsf{defined}(M_1, \dots, M_l) \wedge M\ \mathsf{then}\ P\ \mathsf{else}\ P'$	conditional
$\quad \mathsf{find}\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}$ $\mathsf{suchthat}\ \mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)$ $\mathsf{else}\ P$	array lookup

Figure 1: Syntax of the process calculus

2 A Calculus for Games

2.1 Syntax and Informal Semantics

The syntax of our calculus is summarized in Figure 1. This calculus was inspired by the pi calculus and by the calculi of [34, 35, 40] and of [33]. We denote by η the security parameter, which determines in particular the length of keys.

This calculus assumes a countable set of channel names, denoted by c. There is a mapping maxlen_η from channels to integers, such that $\mathrm{maxlen}_\eta(c)$ is the maximum length of a message sent on channel c. Longer messages are truncated. For all c, $\mathrm{maxlen}_\eta(c)$ is polynomial in η. (This is key to guaranteeing that all processes run in probabilistic polynomial time.)

Our calculus also uses parameters, denoted by n, which correspond to integer values polynomial in the security parameter. So, denoting by $I_\eta(n)$ the interpretation of n for a given value of the security parameter η, $I_\eta(n)$ is a polynomially bounded, efficiently computable function of η.

Our calculus also uses types, denoted by T. For each value of the security parameter η, each type corresponds to a subset $I_\eta(T)$ of $\mathit{Bitstring} \cup \{\perp\}$ where $\mathit{Bitstring}$ is the set of all bitstrings and $\perp$ is a special symbol. The set $I_\eta(T)$ must be recognizable in polynomial time, that is, there exists an algorithm that decides whether $x \in I_\eta(T)$ in time polynomial in the length of

x and the value of η. Let *fixed-length* types be types T such that $I_\eta(T)$ is the set of all bitstrings of a certain length, this length being a function of η bounded by a polynomial. Let *large* types be types T such that $\frac{1}{|I_\eta(T)|}$ is negligible. ($f(\eta)$ is *negligible* when for all polynomials q, there exists $\eta_o \in \mathbb{N}$ such that for all $\eta > \eta_0$, $f(\eta) \leq \frac{1}{q(\eta)}$.) Particular types are predefined: *bool*, such that $I_\eta(\mathit{bool}) = \{\mathrm{true}, \mathrm{false}\}$, where false is 0 and true is 1; *bitstring*, such that $I_\eta(\mathit{bitstring}) = \mathit{Bitstring}$; $\mathit{bitstring}_\bot$ such that $I_\eta(\mathit{bitstring}_\bot) = \mathit{Bitstring} \cup \{\bot\}$; $[1, n]$ where n is a parameter, such that $I_\eta([1, n]) = [1, I_\eta(n)]$. (We consider integers as bitstrings without leading zeroes.)

The calculus also uses function symbols f. Each function symbol comes with a type declaration $f : T_1 \times \ldots \times T_m \to T$. For each value of η, each function symbol f corresponds to a function $I_\eta(f)$ from $I_\eta(T_1) \times \ldots \times I_\eta(T_m)$ to $I_\eta(T)$, such that $I_\eta(f)(x_1, \ldots, x_m)$ is computable in polynomial time in the lengths of $x_1, \ldots, x_m$ and the value of η. Particular functions are predefined, and some of them use the infix notation: $M = N$ for the equality test, $M \neq N$ for the inequality test (both taking two values of the same type T and returning a value of type *bool*), $M \vee N$ for the boolean or, $M \wedge N$ for the boolean and, $\neg M$ for the boolean negation (taking and returning values of type *bool*).

In this calculus, terms represent computations on bitstrings. The replication index i is an integer which serves in distinguishing different copies of a replicated process $!^{i \leq n}$. (Replication indices are typically used as array indices.) The variable access $x[M_1, \ldots, M_m]$ returns the content of the cell of indices $M_1, \ldots, M_m$ of the m-dimensional array variable x. We use x, y, z, u as variable names. The function application $f(M_1, \ldots, M_m)$ returns the result of applying function f to $M_1, \ldots, M_m$.

The calculus distinguishes two kinds of processes: input processes Q are ready to receive a message on a channel; output processes P output a message on a channel after executing some internal computations. The input process 0 does nothing; $Q \mid Q'$ is the parallel composition of Q and Q'; $!^{i \leq n} Q$ represents n copies of Q in parallel, each with a different value of $i \in [1, n]$; $\mathsf{newChannel}\ c; Q$ creates a new private channel c and executes Q; the semantics of the input $c[M_1, \ldots, M_l](x_1[\widetilde{i}] : T_1, \ldots, x_k[\widetilde{i}] : T_k); P$ will be explained below together with the semantics of the output.

The output process $\mathsf{new}\ x[i_1, \ldots, i_m] : T; P$ chooses a new random number uniformly in $I_\eta(T)$, stores it in $x[i_1, \ldots, i_m]$, and executes P. (The type T must be a fixed-length type, because probabilistic polynomial-time Turing machines can choose random numbers uniformly only in such types.) Function symbols represent deterministic functions, so all random numbers must be chosen by $\mathsf{new}\ x[i_1, \ldots, i_m] : T$. Deterministic functions make automatic syntactic manipulations easier: we can duplicate a term without changing its value. The process $\mathsf{let}\ x[i_1, \ldots, i_m] : T = M\ \mathsf{in}\ P$ stores the bitstring value of M (which must be in $I_\eta(T)$) in $x[i_1, \ldots, i_m]$ and executes P. Next, we explain the process $\mathsf{find}\ (\bigoplus_{j=1}^m u_{j1}[\widetilde{i}] \leq n_{j1}, \ldots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}\ \mathsf{suchthat}\ \mathsf{defined}(M_{j1}, \ldots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P$, where $\widetilde{i}$ denotes a tuple $i_1, \ldots, i_{m'}$. The order and array indices on tuples are taken component-wise, so for instance, $u_{j1}[\widetilde{i}] \leq n_{j1}, \ldots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}$ can be further abbreviated $\widetilde{u_j}[\widetilde{i}] \leq \widetilde{n_j}$. A simple example is the following: $\mathsf{find}\ u \leq n\ \mathsf{suchthat}\ \mathsf{defined}(x[u]) \wedge x[u] = a\ \mathsf{then}\ P'\ \mathsf{else}\ P$ tries to find an index u such that $x[u]$ is defined and $x[u] = a$, and when such a u is found, it executes P' with that value of u; otherwise, it executes P. In other words, this find construct looks for the value a in the array x, and when a is found, it stores in u an index such that $x[u] = a$. Therefore, the find construct allows us to access arrays, which is key for our purpose. More generally, $\mathsf{find}\ u_1[\widetilde{i}] \leq n_1, \ldots, u_m[\widetilde{i}] \leq n_m\ \mathsf{suchthat}\ \mathsf{defined}(M_1, \ldots, M_l) \wedge M\ \mathsf{then}\ P'\ \mathsf{else}\ P$ tries to find values of $u_1, \ldots, u_m$ for which $M_1, \ldots, M_l$ are defined and M is true. In case of success, it executes P'. In case of failure, it executes P. This is further generalized to m branches: $\mathsf{find}\ (\bigoplus_{j=1}^m u_{j1}[\widetilde{i}] \leq n_{j1}, \ldots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}\ \mathsf{suchthat}\ \mathsf{defined}(M_{j1}, \ldots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P$ tries to find a branch j in $[1, m]$ such that there are values of $u_{j1}, \ldots, u_{jm_j}$ for which $M_{j1}, \ldots, M_{jl_j}$ are defined and M_j is true. In case of success, it executes P_j. In case of failure for all branches, it executes P. More formally, it evaluates the conditions $\mathsf{defined}(M_{j1}, \ldots, M_{jl_j}) \wedge M_j$ for each j and each value of $u_{j1}[\widetilde{i}], \ldots, u_{jm_j}[\widetilde{i}]$ in $[1, n_{j1}] \times \ldots \times [1, n_{jm_j}]$. If none of these conditions is true, it executes P. Otherwise, it chooses randomly with uniform[1] probability one j and one value of $u_{j1}[\widetilde{i}], \ldots, u_{jm_j}[\widetilde{i}]$ such that the corresponding condition is true and executes P_j. The conditional $\mathsf{if}\ \mathsf{defined}(M_1, \ldots, M_l) \wedge M\ \mathsf{then}\ P\ \mathsf{else}\ P'$ executes P if $M_1, \ldots, M_l$ are defined and M evaluates to true. Otherwise, it executes P'. This conditional is defined as syntactic sugar for $\mathsf{find}\ \mathsf{suchthat}\ \mathsf{defined}(M_1, \ldots, M_l) \wedge M\ \mathsf{then}\ P\ \mathsf{else}\ P'$. The conjunct $\mathsf{defined}(M_1, \ldots, M_l)$ can be omitted when $l = 0$ and M can be omitted when it is true.

Finally, let us explain the output $\overline{c[M_1, \ldots, M_l]}\langle N_1, \ldots, N_k\rangle; Q$. A channel $c[M_1, \ldots, M_l]$ consists of both a channel name c and a tuple of terms $M_1, \ldots, M_l$. Channel names c allow us to define private channels to which the adversary can never have access, by $\mathsf{newChannel}\ c$. (This is useful in the proofs, although all channels of protocols are often public.) Terms $M_1, \ldots, M_l$ are intuitively analogous to IP addresses and ports which are numbers that the adversary may guess. A semantic configuration always consists of a single output process (the process currently being executed) and several input processes. When the output process executes $\overline{c[M_1, \ldots, M_l]}\langle N_1, \ldots, N_k\rangle; Q$, one looks for an input on channel $c[M'_l \ldots, M'_l]$, where $M'_1, \ldots, M'_l$ evaluate to the same bitstrings as $M_1, \ldots, M_l$, and with the same arity k, in the available input processes. If no such input process is found, the process blocks. Otherwise, one such input process $c[M'_1, \ldots, M'_l](x_1[\widetilde{i}] : T_1, \ldots, x_k[\widetilde{i}] : T_k); P$ is chosen randomly with uniform probability. The communication is then executed: for each $j \leq k$, the output message N_j is evaluated, its result is truncated to length $\mathrm{maxlen}_\eta(c)$, the obtained bitstring

[1] A probabilistic polynomial-time Turing machine can choose a random number uniformly in a set of cardinal m only when m is a power of 2. When m is not a power of 2, there exist approximate algorithms: for example, in order to obtain a random integer in $[0, m-1]$, we can choose a random integer r uniformly among $[0, 2^k - 1]$ for a certain k large enough and return $r \mod m$. The distribution can be made as close as we wish to the uniform distribution by choosing k large enough.

is stored in $x_j[\widetilde{i}]$ if it is in $I_\eta(T_j)$ (otherwise the process blocks). Finally, the output process P that follows the input is executed. The input process Q that follows the output is stored in the available input processes for future execution. Note that the syntax requires an output to be followed by an input process, as in [33]. If one needs to output several messages consecutively, one can simply insert fictitious inputs between the outputs. The adversary can then schedule the outputs by sending messages to these inputs.

Using different channels for each input and output allows the adversary to control the network. For instance, we may write $!^{i\leq n} c[i](x[i] : T) \ldots \overline{c'[i]}\langle M\rangle \ldots$ The adversary can then decide which copy of the replicated process receives its message, simply by sending it on $c[i]$ for the appropriate value of i.

An else branch of find or if may be omitted when it is else $\overline{yield}\langle\rangle; 0$. (Note that "else 0" would not be syntactically correct.) A trailing 0 after an output may be omitted.

Variables can be defined by assignments, inputs, restrictions, and array lookups. The *current replication indices* at a certain program point in a process are $i_1, \ldots, i_m$ where the replications above the considered program point are $!^{i_1\leq n_1} \ldots !^{i_m\leq n_m}$. We often abbreviate $x[i_1, \ldots, i_m]$ by x when $i_1, \ldots, i_m$ are the current replication indices, but it should be kept in mind that this is only an abbreviation. Variables defined under a replication must be arrays: for example $!^{i_1\leq n_1} \ldots !^{i_m\leq n_m}$ let $x[i_1, \ldots, i_m] : T = M$ in ... More formally, we require the following invariant:

Invariant 1 (Single definition) The process Q_0 satisfies Invariant 1 if and only if

1. in every definition of $x[i_1, \ldots, i_m]$ in Q_0, the indices $i_1, \ldots, i_m$ of x are the current replication indices at that definition, and
2. two different definitions of the same variable x in Q_0 are in different branches of a find (or if).

Invariant 1 guarantees that each variable is assigned at most once for each value of its indices. (Indeed, item 2 shows that only one definition of each variable can be executed for given indices in each trace.)

Invariant 2 (Defined variables) The process Q_0 satisfies Invariant 2 if and only if every occurrence of a variable access $x[M_1, \ldots, M_m]$ in Q_0 is either

- syntactically under the definition of $x[M_1, \ldots, M_m]$ (in which case $M_1, \ldots, M_m$ are in fact the current replication indices at the definition of x);
- or in a defined condition in a find process;
- or in M'_j or P_j in a process of the form find $(\bigoplus_{j=1}^{m''} \widetilde{u_j}[\widetilde{i}] \leq \widetilde{n_j}$ suchthat defined$(M'_{j1}, \ldots, M'_{jl_j}) \wedge M'_j$ then $P_j)$ else P where for some $k \leq l_j$, $x[M_1, \ldots, M_m]$ is a subterm of M'_{jk}.

Invariant 2 guarantees that variables can be accessed only when they have been initialized. It checks that the definition of the variable access is either in scope (first item) or checked by a find (last item). Both invariants are checked by the prover for the initial game and preserved by all game transformations.

We say that a function $f : T_1 \times \ldots \times T_m \to T$ is *poly-injective* when it is injective and its inverses can be computed in polynomial time, that is, there exist functions $f_j^{-1} : T \to T_j$ $(1 \leq j \leq m)$ such that $f_j^{-1}(f(x_1, \ldots, x_m)) = x_j$ and f_j^{-1} can be computed in polynomial time in the length of $f(x_1, \ldots, x_m)$ and in the security parameter. When f is poly-injective, we define a pattern matching construct let $f(x_1, \ldots, x_m) = M$ in P else Q as an abbreviation for let $y : T = M$ in let $x_1 : T_1 = f_1^{-1}(y)$ in ... let $x_m : T_m = f_m^{-1}(y)$ in if $f(x_1, \ldots, x_m) = y$ then P else Q. We naturally generalize this construct to let $N = M$ in P else Q where N is built from poly-injective functions and variables.

We denote by $\mathrm{var}(P)$ the set of variables that occur in P and by $\mathrm{fc}(P)$ the set of free channels of P. (We use similar notations for input processes.)

2.2 Example

Let us introduce two cryptographic primitives that we use below.

Definition 1 Let T_{mr}, T_{mk}, and T_{ms} be types that correspond intuitively to random seeds, keys, and message authentication codes, respectively; T_{mr} is a fixed-length type. A message authentication code [15] consists of three function symbols:

- $\mathrm{mkgen} : T_{mr} \to T_{mk}$ where $I_\eta(\mathrm{mkgen}) = \mathrm{mkgen}_\eta$ is the key generation algorithm taking as argument a random bitstring and returning a key. (Usually, mkgen is a randomized algorithm; here, since we separate the choice of random numbers from computation, mkgen takes an additional argument representing the random coins.)
- $\mathrm{mac} : \mathit{bitstring} \times T_{mk} \to T_{ms}$ where $I_\eta(\mathrm{mac}) = \mathrm{mac}_\eta$ is the MAC algorithm taking as argument a message and a key, and returning the corresponding tag. (We assume here that mac is deterministic; we could easily encode a randomized mac by adding an additional argument as for mkgen.)
- $\mathrm{check} : \mathit{bitstring} \times T_{mk} \times T_{ms} \to \mathit{bool}$ where $I_\eta(\mathrm{check}) = \mathrm{check}_\eta$ is a checking algorithm such that $\mathrm{check}_\eta(m, k, t) = \mathrm{true}$ if and only if t is a valid MAC of message m under key k. (Since mac is deterministic, $\mathrm{check}_\eta(m, k, t)$ is typically $\mathrm{mac}_\eta(m, k) = t$.)

We have $\forall m \in \mathit{Bitstring}, \forall r \in I_\eta(T_{mr}), \mathrm{check}_\eta(m, \mathrm{mkgen}_\eta(r), \mathrm{mac}_\eta(m, \mathrm{mkgen}_\eta(r))) = \mathrm{true}$.

A MAC is UF-CMA (satisfies unforgeability under chosen message attacks) if and only if for all polynomials q,

$$\max_{\mathcal{A}} \Pr\left[\begin{array}{l} r \stackrel{R}{\leftarrow} I_\eta(T_{mr}); k \leftarrow \mathrm{mkgen}_\eta(r); \\ (m, t) \leftarrow \mathcal{A}^{\mathrm{mac}_\eta(.,k), \mathrm{check}_\eta(.,k,.)} : \mathrm{check}_\eta(m, k, t) \end{array}\right]$$

is negligible, where the adversary $\mathcal{A}$ is any probabilistic Turing machine, running in time $q(\eta)$, with oracle access to $\mathrm{mac}_\eta(., k)$ and $\mathrm{check}_\eta(., k, .)$, and $\mathcal{A}$ has not called $\mathrm{mac}_\eta(., k)$ on message m.

Definition 2 Let T_r and T'_r be fixed-length types; let T_k and T_e be types. A symmetric encryption scheme [13] consists of three function symbols $\mathrm{kgen} : T_r \to T_k$, $\mathrm{enc} : \mathit{bitstring} \times T_k \times T'_r \to T_e$, and $\mathrm{dec} : T_e \times T_k \to \mathit{bitstring}_\perp$, with $I_\eta(\mathrm{kgen}) = \mathrm{kgen}_\eta$, $I_\eta(\mathrm{enc}) = \mathrm{enc}_\eta$, $I_\eta(\mathrm{dec}) = \mathrm{dec}_\eta$, such that for all $m \in \mathit{Bitstring}$, $r \in I_\eta(T_r)$, and $r' \in I_\eta(T'_r)$, $\mathrm{dec}_\eta(\mathrm{enc}_\eta(m, \mathrm{kgen}_\eta(r), r'), \mathrm{kgen}_\eta(r)) = m$.

Let $LR(x, y, b) = x$ if $b = 0$ and $LR(x, y, b) = y$ if $b = 1$, defined only when x and y are bitstrings of the same length. A symmetric encryption scheme is IND-CPA (satisfies indistinguishability under chosen plaintext attacks) if and only if for all polynomials q,

$$\max_{\mathcal{A}} 2 \Pr \begin{bmatrix} b \stackrel{R}{\leftarrow} \{0,1\}; r \stackrel{R}{\leftarrow} I_\eta(T_r); k \leftarrow \mathrm{kgen}_\eta(r); \\ b' \leftarrow \mathcal{A}^{r' \stackrel{R}{\leftarrow} I_\eta(T'_r); \mathrm{enc}_\eta(LR(.,.,b),k,r')} : b' = b \end{bmatrix} - 1$$

is negligible, where the adversary $\mathcal{A}$ is any probabilistic Turing machine, running in time $q(\eta)$, with oracle access to the left-right encryption algorithm which, given two bitstrings a_0 and a_1 of the same length, returns $r' \stackrel{R}{\leftarrow} I_\eta(T'_r); \mathrm{enc}_\eta(LR(a_0, a_1, b), k, r')$, that is, encrypts a_0 when $b = 0$ and a_1 when $b = 1$.

Example 1 Let us consider the following trivial protocol:

$$\begin{array}{ll} A \to B : e, \mathrm{mac}(e, x_{mk}) & \text{where } e = \mathrm{enc}(x'_k, x_k, x''_r) \\ & \text{and } x''_r, x'_k \text{ are fresh random numbers} \end{array}$$

A and B are assumed to share a key x_k for a symmetric encryption scheme and a key x_{mk} for a message authentication code. A creates a fresh key x'_k and sends it encrypted under x_k to B. A MAC is appended to the message, in order to guarantee integrity. The goal of the protocol is that x'_k should be a secret key shared between A and B. This protocol can be modeled in our calculus by the following process Q_0:

$$\begin{aligned} Q_0 = {} & \mathit{start}(); \mathsf{new}\ x_r : T_r; \mathsf{let}\ x_k : T_k = \mathrm{kgen}(x_r)\ \mathsf{in} \\ & \mathsf{new}\ x'_r : T_{mr}; \mathsf{let}\ x_{mk} : T_{mk} = \mathrm{mkgen}(x'_r)\ \mathsf{in} \\ & \overline{c}\langle\rangle; (Q_A \mid Q_B) \\ Q_A = {} & !^{i \le n} c_A[i](); \mathsf{new}\ x'_k : T_k; \mathsf{new}\ x''_r : T'_r; \\ & \mathsf{let}\ x_m : \mathit{bitstring} = \mathrm{enc}(\mathrm{k2b}(x'_k), x_k, x''_r)\ \mathsf{in} \\ & \overline{c_A[i]}\langle x_m, \mathrm{mac}(x_m, x_{mk})\rangle \\ Q_B = {} & !^{i' \le n} c_B[i'](x'_m, x_{ma}); \\ & \mathsf{if}\ \mathrm{check}(x'_m, x_{mk}, x_{ma})\ \mathsf{then} \\ & \mathsf{let}\ \mathrm{i}_\perp(\mathrm{k2b}(x''_k)) = \mathrm{dec}(x'_m, x_k)\ \mathsf{in}\ \overline{c_B[i']}\langle\rangle \end{aligned}$$

When Q_0 receives a message on channel start, it begins execution: it generates the keys x_k and x_{mk} by choosing random coins x_r and $x_{r'}$ and applying the appropriate key generation algorithms. Then it yields control to the context (the adversary), by outputting on channel c. After this output, n copies of processes for A and B are ready to be executed, when the context outputs on channels $c_A[i]$ or $c_B[i]$ respectively. In a session that runs as expected, the context first sends a message on $c_A[i]$. Then Q_A creates a fresh key x'_k (T_k is assumed to be a fixed-length type), encrypts it under x_k with random coins x''_r, computes the MAC under x_{mk} of the ciphertext, and sends the ciphertext and the MAC on $c_A[i]$. The function $\mathrm{k2b} : T_k \to \mathit{bitstring}$ is the natural injection $I_\eta(\mathrm{k2b})(x) = x$; it is needed only for type conversion. The context is then expected to forward this message on $c_B[i]$. When Q_B receives this message, it checks the MAC, decrypts, and stores the obtained key in x''_k. (The function $\mathrm{i}_\perp : \mathit{bitstring} \to \mathit{bitstring}_\perp$ is the natural injection; it is useful to check that decryption succeeded.) This key x''_k should be secret.

The context is responsible for forwarding messages from A to B. It can send messages in unexpected ways in order to mount an attack.

Although we use a trivial running example due to length constraints, this example is sufficient to illustrate the main features of our prover. Section 6 presents results obtained on more realistic protocols.

2.3 Type System

We use a type system to check that bitstrings of the proper type are passed to each function and that array indices are used correctly.

To be able to type variable accesses used not under their definition (such accesses are guarded by a find construct), the type-checking algorithm proceeds in two passes. In the first pass, it builds a type environment $\mathcal{E}$, which maps variable names x to types $[1, n_1] \times \ldots \times [1, n_m] \to T$, where the definition of $x[i_1, \ldots, i_m]$ of type T occurs under replications $!^{i_1 \le n_1}, \ldots, !^{i_m \le n_m}$. The tool checks that all definitions of the same variable x yield the same value of $\mathcal{E}(x)$, so that $\mathcal{E}$ is properly defined.

In the second pass, the process is typechecked in the type environment $\mathcal{E}$ by a simple type system. This type system is detailed in Appendix A. It defines the judgment $\mathcal{E} \vdash Q$, which means that the process Q is well-typed in environment $\mathcal{E}$.

Invariant 3 (Typing) The process Q_0 satisfies Invariant 3 if and only if the type environment $\mathcal{E}$ for Q_0 is well-defined, and $\mathcal{E} \vdash Q_0$.

We require the adversary to be well-typed. This requirement does not restrict its computing power, because it can always define type-cast functions $f : T \to T'$ to bypass the type system. Similarly, the type system does not restrict the class of protocols that we consider, since the protocol may contain type-cast functions. The type system just makes explicit which set of bitstrings may appear at each point of the protocol.

2.4 Formal Semantics

The semantics is defined by a probabilistic reduction relation formally detailed in Appendix B. The notation $E, M \Downarrow a$ means that the term M evaluates to the bitstring a in environment E. We denote by $\Pr[Q \rightsquigarrow_\eta \overline{c}\langle a\rangle]$ the probability that at least one of the outputs of Q on channel c sends the bitstring a. (When c is not free in Q, $\Pr[Q \rightsquigarrow_\eta \overline{c}\langle a\rangle] = 0$.)

Our semantics is such that, for each process Q, there exists a probabilistic polynomial time Turing machine that simulates Q. (Processes run in polynomial time since the number of processes created by a replication and the length of messages sent on channels are bounded by polynomials.) Conversely, our calculus can simulate a probabilistic polynomial-time Turing machine, simply by choosing coins by new and by applying a function symbol defined to perform the same computations as the Turing machine.

2.5 Observational Equivalence

A context is a process containing a hole $[\,]$. An evaluation context C is a context built from $[\,]$, $\mathsf{newChannel}\ c; C$, $Q \mid C$, and $C \mid Q$. We use an evaluation context to represent the adversary. We denote by $C[Q]$ the process obtained by replacing the hole $[\,]$ in the context C with the process Q. Our definition of observational equivalence is adapted from definitions for previous calculi such as [40].

Definition 3 (Observational equivalence) Let Q and Q' be two processes and V a set of variables. Assume that Q and Q' satisfy Invariants 1, 2, and 3 and the variables of V are defined in Q and Q', with the same types.

An evaluation context is said to be *acceptable* for Q, Q', V if and only if $\mathrm{var}(C) \cap (\mathrm{var}(Q) \cup \mathrm{var}(Q')) \subseteq V$ and $C[Q]$ satisfies Invariants 1, 2, and 3. (Then $C[Q']$ also satisfies these invariants.)

We say that Q and Q' are *observationally equivalent* with public variables V, written $Q \approx^V Q'$, when for all evaluation contexts C acceptable for Q, Q', V, for all channels c and bitstrings a, $|\Pr[C[Q] \rightsquigarrow_\eta \overline{c}\langle a\rangle] - \Pr[C[Q'] \rightsquigarrow_\eta \overline{c}\langle a\rangle]|$ is negligible.

Intuitively, the goal of the adversary represented by context C is to distinguish Q from Q'. When it succeeds, it performs a different output, for example $\overline{c}\langle 0\rangle$ when it has recognized Q and $\overline{c}\langle 1\rangle$ when it has recognized Q'. When $Q \approx^V Q'$, the context has negligible probability of distinguishing Q from Q'.

The unusual requirement on variables of C comes from the presence of arrays and of the associated find construct which gives C direct access to variables of Q and Q': the context C is allowed to access variables of Q and Q' only when they are in V. (In more standard settings, the calculus does not have constructs that allow the context to access variables of Q and Q'.) The following result is not difficult to prove:

Lemma 1 *$\approx^V$ is an equivalence relation, and $Q \approx^V Q'$ implies that $C[Q] \approx^{V'} C[Q']$ for all evaluation contexts C acceptable for Q, Q', V and all $V' \subseteq V \cup (\mathrm{var}(C) \setminus (\mathrm{var}(Q) \cup \mathrm{var}(Q')))$.*

We denote by $Q \approx_0^V Q'$ the particular case in which for all evaluation contexts C acceptable for Q, Q', V, for all channels c and bitstrings a, $\Pr[C[Q] \rightsquigarrow_\eta \overline{c}\langle a\rangle] = \Pr[C[Q'] \rightsquigarrow_\eta \overline{c}\langle a\rangle]$. When V is empty, we write $Q \approx Q'$ instead of $Q \approx^V Q'$ and $Q \approx_0 Q'$ instead of $Q \approx_0^V Q'$.

3 Game Transformations

In this section, we describe the game transformations that allow us to transform the process that represents the initial protocol into a process on which the desired security property can be proved directly, by criteria given in Section 4. These transformations are parametrized by the set V of variables that the context can access. As we shall see in Section 4, V contains variables that we would like to prove secret. (The context will contain test queries that access these variables.) These transformations transform a process Q_0 into a process Q'_0 such that $Q_0 \approx^V Q'_0$.

3.1 Syntactic Transformations

RemoveAssign(x): When x is defined by an assignment $\mathsf{let}\ x[i_1, \dots, i_l] : T = M$ in P, we replace x with its value. Precisely, the transformation is performed only when x does not occur in M (non-cyclic assignment). When x has several definitions, we simply replace $x[i_1, \dots, i_l]$ with M in P. (For accesses to x guarded by find, we do not know which definition of x is actually used.) When x has a single definition, we replace everywhere in the game $x[M_1, \dots, M_l]$ with $M\{M_1/i_1, \dots, M_l/i_l\}$. We additionally update the defined conditions of find to preserve Invariant 2 and to make sure that, if a condition of find guarantees that $x[M_1, \dots, M_l]$ is defined in the initial game, then so does the corresponding condition of find in the transformed game. (Essentially, when $y[M'_1, \dots, M'_{l'}]$ occurs in M, the transformation typically creates new occurrences of $y[M''_1, \dots, M''_{l'}]$ for some $M''_1, \dots, M''_{l'}$, so the condition that $y[M''_1, \dots, M''_{l'}]$ is defined must sometimes be explicitly added to conditions of find in order to preserve Invariant 2.) When $x \in V$, its definition is kept unchanged. Otherwise, when x is not referred to at all after the transformation, we remove the definition of x. When x is referred to only at the root of defined tests, we replace its definition with a constant. (The definition point of x is important, but not its value.)

Example 2 In the process of Example 1, the transformation **RemoveAssign**(x_{mk}) substitutes $\mathrm{mkgen}(x'_r)$ for x_{mk} in the whole process and removes the assignment $\mathsf{let}\ x_{mk} : T_{mk} = \mathrm{mkgen}(x'_r)$. After this substitution, $\mathrm{mac}(x_m, x_{mk})$ becomes $\mathrm{mac}(x_m, \mathrm{mkgen}(x'_r))$ and $\mathrm{check}(x'_m, x_{mk}, x_{ma})$ becomes $\mathrm{check}(x'_m, \mathrm{mkgen}(x'_r), x_{ma})$, thus exhibiting terms required in Section 3.2. The situation is similar for **RemoveAssign**(x_k).

SArename(x): The transformation **SArename** (single assignment rename) aims at renaming variables so that each variable has a single definition in the game; this is useful for distinguishing cases depending on which definition of x has set $x[\widetilde{i}]$. This transformation can be applied only when $x \notin V$. When x has $m > 1$ definitions, we rename each definition of x to a different variable $x_1, \dots, x_m$. Terms $x[\widetilde{i}]$ under a definition of $x_j[\widetilde{i}]$ are then replaced with $x_j[\widetilde{i}]$. Each branch of find $FB = \widetilde{u}[\widetilde{i}] \leq \widetilde{n}\ \mathsf{suchthat}\ \mathsf{defined}(M'_1, \dots, M'_{l'}) \wedge M\ \mathsf{then}\ P$ where $x[M_1, \dots, M_l]$ is a subterm of some M'_k for $k \leq l'$ is replaced with m branches $FB\{x_j[M_1, \dots, M_l]/x[M_1, \dots, M_l]\}$ for $1 \leq j \leq m$.

Example 3 Consider the following process

$$start(); \mathsf{new}\ r_A : T_r; \mathsf{let}\ k_A : T_k = \mathrm{kgen}(r_A)\ \mathsf{in}$$
$$\mathsf{new}\ r_B : T_r; \mathsf{let}\ k_B : T_k = \mathrm{kgen}(r_B)\ \mathsf{in}\ \overline{yield}\langle\rangle; (Q_K \mid Q_S)$$

$$
\begin{aligned}
Q_K = \ & !^{i \le n} c[i](h : T_h, k : T_k) \\
& \quad \mathsf{if}\ h = A\ \mathsf{then\ let}\ k' : T_k = k_A\ \mathsf{in}\ \overline{yield}\langle\rangle\ \mathsf{else} \\
& \quad \mathsf{if}\ h = B\ \mathsf{then\ let}\ k' : T_k = k_B\ \mathsf{in}\ \overline{yield}\langle\rangle\ \mathsf{else} \\
& \quad \mathsf{let}\ k' : T_k = k\ \mathsf{in}\ \overline{yield}\langle\rangle \\
Q_S = \ & !^{i' \le n'} c'[i'](h' : T_h); \mathsf{find}\ u \le n\ \mathsf{suchthat} \\
& \quad \mathsf{defined}(h[u], k'[u]) \wedge h' = h[u]\ \mathsf{then}\ P_1(k'[u])\ \mathsf{else}\ P_2
\end{aligned}
$$

The process Q_K stores in (h, k') a table of pairs (host name, key): the key for A is k_A, for B, k_B, and for any other h, the adversary can choose the key k. The process Q_S queries this table of keys to find the key $k'[u]$ of host h', then executes $P_1(k'[u])$. If h' is not found, it executes P_2.

By the transformation **SArename**(k'), we can perform a case analysis, to distinguish the cases in which $k' = k_A$, $k' = k_B$, or $k' = k$. After transformation, we obtain the following processes:

$$
\begin{aligned}
Q'_K = \ & !^{i \le n} c[i](h : T_h, k : T_k) \\
& \quad \mathsf{if}\ h = A\ \mathsf{then\ let}\ k'_1 : T_k = k_A\ \mathsf{in}\ \overline{yield}\langle\rangle\ \mathsf{else} \\
& \quad \mathsf{if}\ h = B\ \mathsf{then\ let}\ k'_2 : T_k = k_B\ \mathsf{in}\ \overline{yield}\langle\rangle\ \mathsf{else} \\
& \quad \mathsf{let}\ k'_3 : T_k = k\ \mathsf{in}\ \overline{yield}\langle\rangle \\
Q'_S = \ & !^{i' \le n'} c'[i'](h' : T_h); \\
& \quad \mathsf{find}\ u \le n\ \mathsf{suchthat\ defined}(h[u], k'_1[u]) \\
& \qquad \wedge h' = h[u]\ \mathsf{then}\ P_1(k'_1[u]) \\
& \quad\ \oplus u \le n\ \mathsf{suchthat\ defined}(h[u], k'_2[u]) \\
& \qquad \wedge h' = h[u]\ \mathsf{then}\ P_1(k'_2[u]) \\
& \quad\ \oplus u \le n\ \mathsf{suchthat\ defined}(h[u], k'_3[u]) \\
& \qquad \wedge h' = h[u]\ \mathsf{then}\ P_1(k'_3[u])\ \mathsf{else}\ P_2
\end{aligned}
$$

After the simplification (sketched below), Q'_S becomes:

$$
\begin{aligned}
Q''_S = \ & !^{i' \le n'} c'[i'](h' : T_h); \\
& \quad \mathsf{find}\ u \le n\ \mathsf{suchthat\ defined}(h[u], k'_1[u]) \\
& \qquad \wedge h' = A\ \mathsf{then}\ P_1(k_A) \\
& \quad\ \oplus u \le n\ \mathsf{suchthat\ defined}(h[u], k'_2[u]) \\
& \qquad \wedge h' = B\ \mathsf{then}\ P_1(k_B) \\
& \quad\ \oplus u \le n\ \mathsf{suchthat\ defined}(h[u], k'_3[u]) \\
& \qquad \wedge h' = h[u]\ \mathsf{then}\ P_1(k[u])\ \mathsf{else}\ P_2
\end{aligned}
$$

since, when $k'_1[u]$ is defined, $k'_1[u] = k_A$ and $h[u] = A$, and similarly for $k'_2[u]$ and $k'_3[u]$.

Simplify: The prover uses a simplification algorithm, based on an equational prover, using an algorithm similar to the Knuth-Bendix completion [30]. This equational prover uses:

- User-defined equations, of the form $\forall x_1 : T_1, \dots, \forall x_m : T_m, M$ which mean that for all environments E, if for all $j \le m$, $E(x_j) \in I_\eta(T_j)$, then $E, M \Downarrow \text{true}$. For example, considering MAC and encryption shemes as in Definitions 1 and 2 respectively, we have:

$$
\begin{aligned}
& \forall r : T_{mr}, \forall m : \mathit{bitstring}, \\
& \quad \text{check}(m, \text{mkgen}(r), \text{mac}(m, \text{mkgen}(r))) = \text{true}
\end{aligned}
\tag{mac}
$$

$$
\begin{aligned}
& \forall m : \mathit{bitstring}; \forall r : T_r, \forall r' : T'_r, \\
& \quad \text{dec}(\text{enc}(m, \text{kgen}(r), r'), \text{kgen}(r)) = \text{i}_\perp(m)
\end{aligned}
\tag{enc}
$$

  We express the poly-injectivity of the function k2b of Example 1 by

$$
\begin{aligned}
& \forall x : T_k, \forall y : T_k, (\text{k2b}(x) = \text{k2b}(y)) = (x = y) \\
& \forall x : T_k, \text{k2b}^{-1}(\text{k2b}(x)) = x
\end{aligned}
\tag{k2b}
$$

  where k2b^{-1} is a function symbol that denotes the inverse of k2b. We have similar formulas for $\text{i}_\perp$.

- Equations that come from the process. For example, in the process $\mathsf{if}\ M\ \mathsf{then}\ P\ \mathsf{else}\ P'$, we have $M = \text{true}$ in P and $M = \text{false}$ in P'.

- The low probability of collision between random values. For example, when x is defined by $\mathsf{new}\ x : T$ and T is a large type, $x[M_1, \dots, M_m] = x[M'_1, \dots, M'_m]$ implies $M_1 = M'_1, \dots, M_m = M'_m$ up to negligible probability.

  Similarly, when 1) x is defined by $\mathsf{new}\ x : T$ and T is a large type, 2) for each value of M_1, there is at most one value of x (or of a part of x of a large type) that can yield that value of M_1, and 3) M_2 does not depend on x, then $M_1 \neq M_2$ up to negligible probability. The fact that M_2 does not depend on x is proved using a dependency analysis.

The prover combines these properties to simplify terms, and uses simplified forms of terms to simplify processes. For example, if M simplifies to true, then $\mathsf{if}\ M\ \mathsf{then}\ P\ \mathsf{else}\ P'$ simplifies to P. Similarly, a branch of find is removed when the associated condition simplifies to false.

Details on the simplification procedure can be found in Appendix C and the proof of the following proposition in Appendix E.1.

Proposition 1 *Let Q_0 be a process that satisfies Invariants 1, 2, and 3 and Q'_0 the process obtained from Q_0 by one of the transformations above. Then Q'_0 satisfies Invariants 1, 2, and 3, and $Q_0 \approx^V Q'_0$.*

3.2 Applying the Definition of Security of Primitives

The security of cryptographic primitives is defined using observational equivalences given as axioms. Importantly, this formalism allows us to specify many different primitives in a generic way. Such equivalences are then used by the prover in order to transform a game into another, observationally equivalent game, as explained below in this section.

The primitives are specified using equivalences of the form $(G_1, \dots, G_m) \approx (G'_1, \dots, G'_m)$ where G is defined by the following grammar, with $l \ge 0$ and $m \ge 1$:

$G ::=$	group of functions
$\quad !^{i \le n} \mathsf{new}\ y_1 : T_1; \dots; \mathsf{new}\ y_l : T_l; (G_1, \dots, G_m)$	replication, restrictions
$\quad (x_1 : T_1, \dots, x_l : T_l) \to FP$	function

$$
\begin{array}{lll}
FP ::= & & \text{functional processes} \\
\quad M & & \text{term} \\
\quad \mathsf{new}\ x[\widetilde{i}] : T; FP & & \text{random number} \\
\quad \mathsf{let}\ x[\widetilde{i}] : T = M\ \mathsf{in}\ FP & & \text{assignment} \\
\quad \mathsf{find}\ (\bigoplus_{j=1}^{m} \widetilde{u_j}[\widetilde{i}] \leq \widetilde{n_j}\ \mathsf{suchthat} & & \\
\qquad \mathsf{defined}(M_{j1}, \ldots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ FP_j)\ \mathsf{else}\ FP & & \\
& & \text{array lookup}
\end{array}
$$

Intuitively, $(x_1 : T_1, \ldots, x_l : T_l) \rightarrow FP$ represents a function that takes as argument values $x_1, \ldots, x_l$ of types $T_1, \ldots, T_l$ respectively and returns a result computed by FP. The observational equivalence $(G_1, \ldots, G_m) \approx (G'_1, \ldots, G'_m)$ expresses that the adversary has a negligible probability of distinguishing functions in the left-hand side from corresponding functions in the right-hand side. Formally, functions can be encoded as processes that input their arguments and output their result on a channel, as shown in Figure 2: $[\![FP]\!]_{\widetilde{i}}^{\widetilde{j}}$ denotes the translation of the functional process FP into an output process; $[\![G]\!]_{\widetilde{i}}^{\widetilde{j}}$ denotes the translation of the group of functions G into an input process. The translation of $!^{i \leq n}\mathsf{new}\ y_1 : T_1; \ldots; \mathsf{new}\ y_l : T_l; (G_1, \ldots, G_m)$ inputs and outputs on channel $c_{\widetilde{j}}$ so that the context can trigger the generation of random numbers $y_1, \ldots, y_l$. The translation of $(x_1 : T_1, \ldots, x_l : T_l) \rightarrow FP$ inputs the arguments of the function on channel $c_{\widetilde{j}}$ and translates FP, which outputs the result of FP on $c_{\widetilde{j}}$. (In the left-hand side of equivalences, the result FP of functions must simply be a term M.) The observational equivalence $(G_1, \ldots, G_m) \approx (G'_1, \ldots, G'_m)$ is then an abbreviation for $[\![(G_1, \ldots, G_m)]\!] \approx [\![(G'_1, \ldots, G'_m)]\!]$.

For example, the security of a MAC (Definition 1) is represented by the equivalence $L \approx R$ where:

$$
\begin{aligned}
L = {} & !^{i'' \leq n''}\mathsf{new}\ r : T_{mr}; (\\
& \quad !^{i \leq n}(x : \mathit{bitstring}) \rightarrow \mathrm{mac}(x, \mathrm{mkgen}(r)), \\
& \quad !^{i' \leq n'}(m : \mathit{bitstring}, ma : T_{ms}) \rightarrow \\
& \qquad \mathrm{check}(m, \mathrm{mkgen}(r), ma)) \\
R = {} & !^{i'' \leq n''}\mathsf{new}\ r : T_{mr}; (\\
& \quad !^{i \leq n}(x : \mathit{bitstring}) \rightarrow \mathrm{mac}'(x, \mathrm{mkgen}'(r)), \\
& \quad !^{i' \leq n'}(m : \mathit{bitstring}, ma : T_{ms}) \rightarrow \\
& \qquad \mathsf{find}\ u \leq n\ \mathsf{suchthat}\ \mathsf{defined}(x[u]) \wedge (m = x[u]) \\
& \qquad \wedge\, \mathrm{check}'(m, \mathrm{mkgen}'(r), ma)\ \mathsf{then}\ \mathrm{true}\ \mathsf{else}\ \mathrm{false})
\end{aligned}
\tag{$\mathrm{mac}_{\mathrm{eq}}$}
$$

where mac', check', and mkgen' are function symbols with the same types as mac, check, and mkgen respectively. (We use different function symbols on the left- and right-hand sides, just to prevent a repeated application of the transformation induced by this equivalence. Since we add these function symbols, we also add the equation

$$
\begin{aligned}
& \forall r : T_{mr}, \forall m : \mathit{bitstring}, \\
& \quad \mathrm{check}'(m, \mathrm{mkgen}'(r), \mathrm{mac}'(m, \mathrm{mkgen}'(r))) = \mathrm{true}
\end{aligned}
\tag{mac'}
$$

which restates (mac) for mac', check', and mkgen'.) Intuitively, the equivalence $L \approx R$ leaves MAC computations unchanged (except for the use of primed function symbols in R), and allows one to replace a MAC checking $\mathrm{check}(m, \mathrm{mkgen}(r), ma)$ with a lookup in the array x of messages whose mac has been computed with key $\mathrm{mkgen}(r)$: if m is found in the array x and $\mathrm{check}(m, \mathrm{mkgen}(r), ma)$, we return true; otherwise, the check fails (up to negligible probability), so we return false. (If the check succeeded with m not in the array x, the adversary would have forged a MAC.) Obviously, the form of L requires that r is used only to compute or check MACs, for the equivalence to be correct. Formally, the following result shows the correctness of our modeling. It is a fairly easy consequence of Definition 1, and is proved in Appendix E.3.

Proposition 2 *If* $(\mathrm{mkgen}, \mathrm{mac}, \mathrm{check})$ *is a UF-CMA message authentication code,* $I_\eta(\mathrm{mkgen}') = I_\eta(\mathrm{mkgen})$, $I_\eta(\mathrm{mac}') = I_\eta(\mathrm{mac})$, *and* $I_\eta(\mathrm{check}') = I_\eta(\mathrm{check})$, *then* $[\![L]\!] \approx [\![R]\!]$.

Similarly, if $(\mathrm{kgen}, \mathrm{enc}, \mathrm{dec})$ is an IND-CPA symmetric encryption scheme (Definition 2), then we have the following equivalence:

$$
\begin{aligned}
& !^{i' \leq n'}\mathsf{new}\ r : T_r; !^{i \leq n}(x : \mathit{bitstring}) \rightarrow \\
& \qquad \mathsf{new}\ r' : T'_r; \mathrm{enc}(x, \mathrm{kgen}(r), r') \\
\approx {} & !^{i' \leq n'}\mathsf{new}\ r : T_r; !^{i \leq n}(x : \mathit{bitstring}) \rightarrow \\
& \qquad \mathsf{new}\ r' : T'_r; \mathrm{enc}'(\mathrm{Z}(x), \mathrm{kgen}'(r), r')
\end{aligned}
\tag{$\mathrm{enc}_{\mathrm{eq}}$}
$$

where enc' and kgen' are function symbols with the same types as enc and kgen respectively, and $\mathrm{Z} : \mathit{bitstring} \rightarrow \mathit{bitstring}$ is the function that returns a bitstring of the same length as its argument, consisting only of zeroes. Using equations such as $\forall x : T, \mathrm{Z}(\mathrm{T2b}(x)) = \mathrm{Z}_T$, we can prove that $\mathrm{Z}(\mathrm{T2b}(x))$ does not depend on x when x is of a fixed-length type and $\mathrm{T2b} : T \rightarrow \mathit{bitstring}$ is the natural injection. The representation of other primitives can be found in Appendix D.3. The equivalences that formalize the security assumptions on primitives are designed and proved correct by hand from security assumptions in a more standard form, as in the MAC example. Importantly, these manual proofs are done only once for each primitive, and the obtained equivalence can be reused for proving many different protocols automatically.

We use such equivalences $L \approx R$ in order to transform a process Q_0 observationally equivalent to $C[[\![L]\!]]$ into a process Q'_0 observationally equivalent to $C[[\![R]\!]]$, for some evaluation context C. In order to check that $Q_0 \approx^V C[[\![L]\!]]$, the prover uses sufficient conditions, which essentially guarantee that all uses of certain secret variables of Q_0, in a set S, can be implemented by calling functions of L. Let $\mathcal{M}$ be a set of occurrences of terms, corresponding to uses of variables of S. Informally, the prover shows the following properties.

- For each $M \in \mathcal{M}$, there exist a term N_M, which is the result of a function of L, and a substitution σ_M such that $M = \sigma_M N_M$. (Precisely, σ_M applies to the abbreviated form of N_M in which we write x instead of $x[\widetilde{i}]$.) Intuitively, the evaluation of M in Q_0 will correspond to a call to the function with result N_M in $C[[\![L]\!]]$.

- The variables of S do not occur in V, are bound by restrictions in Q_0, and occur only in terms $M = \sigma_M N_M \in \mathcal{M}$ in Q_0, at occurrences that are images by σ_M of variables bound by restrictions in L. (To be precise, the variables of

$$
\begin{aligned}
&[\![(G_1,\dots,G_m)]\!] = [\![G_1]\!]^1 \mid \dots \mid [\![G_m]\!]^m\\
&[\![!^{i\le n}\mathsf{new}\ y_1:T_1;\dots;\mathsf{new}\ y_l:T_l;(G_1,\dots,G_m)]\!]^{\widetilde{j}}_{\widetilde{i}} =\\
&\qquad !^{i\le n}c_{\widetilde{j}}[\widetilde{i},i]();\mathsf{new}\ y_1:T_1;\dots;\mathsf{new}\ y_l:T_l;\overline{c_{\widetilde{j}}[\widetilde{i},i]}\langle\rangle;([\![G_1]\!]^{\widetilde{j},1}_{\widetilde{i},i}\mid\dots\mid[\![G_m]\!]^{\widetilde{j},m}_{\widetilde{i},i})\\
&[\![(x_1:T_1,\dots,x_l:T_l)\to FP]\!]^{\widetilde{j}}_{\widetilde{i}} = c_{\widetilde{j}}[\widetilde{i}](x_1:T_1,\dots,x_l:T_l);[\![FP]\!]^{\widetilde{j}}_{\widetilde{i}}\\
&[\![M]\!]^{\widetilde{j}}_{\widetilde{i}} = \overline{c_{\widetilde{j}}[\widetilde{i}]}\langle M\rangle\\
&[\![\mathsf{new}\ x[\widetilde{i}]:T;FP]\!]^{\widetilde{j}}_{\widetilde{i}} = \mathsf{new}\ x[\widetilde{i}]:T;[\![FP]\!]^{\widetilde{j}}_{\widetilde{i}}\\
&[\![\mathsf{let}\ x[\widetilde{i}]:T=M\ \mathsf{in}\ FP]\!]^{\widetilde{j}}_{\widetilde{i}} = \mathsf{let}\ x[\widetilde{i}]:T=M\ \mathsf{in}\ [\![FP]\!]^{\widetilde{j}}_{\widetilde{i}}\\
&[\![\mathsf{find}\ (\textstyle\bigoplus_{j=1}^m \widetilde{u_j}[\widetilde{i}]\le\widetilde{n_j}\ \mathsf{suchthat}\ \mathsf{defined}(M_{j1},\dots,M_{jl_j})\wedge M_j\ \mathsf{then}\ FP_j)\ \mathsf{else}\ FP]\!]^{\widetilde{j}}_{\widetilde{i}} =\\
&\quad \mathsf{find}\ (\textstyle\bigoplus_{j=1}^m \widetilde{u_j}[\widetilde{i}]\le\widetilde{n_j}\ \mathsf{suchthat}\ \mathsf{defined}(M_{j1},\dots,M_{jl_j})\wedge M_j\ \mathsf{then}\ [\![FP_j]\!]^{\widetilde{j}}_{\widetilde{i}})\ \mathsf{else}\ [\![FP]\!]^{\widetilde{j}}_{\widetilde{i}}
\end{aligned}
$$

where $c_{\widetilde{j}}$ are pairwise distinct channels, $\widetilde{i} = i_1,\dots,i_{l'}$, and $\widetilde{j} = j_0,\dots,j_{l'}$.

Figure 2: Translation from functional processes to processes

S are also allowed to occur at the root of defined conditions; in that case, their value does not matter, just the fact that they are defined.)

- Let $\widetilde{i}$ and $\widetilde{i'}$ be the sequences of current replication indices at N_M in L and at M in Q_0, respectively. The prover shows that there exists a function mapIdx_M that maps the array indices at M in Q_0 to the array indices at N_M in L: the evaluation of M when $\widetilde{i'} = \widetilde{a}$ will correspond in $C[[\![L]\!]]$ to the evaluation of N_M when $\widetilde{i} = \mathrm{mapIdx}_M(\widetilde{a})$. Thus, σ_M and mapIdx_M induce a correspondence between terms and variables of Q_0 and variables of L: for all $M \in \mathcal{M}$, for all $x[\widetilde{i''}]$ that occur in N_M, $(\sigma_M x)\{\widetilde{a}/\widetilde{i'}\}$ corresponds to $x[\widetilde{i''}]\{\mathrm{mapIdx}_M(\widetilde{a})/\widetilde{i}\}$, that is, $(\sigma_M x)\{\widetilde{a}/\widetilde{i'}\}$ in a trace of Q_0 has the same value as $x[\widetilde{i''}]\{\mathrm{mapIdx}_M(\widetilde{a})/\widetilde{i}\}$ in the corresponding trace of $C[[\![L]\!]]$ ($\widetilde{i''}$ is a prefix of $\widetilde{i}$). We detail below conditions that this correspondence has to satisfy.

For example, consider a process Q_0 that contains $M_1 = \mathrm{enc}(M'_1, \mathrm{kgen}(x_r), x'_r[i_1])$ under a replication $!^{i_1\le n_1}$ and $M_2 = \mathrm{enc}(M'_2, \mathrm{kgen}(x_r), x''_r[i_2])$ under a replication $!^{i_2\le n_2}$, where x_r, x'_r, x''_r are bound by restrictions. Let $S = \{x_r, x'_r, x''_r\}$, $\mathcal{M} = \{M_1, M_2\}$, and $N_{M_1} = N_{M_2} = \mathrm{enc}(x[i', i], \mathrm{kgen}(r[i']), r'[i', i])$. The functions mapIdx_{M_1} and mapIdx_{M_2} are defined by

$$
\begin{aligned}
&\mathrm{mapIdx}_{M_1}(a_1) = (1, a_1) \text{ for } a_1 \in [1, I_\eta(n_1)]\\
&\mathrm{mapIdx}_{M_2}(a_2) = (1, a_2 + I_\eta(n_1)) \text{ for } a_2 \in [1, I_\eta(n_2)]
\end{aligned}
$$

Then $M'_1\{a_1/i_1\}$ corresponds to $x[1, a_1]$, x_r to $r[1]$, $x'_r[a_1]$ to $r'[1, a_1]$, $M'_2\{a_2/i_2\}$ to $x[1, a_2 + I_\eta(n_1)]$, and $x''_r[a_2]$ to $r'[1, a_2 + I_\eta(n_1)]$. The functions mapIdx_{M_1} and mapIdx_{M_2} are such that $x_{r'}[a_1]$ and $x_{r''}[a_2]$ never correspond to the same cell of r'; indeed, $x_{r'}[a_1]$ and $x_{r''}[a_2]$ are independent random numbers in Q_0, so their images in $C[[\![L]\!]]$ must also be independent random numbers.

The above correspondence must satisfy the following soundness conditions:

- when x is a function argument in L, the term that corresponds to $x[\widetilde{a'}]$ must have the same type as $x[\widetilde{a'}]$, and when two terms correspond to the same $x[\widetilde{a'}]$, they must evaluate to the same value;
- when x is bound by $\mathsf{new}\ x : T$ in L, the term that corresponds to $x[\widetilde{a'}]$ must evaluate to $z[\widetilde{a''}]$ where $z \in S$ and z is bound by $\mathsf{new}\ z : T$ in Q_0, and the relation that associates $z[\widetilde{a''}]$ to $x[\widetilde{a'}]$ is an injective function (so that independent random numbers in L correspond to independent random numbers in Q_0).

It is easy to check that, in the previous example, these conditions are satisfied.

The transformation of Q_0 into Q'_0 consists in two steps. First, we replace the restrictions that define variables of S with restrictions that define fresh variables corresponding to variables bound by new in R. The correspondence between variables of Q_0 and variables of $C[[\![L]\!]]$ is extended to include these fresh variables. Second, we reorganize Q_0 so that each evaluation of a term $M \in \mathcal{M}$ first stores the values of the arguments $x_1, \dots, x_m$ of the function $(x_1 : T_1, \dots, x_m : T_m) \to N_M$ in fresh variables, then computes N_M and stores its result in a fresh variable, and uses this variable instead of M; then we simply replace the computation of N_M with the corresponding functional process of R, taking into account the correspondence of variables.

The full formal description of this transformation is given Appendix D.1. The following proposition shows the soundness of the transformation and is proved in Appendix E.4.

Proposition 3 *Let Q_0 be a process that satisfies Invariants 1, 2, and 3 and Q'_0 the process obtained from Q_0 by the above transformation. Then Q'_0 satisfies Invariants 1, 2, and 3 and, if $[\![L]\!] \approx [\![R]\!]$ for all polynomials $\mathrm{maxlen}_\eta(c_{j_0,\dots,j_l})$ and $I_\eta(n)$ where n is any replication bound of L or R, then $Q_0 \approx^V Q'_0$.*

Example 4 In order to treat Example 1, the prover is given as input the indication that T_{mr}, T_r, T'_r, and T_k are fixed-length types; the type declarations for the functions $\mathrm{mkgen}, \mathrm{mkgen}'$:

$T_{mr} \rightarrow T_{mk}$, $\mathrm{mac}, \mathrm{mac}' : \mathit{bitstring} \times T_{mk} \rightarrow T_{ms}$, $\mathrm{check}, \mathrm{check}' : \mathit{bitstring} \times T_{mk} \times T_{ms} \rightarrow \mathit{bool}$, $\mathrm{kgen}, \mathrm{kgen}' : T_r \rightarrow T_k$, $\mathrm{enc}, \mathrm{enc}' : \mathit{bitstring} \times T_k \times T'_r \rightarrow T_e$, $\mathrm{dec} : T_e \times T_k \rightarrow \mathit{bitstring}_\perp$, $\mathrm{k2b} : T_k \rightarrow \mathit{bitstring}$, $\mathrm{i}_\perp : \mathit{bitstring} \rightarrow \mathit{bitstring}_\perp$, $\mathrm{Z} : \mathit{bitstring} \rightarrow \mathit{bitstring}$, and the constant $\mathrm{Z}_k : \mathit{bitstring}$; the equations (mac), (mac′), (enc), and $\forall x : T_k, \mathrm{Z}(\mathrm{k2b}(x)) = \mathrm{Z}_k$ (which expresses that all keys have the same length); the indication that k2b and $\mathrm{i}_\perp$ are poly-injective (which generates the equations (k2b) and similar equations for $\mathrm{i}_\perp$); equivalences $L \approx R$ for MAC ($\mathrm{mac}_{\mathrm{eq}}$) and encryption ($\mathrm{enc}_{\mathrm{eq}}$); and the process Q_0 of Example 1.

The prover first applies **RemoveAssign**(x_{mk}) to the process Q_0 of Example 1, as described in Example 2. The process can then be transformed using the security of the MAC. Let $S = \{x'_r\}$, $M_1 = \mathrm{mac}(x_m[i], \mathrm{mkgen}(x'_r))$, $M_2 = \mathrm{check}(x'_m[i'], \mathrm{mkgen}(x'_r), x_{ma}[i'])$, and $\mathcal{M} = \{M_1, M_2\}$. We have $N_{M_1} = \mathrm{mac}(x[i'', i], \mathrm{mkgen}(r[i'']))$, $N_{M_2} = \mathrm{check}(m[i'', i'], \mathrm{mkgen}(r[i'']), ma[i'', i'])$, $\mathrm{mapIdx}_{M_1}(a_1) = (1, a_1)$, and $\mathrm{mapIdx}_{M_2}(a_2) = (1, a_2)$, so $x_m[a_1]$ corresponds to $x[1, a_1]$, x'_r to $r[1]$, $x'_m[a_2]$ to $m[1, a_2]$, and $x_{ma}[a_2]$ to $ma[1, a_2]$.

After transformation, we get the following process Q'_0:

$$
\begin{aligned}
&Q'_0 = \mathit{start}(); \mathsf{new}\ x_r : T_r; \mathsf{let}\ x_k : T_k = \mathrm{kgen}(x_r)\ \mathsf{in} \\
&\quad \mathsf{new}\ x'_r : T_{mr}; \overline{c}\langle\rangle; (Q'_A \mid Q'_B) \\
&Q'_A = !^{i \le n} c_A[i](); \mathsf{new}\ x'_k : T_k; \mathsf{new}\ x''_r : T'_r; \\
&\quad \mathsf{let}\ x_m : \mathit{bitstring} = \mathrm{enc}(\mathrm{k2b}(x'_k), x_k, x''_r)\ \mathsf{in} \\
&\quad \overline{c_A[i]}\langle x_m, \mathrm{mac}'(x_m, \mathrm{mkgen}'(x'_r))\rangle \\
&Q'_B = !^{i' \le n} c_B[i'](x'_m, x_{ma}); \\
&\quad \mathsf{find}\ u \le n\ \mathsf{suchthat}\ \mathsf{defined}(x_m[u]) \wedge x'_m = x_m[u] \wedge \\
&\qquad\qquad \mathrm{check}'(x'_m, \mathrm{mkgen}'(x'_r), x_{ma})\ \mathsf{then} \\
&\quad\quad (\\
&\quad\quad\quad \mathsf{if}\ \mathrm{true}\ \mathsf{then}\ \mathsf{let}\ \mathrm{i}_\perp(\mathrm{k2b}(x''_k)) = \mathrm{dec}(x'_m, x_k)\ \mathsf{in} \\
&\qquad\qquad\qquad \overline{c_B[i']}\langle\rangle \\
&\quad\quad) \\
&\quad \mathsf{else} \\
&\quad\quad (\\
&\quad\quad\quad \mathsf{if}\ \mathrm{false}\ \mathsf{then}\ \mathsf{let}\ \mathrm{i}_\perp(\mathrm{k2b}(x''_k)) = \mathrm{dec}(x'_m, x_k)\ \mathsf{in} \\
&\qquad\qquad\qquad \overline{c_B[i']}\langle\rangle \\
&\quad\quad)
\end{aligned}
$$

The initial definition of x'_r is removed and replaced with a new definition, which we still call x'_r. The term $\mathrm{mac}(x_m, \mathrm{mkgen}(x'_r))$ is replaced with $\mathrm{mac}'(x_m, \mathrm{mkgen}'(x'_r))$. The term $\mathrm{check}(x'_m, \mathrm{mkgen}(x'_r), x_{ma})$ becomes find $u \le n$ suchthat defined$(x_m[u]) \wedge x'_m = x_m[u] \wedge \mathrm{check}'(x'_m, \mathrm{mkgen}'(x'_r), x_{ma})$ then true else false, which yields Q'_B after transformation of functional processes into processes. The process looks up the message x'_m in the array x_m, which contains the messages whose MAC has been computed with key $\mathrm{mkgen}(x'_r)$. If the MAC of x'_m has never been computed, the check always fails (it returns false) by the definition of security of the MAC. Otherwise, it returns true when $\mathrm{check}'(x'_m, \mathrm{mkgen}'(x'_r), x_{ma})$.

After applying **Simplify**, Q'_A is unchanged and Q'_B becomes

$$
\begin{aligned}
&Q''_B = !^{i' \le n} c_B[i'](x'_m, x_{ma}); \\
&\quad \mathsf{find}\ u \le n\ \mathsf{suchthat}\ \mathsf{defined}(x_m[u], x'_k[u]) \wedge \\
&\qquad x'_m = x_m[u] \wedge \mathrm{check}'(x'_m, \mathrm{mkgen}'(x'_r), x_{ma})\ \mathsf{then} \\
&\quad \mathsf{let}\ x''_k : T_k = x'_k[u]\ \mathsf{in}\ \overline{c_B[i']}\langle\rangle
\end{aligned}
$$

First, the tests if true then ... and if false then ... are simplified. The term $\mathrm{dec}(x'_m, x_k)$ is simplified knowing $x'_m = x_m[u]$ by the find condition, $x_m[u] = \mathrm{enc}(\mathrm{k2b}(x'_k[u]), x_k, x''_r[u])$ by the assignment that defines x_m, $x_k = \mathrm{kgen}(x_r)$ by the assignment that defines x_k, and $\mathrm{dec}(\mathrm{enc}(m, \mathrm{kgen}(r), r'), \mathrm{kgen}(r)) = \mathrm{i}_\perp(m)$ by (enc). So we have $\mathrm{dec}(x'_m, x_k) = \mathrm{i}_\perp(\mathrm{k2b}(x'_k[u]))$. By injectivity of $\mathrm{i}_\perp$ and k2b, the assignment to x''_k simply becomes $x''_k = x'_k[u]$, using the equations $\forall x : \mathit{bitstring}, \mathrm{i}_\perp^{-1}(\mathrm{i}_\perp(x)) = x$ and $\forall x : T_k, \mathrm{k2b}^{-1}(\mathrm{k2b}(x)) = x$.

After applying **RemoveAssign**(x_k), we apply the security of encryption: $\mathrm{enc}(\mathrm{k2b}(x'_k), \mathrm{kgen}(x_r), x''_r)$ becomes $\mathrm{enc}'(\mathrm{Z}(\mathrm{k2b}(x'_k)), \mathrm{kgen}(x_r), x''_r)$. After **Simplify**, it becomes $\mathrm{enc}'(\mathrm{Z}_k, \mathrm{kgen}(x_r), x''_r)$, using $\forall x : T_k, \mathrm{Z}(\mathrm{k2b}(x)) = \mathrm{Z}_k$ (which expresses that all keys have the same length).

So we obtain the following game:

$$
\begin{aligned}
&Q''_0 = \mathit{start}(); \mathsf{new}\ x_r : T_r; \mathsf{new}\ x'_r : T_{mr}; \overline{c}\langle\rangle; (Q''_A \mid Q''_B) \\
&Q''_A = !^{i \le n} c_A[i](); \mathsf{new}\ x'_k : T_k; \mathsf{new}\ x''_r : T'_r; \\
&\quad \mathsf{let}\ x_m : \mathit{bitstring} = \mathrm{enc}(\mathrm{Z}_k, \mathrm{kgen}(x_r), x''_r)\ \mathsf{in} \\
&\quad \overline{c_A[i]}\langle x_m, \mathrm{mac}'(x_m, \mathrm{mkgen}'(x'_r))\rangle
\end{aligned}
$$

where Q''_B remains as above.

Using arrays instead of lists simplifies this transformation: we do not need to add instructions that insert values in the list, since all variables are always implicitly arrays. Moreover, if there are several occurrences of $\mathrm{mac}(x_i, k)$ with the same key in the initial process, each $\mathrm{check}(m_j, k, ma_j)$ is replaced with a find with one branch for each occurrence of mac. Therefore, the prover distinguishes automatically the cases in which the checked MAC ma_j comes from each occurrence of mac, that is, it distinguishes cases depending on the value of i such that $m_j = x_i$. Typically, distinguishing these cases is useful in the following steps of the proof of the protocol. (A similar situation arises for other cryptographic primitives specified using find.)

4 Criteria for Proving Secrecy Properties

Let us now define syntactic criteria that allow us to prove secrecy properties of protocols. The proofs for these results can be found in Appendix E.5.

Definition 4 (One-session secrecy) The process Q *preserves the one-session secrecy of* x when $Q \mid Q_x \approx Q \mid Q'_x$, where

$$
\begin{aligned}
Q_x = {}& c(u_1 : [1, n_1], \ldots, u_m : [1, n_m]); \\
& \mathsf{if}\ \mathsf{defined}(x[u_1, \ldots, u_m])\ \mathsf{then}\ \overline{c}\langle x[u_1, \ldots, u_m]\rangle
\end{aligned}
$$

$$Q'_x = c(u_1 : [1, n_1], \ldots, u_m : [1, n_m]);$$
$$\quad \mathsf{if\ defined}(x[u_1, \ldots, u_m]) \mathsf{\ then\ new\ } y : T; \overline{c}\langle y \rangle$$

$c \notin \mathrm{fc}(Q)$, $u_1, \ldots, u_m, y \notin \mathrm{var}(Q)$, and $\mathcal{E}(x) = [1, n_1] \times \ldots \times [1, n_m] \to T$.

Intuitively, the adversary cannot distinguish a process that outputs the value of the secret from one that outputs a random number. The adversary performs a single test query, modeled by Q_x and Q'_x.

Proposition 4 (One-session secrecy) *Consider a process Q such that there exists a set of variables S such that 1) the definitions of x are either restrictions* $\mathsf{new}\ x[\widetilde{i}] : T$ *and $x \in S$, or assignments* $\mathsf{let}\ x[\widetilde{i}] : T = z[M_1, \ldots, M_l]$ *where z is defined by restrictions* $\mathsf{new}\ z[i'_1, \ldots, i'_l] : T$*, and $z \in S$, and 2) all accesses to variables $y \in S$ in Q are of the form "*$\mathsf{let}\ y'[\widetilde{i}] : T' = y[M_1, \ldots, M_l]$*" with $y' \in S$. Then $Q \mid Q_x \approx_0 Q \mid Q'_x$, hence Q preserves the one-session secrecy of x.*

Intuitively, only the variables in S depend on the restriction that defines x; the sent messages and the control flow of the process are independent of x, so the adversary obtains no information on x. In the implementation, the set S is computed by fixpoint iteration, starting from x or z and adding variables y' defined by "$\mathsf{let}\ y'[\widetilde{i}] : T' = y[M_1, \ldots, M_l]$" when $y \in S$.

Definition 5 (Secrecy) The process Q *preserves the secrecy of* x when $Q \mid R_x \approx Q \mid R'_x$, where

$$R_x = !^{i \leq n} c(u_1 : [1, n_1], \ldots, u_m : [1, n_m]);$$
$$\quad \mathsf{if\ defined}(x[u_1, \ldots, u_m]) \mathsf{\ then\ } \overline{c}\langle x[u_1, \ldots, u_m] \rangle$$
$$R'_x = !^{i \leq n} c(u_1 : [1, n_1], \ldots, u_m : [1, n_m]);$$
$$\quad \mathsf{if\ defined}(x[u_1, \ldots, u_m]) \mathsf{\ then}$$
$$\quad \mathsf{find\ } u' \leq n \mathsf{\ suchthat\ defined}(y[u'], u_1[u'], \ldots, u_m[u'])$$
$$\qquad \wedge\, u_1[u'] = u_1 \wedge \ldots \wedge u_m[u'] = u_m$$
$$\quad \mathsf{then\ } \overline{c}\langle y[u'] \rangle \mathsf{\ else\ new\ } y : T; \overline{c}\langle y \rangle$$

$c \notin \mathrm{fc}(Q)$, $u_1, \ldots, u_m, u', y \notin \mathrm{var}(Q)$, $\mathcal{E}(x) = [1, n_1] \times \ldots \times [1, n_m] \to T$, and $I_\eta(n) \geq I_\eta(n_1) \times \ldots \times I_\eta(n_m)$.

Intuitively, the adversary cannot distinguish a process that outputs the value of the secret for several indices from one that outputs independent random numbers. In this definition, the adversary can perform several test queries, modeled by R_x and R'_x. This corresponds to the "real-or-random" definition of security [4]. (As shown in [4], this notion is stronger than the more standard approach in which the adversary can perform a single test query and some reveal queries, which always reveal $x[u_1, \ldots, u_m]$.)

Proposition 5 (Secrecy) *Assume that Q satisfies the hypothesis of Proposition 4.*

When $\mathcal{T}$ is a trace of $C[Q]$ for some evaluation context C, we define $\mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a}])$, the defining restriction of $x[\widetilde{a}]$ in trace $\mathcal{T}$, as follows: if $x[\widetilde{a}]$ is defined by $\mathsf{new}\ x[\widetilde{a}] : T$ *in $\mathcal{T}$, $\mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a}]) = x[\widetilde{a}]$; if $x[\widetilde{a}]$ is defined by* $\mathsf{let}\ x[\widetilde{a}] : T = z[M_1, \ldots, M_l]$, $\mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a}]) = z[a'_1, \ldots, a'_l]$ *where $E, M_k \Downarrow a'_k$ for all $k \leq l$ and E is the environment in $\mathcal{T}$ at the definition of $x[\widetilde{a}]$.*

Assume that for all evaluation contexts C acceptable for Q, 0, $\{x\}$, the probability $\Pr[\exists(\mathcal{T}, \widetilde{a}, \widetilde{a'}), C[Q]$ reduces according to $\mathcal{T} \wedge \widetilde{a} \neq \widetilde{a'} \wedge \mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a}]) = \mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a'}])]$ is negligible. Then Q preserves the secrecy of x.

The last hypothesis can be verified using the same equational prover as for **Simplify** in Section 3.1, as detailed in Appendix E.2. Intuitively, this hypothesis guarantees that when $\widetilde{a} \neq \widetilde{a'}$, we have $\mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a}]) \neq \mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a'}])$ except in cases of negligible probability, so $x[\widetilde{a}]$ and $x[\widetilde{a'}]$ are defined by different restrictions, so they are independent random numbers.

As we show in [19], this notion of secrecy composed with correspondence assertions [49] can be used to prove security of a key exchange. (Correspondence assertions are properties of the form "if some event $e(\widetilde{M})$ has been executed then some events $e_i(\widetilde{M_i})$ for $i \leq m$ have been executed". We have recently implemented the verification of correspondence assertions in CryptoVerif [19].)

Lemma 2 *If $Q \approx^{\{x\}} Q'$ and Q preserves the one-session secrecy of x then Q' preserves the one-session secrecy of x. The same result holds for secrecy.*

We can then apply the following technique. When we want to prove that Q_0 preserves the (one-session) secrecy of x, we transform Q_0 by the transformations described in Section 3 with $V = \{x\}$. By Propositions 1 and 3, we obtain a process Q'_0 such that $Q_0 \approx^V Q'_0$. We use Propositions 4 or 5 to show that Q'_0 preserves the (one-session) secrecy of x and finally conclude that Q_0 also preserves the (one-session) secrecy of x by Lemma 2.

Example 5 After the transformations of Example 4, the only variable access to x'_k in the considered process is $\mathsf{let}\ x''_k : T_k = x'_k[u]$ and x''_k is not used in the considered process. So by Proposition 4, the considered process preserves the one-session secrecy of x''_k (with $S = \{x'_k, x''_k\}$). By Lemma 2, the process of Example 1 also preserves the one-session secrecy of x''_k. However, this process does not preserve the secrecy of x''_k, because the adversary can force several sessions of B to use the same key x''_k, by replaying the message sent by A. (Accordingly, the hypothesis of Proposition 5 is not satisfied.)

The criteria given in this section might seem restrictive, but in fact, they should be sufficient for all protocols, provided the previous transformation steps are powerful enough to transform the protocol into a simpler protocol, on which these criteria can then be applied.

5 Proof Strategy

Up to now, we have described the available game transformations. Next, we explain how we organize these transformations in order to prove protocols.

At the beginning of the proof and after each successful cryptographic transformation (that is, a transformation of Section 3.2), the prover executes **Simplify** and tests whether the desired security properties are proved, as described in Section 4. If so, it stops.

In order to perform the cryptographic transformations and the other syntactic transformations, our proof strategy relies of the idea of advice. Precisely, the prover tries to execute each available cryptographic transformation in turn. When such a cryptographic transformation fails, it returns some syntactic transformations that could make the desired transformation work. (These are the advised transformations.) Then the prover tries to perform these syntactic transformations. If they fail, they may also suggest other advised transformations, which are then executed. When the syntactic transformations finally succeed, we retry the desired cryptographic transformation, which may succeed or fail, perhaps with new advised transformations, and so on.

The prover determines the advised transformations as follows:

- Assume that we try to execute a cryptographic transformation, and need to recognize a certain term M of L, but we find in Q_0 only part of M, the other parts being variable accesses $x[\dots]$ while we expect function applications. In this case, we advise **RemoveAssign**(x). For example, if Q_0 contains $\mathrm{enc}(M', x_k, x'_r)$ and we look for $\mathrm{enc}(x_m, \mathrm{kgen}(x_r), x_{r'})$, we advise **RemoveAssign**(x_k). If Q_0 contains $\mathsf{let}\ x_k = \mathrm{mkgen}(x_r)$ and we look for $\mathrm{mac}(x_m, \mathrm{mkgen}(x_r))$, we also advise **RemoveAssign**(x_k). (The transformation of Example 2 is advised for this reason.)

- When we try to execute **RemoveAssign**(x), x has several definitions, and there are accesses to variable x guarded by find in Q_0, we advise **SArename**(x).

- When we check whether x is secret or one-session secret, we have an assignment $\mathsf{let}\ x[\widetilde{i}] : T = y[\widetilde{M}]\ \mathsf{in}\ P$, and there is at least one assignment defining y, we advise **RemoveAssign**(y).

  When we check whether x is secret or one-session secret, we have an assignment $\mathsf{let}\ x[\widetilde{i}] : T = y[\widetilde{M}]\ \mathsf{in}\ P$, y is defined by restrictions, y has several definitions, and some variable accesses to y are not of the form $\mathsf{let}\ y'[\widetilde{i'}] : T = y[\widetilde{M'}]\ \mathsf{in}\ P'$, we advise **SArename**$(y)$.

These pieces of advice are the only ones we use, but one may obviously extend them if needed.

6 Experimental Results

We have successfully tested our prover on a number of protocols given in the literature. All these protocols have been tested in a configuration in which the honest participants are willing to run sessions with the adversary, and we prove secrecy of keys for sessions between honest participants. In these examples, shared-key encryption is encoded using a symmetric encryption scheme and a MAC as in Example 1, public-key encryption is assumed to be IND-CCA2 (indistinguishability under adaptive chosen-ciphertext attacks) [14], public-key signature is assumed to be UF-CMA (unforgeability under chosen message attacks).

For each proof, the prover outputs the sequence of games it has built, a succinct explanation of the transformation performed between consecutive games, and an indication whether the proof succeeded or failed. When the proof fails, the prover still outputs a sequence of games, but the last game of this sequence does not show the desired property and cannot be transformed further by the prover. Manual inspection of this game often makes it possible to understand why the proof failed: because there is an attack (if there is an attack on the last game), because of a limitation of the prover (if it should in fact be able to prove the property or to transform the game further), for other reasons (such as the protocol cannot be proved from the given assumptions; this situation may not lead immediately to a practical attack in the computational model).

Otway-Rees [43] We automatically prove the secrecy of the exchanged key.

Yahalom [21] For the original version of the protocol, our prover cannot show the one-session secrecy of the exchanged key, because the protocol is not secure, at least using encrypt-then-MAC as definition of encryption. Indeed, there is a confirmation round $\{N_B\}_K$ where K is the exchanged key. This message may reveal some information on K. After removing this confirmation round, our prover shows the one-session secrecy of K. However, it cannot show the secrecy of K, since in the absence of a confirmation round, the adversary may force several sessions of Yahalom to use the same key.

Needham-Schroeder shared-key [41] As in the Yahalom protocol, a key confirmation round may reveal some information on the key. After removing this round, our prover shows the one-session secrecy of the exchanged key. It does not prove the secrecy of the exchanged key, because the adversary may force several sessions of the protocol to use the same key. Our prover shows the secrecy for the corrected version [42].

Denning-Sacco public-key [26] Our prover cannot show the one-session secrecy of the exchanged key, since there is an attack against this protocol [2]. The one-session secrecy of the exchanged key is proved for the corrected version [2]. Secrecy is not proved since the adversary can force several sessions of the protocol to use the same key. (We do not model timestamps in this protocol.) In contrast to the previous examples, we give the main proof steps to the prover manually, as follows:

```
SArename Rkey
crypto enc rkB
crypto sign rkS
crypto sign rkA
success
```

The variable `Rkey` defines a table of public keys and is assigned at three places, corresponding to principals A and B, and to other principals defined by the adversary (like the variable k' in Example 3). The instruction `SArename Rkey` allows us to distinguish these three cases. The instruction `crypto enc rkB` means that the prover should apply the definition of security of encryption (primitive `enc`), for the key generated from random number `rkB`. The instruction `success` means that prover should check whether the desired security properties are proved.

Needham-Schroeder public-key [41] This protocol is an authentication protocol. Since our prover cannot check authentication yet, we transform it into a key exchange protocol in several ways, by choosing for the key either one of the nonces N_A and N_B shared between A and B, or $H(N_A, N_B)$ where H is a hash function (in the random oracle model). When the key is $H(N_A, N_B)$, the one-session secrecy of the key cannot be proved for the original protocol, due to the well-known attack [36]. For the corrected version [36], our prover shows secrecy of the key $H(N_A, N_B)$. For both the original and the corrected versions, the prover cannot prove the one-session secrecy of N_A or N_B. For N_B, the failure of the proof corresponds to an attack: the adversary can check whether it is given N_B or a random number by sending $\{N'_B\}_{pk_B}$ to B as the last message of the protocol: B accepts if and only if $N'_B = N_B$. For N_A, the failure of the proof comes from limitations of our prover: the prover cannot take into account that N_A is accepted only after all messages that contain N_A have been sent, which prevents the previous attack. (This is the only case in our examples where the failure of the proof comes from limitations of the prover. This problem could probably be solved by improving the transformation **Simplify**.) Like for the Denning-Sacco protocol, we provided the main proof steps to the prover manually, as follows when the distributed key is N_A or N_B:

```
SArename Rkey
crypto sign rkS
crypto enc rkA
crypto enc rkB
success
```

When the distributed key is $H(N_A, N_B)$, the proof is as follows:

```
SArename Rkey
crypto sign rkS
crypto enc rkA
crypto enc rkB
crypto hash
SArename Na_39
simplify
success
```

The total runtime for all these tests is 77 s on a Pentium M 1.8 GHz, for version 1.03 of our prover CryptoVerif. These examples are included in the CryptoVerif distribution available at `http://www.di.ens.fr/~blanchet/cryptoc-eng.html`.

7 Related Work

Results that show the soundness of the Dolev-Yao model with respect to the computational model, e.g. [24,29,39], make it possible to use Dolev-Yao provers in order to prove protocols in the computational model. However, these results have limitations, in particular in terms of allowed cryptographic primitives (they must satisfy strong security properties so that they correspond to Dolev-Yao style primitives), and they require some restrictions on protocols (such as the absence of key cycles).

Several frameworks exist for formalizing proofs of protocols in the computational model. Backes, Pfitzmann, and Waidner [7, 9, 10] have designed an abstract cryptographic library including symmetric and public-key encryption, message authentication codes, signatures, and nonces and shown its soundness with respect to computational primitives, under arbitrary active attacks. Backes and Pfitzmann [8] relate the computational and formal notions of secrecy in the framework of this library. Recently, this framework has been used for a computationally-sound machine-checked proof of the Needham-Schroeder-Lowe protocol [47]. Canetti [22] introduced the notion of universal composability. With Herzog [23], they show how a Dolev-Yao-style symbolic analysis can be used to prove security properties of protocols within the framework of universal composability, for a restricted class of protocols using public-key encryption as only cryptographic primitive. Then, they use the automatic Dolev-Yao verification tool Proverif [18] for verifying protocols in this framework. Lincoln, Mateus, Mitchell, Mitchell, Ramanathan, Scedrov, and Teague [34, 35, 37, 40, 44] developed a probabilistic polynomial-time calculus for the analysis of security protocols. They define a notion of process equivalence for this calculus, derive compositionality properties, and define an equational proof system for this calculus. Datta, Derek, Mitchell, Shmatikov, and Turuani [25] have designed a computationally sound logic that enables them to prove computational security properties using a logical deduction system. The frameworks mentioned in this paragraph can be used to prove security properties of protocols in the computational sense, but, except for [23] which relies on a Dolev-Yao prover and for the machine-checked proofs of [47], they have not been mechanized up to now, as far as we know.

Laud [31] designed an automatic analysis for proving secrecy for protocols using shared-key encryption, with passive adversaries. He extended it [32] to active adversaries, but with only one session of the protocol. This work is the closest to ours. We extend it considerably by handling more primitives and a polynomial number of sessions.

Recently, Laud [33] designed a type system for proving security protocols in the computational model. This type system handles shared-key and public-key encryption, with an unbounded number of sessions. This system relies on the Backes-Pfitzmann-Waidner library. A type inference algorithm is given in [6].

Barthe, Cerderquist, and Tarento [11, 48] have formalized the generic model and the random oracle model in the interactive theorem prover Coq, and proved signature schemes in this framework. In contrast to our specialized prover, proofs in generic interactive theorem provers require a lot of human effort, in order to build a detailed enough proof for the theorem prover to check it.

Halevi [27] explains that implementing an automatic prover based on sequences of games would be useful and suggests ideas in this direction, but does not actually implement one.

8 Conclusion

This paper presents a prover for security protocols sound in the computational model. This prover works with no or very little help from the user, can handle a wide variety of cryptographic primitives in a generic way, and produces proofs valid for a polynomial number of sessions in the presence of an active adversary. Thus, it represents important progress with respect to previous work in this area.

We have recently extended our prover to provide exact security proofs (that is, proofs with an explicit probability of an attack, instead of the asymptotic result that this probability is negligible) [20] and to prove correspondence assertions [19]. In the future, it would also be interesting to handle even more cryptographic primitives, such as Diffie-Hellman key agreements. (The equivalence $!^{i\leq n}\mathsf{new}\ a : T; \mathsf{new}\ b : T; (() \to g^a, () \to g^b, () \to g^{ab}) \approx\ !^{i\leq n}\mathsf{new}\ a : T; \mathsf{new}\ b : T; \mathsf{new}\ c : T; (() \to g^a, () \to g^b, () \to g^c)$ models the decisional Diffie-Hellman assumption. However, it is not sufficient for our prover to handle protocols that use Diffie-Hellman key agreements, because the corresponding cryptographic transformation would require g^{ab} to be formed only for a and b chosen in the same copy of a single replicated process, which is typically not the case: a and b are chosen by two different participants of the protocol. So a more involved equivalence is needed, and in fact the language of equivalences that we use to specify the security properties of primitives will need to be extended.)

The essential idea of simulating proofs by sequences of games in an automatic tool can be applied to any protocol or cryptographic scheme. However, our tool applies in a fairly direct way the security assumptions on the primitives and cannot perform deep mathematical reasoning. Therefore, it is best suited for proving security protocols that use rather high-level primitives such as encryption and signatures. It is more limited for proving the security of such primitives from lower-level primitives, since more subtle mathematical arguments are often needed.

Acknowledgments

I warmly thank David Pointcheval for his advice and explanations of the computational proofs of protocols. This project would not have been possible without him. I also thank Jacques Stern for initiating this work. This work was partly supported by the ANR project ARA SSIA Formacrypt.

References

[1] M. Abadi and J. Jürjens. Formal eavesdropping and its computational interpretation. In N. Kobayashi and B. Pierce, editors, *Theoretical Aspects of Computer Software (TACS'01)*, volume 2215 of *Lecture Notes on Computer Science*, pages 82–94, Sendai, Japan, Oct. 2001. Springer.

[2] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, Jan. 1996.

[3] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[4] M. Abdalla, P.-A. Fouque, and D. Pointcheval. Password-based authenticated key exchange in the three-party setting. *IEE Proceedings Information Security*, 153(1):27–39, Mar. 2006.

[5] P. Adão, G. Bana, J. Herzog, and A. Scedrov. Soundness of formal encryption in the presence of key-cycles. In S. de Capitani di Vimercati, P. Syverson, and D. Gollmann, editors, *Proceedings of the 10th European Symposium On Research In Computer Security (ESORICS 2005)*, volume 3679 of *Lecture Notes on Computer Science*, pages 374–396, Milan, Italy, Sept. 2005. Springer.

[6] M. Backes and P. Laud. Computationally sound secrecy proofs by mechanized flow analysis. In *Proceedings of 13th ACM Conference on Computer and Communications Security (CCS'06)*, pages 370–379, Alexandria, VA, Nov. 2006. ACM.

[7] M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *17th IEEE Computer Security Foundations Workshop*, pages 204–218, Pacific Grove, CA, June 2004. IEEE.

[8] M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. *IEEE Transactions on Dependable and Secure Computing*, 2(2):109–123, Apr. 2005.

[9] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *10th ACM conference on Computer and communication security (CCS'03)*, pages 220–230, Washington D.C., Oct. 2003. ACM.

[10] M. Backes, B. Pfitzmann, and M. Waidner. Symmetric authentication within a simulatable cryptographic library. In E. Snekkenes and D. Gollman, editors, *Computer Security - ESORICS 2003, 8th European Symposium on Research in Computer Security*, volume 2808 of *Lecture Notes on Computer Science*, pages 271–290, Gjøovik, Norway, Oct. 2003. Springer.

[11] G. Barthe, J. Cederquist, and S. Tarento. A machine-checked formalization of the generic model and the random oracle model. In D. Basin and M. Rusinowitch, editors, *Second International Joint Conference on Automated Reasoning (IJCAR'04)*, volume 3097 of *Lecture Notes on Computer Science*, pages 385–399, Cork, Ireland, July 2004. Springer.

[12] M. Baudet, V. Cortier, and S. Kremer. Computationally sound implementations of equational theories against passive adversaries. In L. Caires and L. Monteiro, editors, *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, volume 3580 of *Lecture Notes on Computer Science*, pages 652–663, Lisboa, Portugal, July 2005. Springer.

[13] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Proceedings of the 38th Symposium on Foundations of Computer Science (FOCS'97)*, pages 394–403, Miami Beach, Florida, Oct. 1997. IEEE. Full paper available at `http://www-cse.ucsd.edu/users/mihir/papers/sym-enc.html`.

[14] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In H. Krawczyk, editor, *Advances in Cryptology – CRYPTO 1998*, volume 1462 of *Lecture Notes on Computer Science*, pages 26–45, Santa Barbara, California, USA, Aug. 1998. Springer.

[15] M. Bellare, J. Kilian, and P. Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences*, 61(3):362–399, Dec. 2000.

[16] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In B. Preneel, editor, *Advances in Cryptology – Proceedings of EUROCRYPT '00*, volume 1807 of *Lecture Notes on Computer Science*, pages 139–155, Bruges, Belgique, 2000. Springer.

[17] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In S. Vaudenay, editor, *Advances in Cryptology – Eurocrypt 2006 Proceedings*, volume 4004 of *Lecture Notes on Computer Science*, pages 409–426, Saint Petersburg, Russia, May 2006. Springer. Extended version available at `http://eprint.iacr.org/2004/331`.

[18] B. Blanchet. Automatic proof of strong secrecy for security protocols. In *IEEE Symposium on Security and Privacy*, pages 86–100, Oakland, California, May 2004.

[19] B. Blanchet. Computationally sound mechanized proofs of correspondence assertions. In *20th IEEE Computer Security Foundations Symposium (CSF'07)*, pages 97–111, Venice, Italy, July 2007. IEEE. Extended version available as ePrint Report 2007/128, `http://eprint.iacr.org/2007/128`.

[20] B. Blanchet and D. Pointcheval. Automated security proofs with sequences of games. In C. Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes on Computer Science*, pages 537–554, Santa Barbara, CA, Aug. 2006. Springer.

[21] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proceedings of the Royal Society of London A*, 426(1871):233–271, dec 1989. A preliminary version appeared as Digital Equipment Corporation Systems Research Center report No. 39, February 1989.

[22] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, Las Vegas, Nevada, Oct. 2001. IEEE. An updated version is available at Cryptology ePrint Archive, `http://eprint.iacr.org/2000/067`.

[23] R. Canetti and J. Herzog. Universally composable symbolic analysis of mutual authentication and key exchange protocols. In S. Halevi and T. Rabin, editors, *Proceedings, Theory of Cryptography Conference (TCC'06)*, volume 3876 of *Lecture Notes on Computer Science*, pages 380–403, New York, NY, Mar. 2006. Springer. Extended version available at `http://eprint.iacr.org/2004/334`.

[24] V. Cortier and B. Warinschi. Computationally sound, automated proofs for security protocols. In M. Sagiv, editor, *Proc. 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes on Computer Science*, pages 157–171, Edimbourg, U.K., Apr. 2005. Springer.

[25] A. Datta, A. Derek, J. C. Mitchell, V. Shmatikov, and M. Turuani. Probabilistic polynomial-time semantics for a protocol security logic. In L. Caires and L. Monteiro, editors, *ICALP 2005: the 32nd International Colloquium on Automata, Languages and Programming*, volume 3580 of *Lecture Notes on Computer Science*, pages 16–29, Lisboa, Portugal, July 2005. Springer.

[26] D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Commun. ACM*, 24(8):533–536, Aug. 1981.

[27] S. Halevi. A plausible approach to computer-aided cryptographic proofs. Cryptology ePrint Archive, Report 2005/181, June 2005. Available at `http://eprint.iacr.org/2005/181`.

[28] J. Herzog. A computational interpretation of Dolev-Yao adversaries. *Theoretical Computer Science*, 340:57–81, June 2005.

[29] R. Janvier, Y. Lakhnech, and L. Mazaré. Completing the picture: Soundness of formal encryption in the presence of active adversaries. In M. Sagiv, editor, *Proc. 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes on Computer Science*, pages 172–185, Edimbourg, U.K., Apr. 2005. Springer.

[30] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, Oxford, U.K., 1970.

[31] P. Laud. Handling encryption in an analysis for secure information flow. In P. Degano, editor, *Programming Languages and Systems, 12th European Symposium on Programming, ESOP'03*, volume 2618 of *Lecture Notes on Computer Science*, pages 159–173, Warsaw, Poland, Apr. 2003. Springer.

[32] P. Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *IEEE Symposium on Security and Privacy*, pages 71–85, Oakland, California, May 2004.

[33] P. Laud. Secrecy types for a simulatable cryptographic library. In *12th ACM Conference on Computer and Communications Security (CCS'05)*, pages 26–35, Alexandria, VA, Nov. 2005. ACM.

[34] P. D. Lincoln, J. C. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *ACM Computer and Communication Security (CCS-5)*, pages 112–121, San Francisco, California, Nov. 1998.

[35] P. D. Lincoln, J. C. Mitchell, M. Mitchell, and A. Scedrov. Probabilistic polynomial-time equivalence and security protocols. In J. Wing, J. Woodcock, and J. Davies, editors, *FM'99 World Congress On Formal Methods in the Development of Computing Systems*, volume 1708 of *Lecture Notes on Computer Science*, pages 776–793, Toulouse, France, Sept. 1999. Springer.

[36] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes on Computer Science*, pages 147–166. Springer, 1996.

[37] P. Mateus, J. Mitchell, and A. Scedrov. Composition of cryptographic protocols in a probabilistic polynomial-time process calculus. In R. Amadio and D. Lugiez, editors, *CONCUR 2003 - Concurrency Theory, 14-th International Conference*, volume 2761 of *Lecture Notes on Computer Science*, pages 327–349, Marseille, France, Sept. 2003. Springer.

[38] D. Micciancio and B. Warinschi. Completeness theorems for the Abadi-Rogaway logic of encrypted expressions. *Journal of Computer Security*, 12(1):99–129, 2004.

[39] D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In M. Naor, editor, *Theory of Cryptography Conference (TCC'04)*, volume 2951 of *Lecture Notes on Computer Science*, pages 133–151, Cambridge, MA, USA, Feb. 2004. Springer.

[40] J. C. Mitchell, A. Ramanathan, A. Scedrov, and V. Teague. A probabilistic polynomial-time calculus for the analysis of cryptographic protocols. *Theoretical Computer Science*, 353(1–3):118–164, Mar. 2006.

[41] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, Dec. 1978.

[42] R. M. Needham and M. D. Schroeder. Authentication revisited. *Operating Systems Review*, 21(1):7, 1987.

[43] D. Otway and O. Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1):8–10, 1987.

[44] A. Ramanathan, J. Mitchell, A. Scedrov, and V. Teague. Probabilistic bisimulation and equivalence for security analysis of network protocols. In I. Walukiewicz, editor, *FOSSACS 2004 - Foundations of Software Science and Computation Structures*, volume 2987 of *Lecture Notes on Computer Science*, pages 468–483, Barcelona, Spain, Mar. 2004. Springer.

[45] V. Shoup. A proposal for an ISO standard for public-key encryption, Dec. 2001. ISO/IEC JTC 1/SC27.

[46] V. Shoup. OAEP reconsidered. *Journal of Cryptology*, 15(4):223–249, Sept. 2002.

[47] C. Sprenger, M. Backes, D. Basin, B. Pfitzmann, and M. Waidner. Cryptographically sound theorem proving. In *19th IEEE Computer Security Foundations Workshop (CSFW-19)*, pages 153–166, Venice, Italy, July 2006. IEEE.

[48] S. Tarento. Machine-checked security proofs of cryptographic signature schemes. In S. de Capitani di Vimercati, P. Syverson, and D. Gollmann, editors, *Proceedings of the 10th European Symposium On Research In Computer Security (ESORICS 2005)*, volume 3679 of *Lecture Notes on Computer Science*, pages 140–158, Milan, Italy, Sept. 2005. Springer.

[49] T. Y. C. Woo and S. S. Lam. A semantic model for authentication protocols. In *Proceedings IEEE Symposium on Research in Security and Privacy*, pages 178–194, Oakland, California, May 1993.

Appendices

A Type System

In this section, we define the type system, used in our calculus to check that bitstrings belong to the expected type.

To be able to type variable accesses used not under their definition (such accesses are guarded by a find construct), the type-checking algorithm proceeds in two passes. In the first pass, we build a type environment $\mathcal{E}$, which maps variable names x to types $T_1 \times \ldots \times T_m \to T$, where $T_1, \ldots, T_m$ are the interval types of the indices of x, and T is the type of $x[i_1, \ldots, i_m]$. This type environment is built as follows:

- If x is defined by $\mathsf{new}\ x[i_1, \ldots, i_m] : T$, $\mathsf{let}\ x[i_1, \ldots, i_m] : T = M$, or $c[M_1, \ldots, M_l](\ldots, x[i_1, \ldots, i_m] : T, \ldots)$, and the replications above this subprocess are $!^{i_1 \leq n_1}$, $\ldots$, $!^{i_m \leq n_m}$, then $\mathcal{E}(x) = [1, n_1] \times \ldots \times [1, n_m] \to T$.
- If u is defined by $\mathsf{find} \ldots \oplus \ldots u[i_1, \ldots, i_m] \leq n \ldots\ \mathsf{suchthat}\ \mathsf{defined}(\ldots) \wedge \ldots \mathsf{then} \ldots \oplus \ldots$ and the replications above this find are $!^{i_1 \leq n_1}$, $\ldots$, $!^{i_m \leq n_m}$, then $\mathcal{E}(u) = [1, n_1] \times \ldots \times [1, n_m] \to [1, n]$.

We require that all definitions of the same variable x yield the same value of $\mathcal{E}(x)$, so that $\mathcal{E}$ is properly defined.

A process can then be typechecked in the type environment $\mathcal{E}$ using the rules of Figure 3. This figure defines three judgments:

$$\frac{\mathcal{E}(i) = T}{\mathcal{E} \vdash i : T} \quad \text{(TIndex)}$$

$$\frac{\mathcal{E}(x) = T_1 \times \ldots \times T_m \to T \qquad \forall j \leq m, \mathcal{E} \vdash M_j : T_j}{\mathcal{E} \vdash x[M_1, \ldots, M_m] : T} \quad \text{(TVar)}$$

$$\frac{f : T_1 \times \ldots \times T_m \to T \qquad \forall j \leq m, \mathcal{E} \vdash M_j : T_j}{\mathcal{E} \vdash f(M_1, \ldots, M_m) : T} \quad \text{(TFun)}$$

$$\mathcal{E} \vdash 0 \quad \text{(TNil)}$$

$$\frac{\mathcal{E} \vdash Q \qquad \mathcal{E} \vdash Q'}{\mathcal{E} \vdash Q \mid Q'} \quad \text{(TPar)}$$

$$\frac{\mathcal{E}[i \mapsto [1, n]] \vdash Q}{\mathcal{E} \vdash !^{i \leq n} Q} \quad \text{(TRepl)}$$

$$\frac{\mathcal{E} \vdash Q}{\mathcal{E} \vdash \mathsf{newChannel}\ c; Q} \quad \text{(TNewChannel)}$$

$$\frac{\forall j \leq l, \mathcal{E} \vdash M_j : T'_j \qquad \forall j \leq k, \mathcal{E} \vdash x_j[\widetilde{i}] : T_j \qquad \mathcal{E} \vdash P}{\mathcal{E} \vdash c[M_1, \ldots, M_l](x_1[\widetilde{i}] : T_1, \ldots, x_k[\widetilde{i}] : T_k); P} \quad \text{(TIn)}$$

$$\frac{\forall j \leq l, \mathcal{E} \vdash M_j : T'_j \qquad \forall j \leq k, \mathcal{E} \vdash N_j : T_j \qquad \mathcal{E} \vdash Q}{\mathcal{E} \vdash \overline{c[M_1, \ldots, M_l]} \langle N_1, \ldots, N_k \rangle; Q} \quad \text{(TOut)}$$

$$\frac{T \text{ fixed-length type} \qquad \mathcal{E} \vdash x[\widetilde{i}] : T \qquad \mathcal{E} \vdash P}{\mathcal{E} \vdash \mathsf{new}\ x[\widetilde{i}] : T; P} \quad \text{(TNew)}$$

$$\frac{\mathcal{E} \vdash M : T \qquad \mathcal{E} \vdash x[\widetilde{i}] : T \qquad \mathcal{E} \vdash P}{\mathcal{E} \vdash \mathsf{let}\ x[\widetilde{i}] : T = M \mathsf{\ in\ } P} \quad \text{(TLet)}$$

$$\frac{\begin{array}{c}\forall j \leq m, \forall k \leq m_j, \mathcal{E} \vdash u_{jk}[\widetilde{i}] : [1, n_{jk}] \\ \forall j \leq m, \forall k \leq l_j, \mathcal{E} \vdash M_{jk} : T_{jk} \\ \forall j \leq m, \mathcal{E} \vdash M_j : \mathit{bool} \qquad \forall j \leq m, \mathcal{E} \vdash P_j \qquad \mathcal{E} \vdash P\end{array}}{\begin{array}{c}\mathcal{E} \vdash \mathsf{find}\ (\bigoplus_{j=1}^m u_{j1}[\widetilde{i}] \leq n_{j1}, \ldots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j} \mathsf{\ suchthat} \\ \mathsf{defined}(M_{j1}, \ldots, M_{jl_j}) \wedge M_j \mathsf{\ then\ } P_j) \mathsf{\ else\ } P\end{array}} \quad \text{(TFind)}$$

Figure 3: Typing rules

- $\mathcal{E} \vdash M : T$ means that term M has type T in environment $\mathcal{E}$.
- $\mathcal{E} \vdash P$ and $\mathcal{E} \vdash Q$ mean that the output process P and the input process Q are well-typed in environment $\mathcal{E}$, respectively.

In $x[M_1, \ldots, M_m]$, $M_1, \ldots, M_m$ must be of the suitable interval type. When $f(M_1, \ldots, M_m)$ is called and $f : T_1 \times \ldots \times T_m \to T$, M_j must be of type T_j, and $f(M_1, \ldots, M_m)$ is then of type T. The type system requires each subterm to be well-typed. Furthermore, in let $x : T = M$ in P, M must be of type T. In

$$\mathsf{find}\ (\bigoplus_{j=1}^m u_{j1}[\widetilde{i}] \leq n_{j1}, \ldots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j} \mathsf{\ suchthat}\ \mathsf{defined}(M_{j1}, \ldots, M_{jl_j}) \wedge M_j \mathsf{\ then\ } P_j) \mathsf{\ else\ } P$$

M_j is of type *bool* for all $j \leq m$. In $!^{i \leq n} Q$, i is of type $[1, n]$ in Q. For new $x[\widetilde{i}] : T$, T must be a fixed-length type.

We say that an occurrence of a term M in a process Q is of type T when $\mathcal{E} \vdash M : T$ where $\mathcal{E}$ is the type environment of Q extended with $i \mapsto [1, n]$ for each replication $!^{i \leq n}$ above M in Q.

B Formal Semantics

B.1 Definition of the Semantics

The formal semantics of our calculus is presented in Figures 4 and 5. In this figure and in the rest of the appendix, we use $\uplus$ for multiset union. When S is a multiset, $S(x)$ is the number of elements of S equal to x. A semantic configuration is a quadruple $E, P, \mathcal{Q}, \mathcal{C}$, where E is an environment mapping array cells to bitstrings or $\bot$, P is the output process currently scheduled, $\mathcal{Q}$ is the multiset of input processes running in parallel with P, $\mathcal{C}$ is the set of channels already created. The semantics is defined by reduction rules of the form $E, P, \mathcal{Q}, \mathcal{C} \xrightarrow{p}_{\eta, t} E', P', \mathcal{Q}', \mathcal{C}'$ meaning that $E, P, \mathcal{Q}, \mathcal{C}$ reduces to $E', P', \mathcal{Q}', \mathcal{C}'$ with probability p, when the security parameter is η. The value of the security parameter is often omitted to lighten the notation. The index t just serves in distinguishing reductions that yield the same configuration with the same probability in different ways, so that the probability of a certain reduction can be computed correctly:

$$\Pr[E, P, \mathcal{Q}, \mathcal{C} \to_\eta E', P', \mathcal{Q}', \mathcal{C}'] = \sum_{E, P, \mathcal{Q}, \mathcal{C} \xrightarrow{p}_{\eta, t} E', P', \mathcal{Q}', \mathcal{C}'} p$$

The probability of a trace is computed as follows:

$$\Pr[E_1, P_1, \mathcal{Q}_1, \mathcal{C}_1 \to_\eta \ldots \to_\eta E'_m, P'_m, \mathcal{Q}'_m, \mathcal{C}'_m] = \prod_{j=1}^{m-1} \Pr[E_j, P_j, \mathcal{Q}_j, \mathcal{C}_j \to_\eta E'_{j+1}, P'_{j+1}, \mathcal{Q}'_{j+1}, \mathcal{C}'_{j+1}]$$

We define an auxiliary relation for evaluating terms: $E, M \Downarrow_\eta a$, or simply $E, M \Downarrow a$, means that the term M evaluates to the bitstring a in environment E. Rule (Cst) simply evaluates constants to themselves. This rule serves for replication indices, which are substituted with constant values when reducing the

Terms and find conditions:

$$E, a \Downarrow a \tag{Cst}$$

$$\frac{\forall j \leq m, E, M_j \Downarrow a_j \qquad x[a_1, \ldots, a_m] \in \mathrm{Dom}(E)}{E, x[M_1, \ldots, M_m] \Downarrow E(x[a_1, \ldots, a_m])} \tag{Var}$$

$$\frac{\begin{array}{c}\forall j \leq m, E, M_j \Downarrow a_j \qquad f : T_1 \times \ldots \times T_m \to T \\ \forall j \leq m, a_j \in I_\eta(T_j)\end{array}}{E, f(M_1, \ldots, M_m) \Downarrow I_\eta(f)(a_1, \ldots, a_m)} \tag{Fun}$$

$$\frac{\neg \forall k \leq l, \exists a_k, E, M_k \Downarrow a_k}{E, (\mathsf{defined}(M_1, \ldots, M_l) \wedge M) \Downarrow \mathrm{false}} \tag{Def1}$$

$$\frac{\forall k \leq l, \exists a_k, E, M_k \Downarrow a_k \qquad E, M \Downarrow a \qquad a \in \{\mathrm{false}, \mathrm{true}\}}{E, (\mathsf{defined}(M_1, \ldots, M_l) \wedge M) \Downarrow a} \tag{Def2}$$

Input processes:

$$E, \{0\} \uplus \mathcal{Q}, \mathcal{C} \rightsquigarrow E, \mathcal{Q}, \mathcal{C} \tag{Nil}$$

$$E, \{Q_1 \mid Q_2\} \uplus \mathcal{Q}, \mathcal{C} \rightsquigarrow E, \{Q_1, Q_2\} \uplus \mathcal{Q}, \mathcal{C} \tag{Par}$$

$$E, \{!^{i \leq n} Q\} \uplus \mathcal{Q}, \mathcal{C} \rightsquigarrow E, \{Q\{a/i\} \mid a \in [1, I_\eta(n)]\} \uplus \mathcal{Q}, \mathcal{C} \tag{Repl}$$

$$\frac{c' \notin \mathcal{C}}{\begin{array}{c}E, \{\mathsf{newChannel}\ c; Q\} \uplus \mathcal{Q}, \mathcal{C} \\ \rightsquigarrow E, \{Q\{c'/c\}\} \uplus \mathcal{Q}, \mathcal{C} \cup \{c'\}\end{array}} \tag{NewChannel}$$

$$\frac{\forall j \leq l, E, M_j \Downarrow a_j}{\begin{array}{c}E, \{c[M_1, \ldots, M_l](x_1[\widetilde{a'}] : T_1, \ldots, x_k[\widetilde{a'}] : T_k); P\} \uplus \mathcal{Q}, \mathcal{C} \\ \rightsquigarrow E, \{c[a_1, \ldots, a_l](x_1[\widetilde{a'}] : T_1, \ldots, x_k[\widetilde{a'}] : T_k); P\} \uplus \mathcal{Q}, \mathcal{C}\end{array}} \tag{Input}$$

$\mathrm{reduce}(E, \mathcal{Q}, \mathcal{C})$ is the normal form of $E, \mathcal{Q}, \mathcal{C}$ by $\rightsquigarrow$

Figure 4: Semantics (1)

Output processes:

$$\frac{T \text{ fixed-length type} \qquad a \in I_\eta(T)}{E, \mathsf{new}\ x[\widetilde{a'}] : T; P, \mathcal{Q}, \mathcal{C} \xrightarrow{\frac{1}{|I_\eta(T)|}}_{N(a)} E[x[\widetilde{a'}] \mapsto a], P, \mathcal{Q}, \mathcal{C}} \tag{New}$$

$$\frac{E, M \Downarrow a \qquad a \in I_\eta(T)}{E, \mathsf{let}\ x[\widetilde{a'}] : T = M\ \mathsf{in}\ P, \mathcal{Q}, \mathcal{C} \xrightarrow{1}_L E[x[\widetilde{a'}] \mapsto a], P, \mathcal{Q}, \mathcal{C}} \tag{Let}$$

$$\frac{\begin{array}{c}\forall j \leq m, \forall \widetilde{v} \leq \widetilde{n_j}, E[\widetilde{u_j}[\widetilde{a'}] \mapsto \widetilde{v}], (D_j \wedge M_j) \Downarrow a_{j,\widetilde{v}} \\ S = \{j, \widetilde{v} \mid a_{j,\widetilde{v}} = \mathrm{true}\} \qquad a_{j_0,\widetilde{v_0}} = \mathrm{true} \\ E_{j_0,\widetilde{v_0}} = E[\widetilde{u_{j_0}}[\widetilde{a'}] \mapsto \widetilde{v_0}]\end{array}}{\begin{array}{c}E, \mathsf{find}\ (\bigoplus_{j=1}^m \widetilde{u_j}[\widetilde{a'}] \leq \widetilde{n_j}\ \mathsf{suchthat}\ D_j \wedge M_j\ \mathsf{then}\ P_j) \\ \mathsf{else}\ P, \mathcal{Q}, \mathcal{C} \xrightarrow{\mathrm{among}(S)}_{F1(j_0,\widetilde{v_0})} E_{j_0,\widetilde{v_0}}, P_{j_0}, \mathcal{Q}, \mathcal{C}\end{array}} \tag{Find1}$$

$$\frac{\forall j \leq m, \forall \widetilde{v} \leq \widetilde{n_j}, E[\widetilde{u_j}[\widetilde{a'}] \mapsto \widetilde{v}], (D_j \wedge M_j) \Downarrow \mathrm{false}}{\begin{array}{c}E, \mathsf{find}\ (\bigoplus_{j=1}^m \widetilde{u_j}[\widetilde{a'}] \leq \widetilde{n_j}\ \mathsf{suchthat}\ D_j \wedge M_j\ \mathsf{then}\ P_j) \\ \mathsf{else}\ P, \mathcal{Q}, \mathcal{C} \xrightarrow{1}_{F2} E, P, \mathcal{Q}, \mathcal{C}\end{array}} \tag{Find2}$$

$$\frac{\begin{array}{c}\forall j \leq l, E, M_j \Downarrow a_j \qquad \forall j \leq k, E, N_j \Downarrow b_j \\ E, \mathcal{Q}', \mathcal{C}' = \mathrm{reduce}(E, \{Q''\}, \mathcal{C}) \\ S = \{Q \in \mathcal{Q} \mid \text{for some } x'_1, \ldots, x'_k, \widetilde{a''}, T'_1, \ldots, T'_k, P', \\ Q = c[a_1, \ldots, a_l](x'_1[\widetilde{a''}] : T'_1, \ldots, x'_k[\widetilde{a''}] : T'_k).P'\} \\ Q_0 = c[a_1, \ldots, a_l](x_1[\widetilde{a'}] : T_1, \ldots, x_k[\widetilde{a'}] : T_k).P \in S \\ \forall j \leq k, b'_j = b_j \& (2^{\mathrm{maxlen}_\eta(c)} - 1) \in I_\eta(T_j)\end{array}}{\begin{array}{l}E, \overline{c[M_1, \ldots, M_l]}\langle N_1, \ldots, N_k\rangle.Q'', \mathcal{Q}, \mathcal{C} \\ \xrightarrow{S(Q_0) \times \mathrm{among}(S)}_{O(Q_0)} \\ \quad E[x_1[\widetilde{a'}] \mapsto b'_1, \ldots, x_k[\widetilde{a'}] \mapsto b'_k], P, \mathcal{Q} \uplus \mathcal{Q}' \setminus \{Q_0\}, \mathcal{C}'\end{array}} \tag{Output}$$

Figure 5: Semantics (2)

replication. Rule (Var) looks for the value of the array variable in the environment. Rule (Fun) evaluates the function call. Rules (Def1) and (Def2) evaluate conditions of find: When some M_k is not defined, $\mathsf{defined}(M_1, \ldots, M_l) \wedge M$ returns false by (Def1). Otherwise, it returns the boolean value of M by (Def2).

We use an auxiliary reduction relation $\rightsquigarrow_\eta$, or simply $\rightsquigarrow$, for reducing input processes. This relation transforms configurations of the form $E, \mathcal{Q}, \mathcal{C}$. Rule (Nil) removes nil processes. Rules (Par) and (Repl) expand parallel compositions and replications, respectively. Rule (NewChannel) creates a new channel and adds it to $\mathcal{C}$. Semantic configurations are considered equivalent modulo renaming of channels in $\mathcal{C}$, so that a single semantic configuration is obtained after applying (NewChannel). Rule (Input) evaluates the terms in the input channel. The input itself is not executed: the communication is done by the (Output) rule. The relation $\rightsquigarrow$ is convergent (confluent and terminating), so it has normal forms. Since processes in $\mathcal{Q}$ in configurations $E, P, \mathcal{Q}, \mathcal{C}$ are in normal form by $\rightsquigarrow$, they always start with an input.

Rules (New) to (Find2) simply reduce the scheduled process. As explained in the footnote page 3, we use an approximately uniform probability distribution for choosing an element among a set S when $m = |S|$ is not a power of 2. Let k be the smallest integer such that $2^k \geq m$. We choose a random integer r uniformly among $[0, 2^{k+f(\eta)}-1]$ for a certain function f. When r is in $[0, (2^{k+f(\eta)} \operatorname{div} m \times m)-1]$, $r \mod m$ returns a random integer in $[0, m-1]$, with the same probability for all elements of $[0, m-1]$. When r is in $[2^{k+f(\eta)} \operatorname{div} m \times m, 2^{k+f(\eta)}-1]$, we can do anything; we choose to block. The probability of being in this case is $(2^{k+f(\eta)} \mod m)/2^{k+f(\eta)} \leq m/2^{k+f(\eta)} \leq 1/2^{f(\eta)}$, so it can be made as small as we wish by choosing $f(\eta)$ large enough. We choose $f(\eta) \geq \alpha\eta$ for some $\alpha > 0$, so that it is negligible. The probability of choosing each element of S is then $\mathrm{among}(S) = \frac{2^{k+f(\eta)} \operatorname{div} m}{2^{k+f(\eta)}}$. Then $\mathrm{among}(S)$ approximates $1/m$. Rules (Find1) and (Find2) evaluate a find. They compute the value of all conditions $D_j \wedge M_j$ of this find for all possible values $\widetilde{v}$ of the indices $\widetilde{u_j}[\widetilde{a'}]$. When all these conditions are false, rule (Find2) executes the else branch of the find. When at least one of these conditions is true, rule (Find1) chooses one such true case (for $j = j_0$ and $\widetilde{v} = \widetilde{v_0}$) with approximately uniform probability, and executes the corresponding then branch of the find.

Rule (Output) performs communications: it evaluates the terms in the channel and the sent messages, selects an input on the desired channel randomly, and immediately executes the communication. The scheduled process after this rule is the receiving process. (The process blocks if no suitable input is available.)

The initial configuration for running process Q_0 is $\mathrm{initConfig}(Q_0) = \emptyset, \overline{start}\langle\rangle, \mathcal{Q}, \mathcal{C}$ where $\emptyset, \mathcal{Q}, \mathcal{C} = \mathrm{reduce}(\emptyset, \{Q_0\}, \mathrm{fc}(Q_0))$.

Definition 6 Let c be a channel name and a be a bitstring. We say that $E, P, \mathcal{Q}, \mathcal{C}$ *executes* $\overline{c}\langle a\rangle$ *immediately* when $P = \overline{c}\langle M\rangle.Q$ and $E, M \Downarrow a$ for some Q and M.

The probability that Q executes $\overline{c}\langle a\rangle$ is denoted $\Pr[Q \rightsquigarrow_\eta \overline{c}\langle a\rangle]$. When $c \in \mathrm{fc}(Q)$, $\Pr[Q \rightsquigarrow_\eta \overline{c}\langle a\rangle] = \sum_{\mathcal{T}\in\mathbb{T}} \Pr[\mathcal{T}]$ where $\mathbb{T}$ is the set of traces $\mathrm{initConfig}(Q) \rightarrow_\eta \ldots \rightarrow_\eta E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m$ such that $E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m$ executes $\overline{c}\langle a\rangle$ immediately and for all $j < m$, $E_j, P_j, \mathcal{Q}_j, \mathcal{C}_j$ does not execute $\overline{c}\langle a\rangle$ immediately. When $c \notin \mathrm{fc}(Q)$, $\Pr[Q \rightsquigarrow_\eta \overline{c}\langle a\rangle] = 0$.

B.2 Each Variable is Defined at Most Once

In this section, we show that Invariant 1 implies that each array cell is assigned at most once during the execution of a process.

When S and S' are multisets, $\max(S, S')$ is the multiset such that $\max(S, S')(x) = \max(S(x), S'(x))$. We define the multiset of variable accesses that may be defined by a process as follows:

$$\mathit{Defined}(0) = \emptyset$$
$$\mathit{Defined}(Q_1 \mid Q_2) = \mathit{Defined}(Q_1) \uplus \mathit{Defined}(Q_2)$$
$$\mathit{Defined}(!^{i\leq n}Q) = \biguplus_{a\in[1,I_\eta(n)]} \mathit{Defined}(Q\{a/i\})$$
$$\mathit{Defined}(\mathsf{newChannel}\ c; Q) = \mathit{Defined}(Q)$$
$$\mathit{Defined}(c[M_1, \ldots, M_l](x_1[\widetilde{a}] : T_1, \ldots, x_k[\widetilde{a}] : T_k); P) = \{x_j[\widetilde{a}] \mid j \leq k\} \uplus \mathit{Defined}(P)$$
$$\mathit{Defined}(\overline{c[M_1, \ldots, M_l]}\langle N_1, \ldots, N_k\rangle; Q) = \mathit{Defined}(Q)$$
$$\mathit{Defined}(\mathsf{new}\ x[\widetilde{a}] : T; P) = \{x[\widetilde{a}]\} \uplus \mathit{Defined}(P)$$
$$\mathit{Defined}(\mathsf{let}\ x[\widetilde{a}] : T = M\ \mathsf{in}\ P) = \{x[\widetilde{a}]\} \uplus \mathit{Defined}(P)$$
$$\mathit{Defined}(\mathsf{find}\ (\textstyle\bigoplus_{j=1}^m \widetilde{u_j}[\widetilde{a}] \leq \widetilde{n_j}\ \mathsf{suchthat defined}(M_{j1}, \ldots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P) = \max(\max_{j=1}^{m}\{\widetilde{u_j}[\widetilde{a}]\} \uplus \mathit{Defined}(P_j), \mathit{Defined}(P))$$

We define $\mathit{Defined}(E) = \mathrm{Dom}(E)$, $\mathit{Defined}(E, P, \mathcal{Q}, \mathcal{C}) = \mathit{Defined}(E) \uplus \mathit{Defined}(P) \uplus \biguplus_{Q\in\mathcal{Q}} \mathit{Defined}(Q)$.

Invariant 4 (Single definition, for executing games) The semantic configuration $E, P, \mathcal{Q}, \mathcal{C}$ satisfies Invariant 4 if and only if $\mathit{Defined}(E, P, \mathcal{Q}, \mathcal{C})$ does not contain duplicate elements.

Lemma 3 *If Q_0 satisfies Invariant 1, then* $\mathrm{initConfig}(Q_0)$ *satisfies Invariant 4.*

Lemma 4 *If $E, P, \mathcal{Q}, \mathcal{C} \xrightarrow{p}_t E', P', \mathcal{Q}', \mathcal{C}'$ with $p > 0$ and $E, P, \mathcal{Q}, \mathcal{C}$ satisfies Invariant 4, then so does $E', P', \mathcal{Q}', \mathcal{C}'$.*

Proof sketch We show by cases following the definition of $\xrightarrow{p}_t$ that if $E, P, \mathcal{Q}, \mathcal{C} \xrightarrow{p}_t E', P', \mathcal{Q}', \mathcal{C}'$ then $\mathit{Defined}(E, P, \mathcal{Q}, \mathcal{C}) \subseteq \mathit{Defined}(E', P', \mathcal{Q}', \mathcal{C}')$. The result follows. $\square$

Therefore, if Q_0 satisfies Invariant 1, then each variable is defined at most once for each value of its array indices in a trace of Q_0. Indeed, by Invariant 4, just before executing a definition of $x[\widetilde{a}]$, $\mathit{Defined}(E, P, \mathcal{Q}, \mathcal{C})$ does not contain duplicate elements, so $x[\widetilde{a}] \notin \mathrm{Dom}(E)$ since $x[\widetilde{a}] \in \mathit{Defined}(P) \uplus \mathit{Defined}(\mathcal{Q})$.

B.3 Variables are Defined Before Being Used

In this section, we show that Invariant 2 implies that all variables are defined before being used. In order to show this property, we use the following invariant:

Invariant 5 (Defined variables, for executing games) The semantic configuration $E, P, \mathcal{Q}, \mathcal{C}$ satisfies Invariant 5 if and only if every occurrence of a variable access $x[M_1, \ldots, M_m]$ in P or $\mathcal{Q}$ is either

- present in $\mathrm{Dom}(E)$: for all $j \leq m$, $E, M_j \Downarrow a_j$ and $x[a_1, \ldots, a_m] \in \mathrm{Dom}(E)$;
- or syntactically under the definition of $x[M_1, \ldots, M_m]$ (in which case for all $j \leq m$, M_j is a constant or variable replication index);
- or in a defined condition in a find process;
- or in M'_j or P_j in a process of the form $\mathsf{find}\ (\bigoplus_{j=1}^{m''} \widetilde{u_j}[\widetilde{i}] \leq \widetilde{n_j}\ \mathsf{suchthat}\ \mathsf{defined}(M'_{j1}, \ldots, M'_{jl_j}) \wedge M'_j\ \mathsf{then}\ P_j\ \mathsf{else}\ P$ where for some $k \leq l_j$, $x[M_1, \ldots, M_m]$ is a subterm of M'_{jk}.

Lemma 5 *If Q_0 satisfies Invariant 2, then* $\mathrm{initConfig}(Q_0)$ *satisfies Invariant 5.*

Lemma 6 *If $E, P, \mathcal{Q}, \mathcal{C} \xrightarrow{p}_t E', P', \mathcal{Q}', \mathcal{C}'$ with $p > 0$ and $E, P, \mathcal{Q}, \mathcal{C}$ satisfies Invariant 5, then so does $E', P', \mathcal{Q}', \mathcal{C}'$.*

Proof sketch If $x[M_1, \dots, M_m]$ is in the second case of Invariant 5, and we execute the definition of $x[M_1, \dots, M_m]$, then for all $j \leq m$, M_j is a constant replication index and $x[M_1, \dots, M_m]$ is added to $\mathrm{Dom}(E)$ by rules (New), (Let), (Find1), or (Output), so it moves to the first case of Invariant 5.

If $x[M_1, \dots, M_m]$ is in the third case of Invariant 5, and we execute the corresponding find, this access to x simply disappears.

If $x[M_1, \dots, M_m]$ is in the last case of Invariant 5, and we execute the find selecting branch j, then $x[M_1, \dots, M_m]$ is a subterm of M'_{jk} for $k \leq l_j$. We show by induction on M that, if $E, M \Downarrow a$, then for all subterms $x[M_1, \dots, M_m]$ of M, for all $j' \leq m$, $E, M_{j'} \Downarrow a_{j'}$ and $x[a_1, \dots, a_m]$ is in $\mathrm{Dom}(E)$. Therefore, by hypothesis of the semantic rule for find, for all $j' \leq m$, $E, M_{j'} \Downarrow a_{j'}$ and $x[a_1, \dots, a_m]$ is in $\mathrm{Dom}(E)$. So $x[M_1, \dots, M_m]$ also moves to the first case of Invariant 5.

In all other cases, the situation remains unchanged. □

Therefore, if Q_0 satisfies Invariant 2, then in traces of Q_0, the test $x[a_1, \dots, a_m] \in \mathrm{Dom}(E)$ in rule (Var) always succeeds, except when the considered term occurs in a defined condition of a find.

Indeed, consider an application of rule (Var), where the array access $x[M_1, \dots, M_m]$ is not in a defined condition of a find. Then, this array access is not under any variable definition or find, so for all $j \leq m$, $E, M_j \Downarrow a_j$ and $x[a_1, \dots, a_m] \in \mathrm{Dom}(E)$. Hence, the test $x[a_1, \dots, a_m] \in \mathrm{Dom}(E)$ succeeds.

B.4 Typing

In this section, we show that our type system is compatible with the semantics of the calculus, that is, we define a notion of typing for semantic configurations and show that typing is preserved by reduction (subject reduction). Finally, the property that semantic configurations are well-typed shows that certain conditions in the semantics always hold.

We say that $\mathcal{E} \vdash_\eta E$ if and only if $E(x[a_1, \dots, a_m]) = a$ implies $\mathcal{E}(x) = T_1 \times \ldots \times T_m \to T$ with for all $j \leq m$, $a_j \in I_\eta(T_j)$ and $a \in I_\eta(T)$. We define $\mathcal{E} \vdash_\eta P$ as $\mathcal{E} \vdash P$, $\mathcal{E} \vdash_\eta Q$ as $\mathcal{E} \vdash Q$, and $\mathcal{E} \vdash_\eta M : T$ as $\mathcal{E} \vdash M : T$, with the additional rule $\mathcal{E} \vdash_\eta a : T$ if and only if $a \in I_\eta(T)$. (This rule is useful to type constant replication indices. In the formulas giving the typing rules, replication indices i may then also be constants a.) We say that $\mathcal{E} \vdash_\eta E, P, \mathcal{Q}, \mathcal{C}$ if and only if $\mathcal{E} \vdash_\eta E$, $\mathcal{E} \vdash_\eta P$, and for all $Q \in \mathcal{Q}$, $\mathcal{E} \vdash_\eta Q$. Similarly, $\mathcal{E} \vdash_\eta E, \mathcal{Q}, \mathcal{C}$ if and only if $\mathcal{E} \vdash_\eta E$ and for all $Q \in \mathcal{Q}$, $\mathcal{E} \vdash_\eta Q$.

Lemma 7 *If $\mathcal{E} \vdash_\eta E$, $\mathcal{E} \vdash_\eta M : T$, and $E, M \Downarrow a$, then $\mathcal{E} \vdash_\eta a : T$*

Proof sketch By induction on the derivation of $E, M \Downarrow a$. □

Lemma 8 *If $\mathcal{E} \vdash_\eta E, \mathcal{Q}, \mathcal{C}$ and $E, \mathcal{Q}, \mathcal{C} \rightsquigarrow E', \mathcal{Q}', \mathcal{C}'$, then $\mathcal{E} \vdash_\eta E', \mathcal{Q}', \mathcal{C}'$.*

So, if $\mathcal{E} \vdash_\eta E, \mathcal{Q}, \mathcal{C}$, then $\mathcal{E} \vdash_\eta \mathrm{reduce}(E, \mathcal{Q}, \mathcal{C})$.

Proof sketch By cases on the derivation of $E, \mathcal{Q}, \mathcal{C} \rightsquigarrow E', \mathcal{Q}', \mathcal{C}'$. In the case of the replication, we use a substitution lemma, noticing that $a \in I_\eta([1, n])$, so $\mathcal{E} \vdash_\eta a : [1, n]$. In the case of the input, we use Lemma 7. □

Lemma 9 *If $\mathcal{E} \vdash Q_0$, then $\mathcal{E} \vdash_\eta \mathrm{initConfig}(Q_0)$.*

Proof sketch By Lemma 8 and the previous definitions. □

Lemma 10 (Subject reduction) *If $\mathcal{E} \vdash_\eta E, P, \mathcal{Q}, \mathcal{C}$ and $E, P, \mathcal{Q}, \mathcal{C} \xrightarrow{p}_t E', P', \mathcal{Q}', \mathcal{C}'$ with $p > 0$, then $\mathcal{E} \vdash_\eta E', P', \mathcal{Q}', \mathcal{C}'$.*

Proof sketch By cases on the derivation of $E, P, \mathcal{Q}, \mathcal{C} \xrightarrow{p}_t E', P', \mathcal{Q}', \mathcal{C}'$, using Lemmas 7 and 8. □

As an immediate consequence of Lemmas 9, 10, and 7, we obtain: if Q_0 satisfies Invariant 3, then in traces of Q_0, the tests *T fixed-length type* in rule (New), $a \in I_\eta(T)$ in rule (Let), $\forall j \leq m, a_j \in I_\eta(T_j)$ in rule (Fun), and the test $a \in \{\mathrm{false}, \mathrm{true}\}$ in rule (Def2) always succeed.

B.5 Runtime

Proposition 6 *For each process Q, there exists a probabilistic polynomial time Turing machine that simulates Q.*

Proof We give a very brief sketch of this proof here. We refer the reader to [40] for a more detailed proof for a different calculus; their proof could be adapted to our calculus.

The length of all bitstrings manipulated by processes is polynomial in the security parameter η. Indeed, by hypothesis, the length of received messages is limited by maxlen_η, so polynomial in the security parameter η. The length of random bitstrings is also polynomial in the security parameter by hypothesis on the types. Function symbols correspond to functions that run in polynomial time, so they output bitstrings of size polynomial in the size of their inputs, so also polynomial in the security parameter.

Since the number of copies generated by each replication is polynomial in the security parameter, the total number of executed instructions is polynomial in the security parameter. Moreover, it is easy to see that each instruction runs in polynomial time since bitstrings are of polynomial length. Therefore, processes run in polynomial time. □

C Simplification

In this section, we define the transformation **Simplify**, which is used to simplify games. The simplification proceeds as follows. It uses information from several sources: equations and rewrite rules given by user, that come in particular from algebraic properties of cryptographic primitives; facts that hold at certain points in the game due to the form of the game; dependency information obtained by two dependency analyses. (The global dependency analysis tracks which variables depend on any element of the array x at any program point. The local dependency analysis tracks which terms depend on the current cell of the array x, $x[\widetilde{i}]$, at each program point.) The simplification algorithm uses this information in order to infer equalities using a Knuth-Bendix-like equational prover. The obtained equalities are used to simplify the game, by replacing a term with an equal term or by simplifying find when the system proves that some branches cannot be taken.

C.1 User-defined Rewrite Rules

The user can give two kinds of information:

- claims of the form $\forall x_1 : T_1, \dots, \forall x_m : T_m, M$ which mean that for all environments E, if for all $j \leq m$, $E(x_j) \in I_\eta(T_j)$, then $E, M \Downarrow \text{true}$.

  Such claims must be well-typed, that is, $\{x_1 \mapsto T_1, \dots, x_m \mapsto T_m\} \vdash M : \mathit{bool}$.

  They are translated into rewrite rules as follows:

  - If M is of the form $M_1 = M_2$ and $\text{var}(M_2) \subseteq \text{var}(M_1)$, we generate the rewrite rule $\forall x_1 : T_1, \dots, \forall x_m : T_m, M_1 \rightarrow M_2$.
  - If M is of the form $M_1 \neq M_2$, we generate the rewrite rules $\forall x_1 : T_1, \dots, \forall x_m : T_m, (M_1 = M_2) \rightarrow \text{false}$, $\forall x_1 : T_1, \dots, \forall x_m : T_m, (M_1 \neq M_2) \rightarrow \text{true}$. (Such rules are used for instance to express that different constants are different.)
  - Otherwise, we generate the rewrite rule $\forall x_1 : T_1, \dots, \forall x_m : T_m, M \rightarrow \text{true}$.

  The term M reduces into M' by the rewrite rule $\forall x_1 : T_1, \dots, \forall x_m : T_m, M_1 \rightarrow M_2$ if and only if $M = C[\sigma M_1]$, $M' = C[\sigma M_2]$, where C is a term context and σ is a substitution that maps x_j to any term of type T_j for all $j \leq m$.

- claims of the form $\mathsf{new}\ y_1 : T'_1, \dots, \mathsf{new}\ y_l : T'_l, \forall x_1 : T_1, \dots, \forall x_m : T_m, M_1 \approx M_2$ with $\text{var}(M_2) \subseteq \text{var}(M_1)$. Informally, these claims mean that M_1 and M_2 evaluate to the same bitstring except in cases of negligible probability, provided that $y_1, \dots, y_l$ are chosen randomly with uniform probability and independently among $T'_1, \dots, T'_l$ respectively, and that $x_1, \dots, x_m$ are of type $T_1, \dots, T_m$. ($x_1, \dots, x_m$ may depend on $y_1, \dots, y_l$.) Formally, these claims are defined as: for all polynomials q, there exists a negligible $p(\eta)$ such that

$$\begin{aligned}\max_{\mathcal{A}} \Pr[&E(y_1) \stackrel{R}{\leftarrow} I_\eta(T'_1); \dots E(y_l) \stackrel{R}{\leftarrow} I_\eta(T'_l);\\ &(E(x_1), \dots, E(x_m)) \leftarrow \mathcal{A}(E(y_1), \dots, E(y_l));\\ &E, M_1 \Downarrow a; E, M_2 \Downarrow a' : a \neq a'] \leq p(\eta)\end{aligned}$$

  where $\mathcal{A}$ is a probabilistic Turing machine running in time $q(\eta)$.

  The above claim must be well-typed, that is, $\{x_1 \mapsto T_1, \dots, x_m \mapsto T_m, y_1 \mapsto T'_1, \dots, y_l \mapsto T'_l\} \vdash M_1 = M_2$.

  This claim is translated into the rewrite rule $\mathsf{new}\ y_1 : T'_1, \dots, \mathsf{new}\ y_l : T'_l, \forall x_1 : T_1, \dots, \forall x_m : T_m, M_1 \rightarrow M_2$.

The prover has built-in rewrite rules for defining boolean functions:

$$\neg\text{true} \rightarrow \text{false} \qquad \neg\text{false} \rightarrow \text{true} \qquad \forall x : \mathit{bool}, \neg(\neg x) \rightarrow x$$
$$\forall x : T, \forall y : T, \neg(x = y) \rightarrow x \neq y$$
$$\forall x : T, \forall y : T, \neg(x \neq y) \rightarrow x = y$$
$$\forall x : T, x = x \rightarrow \text{true} \qquad \forall x : T, x \neq x \rightarrow \text{false}$$
$$\forall x : \mathit{bool}, \forall y : \mathit{bool}, \neg(x \wedge y) \rightarrow (\neg x) \vee (\neg y)$$
$$\forall x : \mathit{bool}, \forall y : \mathit{bool}, \neg(x \vee y) \rightarrow (\neg x) \wedge (\neg y)$$
$$\forall x : \mathit{bool}, x \wedge \text{true} \rightarrow x \qquad \forall x : \mathit{bool}, x \wedge \text{false} \rightarrow \text{false}$$
$$\forall x : \mathit{bool}, x \vee \text{true} \rightarrow \text{true} \qquad \forall x : \mathit{bool}, x \vee \text{false} \rightarrow x$$

The prover also has support for commutative function symbols, that is, binary function symbols $f : T \times T \rightarrow T'$ such that for all $x, y \in I_\eta(T)$, $I_\eta(f)(x, y) = I_\eta(f)(y, x)$. For such symbols, all equality and matching tests are performed modulo commutativity. The functions $\wedge$, $\vee$, $=$, and $\neq$ are commutative. So, for instance, the last four rewrite rules above may also be used to rewrite $\text{true} \wedge M$ into M, $\text{false} \wedge M$ into false, $\text{true} \vee M$ into true, and $\text{false} \vee M$ into M. Used-defined functions may also be declared commutative; xor is an example of such a commutative function.

C.2 Collecting True Facts from a Game

We use *facts* to represent properties that hold at certain program points in processes. We consider two kinds of facts: $\mathsf{defined}(M)$ means that M is defined, and a term M means that M is true (the boolean term M evaluates to true). In this section, we show how to compute a set of facts $\mathcal{F}_P$ that are guaranteed to hold at the program point P of the game.

The function collectFacts collects facts that hold at each program point of the game. More precisely, for each occurrence P of a subprocess of the game, it computes a set $\mathcal{F}_P$ of facts that hold at that occurrence. (It is important that P is an occurrence and not a process: processes at several occurrences may be equal and must be distinguished from one another here.) The function collectFacts also computes a set $\mathcal{D}$ containing pairs $(x[\widetilde{i}], P)$ where $x[\widetilde{i}]$ has been defined just above process P. (If there are several definitions of x, there is one such pair for each definition of x.) Finally, for output processes P, $\text{collectFacts}(P)$ returns a set of facts that will hold when the next output is executed and stores this set in $\mathcal{F}_P^{\text{Fut}}$. (The superscript Fut stands for *future*, since these facts do not hold yet at P, but will hold in the future.)

The function collectFacts is defined in Figure 6. It is initially called by $\text{collectFacts}(Q_0)$. It takes into account that $x[\widetilde{i}]$ may be defined by an input, a restriction, a let, or a find and updates $\mathcal{D}$ accordingly. Furthermore, when we execute $\mathsf{let}\ x[\widetilde{i}] : T = M\ \mathsf{in}\ P'$, $x[\widetilde{i}] = M$ holds in P' and $x[\widetilde{i}]$ is defined in P'. When we execute $\mathsf{find}\ (\bigoplus_{j=1}^m u_{j1}[\widetilde{i}] \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}\ \mathsf{suchthat}\ \mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P'$, M_j holds in P_j, $M_{j1}, \dots, M_{jl_j}, u_{j1}[\widetilde{i}], \dots, u_{jm_j}[\widetilde{i}]$ are defined in P_j, and $\neg M_j$ holds in P' when $m_j = l_j = 0$.

After calling $\text{collectFacts}(Q_0)$, we complete the computed sets $\mathcal{F}_P$ (where P may be an input or output process) by adding facts that come from processes above P:

$$\mathcal{F}_P \leftarrow \mathcal{F}_P \cup \mathcal{F}_{P'} \text{ if } P \text{ is immediately under } P'$$

We also add facts that we can deduce from facts $\mathsf{defined}(M)$. Precisely, if $\mathsf{defined}(M) \in \mathcal{F}_P$ and $x[M_1, \ldots, M_m]$ is a subterm of M, then we take into account facts that are known to be true at the definitions of x by adding them to $\mathcal{F}_P$ as follows:

$$\mathcal{F}_P \leftarrow \mathcal{F}_P \cup \left(\bigcap_{(x[i_1,\ldots,i_m],P') \in \mathcal{D}} \begin{cases} \sigma(\mathcal{F}_{P'} \cup (\mathcal{F}_{P'}^{\mathrm{Fut}} \cap \mathcal{F}_P)) \\ \qquad\qquad \text{if } P \text{ is under } P' \\ \sigma(\mathcal{F}_{P'} \cup \mathcal{F}_{P'}^{\mathrm{Fut}}) \text{ otherwise} \end{cases} \right)$$

where $\sigma = \{M_1/i_1, \ldots, M_m/i_m\}$. Indeed, if $\mathsf{defined}(M) \in \mathcal{F}_P$ and $x[M_1, \ldots, M_m]$ is a subterm of M, then $x[M_1, \ldots, M_m]$ is defined at P, so some definition of $x[M_1, \ldots, M_m]$, just above the process P', must have been executed before reaching P, so the facts that hold at P' also hold at P, with a suitable substitution of indices: we have $\sigma\mathcal{F}_{P'}$, that is, $\mathcal{F}_{P'}\{M_1/i_1, \ldots, M_m/i_m\}$. Moreover, if the occurrence P is not syntactically under the occurrence P', then the code of P' must have been executed until the next output before yielding control to some other code and reaching P, so in fact $\sigma(\mathcal{F}_{P'} \cup \mathcal{F}_{P'}^{\mathrm{Fut}})$ hold. If P is syntactically under P', it is possible that the code of P' has been executed until reaching P instead of until reaching the next output, so we have only $\sigma(\mathcal{F}_{P'} \cup (\mathcal{F}_{P'}^{\mathrm{Fut}} \cap \mathcal{F}_P))$. If there are several definitions of x, we do not know which one has been executed, so we only add to $\mathcal{F}_P$ the facts that hold in all cases, by taking the intersection on all definitions of x.

This operation may add new defined facts to $\mathcal{F}_P$, so it is executed until a fixpoint is reached, except that, in order to avoid infinite loops, we do not execute this step for definitions $\mathsf{defined}(M)$ in which M contains nested occurrences of the same symbol (such as $x[\ldots x[\ldots] \ldots]$).

We also consider an additional fact that serves in expressing that the condition part of a find failed. Precisely, the fact $\mathit{elsefind}((u_1 \leq n_1, \ldots, u_m \leq n_m), (M_1, \ldots, M_l), M)$ means that for all $u_1 \in [1, n_1], \ldots, u_m \in [1, n_m]$, the terms $M_1, \ldots, M_l$ are not all defined or M is false. The function collectElseFind described in Figure 7 collects *elsefind* facts that hold at each occurrence. The function $\mathrm{collectElseFind}(P, \mathcal{F})$ is called when $\mathcal{F}$ is the set of true *elsefind* facts at occurrence P. It sets the value of $\mathcal{F}_P^{\mathrm{ElseFind}}$ to $\mathcal{F}$.

- In the case of restrictions, assignments, and then branches of find, it takes into account that a variable x or $u_{j1}, \ldots, u_{jm_j}$ is newly defined. Hence *elsefind* facts that claim that one of these variables is not defined are removed.

- In the case of the else branch of a find, it adds the new *elsefind* facts that hold when the conditions of the find fail. These conditions express that each then branch of the find fails by a *elsefind* fact. To construct this fact, we replace (by applying σ_j) the terms $u_{j1}[\widetilde{i}], \ldots, u_{jm_j}[\widetilde{i}]$ with fresh variables $u_1, \ldots, u_{m_j}$, respectively.

- In the case of an output, any code may be executed before the input processes under it, so any variable may be defined by that code, and all *elsefind* facts are removed. That is

$\mathrm{collectFacts}(Q) =$

  if $Q = Q_1 \mid Q_2$ then $\mathrm{collectFacts}(Q_1); \mathrm{collectFacts}(Q_2)$

  if $Q = !^{i \leq n} Q'$ then $\mathrm{collectFacts}(Q')$

  if $Q = \mathsf{newChannel}\ c; Q'$ then $\mathrm{collectFacts}(Q')$

  if $Q = c[M_1, \ldots, M_l](x_1[\widetilde{i}] : T_1, \ldots, x_k[\widetilde{i}] : T_k); P$ then

    $\mathcal{F}_P = \{\mathsf{defined}(x_j[\widetilde{i}]) \mid j \leq k\}; \mathcal{F}_P^{\mathrm{Fut}} = \mathrm{collectFacts}(P)$

    $\mathcal{D} = \mathcal{D} \cup \{(x_j[\widetilde{i}], P) \mid j \leq k\}$

$\mathrm{collectFacts}(P) =$

  if $P = \overline{c[M_1, \ldots, M_l]}\langle N_1, \ldots, N_k \rangle; Q$ then

    $\mathrm{collectFacts}(Q); \mathrm{return}\ \emptyset$

  if $P = \mathsf{new}\ x[\widetilde{i}] : T; P'$ then

    $\mathcal{F}_{P'} = \{\mathsf{defined}(x[\widetilde{i}])\}; \mathcal{F}_{P'}^{\mathrm{Fut}} = \mathrm{collectFacts}(P')$

    $\mathcal{D} = \mathcal{D} \cup \{(x[\widetilde{i}], P')\}; \mathrm{return}\ \mathcal{F}_{P'} \cup \mathcal{F}_{P'}^{\mathrm{Fut}}$

  if $P = \mathsf{let}\ x[\widetilde{i}] : T = M\ \mathsf{in}\ P'$ then

    $\mathcal{F}_{P'} = \{\mathsf{defined}(x[\widetilde{i}]), x[\widetilde{i}] = M\}$

    $\mathcal{F}_{P'}^{\mathrm{Fut}} = \mathrm{collectFacts}(P')$

    $\mathcal{D} = \mathcal{D} \cup \{(x[\widetilde{i}], P')\}; \mathrm{return}\ \mathcal{F}_{P'} \cup \mathcal{F}_{P'}^{\mathrm{Fut}}$

  if $P = \mathsf{find}\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] \leq n_{j1}, \ldots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}$

  $\mathsf{suchthat}\ \mathsf{defined}(M_{j1}, \ldots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\, P'$

  then

    for each $j \leq m$,

      $\mathcal{F}_{P_j} = \{\mathsf{defined}(u_{j1}[\widetilde{i'}]), \ldots, \mathsf{defined}(u_{jm_j}[\widetilde{i'}]),$

        $\mathsf{defined}(M_{j1}), \ldots, \mathsf{defined}(M_{jl_j}), M_j\}$

      $\mathcal{F}_{P_j}^{\mathrm{Fut}} = \mathrm{collectFacts}(P_j);$

      $\mathcal{D} = \mathcal{D} \cup \{(u_{j1}[\widetilde{i'}], P_j), \ldots, (u_{jm_j}[\widetilde{i'}], P_j)\}$

    $\mathcal{F}_{P'} = \{\neg M_j \mid m_j = l_j = 0\}; \mathcal{F}_{P'}^{\mathrm{Fut}} = \mathrm{collectFacts}(P')$

    $\mathrm{return}\ (\mathcal{F}_{P'} \cup \mathcal{F}_{P'}^{\mathrm{Fut}}) \cap \bigcap_{j=1}^{m} (\mathcal{F}_{P_j} \cup \mathcal{F}_{P_j}^{\mathrm{Fut}})$

Figure 6: The function collectFacts

why the function collectElseFind for input processes has no $\mathcal{F}$ argument (this argument would always be empty) and calls $\mathrm{collectElseFind}(P, \emptyset)$ for processes P that follow an input.

The *elsefind* facts can be used to add new facts to the facts $\mathcal{F}_P$. Indeed, if $\mathcal{F}_P$ implies that $M_1, \dots, M_l$ are defined for some values of $u_1, \dots, u_m$, then the fact $\mathit{elsefind}((u_1 \leq n_1, \dots, u_m \leq n_m), (M_1, \dots, M_l), M)$ implies that M is false for these values of $u_1, \dots, u_m$. Precisely, we execute:

$$\begin{aligned}&\mathcal{F}_P \leftarrow \mathcal{F}_P \cup \{\neg\sigma M \mid \mathit{elsefind}((u_1 \leq n_1, \dots, u_m \leq n_m),\\ &\quad (M_1, \dots, M_l), M) \in \mathcal{F}_P^{\mathrm{ElseFind}}, \mathrm{Dom}(\sigma) = \{u_1, \dots, u_m\},\\ &\quad \text{for each } j \in \{1, \dots, l\}, \sigma M_j \text{ is a subterm of } M'_j \text{ and}\\ &\quad \mathsf{defined}(M'_j) \in \mathcal{F}_P\}\end{aligned}$$

The possible images of σ are found by exploring the set of defined facts in $\mathcal{F}_P$.

In the implementation, an additional fact expresses that a pattern-matching failed: in the else branch of $\mathsf{let}\ N = M\ \mathsf{in}\ P\ \mathsf{else}\ Q$, we know that $\forall x_1, \dots, x_l, N \neq M$, where $x_1, \dots, x_l$ are the variables bound in the pattern N. In this report, we consider that the pattern-matching is encoded using assignments and tests, so we do not consider this fact further.

Furthermore, when the previous update of $\mathcal{F}_P$ adds facts, we again complete the computed sets $\mathcal{F}_P$ by adding facts that come from processes above P:

$$\mathcal{F}_P \leftarrow \mathcal{F}_P \cup \mathcal{F}_{P'} \text{ if } P \text{ is immediately under } P'$$

We could also iterate the addition of consequences of defined facts. (However, for simplicity, the current implementation does not perform such an iteration.)

$$\begin{aligned}
&\mathrm{collectElseFind}(Q) =\\
&\quad \text{if } Q = Q_1 \mid Q_2 \text{ then}\\
&\qquad \mathrm{collectElseFind}(Q_1); \mathrm{collectElseFind}(Q_2)\\
&\quad \text{if } Q = !^{i \leq n} Q' \text{ then } \mathrm{collectElseFind}(Q')\\
&\quad \text{if } Q = \mathsf{newChannel}\ c; Q' \text{ then } \mathrm{collectElseFind}(Q')\\
&\quad \text{if } Q = c[M_1, \dots, M_l](x_1[\widetilde{i}] : T_1, \dots, x_k[\widetilde{i}] : T_k); P \text{ then}\\
&\qquad \mathrm{collectElseFind}(P, \emptyset)
\end{aligned}$$

$$\begin{aligned}
&\mathrm{collectElseFind}(P, \mathcal{F}) =\\
&\quad \mathcal{F}_P^{\mathrm{ElseFind}} = \mathcal{F}\\
&\quad \text{if } P = \overline{c[M_1, \dots, M_l]}\langle N_1, \dots, N_k\rangle; Q \text{ then}\\
&\qquad \mathrm{collectElseFind}(Q)\\
&\quad \text{if } P = \mathsf{new}\ x[\widetilde{i}] : T; P'\\
&\quad \text{or } P = \mathsf{let}\ x[\widetilde{i}] : T = M\ \mathsf{in}\ P' \text{ then}\\
&\qquad \mathcal{F}' = \{\mathit{elsefind}((\widetilde{u} \leq \widetilde{n}), (M_1, \dots, M_l), M) \in \mathcal{F} \mid\\
&\qquad\qquad x \text{ does not occur in } M_1, \dots, M_l\}\\
&\qquad \mathrm{collectElseFind}(P', \mathcal{F}')\\
&\quad \text{if } P = \mathsf{find}\ (\textstyle\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}\\
&\quad \mathsf{suchthat}\ \mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P'\\
&\quad \text{then}\\
&\qquad \text{for each } j \leq m,\\
&\qquad\quad \mathcal{F}'_j = \{\mathit{elsefind}((\widetilde{u} \leq \widetilde{n}), (M_1, \dots, M_l), M) \in \mathcal{F}\\
&\qquad\qquad \mid u_{j1}, \dots, u_{jm_j} \text{ do not occur in } M_1, \dots, M_l\}\\
&\qquad\quad \mathrm{collectElseFind}(P_j, \mathcal{F}'_j)\\
&\qquad \sigma_j = \{u_1/u_{j1}[\widetilde{i}], \dots, u_{m_j}/u_{jm_j}[\widetilde{i}]\}\\
&\qquad \mathrm{collectElseFind}(P', \mathcal{F} \cup\\
&\qquad\quad \{\mathit{elsefind}((u_1 \leq n_{j1}, \dots, u_{m_j} \leq n_{jm_j}),\\
&\qquad\quad \sigma_j(M_{j1}, \dots, M_{jl_j}), \sigma_j M_j) \mid j \in \{1, \dots, m\}\})
\end{aligned}$$

Figure 7: The function collectElseFind

C.3 Global Dependency Analysis

For each variable x, the global dependency analysis tries to find a set of variables S such that only variables in S depend on x. In particular, when the global dependency analysis succeeds, the control flow and the view of the adversary do not depend on x, except in cases of negligible probability.

Let x be a variable defined only by restrictions $\mathsf{new}\ x : T$ where T is a large type. Let S_{def} be a set of variables defined only by assignments. Let S_{dep} be a set of variables containing x. (Intuitively, S_{dep} will be a superset of variables that depend on x.)

We say that a function $f : T \to T'$ is *uniform* when each element of $I_\eta(T')$ has at most $|I_\eta(T)|/|I_\eta(T')|$ antecedents by f. In particular, this is true in the following two cases:

- f is such that $f(x)$ is uniformly distributed in $I_\eta(T')$ if x is uniformly distributed in $I_\eta(T)$.
- f is the restriction to the image of f' of an inverse of f', where f' is a poly-injective function. (We consider that $f(x)$ is undefined when x is not in the image of f'. Here, in contrast to the rest of the paper, we allow $f : T \to T'$ to be defined only on a subset of $I_\eta(T)$.) Precisely, when $x_k \in S_{\mathrm{def}}$ is defined by a pattern-matching $\mathsf{let}\ f'(x_1, \dots, x_n) =$

M in P else P', we have $x_k = {f'}_k^{-1}(M)$, but furthermore when x_k is defined we know that the value of M is in the image of f', so we have $x_k = f(M)$ where $f = {f'}_k^{-1}{}_{|\mathrm{im}\ f'}$.

We say that M *characterizes a part of* x with S_{def},S_{dep} when for all M_0 obtained from M by substituting variables of S_{def} with their definition (when there is a dependency cycle among variables of S_{def}, we do not substitute a variable inside its definition), $\alpha M_0 = M_0$ implies $f_1(\dots f_k((\alpha x)[\widetilde{M'}])) = f_1(\dots f_k(x[\widetilde{M}]))$ for some uniform functions $f_1, \dots, f_k$ and for some $\widetilde{M}$ and $\widetilde{M'}$, where α is a renaming of variables of S_{dep} to fresh variables, $x[\widetilde{M}]$ is a subterm of M_0, $(\alpha x)[\widetilde{M'}]$ is a subterm of αM_0, the variables in S_{dep} do not occur in $\widetilde{M}$ or $\widetilde{M'}$, T is the type of the result of f_1 (or of x when $k = 0$), and T is a large type. In that case, the value of M uniquely determines the value of $f_1(\dots f_k(x[\widetilde{M}]))$.

We use a simple rewriting prover to determine that. We consider the set of terms $\mathcal{M}_0 = \{\alpha M_0 = M_0\}$, and we rewrite elements of $\mathcal{M}_0$ using the first kind of user-defined rewrite rules mentioned in Section C.1 and the rule $\{M_1 \wedge M_2\} \cup \mathcal{M}' \to \{M_1, M_2\} \cup \mathcal{M}'$.

When $\mathcal{M}_0$ can be rewritten to a set that contains an equality of the form $f_1(\dots f_k(x[\widetilde{M}])) = f_1(\dots f_k((\alpha x)[\widetilde{M'}]))$ or $f_1(\dots f_k((\alpha x)[\widetilde{M'}])) = f_1(\dots f_k(x[\widetilde{M}]))$ for some $\widetilde{M}$ and $\widetilde{M'}$ such that the variables in S_{dep} do not occur in $\widetilde{M}$ or $\widetilde{M'}$, we have that M characterizes a part of x with $S_{\mathrm{def}}, S_{\mathrm{dep}}$.

We say that M *characterizes a part of* x when M characterizes a part of x with $\emptyset, S'$ where S' is $\{x\}$ union the set of all variables except those defined by restrictions. (We know that variables different from x and defined by restrictions do not depend on x, so in the absence of more precise information, we can set $S_{\mathrm{dep}} = S'$.)

We say that $\mathrm{only\_dep}(x) = S$ when intuitively, only variables in S depend on x, and the adversary cannot see the value of x. Formally, $\mathrm{only\_dep}(x) = S$ when

- $S \cap V = \emptyset$.
- Variables of S do not occur in input or output channels or messages, that is, they do not occur in the terms $M_1, \dots, M_m, N_1, \dots, N_k$ in the input $c[M_1, \dots, M_m](x_1[\widetilde{i}] : T_1, \dots, x_k[\widetilde{i}] : T_k)$ or in the output $\overline{c[M_1, \dots, M_m]}\langle N_1, \dots, N_k\rangle$.
- Variables of S except x are defined only by assignments.
- If a variable $y \in S$ occurs in M in let $z : T = M$ in P, then $z \in S$.
- Variables in S may occur in defined conditions of find but only at the root of them.
- All terms M_j in processes find $(\bigoplus_{j=1}^m \widetilde{u_j}[\widetilde{i}] \leq \widetilde{n_j}$ suchthat defined$(M_{j1}, \dots, M_{jl_j}) \wedge M_j$ then $P_j)$ else P' are combinations by $\wedge$, $\vee$, or $\neg$ of terms that either do not contain variables in S or are of the form $M_1 = M_2$ or $M_1 \neq M_2$ where M_1 characterizes a part of x with $S \setminus \{x\}, S$ and no variable of S occurs in M_2, or M_2 characterizes a part of x with $S \setminus \{x\}, S$ and no variable of S occurs in M_1.

The last item implies that the result of tests does not depend on the values of variables in S, except in cases of negligible probability. Indeed, the tests $M_1 = M_2$ with M_1 characterizes a part of x with $S \setminus \{x\}, S$ and M_2 does not depend on variables in S are false except in cases of negligible probability, since the value of M_1 uniquely determines the value of $f_1(\dots f_k(x[\widetilde{M}]))$ and M_2 does not depend on $f_1(\dots f_k(x[\widetilde{M}]))$, so the equality $M_1 = M_2$ happens for a single value of $f_1(\dots f_k(x[\widetilde{M}]))$, which yields a negligible probability because $f_1, \dots, f_k$ are uniform, x is chosen with uniform probability, and the type of the result of f_1 is large. Similarly, the tests $M_1 \neq M_2$ are true except in cases of negligible probability.

In checking the conditions of $\mathrm{only\_dep}(x) = S$, we do not consider the parts of the code that are unreachable due to tests whose result is known by the conditions above.

The set S is computed by a fixpoint iteration, starting from $\{x\}$ and adding variables defined by assignments that depend on variables already in S.

If we manage to show that $\mathrm{only\_dep}(x) = S$, we transform the game as follows:

- We replace with false terms $M_1 = M_2$ in conditions of find where M_1 characterizes a part of x with $S \setminus \{x\}, S$ and no variable of S occurs in M_2, or symmetrically.
- We replace with true terms $M_1 \neq M_2$ in conditions of find where M_1 characterizes a part of x with $S \setminus \{x\}, S$ and no variable of S occurs in M_2, or symmetrically.

C.4 Local Dependency Analysis

For each program point P and each variable x, the local dependency analysis tries to find which variables and terms depend on $x[\widetilde{i}]$ at program point P, where $\widetilde{i}$ denotes the current replication indices at the definition of x. It simplifies the game on-the-fly when possible.

For each occurrence of a process P and each variable x such that a restriction new $x : T$ occurs above P and T is a large type, we compute a set of terms $\mathrm{indep}_P(x)$ that are independent of $x[\widetilde{i}]$ where $\widetilde{i}$ denotes the current replication indices at the definition of x.

For each occurrence of a process P and each variable x such that a restriction new $x : T$ occurs above P and T is a large type, we also compute $\mathrm{depend}_P(x)$ which can be either $\top$ (I don't know) or a set of pairs (y, M) where $y[\widetilde{i}]$ depends on $x[\widetilde{i}]$ by assignments, and M is a term defining $y[\widetilde{i}]$ as a function of $x[\widetilde{i}]$. (The tuple $\widetilde{i}$ denotes the current replication indices at the definition of x and of y.)

We define "M *characterizes a part of* $x[\widetilde{i}]$ at P" as follows. Let α be defined by $\alpha(f(M_1, \dots, M_m)) = f(\alpha M_1, \dots, \alpha M_m)$; $\alpha(i) = i$ where i is a replication index; $\alpha(M') = M'$ when $M' \in \mathrm{indep}_P(x)$; $\alpha(y[M_1, \dots, M_{m'}]) = y[\alpha M_1, \dots, \alpha M_{m'}]$ when $y \neq x$ and y either is defined only by restrictions or $\mathrm{depend}_P(x) \neq \top$ and $(y, M') \notin \mathrm{depend}_P(x)$ for any M'; $\alpha(y[M_1, \dots, M_{m'}]) = y'[\alpha M_1, \dots, \alpha M_{m'}]$ where y' is a fresh variable, otherwise. We write $y' = \alpha y$ in this case. We say that M *characterizes a part of* $x[\widetilde{i}]$ at P when $\alpha M = M$ implies $f_1(\dots f_k((\alpha x)[\widetilde{i}])) = f_1(\dots f_k(x[\widetilde{i}]))$ for

$\mathrm{depAnal}(Q, \mathrm{indep}) =$

$\quad \forall y, \mathrm{depend}_Q(y) = \top; \mathrm{indep}_Q = \mathrm{indep}$

$\quad$ if $Q = Q_1 \mid Q_2$ then

$\quad\quad \mathrm{depAnal}(Q_1, \mathrm{indep}); \mathrm{depAnal}(Q_2, \mathrm{indep})$

$\quad$ if $Q = !^{i \leq n} Q'$ then $\mathrm{depAnal}(Q', \mathrm{indep})$

$\quad$ if $Q = \mathsf{newChannel}\ c; Q'$ then $\mathrm{depAnal}(Q', \mathrm{indep})$

$\quad$ if $Q = c[M_1, \dots, M_l](x_1[\widetilde{i}] : T_1, \dots, x_k[\widetilde{i}] : T_k); P$ then

$\quad\quad \mathrm{depAnal}(P, \{\forall y, y \mapsto \top\}, \mathrm{indep})$

Figure 8: Local dependency analysis (1)

some uniform functions $f_1, \dots, f_k$, where $x[\widetilde{i}]$ is a subterm of M, $(\alpha x)[\widetilde{i}]$ is a subterm of αM, T' is the type of the result of f_1 (or of x when $k = 0$), and T' is a large type. In that case, the value of M uniquely determines the value of $f_1(\dots f_k(x[\widetilde{i}]))$. This property is shown by a simple rewriting prover, as in the global dependency analysis.

We denote by $\mathrm{subterms}(M)$ the set of subterms of the term M.

We say that M does not depend on x at P when M is built by function applications from terms in $\mathrm{indep}_P(x)$, replications indices, and terms $y[M_1, \dots, M_m]$ such that $M_1, \dots, M_m$ do not depend on x at P, $y \neq x$, and either y is defined only by restrictions or $\mathrm{depend}_P(x) \neq \top$ and $y \neq y'$ for all $(y', M') \in \mathrm{depend}_P(x)$. Since terms in $\mathrm{indep}_P(x)$ do not depend on $x[\widetilde{i}]$ and when $\mathrm{depend}_P(x) \neq \top$, variables not in the first component of $\mathrm{depend}_P(x)$ do not depend on $x[\widetilde{i}]$, the conditions above guarantee that M does not depend on $x[\widetilde{i}]$, where $\widetilde{i}$ are the current replication indices at the definition of x.

When $\mathrm{depend} \neq \top$, we denote by $M\mathrm{depend}$ the term obtained from M by replacing $y[\widetilde{i}]$ with M' for each $(y, M') \in \mathrm{depend}$, where $\widetilde{i}$ denotes the replication indices at the definition of y.

We define $\mathrm{simplifyTerm}$ such that $\mathrm{simplifyTerm}(M, P)$ is a simplified version of M, equal to M except in cases of negligible probability. The term $\mathrm{simplifyTerm}(M, P)$ is defined as follows:

- Case 1: M is $M_1 = M_2$. For each x, we proceed as follows. If $\mathrm{depend}_P(x) = \top$, let $M_0 = M_1$; otherwise, let $M_0 = M_1\mathrm{depend}_P(x)$. Let M_0' and M_2' be obtained respectively from M_0 and M_2 by replacing all array indices that depend on x at P with fresh replication indices. If M_0' characterizes a part of $x[\widetilde{i}]$ at P, and M_2' does not depend on x at P, then $\mathrm{simplifyTerm}(M, P) = \mathrm{false}$. Indeed, M is equal to false up to negligible probability in this case. We have similar cases swapping M_1 and M_2 or when M is $M_1 \neq M_2$. (In the latter case, $\mathrm{simplifyTerm}(M, P) = \mathrm{true}$.)

- Case 2: M is $M_1 \wedge M_2$. Let $M_1' = \mathrm{simplifyTerm}(M_1, P)$ and $M_2' = \mathrm{simplifyTerm}(M_2, P)$. If M_1' or M_2' are false, we return false. If M_1' is true, we return M_2'. If M_2' is true, we return M_1'. Otherwise, we return $M_1' \wedge M_2'$. We have similar cases when M is $M_1 \vee M_2$ or $\neg M_1$.

$\mathrm{depAnal}(P, \mathrm{depend}, \mathrm{indep}) =$

$\quad \mathrm{depend}_P = \mathrm{depend}; \mathrm{indep}_P = \mathrm{indep}$

$\quad$ if $P = \overline{c[M_1, \dots, M_l]}\langle N_1, \dots, N_k\rangle; Q$ then

$\quad\quad \mathrm{depAnal}(Q, \mathrm{indep})$

$\quad$ if $P = \mathsf{new}\ x[\widetilde{i}] : T; P'$ then

$\quad\quad$ if T is a large type then

$\quad\quad\quad \mathrm{depend}'(x) = \emptyset$

$\quad\quad\quad \mathrm{indep}'(x) = \bigcup_{\mathsf{defined}(M) \in \mathcal{F}_P} \mathrm{subterms}(M)$

$\quad\quad \forall y \neq x, \mathrm{depend}'(y) = \mathrm{depend}(y),$

$\quad\quad\quad \mathrm{indep}'(y) = \mathrm{indep}(y) \cup \{x[\widetilde{i}]\}$

$\quad\quad \mathrm{depAnal}(P', \mathrm{depend}', \mathrm{indep}')$

$\quad$ if $P = \mathsf{let}\ x[\widetilde{i}] : T = M\ \mathsf{in}\ P'$ then

$\quad\quad \forall y,$ if M does not depend on y at P then

$\quad\quad\quad \mathrm{depend}'(y) = \mathrm{depend}(y)$

$\quad\quad\quad \mathrm{indep}'(y) = \{x[\widetilde{i}]\} \cup \mathrm{indep}(y)$

$\quad\quad$ else

$\quad\quad\quad$ if $\mathrm{depend}(y) \neq \top$ then

$\quad\quad\quad\quad \mathrm{depend}'(y) = \mathrm{depend}(y) \cup \{(x, M\mathrm{depend}(y))\}$

$\quad\quad\quad$ else

$\quad\quad\quad\quad \mathrm{depend}'(y) = \top$

$\quad\quad\quad \mathrm{indep}'(y) = \mathrm{indep}(y)$

$\quad\quad \mathrm{depAnal}(P', \mathrm{depend}', \mathrm{indep}')$

$\quad$ if $P = \mathsf{find}\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}$ $\mathsf{suchthat}\ \mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P'$ then

$\quad\quad$ for each $j \leq m, M_j' = \mathrm{simplifyTerm}(M_j, P)$

$\quad\quad\quad$ replace M_j with M_j'

$\quad\quad\quad$ if $M_j' = \mathrm{false}$ then remove the j-th branch

$\quad\quad\quad$ if $M_j' = \mathrm{true}$ and $l_j = 0$ then replace P' with $\overline{yield}\langle\rangle$

$\quad\quad$ if $m = 0$ then

$\quad\quad\quad$ replace P with P'; $\mathrm{depAnal}(P', \mathrm{depend}, \mathrm{indep})$

$\quad\quad$ else if $m = 1, m_1 = l_1 = 0$, and $M_1 = \mathrm{true}$ then

$\quad\quad\quad$ replace P with P_1; $\mathrm{depAnal}(P_1, \mathrm{depend}, \mathrm{indep})$

$\quad\quad$ else

$\quad\quad \forall y$, if $\forall j, k, M_{jk}$ and M_j' do not depend on y at P then

$\quad\quad\quad \mathrm{depend}'(y) = \mathrm{depend}(y)$

$\quad\quad\quad$ for each $j \leq m, \mathrm{indep}_j(y) = \mathrm{indep}(y) \cup \{M' \mid$

$\quad\quad\quad\quad M' \in \mathrm{subterms}(M)$ for some $\mathsf{defined}(M) \in \mathcal{F}_{P_j}$,

$\quad\quad\quad\quad M'$ does not depend on y at $P\}$

$\quad\quad\quad$ else

$\quad\quad\quad \mathrm{depend}'(y) = \top$

$\quad\quad\quad$ for each $j \leq m, \mathrm{indep}_j(y) = \mathrm{indep}(y)$

$\quad\quad$ for each $j \leq m, \mathrm{depAnal}(P_j, \mathrm{depend}', \mathrm{indep}_j)$

$\quad\quad \mathrm{depAnal}(P', \mathrm{depend}', \mathrm{indep})$

Figure 9: Local dependency analysis (2)

- In all other cases, $\mathrm{simplifyTerm}(M, P) = M$.

The local dependency analysis is defined in Figures 8 and 9. The function $\mathrm{depAnal}$ is initially called with $\mathrm{depAnal}(Q_0, \emptyset)$ where $\emptyset$ designates the function defined nowhere.

- For input processes, $\mathrm{depAnal}$ sets $\mathrm{depend}_Q(y)$ to $\top$, so that depend_Q gives no information, and propagates indep. Indeed, when $y[\widetilde{i'}]$ is set in some output process P_0, the value of $y[\widetilde{i'}]$ may be output by P_0 or read by find in other output processes executed after P_0, so as soon as P_0 passes control to another process by the first output after the definition of y, we lose track of exactly which variables depend on $y[\widetilde{i'}]$. However, variables already defined before P_0 passes control to another process and proved to be independent of $y[\widetilde{i'}]$ remain independent of $y[\widetilde{i'}]$, so we can propagate indep in all subprocesses of P_0.
- In the case of an output, $\mathrm{depAnal}$ forgets the information in depend_P as mentioned above.
- In the case of a restriction $\mathsf{new}\ x[\widetilde{i}] : T$, if T is a large type, we create the dependency information for the newly defined variable x: no variable depends on $x[\widetilde{i}]$, and all terms already defined before the restriction are independent of $x[\widetilde{i}]$. We also note that $x[\widetilde{i}]$ is independent of $y[\widetilde{i'}]$ for other variables y by adding $x[\widetilde{i}]$ to $\mathrm{indep}(y)$.
- In the case of an assignment $\mathsf{let}\ x[\widetilde{i}] : T = M$, if M depends on $y[\widetilde{i'}]$ for some variable y, then $x[\widetilde{i}]$ depends on $y[\widetilde{i'}]$, so x is added to $\mathrm{depend}(y)$ (if it is not $\top$); otherwise, $x[\widetilde{i}]$ does not depend on $y[\widetilde{i'}]$ so it is added to $\mathrm{indep}(y)$.
- In the case of a find, we first simplify each condition of the find, remove branches when we can prove that they are taken with negligible probability, and remove the find itself when we know which branch is taken and this branch of the find does not define variables. Furthermore, if some condition of find depends on $y[\widetilde{i}]$ for some variable y, $\mathrm{depend}'(y)$ is set to $\top$: the control flow depends on $y[\widetilde{i}]$ so future assignments in fact depend on $y[\widetilde{i}]$ even if the assigned expression itself does not, so we can no longer keep track precisely of which variables depend on $y[\widetilde{i}]$. Otherwise, we add all terms that are guaranteed to be defined and independent of $y[\widetilde{i}]$ to $\mathrm{indep}(y)$.

C.5 Equational Prover

We use an algorithm inspired by the Knuth-Bendix completion algorithm [30], with differences detailed below.

The prover manipulates pairs $\mathcal{F}, \mathcal{R}$ where $\mathcal{F}$ is a set of facts (M or $\mathsf{defined}(M)$) and $\mathcal{R}$ is a set of rewrite rules $M_1 \to M_2$. We say that M reduces into M' by $M_1 \to M_2$ when $M = C[M_1]$ and $M' = C[M_2]$ for some term context C. (That is, all variables in rewrite rules of $\mathcal{R}$ are considered as constants.) The prover starts with a certain set of facts $\mathcal{F}$ and $\mathcal{R} = \emptyset$. Then the prover transforms the pairs $(\mathcal{F}, \mathcal{R})$ by the following rules (the rule $\frac{\mathcal{F},\mathcal{R}}{\mathcal{F}',\mathcal{R}'}$ means that $\mathcal{F}, \mathcal{R}$ is transformed into $\mathcal{F}', \mathcal{R}'$):

$$\frac{\mathcal{F} \cup \{F\}, \mathcal{R}}{\mathcal{F} \cup \{F'\}, \mathcal{R}} \quad \text{if } F \text{ reduces into } F' \text{ by a rule of } \mathcal{R} \text{ or a user-defined rewrite rule knowing } \mathcal{F}, \mathcal{R} \tag{1}$$

$$\frac{\mathcal{F} \cup \{M_1 \wedge M_2\}, \mathcal{R}}{\mathcal{F} \cup \{M_1, M_2\}, \mathcal{R}} \tag{2}$$

$$\frac{\mathcal{F} \cup \{x[M_1, \ldots, M_m] = x[M'_1, \ldots, M'_m]\}, \mathcal{R}}{\mathcal{F} \cup \{M_1 = M'_1, \ldots, M_m = M'_m\}, \mathcal{R}} \tag{3}$$

when x is defined only by restrictions $\mathsf{new}\ x : T$ and T is a large type

$$\frac{\mathcal{F} \cup \{M_1 = M_2\}, \mathcal{R}}{\{\mathrm{false}\}, \mathcal{R}} \quad \text{when one of the following conditions holds:}$$

- denoting by M'_1 the term obtained from M_1 by replacing all array indices that are not replication indices with fresh replication indices, we have the following properties: x occurs in M'_1, x is defined only by restrictions $\mathsf{new}\ x : T$, T is a large type, M'_1 characterizes a part of x, and M_2 is obtained by optionally applying function symbols to terms of the form $y[\widetilde{M}]$ where y is defined only by restrictions and $y \neq x$;
- $\mathrm{simplifyTerm}(M_1 = M_2, P) = \mathrm{false}$, where P is the current program point.

(4)

$$\frac{\mathcal{F} \cup \{M = M'\}, \mathcal{R}}{\mathcal{F}, \mathcal{R} \cup \{M \to M'\}} \text{ if } M > M' \tag{5}$$

$$\frac{\mathcal{F}, \mathcal{R} \cup \{M_1 \to M_2\}}{\mathcal{F} \cup \{M_1 = M'_2\}, \mathcal{R}} \quad \text{if } M_2 \text{ reduces into } M'_2 \text{ by a rule of } \mathcal{R} \text{ or a user-defined rewrite rule knowing } \mathcal{F}, \mathcal{R} \tag{6}$$

$$\frac{\mathcal{F}, \mathcal{R} \cup \{M_1 \to M_2\}}{\mathcal{F} \cup \{M'_1 = M_2\}, \mathcal{R}} \text{ if } M_1 \text{ reduces into } M'_1 \text{ by a rule of } \mathcal{R} \tag{7}$$

We also use the symmetrics of Rules (4) and (5) obtained by swapping the two sides of the equality.

Rule (1) simplifies facts using rewrite rules. Let us define when M reduces into M' by a user-defined rewrite rule knowing $\mathcal{F}, \mathcal{R}$. For the first kind of rewrite rules of Section C.1, this is simply when M reduces into M' by the considered rewrite rule. For the second kind of rewrite rules of Section C.1, consider a rewrite rule $\mathsf{new}\ y_1 : T'_1, \ldots, \mathsf{new}\ y_l : T'_l, \forall x_1 : T_1, \ldots, \forall x_m : T_m, M_1 \to M_2$. Suppose that $M = C[\sigma M_1]$, where C is a term context and σ is a substitution that maps x_j to any term of type T_j for all $j \leq m$ and y_j to terms to the form $z_j[\widetilde{M_j}]$ where z_j is defined only by restrictions $\mathsf{new}\ z_j : T'_j$ for all $j \leq l$. Let $\mathcal{C}ond = \{\widetilde{M_j} = \widetilde{M_{j'}} \mid j \neq j' \wedge z_j = z_{j'}\}$.

- If $\mathcal{C}ond = \emptyset$, the restrictions $z_j[\widetilde{M_j}]$ are independent, so $M = C[\sigma M_1]$ reduces into $M' = C[\sigma M_2]$.
- If $\mathcal{C}ond \neq \emptyset$, let $\mathcal{C}ond = \{cond_1, \ldots, cond_k\}$. When $cond_1 \vee \ldots \vee cond_k$ is true, the restrictions $z_j[\widetilde{M_j}]$ are not independent, so we must leave M as it is. When $cond_1 \vee \ldots \vee cond_k$ is false, M reduces into $M' = C[\sigma M_2]$. Suppose that $M' = \mathrm{false}$. Hence, we could rewrite M into $\mathsf{if}\ cond_1 \vee \ldots \vee cond_k\ \mathsf{then}\ M\ \mathsf{else\ false}$, that is, $(cond_1 \vee \ldots \vee cond_k) \wedge M$, that is, $(cond_1 \wedge M) \vee \ldots \vee (cond_k \wedge M)$. However, since this term itself contains M, this transformation could lead to a loop. We avoid it as follows. If for all $k' = 1, \ldots, k$, M is transformed into $M'_{k'}$ by rewrite rules generated by our equational prover from $\mathcal{F} \cup \{cond_{k'}\}, \mathcal{R}$

and user-defined rewrite rules, using only the first kind of user-defined rewrite rules of Section C.1, and $M'_{k'} \neq M_k$, then we rewrite M into $(cond_1 \wedge M'_1) \vee \ldots \vee (cond_k \wedge M'_k)$. Otherwise, M does not rewrite by the considered user-defined rewrite rule knowing $\mathcal{F}, \mathcal{R}$.

Rule (2) decomposes conjunctions of facts. Rules (3) and (4) exploit the elimination of collisions between random values. Rule (3) takes into account that, when x is defined by a restriction of a large type, two different cells of x have a negligible probability of containing the same value. So when two cells of x contain the same value, we can conclude up to negligible probability that they are the same cell. Rule (4) expresses that M_1 and M_2 have a negligible probability of being equal when x is defined by a restriction of a large type, M_1 characterizes a part of x, and M_2 does not depend of x. The first item of (4) establishes these properties without further dependency analysis and the second item exploits the local dependency analysis. Additionally, if $\mathcal{F}$ contains $M_1 = M_2$ such that x occurs in M_1, x is defined only by restrictions new $x : T$, T is a large type, and M_1 characterizes a part of x, we trigger the global dependency analysis (Section C.3). If the global dependency analysis succeeds in transforming the game, the simplification is restarted from the game obtained after global dependency analysis.

Rule (5) is applied only when Rules (1) to (4) cannot be applied. Rule (5) transforms equations into rewrite rules by orienting them. We say that $M > M'$ when either M is the form $x[\widetilde{M}]$, x does not occur in M', and x is not defined only by restrictions, or $M = x[M_1, \ldots, M_m]$, $M' = x[M'_1, \ldots, M'_m]$, and for all $j \leq m$, $M_j > M'_j$. Intuitively, our goal is to replace M with M' when M' defines the content of the variable M. (Notice that this is not an ordering; the Knuth-Bendix algorithm normally uses a reduction ordering to orient equations. However, we tried some reduction orderings, namely the lexicographic path ordering and the Knuth-Bendix ordering, and obtained disappointing results: the prover fails to prove many equalities because too many equations are left unoriented. The simple heuristic given above succeeds more often, at the expense of a greater risk of non-termination, but that does not cause problems in practice on our examples. We believe that this comes from the particular structure of equations, which come from let definitions and from conditions of find or if, and tend to define variables from other variables without creating dependency cycles.)

Rules (6) and (7) are systematically applied to simplify all rewrite rules of $\mathcal{R}$ after a new rewrite rule has been added by Rule (5). Since all terms in rewrite rules of $\mathcal{R}$ are considered as constants, Rule (7) in fact includes the deduction of equations from critical pairs done by the standard Knuth-Bendix completion algorithm.

We say that $\mathcal{F}$ *yields a contradiction* when the prover, starting from $(\mathcal{F}, \emptyset)$, derives false.

C.6 Game Simplification

We use the following transformations in order to simplify games. These transformations exploit the information collected as explained in the previous sections.

- Each term M in the game is replaced with a simplified term M' obtained by reducing M by user-defined rewrite rules knowing $\mathcal{F}_{P_M}$ (see Sections C.1 and C.5) and the rewrite rules obtained from $\mathcal{F}_{P_M}$ by the above equational prover where P_M is the smallest process containing M. The replacement is performed only when at least one user-defined rewrite rule has been used, to avoid complicating the game by substituting all variables with their value.

- If $P = \mathsf{find}\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] \leq n_{j1}, \ldots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}$ $\mathsf{suchthat\ defined}(M_{j1}, \ldots, M_{jl_j}) \wedge M_j$ $\mathsf{then}\ P_j)$ $\mathsf{else}\ P'$, $u_{jk}[\widetilde{i}]$ reduces into M' by user-defined rewrite rules knowing $\mathcal{F}_{P_j}$ (see Sections C.1 and C.5) and the rewrite rules obtained from $\mathcal{F}_{P_j}$, u_{jk} does not occur in M', and M_{j1}, $\ldots$, M_{jl_j}, M_j make no array accesses to u_{jk} (with indices different from the current indices), then u_{jk} is removed from the j-th branch of this find, $u_{jk}[\widetilde{i}]$ is replaced with M' in $M_{j1}, \ldots, M_{jl_j}, M_j$ and P_j is replaced with $\mathsf{let}\ u_{jk}[\widetilde{i}] : [1, n_{jk}] = M'$ $\mathsf{in}\ P_j$. (Intuitively, $u_{jk}[\widetilde{i}] = M'$, so the value of $u_{jk}[\widetilde{i}]$ can be computed by evaluating M' instead of performing an array lookup. We remove $u_{jk}[\widetilde{i}]$ from the variables looked up by find and replace $u_{jk}[\widetilde{i}]$ with its value M'.)

- Suppose that $P = \mathsf{find}\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] \leq n_{j1}, \ldots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}$ $\mathsf{suchthat\ defined}(M_{j1}, \ldots, M_{jl_j}) \wedge M_j$ $\mathsf{then}\ P_j)$ $\mathsf{else}\ P'$, there exists a term M such that $\mathsf{defined}(M) \in \mathcal{F}_{P_j}$, $x[N_1, \ldots, N_l]$ is a subterm of M, $x \neq u_{jk}$ for all $k \leq m_j$, and none of the following conditions holds: a) P is under a definition of x in Q_0; b) Q_0 contains $Q_1 \mid Q_2$ such that a definition of x occurs in Q_1 and P is under Q_2 or a definition of x occurs in Q_2 and P is under Q_1; c) Q_0 contains $lp+1$ replications above a process Q that contains a definition of x and P, where lp is the length of the longest common prefix between $N_1, \ldots, N_l$ and the current replication indices at the definitions of x. Then the j-th branch of the find is removed. (In this case, $x[N_1, \ldots, N_l]$ cannot be defined at P, so the j-th branch of the find cannot be taken.)

- Suppose that $P = \mathsf{find}\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] \leq n_{j1}, \ldots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}$ $\mathsf{suchthat\ defined}(M_{j1}, \ldots, M_{jl_j}) \wedge M_j$ $\mathsf{then}\ P_j)$ $\mathsf{else}\ P'$, there exist terms M, M' such that $\mathsf{defined}(M) \in \mathcal{F}_{P_j}$, $x[N_1, \ldots, N_l]$ is a subterm of M, $\mathsf{defined}(M') \in \mathcal{F}_{P_j}$, $x'[N'_1, \ldots, N'_{l'}]$ is a subterm of M', $N_k = N'_k$ for all $k \leq \min(l, l')$, $x \neq x'$, and x and x' are incompatible, then the j-th branch of the find is removed. Two variables x and x' are said to be compatible when either there exists $Q_1 \mid Q_2$ in the game such that x is defined in Q_1 and x' is defined in Q_2, or there is a definition of x' under a definition of x, or symmetrically.

- If $P = \mathsf{find}\ (\bigoplus_{j=1}^{m} \widetilde{u_j}[\widetilde{i}] \leq \widetilde{n_j}$ $\mathsf{suchthat\ defined}(M_{j1}, \ldots, M_{jl_j}) \wedge M_j$ $\mathsf{then}\ P_j)$ $\mathsf{else}\ P'$ and $\mathcal{F}_{P_j}$ yields a contradiction, then the j-th branch of the find is removed.

- If $P = \mathsf{find\ else}\ P'$, then P is replaced with P'.

- If find $(\bigoplus_{j=1}^{m} \widetilde{u_j}[\widetilde{i}] \leq \widetilde{n_j}$ suchthat defined$(M_{j1}, \ldots, M_{jl_j}) \wedge M_j$ then $P_j)$ else P' and $\mathcal{F}_{P'}$ yields a contradiction, then P' is replaced with $\overline{yield}\langle\rangle$.

- If $P =$ find $\widetilde{u}[\widetilde{i}] \leq \widetilde{n}$ suchthat M then P_1 else P', $\mathcal{F}_{P'}$ yields a contradiction, and the variables in $\widetilde{u}$ are not used outside P and are not in V, then P is replaced with P_1. (When the find defines variables $\widetilde{u}$ used elsewhere, we cannot remove it.)

- If $P =$ find $(\bigoplus_{j=1}^{m} \widetilde{u_j}[\widetilde{i}] \leq \widetilde{n_j}$ suchthat defined$(M_{j1}, \ldots, M_{jl_j}) \wedge M_j$ then $\overline{yield}\langle\rangle)$ else $\overline{yield}\langle\rangle$ and the variables in $\widetilde{u_j}$ are not used outside P and are not in V, then P is replaced with $\overline{yield}\langle\rangle$.

- The defined conditions of find are updated so that Invariant 2 is satisfied. (When such a defined condition guarantees that M is defined, defined(M) implies defined(M'), and after simplification M' appears in the scope of this condition, then M' has to be added to this condition if it is not already present.)

- If $P =$ new $x : T; P'$ or let $x : T = M$ in P' and x is not used in the game and is not in V, then P is replaced with P'.

C.7 Further Simplifications

After applying the game simplifications described above, we further apply the following transformations:

Move(all): We move restrictions and assignments downwards in the code as much as possible. A restriction new $x[\widetilde{i}] : T$ or assignment let $x[\widetilde{i}] : T = M$ cannot be moved under a replication, or under a parallel composition when both sides use x, or a let let $y[\widetilde{i}] : T = M$ in ..., input $c[M_1, \ldots, M_l](x_1[\widetilde{i}] : T_1, \ldots, x_k[\widetilde{i}] : T_k)$, output $\overline{c[M_1, \ldots, M_l]}\langle N_1, \ldots, N_k\rangle$ when x occurs in $M, M_1, \ldots, M_l, N_1, \ldots, N_k$, or a find (or if) when the conditions use x. It can be moved under the other constructs, duplicating it if necessary, when we move it under a find (or if) that uses x in several branches. Note that when the restriction new $x[\widetilde{i}] : T$ or assignment let $x[\widetilde{i}] : T = M$ cannot be moved under an input, a parallel composition, or a replication, it must be written above the output that is located above the considered input, parallel composition or replication, so that the syntax of processes is not violated. When there are array accesses to x, the restriction new $x[\widetilde{i}] : T$ or assignment let $x[\widetilde{i}] : T = M$ can be moved only inside the same output process, without moving it under an output or under a find that makes an array access to x.

The conditions above are necessary for the soundness of the move. Furthermore, the move is performed only if it is helpful, following conditions that we detail next.

For restrictions, the move is performed only when the restriction can be moved under a find (or if). When this transformation duplicates a new $x[\widetilde{i}] : T$ by moving it under a find that uses x in several branches, a subsequent **SArename**(x) enables us to distinguish several cases depending in which branch x is created, which is useful in some proofs.

For assignments, the assignment to x is moved only when there are no array accesses to x, the assignment to x can be moved under a find (or if), and x is used in a single branch of that find (or if). In this case, the assignment can be performed only in the branch that uses x, so it will be computed in fewer cases thanks to the move.

RemoveAssign(useless): As a particular case of the transformation **RemoveAssign**, we remove useless assignments, that is, assignments to x when x is unused and assignments let $x[\widetilde{i}] : T = y[\widetilde{M}]$. Since removing such assignments may also remove uses of other variables, we repeat this removal until a fixpoint is reached.

SArename(auto): As a particular case of the transformation **SArename**, when x has $m > 1$ definitions and all variable accesses to x are of the form $x[i_1, \ldots, i_l]$ under a definition of $x[i_1, \ldots, i_l]$, where $i_1, \ldots, i_l$ are the current replication indices at this definition of x (that is, x has no array access using find), we rename x to $x_1, \ldots, x_m$ with a different name for each definition.

D Applying the Definition of Security of Primitives

D.1 Formalization of the Transformation

In this appendix, we formalize the transformation performed by exploiting equivalences that come from the definition of security of cryptographic primitives. We require the following conditions for the equivalences $L \approx R$ that model cryptographic primitives:

H0. $[\![L]\!]$ and $[\![R]\!]$ satisfy Invariants 1, 2, and 3. Furthermore, the result of each function in R has the same type as the result of the corresponding function of L.

H1. In L, the functional processes FP are simply terms M; all their array accesses use the current replication indices. (Allowing let or find in L is difficult, because we need to recognize the terms M in a context and in a possibly syntactically modified form.)

H2. L and R have the same structure: same replications, same number of functions, same number of arguments with the same types for each function.

H3. The variables y_j defined by new and x_j defined by function inputs in L and R are distinct from other variables defined in R.

H4. Under $!^{i \leq n}$ with no restriction in L, one can have only a single function $(x_1 : T_1, \ldots, x_l : T_l) \rightarrow FP$. (One can transform $!^{i \leq n}((\widetilde{x_1} : \widetilde{T_1}) \rightarrow FP_1, \ldots, (\widetilde{x_m} : \widetilde{T_m}) \rightarrow FP_m, !^{i_1 \leq n_1} \ldots, \ldots, !^{i_{m'} \leq n_{m'}} \ldots)$ into $(!^{i \leq n}(\widetilde{x_1} : \widetilde{T_1}) \rightarrow FP_1, \ldots, !^{i \leq n}(\widetilde{x_m} : \widetilde{T_m}) \rightarrow FP_m, !^{i_1 \leq n'_1} \ldots, \ldots, !^{i_{m'} \leq n'_{m'}} \ldots)$ in order to eliminate situations that do not satisfy this requirement.)

H5. Replications in L (resp. R) must have pairwise distinct bounds n. (This strengthens the typing: the typing then guarantees that, if several variables are accessed with the same array indices, then these variables are defined under the same replication.)

H6. For all restrictions new $y : T$ that occur above a term M in L, y occurs in M. (This guarantees that, in Hypothesis H′3.1 below, $z_{jk}[M_{j1}, \dots, M_{jq_j}]$ is defined for all $j \leq l$ and $k \leq m_j$. With Hypothesis H4, this guarantees that index_j is well-defined in Hypothesis H′3.1.3 below.)

H7. Finds in R are of the form

$$\mathsf{find}\ (\bigoplus_{j=1}^{m} \widetilde{u_j} \leq \widetilde{n_j}\ \mathsf{suchthat\ defined}(z_{j1}[\widetilde{u_{j1}}], \dots, z_{jl_j}[\widetilde{u_{jl_j}}]) \wedge M_j\ \mathsf{then}\ FP_j)\ \mathsf{else}\ FP'$$

where the following conditions are satisfied:

- For all $1 \leq k \leq l_j$, $\widetilde{u_{jk}}$ is the concatenation of a prefix of the current replication indices (the same prefix for all k) and a non-empty prefix of $\widetilde{u_j}$.
- When $\widetilde{u_j}$ is non-empty, at least one $\widetilde{u_{jk}}$ for $1 \leq k \leq l_j$ is the concatenation of a prefix of the current replication indices with the whole sequence $\widetilde{u_j}$.
- When $l_j \neq 0$, there exists $k \in \{1, \dots, l_j\}$ such that for all $k' \neq k$, $z_{jk'}$ is defined syntactically above all definitions of z_{jk} and $\widetilde{u_{jk'}}$ is a prefix of $\widetilde{u_{jk}}$. (This implies that the same find cannot access variables defined in different functions under the same replication in R.)
- Finally, variables z_{jk} are not defined by a find in R. (Otherwise, the transformation would be considerably more complicated.)

Such equivalences $L \approx R$ are used by the prover by replacing a process Q_0 observationally equivalent to $C[\llbracket L \rrbracket]$ with a process Q_0' observationally equivalent to $C[\llbracket R \rrbracket]$, for some evaluation context C. We now give sufficient conditions for a process to be equivalent to $C[\llbracket L \rrbracket]$. These conditions essentially guarantee that all uses of certain secret variables of Q_0, in a set S, can be implemented by calling functions of L. These conditions are explained in more detail below.

We first define the function extract used in order to extract information from the left- or right-hand sides of the equivalence.

$$\begin{aligned}
&\mathrm{extract}((x_1 : T_1, \dots, x_l : T_l) \to M, ()) = \\
&\quad (x_1 : T_1, \dots, x_l : T_l) \to M \\
&\mathrm{extract}(!^{i \leq n} \mathsf{new}\ y_1 : T_1; \dots; \mathsf{new}\ y_l : T_l; (G_1, \dots, G_m), \\
&\quad (j_1, \dots, j_k)) = \\
&\quad (y_1 : T_1, \dots, y_l : T_l), \mathrm{extract}(G_{j_1}, (j_2, \dots, j_k)) \\
&\mathrm{extract}((G_1, \dots, G_m), (j_0, \dots, j_k)) = \\
&\quad \mathrm{extract}(G_{j_0}, (j_1, \dots, j_k))
\end{aligned}$$

We rename the variables of Q_0 such that variables of L and R do not occur in Q_0. Assume that there exist a set of variables S and a set $\mathcal{M}$ of occurrences of terms in Q_0 such that:

H′1. $S \cap V = \emptyset$.

H′2. No term in $\mathcal{M}$ occurs in the condition part of a find $(\mathsf{defined}(M_1, \dots, M_l) \wedge M)$ or in the channel of an input.

H′3. For each $M \in \mathcal{M}$, there exist a sequence $BL(M) = (j_0, \dots, j_l)$ such that $\mathrm{extract}(L, BL(M)) = (y_{11} : T_{11}, \dots, y_{1m_1} : T_{1m_1}), \dots, (y_{l1} : T_{l1}, \dots, y_{lm_l} : T_{lm_l}), (x_1 : T_1, \dots, x_m : T_m) \to N$ and a substitution σ such that $M = \sigma N$ (σ applies to the abbreviated form of N in which we write x instead of $x[\widetilde{i}]$) and the following conditions hold:

- H′3.1. For all $j \leq l$ and $k \leq m_j$, σy_{jk} is a variable access $z_{jk}[M_{j1}, \dots, M_{jq_j}]$, with $z_{jk} \in S$. We define $z_{jk} = \mathrm{varImL}(y_{jk}, M)$.
  - H′3.1.1. All definitions of z_{jk} in Q_0 are of the form new $z_{jk}[\dots] : T_{jk}$, and for all $k \leq m_j$, they occur under the same replications (but they may occur under different replications for different values of j).
  - H′3.1.2. When $j \neq j'$ or $k \neq k'$, $z_{jk} \neq z_{j'k'}$.
  - H′3.1.3. The sequence of array indices $M_{j1}, \dots, M_{jq_j}$ is the same for all $k \leq m_j$ (but may depend on j). We denote by $\mathrm{index}_j(M)$ a substitution that maps the current replication indices at the definition of z_{jk} to $M_{j1}, \dots, M_{jq_j}$ respectively. If $m_l = 0$, $\mathrm{index}_l(M)$ is not set by the previous definition, so we set $\mathrm{index}_l(M)$ to map the current replication indices at M to themselves. For each $j < l$, there exists a substitution $\rho_j(M)$ such that $\mathrm{index}_j(M) = \mathrm{index}_{j+1}(M) \circ \rho_j(M)$ and the image of $\rho_j(M)$ does not contain the current replication indices at M. We denote by $\mathrm{im}\ \mathrm{index}_j(M)$ the sequence image by $\mathrm{index}_j(M)$ of the sequence of current replication indices at the definition of z_{jk} (so, $\mathrm{im}\ \mathrm{index}_j(M) = (M_{j1}, \dots, M_{jq_j})$). We define $\mathrm{im}\ \rho_j(M)$ similarly.
- H′3.2. For all $j \leq m$, σx_j is a term of type T_j.
- H′3.3. All occurrences in Q_0 of a variable in S are either as z_{jk} above or at the root of an argument of a defined test in a find process.

To make it precise which term M each element refers to, we add M as a subscript, writing $y_{jk,M}$ for y_{jk}, $z_{jk,M}$ for z_{jk}, $T_{jk,M}$ for T_{jk}, $x_{j,M}$ for x_j, $T_{j,M}$ for T_j, N_M for N, and σ_M for σ. We also define $\mathrm{nNew}_{j,M} = m_j$, $\mathrm{nNewSeq}_M = l$, and $\mathrm{nInput}_M = m$.

H′4. We say that two terms $M, M' \in \mathcal{M}$ share the first l' sequences of random variables when $y_{jk,M} = y_{jk,M'}$ and $z_{jk,M} = z_{jk,M'}$ for all $j \leq l'$ and $k \leq \mathrm{nNew}_{j,M} = \mathrm{nNew}_{j,M'} \neq 0$. Let l' be the greatest integer such that M and M' share the first l' sequences of random variables. Then we require:

- H′4.1. The sets of variables $\{z_{jk,M} \mid j > l' \text{ and } k \leq \mathrm{nNew}_{j,M}\}$ and $\{z_{jk,M'} \mid j > l' \text{ and } k \leq \mathrm{nNew}_{j,M'}\}$ must be disjoint.

H′4.2. $\rho_j(M) = \rho_j(M')$ for all $j < l'$.

H′4.3. If $l' = \mathrm{nNewSeq}_M$ and $N_M = N_{M'}$, then there exists M_0 such that $M = (\mathrm{index}_{l'}(M))M_0$, $M' = (\mathrm{index}_{l'}(M'))M_0$, and M_0 does not contain the current replication indices at M or M'.

When these conditions are satisfied, there exists a context C such that $Q_0 \approx_0^V C[\![L]\!]$.

Terms in $\mathcal{M}$ must not occur in conditions of find (Hypothesis H′2) because such terms may refer to variables defined by find, and by the transformation, these variables might be moved outside their scope, thus violating Invariant 2. Terms in $\mathcal{M}$ must not occur in the channel of an input, because otherwise, after the transformation, the input process might need to perform computations by find or let, forbidden by the syntax. (This requirement is not a limitation in practice, since terms in channels of inputs are typically the current replication indices, so they do not contain cryptographic primitives.)

In Hypothesis H′3, the sequence $BL(M)$ indicates which branch of L corresponds to the term M.

Hypothesis H′3.2 checks that the values received by inputs in L are of the proper type. Hypothesis H′3.1.1 checks that variables $z_{jk,M}$ that correspond to variables defined by new in L are of the proper type. The variables y_{jk} defined by new in L are used only in terms N in L. Correspondingly, Hypothesis H′3.3 checks that the corresponding variables $z_{jk,M} \in S$ are not used elsewhere in Q_0 and Hypothesis H′1 checks that they cannot be used directly by the context.

In L, for distinct j, k, the variables y_{jk} correspond to independent random numbers. Correspondingly, Hypothesis H′3.1.2 requires that the variables $z_{jk,M}$ are created by different restrictions for distinct j, k. In L, the variables y_{jk} are accessed with the same indices for any k (but a fixed j). Correspondingly, Hypothesis H′3.1.3 requires that the variables $z_{jk,M}$ are accessed with the same indices im $\mathrm{index}_j(M)$ for any k. When instances of N and N' both refer to y_{jk} with the same indices, then they also refer to $y_{j'k'}$ with the same indices when $j' \leq j$. Correspondingly, if M and M' refer to the same z_{jk}, by Hypothesis H′4.1, they also refer to the same $z_{j'k'}$ for $j' \leq j$. Moreover, if $\mathrm{index}_j(M)$ and $\mathrm{index}_j(M')$ evaluate to the same bitstrings, then $\mathrm{index}_{j'}(M)$ and $\mathrm{index}_{j'}(M')$ also evaluate to the same bitstrings, since $\mathrm{index}_{j'}(M) = \mathrm{index}_j(M) \circ \rho_{j-1}(M) \circ \ldots \circ \rho_{j'}(M)$ by Hypothesis H′3.1.3 and $\rho_k(M) = \rho_k(M')$ for $k < j$ by Hypothesis H′4.2. These conditions guarantee that we can establish a correspondence from the array cells of variables of S in Q_0 to the array cells of variables defined by new in L, and that this correspondence is an injective function, as required in Section 3.2.

Finally, a term N in L is evaluated at most once for each value of the indices of $y_{l1}, \ldots, y_{lm_l}$, so N is computed for a single value of the arguments $x_1, \ldots, x_m$. Correspondingly, by Hypothesis H′4.3, when M and M' share the $l = \mathrm{nNewSeq}_M$ sequences of random variables and $\mathrm{index}_l(M)$ and $\mathrm{index}_l(M')$ evaluate to the same bitstring, then M and M' evaluate to the same bitstring.

We compute the possible values of the sets S and $\mathcal{M}$ by fixpoint iteration. We start with $\mathcal{M} = \emptyset$ and S containing a single variable of Q_0 bound by a restriction. (We try all possible variables.) When a term M of Q_0 contains a variable in S, we try to find a function in L that corresponds to M, and if we succeed, we add M to $\mathcal{M}$, and add to S variables in M that correspond to variables bound by restrictions in L. (If we fail, the transformation is not possible.) We continue until a fixpoint is reached, in which case all occurrences of variables of S are in terms of $\mathcal{M}$.

We now describe how we construct a process Q_0' such that $Q_0' \approx_0^V C[\![R]\!]$.

1. We first move restrictions in the right-hand side of the equivalence, so that they occur above the reception of the arguments of functional processes instead of inside functional processes. As explained below, this is necessary for the correctness of the subsequent transformation of Q_0, when restrictions appear in the corresponding part of the left-hand side. More precisely, we transform the right-hand side of the equivalence, R, as follows: for each $j_1, \ldots, j_l$, if $\mathrm{extract}(L, (j_1, \ldots, j_l)) = (y_{11} : T_{11}, \ldots, y_{1m_1} : T_{1m_1}), \ldots, (y_{l1} : T_{l1}, \ldots, y_{lm_l} : T_{lm_l}), (x_1 : T_1, \ldots, x_m : T_m) \to N$ with $m_l \neq 0$ and $\mathrm{extract}(R, (j_1, \ldots, j_l)) = (y'_{11} : T'_{11}, \ldots, y'_{1m'_1} : T'_{1m'_1}), \ldots, (y'_{l1} : T'_{l1}, \ldots, y'_{lm'_l} : T'_{lm'_l}), (x_1 : T_1, \ldots, x_m : T_m) \to FP$, for each new $z : T$ in FP,
   - we add $z : T$ in the sequence of random variables $y'_{l1} : T'_{l1}, \ldots, y'_{lm'_l} : T'_{lm'_l}$;
   - if z does not occur in defined conditions of find in R, we remove new $z : T$ from FP;
   - otherwise, we replace new $z : T$ with let $z' : T = cst$ for some constant cst and add $z'[\widetilde{M}]$ to each defined condition of R that contains $z[\widetilde{M}]$.

   This transformation is needed, because in the right-hand side, a new random number must be chosen exactly for each different call to the function $(x_1 : T_1, \ldots, x_m : T_m) \to FP$. This would not be guaranteed without that transformation, because when the left-hand side N is evaluated at several occurrences with the same random numbers $y_{l1} : T_{l1}, \ldots, y_{lm_l} : T_{lm_l}$ ($m_l \neq 0$), these occurrences all correspond to a single call to $(x_1 : T_1, \ldots, x_m : T_m) \to N$, so a single call to $(x_1 : T_1, \ldots, x_m : T_m) \to FP$, but we create a copy of FP for each occurrence. After the transformation, FP contains no choice of random numbers, so we can evaluate it several times without changing the result. When $m_l = 0$, evaluations of N at several occurrences can correspond to different calls to $(x_1 : T_1, \ldots, x_m : T_m) \to N$, so the transformation is not necessary.

2. Next, we create fresh variables corresponding to variables of the right-hand side of the equivalence. For each $M \in \mathcal{M}$, let $\mathrm{extract}(R, BL(M)) = (y'_{11,M} : T'_{11,M}, \ldots, y'_{1m'_1,M} : T'_{1m'_1,M}), \ldots, (y'_{l1,M} : T'_{l1,M}, \ldots, y'_{lm'_l,M} : T'_{lm'_l,M}), (x_{1,M} : T_{1,M}, \ldots, x_{m,M} : T_{m,M}) \to FP_M$ with $l = \mathrm{nNewSeq}_M$, $m = \mathrm{nInput}_M$ and we define $\mathrm{nNew}'_{j,M} = m'_j$. We create fresh variables $z'_{jk,M} = \mathrm{varImR}(y'_{jk,M}, M)$ for each $j \leq \mathrm{nNewSeq}_M$, $k \leq \mathrm{nNew}'_{j,M}$, and $M \in \mathcal{M}$, such that if M and M' share the first l' sequences of random variables, then $z'_{jk,M} = z'_{jk,M'}$

for $j \le l'$ and $k \le \mathrm{nNew}'_{j,M}$. All variables $z'_{jk,M}$ are otherwise pairwise distinct.

We also create a fresh variable $\mathrm{varImR}(x_{j,M}, M)$ for each $j \le \mathrm{nInput}_M$ and each $M \in \mathcal{M}$, and a fresh variable $\mathrm{varImR}(z, M)$ for each variable z defined by let or new in FP_M and each $M \in \mathcal{M}$.

3. We update the defined conditions of finds, in order to preserve Invariant 2. More precisely, if a defined condition of a find contains $z_{j1,M}[M_1, \dots, M_{l'}]$ for some M, we add $\mathsf{defined}(z'_{jk',M}[M_1, \dots, M_{l'}])$ for all $k' \le \mathrm{nNew}'_{j,M}$ to this condition. (So that accesses to $z'_{jk',M}[M_1, \dots, M_{l'}]$ created when transforming term M satisfy Invariant 2, since accesses to $z_{j1,M}[M_1, \dots, M_{l'}]$ occur in M and satisfy Invariant 2.)

4. We update restrictions corresponding to restrictions of the left-hand side of the equivalence: we either remove them or replace them with restrictions corresponding to the right-hand side of the equivalence. More precisely, when $x \in S$ occurs at the root of a term M_k in a condition $\mathsf{defined}(M_1, \dots, M_l)$, we replace its definition $\mathsf{new}\ x : T; Q$ with $\mathsf{let}\ x : T = cst\ \mathsf{in}\ Q$ for some constant cst; when it does not occur in defined tests, we remove its definition. If $x = z_{j1,M}$ for some M, we add $\mathsf{new}\ z'_{jk,M} : T'_{jk,M}$ for each $k \le \mathrm{nNew}'_{j,M}$ where $\mathsf{new}\ x : T$ was.

5. Finally, we transform the terms $M \in \mathcal{M}$ corresponding to functions of the left-hand side of the equivalence into their corresponding functional process in the right-hand side. For each term $M \in \mathcal{M}$, let $P_M = C_M[M]$ be the smallest process containing M. (Note that M never occurs in an input, so P_M is an output process.) Let $l = \mathrm{nNewSeq}_M$. We replace P_M with $(\mathsf{new}\ z'_{lk,M} : T'_{lk,M};)_{k \le \mathrm{nNew}'_{l,M}} P'_M$ if $\mathrm{nNew}_{l,M} = 0$ and $\mathrm{nNew}'_{l,M} > 0$, and with P'_M otherwise, where

   – $P'_M = (\mathsf{let}\ \mathrm{varImR}(x_{k,M}, M) : T_{k,M} = \sigma_M x_{k,M}\ \mathsf{in})_{k \le \mathrm{nInput}_M} \mathrm{transf}_{\phi_0, C_M}(FP_M)$.

   – ϕ_0 is defined as follows:

$$\phi_0(x_{j,M}[i_1, \dots, i_l]) = \mathrm{varImR}(x_{j,M}, M)[i'_1, \dots, i'_{l'}]$$
$$\phi_0(z[i_1, \dots, i_l]) = \mathrm{varImR}(z, M)[i'_1, \dots, i'_{l'}]$$
$$\phi_0(y'_{jk,M}[i_1, \dots, i_j]) = \mathrm{varImR}(y'_{jk,M}, M)[\mathrm{im\ index}_j(M)]$$

   where $i_1, \dots, i_l$ are the current replication indices at the definition of $x_{j,M}$ in R, $i'_1, \dots, i'_{l'}$ are the current replication indices at M in Q_0, and z is a variable defined by let or new in FP_M.

   – A function ϕ from array accesses to array accesses is extended to terms as a substitution, by $\phi(f(M_1, \dots, M_m)) = f(\phi(M_1), \dots, \phi(M_m))$.

   – $\mathrm{transf}_{\phi, C_M}(FP)$ is defined recursively as follows:

$$\mathrm{transf}_{\phi, C_M}(M') = C_M[\phi(M')]$$
$$\mathrm{transf}_{\phi, C_M}(\mathsf{new}\ z : T; FP') = \mathsf{new}\ \mathrm{varImR}(z, M) : T; \mathrm{transf}_{\phi, C_M}(FP')$$
$$\mathrm{transf}_{\phi, C_M}(\mathsf{let}\ z : T = M'\ \mathsf{in}\ FP') = \mathsf{let}\ \mathrm{varImR}(z, M) : T = \phi(M')\ \mathsf{in}\ \mathrm{transf}_{\phi, C_M}(FP')$$
$$\mathrm{transf}_{\phi, C_M}(\mathsf{find}(\textstyle\bigoplus_{j=1}^{m} FB_j)\ \mathsf{else}\ FP') = \mathsf{find}(\textstyle\bigoplus_{j=1}^{m} \mathrm{transf}_{\phi, C_M}(FB_j))\ \mathsf{else}\ \mathrm{transf}_{\phi, C_M}(FP')$$

   and for find branches FB, $\mathrm{transf}_{\phi, C_M}(FB)$ is defined as follows:

$$\mathrm{transf}_{\phi, C_M}(\mathsf{suchthat}\ M'\ \mathsf{then}\ FP') = \mathsf{suchthat}\ \phi(M')\ \mathsf{then}\ \mathrm{transf}_{\phi, C_M}(FP')$$
$$\begin{aligned}&\mathrm{transf}_{\phi, C_M}(\widetilde{u} \le \widetilde{n}\ \mathsf{suchthat}\\ &\quad \mathsf{defined}(z_k[M_{k1}, \dots, M_{kl'_k}]_{1 \le k \le l}) \wedge M_1\ \mathsf{then}\ FP') =\\ &\quad \textstyle\bigoplus_{M' \in \mathcal{M}'} \widetilde{u'} \le \widetilde{n'}\ \mathsf{suchthat}\\ &\qquad \mathsf{defined}(\phi'(z_k[M_{k1}, \dots, M_{kl'_k}])_{1 \le k \le l}) \wedge\\ &\qquad \mathrm{im\ index}_{j_1}(M')\{\widetilde{u'}/\widetilde{i'}\} = \mathrm{im\ index}_{j_1}(M) \wedge\\ &\qquad \phi'(M_1)\ \mathsf{then}\ \mathrm{transf}_{\phi', C_M}(FP')\end{aligned}$$

   where $l \neq 0$; j_1 is the length of the prefix of the current replication indices that occurs in $M_{k1}, \dots, M_{kl'_k}$ (by Hypothesis H7); $\mathcal{M}'$ is the set of $M' \in \mathcal{M}$ such that $\mathrm{varImR}(z_k, M')$ is defined for $k \le l$ and M' and M share the first j_1 sequences of random variables; $\widetilde{i'}$ is the sequence of current replication indices at M'; $\widetilde{u'}$ is a sequence formed with a fresh variable for each variable in $\widetilde{i'}$; $\widetilde{n'}$ is the sequence of bounds of replications above M'; ϕ' is an extension of ϕ with $\phi'(z_k[M_{k1}, \dots, M_{kl'_k}]) = \mathrm{varImR}(z_k, M')[\mathrm{im\ index}_j(M')\{\widetilde{u'}/\widetilde{i'}\}]$ if $z_k = y'_{jk',M'}$ for some k', and $\phi'(z_k[M_{k1}, \dots, M_{kl'_k}]) = \mathrm{varImR}(z_k, M')[\widetilde{u'}]$ if z_k is defined by let or by a function input. Optimizations for the definition of $\mathrm{transf}_{\phi, C_M}(FB)$ are presented in Appendix D.2.1.

The two essential parts of the transformation are the last two ones, numbered 4 and 5. In step 4, we add the restrictions to create random variables that correspond to random variables of R. We create the variables $z'_{jk,M}$ at the place where $z_{j1,M}$ was created in the initial game (We could have chosen $z_{jk',M}$ for any k'.), or when there is no $z_{j1,M}$, we have $j = \mathrm{nNewSeq}_M$ and we create $z'_{jk,M}$ just before evaluating M. In step 5, we transform the term M itself into the corresponding functional process of R, FP_M. The only delicate part for evaluating FP_M is the case of find: instead of looking up arrays of R, we look up the corresponding arrays of Q'_0 given by the mapping ϕ.

D.2 Extensions

D.2.1 Optimizations for $\mathrm{transf}_{\phi, C_M}(FB)$

We can apply two optimizations to the definition of $\mathrm{transf}_{\phi, C_M}(FB)$:

- When $\mathrm{im\ index}_{j_1}(M')$ is a prefix of $\widetilde{i'}$, $\mathrm{im\ index}_{j_1}(M')\{\widetilde{u'}/\widetilde{i'}\}$ is a prefix of $\widetilde{u'}$, so the equality $\mathrm{im\ index}_{j_1}(M')\{\widetilde{u'}/\widetilde{i'}\} = \mathrm{im\ index}_{j_1}(M)$ defines the value of a prefix of $\widetilde{u'}$. We simply substitute the fixed elements of $\widetilde{u'}$ with their value, and remove them from the sequence of variables to be looked up by find.

- When all variables z_k are $y_{jk',M'}$ for some j, k', and M', with $\max j = j_0$, we use the following definition instead:

$$\begin{aligned}&\mathrm{transf}_{\phi,C_M}(\widetilde{i} \leq \widetilde{n}\ \mathsf{suchthat}\\ &\quad \mathsf{defined}(z_k[M_{k1},\dots,M_{kl'_k}]_{1\leq k\leq l}) \wedge M_1\ \mathsf{then}\ FP') =\\ &\bigoplus_{M'\in\mathcal{M}'} \widetilde{u'} \leq \widetilde{n'}\ \mathsf{suchthat}\\ &\quad \mathsf{defined}(\phi'(z_k[M_{k1},\dots,M_{kl'_k}])_{1\leq k\leq l})\wedge\\ &\quad \mathrm{im}\ (\rho_{j_0-1}(M')\circ\ldots\circ\rho_{j_1}(M'))\{\widetilde{u'}/\widetilde{i'}\} =\\ &\quad \mathrm{im}\ \mathrm{index}_{j_1}(M)\wedge\phi'(M_1)\ \mathsf{then}\ \mathrm{transf}_{\phi',C_M}(FP')\end{aligned}$$

  where j_1 is the length of the prefix of the current replication indices that occurs in $M_{k1},\dots,M_{kl'_k}$ (by Hypothesis H7); $\mathcal{M}'$ is the set of $M' \in \mathcal{M}$ such that $\mathrm{varImR}(z_k, M')$ is defined for $k \leq l$ and M' and M share the j_1 first sequences of random variables; $\widetilde{i'}$ is the sequence of current replication indices at the definition of $z_{j_0k,M'}$; $\widetilde{u'}$ is a sequence formed with a fresh variable for each variable in $\widetilde{i'}$; $\widetilde{n'}$ is the sequence of bounds of replications above the definition of $z_{j_0k,M'}$; ϕ' is an extension of ϕ with $\phi'(z_k[M_{k1},\dots,M_{kl'_k}]) = \mathrm{varImR}(z_k, M')[\mathrm{im}\ (\rho_{j_0-1}(M')\circ\ldots\circ\rho_j(M'))\{\widetilde{u'}/\widetilde{i'}\}]$ if $z_k = y'_{jk,M'}$.

  The composition $\rho_{j_0-1}(M')\circ\ldots\circ\rho_j(M')$ computes the indices of $z'_{jk',M'}$ for any k' from the indices of $z'_{j_0k'',M'}$ for any k''.

  When several terms $M' \in \mathcal{M}$ share the first j_0 sequences of random variables, they generate the same ϕ', so only one find branch needs to be added for all of them, which can reduce considerably the number of find branches to add.

  An optimization similar to the first one above also applies to this case, when $\mathrm{im}\ (\rho_{j_0-1}(M')\circ\ldots\circ\rho_{j_1}(M'))$ is a prefix of $\widetilde{i'}$.

D.2.2 Guiding the Application of Equivalences

We introduce a small extension to the equivalences $(G_1,\dots, G_m) \approx (G'_1,\dots,G'_m)$ described in Section 3.2. These equivalences become $(G_1\ mode_1,\dots,G_m\ mode_m) \approx (G'_1,\dots,G'_m)$, where $mode_j$ is either empty or $[all]$. The mode $[all]$ is an indication for the prover, to guide the application of the equivalence without changing its semantics. When $mode_j = [all]$, $\mathcal{M}$ must contain all occurrences in the initial game Q of the root function symbols of terms M inside G_j. When $mode_j$ is empty, at least one variable defined by new in G_j must correspond to a variable in S.

The following hypotheses guarantee the good usage of modes:

H8. At most one $mode_j$ can be empty. (Otherwise, when several sets of random variables can be chosen for each G_j, there are many possible combinations for applying the transformation.)

H9. If G_j is of the form $!^{i\leq n}(x_1 : T_1,\dots,x_l : T_l) \to FP$ without any restriction, then $mode_j = [all]$. (A restriction is needed in the definition of empty mode.)

An equivalence can be declared $[manual]$. In this case, this equivalence is not applied by the automatic proof strategy. It can be used only by manual proof indications. This is useful, for instance, if applying the equivalence would yield an infinite loop.

Each function in the left-hand side can also be labeled with an integer $[n]$, which represents a priority: CryptoVerif preferably uses functions labeled with a lower integer. (The absence of label corresponds to the label [0].) Each function in the left-hand side can finally be labeled with $[useful\_change]$. This indication is also used for the proof strategy: if at least one $[useful\_change]$ indication is present, CryptoVerif applies the transformation defined by the equivalence only when at least one $[useful\_change]$ function is called in the game.

D.2.3 Relaxing Hypothesis H6

Hypothesis H6 requires that for all restrictions new $y : T$ that occur above a term N in the left-hand side of an equivalence, y occurs in N. We can relax this hypothesis, by allowing that some random variables y do not occur in N, provided that the missing variables can be determined using Hypothesis H′4.1: when some term M shares some variable y in the l'-th sequence of random variables with some other term M', we know that it must also share with M' all random variables in sequences above and including the l'-th sequence; so, knowing the random variables associated to M', we can determine some of those associated to M. The transformation simply fails when the algorithm described above cannot fully determine the random variables associated to some term M.

With this extension, we additionally need to make sure that, when an expression of the right-hand side uses a variable y defined by a restriction, this variable will be defined in the transformed game before the expression is evaluated. To do that, we establish a correspondence between the restrictions before the transformation and the restrictions after, such that, as far as possible, if a variable is used in a transformed expression, the corresponding variable before transformation is also used in the initial expression. For variables that appear in a transformed expression but are not used in the initial expression, we check when performing the game transformation that they will correctly be defined. If they are not correctly defined, the transformation fails.

D.2.4 Relaxing Hypothesis H′2

Hypothesis H′2 requires that no term N transformed by the equivalence occurs in the condition part of a find ($\mathsf{defined}(M_1,\dots,M_l) \wedge M$). We can relax this hypothesis by allowing N to occur in M (but not in the defined test), provided

- the variables $\widetilde{u}$ bound by this find do not occur in the following terms in the transformed expression of N:
  - N' in processes of the form let $x : T = N'$ in $\ldots$;
  - N'_{jk} and N'_j in processes of the form find $(\bigoplus_{j=1}^m u_{j1}[\widetilde{i}] \leq n_{j1},\dots,u_{jm_j}[\widetilde{i}] \leq n_{jm_j}$ suchthat $\mathsf{defined}(N'_{j1},\dots,N'_{jl_j}) \wedge N'_j$ then $\ldots)$ else $\ldots$.

(If the variables $\widetilde{u}$ bound by find occurred in such terms, the transformation would move them outside the scope of their definition.)

- if N depends on the variables $\widetilde{u}$ bound by this find, then the corresponding left-hand side of the equivalence N' is under a replication $!^{i \leq n}$ with no restriction in L. (Otherwise, L contains $\mathsf{new}\ y_1 : T_1; \ldots; \mathsf{new}\ y_l : T_l; (\ldots, (x_1 : T'_1, \ldots, x_{l'} : T'_{l'}) \rightarrow N', \ldots)$ and L allows a single evaluation of N' for each execution of the restrictions $\mathsf{new}\ y_1 : T_1; \ldots; \mathsf{new}\ y_l : T_l$, while the term N is evaluated several times, once for each value of $\widetilde{u}$, so the transformation is impossible. In other words, the variables $\widetilde{u}$ must be considered as replication indices in this transformation.)

D.3 Modeling other Primitives

This appendix gives the definition of a number of cryptographic primitives in our prover.

D.3.1 Shared-key Encryption

IND-CPA and INT-CTXT Encryption

T_r large, fixed length; T'_r fixed length
$\mathrm{enc} : \mathit{bitstring} \times T_k \times T'_r \rightarrow T_e$
$\mathrm{dec} : T_e \times T_k \rightarrow \mathit{bitstring}_\perp$
$\mathrm{kgen} : T_r \rightarrow T_k$
$Z : \mathit{bitstring} \rightarrow \mathit{bitstring}$
$\mathrm{i}_\perp : \mathit{bitstring} \rightarrow \mathit{bitstring}_\perp$

$\forall m : T, \forall r : T_r, \forall r' : T'_r,$
$\quad \mathrm{dec}(\mathrm{enc}(m, \mathrm{kgen}(r), r'), \mathrm{kgen}(r)) = \mathrm{i}_\perp(m)$

$!^{i_k \leq n_k} \mathsf{new}\ r : T_r;$
$\quad !^{i_e \leq n_e}(x : \mathit{bitstring}) \rightarrow \mathsf{new}\ r' : T'_r; \mathrm{enc}(x, \mathrm{kgen}(r), r')$
$\approx$
$!^{i_k \leq n_k} \mathsf{new}\ r : T_r;$
$\quad !^{i_e \leq n_e}(x : \mathit{bitstring}) \rightarrow \mathsf{new}\ r' : T'_r; \mathrm{enc}'(\mathrm{Z}(x), \mathrm{kgen}'(r), r')$

$!^{i_k \leq n_k} \mathsf{new}\ r : T_r; ($
$\quad !^{i_e \leq n_e} \mathsf{new}\ r' : T'_r; (x : \mathit{bitstring}) \rightarrow \mathrm{enc}(x, \mathrm{kgen}(r), r'),$
$\quad !^{i_d \leq n_d}(y : T_e) \rightarrow \mathrm{dec}(y, \mathrm{kgen}(r)))$
$\approx$
$!^{i_k \leq n_k} \mathsf{new}\ r : T_r; ($
$\quad !^{i_e \leq n_e} \mathsf{new}\ r' : T'_r; (x : \mathit{bitstring}) \rightarrow$
$\qquad \mathsf{let}\ z : T_e = \mathrm{enc}(x, \mathrm{kgen}'(r), r')\ \mathsf{in}\ z,$
$\quad !^{i_d \leq n_d}(y : T_e) \rightarrow$
$\qquad \mathsf{find}\ u \leq n_e\ \mathsf{suchthat}\ \mathsf{defined}(x[u], r'[u], z[u]) \wedge z[u] = y$
$\qquad \mathsf{then}\ \mathrm{i}_\perp(x[u])\ \mathsf{else}\ \perp).$

The first equivalence represents the IND-CPA property, as explained in the body of the paper. The second one represents the INT-CTXT property (ciphertext integrity). This property means that, up to negligible probability, decryption succeeds only if the given ciphertext y has been generated by the encryption oracle ($y = z[u]$ for some $u \leq n_e$). In this case, decryption returns the corresponding cleartext $x[u]$; otherwise, it fails by returning the special symbol $\perp$.

IND-CCA2 and INT-PTXT Encryption

T_r large, fixed length; T'_r fixed length
$\mathrm{enc} : \mathit{bitstring} \times T_k \times T'_r \rightarrow T_e$
$\mathrm{dec} : T_e \times T_k \rightarrow \mathit{bitstring}_\perp$
$\mathrm{kgen} : T_r \rightarrow T_k$
$Z : \mathit{bitstring} \rightarrow \mathit{bitstring}$
$\mathrm{i}_\perp : \mathit{bitstring} \rightarrow \mathit{bitstring}_\perp$

$\forall m : T, \forall r : T_r, \forall r' : T'_r,$
$\quad \mathrm{dec}(\mathrm{enc}(m, \mathrm{kgen}(r), r'), \mathrm{kgen}(r)) = \mathrm{i}_\perp(m)$

$!^{i_k \leq n_k} \mathsf{new}\ r : T_r; ($
$\quad !^{i_e \leq n_e} \mathsf{new}\ r' : T'_r; (x : \mathit{bitstring}) \rightarrow \mathrm{enc}(x, \mathrm{kgen}(r), r'),$
$\quad !^{i_d \leq n_d}(y : T_e) \rightarrow \mathrm{dec}'(y, \mathrm{kgen}(r)))$
$\approx$
$!^{i_k \leq n_k} \mathsf{new}\ r : T_r; ($
$\quad !^{i_e \leq n_e} \mathsf{new}\ r' : T'_r; (x : \mathit{bitstring}) \rightarrow$
$\qquad \mathsf{let}\ z : T_e = \mathrm{enc}'(\mathrm{Z}(x), \mathrm{kgen}'(r), r')\ \mathsf{in}\ z,$
$\quad !^{i_d \leq n_d}(y : T_e) \rightarrow$
$\qquad \mathsf{find}\ u \leq n_e\ \mathsf{suchthat}\ \mathsf{defined}(x[u], r'[u], z[u]) \wedge$
$\qquad\quad y = z[u]\ \mathsf{then}\ \mathrm{i}_\perp(x[u])\ \mathsf{else}\ \mathrm{dec}'(y, \mathrm{kgen}'(r)))$

$!^{i_k \leq n_k} \mathsf{new}\ r : T_r; ($
$\quad !^{i_e \leq n_e} \mathsf{new}\ r' : T'_r; (x : \mathit{bitstring}) \rightarrow \mathrm{enc}(x, \mathrm{kgen}(r), r'),$
$\quad !^{i_d \leq n_d}(y : T_e) \rightarrow \mathrm{dec}(y, \mathrm{kgen}(r)))$
$\approx$
$!^{i_k \leq n_k} \mathsf{new}\ r : T_r; ($
$\quad !^{i_e \leq n_e} \mathsf{new}\ r' : T'_r; (x : \mathit{bitstring}) \rightarrow \mathrm{enc}(x, \mathrm{kgen}(r), r'),$
$\quad !^{i_d \leq n_d}(y : T_e) \rightarrow$
$\qquad \mathsf{let}\ z : T = \mathrm{dec}'(y, \mathrm{kgen}(r))\ \mathsf{in}$
$\qquad \mathsf{find}\ u \leq n_e\ \mathsf{suchthat}\ \mathsf{defined}(x[u]) \wedge z = \mathrm{i}_\perp(x[u])$
$\qquad \mathsf{then}\ \mathrm{i}_\perp(x[u])\ \mathsf{else}\ \perp).$

The first equivalence represents the IND-CCA2 property. It replaces encryptions with encryptions of zeroes. For decryption, if the ciphertext received as argument has been produced by the encryption oracle, it returns the corresponding cleartext. Otherwise, it decrypts normally. The decryption oracle uses the symbol dec' in the left-hand side, because the IND-CCA2 property will be applied after the INT-PTXT property, which transforms dec into dec'.

The second equivalence represents the INT-PTXT property (plaintext integrity). It leaves encryptions unchanged. Up to negligible probability, decryption succeeds only when the plaintext obtained by decryption z has been given to the encryption oracle. Otherwise, decryption fails by returning the special symbol $\perp$.

Super-Pseudo-Random Permutations (SPRP)

T_r large, fixed length; T large, fixed length
$\mathrm{enc}, \mathrm{dec} : T \times T_k \to T$
$\mathrm{kgen} : T_r \to T_k$

$\forall m : T, \forall r : T_r, \mathrm{dec}(\mathrm{enc}(m, \mathrm{kgen}(r)), \mathrm{kgen}(r)) = m$
$\forall m : T, \forall r : T_r, \mathrm{enc}(\mathrm{dec}(m, \mathrm{kgen}(r)), \mathrm{kgen}(r)) = m$

$!^{i_k \le n_k} \mathsf{new}\ r : T_r; ($
$\quad !^{i_e \le n_e}(me : T) \to \mathrm{enc}(me, \mathrm{kgen}(r)),$
$\quad !^{i_d \le n_d}(md : T) \to \mathrm{dec}(md, \mathrm{kgen}(r)))$
$\approx$
$!^{i_k \le n_k} \mathsf{new}\ r : T_r; ($
$\quad !^{i_e \le n_e}(me : T) \to$
$\quad\quad \mathsf{find}\ u \le n_e\ \mathsf{suchthat}\ \mathsf{defined}(me[u], re[u]) \wedge$
$\quad\quad\quad me = me[u]\ \mathsf{then}\ re[u]$
$\quad\quad \oplus\ u \le n_d\ \mathsf{suchthat}\ \mathsf{defined}(md[u], rd[u]) \wedge$
$\quad\quad\quad me = rd[u]\ \mathsf{then}\ md[u]$
$\quad\quad \mathsf{else\ new}\ re : T; re,$
$\quad !^{i_d \le n_d}(md : T) \to$
$\quad\quad \mathsf{find}\ u \le n_e\ \mathsf{suchthat}\ \mathsf{defined}(me[u], re[u]) \wedge$
$\quad\quad\quad md = re[u]\ \mathsf{then}\ me[u]$
$\quad\quad \oplus\ u \le n_d\ \mathsf{suchthat}\ \mathsf{defined}(md[u], rd[u]) \wedge$
$\quad\quad\quad md = md[u]\ \mathsf{then}\ rd[u]$
$\quad\quad \mathsf{else\ new}\ rd : T; rd)$

This equivalence expresses that the encryption and decryption oracles can be replaced with inverse random permutations. These random permutations are built as follows for the encryption oracle: when we receive an argument me already passed to the encryption oracle, we return the previous result; when we receive the result of a previous call to the decryption oracle, we return the argument of the decryption oracle in that call; otherwise, we return a fresh random number. (Collisions between random numbers in T_r have negligible probability, so we obtain permutations except in cases of negligible probability.) The construction is similar for the decryption oracle.

Pseudo-Random Permutations (PRP)

T_r large, fixed length; T large, fixed length
$\mathrm{enc}, \mathrm{dec} : T \times T_k \to T$
$\mathrm{kgen} : T_r \to T_k$

$\forall m : T, \forall r : T_r, \mathrm{dec}(\mathrm{enc}(m, \mathrm{kgen}(r)), \mathrm{kgen}(r)) = m$

$!^{i_k \le n_k} \mathsf{new}\ r : T_r;$
$\quad !^{i_e \le n_e}(x : T) \to \mathrm{enc}(x, \mathrm{kgen}(r))$
$\approx$
$!^{i_k \le n_k} \mathsf{new}\ r : T_r;$
$\quad !^{i_e \le n_e}(me : T) \to$
$\quad\quad \mathsf{find}\ u \le n_e\ \mathsf{suchthat}\ \mathsf{defined}(me[u], re[u]) \wedge$
$\quad\quad\quad me = me[u]\ \mathsf{then}\ re[u]$
$\quad\quad \mathsf{else\ new}\ re : T; re$

This model is obtained from SPRP by removing the decryption oracle. It means that encryption is a random permutation. (We eliminate collisions between random elements re, which have negligible probability.)

D.3.2 Public-Key Cryptography

UF-CMA Signature

T_r large, fixed length; T'_r fixed length
$\mathrm{sign}, \mathrm{sign}' : T \times T_{sk} \times T'_r \to T_s$
$\mathrm{check}, \mathrm{check}' : T \times T_{pk} \times T_s \to \mathit{bool}$
$\mathrm{skgen}, \mathrm{skgen}' : T_r \to T_{sk}$
$\mathrm{pkgen}, \mathrm{pkgen}' : T_r \to T_{pk}$

$\forall m : T, \forall r : T_r, \forall r' : T'_r,$
$\quad \mathrm{check}(m, \mathrm{pkgen}(r), \mathrm{sign}(m, \mathrm{skgen}(r), r')) = \mathrm{true}$
$\forall m : T, \forall r : T_r, \forall r' : T'_r,$
$\quad \mathrm{check}'(m, \mathrm{pkgen}'(r), \mathrm{sign}'(m, \mathrm{skgen}'(r), r')) = \mathrm{true}$
$\mathsf{new}\ x : T_r; \mathsf{new}\ y : T_r; f(x) = f(y) \approx \mathrm{false}$
$\quad \mathsf{for}\ f \in \{\mathrm{pkgen}, \mathrm{skgen}, \mathrm{pkgen}', \mathrm{skgen}'\}$
$\mathsf{new}\ x : T_r; \mathsf{new}\ y : T_r; \mathrm{pkgen}(x) = \mathrm{pkgen}'(y) \approx \mathrm{false}$
$\mathsf{new}\ x : T_r; \mathsf{new}\ y : T_r; \mathrm{skgen}(x) = \mathrm{skgen}'(y) \approx \mathrm{false}$

$!^{i_k \le n_k} \mathsf{new}\ r : T_r; ($
$\quad () \to_{[2]} \mathrm{pkgen}(r),$
$\quad !^{i_s \le n_s} \mathsf{new}\ r' : T'_r; (x : T) \to \mathrm{sign}(x, \mathrm{skgen}(r), r')$
$\quad !^{i_c \le n_c}(m' : T, si' : T_s) \to \mathrm{check}(m', \mathrm{pkgen}(r), si')),$
$!^{i \le n}(m : T, y : T_{pk}, si : T_s) \to_{[3]} \mathrm{check}(m, y, si)\ [\mathit{all}]$
$\approx$
$!^{i_k \le n_k} \mathsf{new}\ r : T_r; ($
$\quad () \to \mathrm{pkgen}'(r),$
$\quad !^{i_s \le n_s} \mathsf{new}\ r' : T'_r; (x : T) \to \mathrm{sign}'(x, \mathrm{skgen}'(r), r')$
$\quad !^{i_c \le n_c}(m' : T, si' : T_s) \to$
$\quad\quad \mathsf{find}\ u_s \le n_s\ \mathsf{suchthat}\ \mathsf{defined}(x[u_s]) \wedge m' = x[u_s]$
$\quad\quad\quad \wedge\ \mathrm{check}'(m', \mathrm{pkgen}(r), si')\ \mathsf{then}\ \mathrm{true}\ \mathsf{else}\ \mathrm{false}),$
$!^{i \le n}(m : T, y : T_{pk}, si : T_s) \to$
$\quad \mathsf{find}\ u \le n_k, u_s \le n_s\ \mathsf{suchthat}\ \mathsf{defined}(r[u], x[u, u_s])$
$\quad\quad \wedge\ y = \mathrm{pkgen}'(r[u]) \wedge m = x[u, u_s]$
$\quad\quad \wedge\ \mathrm{check}'(m, y, si)\ \mathsf{then}\ \mathrm{true}\ \mathsf{else}$
$\quad \mathsf{find}\ u \le n_k\ \mathsf{suchthat}\ \mathsf{defined}(r[u])$
$\quad\quad \wedge\ y = \mathrm{pkgen}'(r[u])\ \mathsf{then}\ \mathrm{false}\ \mathsf{else}$
$\quad \mathrm{check}(m, y, si)$

The first three lines of each side of the equivalence express that the generation of public keys and the computation of the signature are left unchanged in the transformation. The verification of a signature $\mathrm{check}(m', \mathrm{pkgen}(r), si')$ is replaced with a lookup in the previously computed signatures: the signature can be valid only when it has been computed by the signature oracle $\mathrm{sign}'(x, \mathrm{skgen}'(r), r')$, that is, when $m = x[u_s]$ for some u_s. Lines 5-6 of the right-hand side of the equivalence try to find such a u_s and return true when they succeed, and false otherwise.

The last oracle considers signature verifications in which the verification key may not be explicitly known: it can be any term, which we represent by a variable y. This can happen fort instance when the verification key is received in a previous message. In this case, the verification of a signature $\mathrm{check}(m, y, si)$ is also replaced with a lookup in the previously computed signatures: if the signature is checked using one of the keys $\mathrm{pkgen}'(r[u])$ (that is, if $y = \mathrm{pkgen}'(r[u])$), then it can be valid only when it has been computed by the signature oracle $\mathrm{sign}'(x, \mathrm{skgen}'(r[u]), r')$, that is, when $m = x[u, u_s]$ for some u_s. Lines 8-10 of the right-hand side of the equivalence try to find such a u' and return true when they succeed. Lines 11-12 of the right-hand side returns false when no such u' is found in lines 8-10, but $y = \mathrm{pkgen}'(r[u])$ for some u. The last line handles the case when the key y is not $\mathrm{pkgen}'(r[u])$. In this case, we check the signature as before. (Using c and not c' in the last line of the transformation allows to reapply this transformation with another value of r.)

The priorities [2] and [3] in the left-hand side on the equivalence specify that the oracle $\mathrm{check}(m', \mathrm{pkgen}(r), si')$ should be used when possible, rather than using the two oracles $\mathrm{pkgen}(r)$ and $\mathrm{check}(m, y, si)$ to encode the same computation. (Priorities are defined in Section D.2.2.) Using the oracle $\mathrm{check}(m', \mathrm{pkgen}(r), si')$ yields a simpler transformed game.

We can model deterministic signatures in a similar way, by removing the third argument of s.

SUF-CMA Signature Strongly unforgeable signatures can be modeled in a similar way. The equivalence is replaced with:

$!^{i_k \le n_k}$ new $r : T_r;($
  $() \to_{[2]} \mathrm{pkgen}(r),$
  $!^{i_s \le n_s}$ new $r' : T_r'; (x : T) \to \mathrm{sign}(x, \mathrm{skgen}(r), r')$
  $!^{i_c \le n_c}(m' : T, si' : T_s) \to \mathrm{check}(m', \mathrm{pkgen}(r), si')),$
$!^{i \le n}(m : T, y : T_{pk}, si : T_s) \to_{[3]} \mathrm{check}(m, y, si)\ [all]$
$\approx$

1. $!^{i_k \le n_k}$ new $r : T_r;($
2. $\quad () \to \mathrm{pkgen}'(r),$
3. $\quad !^{i_s \le n_s}$ new $r' : T_r'; (x : T) \to$
   $\quad\quad$ let $s : T_s = \mathrm{sign}'(x, \mathrm{skgen}'(r), r')$ in s
4. $\quad !^{i_c \le n_c}(m' : T, si' : T_s) \to$
5. $\quad\quad$ find $u_s \le n_s$ suchthat defined$(x[u_s]) \wedge m' = x[u_s]$
6. $\quad\quad\quad \wedge\, si' = s[u_s]$ then true else $\mathrm{false}),$
7. $!^{i \le n}(m : T, y : T_{pk}, si : T_s) \to$
8. $\quad$ find $u \le n_k, u_s \le n_s$ suchthat defined$(r[u], x[u, u_s])$
9. $\quad\quad \wedge\, y = \mathrm{pkgen}'(r[u]) \wedge m = x[u, u_s]$
10. $\quad\quad \wedge\, si = s[u, u_s]$ then true else
11. $\quad$ find $u \le n_k$ suchthat defined$(r[u])$
12. $\quad\quad \wedge\, y = \mathrm{pkgen}'(r[u])$ then false else
13. $\quad \mathrm{check}(m, y, si)$

This equivalence requires the signature given to the verification oracle to be exactly the signature generated by the signature oracle ($si' = s[u_s]$ line 6 and $si = s[u, u_s]$ line 10).

SUF-CMA MACs can be modeled by modifying the definition of UF-CMA MACs ($\mathrm{mac}_{\mathsf{eq}}$) along similar lines.

IND-CCA2 Public-Key Encryption

T_r large, fixed length; T_r' fixed length
$\mathrm{enc}, \mathrm{enc}' : T \times T_{pk} \times T_r' \to T_e$
$\mathrm{dec}, \mathrm{dec}' : T_e \times T_{sk} \to T_\perp$
$\mathrm{skgen}, \mathrm{skgen}' : T_r \to T_{sk}$
$\mathrm{pkgen}, \mathrm{pkgen}' : T_r \to T_{pk}$
$\mathrm{i}_\perp : T \to T_\perp$ (poly-injective)
$\mathrm{Z}_T : T$

$\forall m : T, \forall r : T_r, \forall r' : T_r',$
$\quad \mathrm{dec}(\mathrm{enc}(m, \mathrm{pkgen}(r), r'), \mathrm{skgen}(r)) = \mathrm{i}_\perp(m)$
$\forall m : T, \forall r : T_r, \forall r' : T_r',$
$\quad \mathrm{dec}'(\mathrm{enc}'(m, \mathrm{pkgen}'(r), r'), \mathrm{skgen}'(r)) = \mathrm{i}_\perp(m)$
new $x : T_r$; new $y : T_r; f(x) = f(y) \approx \mathrm{false}$
$\quad$ for $f \in \{\mathrm{pkgen}, \mathrm{pkgen}', \mathrm{skgen}, \mathrm{skgen}'\}$
new $x : T_r$; new $y : T_r; \mathrm{pkgen}(x) = \mathrm{pkgen}'(y) \approx \mathrm{false}$
new $x : T_r$; new $y : T_r; \mathrm{skgen}(x) = \mathrm{skgen}'(y) \approx \mathrm{false}$

$!^{i_k \le n_k}$ new $r : T_r;($
  $() \to_{[2]} \mathrm{pkgen}(r),$
  $!^{i_e \le n_e}$ new $r' : T_r'; (x' : T) \to \mathrm{enc}(x', \mathrm{pkgen}(r), r')$
  $!^{i_d \le n_d}(m : T_e) \to \mathrm{dec}(m, \mathrm{skgen}(r))),$
$!^{i \le n}$ new $r'' : T_r'; (x : T, y : T_{pk}) \to_{[3]} \mathrm{enc}(x, y, r'')\ [all]$
$\approx$
$!^{i_k \le n_k}$ new $r : T_r;($
  $() \to \mathrm{pkgen}'(r),$
  $!^{i_e \le n_e}$ new $r' : T_r'; (x' : T) \to$
    let $m' : T_e = \mathrm{enc}'(\mathrm{Z}_T, \mathrm{pkgen}'(r), r')$ in m',
  $!^{i_d \le n_d}(m : T_e) \to$
    find $u \le n_e$ suchthat defined$(m'[u], x'[u]) \wedge m = m'[u]$
      then $\mathrm{i}_\perp(x'[u])$ else
    find $u \le n$ suchthat defined$(m''[u], x[u], y[u]) \wedge$
      $y[u] = \mathrm{pkgen}'(r) \wedge m = m''[u]$ then $\mathrm{i}_\perp(x[u])$ else
    $\mathrm{dec}'(m, \mathrm{skgen}'(r))),$

$$
\begin{array}{l}
!^{i \le n}(x : T, y : T_{pk}) \to \\
\quad \mathsf{find}\ u_k \le n_k\ \mathsf{suchthat\ defined}(r[u_k]) \wedge y = \mathrm{pkgen}'(r[u_k]) \\
\quad \mathsf{then\ new}\ r'' : T'_r; \\
\qquad \mathsf{let}\ m'' : T_e = \mathrm{enc}'(\mathrm{Z}_T, \mathrm{pkgen}'(r[u_k]), r'')\ \mathsf{in}\ m'' \\
\quad \mathsf{else\ new}\ r''' : T'_r; \mathrm{enc}(x, y, r''')
\end{array}
$$

In the right-hand side, the encryption oracles encrypt zeroes (Z_T) instead of the real plaintext, when the encryption key is a $\mathrm{pkgen}'(r)$. The decryption oracle uses array lookups to see if the received ciphertext m is the result of an encryption oracle. If it is, it returns the corresponding plaintext. Otherwise, it decrypts normally.

The result of decryption can be either the plaintext, or the special symbol $\perp$, which represents a failure of decryption. Hence, when decryption returns $\mathrm{i}_\perp(x)$, it means that decryption succeeded with plaintext x.

Similarly to signatures, the priorities [2] and [3] in the left-hand side on the equivalence specify that the oracle $\mathrm{enc}(x', \mathrm{pkgen}(r), r')$ should be used when possible, rather than using the two oracles $\mathrm{pkgen}(r)$ and $\mathrm{enc}(x, y, r'')$ to encode the same computation.

When the decryption oracle is removed from the equivalence, we obtain a model of IND-CPA public key encryption.

D.3.3 Hash Functions

Collision Resistant Hash Function

$$
\begin{array}{l}
T_k \text{ fixed length} \\
\mathrm{h} : T_k \times \mathit{bitstring} \to T \\
\mathsf{new}\ k : T_k; \forall x : \mathit{bitstring}, \forall y : \mathit{bitstring}, \\
\quad \mathrm{h}(k, x) = \mathrm{h}(k, y) \approx x = y
\end{array}
$$

Hash Function in the Random Oracle Model

$$
\begin{array}{l}
T_k, T \text{ fixed length} \\
\mathrm{h} : T_k \times \mathit{bitstring} \to T \\
!^{ih \le nh}\ \mathsf{new}\ k : T_k; !^{i \le n}(x : \mathit{bitstring}) \to \mathrm{h}(k, x) \\
\approx_0 \\
!^{ih \le nh}\ !^{i \le n}(x : \mathit{bitstring}) \to \\
\qquad \mathsf{find}\ u \le n\ \mathsf{suchthat\ defined}(x[u], r[u]) \wedge x = x[u] \\
\qquad \mathsf{then}\ r[u] \\
\qquad \mathsf{else\ new}\ r : T; r
\end{array}
$$

In this model, the hash function is keyed: the choice of the key models the choice of the hash function. The key k must be chosen randomly at the beginning of the game, and the game must include, in parallel with the protocol to verify, the process $!^{i \le n} c(x : \mathit{bitstring}); \overline{c}\langle \mathrm{h}(k, x) \rangle$ which represent a hash oracle. Otherwise, CryptoVerif would incorrectly assume that the adversary cannot compute the hash function. This particularity is related to the fact that a random oracle is unimplementable: otherwise, the adversary could implement it without being explicitly given access to it.

D.3.4 Ideal Cipher Model

$$
\begin{array}{l}
T_{ck} \text{ fixed length}; T_b \text{ large, fixed length} \\
\mathrm{enc}, \mathrm{dec} : T_{ck} \times T_b \times T_k \to T_b \\
\forall ck : T_{ck}, \forall m : T_b, \forall k : T_k, \mathrm{dec}(ck, \mathrm{enc}(ck, m, k), k) = m \\
!^{ick \le nck}\ \mathsf{new}\ ck : T_{ck}; \\
\quad (!^{ie \le ne}(me : T_b, ke : T_k) \to \mathrm{enc}(ck, me, ke), \\
\quad\ !^{id \le nd}(md : T_b, kd : T_k) \to \mathrm{dec}(ck, md, kd)) \\
\approx \\
!^{ick \le nck} \\
\quad (!^{ie \le ne}(me : T_b, ke : T_k) \to \\
\qquad \mathsf{find}\ j \le ne\ \mathsf{suchthat\ defined}(me[j], ke[j], re[j]) \wedge \\
\qquad\quad me = me[j] \wedge ke = ke[j]\ \mathsf{then}\ re[j] \\
\qquad \oplus\ k \le ne\ \mathsf{suchthat\ defined}(rd[k], md[k], kd[k]) \wedge \\
\qquad\quad me = rd[k] \wedge ke = kd[k]\ \mathsf{then}\ md[k] \\
\qquad \mathsf{else\ new}\ re : T_b; re, \\
\quad !^{id \le nd}(md : T_b, kd : T_k) \to \\
\qquad \mathsf{find}\ j \le ne\ \mathsf{suchthat\ defined}(me[j], ke[j], re[j]) \wedge \\
\qquad\quad md = re[j] \wedge kd = ke[j]\ \mathsf{then}\ me[j] \\
\qquad \oplus\ k \le nd\ \mathsf{suchthat\ defined}(rd[k], md[k], kd[k]) \wedge \\
\qquad\quad md = md[k] \wedge kd = kd[k]\ \mathsf{then}\ rd[k] \\
\qquad \mathsf{else\ new}\ rd : T_b; rd)
\end{array}
$$

The ideal cipher model has been introduced in [16] in order to model block ciphers. It is in the same vein as the random oracle model, but with families of perfectly random permutations. The encryption and decryption functions map bitstrings of type T_b (for a block) to bitstrings of type T_b; they take two keys as additional arguments: the standard encryption/decryption key of type T_k, but also a key ck of type T_{ck} that models the choice of the scheme itself (much like in the random oracle model). This key ck must be chosen randomly at the beginning of the game, and the game must include, in parallel with the protocol to verify, the process $!^{i \le n} c(x : T_b, k : T_k); \overline{c}\langle \mathrm{enc}(ck, x, k) \rangle \mid !^{i' \le n'} c'(x : T_b, k : T_k); \overline{c'}\langle \mathrm{dec}(ck, x, k) \rangle$ which represent the encryption and decryption oracles.

The left- and right-hand sides of the equivalence define the encryption and decryption oracles. In the left-hand side, they call the encryption and decryption functions. In the right-hand side, they are replaced with lookups in previous encryption/decryption queries. For instance, for encryption, we look for a previous encryption query of the same cleartext ($me = me[j]$) under the same key ($ke = ke[j]$) and, if we find one, we return the same ciphertext $re[j]$. We also look for a previous decryption query that has returned as cleartext the cleartext to encrypt ($me = rd[k]$) using the same key ($ke = kd[k]$) and, if we find one, we return the corresponding ciphertext $md[k]$. Otherwise, we return a fresh random ciphertext re. This definition does not yield random permutations, because the random choices of re and rd may return several times the same result and may collide with previous values of me and md. However, these collisions have a negligible probability of occurring, so the equivalence holds.

D.3.5 Xor

$$\begin{array}{l}\mathrm{xor} : T \times T \to T \text{ (commutative)}\\ \forall x : T, y : T, \mathrm{xor}(x, \mathrm{xor}(x, y)) = y.\\ \forall x : T, y : T, z : T, (\mathrm{xor}(x, z) = \mathrm{xor}(y, z)) = (x = y).\\ !^{i \leq n} \mathsf{new}\ k : T; (x : T) \to \mathrm{xor}(x, k)\\ \quad\approx_0\\ !^{i \leq n} \mathsf{new}\ k : T; (x : T) \to k\end{array}$$

This modeling of xor could be improved by taking into account more equations, in particular associativity.

E Proofs

E.1 Proof of Proposition 1

The proof that Q'_0 satisfies Invariants 1, 2, and 3 is in general easy, and the proof of $Q_0 \approx_0^V Q'_0$ relies on a correspondence between traces of $C[Q_0]$ and traces of $C[Q'_0]$, with the same probability and such that a configuration of the trace of $C[Q_0]$ executes $\overline{c}\langle a\rangle$ immediately if and only if the corresponding configuration of the corresponding trace of $C[Q'_0]$ executes $\overline{c}\langle a\rangle$ immediately. This correspondence is obtained by replacing some internal actions of Q_0 with corresponding internal actions of Q'_0. We sketch the proof only for the cases of **SArename**(x) and **Simplify**, and leave the case of **RemoveAssign**(x) to the reader.

Proof sketch of Proposition 1 for SArename(x) The process Q'_0 satisfies Invariant 1 because definitions of variables duplicated by **SArename** all occur in a different branch of a find.

For Invariant 2, each variable access $x_j[M_1, \dots, M_l]$ in Q'_0 comes from a variable access $x[M_1, \dots, M_l]$ in Q_0. Since Q_0 satisfies Invariant 2, either this access is under its definition, in which case **SArename**(x) has replaced this definition of x with a definition of x_j, so $x_j[M_1, \dots, M_l]$ is under its definition in Q'_0; or this access is in a defined test, in which case it is also in a defined test in Q'_0; or this access is in a branch of find with a condition defined$(N_1, \dots, N_{l'})$ such that $x[M_1, \dots, M_l]$ is a subterm of N_j for some $j \leq l'$, in which case $x[M_1, \dots, M_l]$ has been substituted with $x_j[M_1, \dots, M_l]$ in this branch of find, so $x_j[M_1, \dots, M_l]$ is under a suitable defined condition. Therefore, Q'_0 satisfies Invariant 2.

For Invariant 3, the type environment $\mathcal{E}'$ for Q'_0 is obtained from the type environment $\mathcal{E}$ for Q_0, by setting $\mathcal{E}'(x_1) = \ldots = \mathcal{E}'(x_m) = \mathcal{E}(x)$ and $\mathcal{E}'(x)$ is not defined. (Indeed, all definitions of x in Q_0 have the same type $\mathcal{E}(x)$, which is therefore the type of the definitions of $x_j, j \leq m$ in Q'_0.) The proof of $\mathcal{E}' \vdash Q'_0$ is obtained from the proof of $\mathcal{E} \vdash Q_0$, by replacing requests to $\mathcal{E}(x)$ with requests to $\mathcal{E}(x_j)$ for some $j \leq m$, and duplicating parts of the proof of $\mathcal{E} \vdash Q_0$ that correspond to duplicated branches of find.

Finally, let us prove that $Q_0 \approx_0^V Q'_0$. We denote by $\mathit{SArename}(x, Q)$ the process obtained by applying **SArename**(x) to Q. Let j be a partial function from l-tuples of indices $a_1, \dots, a_l$ to subscripts $1, \dots, m$ of variable x. Informally, j is such that $x[a_1, \dots, a_l]$ in a trace of Q_0 corresponds to $x_{j(a_1,\dots,a_l)}[a_1, \dots, a_l]$ in the corresponding trace of Q'_0. We define a function $\mathit{SArename}_j$ that relates configurations in a trace of Q_0 to configurations in a trace of the renamed process Q'_0. Below, we will show that this function maps traces of Q_0 to traces of Q'_0 of the same probability, which will show the desired equivalence $Q_0 \approx_0^V Q'_0$.

- We define $\mathit{SArename}_j$ for terms so that $\mathit{SArename}_j(x, E, M)$ replaces occurrences of x in M with the appropriate x_j. More precisely,

$$\begin{array}{l}\mathit{SArename}_j(x, E, x[M_1, \dots, M_l]) =\\ \quad x_{j(a_1,\dots,a_l)}[\mathit{SArename}_j(x, E, M_1), \dots,\\ \qquad\qquad \mathit{SArename}_j(x, E, M_l)]\\ \qquad \text{when } E, M_k \Downarrow a_k \text{ for } k \leq l \text{ and}\\ \qquad x[a_1, \dots, a_l] \in \mathrm{Dom}(E);\\ \mathit{SArename}_j(x, E, y[M_1, \dots, M_l]) =\\ \quad y[\mathit{SArename}_j(x, E, M_1), \dots, \mathit{SArename}_j(x, E, M_l)]\\ \qquad \text{when } y \neq x;\\ \mathit{SArename}_j(x, E, f(M_1, \dots, M_l)) =\\ \quad f(\mathit{SArename}_j(x, E, M_1), \dots, \mathit{SArename}_j(x, E, M_l));\\ \mathit{SArename}_j(x, E, i) = i\end{array}$$

- We define $\mathit{SArename}_j$ for (input and output) processes as follows: $\mathit{SArename}_j(x, E, P_1)$ first computes $\mathit{SArename}(x, P_1) = P_2$. More precisely, it renames each definition of x to the name used when renaming the whole process Q_0; it replaces variable accesses to x with variable accesses to x_j when the definition of x that caused this replacement in Q_0 also occurs in P_1; it duplicates branches of find as $\mathit{SArename}(x, Q_0)$, renaming variable accesses to x into variable accesses to x_j when the find that caused this replacement in Q_0 also occurs in P_1. (When a variable access to x is under both a definition of x and find, or under several nested finds that guarantee that it is defined, it is important to follow exactly the renaming procedure that happened in Q_0. Formally, this can be done by annotating each construct in processes with a distinct occurrence symbol and by reducing annotated processes. When we perform $\mathit{SArename}(x, Q_0)$, we can then remember the occurrence symbols of the constructs that cause each variable renaming.) Finally, $\mathit{SArename}_j$ replaces each term M in P_2 with $\mathit{SArename}_j(x, E, M)$.

- We also define $\mathit{SArename}_j$ for environments: $E' = \mathit{SArename}_j(x, E)$ if and only if $E'(x_{j(a_1,\dots,a_l)}[a_1, \dots, a_l]) = E(x[a_1, \dots, a_l])$ when $x[a_1, \dots, a_l] \in \mathrm{Dom}(E)$, $E'(y[a_1, \dots, a_l]) = E(y[a_1, \dots, a_l])$ when $y \neq x$ and $y[a_1, \dots, a_l] \in \mathrm{Dom}(E)$, and $E'(y[a_1, \dots, a_l])$ is undefined in all other cases.

- We extend $\mathit{SArename}_j$ to semantic configurations:

$$\begin{array}{l}\mathit{SArename}_j(x, (E, P, \mathcal{Q}, \mathcal{C})) =\\ \quad (\mathit{SArename}_j(x, E), \mathit{SArename}_j(x, E, P),\\ \quad\ \{\mathit{SArename}_j(x, E, Q_1) \mid Q_1 \in \mathcal{Q}\}, \mathcal{C})\end{array}$$

  We also define $\mathit{SArename}_j(x, (E, \mathcal{Q}, \mathcal{C}))$ in the same way.

We first show that if $E, M \Downarrow a$, then

$$\mathit{SArename}_j(x, E), \mathit{SArename}_j(x, E, M) \Downarrow a$$

The proof proceeds by induction on M. The only interesting case is $M = x[M_1, \dots, M_l]$. Since $E, M \Downarrow a$ has been derived by (Var), $E, M_k \Downarrow a_k$ for all $k \leq l$ and $a = E(x[a_1, \dots, a_l])$. By induction hypothesis, $\mathit{SArename}_j(x, E), \mathit{SArename}_j(x, E, M_k) \Downarrow a_k$ for all $k \leq l$. Moreover,

$$\begin{aligned}&\mathit{SArename}_j(x, E, x[M_1, \dots, M_l]) = \\ &\quad x_{j(a_1,\dots,a_l)}[\mathit{SArename}_j(x, E, M_1), \dots, \\ &\qquad \mathit{SArename}_j(x, E, M_l)]\end{aligned}$$

and

$$\begin{aligned}&\mathit{SArename}_j(x, E)(x_{j(a_1,\dots,a_l)}[a_1, \dots, a_l]) = \\ &\quad E(x[a_1, \dots, a_l]) = a\end{aligned}$$

so $\mathit{SArename}_j(x, E), \mathit{SArename}_j(x, E, M) \Downarrow a$.

Next, we can show by cases on the reduction $E, \mathcal{Q}, \mathcal{C} \rightsquigarrow E', \mathcal{Q}', \mathcal{C}'$ that, if $E, \mathcal{Q}, \mathcal{C} \rightsquigarrow E', \mathcal{Q}', \mathcal{C}'$, then

$$\mathit{SArename}_j(x, (E, \mathcal{Q}, \mathcal{C})) \rightsquigarrow \mathit{SArename}_j(x, (E', \mathcal{Q}', \mathcal{C}')).$$

Hence

$$\begin{aligned}&\mathit{SArename}_j(x, \mathrm{reduce}(E, \mathcal{Q}, \mathcal{C})) = \\ &\quad \mathrm{reduce}(\mathit{SArename}_j(x, (E, \mathcal{Q}, \mathcal{C})))\end{aligned}$$

Let C be any evaluation context acceptable for Q_0, Q_0', V. We show that for each trace $\mathrm{initConfig}(C[Q_0]) \to_\eta \dots \to_\eta E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m$, there exists a trace $\mathrm{initConfig}(C[Q_0']) \to_\eta \dots \to_\eta E_m', P_m', \mathcal{Q}_m', \mathcal{C}_m$ with the same probability, and a function j_m such that $E_m', P_m' \mathcal{Q}_m', \mathcal{C}_m = \mathit{SArename}_{j_m}(x, (E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m))$. The proof proceeds by induction on the length m of the trace. For the induction step, we distinguish cases depending on the last reduction step of the trace.

- Initial case $m = 0$: $\mathrm{fc}(C[Q_0]) = \mathrm{fc}(C[Q_0'])$ since the transformation **SArename** does not modify channels. Let j_0 be the function defined nowhere. We have, $C[Q_0'] = \mathit{SArename}_{j_0}(x, \emptyset, C[Q_0])$. Indeed, since $x \notin V$, $x \notin \mathrm{var}(C)$, so

  $$\begin{aligned}&\mathit{SArename}_{j_0}(x, \emptyset, C[Q_0]) = \mathit{SArename}(x, C[Q_0]) = \\ &\quad C[\mathit{SArename}(x, Q_0)] = C[Q_0']\end{aligned}$$

  Therefore,

  $$\begin{aligned}&\mathit{SArename}_{j_0}(x, (\emptyset, \{C[Q_0]\}, \mathrm{fc}(C[Q_0]))) = \\ &\quad (\emptyset, \{C[Q_0']\}, \mathrm{fc}(C[Q_0']))\end{aligned}$$

  Hence we have

  $$\begin{aligned}&\mathit{SArename}_{j_0}(x, \mathrm{reduce}(\emptyset, \{C[Q_0]\}, \mathrm{fc}(C[Q_0]))) = \\ &\quad \mathrm{reduce}(\emptyset, \{C[Q_0']\}, \mathrm{fc}(C[Q_0']))\end{aligned}$$

  Thus,

  $$\begin{aligned}&\mathit{SArename}_{j_0}(x, \mathrm{initConfig}(C[Q_0])) = \\ &\quad \mathrm{initConfig}(C[Q_0'])\end{aligned}$$

- The last step of the trace is a definition of $x[a_1, \dots, a_l]$: By induction hypothesis, we have a trace of length $m-1$, with an associated function j_{m-1}. Since $C[Q_0]$ satisfies Invariant 1, the configuration $E_{m-1}, P_{m-1}, \mathcal{Q}_{m-1}, \mathcal{C}_{m-1}$ satisfies Invariant 4, so $x[a_1, \dots, a_l] \notin \mathrm{Dom}(E_{m-1})$. Since $P_{m-1}' = \mathit{SArename}_{j_{m-1}}(x, E_{m-1}, P_{m-1})$, the first instruction of P_{m-1}' is a definition of $x_k[a_1, \dots, a_l]$ for some k (using the property "if $E, M \Downarrow a$, then $\mathit{SArename}_j(x, E), \mathit{SArename}_j(x, E, M) \Downarrow a$" shown above to prove that the indices of x, resp. x_k, are the same in the execution of P_{m-1} and of P_{m-1}'). We define $j_m = j_{m-1}[(a_1, \dots, a_l) \mapsto k]$, and show that we obtain a suitable trace of length m with this function j_m.

- The last step of the trace is a find whose defined condition refers to x: By induction hypothesis, we have a trace of length $m-1$, with an associated function j_{m-1}. If a branch FB of the find in P_{m-1} succeeds for certain values of the variables defined by find, exactly one of its copies succeeds in P_{m-1}', the copy whose defined condition refers to $x_{j_{m-1}(a_1,\dots,a_l)}[a_1, \dots, a_l]$ when the defined condition of the branch FB in P_{m-1} refers to $x[a_1, \dots, a_l]$. If a branch of the find fails in P_{m-1}, all its copies fail in P_{m-1}'. Therefore, the number $|S|$ of successful choices of the find is the same in P_{m-1} and in P_{m-1}'. Hence, the probability that each successful branch is taken is the same. When P_{m-1} executes a successful branch, we build the corresponding trace of P_{m-1}' by executing the successful copy of this branch. When P_{m-1} executes the else branch, P_{m-1}' also executes the else branch. So we obtain a suitable trace of length m with associated function $j_m = j_{m-1}$ (except when the find also defines $x[a_1', \dots, a_l']$, in which case the previous item of the proof must also be applied).

- All other cases are easy: they execute in the same way in P_{m-1} and in P_{m-1}'.

We also show the converse property, that for each trace $\mathrm{initConfig}(C[Q_0']) \to_\eta \dots \to_\eta E_m', P_m', \mathcal{Q}_m', \mathcal{C}_m$, there exists a trace $\mathrm{initConfig}(C[Q_0]) \to_\eta \dots \to_\eta E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m$ with the same probability and

$$E_m', P_m' \mathcal{Q}_m', \mathcal{C}_m = \mathit{SArename}_{j_m}(x, (E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m)).$$

The proof is similar to the proof above.

If $E_m', P_m' \mathcal{Q}_m', \mathcal{C}_m = \mathit{SArename}_{j_m}(x, (E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m))$, then for all channels c and bitstrings a, $E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m$ executes $\overline{c}\langle a\rangle$ immediately if and only if $E_m', P_m', \mathcal{Q}_m', \mathcal{C}_m$ executes $\overline{c}\langle a\rangle$ immediately. So $\Pr[C[Q_0] \rightsquigarrow_\eta \overline{c}\langle a\rangle] = \Pr[C[Q_0'] \rightsquigarrow_\eta \overline{c}\langle a\rangle]$. Therefore, $Q_0 \approx_0^V Q_0'$. $\square$

Proof sketch of Proposition 1 for Simplify The proof of Invariants 1, 2, and 3 is relatively easy, so we focus on the proof of $Q_0 \approx^V Q_0'$.

Let C be any evaluation context acceptable for Q_0, Q_0', V. Let $q(\eta)$ be the maximum runtime of $C[Q_0]$, where q is a polynomial. We denote by $\mathbb{C}_0$ the initial configuration of $C[Q_0]$, $\mathrm{initConfig}(C[Q_0])$.

We define $p_{\max}(\eta) = \max(\{\frac{1}{|I_\eta(T)|} \mid T \text{ is a large type}\} \cup \{p(\eta)$ associated to user-defined rewrite rules, for an adversary of runtime $q(\eta)\})$. The probability $p_{\max}(\eta)$ is negligible, since it is the maximum of a constant number of negligible functions. We shall prove below that the probability that a desired fact does not hold is at most $q'(\eta)p_{\max}(\eta)$, where q' is a polynomial, so it is negligible.

The proof follows the structure of the simplification algorithm: we prove the correctness of each component of the algorithm separately.

Correctness of the collection of true facts. We consider a slightly modified semantics for our calculus, in which each process is accompanied with a substitution that defines the values of the replication indices in that process. For example, the rule (Repl) becomes in this semantics:

$$\begin{aligned}&E, \{(\sigma, !^{i \le n} Q)\} \uplus \mathcal{Q}, \mathcal{C} \rightsquigarrow \\ &\qquad E, \{(\sigma[i \mapsto a], Q) \mid a \in [1, I_\eta(n)]\} \uplus \mathcal{Q}, \mathcal{C}\end{aligned}$$

When evaluating a term M in a process with substitution (σ, Q) or (σ, P), we now use $E, \sigma, M \Downarrow a$ instead of $E, M \Downarrow a$, with the rule $E, \sigma, i \Downarrow \sigma i$ instead of (Cst), and the other rules modified accordingly.

The judgment $E, \sigma \vdash F$ means that a fact F holds in environment E and substitution σ. It is defined by $E, \sigma \vdash M$ if and only if $E, \sigma, M \Downarrow \mathrm{true}$; $E, \sigma \vdash \mathsf{defined}(M)$ if and only if $E, \sigma, M \Downarrow a$ for some a; $E, \sigma \vdash \mathit{elsefind}((u_1 \le n_1, \dots, u_m \le n_m), (M_1, \dots, M_l), M)$ if and only if for all $x_1 \in [1, I_\eta(n_1)], \dots, x_m \in [1, I_\eta(n_m)]$, we have $E, \sigma', (\mathsf{defined}(M_1, \dots, M_l) \wedge M) \Downarrow \mathrm{false}$ where $\sigma' = \sigma[u_1 \mapsto x_1, \dots, u_m \mapsto x_m]$. We extend this definition to sets of facts naturally. We say that $\mathcal{F}_P$ *is correct for all* P when $\mathbb{C}_0 \xrightarrow{p}_t \dots \xrightarrow{p'}_{t'} E, (\sigma, P), \mathcal{Q}, \mathcal{C}$ implies $E, \sigma \vdash \mathcal{F}_P$. Our goal is to show that $\mathcal{F}_P$ is indeed correct for all P.

For occurrences of processes P, Q in C and in the process $\overline{\mathit{start}}\langle\rangle; 0$ used in the initial configuration, we let $\mathcal{F}_P = \mathcal{F}_Q = \mathcal{F}_P^{\mathrm{Fut}} = \mathcal{F}_P^{\mathrm{ElseFind}} = \mathcal{F}_Q^{\mathrm{ElseFind}} = \emptyset$.

We show **S0**: immediately after calling collectFacts, if $E_1, (\sigma_1, P_1), \mathcal{Q}_1, \mathcal{C}_1 \xrightarrow{p}_t E, (\sigma, P), \mathcal{Q}, \mathcal{C}$ then $E, \sigma \vdash \mathcal{F}_P$. If the reduced process is in C, the result is obvious since $\mathcal{F}_P = \emptyset$. Otherwise, we proceed by cases on the reduction $E_1, (\sigma_1, P_1), \mathcal{Q}_1, \mathcal{C}_1 \xrightarrow{p}_t E, (\sigma, P), \mathcal{Q}, \mathcal{C}$. For example, in the case (Let), $E_1, \sigma, M \Downarrow a$, $a \in I_\eta(T)$, and $E_1, (\sigma, \mathsf{let}\ x[\widetilde{i}] : T = M \mathsf{\ in\ } P), \mathcal{Q}, \mathcal{C} \xrightarrow{1}_L E = E_1[x[\widetilde{\sigma i}] \mapsto a], (\sigma, P), \mathcal{Q}, \mathcal{C}$. We have $\mathcal{F}_P = \{\mathsf{defined}(x[\widetilde{i}]), x[\widetilde{i}] = M\}$. Since $E, \sigma, x[\widetilde{i}] \Downarrow a$, we have $E, \sigma \vdash \mathsf{defined}(x[\widetilde{i}])$. We also have $E, \sigma, M \Downarrow a$, so $E, \sigma \vdash x[\widetilde{i}] = M$, so $E, \sigma \vdash \mathcal{F}_P$. We proceed in a similar way for the other cases.

We show that, immediately after calling collectFacts, $\mathcal{F}_P$ is correct for all P, that is, if $\mathbb{C}_0 \xrightarrow{p}_t \dots \xrightarrow{p'}_{t'} E, (\sigma, P), \mathcal{Q}, \mathcal{C}$ then $E, \sigma \vdash \mathcal{F}_P$. For the initial configuration, the property is obvious since $\mathcal{F}_P = \emptyset$. For the other configurations, we conclude by (S0).

We show the invariant **S1**: $\mathcal{F}_{C[Q_0]} = \emptyset$ and if Q is an input process and P is the input or output process just above Q, then $\mathcal{F}_Q \subseteq \mathcal{F}_P$. This property is obvious after collectFacts since $\mathcal{F}_Q = \emptyset$, and it is preserved by all updates to $\mathcal{F}_Q$ (provided the consequences of defined facts are not added in Q before they are added in P, which we can easily satisfy).

We prove **S2**: if $E, \mathcal{Q}, \mathcal{C} \rightsquigarrow E', \mathcal{Q}', \mathcal{C}'$ and for all $(\sigma, Q) \in \mathcal{Q}$, $E, \sigma \vdash \mathcal{F}_Q$, then for all $(\sigma, Q) \in \mathcal{Q}'$, $E', \sigma \vdash \mathcal{F}_Q$. The proof is easy by cases on the derivation of $E, \mathcal{Q}, \mathcal{C} \rightsquigarrow E', \mathcal{Q}', \mathcal{C}'$, using (S1). Therefore, we have **S2'**: if $E', \mathcal{Q}', \mathcal{C}' = \mathrm{reduce}(E, \mathcal{Q}, \mathcal{C})$ and for all $(\sigma, Q) \in \mathcal{Q}$, $E, \sigma \vdash \mathcal{F}_Q$, then for all $(\sigma, Q) \in \mathcal{Q}'$, $E', \sigma \vdash \mathcal{F}_Q$.

Next, we prove that if $\mathcal{F}_P$ is correct for all P, then $\mathcal{F}'_P$ obtained by

$$\mathcal{F}'_P = \mathcal{F}_P \cup \mathcal{F}_{P'} \text{ if } P \text{ is immediately under } P'$$

is correct for all P. We show that, if $\mathbb{C}_0 \xrightarrow{p}_t \dots \xrightarrow{p'}_{t'} E, (\sigma, P), \mathcal{Q}, \mathcal{C}$ then for all $(\sigma', P') \in \{(\sigma, P)\} \uplus \mathcal{Q}$, $E, \sigma' \vdash \mathcal{F}'_{P'}$. The proof proceeds by induction on the length of the trace. For the initial configuration, $\mathcal{F}_{C[Q_0]} = \emptyset$ by (S1), so $\emptyset, \emptyset \vdash \mathcal{F}_{C[Q_0]}$, and $\emptyset, \emptyset \vdash \mathcal{F}_{\overline{\mathit{start}}\langle\rangle}$, so the property follows immediately from (S2'). For the inductive step, if the last reduction of the trace is (Output), we have $E_1, (\sigma_1, P_1), \{(\sigma, Q)\} \uplus \mathcal{Q}_1, \mathcal{C}_1 \xrightarrow{p'}_{t'} E, (\sigma, P), \mathcal{Q}, \mathcal{C}$ with $P_1 = \overline{c[M_1, \dots, M_l]}\langle N_1, \dots, N_k\rangle.Q_1$, $Q = c[M'_1, \dots, M'_l](x_1[\widetilde{i}] : T_1, \dots, x_k[\widetilde{i}] : T_k).P$, $E = E_1[x_1[\widetilde{\sigma i}] \mapsto \dots, \dots, x_k[\widetilde{\sigma i}] \mapsto \dots]$, $\mathcal{Q} = \mathcal{Q}_1 \uplus \mathcal{Q}_2$, and $E_1, \mathcal{Q}_2, \mathcal{C} = \mathrm{reduce}(E_1, \{(\sigma_1, Q_1)\}, \mathcal{C}_1)$. If P is in C, $\mathcal{F}'_P = \emptyset$, so $E, \sigma \vdash \mathcal{F}'_P$. Otherwise, $E, \sigma \vdash \mathcal{F}'_Q$ by induction hypothesis. Moreover $E, \sigma \vdash \mathcal{F}_P$ since $\mathcal{F}_P$ is correct for all P, so $E, \sigma \vdash \mathcal{F}'_P$ since $\mathcal{F}'_P = \mathcal{F}_Q \cup \mathcal{F}_P \subseteq \mathcal{F}'_Q \cup \mathcal{F}_P$. By induction hypothesis, for all $(\sigma', Q') \in \mathcal{Q}_1$, $E_1, \sigma' \vdash \mathcal{F}'_{Q'}$. Also by induction hypothesis, $E_1, \sigma_1 \vdash \mathcal{F}'_{P_1}$, so $E_1, \sigma_1 \vdash \mathcal{F}'_{Q_1} \subseteq \mathcal{F}'_{P_1}$ by (S1). By (S2'), for all $(\sigma', Q') \in \mathcal{Q}_2$, $E_1, \sigma' \vdash \mathcal{F}'_{Q'}$. So for all $(\sigma', Q') \in \mathcal{Q} = \mathcal{Q}_1 \uplus \mathcal{Q}_2$, $E_1, \sigma' \vdash \mathcal{F}'_{Q'}$, so $E, \sigma' \vdash \mathcal{F}'_{Q'}$ since E is an extension of E_1. If the last reduction is not (Output), it is of the form $E_1, (\sigma, P'), \mathcal{Q}, \mathcal{C} \xrightarrow{p}_t E, (\sigma, P), \mathcal{Q}, \mathcal{C}$ where E is an extension of E_1. By induction hypothesis, for all $(\sigma', Q') \in \mathcal{Q}$, $E_1, \sigma' \vdash \mathcal{F}'_{Q'}$, so for all $(\sigma', Q') \in \mathcal{Q}$, $E, \sigma', \vdash \mathcal{F}'_{Q'}$. Since $\mathcal{F}_P$ is correct for all P, $E, \sigma \vdash \mathcal{F}_P$ and $E_1, \sigma \vdash \mathcal{F}_{P'}$, so $E, \sigma \vdash \mathcal{F}_{P'}$, so $E, \sigma \vdash \mathcal{F}'_P = \mathcal{F}_P \cup \mathcal{F}_{P'}$.

We show **S3**: if $E, (\sigma, P), \mathcal{Q}, \mathcal{C} \xrightarrow{p}_t \dots \xrightarrow{p'}_{t'} E', (\sigma', P'), \mathcal{Q}', \mathcal{C}'$ where P' is an output and no process before P' in this trace is an output, then $E', \sigma' \vdash \mathcal{F}_P^{\mathrm{Fut}}$. Since no process before P' in this trace is an output, this trace does not contain the reduction rule (Output). The proof proceeds by induction on P. If P is an output, the result is obvious since $\mathcal{F}_P^{\mathrm{Fut}} = \mathrm{collectFacts}(P) = \emptyset$. Otherwise, let $P_1, \dots, P_m$ be the immediate subprocesses of P. We have $E, (\sigma, P), \mathcal{Q}, \mathcal{C} \xrightarrow{p}_t E_1, (\sigma, P_j), \mathcal{Q}, \mathcal{C}$ for some extension E_1 of E and some $j \in \{1, \dots, m\}$. Moreover, by definition of collectFacts, $\mathcal{F}_P^{\mathrm{Fut}} = \mathrm{collectFacts}(P) = \bigcap_{j=1}^m (\mathcal{F}_{P_j} \cup \mathcal{F}_{P_j}^{\mathrm{Fut}})$, where the value of $\mathcal{F}_{P_j}$ is considered immediately after calling collectFacts. By (S0), $E_1, \sigma \vdash \mathcal{F}_{P_j}$, so $E', \sigma' \vdash \mathcal{F}_{P_j}$ since E' is an extension of E_1 and $\sigma' = \sigma$ since no (Output) reduction occurs in this trace. By induction hypothesis, $E', \sigma' \vdash \mathcal{F}_{P_j}^{\mathrm{Fut}}$, so $E', \sigma' \vdash \mathcal{F}_{P_j} \cup \mathcal{F}_{P_j}^{\mathrm{Fut}}$ for some $j \in \{1, \dots, m\}$. Therefore, $E', \sigma' \vdash \mathcal{F}_P^{\mathrm{Fut}}$.

We now show that if $\mathcal{F}_P$ is correct for all P, and $\mathcal{F}'_P$ is obtained by

$$\mathcal{F}'_P = \mathcal{F}_P \cup \left(\bigcap_{(x[i_1, \dots, i_m], P') \in \mathcal{D}} \begin{cases} \sigma'(\mathcal{F}_{P'} \cup (\mathcal{F}_{P'}^{\mathrm{Fut}} \cap \mathcal{F}_P)) & \\ \qquad\qquad \text{if } P \text{ is under } P' & \\ \sigma'(\mathcal{F}_{P'} \cup \mathcal{F}_{P'}^{\mathrm{Fut}}) \text{ otherwise} & \end{cases} \right)$$

where $\sigma' = \{M_1/i_1, \dots, M_m/i_m\}$, when $\mathsf{defined}(M) \in \mathcal{F}_P$ and $x[M_1, \dots, M_m]$ is a subterm of M, and $\mathcal{F}'_P = \mathcal{F}_P$ otherwise, then $\mathcal{F}'_P$ is also correct for all P. We assume that $\mathbb{C}_0 \xrightarrow{p}_t \dots \xrightarrow{p'}_{t'} E, (\sigma, P), \mathcal{Q}, \mathcal{C}$ and show that $E, \sigma \vdash \mathcal{F}'_P$. Since $\mathcal{F}_P$ is correct for all P, $E, \sigma \vdash \mathcal{F}_P$. Since $E, \sigma \vdash \mathsf{defined}(M)$, $E, \sigma, M_j \Downarrow a_j$ for all $j \leq m$ and $x[a_1, \dots, a_m] \in \mathrm{Dom}(E)$. Therefore, some definition of $x[a_1, \dots, a_m]$ has been executed in the considered trace. Next, we show that, for some $(x[i_1, \dots, i_m], P') \in \mathcal{D}$, we have $E, \sigma_1 \vdash \mathcal{F}_{P'}$; if P is under P' then $E, \sigma_1 \vdash \mathcal{F}^{\mathrm{Fut}}_{P'} \cap \mathcal{F}_P$; and if P is not under P' then $E, \sigma_1 \vdash \mathcal{F}^{\mathrm{Fut}}_{P'}$, where $\sigma_1(i_1) = a_1, \dots, \sigma_1(i_m) = a_m$. The desired result follows. Let $E_1, (\sigma_1, P_1), \mathcal{Q}_1, \mathcal{C}_1 \xrightarrow{p_1}_{t_1} E_2, (\sigma_1, P_2), \mathcal{Q}_2, \mathcal{C}_2$ be the reduction that defines $x[a_1, \dots, a_m]$ in the considered trace. We have $E_2, \sigma_1 \vdash \mathcal{F}_{P_2}$ since $\mathcal{F}_P$ is correct for all P. So $E, \sigma_1 \vdash \mathcal{F}_{P_2}$ since E is an extension of E_2 so all facts that hold in E_2 also hold in E. We have $(x[i_1, \dots, i_m], P_2) \in \mathcal{D}$. If P is not under P_2, the trace $E_2, (\sigma_1, P_2), \mathcal{Q}_2, \mathcal{C}_2 \xrightarrow{p_2}_{t_2} \dots \xrightarrow{p'}_{t'} E, (\sigma, P), \mathcal{Q}, \mathcal{C}$ must execute an output, so by (S3), $E_3, \sigma_3 \vdash \mathcal{F}^{\mathrm{Fut}}_{P_2}$ where the configuration in which the first output after $E_2, (\sigma_1, P_2), \mathcal{Q}_2, \mathcal{C}_2$ is executed is $E_3, (\sigma_3, P_3), \mathcal{Q}_3, \mathcal{C}_3$, so $E, \sigma_1 \vdash \mathcal{F}^{\mathrm{Fut}}_{P_2}$. (We have $\sigma_3 = \sigma_1$, since the substitution σ is changed only when executing a communication.) If P is under P_2, two cases can happen. Either the trace $E_2, (\sigma_1, P_2), \mathcal{Q}_2, \mathcal{C}_2 \xrightarrow{p_2}_{t_2} \dots \xrightarrow{p'}_{t'} E, (\sigma, P), \mathcal{Q}, \mathcal{C}$ executes an output, and we have $E, \sigma_1 \vdash \mathcal{F}^{\mathrm{Fut}}_{P_2}$ as above, or $E_2, (\sigma_1, P_2), \mathcal{Q}_2, \mathcal{C}_2 \xrightarrow{p_2}_{t_2} \dots \xrightarrow{p'}_{t'} E, (\sigma, P), \mathcal{Q}, \mathcal{C}$ executes no output, so $\sigma = \sigma_1$. (The substitution σ is changed only when executing a communication.) Since $\mathcal{F}_P$ is correct for all P, $E, \sigma \vdash \mathcal{F}_P$, hence $E, \sigma_1 \vdash \mathcal{F}_P$. Then, in both cases, $E, \sigma_1 \vdash \mathcal{F}^{\mathrm{Fut}}_{P_2} \cap \mathcal{F}_P$.

Next, we show **S4**: if $\mathbb{C}_0 \xrightarrow{p}_t \dots \xrightarrow{p'}_{t'} E, (\sigma, P), \mathcal{Q}, \mathcal{C}$ then $E, \sigma \vdash \mathcal{F}^{\mathrm{ElseFind}}_P$. The proof proceeds by induction on the length of the trace. For the initial configuration, the result is obvious since $\mathcal{F}^{\mathrm{ElseFind}}_P = \emptyset$. For the inductive step, if the reduced process is in C, the result is obvious since $\mathcal{F}^{\mathrm{ElseFind}}_P = \emptyset$. Otherwise, we proceed by cases on the last reduction of the trace. In the (Output) case, the result is obvious since $\mathcal{F}^{\mathrm{ElseFind}}_P = \emptyset$. In the (New), (Let), and (Find1) cases, σ is unchanged, E is extended with definitions for some variables, and *elsefind* facts that claim that these variables are not defined are removed from $\mathcal{F}^{\mathrm{ElseFind}}_P$, so we still have $E, \sigma \vdash \mathcal{F}^{\mathrm{ElseFind}}_P$. In the (Find2) case for $P' = \mathsf{find}\ (\bigoplus_{j=1}^m u_{j1}[\widetilde{i}] \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}$ $\mathsf{suchthat}\ \mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P$, E and σ are unchanged and since (Find2) is executed, $\forall j \leq m$, $\forall a_1 \in [1, I_\eta(n_{j1})]$, $\dots$, $\forall a_{m_j} \in [1, I_\eta(n_{jm_j})]$, $E[u_{j1}[\widetilde{\sigma i}] \mapsto a_1, \dots, u_{jm_j}[\widetilde{\sigma i}] \mapsto a_{m_j}], \sigma, (\mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j) \Downarrow$ false. $\mathcal{F}^{\mathrm{ElseFind}}_P = \mathcal{F}^{\mathrm{ElseFind}}_{P'} \cup \{\mathit{elsefind}((u_1 \leq n_{j1}, \dots, u_{m_j} \leq n_{jm_j}), \sigma_j(M_{j1}, \dots, M_{jl_j}), \sigma_j M_j) \mid j \in \{1, \dots, m\}\}$ where $\sigma_j = \{u_1/u_{j1}[\widetilde{i}], \dots, u_{m_j}/u_{jm_j}[\widetilde{i}]\}$. By induction hypothesis $E, \sigma \vdash \mathcal{F}^{\mathrm{ElseFind}}_{P'}$. Moreover, $E, \sigma \vdash \mathit{elsefind}((u_1 \leq n_{j1}, \dots, u_{m_j} \leq n_{jm_j}), \sigma_j(M_{j1}, \dots, M_{jl_j}), \sigma_j M_j)$ for $j \in \{1, \dots, m\}$, so $E, \sigma \vdash \mathcal{F}^{\mathrm{ElseFind}}_P$.

We now show that if $\mathcal{F}_P$ is correct for all P, then $\mathcal{F}'_P$ obtained by

$$\mathcal{F}'_P = \mathcal{F}_P \cup \{\neg\sigma' M \mid \mathit{elsefind}((u_1 \leq n_1, \dots, u_m \leq n_m), (M_1, \dots, M_l), M) \in \mathcal{F}^{\mathrm{ElseFind}}_P, \mathrm{Dom}(\sigma') = \{u_1, \dots, u_m\}, \text{for each } j \in \{1, \dots, l\}, \sigma' M_j \text{ is a subterm of } M'_j \text{ and } \mathsf{defined}(M'_j) \in \mathcal{F}_P\}$$

is also correct for all P. Assuming that $\mathbb{C}_0 \xrightarrow{p}_t \dots \xrightarrow{p'}_{t'} E, (\sigma, P), \mathcal{Q}, \mathcal{C}$, we show that $E, \sigma \vdash \mathcal{F}'_P$. Since $\mathcal{F}_P$ is correct for all P, $E, \sigma \vdash \mathcal{F}_P$. By (S4), $E, \sigma \vdash \mathcal{F}^{\mathrm{ElseFind}}_P$. Assume $\mathit{elsefind}((u_1 \leq n_1, \dots, u_m \leq n_m), (M_1, \dots, M_l), M) \in \mathcal{F}^{\mathrm{ElseFind}}_P$ and for each $j \in \{1, \dots, l\}$, $\sigma' M_j$ is a subterm of M'_j and $\mathsf{defined}(M'_j) \in \mathcal{F}_P$. Let a_k be such that $E, \sigma, \sigma' u_k \Downarrow a_k$ for each $k \in \{1, \dots, m\}$. Let $\sigma'' = \sigma[u_1 \mapsto a_1, \dots, u_m \mapsto a_m]$. Since $E, \sigma \vdash \mathsf{defined}(M'_j)$, we have $E, \sigma, M'_j \Downarrow a'_j$ for some a'_j so $E, \sigma, \sigma' M_j \Downarrow a''_j$ for some a''_j, so $E, \sigma'', M_j \Downarrow a''_j$. (This is proved by induction on M_j.) By definition of *elsefind* facts, $E, \sigma'', (\mathsf{defined}(M_1, \dots, M_l) \wedge M) \Downarrow$ false so $E, \sigma'', M \Downarrow$ false, that is, $E, \sigma, \sigma' M \Downarrow$ false, so $E, \sigma \vdash \neg\sigma' M$. So $E, \sigma \vdash \mathcal{F}'_P$.

Therefore, we conclude that at the end of the computation, $\mathcal{F}_P$ is correct for all P.

Correctness of the local dependency analysis. As above in the correctness of the collection of true facts, we denote by P an occurrence of a process, so that we can distinguish identical subprocesses that occur at several occurrences in a process.

We first show the soundness of the local dependency analysis ignoring modifications in the game performed by $\mathrm{depAnal}$. Then we will show the soundness of the game modifications, that is, that these modifications change the behavior of the game only with negligible probability. Since the game modifications do not change the part of the computation of depend and indep performed before the modification, the $\mathrm{depAnal}$ procedure is equivalent to performing a full dependency analysis without game modification, performing game modification, redoing the whole dependency analysis analysis on the modified game, and so on, until a fixpoint is reached. Therefore, the separate proof of the dependency analysis and the game modifications outlined above is sufficient to prove the correctness of the $\mathrm{depAnal}$ procedure.

We have **S5**: if y is defined only by restrictions and $y \neq x$, then there exists no M such that $(y, M) \in \mathrm{depend}_P(x)$. This property is obvious since the only case in which an element (y, M) is added in $\mathrm{depend}_P(x)$ is in the assignment let $y[\widetilde{i}] : T = M'$ in P', so such an addition cannot happen when y is defined only by restrictions.

For each σ, depend, indep, we define an equivalence relation $\sim_{\sigma,\mathrm{depend},\mathrm{indep}}$ on environments by $E \sim_{\sigma,\mathrm{depend},\mathrm{indep}} E'$ if and only if

- for all $M \in \mathrm{indep}$, for all b, $E, \sigma, M \Downarrow b$ if and only if $E', \sigma, M \Downarrow b$;
- if $\mathrm{depend} \neq \top$, then for all $z[\widetilde{a}]$ such that $z[\widetilde{a}] \neq x[\widetilde{\sigma i}]$ and there exists no $(y, M) \in \mathrm{depend}$ such that $z[\widetilde{a}] = y[\widetilde{\sigma i}]$, $E(z[\widetilde{a}])$ is defined if and only if $E'(z[\widetilde{a}])$ is defined and when they are defined, $E(z[\widetilde{a}]) = E'(z[\widetilde{a}])$ ($\widetilde{i}$ denotes the current replication indices at definition of x);

- and for all y such that $y \neq x$ and y is defined only by restrictions, for all $\widetilde{a}$, $E(y[\widetilde{a}])$ is defined if and only if $E'(y[\widetilde{a}])$ is defined and when they are defined, $E(y[\widetilde{a}]) = E'(y[\widetilde{a}])$.

When $E \sim_{\sigma,\mathrm{depend},\mathrm{indep}} E'$, the environments E and E' differ only by variables that depend on $x[\widetilde{\sigma i}]$, according to the information contained in depend and indep. That is, terms in indep have the same value in E and E' (first item); when $\mathrm{depend} \neq \top$, variables not in depend have the same value in E and E' (second item); variables defined only by restrictions have the same value in E and E' (third item). We abbreviate $\sim_{\sigma,\mathrm{depend}_P(x),\mathrm{indep}_P(x)}$ by $\sim_{\sigma,P}$.

We show **S6**: if M' does not depend on x at P and $E \sim_{\sigma,P} E'$, then $E, \sigma, M' \Downarrow b$ if and only if $E', \sigma, M' \Downarrow b$. This property expresses the correctness of the definition of "M' does not depend on x at P". We prove that if $E, \sigma, M' \Downarrow b$ then $E', \sigma, M' \Downarrow b$, by induction on the derivation that M' does not depend on x at P. The converse follows immediately by swapping the roles of E and E'.

- Case $M' = f(M'_1, \dots, M'_m)$ and for all $j \leq m$, M'_j does not depend on x at P. Since $E, \sigma, M' \Downarrow b$, $E, \sigma, M'_j \Downarrow b_j$ and $I_\eta(f)(b_1, \dots, b_m) = b$ for some $b_1, \dots, b_m$. Hence by induction hypothesis, $E', \sigma, M'_j \Downarrow b_j$, so $E', \sigma, M' \Downarrow b$.
- Case $M' \in \mathrm{indep}_P(x)$. The result comes from the definition of $\sim_{\sigma,P}$.
- Case M' is a replication index. We have $E, \sigma, M' \Downarrow \sigma M'$ and $E', \sigma, M' \Downarrow \sigma M'$, so the result holds.
- Case $M' = y[M'_1, \dots, M'_m]$, $M'_1, \dots, M'_m$ do not depend on x at P', $y \neq x$, and either y is defined only by restrictions or $\mathrm{depend}_P(x) \neq \top$ and $y \neq y'$ for all $(y', M'') \in \mathrm{depend}_P(x)$. Since $E, \sigma, M' \Downarrow b$, $E, \sigma, M'_j \Downarrow b_j$ and $E(y[b_1, \dots, b_k]) = b$ for some $b_1, \dots, b_k$. Hence by induction hypothesis, $E', \sigma, M'_j \Downarrow b_j$. By definition of $\sim_{\sigma,P}$, $E'(y[b_1, \dots, b_k]) = E(y[b_1, \dots, b_k]) = b$, so $E', \sigma, M' \Downarrow b$.

Let us consider the following property **L0**:

1. If $\Pr[\mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C}] > 0$, $\mathrm{depend}_P(x) \neq \top$, and $(y, M) \in \mathrm{depend}_P(x)$, then $E, \sigma, M \Downarrow E(y[\widetilde{\sigma i}])$ where $\widetilde{i}$ denotes the current replication indices at P;
2. If $\Pr[\mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C}] > 0$ and $M \in \mathrm{indep}_P(x)$, then $E, \sigma, M \Downarrow a$ for some a;
3. For each $b \in I_\eta(T)$, for each σ, for each E_0, $\Pr[\exists E, \exists \mathcal{Q}, \exists \mathcal{C}, \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge E \sim_{\sigma,P} E_0 \wedge E(x[\widetilde{\sigma i}]) = b] \leq \frac{1}{|I_\eta(T)|} \Pr[\exists E, \exists \mathcal{Q}, \exists \mathcal{C}, \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge E \sim_{\sigma,P} E_0]$ where $\widetilde{i}$ denotes the current replication indices at the definition of x.

We will show that if $\mathrm{depend}_P(x) \neq \top$, then (L0) holds at P. This property expresses the correctness of the local dependency analysis at P, when $\mathrm{depend}_P(x) \neq \top$. (We will consider the general case below, Property L1.) Item 1 says that, when $(y, M) \in \mathrm{depend}_P(x)$, M evaluates to the contents of y. Item 2 says that, when $M \in \mathrm{indep}_P(x)$, the value of M is always defined at P. Finally, the last item is the most important one: it expresses the independence properties. Essentially, the traces that differ by the value of $x[\widetilde{\sigma i}]$ all have the same probability, and differ only by the values of variables that depend on $x[\widetilde{\sigma i}]$, collected in $\mathrm{depend}_P(x)$, so their environments are related by $\sim_{\sigma,P}$. When the value of $x[\widetilde{\sigma i}]$ is fixed to b, the probability of reaching an environment related to E_0 by $\sim_{\sigma,P}$ is then $\frac{1}{|I_\eta(T)|}$ times the probability of reaching such an environment for any value of $x[\widetilde{\sigma i}]$.

We first show **S7**: if (L0) holds at P with indep instead of $\mathrm{indep}_P(x)$, for all E, σ such that $\Pr[\mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C}] > 0$, $E, \sigma, M' \Downarrow a$ for some a, and M' does not depend on x at P with indep instead of $\mathrm{indep}_P(x)$, then (L0) also holds at P with $\mathrm{indep} \cup \{M'\}$ instead of $\mathrm{indep}_P(x)$. Essentially, this property means that M' can be added to $\mathrm{indep}_P(x)$ when M' does not depend on x at P. Items 1 and 2 of (L0) hold by hypothesis. If $E \sim_{\sigma,\mathrm{depend}_P(x),\mathrm{indep}} E'$, by (S6), $E, \sigma, M' \Downarrow b$ if and only if $E', \sigma, M' \Downarrow b$, so $E \sim_{\sigma,\mathrm{depend}_P(x),\mathrm{indep}\cup\{M'\}} E'$. Conversely, we have obviously: if $E \sim_{\sigma,\mathrm{depend}_P(x),\mathrm{indep}\cup\{M'\}} E'$, then $E \sim_{\sigma,\mathrm{depend}_P(x),\mathrm{indep}} E'$, so $\sim_{\sigma,\mathrm{depend}_P(x),\mathrm{indep}} = \sim_{\sigma,\mathrm{depend}_P(x),\mathrm{indep}\cup\{M'\}}$. This proves Item 3 of (L0), and concludes the proof of (L0).

Next, we prove **S8**: if $\mathrm{depend}_P(x) \neq \top$ then (L0) holds at P, by decreasing induction on the process P. The only cases in which $\mathrm{depend}_P(x) \neq \top$ are as follows:

- P occurs in $P' = \mathsf{new}\ x[\widetilde{i}] : T; P$ where T is a large type. We have $\mathrm{depend}_P(x) = \emptyset$ and $\mathrm{indep}_P(x) = \bigcup_{\mathsf{defined}(M) \in \mathcal{F}_{P'}} \mathrm{subterms}(M)$. Item 1 of (L0) holds trivially. For all traces of non-zero probability that reach P, the last reduction reduces P' by (New), so these traces are all of the form $\mathbb{C}_0 \to^* E', (\sigma, P'), \mathcal{Q}, \mathcal{C} \to E, (\sigma, P), \mathcal{Q}, \mathcal{C}$ where $E = E'[x[\widetilde{\sigma i}] \mapsto a']$ for some $a' \in I_\eta(T)$. Since $\mathcal{F}_P$ is correct for all P, $E', \sigma \vdash \mathcal{F}_{P'}$, so for all $M' \in \mathrm{subterms}(M)$ such that $\mathsf{defined}(M) \in \mathcal{F}_{P'}$, $E', \sigma, M' \Downarrow a$ for some a, hence $E, \sigma, M' \Downarrow a$ since E is an extension of E', which proves Item 2 of (L0). By the semantic rule (New), for all $b \in I_\eta(T)$, $\Pr[\exists E, \exists \mathcal{Q}, \exists \mathcal{C}, \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge E \sim_{\sigma,P} E_0 \wedge E(x[\widetilde{\sigma i}]) = b] = \frac{1}{|I_\eta(T)|} \Pr[\exists E, \exists \mathcal{Q}, \exists \mathcal{C}, \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge E \sim_{\sigma,P} E_0]$ since the condition $E \sim_{\sigma,P} E_0$ does not use the value of $E(x[\widetilde{\sigma i}])$. (The first item of $E \sim_{\sigma,P} E_0$ does not use the value of $E(x[\widetilde{\sigma i}])$ because the elements of $\mathrm{indep}_P(x)$ are all defined in E' and $E'(x[\widetilde{\sigma i}])$ is not defined. The other two items never use $E(x[\widetilde{\sigma i}])$.) Therefore, we obtain Item 3 of (L0).
- P occurs in $P' = \mathsf{new}\ y[\widetilde{i}] : T'; P$ with $y \neq x$. We have $\mathrm{depend}_P(x) = \mathrm{depend}_{P'}(x)$ and $\mathrm{indep}_P(x) = \mathrm{indep}_{P'}(x) \cup \{y[\widetilde{i}]\}$. For all traces of non-zero probability that reach P, the last reduction reduces P' by (New), so these traces are all of the form $\mathbb{C}_0 \to^* E', (\sigma, P'), \mathcal{Q}, \mathcal{C} \to E, (\sigma, P), \mathcal{Q}, \mathcal{C}$ where $E = E'[y[\widetilde{\sigma i}] \mapsto a']$ for some $a' \in I_\eta(T')$. Item 1 of (L0) comes from the induction hypothesis (at P') and the fact that E is an extension of E'. Item 2 of (L0) comes from the induction hypothesis (at P'), the fact that E is an extension of E', and the fact that

$E(y[\sigma\widetilde{i}])$ is defined. Let $E'_0 = E_{0|\overline{y[\sigma\widetilde{i}]}}$ be the environment E_0 restricted to the variables defined at P'.

$$\begin{aligned}
&\Pr\begin{bmatrix}\exists(E,\mathcal{Q},\mathcal{C}), \mathbb{C}_0 \to^* E,(\sigma,P),\mathcal{Q},\mathcal{C}\\ \wedge\, E \sim_{\sigma,P} E_0 \wedge E(x[\sigma\widetilde{i}]) = b\end{bmatrix}\\
&= \frac{1}{|I_\eta(T')|}\Pr\begin{bmatrix}\exists(E',\mathcal{Q},\mathcal{C}), \mathbb{C}_0 \to^* E',(\sigma,P'),\mathcal{Q},\mathcal{C}\\ \wedge\, E' \sim_{\sigma,P'} E'_0 \wedge E(x[\sigma\widetilde{i}]) = b\end{bmatrix}\\
&\le \frac{1}{|I_\eta(T')|}\frac{1}{|I_\eta(T)|}\Pr\begin{bmatrix}\exists(E',\mathcal{Q},\mathcal{C}),\\ \mathbb{C}_0 \to^* E',(\sigma,P'),\mathcal{Q},\mathcal{C}\\ \wedge\, E' \sim_{\sigma,P'} E'_0\end{bmatrix}\\
&\le \frac{1}{|I_\eta(T)|}\Pr\begin{bmatrix}\exists(E,\mathcal{Q},\mathcal{C}), \mathbb{C}_0 \to^* E,(\sigma,P),\mathcal{Q},\mathcal{C}\\ \wedge\, E \sim_{\sigma,P} E_0\end{bmatrix}
\end{aligned}$$

The first step is by the semantic rule (New), the second step by induction hypothesis, and the last step by the semantic rule (New) again. Therefore, we obtain Item 3 of (L0).

- P occurs in $P' = \mathsf{let}\ y[\widetilde{i}] : T' = M\ \mathsf{in}\ P$ with $y \neq x$. For all traces of non-zero probability that reach P, the last reduction reduces P' by (Let), so these traces are all of the form $\mathbb{C}_0 \to^* E',(\sigma,P'),\mathcal{Q},\mathcal{C} \to E,(\sigma,P),\mathcal{Q},\mathcal{C}$ where $E',\sigma,M \Downarrow a'$ and $E = E'[y[\sigma\widetilde{i}] \mapsto a']$. Let $E'_0 = E_{0|\overline{y[\sigma\widetilde{i}]}}$.

  If M does not depend on x at P', we have $\mathrm{depend}_P(x) = \mathrm{depend}_{P'}(x)$ and $\mathrm{indep}_P(x) = \mathrm{indep}_{P'}(x) \cup \{y[\widetilde{i}]\}$. In this case, by (S6), $E',\sigma,M \Downarrow a'$ if and only if $E'_0,\sigma,M \Downarrow a'$ (where $E' \sim_{\sigma,P'} E'_0$ are environments at P'). We can then show that (L0) holds at P using the induction hypothesis. (We have $E \sim_{\sigma,P} E_0$ if and only if $E' \sim_{\sigma,P'} E'_0$ and $E_0 = E'_0[y[\sigma\widetilde{i}] \mapsto a']$.)

  Otherwise, we have $\mathrm{depend}_P(x) = \mathrm{depend}_{P'}(x) \cup \{(y, M\mathrm{depend}_{P'}(x))\}$ and $\mathrm{indep}_P(x) = \mathrm{indep}_{P'}(x)$. By induction hypothesis, for all $(y',M') \in \mathrm{depend}_{P'}(x)$, $E',\sigma,M' \Downarrow E'(y'[\sigma\widetilde{i}])$, so $E,\sigma,M' \Downarrow E(y'[\sigma\widetilde{i}])$, hence $E,\sigma,M\mathrm{depend}_{P'}(x) \Downarrow a' = E(y[\sigma\widetilde{i}])$, so we obtain Item 1 of (L0). Item 2 of (L0) follows immediately from the induction hypothesis. Item 3 of (L0) also follows from the induction hypothesis. (We have $E \sim_{\sigma,P} E_0$ if and only if $E' \sim_{\sigma,P'} E'_0$.)

- P occurs in $P' = \mathsf{find}\ (\bigoplus_{j=1}^m u_{j1}[\widetilde{i}] \le n_{j1},\dots,u_{jm_j}[\widetilde{i}] \le n_{jm_j}\ \mathsf{suchthat}\ \mathsf{defined}(M_{j1},\dots,M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P''$, P is either P'' or P_j for some $j \le m$, and for all j,k, M_{jk} and M'_j do not depend on x at P'. We have $\mathrm{depend}_P(x) = \mathrm{depend}_{P'}(x)$, $\mathrm{indep}_P(x) = \mathrm{indep}_{P'}(x)$ if $P = P''$, and $\mathrm{indep}_P(x) = \mathrm{indep}_{P'}(x) \cup \{M' \mid M' \in \mathrm{subterms}(M)$ for some $\mathsf{defined}(M) \in \mathcal{F}_{P_j}$, M' does not depend on x at $P'\}$ if $P = P_j$. By (S6), we can show that the same branch of the find is taken with the same probability for all E such that $E \sim_{\sigma,P'} E_0$ for the same E_0. Using the induction hypothesis, we can then show that (L0) holds at P with $\mathrm{indep}_{P'}(x)$ instead of $\mathrm{indep}_P(x)$. This concludes the proof when $P = P''$. When $P = P_j$, let $M''_1,\dots,M''_l$ be the terms M' such that $M' \in \mathrm{subterms}(M)$ for some $\mathsf{defined}(M) \in \mathcal{F}_{P_j}$ and M' does not depend on x at P'. Since $\mathcal{F}_{P_j}$ is correct and $\Pr[\mathbb{C}_0 \to^* E,(\sigma,P),\mathcal{Q},\mathcal{C}] = p > 0$ then $E,\sigma \vdash \mathcal{F}_{P_j}$, so $E,\sigma,M''_k \Downarrow a$ for some a. The term M''_k does not depend on x at P with $\mathrm{indep}_{P'}(x) \cup \{M''_1,\dots,M''_{k-1}\}$ instead of $\mathrm{indep}_P(x)$. By (S7) applied at P with $\mathrm{indep}_{P'}(x) \cup \{M''_1,\dots,M''_{k-1}\}$ instead of $\mathrm{indep}_P(x)$, if (L0) holds at P with $\mathrm{indep}_{P'}(x) \cup \{M''_1,\dots,M''_{k-1}\}$, then (L0) holds at P with $\mathrm{indep}_{P'}(x) \cup \{M''_1,\dots,M''_k\}$. So (L0) holds at P with $\mathrm{indep}_{P'}(x) \cup \{M''_1,\dots,M''_l\} = \mathrm{indep}_P(x)$.

For each σ, P, we define a special semantics of processes. This semantics executes the process $C[Q_0]$ normally until it reaches a configuration $E',(\sigma',P'),\mathcal{Q},\mathcal{C}$ such that P' is the smallest superprocess of P such that $\mathrm{depend}_{P'}(x) \neq \top$ and $\sigma'(i) = \sigma(i)$ for all $i \in \mathrm{Dom}(\sigma')$. After reaching this configuration, it executes restrictions for all variables defined only by restrictions in $C[Q_0]$ that have not been assigned yet and executes the not-executed-yet restrictions and the assignments $P_1 = \mathsf{let}\ y : T = M\ \mathsf{in}\ P_2$ such that M does not depend on x at P_1 between P' and P. In the second part of the trace, a configuration is only $E'',(\sigma'',P'')$; σ'' is always set to be σ restricted to the current replication indices at P''. We write $\mathbb{C}_0 {\to'}^* E,(\sigma,P)$ to designate a trace in this special semantics. (When $\mathrm{depend}_P(x) \neq \top$, this semantics executes the process normally, and finally executes restrictions for all variables defined only by restrictions that have not been assigned yet.)

We will show the following property **L1**:

1. If $\Pr[\mathbb{C}_0 \to^* E,(\sigma,P),\mathcal{Q},\mathcal{C}] > 0$, $\mathrm{depend}_P(x) \neq \top$, and $(y,M) \in \mathrm{depend}_P(x)$, then $E,\sigma,M \Downarrow E(y[\sigma\widetilde{i}])$ where $\widetilde{i}$ denotes the current replication indices at P;

2. If $\Pr[\mathbb{C}_0 \to^* E,(\sigma,P),\mathcal{Q},\mathcal{C}] > 0$ and $M \in \mathrm{indep}_P(x)$, then $E,\sigma,M \Downarrow a$ for some a;

3. For each $b \in I_\eta(T)$, for each σ, for each E_0, $\Pr[\exists(E,\mathcal{Q},\mathcal{C}), \mathbb{C}_0 \to^* E,(\sigma,P),\mathcal{Q},\mathcal{C} \wedge E \sim_{\sigma,P} E_0 \wedge E(x[\sigma\widetilde{i}]) = b] \le \frac{1}{|I_\eta(T)|}\Pr[\exists E_1, \mathbb{C}_0 {\to'}^* E_1,(\sigma,P) \wedge E_{1|\mathrm{Dom}(E_0)} \sim_{\sigma,P} E_0]$ where $\widetilde{i}$ denotes the current replication indices at the definition of x.

Property (L1) expresses the correctness of the local dependency analysis at P. It differs from (L0) by the use of the special semantics $\to'$ in Item 3. This semantics is necessary when $\mathrm{depend}_P(x) = \top$, because in that case the control-flow may also depend on the value of $x[\sigma\widetilde{i}]$, so P may not be reachable for certain values of $x[\sigma\widetilde{i}]$, which breaks the inequality between probabilities of (L0), Item 3. In contrast, the special semantics $\to'$ computes $E_1,(\sigma,P)$ without taking into account the control-flow, so this problem is avoided.

Property (S7) also holds for (L1), with the same proof as for (L0).

We show **S9**: if (L0) holds at P, then (L1) holds at P. Let E_1 be E extended with values for all variables defined only by restrictions. If $E \sim_{\sigma,P} E_0$, the variables defined only by restrictions are defined for the same indices in E and in E_0, so $E_{1|\mathrm{Dom}(E_0)} = E$, hence $E_{1|\mathrm{Dom}(E_0)} \sim_{\sigma,P} E_0$. Therefore, $\Pr[\exists(E,\mathcal{Q},\mathcal{C}), \mathbb{C}_0 \to^* E,(\sigma,P),\mathcal{Q},\mathcal{C} \wedge E \sim_{\sigma,P} E_0] \le \Pr[\exists E_1, \mathbb{C}_0 {\to'}^* E_1,(\sigma,P) \wedge E_{1|\mathrm{Dom}(E_0)} \sim_{\sigma,P} E_0]$, which proves (L1).

We show **S9'**: if (L0) holds at P, then (L1) holds at P with $\sim_{\sigma,\top,\mathrm{indep}_P(x)}$ instead of $\sim_{\sigma,P}$. If $E \sim_{\sigma,P} E'$, then $E \sim_{\sigma,\top,\mathrm{indep}_P(x)} E'$. So each equivalence class of $\sim_{\sigma,\top,\mathrm{indep}_P(x)}$ is a union of equivalence classes of $\sim_{\sigma,P}$. So we obtain (L0) with $\sim_{\sigma,\top,\mathrm{indep}_P(x)}$ instead of $\sim_{\sigma,P}$ by adding probabilities. We conclude that (L1) holds at P with $\sim_{\sigma,\top,\mathrm{indep}_P(x)}$ instead of $\sim_{\sigma,P}$ using a proof similar to that of (S9).

We show **S10**: If P is an output process, P' is the smallest output process such that P is a strict subprocess of P', (L1) holds at P' with $\sim_{\sigma,\top,\mathrm{indep}_{P'}(x)}$ instead of $\sim_{\sigma,P'}$, and $\mathrm{depend}_P(x) = \top$, then (L1) holds at P with $\mathrm{indep}_{P'}(x)$ instead of $\mathrm{indep}_P(x)$. The equivalence between environments for (L1) at P with $\mathrm{indep}_{P'}(x)$ instead of $\mathrm{indep}_P(x)$ is also $\sim_{\sigma,\top,\mathrm{indep}_{P'}(x)}$, since $\mathrm{depend}_P(x) = \top$. Item 1 of (L1) holds trivially at P since $\mathrm{depend}_P(x) = \top$. For the proof of Item 3 of (L1), we let $p = \Pr[\exists(E, \mathcal{Q}, \mathcal{C}), \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge E \sim_{\sigma,\top,\mathrm{indep}_{P'}(x)} E_0 \wedge E(x[\sigma\widetilde{i}]) = b]$.

- Case $P' = \mathsf{let}\ y[\widetilde{i}] : T' = M \mathsf{\ in\ } P$. In traces of non-zero probability that reach P, the last reduction of the trace reduces P' by (Let), so these traces are all of the form:

  $$\mathbb{C}_0 \to^* E', (\sigma, P'), \mathcal{Q}, \mathcal{C} \to E, (\sigma, P), \mathcal{Q}, \mathcal{C}$$

  where $E', \sigma, M \Downarrow a$ and $E = E'[y[\sigma\widetilde{i}] \mapsto a]$ and the corresponding trace of $\to'$ is

  $$\mathbb{C}_0 {\to'}^* E'_1, (\sigma, P') \to' E_1, (\sigma, P)$$

  where $E'_1, \sigma, M \Downarrow a'$ and $E_1 = E'_1[y[\sigma\widetilde{i}] \mapsto a']$. Let $E'_0 = E_{0|\overline{y[\sigma\widetilde{i}]}}$ be the environment E_0 restricted to the variables defined at P'. For all $M' \in \mathrm{indep}_{P'}(x)$, $E_1, \sigma, M' \Downarrow b$ for some b since (L1) holds at P'. Then $E, \sigma, M' \Downarrow b$, so Item 2 of (L1) holds at P with $\mathrm{indep}_{P'}(x)$ instead of $\mathrm{indep}_P(x)$. Since all elements of $\mathrm{indep}_{P'}(x)$ must be defined at P' (by Item 2 of (L1) at P'), $y[\sigma\widetilde{i}]$ is not defined at P', and y is not defined only by restrictions, the condition $E \sim_{\sigma,\top,\mathrm{indep}_{P'}(x)} E_0$ in Item 3 of (L1) at P with $\mathrm{indep}_{P'}(x)$ instead of $\mathrm{indep}_P(x)$ does not use the value of $E(y[\sigma\widetilde{i}])$, hence $E \sim_{\sigma,\top,\mathrm{indep}_{P'}(x)} E_0$ if and only if $E' \sim_{\sigma,\top,\mathrm{indep}_{P'}(x)} E'_0$, and $E_{1|\mathrm{Dom}(E_0)} \sim_{\sigma,\top,\mathrm{indep}_{P'}(x)} E_0$ if and only if $E'_{1|\mathrm{Dom}(E'_0)} \sim_{\sigma,\top,\mathrm{indep}_{P'}(x)} E'_0$, so the probabilities that occur in Item 3 of (L1) are the same for P' and for P with $\mathrm{indep}_{P'}(x)$ instead of $\mathrm{indep}_P(x)$. Therefore, Item 3 of (L1) holds at P with $\mathrm{indep}_{P'}(x)$ instead of $\mathrm{indep}_P(x)$.

- Case $P' = \mathsf{new}\ y[\widetilde{i}] : T'; P$, where y is not defined only by restrictions. In traces of non-zero probability that reach P, the last reduction of the trace reduces P' by (New). This case is similar to the let case above.

- Case $P' = \mathsf{new}\ y[\widetilde{i}] : T'; P$, where y is defined only by restrictions. In traces of non-zero probability that reach P, the last reduction of the trace reduces P' by (New). Item 2 is proved as in the let case above. Let us consider Item 3. Let $E'_0 = E_{0|\overline{y[\sigma\widetilde{i}]}}$ be the environment E_0 restricted to the variables defined at P'. Let $\widetilde{i}'$ be the replication indices at the definition of x. ($\widetilde{i}'$ is a prefix of $\widetilde{i}$.)

  $$\begin{aligned}
  p &= \Pr\begin{bmatrix}\exists(E, E', \mathcal{Q}, \mathcal{C}),\\ \mathbb{C}_0 \to^* E', (\sigma, P'), \mathcal{Q}, \mathcal{C} \to E, (\sigma, P), \mathcal{Q}, \mathcal{C}\\ \wedge E \sim_{\sigma,\top,\mathrm{indep}_{P'}(x)} E_0 \wedge E(x[\sigma\widetilde{i}']) = b\end{bmatrix}\\
  &= \frac{1}{|I_\eta(T')|}\Pr\begin{bmatrix}\exists(E', \mathcal{Q}, \mathcal{C}), \mathbb{C}_0 \to^* E', (\sigma, P'), \mathcal{Q}, \mathcal{C}\\ \wedge E' \sim_{\sigma,\top,\mathrm{indep}_{P'}(x)} E'_0\\ \wedge E'(x[\sigma\widetilde{i}']) = b\end{bmatrix}\\
  &\leq \frac{1}{|I_\eta(T')|}\frac{1}{|I_\eta(T)|}\Pr\begin{bmatrix}\exists E'_1, \mathbb{C}_0 {\to'}^* E'_1, (\sigma, P') \wedge\\ E'_{1|\mathrm{Dom}(E'_0)} \sim_{\sigma,\top,\mathrm{indep}_{P'}(x)} E'_0\end{bmatrix}\\
  &\leq \frac{1}{|I_\eta(T)|}\Pr\begin{bmatrix}\exists E_1, \mathbb{C}_0 {\to'}^* E_1, (\sigma, P)\\ \wedge E_{1|\mathrm{Dom}(E_0)} \sim_{\sigma,\top,\mathrm{indep}_{P'}(x)} E_0\end{bmatrix}
  \end{aligned}$$

  The first step comes from the semantic rule (New), the second step from (L1) at P', the last step from the assignment of variables defined only by restrictions in the special $\to'$ semantics. (Note that $E'_1 = E_1$, but the condition $E'_{1|\mathrm{Dom}(E'_0)} \sim_{\sigma,\top,\mathrm{indep}_{P'}(x)} E'_0$ does not use the value of $E'_1(y[\sigma\widetilde{i}])$.) This inequality proves (L1) at P with $\mathrm{indep}_{P'}(x)$ instead of $\mathrm{indep}_P(x)$.

- Cases in which there is no assignment and no restriction between P and P'. Everything that is defined at P' is also defined at P, since the environment at P is an extension of the environment at P', so Item 2 of (L1) holds at P since it holds at P'. Let us now prove Item 3 of (L1). The final environment E' of the $\to'$ trace is the same for P and for P', so the right-hand side of the inequality is the same for P and for P'. The left-hand side decreases from P' to P, since all traces that reach P must first have reached P', so the inequality still holds.

From the previous results, we show that (L1) holds at all output processes P. The proof proceeds by decreasing induction on P. If $\mathrm{depend}_P(x) \neq \top$, we have the result using (S8) and (S9). Otherwise, let P' be the smallest output process such that P is a strict subprocess of P'. If $\mathrm{depend}_{P'}(x) \neq \top$, by (S8) and (S9'), (L1) holds at P' with $\sim_{\sigma,\top,\mathrm{indep}_{P'}(x)}$ instead of $\sim_{\sigma,P'}$. If $\mathrm{depend}_{P'}(x) = \top$, by induction hypothesis, (L1) holds at P', that is, (L1) holds at P' with $\sim_{\sigma,\top,\mathrm{indep}_{P'}(x)}$ instead of $\sim_{\sigma,P'}$. In both cases, by (S10), (L1) holds at P with $\mathrm{indep}_{P'}(x)$ instead of $\mathrm{indep}_P(x)$. The only cases in which $\mathrm{indep}_{P'}(x) \neq \mathrm{indep}_P(x)$ are as follows:

- Case $P' = \mathsf{new}\ y[\widetilde{i}] : T'; P$, $y \neq x$, $\mathrm{indep}_P(x) = \mathrm{indep}_{P'}(x) \cup \{y[\widetilde{i}]\}$. When y is defined only by restrictions, $y[\widetilde{i}]$ does not depend on x at P with $\mathrm{indep}_{P'}(x)$ instead of $\mathrm{indep}_P(x)$, so, by (S7), (L1) holds at P. Otherwise, in traces of non-zero probability that reach P, the last reduction of the trace reduces P' by (New), so these traces are all of the form:

  $$\mathbb{C}_0 \to^* E', (\sigma, P'), \mathcal{Q}, \mathcal{C} \to E, (\sigma, P), \mathcal{Q}, \mathcal{C}$$

  where $E = E'[y[\sigma\widetilde{i}] \mapsto a]$ for some $a \in I_\eta(T')$. So Item 2 of (L1) holds at P. Let $E'_0 = E_{0|\overline{y[\sigma\widetilde{i}]}}$. Let $\widetilde{i}'$ be the replication indices at the definition of x. ($\widetilde{i}'$ is a prefix of $\widetilde{i}$.) We prove Item 3 of (L1) as follows:

$$
\begin{aligned}
p &= \Pr\begin{bmatrix}\exists(E, E', \mathcal{Q}, \mathcal{C}),\\ \mathbb{C}_0 \to^* E', (\sigma, P'), \mathcal{Q}, \mathcal{C} \to E, (\sigma, P), \mathcal{Q}, \mathcal{C}\\ \wedge E \sim_{\sigma,\top,\mathrm{indep}_P(x)} E_0 \wedge E(x[\sigma\widetilde{i'}]) = b\end{bmatrix}\\
&= \frac{1}{|I_\eta(T')|}\Pr\begin{bmatrix}\exists(E', \mathcal{Q}, \mathcal{C}), \mathbb{C}_0 \to^* E', (\sigma, P'), \mathcal{Q}, \mathcal{C}\\ \wedge E' \sim_{\sigma,\top,\mathrm{indep}_{P'}(x)} E'_0\\ \wedge E'(x[\sigma\widetilde{i'}]) = b\end{bmatrix}\\
&\le \frac{1}{|I_\eta(T')|}\frac{1}{|I_\eta(T)|}\Pr\begin{bmatrix}\exists E'_1, \mathbb{C}_0 {\to'}^* E'_1, (\sigma, P') \wedge\\ E'_{1|\mathrm{Dom}(E'_0)} \sim_{\sigma,\top,\mathrm{indep}_{P'}(x)} E'_0\end{bmatrix}\\
&\le \frac{1}{|I_\eta(T)|}\Pr\begin{bmatrix}\exists(E_1, E'_1),\\ \mathbb{C}_0 {\to'}^* E'_1, (\sigma, P') \to' E_1, (\sigma, P)\\ \wedge E_{1|\mathrm{Dom}(E_0)} \sim_{\sigma,\top,\mathrm{indep}_P(x)} E_0\end{bmatrix}
\end{aligned}
$$

The first step comes from the semantic rule (New), the second step from (L1) at P', the last step from the special $\to'$ semantics of new. This inequality proves (L1) at P.

- Case $P' = \mathsf{find}\ (\bigoplus_{j=1}^m u_{j1}[\widetilde{i}] \le n_{j1}, \dots, u_{jm_j}[\widetilde{i}] \le n_{jm_j}$ $\mathsf{suchthat\ defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}$ P'', $\mathrm{depend}_P(x) = \mathrm{depend}_{P'}(x) = \top$, $P = P_j$, $\mathrm{indep}_P(x) = \mathrm{indep}_{P'}(x) \cup \{M' \mid M' \in \mathrm{subterms}(M)$ for some $\mathsf{defined}(M) \in \mathcal{F}_{P_j}$, M' does not depend on x at $P'\}$. For all M' such that $M' \in \mathrm{subterms}(M)$ for some $\mathsf{defined}(M) \in \mathcal{F}_{P_j}$ and M' does not depend on x at P', M' does not depend on x at P with $\mathrm{indep}_{P'}(x)$ instead of $\mathrm{indep}_P(x)$. Since $\mathcal{F}_P$ is correct for all P, for all E, σ such that $\Pr[\mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C}] > 0$, we have $E, \sigma \vdash \mathcal{F}_P$, so $E, \sigma, M' \Downarrow a$ for some a. So, by (S7), (L1) holds at P.
- Case $P' = \mathsf{let}\ y[\widetilde{i}] : T' = M\ \mathsf{in}\ P$, $y \neq x$, M does not depend on x at P'. The term M does not depend on x at P with $\mathrm{indep}_{P'}(x)$ instead of $\mathrm{indep}_P(x)$. By (S7), (L1) holds at P with $\mathrm{indep}_{P'}(x) \cup \{M\}$ instead of $\mathrm{indep}_P(x)$. In all traces (of non-zero probability) considered in (L1), we have $E, \sigma, y[\widetilde{i}] \Downarrow b$ if and only if $E, \sigma, M \Downarrow b$ and $E_1, \sigma, y[\widetilde{i}] \Downarrow b$ if and only if $E_1, \sigma, M \Downarrow b$, so (L1) holds at P with $\mathrm{indep}_P(x) = \mathrm{indep}_{P'}(x) \cup \{y[\widetilde{i}]\}$.

This result concludes the proof of soundness of the dependency analysis.

We now show the soundness of $\mathrm{simplifyTerm}$. Essentially, when M simplifies to M', M and M' evaluate to the same value except in cases of negligible probability. More precisely, we show **S11**: for each P, M, M', if $M' = \mathrm{simplifyTerm}(M, P)$, then $\Pr[\exists(E, \sigma, \mathcal{Q}, \mathcal{C}), \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge E, \sigma, (M' = M) \Downarrow \mathrm{false}] \le q'(\eta) p_{\max}(\eta)$ for some polynomial q'. The proof proceeds by induction on the derivation that $M' = \mathrm{simplifyTerm}(M, P)$. We only consider the case $\mathrm{simplifyTerm}(M_1 = M_2, P) = \mathrm{false}$; the other cases are similar or easy. We show that if $\mathrm{simplifyTerm}(M_1 = M_2, P) = \mathrm{false}$ then $p = \Pr[\exists(E, \sigma, \mathcal{Q}, \mathcal{C}), \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge E, \sigma, (M_1 = M_2) \Downarrow \mathrm{true}] \le q'(\eta) p_{\max}(\eta)$ for some polynomial q'. When $\mathrm{depend}_P(x) = \top$, let $M_0 = M_1$; otherwise, let $M_0 = M_1 \mathrm{depend}_P(x)$. Let M'_0 and M'_2 be obtained respectively from M_0 and M_2 by replacing all array indices that depend on x at P with fresh replication indices. We assume that M'_0 characterizes a part of $x[\widetilde{i}]$ at P, and M'_2 does not depend on x at P.

Let σ and σ' be fixed, such that σ' is an extension of σ to the fresh replication indices of M'_0 and M'_2. We denote by $\overline{E}$ equivalence classes for $\sim_{\sigma,P} = \sim_{\sigma',P}$. We show that for all a, for all $\overline{E}$, there exists b such that for all $E \in \overline{E}$, if $E, \sigma', M'_0 \Downarrow a$, then $E, \sigma', f_1(\dots f_k(x[\widetilde{i}])) \Downarrow b$.

- Assume that there exists $E' \in \overline{E}$ such that $E', \sigma', M'_0 \Downarrow a$. We define an environment E'' by $E''(y[\widetilde{a}]) = E(y[\widetilde{a}])$ for all $y[\widetilde{a}] \in \mathrm{Dom}(E)$ and $E''((\alpha y)[\widetilde{a}]) = E'(y[\widetilde{a}])$ for variables y renamed to fresh variables by α. We have $E''((\alpha y)[\widetilde{a}]) = E'(y[\widetilde{a}])$ for all $y[\widetilde{a}] \in \mathrm{Dom}(E')$, since when $\alpha y = y$, $E'(y[\widetilde{a}]) = E(y[\widetilde{a}])$ since $E \sim_{\sigma,P} E'$. Hence $E'', \sigma', M'_0 \Downarrow a$ and $E'', \sigma', \alpha M'_0 \Downarrow a$, so $E'', \sigma', (\alpha M'_0 = M'_0) \Downarrow \mathrm{true}$. So by rewriting, $E'', \sigma', (f_1(\dots f_k((\alpha x)[\widetilde{i}])) = f_1(\dots f_k(x[\widetilde{i}]))) \Downarrow \mathrm{true}$. Let b such that $E'', \sigma', f_1(\dots f_k((\alpha x)[\widetilde{i}])) \Downarrow b$. Then $E'', \sigma', f_1(\dots f_k(x[\widetilde{i}])) \Downarrow b$.
- Otherwise, there exists no $E \in \overline{E}$ such that $E, \sigma', M'_0 \Downarrow a$, so the result holds trivially.

So there exists a function f such that for all a, for all $\overline{E}$, for all $E \in \overline{E}$, if $E, \sigma', M'_0 \Downarrow a$, then $E, \sigma', f_1(\dots f_k(x[\widetilde{i}])) \Downarrow f(a, \sigma', \overline{E})$.

If $E, \sigma, (M_1 = M_2) \Downarrow \mathrm{true}$ and $E \in \overline{E}$, $E, \sigma, M_1 \Downarrow a$ and $E, \sigma, M_2 \Downarrow a$ for some a. Then $E, \sigma, M_0 \Downarrow a$ by Item 1 of (L1). So there exists an extension σ' of σ to the fresh replication indices of M'_0 and M'_2 such that $E, \sigma', M'_0 \Downarrow a$ and $E, \sigma', M'_2 \Downarrow a$. Then $E, \sigma', f_1(\dots f_k(x[\widetilde{i}])) \Downarrow f(a, \sigma', \overline{E})$. Since $E, \sigma', M'_2 \Downarrow a$ and M'_2 does not depend on x at P, by (S6), we have $a = f'(\sigma', \overline{E})$ for some function f', hence $E(x[\sigma\widetilde{i}]) \in S_x(\sigma', \overline{E}) = (I_\eta(f_1) \circ \dots \circ I_\eta(f_k))^{-1}(f(f'(\sigma', \overline{E}), \sigma', \overline{E}))$. Let $T_1, \dots, T_k$ be the types of the arguments of $f_1, \dots, f_k$ respectively; let $T_0 = T'$ be the type of the result of f_1; $T_k = T$. We have $|S_x(\sigma', \overline{E})| \le \frac{|I_\eta(T_1)|}{|I_\eta(T_0)|} \times \dots \times \frac{|I_\eta(T_k)|}{|I_\eta(T_{k-1})|} = \frac{|I_\eta(T_k)|}{|I_\eta(T_0)|} = \frac{|I_\eta(T)|}{|I_\eta(T')|}$, since $f_1, \dots, f_k$ are uniform. Let $\widetilde{i'} = \mathrm{Dom}(\sigma)$ be the current replication indices at P.

$$
\begin{aligned}
p &= \Pr\begin{bmatrix}\exists(E, \sigma, \mathcal{Q}, \mathcal{C}), \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C}\\ \wedge E, \sigma, (M_1 = M_2) \Downarrow \mathrm{true}\end{bmatrix}\\
&\le \sum_{\overline{E}}\sum_{\sigma'}\Pr\begin{bmatrix}\exists(E, \mathcal{Q}, \mathcal{C}), \mathbb{C}_0 \to^* E, (\sigma'_{|\widetilde{i'}}, P), \mathcal{Q}, \mathcal{C}\\ \wedge E \in \overline{E} \wedge E(x[\sigma'\widetilde{i}]) \in S_x(\sigma', \overline{E})\end{bmatrix}\\
&\le \sum_{\overline{E}}\sum_{\sigma'}\sum_{b \in S_x(\sigma', \overline{E})}\Pr\begin{bmatrix}\exists(E, \mathcal{Q}, \mathcal{C}),\\ \mathbb{C}_0 \to^* E, (\sigma'_{|\widetilde{i'}}, P), \mathcal{Q}, \mathcal{C}\\ \wedge E \in \overline{E} \wedge E(x[\sigma'\widetilde{i}]) = b\end{bmatrix}\\
&\le \sum_{\overline{E}}\sum_{\sigma'}\sum_{b \in S_x(\sigma', \overline{E})}\frac{1}{|I_\eta(T)|}\Pr\begin{bmatrix}\exists E', \mathbb{C}_0 {\to'}^* E', (\sigma'_{|\widetilde{i'}}, P)\\ \wedge E'_{|\mathrm{Dom}(\overline{E})} \in \overline{E}\end{bmatrix}
\end{aligned}
$$

by Item 3 of (L1). ($\mathrm{Dom}(\overline{E})$ denotes the domain of an element of $\overline{E}$, for instance the smallest one.)

$$
\begin{aligned}
p &\le \frac{1}{|I_\eta(T')|}\sum_{\sigma'}\sum_{\overline{E}}\Pr\begin{bmatrix}\exists E', \mathbb{C}_0 {\to'}^* E', (\sigma'_{|\widetilde{i'}}, P)\\ \wedge E'_{|\mathrm{Dom}(\overline{E})} \in \overline{E}\end{bmatrix}\\
&\le \frac{q_1(\eta)}{|I_\eta(T')|}
\end{aligned}
$$

where $q_1(\eta)$ is the number of possible σ', which is polynomial in η.

We now show the correctness of the game simplifications performed in $\mathrm{depAnal}$. If Q_0 is the process before simplification and Q'_0 the process after simplification, we show that $Q_0 \approx^V Q'_0$. For simplicity, we consider one transformation at a time, and use transitivity of $\approx^V$ to conclude when several transformations are applied. For each trace $\mathrm{initConfig}(C[Q_0]) \to^* E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m$, except in cases of negligible probability, we show that there exists a corresponding trace $\mathrm{initConfig}(C[Q'_0]) \to^* E'_{m'}, P'_{m'}, \mathcal{Q}'_{m'}, \mathcal{C}'_{m'}$ with $E'_{m'} = E_m$, $P'_{m'}$ is obtained from P_m by the same transformation as Q'_0 from Q_0, $\mathcal{Q}'_{m'}$ is obtained from $\mathcal{Q}_m$ by the same transformation as Q'_0 from Q_0, $\mathcal{C}'_{m'} = \mathcal{C}_m$, with the same probability. The proof proceeds by induction on m. The case $m = 0$ is obvious, since the game simplifications do not change input processes. For the inductive step, we reason by cases on the last reduction step of the trace of $C[Q_0]$. We consider only the cases in which the transition may be altered by the game simplification.

- Case 1: When $\mathrm{simplifyTerm}(M, P) = M'$, we replace M with M' in P. We exclude traces such that $E, \sigma \not\vdash M = M'$. (They have negligible probability by (S11).) In the remaining traces, $E, \sigma \vdash M = M'$. So $E, \sigma, M \Downarrow a$ if and only if $E, \sigma, M' \Downarrow a$, and the transformed process reduces in the same way as the initial process.

- Case 2: When $M_j = \mathrm{false}$, we remove the j-th branch of find $(\bigoplus_{j=1}^m u_{j1}[\widetilde{i}] \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}$ $\mathsf{suchthat}$ $\mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j$ then $P_j)$ else P' In all traces $E, \sigma, (\mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j) \Downarrow \mathrm{false}$, so in the reduction rule (Find1), the set S never contains $(j, \widetilde{v})$ for any $\widetilde{v}$, hence by (Find1) or (Find2), the process takes the same branch of the find with the same probability, whether or not the j-th branch is present.

- The other cases are similar.

We also show the converse property: for each trace of $C[Q'_0]$, except in cases of negligible probability, there exists a corresponding trace of $C[Q_0]$ with the same probability. Moreover, for all channels c and bitstrings a, $E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m$ executes $\overline{c}\langle a\rangle$ immediately if and only if $E'_{m'}, P'_{m'}, \mathcal{Q}'_{m'}, \mathcal{C}'_{m'}$ executes $\overline{c}\langle a\rangle$ immediately, so $\Pr[C[Q_0] \rightsquigarrow_\eta \overline{c}\langle a\rangle] = \Pr[C[Q'_0] \rightsquigarrow_\eta \overline{c}\langle a\rangle]$, which yields the desired equivalence $Q_0 \approx^V Q'_0$.

Correctness of the equational prover. We say that $E, \sigma \vdash (\mathcal{F}, \mathcal{R})$ when $E, \sigma \vdash \mathcal{F}$ and for all $(M_1 \to M_2) \in \mathcal{R}$, $E, \sigma \vdash M_1 = M_2$. For each P, the equational prover rewrites pairs $\mathcal{F}, \mathcal{R}$ starting from $(\mathcal{F}_P, \emptyset)$ according to a certain sequence. We denote by $(\mathcal{F}_j, \mathcal{R}_j)(P)$ the j-th element of this sequence. So we have $(\mathcal{F}_0, \mathcal{R}_0)(P) = (\mathcal{F}_P, \emptyset)$, and for all j, we have $\frac{(\mathcal{F}_{j-1}, \mathcal{R}_{j-1})(P)}{(\mathcal{F}_j, \mathcal{R}_j)(P)}$. Let $p_{m'}(P) = \Pr[\exists (E, \sigma, \mathcal{Q}, \mathcal{C}), \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge E, \sigma \not\vdash (\mathcal{F}_{m'}, \mathcal{R}_{m'})(P)]$. We show **S12**: for each P, $p_{m'}(P) \leq q'(\eta) p_{\max}(\eta)$ for some polynomial q'. We first prove (S12) for the case in which we use only the first kind of user-defined rewrite rules of Section C.1. The proof proceeds by induction on m'. For $m' = 0$, this is an immediate consequence of the property that $E, \sigma \vdash (\mathcal{F}_0, \mathcal{R}_0)(P) = (\mathcal{F}_P, \emptyset)$ since $\mathcal{F}_P$ is correct for all P, with $q'(\eta) = 0$. For the inductive step,

$$p_{m'}(P) \leq p_{m'-1}(P) + \Pr \begin{bmatrix} \exists (E, \sigma, \mathcal{Q}, \mathcal{C}), \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \\ \wedge E, \sigma \vdash (\mathcal{F}_{m'-1}, \mathcal{R}_{m'-1})(P) \\ \wedge E, \sigma \not\vdash (\mathcal{F}_{m'}, \mathcal{R}_{m'})(P) \end{bmatrix}$$

By induction hypothesis, $p_{m'-1}(P) \leq q'(\eta) p_{\max}(\eta)$ for some polynomial q'. So we just have to show that if $\frac{\mathcal{F}, \mathcal{R}}{\mathcal{F}', \mathcal{R}'}$ then $\Pr[\exists (E, \sigma, \mathcal{Q}, \mathcal{C}), \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge E, \sigma \vdash (\mathcal{F}, \mathcal{R}) \wedge E, \sigma \not\vdash (\mathcal{F}', \mathcal{R}')] \leq q'(\eta) p_{\max}(\eta)$ for some polynomial q'. We proceed by cases on the derivation of $\frac{\mathcal{F}, \mathcal{R}}{\mathcal{F}', \mathcal{R}'}$.

- The cases (2), (5), (7), as well as the cases (1) and (6) when the reduction uses a rule of $\mathcal{R}$ or a user-defined rule of the first kind, are obvious and there is no loss of probability (that is, $q'(\eta) = 0$.)

- Case (3): Assume that $E, \sigma \vdash (\mathcal{F}, \mathcal{R})$ and $E, \sigma \not\vdash (\mathcal{F}', \mathcal{R}')$. So for all $j \leq m$, $E, \sigma, M_j \Downarrow a_j$, $E, \sigma, M'_j \Downarrow a'_j$, $(a_1, \dots, a_m) \neq (a'_1, \dots, a'_m)$, and $E(x[a_1, \dots, a_m]) = E(x[a_1, \dots, a_m])$. Since for each $a_1, \dots, a_m$, $x[a_1, \dots, a_m]$ is chosen randomly with uniform probability among $|I_\eta(T)|$ values, the probability that this happens is smaller than $\frac{q''(\eta)(q''(\eta)-1)}{2|I_\eta(T)|}$ where $q''(\eta)$ is the number of possible values of $a_1, \dots, a_m$, which is a polynomial in η.

- Case (4): We first show that, if M characterizes a part of x with $S_{\mathrm{def}}, S_{\mathrm{dep}}$, then for all M_0 obtained from M by substituting variables of S_{def} with their definition, there exist a tuple of terms $\widetilde{M}$, a large type T, and uniform functions $f_1, \dots, f_k$ such that T is the type of the result of f_1 (or of x when $k = 0$) and for each a, E_0, and σ, there exists b such that for all E such that E equals E_0 on variables not in S_{dep}, if $E, \sigma, M_0 \Downarrow a$ then $E, \sigma, f_1(\dots f_k(x[\widetilde{M}])) \Downarrow b$. Indeed, $\mathcal{M}_0 = \{\alpha M_0 = M_0\}$ is rewritten into a set that contains $f_1(\dots f_k((\alpha x)[\widetilde{M'}])) = f_1(\dots f_k(x[\widetilde{M}]))$. Due to the form of rewrite rules, $(\alpha x)[\widetilde{M'}]$ is a subterm of αM_0 and $x[\widetilde{M}]$ is a subterm of M_0. Moreover, the variables in S_{dep} do not occur in $\widetilde{M}$ or $\widetilde{M'}$.
  - If a is such that there exists E' such that E' equals E_0 on variables not in S_{dep}, $E', \sigma, \alpha M_0 \Downarrow a$ and E' defines variables of αM_0, let b such that $E', \sigma, f_1(\dots f_k((\alpha y)[\widetilde{M'}])) \Downarrow b$. Then for all E such that E equals E_0 on variables not in S_{dep} and $E, \sigma, M_0 \Downarrow a$, we can define the E'' that maps variables of M_0 as E and variables of αM_0 as E'. Then $E'', \sigma, (\alpha M_0 = M_0) \Downarrow \mathrm{true}$, so by rewriting $E'', \sigma, f_1(\dots f_k((\alpha x)[\widetilde{M'}])) = f_1(\dots f_k(x[\widetilde{M}])) \Downarrow \mathrm{true}$, so $E, \sigma, f_1(\dots f_k(x[\widetilde{M}])) \Downarrow b$.
  - Otherwise, there is no E such that E equals E_0 on variables not in S_{dep} and $E, \sigma, M_0 \Downarrow a$, so the result holds trivially.

  So there exists a function f such that for each a, σ, E, if $E, \sigma, M_0 \Downarrow a$ then $E, \sigma, f_1(\dots f_k(x[\widetilde{M}])) \Downarrow f(a, \sigma,$

$E_{|\overline{S_{\rm dep}}})$. Since the variables in $S_{\rm dep}$ do not occur in $\widetilde{M}$, there exists a tuple of functions $\widetilde{f}$ such that $E,\sigma,\widetilde{M} \Downarrow \widetilde{f}(\sigma, E_{|\overline{S_{\rm dep}}})$. So $E,\sigma, f_1(\dots f_k(x[\widetilde{f}(\sigma, E_{|\overline{S_{\rm dep}}})])) \Downarrow f(a,\sigma,E_{|\overline{S_{\rm dep}}})$.

Let us now consider the three cases of Rule (4). In each case, we show that $p = \Pr[\exists E, \exists\sigma, \exists\mathcal{Q}, \exists\mathcal{C}, \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge E, \sigma \vdash M_1 = M_2] \leq q'(\eta) p_{\max}(\eta)$ for some polynomial q' and for M_1, M_2 that satisfy the hypothesis of Rule (4).

– First case: M_1' is obtained from M_1 by replacing all array indices that are not replication indices with fresh replication indices, x occurs in M_1', x is defined by restrictions $\mathsf{new}\ x : T'$, T' is a large type, M_1' characterizes a part of x, and M_2 is obtained by optionally applying function symbols to terms of the form $y[\widetilde{M'}]$ where y is defined by restrictions and $y \neq x$.

Let M_2' be obtained from M_2 by replacing all array indices that are not replication indices with fresh replication indices. Let $S_{\rm indep}$ be the set of variables defined only by restrictions, excluding x. Since M_1' characterizes a part of x, there exist a large type T, functions f and $\widetilde{f}$, and uniform functions $f_1, \dots, f_k$ such that T is the type of the result of f_1 (or of x when $k = 0$) and for each a, E, and σ, if $E, \sigma, M_1 \Downarrow a$ then $E,\sigma, f_1(\dots f_k(x[\widetilde{f}(\sigma, E_{|S_{\rm indep}})])) \Downarrow f(a, \sigma, E_{|S_{\rm indep}})$.

If $E, \sigma \vdash M_1 = M_2$ then we have $E, \sigma, M_1 \Downarrow a$ and $E, \sigma, M_2 \Downarrow a$ for some a. Then there exists an extension σ' of σ to the fresh replication indices of M_1' and M_2' such that $E, \sigma', M_1' \Downarrow a$ and $E, \sigma', M_2' \Downarrow a$. So $E, \sigma', f_1(\dots f_k(x[\widetilde{f}(\sigma', E_{|S_{\rm indep}})])) \Downarrow f(a, \sigma', E_{|S_{\rm indep}})$ and since only the variables of $S_{\rm indep}$ occur in M_2', there is a function f' such that $a = f'(\sigma', E_{|S_{\rm indep}})$. So

$$E(x[\widetilde{f}(\sigma', E_{|S_{\rm indep}})]) \in S_x(\sigma, E_{|S_{\rm indep}}) = (I_\eta(f_1) \circ \dots \circ I_\eta(f_k))^{-1}(f(f'(\sigma', E_{|S_{\rm indep}}), \sigma', E_{|S_{\rm indep}})).$$

Let $T_1, \dots, T_k$ be the types of the arguments of $f_1, \dots, f_k$ respectively; $T_0 = T$, $T_k = T'$. We have $|S_x(\sigma, E_{|S_{\rm indep}})| \leq \frac{|I_\eta(T_1)|}{|I_\eta(T_0)|} \times \dots \times \frac{|I_\eta(T_k)|}{|I_\eta(T_{k-1})|} = \frac{|I_\eta(T_k)|}{|I_\eta(T_0)|} = \frac{|I_\eta(T')|}{|I_\eta(T)|}$ since $f_1, \dots, f_k$ are uniform. Let $E_{\rm indep}$ be an environment giving values to variables of $S_{\rm indep}$. Let $\widetilde{i'} = \mathrm{Dom}(\sigma)$ be the current replication indices at P.

$$\begin{aligned}
p &\leq \sum_{\sigma'} \sum_{E_{\rm indep}} \Pr\begin{bmatrix}\exists(E, \mathcal{Q}, \mathcal{C}), \mathbb{C}_0 \to^* E, (\sigma'_{\widetilde{i'}}, P), \mathcal{Q}, \mathcal{C}\\ \wedge E_{|S_{\rm indep}} = E_{\rm indep} \wedge\\ E(x[\widetilde{f}(\sigma', E_{\rm indep})]) \in S_x(\sigma', E_{\rm indep})\end{bmatrix}\\
&\leq \sum_{\sigma'} \frac{1}{|I_\eta(T)|} \sum_{E_{\rm indep}} \Pr\begin{bmatrix}\exists(E, \mathcal{Q}, \mathcal{C}),\\ \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C}\\ \wedge E_{|S_{\rm indep}} = E_{\rm indep}\end{bmatrix}\\
&\leq \frac{q_1(\eta)}{|I_\eta(T)|}
\end{aligned}$$

where $q_1(\eta)$ is the number of possible σ', which is polynomial in η. So the result follows with $q'(\eta) = q_1(\eta)$.

– Second case: $\mathrm{simplifyTerm}(M_1 = M_2, P) = \mathrm{false}$. The result follows immediately from the correctness of the local dependency analysis, Property (S11).

The case in which global dependency analysis is triggered leads to a separate game transformation, which is proved correct below.

Let us now prove (S12) in the general case in which we can use any user-defined rewrite rule of Section C.1. The proof proceeds as in the previous case, except that we now have to consider cases (1) and (6) when the reduction uses a user-defined rewrite rule of the second kind $\mathsf{new}\ y_1 : T_1', \dots, \mathsf{new}\ y_l : T_l', \forall x_1 : T_1, \dots, \forall x_m : T_m, M_1 \to M_2$, with associated probability $p(\eta)$. Suppose that this user-defined claim is correct. We first prove that, if σ' is a substitution that maps x_j to any term of type T_j for all $j \leq m$ and y_j to terms to the form $z_j[\widetilde{M_j}]$ where x_j is defined only by restrictions $\mathsf{new}\ z_j : T_j'$ for all $j \leq l$, and $\mathcal{C}ond = \{\widetilde{M_j} = \widetilde{M_{j'}} \mid j \neq j' \wedge z_j = z_{j'}\} = \{cond_1, \dots, cond_k\}$, then

$$\begin{aligned}
&\Pr[\exists(E, \sigma, \mathcal{Q}, \mathcal{C}), \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge\\
&\qquad E, \sigma \vdash \sigma' M_1 \neq \sigma' M_2 \wedge \neg(cond_1 \vee \dots \vee cond_k)]\\
&\qquad\leq q(\eta) p_{\max}(\eta)
\end{aligned} \tag{8}$$

for some polynomial q.

By definition of the user-defined rewrite rule,

$$\begin{aligned}
&\Pr[E'(y_1) \xleftarrow{R} I_\eta(T_1'); \dots E'(y_l) \xleftarrow{R} I_\eta(T_l');\\
&\quad (E'(x_1), \dots, E'(x_m)) \leftarrow \mathcal{A}(E'(y_1), \dots, E'(y_l));\\
&\quad E', M_1 \Downarrow a; E', M_2 \Downarrow a' : a \neq a'] \leq p_{\max}(\eta)
\end{aligned}$$

where $\mathcal{A}$ is a probabilistic Turing machine running in time at most the maximum runtime of $C[Q_0]$.

Let us define $\mathcal{A}$ as follows. Let S denote the set of possible values $(\widetilde{a}_1, \dots, \widetilde{a}_l)$ for the indices of $z_1, \dots, z_l$, such that $(z_j, \widetilde{a}_j) \neq (z_{j'}, \widetilde{a}_{j'})$ when $j \neq j'$. $\mathcal{A}$ chooses indices $\widetilde{a}_1, \dots, \widetilde{a}_l$ at random uniformly in S. Next, it runs the game starting from $\mathbb{C}_0$ normally, except that it sets $z_j[\widetilde{a}_j]$ to $E'(y_j)$ for all $j \leq l$ instead of choosing $z_j[\widetilde{a}_j]$ randomly. If for all $j \leq l$, $E, \widetilde{M}_j \Downarrow \widetilde{a}_j$, then $\mathcal{A}$ returns $(b_1, \dots, b_m)$ such that for all $j \leq m$, $E, \sigma, \sigma' x_j \Downarrow b_j$. Otherwise, $\mathcal{A}$ fails. If $\mathcal{A}$ does not fail, then $E', M_1 \Downarrow a$ such that $E, \sigma, \sigma' M_1 \Downarrow a$ and $E', M_2 \Downarrow a'$ such that $E, \sigma, \sigma' M_2 \Downarrow a'$. Hence, if $E, \sigma \vdash \sigma' M_1 \neq \sigma' M_2$, then $a \neq a'$.

Let $q(\eta)$ be the cardinal of S, which is polynomial in η. Then

$$\begin{aligned}
&\Pr\begin{bmatrix}E'(y_1) \xleftarrow{R} I_\eta(T_1'); \dots E'(y_l) \xleftarrow{R} I_\eta(T_l');\\ (E'(x_1), \dots, E'(x_m)) \leftarrow \mathcal{A}(E'(y_1), \dots, E'(y_l));\\ E', M_1 \Downarrow a; E', M_2 \Downarrow a' : a \neq a'\end{bmatrix}\\
&= \Pr\begin{bmatrix}\exists(E, \sigma, \mathcal{Q}, \mathcal{C}), (\widetilde{a}_1, \dots, \widetilde{a}_l) \xleftarrow{R} S; \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C}\\ \wedge E, \sigma \vdash \sigma' M_1 \neq \sigma' M_2 \wedge \neg(cond_1 \vee \dots \vee cond_k)\\ \wedge \forall j \leq l,\ E, \widetilde{M}_j \Downarrow \widetilde{a}_j\end{bmatrix}\\
&= \frac{1}{q(\eta)} \Pr\begin{bmatrix}\exists(E, \sigma, \mathcal{Q}, \mathcal{C}), \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge\\ E, \sigma \vdash \sigma' M_1 \neq \sigma' M_2 \wedge \neg(cond_1 \vee \dots \vee cond_k)\end{bmatrix}
\end{aligned}$$

since, when the other conditions are satisfied, the last condition $\forall j \leq l$, $E, \widetilde{M}_j \Downarrow \widetilde{a}_j$ holds for exactly one element of S, and there are $q(\eta)$ elements in S. Since the initial probability is at most $p_{\max}(\eta)$, we obtain (8).

When $E, \sigma \vdash (\mathcal{F}, \mathcal{R})$ but $E, \sigma \not\vdash (\mathcal{F}', \mathcal{R}')$, we have that M reduces into M' by a user-defined rewrite rule knowing $\mathcal{F}, \mathcal{R}$, and $E, \sigma \vdash M \neq M'$.

- First case: in the definition of M reduces into M' by the user-defined rewrite rule, $\mathcal{C}ond = \emptyset$.

  We have $M = C[\sigma' M_1]$, $M' = C[\sigma' M_2]$, where C is a term context and σ' is a substitution that maps x_j to any term of type T_j for all $j \leq m$ and y_j to terms to the form $z_j[\widetilde{M_j}]$ where x_j is defined only by restrictions new $z_j : T'_j$ for all $j \leq l$. We have $\mathcal{C}ond = \{\widetilde{M_j} = \widetilde{M_{j'}} \mid j \neq j' \wedge z_j = z_{j'}\} = \emptyset$.

  Therefore,

$$\begin{aligned}
&\Pr\begin{bmatrix}\exists(E, \sigma, \mathcal{Q}, \mathcal{C}), \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge \\ E, \sigma \vdash (\mathcal{F}, \mathcal{R}) \wedge E, \sigma \not\vdash (\mathcal{F}', \mathcal{R}')\end{bmatrix} \\
&\leq \Pr\begin{bmatrix}\exists(E, \sigma, \mathcal{Q}, \mathcal{C}), \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge \\ E, \sigma \vdash (\mathcal{F}, \mathcal{R}) \wedge E, \sigma \vdash M \neq M'\end{bmatrix} \\
&\leq \Pr\begin{bmatrix}\exists(E, \sigma, \mathcal{Q}, \mathcal{C}), \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge \\ E, \sigma \vdash \sigma' M_1 \neq \sigma' M_2\end{bmatrix} \\
&\leq q(\eta) p_{\max}(\eta)
\end{aligned}$$

  by (8).

- Second case: in the definition of M reduces into M' by the user-defined rewrite rule, $\mathcal{C}ond \neq \emptyset$.

  We have $M = C[\sigma' M_1]$, $C[\sigma' M_2] = \text{false}$, where C is a term context and σ' is a substitution that maps x_j to any term of type T_j for all $j \leq m$ and y_j to terms to the form $z_j[\widetilde{M_j}]$ where x_j is defined only by restrictions new $z_j : T'_j$ for all $j \leq l$. We have $\mathcal{C}ond = \{\widetilde{M_j} = \widetilde{M_{j'}} \mid j \neq j' \wedge z_j = z_{j'}\} = \{cond_1, \dots, cond_k\}$. M is transformed into $M'_{k'}$ by rewrite rules generated by our equational prover from $\mathcal{F} \cup \{cond_{k'}\}, \mathcal{R}$ and user-defined rewrite rules, using only the first kind of user-defined rewrite rules of Section C.1, and $M' = (cond_1 \wedge M'_1) \vee \dots \vee (cond_k \wedge M'_k)$.

  Therefore,

$$\begin{aligned}
&\Pr\begin{bmatrix}\exists(E, \sigma, \mathcal{Q}, \mathcal{C}), \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge \\ E, \sigma \vdash (\mathcal{F}, \mathcal{R}) \wedge E, \sigma \not\vdash (\mathcal{F}', \mathcal{R}')\end{bmatrix} \\
&\leq \Pr\begin{bmatrix}\exists(E, \sigma, \mathcal{Q}, \mathcal{C}), \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge \\ E, \sigma \vdash (\mathcal{F}, \mathcal{R}) \wedge E, \sigma \vdash M \neq M'\end{bmatrix} \\
&\leq \Pr\begin{bmatrix}\exists(E, \sigma, \mathcal{Q}, \mathcal{C}), \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge \\ E, \sigma \vdash M \neq (cond_1 \vee \dots \vee cond_k) \wedge M\end{bmatrix} \\
&+ \Pr\begin{bmatrix}\exists(E, \sigma, \mathcal{Q}, \mathcal{C}), \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge \\ E, \sigma \vdash (\mathcal{F}, \mathcal{R}) \wedge E, \sigma \vdash ((cond_1 \vee \dots \vee cond_k) \\ \wedge M \neq (cond_1 \wedge M'_1) \vee \dots \vee (cond_k \wedge M'_k))\end{bmatrix} \\
&\leq \Pr\begin{bmatrix}\exists(E, \sigma, \mathcal{Q}, \mathcal{C}), \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge \\ E, \sigma \vdash M \neq \text{false} \wedge \neg(cond_1 \vee \dots \vee cond_k)\end{bmatrix} \\
&+ \sum_{k'=1}^{k} \Pr\begin{bmatrix}\exists(E, \sigma, \mathcal{Q}, \mathcal{C}), \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge \\ E, \sigma \vdash (\mathcal{F}, \mathcal{R}) \wedge \\ E, \sigma \vdash cond_{k'} \wedge M \neq cond_{k'} \wedge M'_{k'}\end{bmatrix} \\
&\leq \Pr\begin{bmatrix}\exists(E, \sigma, \mathcal{Q}, \mathcal{C}), \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge \\ E, \sigma \vdash \sigma' M_1 \neq \sigma' M_2 \wedge \neg(cond_1 \vee \dots \vee cond_k)\end{bmatrix} \\
&+ \sum_{k'=1}^{k} \Pr\begin{bmatrix}\exists(E, \sigma, \mathcal{Q}, \mathcal{C}), \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge \\ E, \sigma \vdash (\mathcal{F} \cup \{cond_{k'}\}, \mathcal{R}) \wedge \\ E, \sigma \vdash M \neq M'_{k'}\end{bmatrix} \\
&\leq q(\eta) p_{\max}(\eta) \\
&+ \sum_{k'=1}^{k} \Pr\begin{bmatrix}\exists(E, \sigma, \mathcal{Q}, \mathcal{C}), \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge \\ E, \sigma \vdash (\mathcal{F} \cup \{cond_{k'}\}, \mathcal{R}) \wedge \\ E, \sigma \not\vdash (\mathcal{F}'_{k'}, \mathcal{R}'_{k'})\end{bmatrix}
\end{aligned}$$

  where the first term has been bounded by (8) and in the second term, the equational prover applied to $\mathcal{F} \cup \{cond_{k'}\}, \mathcal{R}$ yields $\mathcal{F}'_{k'}, \mathcal{R}'_{k'}$ using only the first kind of user-defined rewrite rules. (When $E, \sigma \vdash (\mathcal{F}'_{k'}, \mathcal{R}'_{k'})$, all rewrite rules of $\mathcal{R}'_{k'}$ hold, so $E, \sigma \vdash M = M'_{k'}$.) By the correctness of the equational prover using only the first kind of user-defined rewrite rules (shown above), the probability is then bounded by $q(\eta) p_{\max}(\eta) + \sum_{k'=1}^{k} q_{k'}(\eta) p_{\max}(\eta)$ for some polynomials $q_{k'}$.

This concludes the proof of (S12).

Similarly, we also have **S12'**: For each Q', $\Pr[\exists(E, \sigma, P, \mathcal{Q}, \mathcal{C}, c, M_1, \dots, M_l, \underline{N_1, \dots, N_k}, Q'', \sigma', \mathcal{Q}', \mathcal{C}'), \mathbb{C}_0 \to^* E, (\sigma, P), \mathcal{Q}, \mathcal{C} \wedge P = c[M_1, \dots, M_l]\langle N_1, \dots, N_k\rangle.Q'' \wedge E, \{(\sigma, Q'')\}, \mathcal{C} \rightsquigarrow^* E, \mathcal{Q}', \mathcal{C}' \wedge (\sigma', Q') \in \mathcal{Q}' \wedge E, \sigma' \not\vdash (\mathcal{F}_{m'}, \mathcal{R}_{m'})(Q')] \leq q'(\eta) p_{\max}(\eta)$ for some polynomial q'.

We have $\mathcal{F}_{Q'} = \mathcal{F}_P$, hence $(\mathcal{F}_{m'}, \mathcal{R}_{m'})(Q') = (\mathcal{F}_{m'}, \mathcal{R}_{m'})(P)$, and σ' is an extension of σ, so $E, \sigma \vdash (\mathcal{F}_{m'}, \mathcal{R}_{m'})(P)$ implies $E, \sigma' \vdash (\mathcal{F}_{m'}, \mathcal{R}_{m'})(Q')$. So the result follows from (S12).

Correctness of game simplification. For simplicity, we consider one transformation at a time, and use transitivity of $\approx^V$ to conclude when several transformations are applied. For each trace $\text{initConfig}(C[Q_0]) \to^* E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m$, except in cases of negligible probability, we show that there exists a corresponding trace $\text{initConfig}(C[Q'_0]) \to^* E'_{m'}, P'_{m'}, \mathcal{Q}'_{m'}, \mathcal{C}'_{m'}$ with $E'_{m'} = E_m$, $P'_{m'}$ is obtained from P_m by the same transformation as Q'_0 from Q_0, $\mathcal{Q}'_{m'}$ is obtained from $\mathcal{Q}_m$ by the same transformation as Q'_0 from Q_0, $\mathcal{C}'_{m'} = \mathcal{C}_m$, with the same probability. The proof proceeds by induction on m.

For the case $m = 0$, the only simplification that can be applied to input processes is the simplification of terms in input channels. Moreover, if Q' is the transformed process, $\mathcal{F}_{Q'} = \emptyset$ since $\mathcal{F}_{C[Q_0]} = \emptyset$ and Q' is obtained from $C[Q_0]$ by $\rightsquigarrow$, which reduces only input processes. So $(\mathcal{F}_0, \mathcal{R}_0)(Q') = (\emptyset, \emptyset)$. No rule of the equational prover applies on $(\emptyset, \emptyset)$, so $(\mathcal{F}_{m'}, \mathcal{R}_{m'})(Q') = (\emptyset, \emptyset)$, hence no rewrite rule of $\mathcal{R}_{m'}$ can be applied. So one can only simplify terms in the input channel of Q' by a user-defined rewrite rule. The proof then proceeds exactly as in Case 1 below.

For the inductive step, we reason by cases on the last reduction step of the trace of $C[Q_0]$. We consider only the cases in which the transition may be altered by the game simplification.

- Case 1: M reduces into M' by a user-defined rewrite rule, and we replace M with M' in the smallest (input or output) process $P_M = C_M[M]$ that contains M. If $E, \sigma, M \Downarrow a$ then $E, \sigma, M' \Downarrow a'$ (since the variable accesses in M' are included in those of M and M and M' are well-typed). If the user-defined rewrite rule is of the first kind, we always have $a = a'$. If the user-defined rewrite rule is of the second kind, we show as in the proof of (S12), that the probability that $a \neq a'$ (that is, $E, \sigma \vdash M \neq M'$) is negligible, so this situation can be excluded. Otherwise, $a = a'$, and $C_M[M']$ reduces in the same way as $P_M = C_M[M]$.

- Case 2: M reduces into M' by a rule of $\mathcal{R}$, and we replace M with M' in the smallest process $P_M = C_M[M]$ that contains M, where $\mathcal{R}$ is the set of rewrite rules obtained by the equational prover from $\mathcal{F}_{P_M}$. We first assume that P_M is an output process. We exclude traces such that $E, \sigma \not\vdash (\mathcal{F}_{m'}, \mathcal{R}_{m'})(P_M)$. (They have negligible probability by (S12).) In the remaining traces, for all $(M_1 \to M_2) \in \mathcal{R} = \mathcal{R}_{m'}$, $E, \sigma \vdash M_1 = M_2$, so $E, \sigma \vdash M = M'$. So $E, \sigma, M \Downarrow a$ if and only if $E, \sigma, M' \Downarrow a$, and $C_M[M']$ reduces in the same way as $P_M = C_M[M]$. When we reduce a term in the channel of an input, we have a similar proof with an input process $Q_M = C_M[M]$ instead of P_M and using (S12') instead of (S12).

- Case 3: $P = \mathsf{find}\ (\bigoplus_{j=1}^m u_{j1}[\widetilde{i}] \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}\ \mathsf{suchthat}\ \mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)$ $\mathsf{else}\ P'$, $\mathcal{F}_{P_j}$ yields a contradiction, and we remove the j-th branch of the find. We exclude traces in which $E, \sigma, (\mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j) \Downarrow \mathrm{true}$. Let S we the set defined in the reduction rule (Find1). We have $|S| \leq \sum_{j=1}^m \prod_{l=1}^{m_j} n_{jl} = q(\eta)$ for some polynomial q, and $\mathrm{among}(S) = \frac{2^{k+f(\eta)} \operatorname{div} |S|}{2^{k+f(\eta)}}$ where k is the smallest integer such that $2^k \geq |S|$, so $\mathrm{among}(S) \geq \frac{2^{f(\eta)}}{2^{k+f(\eta)}} \geq \frac{1}{2^k} \geq \frac{1}{2|S|} \geq \frac{1}{2q(\eta)}$. By (Find1), P reduces into P_j with probability at least $\mathrm{among}(S)$, so at least $\frac{1}{2q(\eta)}$, when $E, \sigma, (\mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j) \Downarrow \mathrm{true}$. Therefore,
$$\begin{aligned}&\Pr\begin{bmatrix}\exists(E,\sigma,\mathcal{Q},\mathcal{C}), \mathbb{C}_0 \to^* E,(\sigma,P),\mathcal{Q},\mathcal{C}\\ \wedge E,\sigma,(\mathsf{defined}(M_{j1},\dots,M_{jl_j}) \wedge M_j) \Downarrow \mathrm{true}\end{bmatrix}\\ &\leq 2q(\eta)\Pr\left[\exists(E,\sigma,\mathcal{Q},\mathcal{C}), \mathbb{C}_0 \to^* E,(\sigma,P_j),\mathcal{Q},\mathcal{C}\right]\\ &\leq 2q(\eta)\Pr\begin{bmatrix}\exists(E,\sigma,\mathcal{Q},\mathcal{C}), \mathbb{C}_0 \to^* E,(\sigma,P_j),\mathcal{Q},\mathcal{C}\\ \wedge E,\sigma \not\vdash (\mathcal{F}_{m'},\mathcal{R}_{m'})(P_j)\end{bmatrix}\end{aligned}$$
since $E, \sigma \not\vdash (\mathcal{F}_{m'}, \mathcal{R}_{m'})(P_j)$ is always true since $\mathcal{F}_{P_j}$ yields a contradiction. So the excluded traces have negligible probability by (S12). In the remaining traces, $E, \sigma, (\mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j) \Downarrow \mathrm{false}$, so the set S never contains $(j, \widetilde{v})$ for any $\widetilde{v}$, hence by (Find1) or (Find2), the process takes the same branch of the find with the same probability, whether or not the j-th branch is present.

- Case 4: $P_0 = \mathsf{find}\ (\bigoplus_{j=1}^m u_{j1}[\widetilde{i}] \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] \leq n_{jm_j}\ \mathsf{suchthat}\ \mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P'$, there exists a term M such that $\mathsf{defined}(M) \in \mathcal{F}_{P_j}$, $x[N_1, \dots, N_l]$ is a subterm of M, $x \neq u_{jk}$ for all $k \leq m_j$, and none of the following conditions holds: a) P_0 is under a definition of x in Q_0; b) Q_0 contains $Q_1 \mid Q_2$ such that a definition of x occurs in Q_1 and P_0 is under Q_2 or a definition of x occurs in Q_2 and P_0 is under Q_1; c) Q_0 contains $lp + 1$ replications above a process Q that contains a definition of x and P_0, where lp is the length of the longest common prefix between $N_1, \dots, N_l$ and the current replication indices at the definitions of x. The j-th branch of the find is removed.

We show that $x[N_1, \dots, N_l]$ cannot be defined at P_0 as follows. We say that the formula $\phi(E, (\sigma, P), \mathcal{Q}, \mathcal{C})$ is true when one of the following condition holds:

A. $x[a_1, \dots, a_m] \in \mathrm{Dom}(E)$, $(\sigma'', P'') \in \mathcal{Q} \uplus \{(\sigma, P)\}$, P_0 is under P'', and $\sigma'' i''_k = a_k$ for all $k \leq \min(lp, |\mathrm{Dom}(\sigma'')|)$, where i''_k is the k-th replication index at P'';

B. $\{(\sigma', P'), (\sigma'', P'')\} \subseteq \mathcal{Q} \uplus \{(\sigma, P)\}$ (multiset inclusion), P' contains a definition of x, P_0 is under P'', $\sigma' i'_k = \sigma'' i''_k$ for all $k \leq \min(lp, |\mathrm{Dom}(\sigma')|, |\mathrm{Dom}(\sigma'')|)$ where i'_k is the k-th replication index at P' and i''_k is the k-th replication index at P'';

C. $(\sigma', P') \in \mathcal{Q} \uplus \{(\sigma, P)\}$ where

   C.a. P_0 is under a definition of x in P';

   C.b. or P' contains $Q_1 \mid Q_2$ such that a definition of x occurs in Q_1 and P_0 is under Q_2 or a definition of x occurs in Q_2 and P_0 is under Q_1;

   C.c. or P' contains $lp + 1 - |\mathrm{Dom}(\sigma')|$ replications above a process Q that contains a definition of x and P_0.

If the j-th branch of the find is taken in configuration $E, (\sigma, P_0), \mathcal{Q}, \mathcal{C}$, this configuration reduces into $E', (\sigma, P_j), \mathcal{Q}, \mathcal{C}$ by executing the find. By correctness of the collection of true facts, $E', \sigma \vdash \mathcal{F}_{P_j}$, so $E', \sigma \vdash \mathsf{defined}(M)$, so $E', \sigma \vdash \mathsf{defined}(x[N_1, \dots, N_l])$, that is, by definition of lp, $E', \sigma \vdash \mathsf{defined}(x[i_1, \dots, i_{lp}, N_{lp+1}, \dots, N_l])$ where i_k is the k-th replication index at P_0. Hence, $x[a_1, \dots, a_m] \in \mathrm{Dom}(E')$ where $\sigma i_k = a_k$ for all $k \leq lp$. Since $E' = E[u_{j1} \mapsto \dots, \dots, u_{jm_j} \mapsto \dots]$ and $x \neq u_{jk}$ for all $k \leq m_j$, $x[a_1, \dots, a_m] \in \mathrm{Dom}(E)$. Therefore, $\phi(E, (\sigma, P_0), \mathcal{Q}, \mathcal{C})$ holds (Case A with $(\sigma'', P'') = (\sigma, P_0)$).

Next, we show that if a configuration in the trace satisfies ϕ, then the previous configuration also satisfies ϕ.

More precisely, we first show that if $\phi(E, (\sigma, P), \mathcal{Q}'' \uplus \mathcal{Q}', \mathcal{C}')$ and $E, \mathcal{Q}, \mathcal{C} \rightsquigarrow E, \mathcal{Q}', \mathcal{C}'$, then $\phi(E, (\sigma, P), \mathcal{Q}'' \uplus \mathcal{Q}, \mathcal{C})$. The proof is by cases on the reduction rule of $\rightsquigarrow$. Case (Nil) is obvious. For rule (Par), if we are in case B and both processes P' and P'' are generated by (Par), then before applying (Par), we are in case C.b. In all other cases, we remain in the same case of the definition of ϕ before applying (Par). For rule (Repl), if we are in case B and both processes P' and P'' are generated by (Repl), then before applying (Repl), we are in case C.c. In all other cases, we remain in the same case before applying (Repl). For rules (NewChannel) and (Input), we remain in the same case.

Therefore, if $\phi(E, (\sigma, P), \mathcal{Q}'' \uplus \mathcal{Q}', \mathcal{C}')$ and $E, \mathcal{Q}', \mathcal{C}' = \mathrm{reduce}(E, \mathcal{Q}, \mathcal{C})$, then $\phi(E, (\sigma, P), \mathcal{Q}'' \uplus \mathcal{Q}, \mathcal{C})$.

We also show that, if $\phi(E', (\sigma', P'), \mathcal{Q}', \mathcal{C}')$ and $E, (\sigma, P), \mathcal{Q}, \mathcal{C} \xrightarrow{p}_t E', (\sigma', P'), \mathcal{Q}', \mathcal{C}'$, then $\phi(E,$

$(\sigma, P), \mathcal{Q}, \mathcal{C})$. The proof is by cases on the reduction rule of $\xrightarrow{p}_t$. For rule (Find2), we remain in the same case of the definition of ϕ. For rules (New), (Let), (Find1), if we are in case A after applying the reduction and the reduction defines $x[a_1, \dots, a_m]$, then we are in case C.a before the reduction if (σ'', P'') is (σ, P) and in case B otherwise. Otherwise, we remain in the same case. For rule (Output), $E, (\sigma, \overline{c[\widetilde{M}]\langle N_1, \dots, N_k\rangle.Q''}),$ $\{(\sigma', c[\widetilde{a}](x_1[\widetilde{a'}] : T_1, \dots, x_k[\widetilde{a'}] : T_k).P)\} \uplus \mathcal{Q}, \mathcal{C}$ is transformed into $E', (\sigma', P), \mathcal{Q} \uplus \{(\sigma, Q'')\}, \mathcal{C}$, where $E' = E[x_1[\widetilde{a'}] \mapsto \dots, \dots, x_k[\widetilde{a'}] \mapsto \dots]$, then we reduce $E', \{(\sigma, Q'')\}, \mathcal{C}$ by the function reduce. By the property shown for reduce, we have $\phi(E', (\sigma', P), \mathcal{Q} \uplus \{(\sigma, Q'')\}, \mathcal{C})$. If we are in case A and the input defines $x[a_1, \dots, a_m]$, then before (Output), we are in case C.a if (σ'', P'') is (σ, P) and in case B otherwise. Otherwise, we remain in the same case.

Next, we show that if the j-th branch of the find is taken by (Find1) when evaluating P_0, then the last configuration of the trace satisfies ϕ. In this case, $x[a_1, \dots a_m] \in \mathrm{Dom}(E)$ in a configuration $E, (\sigma, P_0), \mathcal{Q}, \mathcal{C}$ such that $\sigma i_k = a_k$ for all $k \le lp$, where i_k is the k-th replication index at P_0. So $\phi(E, (\sigma, P_0), \mathcal{Q}, \mathcal{C})$ (case A).

Therefore, by the previous proof, ϕ holds for the initial configuration, so we have $\phi(\emptyset, (\emptyset, \overline{start}\langle\rangle), \{(\emptyset, C[Q_0])\}, \emptyset)$. Case A cannot happen because E is empty; case B cannot happen because $\overline{start}\langle\rangle$ contains neither P_0 nor a definition of x and (σ', P') and (σ'', P'') cannot be the same process $(\emptyset, C[Q_0])$. So we are in case C with $P' = C[Q_0]$ and $\sigma' = \emptyset$. Since C contains neither P_0 nor a definition of x, we obtain that one of the conditions a), b), c) holds, which contradicts the hypothesis. So the j-th branch of the find cannot be taken, and can be removed.

- Case 5: $P_0 = \mathsf{find}\ (\bigoplus_{j=1}^m u_{j1}[\widetilde{i}] \le n_{j1}, \dots, u_{jm_j}[\widetilde{i}] \le n_{jm_j}\ \mathsf{suchthat}\ \mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P'$, there exist terms M, M' such that $\mathsf{defined}(M) \in \mathcal{F}_{P_j}$, $x[N_1, \dots, N_l]$ is a subterm of M, $\mathsf{defined}(M') \in \mathcal{F}_{P_j}$, $x'[N'_1, \dots, N'_{l'}]$ is a subterm of M', $N_k = N'_k$ for all $k \le \min(l, l')$, $x \neq x'$, and x and x' are incompatible. The j-th branch of the find is removed.

  If the j-th branch of the find is taken, yielding configuration $E, (\sigma, P_j), \mathcal{Q}, \mathcal{C}$, by correctness of the collection of true facts, we have $E, \sigma \vdash \mathsf{defined}(x[N_1, \dots, N_l])$ and $E, \sigma \vdash \mathsf{defined}(x'[N'_1, \dots, N'_{l'}])$. Suppose that $E, \sigma, N_k \Downarrow a_k$ and $E, \sigma, N'_k \Downarrow a'_k$. Then $x[a_1, \dots, a_l] \in \mathrm{Dom}(E)$, $x'[a'_1, \dots, a'_{l'}] \in \mathrm{Dom}(E)$, and $a_k = a'_k$ for all $k \le \min(l, l')$.

  Suppose for instance that $x'[a'_1, \dots, a'_{l'}]$ is defined after $x[a_1, \dots, a_l]$. (The other case is symmetric.) At the definition of $x'[a'_1, \dots, a'_{l'}]$, we are in a configuration $E', (\sigma', P'), \mathcal{Q}', \mathcal{C}'$ with $x[a_1, \dots, a_l] \in \mathrm{Dom}(E')$, $\sigma' i_k = a'_k$ for all $k \le l'$ where i_k is the k-th replication index at P'. Therefore, $\phi(E', (\sigma', P'), \mathcal{Q}', \mathcal{C}')$ holds, where ϕ has been defined in Case 4 above: we are in case A, with $(\sigma'', P'') = (\sigma', P')$, $lp = l$, $P_0 = P'$ is the process that defines x'. By the proof of Case 4, ϕ holds for the initial configuration $(\emptyset, (\emptyset, \overline{start}\langle\rangle), \{(\emptyset, C[Q_0])\}, \emptyset)$, and in this configuration, cases A and B are impossible, so we are in case C, so

  - Case C.a: P_0 is under a definition of x in $C[Q_0]$, therefore x' is defined under a definition of x.
  - or Case C.b: $C[Q_0]$ contains $Q_1 \mid Q_2$ such that a definition of x occurs in Q_1 and P_0 is under Q_2 or a definition of x occurs in Q_2 and P_0 is under Q_1, therefore x is defined in Q_1 and x' is defined in Q_2 or symmetrically. (Moreover, the process $Q_1 \mid Q_2$ is inside Q_0, because no definition of x or x' occurs in C.)
  - or Case C.c: $C[Q_0]$ contains $lp{+}1 = l{+}1$ replications above a process Q that contains a definition of x and P_0. This case is impossible because the definition of x is under l replications.

  Therefore, in all cases, x and x' are compatible. Contradiction. So the j-th branch of the find cannot be taken, and can be removed.

- The other cases can be handled in a way similar to cases 1–3.

We also show the converse property: for each trace of $C[Q'_0]$, except in cases of negligible probability, there exists a corresponding trace of $C[Q_0]$ with the same probability. Moreover, for all channels c and bitstrings a, $E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m$ executes $\overline{c}\langle a\rangle$ immediately if and only if $E'_{m'}, P'_{m'}, \mathcal{Q}'_{m'}, \mathcal{C}'_{m'}$ executes $\overline{c}\langle a\rangle$ immediately, so $\Pr[C[Q_0] \rightsquigarrow_\eta \overline{c}\langle a\rangle] = \Pr[C[Q'_0] \rightsquigarrow_\eta \overline{c}\langle a\rangle]$, which yields the desired equivalence.

Correctness of global dependency analysis Suppose that global dependency analysis succeeds in transforming the game, so that $\mathrm{only\_dep}(x) = S$. We obtain a game Q'_0 from Q_0 by replacing tests $M_1 = M_2$ with false and $M_1 \neq M_2$ with true in Q_0, where M_1 characterizes a part of x with $S \setminus \{x\}, S$, and no variable in S occurs in M_2.

We consider traces of $C[Q'_0]$ that differ by the choices of values of x. Since $\mathrm{only\_dep}(x) = S$, these traces differ only by the values of variables in S.

If $C[Q_0]$ behaves differently from $C[Q'_0]$, then there is a test $M_1 = M_2$ or $M_1 \neq M_2$ in Q_0, such that $E, \sigma, (M_1 = M_2) \Downarrow \mathrm{true}$, M_1 characterizes a part of x with $S \setminus \{x\}, S$, and no variable in S occurs in M_2.

In the considered traces of $C[Q'_0]$, the value of M_2 is the same a, which is therefore a function of σ and $E_{|\overline{S}}$, so $a = f'(\sigma, E_{|\overline{S}})$. Since $E, \sigma, (M_1 = M_2) \Downarrow \mathrm{true}$, we have $E, \sigma, M_1 \Downarrow a$. Then there is some M_0 obtained from M_1 by substituting variables in $S \setminus \{x\}$ with their definition such that $E, \sigma, M_0 \Downarrow a$. (We choose the definition of these variables used to set them in environment E.) The number of choices of M_0 is independent of η: it can be bounded knowing the number of different definitions of variables in S and the number of occurrences of these variables in the terms M_1.

Due to the properties of "characterize", there exist a large type T, functions f and $\widetilde{f}$, and uniform functions $f_1, \dots, f_k$ such that T is the type of the result of f_1 (or of x when $k = 0$) and for each

a, σ, E, if $E, \sigma, M_0 \Downarrow a$ then $E, \sigma, f_1(\dots f_k(x[\widetilde{f}(\sigma, E_{|\overline{S}})])) \Downarrow f(a, \sigma, E_{|\overline{S}})$. So $E(x[\widetilde{f}(\sigma, E_{|\overline{S}})]) \in S_x(\sigma, E_{|\overline{S}}) = (I_\eta(f_1) \circ \dots \circ I_\eta(f_k))^{-1}(f(f'(\sigma, E_{|\overline{S}}), \sigma, E_{|\overline{S}})$. Let $T_1, \dots, T_k$ be the types of the arguments of $f_1, \dots, f_k$ respectively; $T_0 = T$, $T_k = T'$. We have $|S_x(\sigma, E_{|\overline{S}})| \leq \frac{|I_\eta(T_1)|}{|I_\eta(T_0)|} \times \dots \times \frac{|I_\eta(T_k)|}{|I_\eta(T_{k-1})|} = \frac{|I_\eta(T_k)|}{|I_\eta(T_0)|} = \frac{|I_\eta(T')|}{|I_\eta(T)|}$ since $f_1, \dots, f_k$ are uniform.

The probability that $E, \sigma, (M_1 = M_2) \Downarrow \text{true}$ is at most the sum for all choices of M_0 of the probability that $E(x[\widetilde{f}(\sigma, E_{|\overline{S}})]) \in S_x(\sigma, E_{|\overline{S}})$, so it is at most $\sum_{M_0} \frac{1}{|I_\eta(T)|}$. (Note that T may depend on the choice of M_0.) Therefore, the probability that $C[Q_0]$ behaves differently from $C[Q_0']$ is at most $\sum_{M_1, M_2} \sum_{M_0} \frac{q_1(\eta)}{|I_\eta(T)|}$ where the number of possible σ, that is, the number of executions of the test $M_1 = M_2$ or $M_1 \neq M_2$ is at most $q_1(\eta)$, polynomial in η. This probability is negligible, so $Q_0 \approx^V Q_0'$.

We leave the proof of the additional transformations **Move(all)**, **RemoveAssign(useless)**, and **SArename(auto)** to the reader. The proof technique is similar to that for **SArename**(x). □

E.2 Proving the Last Hypothesis of Proposition 5

In this section, we show how to prove the last hypothesis of Proposition 5. We use the notations of Proposition 5 and of the proof of **Simplify** in the previous section.

For each definition P of x in Q, we define $\mathrm{defRestr}_P(x[\widetilde{i}])$ as follows:

$$\mathrm{defRestr}_P(x[\widetilde{i}]) = \begin{cases} x[\widetilde{i}] & \text{if } P = \mathsf{new}\ x[\widetilde{i'}] : T; P' \\ z[M_1, \dots, M_l]\{\widetilde{i}/\widetilde{i'}\} & \\ \quad \text{if } P = \mathsf{let}\ x[\widetilde{i'}] : T = z[M_1, \dots, M_l]\ \mathsf{in}\ P' & \end{cases}$$

Let $\mathcal{F}_P[\widetilde{i}]$ denote the facts that hold at P with current replication indices renamed to $\widetilde{i}$, that is, $\mathcal{F}_P[\widetilde{i}] = \mathcal{F}_P\{\widetilde{i}/\widetilde{i'}\}$ where the replication indices at P are $\widetilde{i'}$.

For each pair of definitions of x, P, P', we check that, if $\mathrm{defRestr}_P(x[\widetilde{i}]) = z[M_1, \dots, M_l]$ and $\mathrm{defRestr}_{P'}(x[\widetilde{i'}]) = z[M_1', \dots, M_l']$, then $\mathcal{F}_P[\widetilde{i}] \cup \mathcal{F}_{P'}[\widetilde{i'}] \cup \{\widetilde{i} \neq \widetilde{i'}, M_1 = M_1', \dots, M_l = M_l'\}$ yields a contradiction. That is, $\widetilde{i} \neq \widetilde{i'} \wedge M_1 = M_1' \wedge \dots \wedge M_l = M_l'$ is false except in cases of negligible probability, taking into account the facts that are known to hold at P and P'. When this check succeeds, the last hypothesis of Proposition 5 holds, as shown by the next proposition.

Proposition 7 *Assume that, for all pairs P, P' of definitions of x in Q, if $\mathrm{defRestr}_P(x[\widetilde{i}]) = z[M_1, \dots, M_l]$ and $\mathrm{defRestr}_{P'}(x[\widetilde{i'}]) = z[M_1', \dots, M_l']$, then $\mathcal{F}_P[\widetilde{i}] \cup \mathcal{F}_{P'}[\widetilde{i'}] \cup \{\widetilde{i} \neq \widetilde{i'}, M_1 = M_1', \dots, M_l = M_l'\}$ yields a contradiction (with local dependency analysis disabled).*

Then $\Pr[\exists(\mathcal{T}, \widetilde{a}, \widetilde{a'}), C[Q]$ reduces according to $\mathcal{T} \wedge \widetilde{a} \neq \widetilde{a'} \wedge \mathrm{defRestr}_\mathcal{T}(x[\widetilde{a}]) = \mathrm{defRestr}_\mathcal{T}(x[\widetilde{a'}])]$ is negligible.

The local dependency analysis is disabled because it gives information valid only at a certain process occurrence, and here we combine facts obtained at two occurrences P and P'.

Proof Consider a trace $\mathcal{T}$ of $C[Q]$ and $\widetilde{a} \neq \widetilde{a'}$ such that $\mathrm{defRestr}_\mathcal{T}(x[\widetilde{a}]) = \mathrm{defRestr}_\mathcal{T}(x[\widetilde{a'}])]$. Let P and P' be the processes that define $x[\widetilde{a}]$ and $x[\widetilde{a'}]$, respectively, in this trace. Let σ be mapping the replication indices at P to $\widetilde{a}$, σ' be mapping the replication indices at P' to $\widetilde{a'}$, and σ'' be mapping $\widetilde{i}$ to $\widetilde{a}$ and $\widetilde{i'}$ to $\widetilde{a'}$. Let E'' be the environment at the end of $\mathcal{T}$.

Just before the definition of $x[\widetilde{a}]$ is executed, the configuration of $\mathcal{T}$ is of the form $E, (\sigma, P), \dots$, so, since $\mathcal{F}_P$ is correct for all P, $E, \sigma \vdash \mathcal{F}_P$, so $E'', \sigma'' \vdash \mathcal{F}_P[\widetilde{i}]$. Similarly, $E'', \sigma'' \vdash \mathcal{F}_{P'}[\widetilde{i'}]$. Since $\widetilde{a} \neq \widetilde{a'}$, $E'', \sigma'' \vdash \widetilde{i} \neq \widetilde{i'}$. Since $\mathrm{defRestr}_\mathcal{T}(x[\widetilde{a}]) = \mathrm{defRestr}_\mathcal{T}(x[\widetilde{a'}])]$, $\mathrm{defRestr}_P(x[\widetilde{i}]) = z[M_1, \dots, M_l]$, $\mathrm{defRestr}_{P'}(x[\widetilde{i'}]) = z[M_1', \dots, M_l']$, for some $z, M_1, \dots, M_l, M_1', \dots, M_l'$, and $E'', \sigma'' \vdash M_1 = M_1'$, …, $E'', \sigma'' \vdash M_l = M_l'$. So $E'', \sigma'' \vdash \mathcal{F}_{P,P'}$, where $\mathcal{F}_{P,P'} = \mathcal{F}_P[\widetilde{i}] \cup \mathcal{F}_{P'}[\widetilde{i'}] \cup \{\widetilde{i} \neq \widetilde{i'}, M_1 = M_1', \dots, M_l = M_l'\}$.

Hence $\Pr[\exists(\mathcal{T}, \widetilde{a}, \widetilde{a'}), C[Q]$ reduces according to $\mathcal{T} \wedge \widetilde{a} \neq \widetilde{a'} \wedge \mathrm{defRestr}_\mathcal{T}(x[\widetilde{a}]) = \mathrm{defRestr}_\mathcal{T}(x[\widetilde{a'}])] \leq \sum_{P,P'} \Pr[\exists(E'', \sigma''), \mathbb{C}_0 \to^* E'', \dots \wedge E'', \sigma'' \vdash \mathcal{F}_{P,P'}]$.

When the local dependency analysis is disabled, the proof of correctness of the equational prover (S12) shown in the previous section also shows that, if $\frac{\mathcal{F}, \mathcal{R}}{\mathcal{F}', \mathcal{R}'}$, then

$$\Pr\begin{bmatrix} \exists(E'', \sigma''), \mathbb{C}_0 \to^* E'', \dots \\ \wedge E'', \sigma'' \vdash \mathcal{F}, \mathcal{R} \wedge E'', \sigma'' \not\vdash \mathcal{F}', \mathcal{R}' \end{bmatrix}$$

is negligible. Moreover, for all P and P' definitions of x in Q, since $\mathcal{F}_{P,P'}$ yields a contradiction, $\mathcal{F}_{P,P'}, \emptyset$ is transformed into $\mathrm{false}, \mathcal{R}'$ by the equational prover, so $\Pr[\exists(E'', \sigma''), \mathbb{C}_0 \to^* E'', \dots \wedge E'', \sigma'' \vdash \mathcal{F}_{P,P'}]$ is negligible, which shows the desired result. □

E.3 Proof of Proposition 2

Proof of Proposition 2 The idea of the proof is to show that if an adversary (represented by a context C) distinguishes $[\![L]\!]$ from $[\![R]\!]$, then we can build an adversary $\mathcal{A}_a$ against the security of the mac for the key $\mathrm{mkgen}(r[a])$, for some $a \in I_\eta(n'')$.

Let C be an evaluation context acceptable for $[\![L]\!]$, $[\![R]\!]$, $\emptyset$.

We define a probabilistic polynomial Turing machine $\mathcal{A}_a$, for $a \in [1, I_\eta(n'')]$, as follows. $\mathcal{A}_a$ uses oracles $\mathrm{mac}(., k)$ and $\mathrm{check}(., k, .)$. $\mathcal{A}_a$ simulates $C[[\![L]\!]]$ except that:

- for $a' < a$, in copies corresponding to $i'' = a'$ of L, $\mathcal{A}_a$ computes $\mathsf{find}\ u \leq n\ \mathsf{suchthat}\ \mathsf{defined}(x[u]) \wedge (m = x[u]) \wedge \mathrm{check}(m, \mathrm{mkgen}(r), ma)\ \mathsf{then}\ \mathrm{true}\ \mathsf{else}\ \mathrm{false}$ instead of $\mathrm{check}(m, \mathrm{mkgen}(r), ma)$, and
- in the copy corresponding to $i'' = a$, $\mathcal{A}_a$ does not choose a random number $r[a]$, it calls the oracle $\mathrm{mac}(., k)$ on x instead of computing $\mathrm{mac}(x, \mathrm{mkgen}(r))$, and instead of computing $\mathrm{check}(m, \mathrm{mkgen}(r), ma)$, it computes $b_1 = \mathrm{check}(m, k, ma)$ using the oracle $\mathrm{check}(., k, .)$ and $b_2 = \mathsf{find}\ u \leq n\ \mathsf{suchthat}\ \mathsf{defined}(x[u]) \wedge (m = x[u]) \wedge b_1\ \mathsf{then}\ \mathrm{true}\ \mathsf{else}\ \mathrm{false}$; if $b_1 \neq b_2$, the execution of the Turing machine stops, with result (m, ma); otherwise, the execution continues using value $b_1 = b_2$.

When $\mathcal{A}_a$ has not stopped due to the last item above, it returns $\perp$ when the simulation of $C[[\![L]\!]]$ terminates.

When $\mathcal{A}_a$ returns (m,t), $b_1 \neq b_2$. Moreover, if $b_1 = 0$, then $b_2 = 0$ by definition of b_2. So $b_1 = 1$ and $b_2 = 0$. Therefore, there is no u such that $m = x[u]$, hence $\mathcal{A}_a$ has not called the oracle $\mathrm{mac}(.,k)$ on m. Moreover, there exists a polynomial q such that for all a, $\mathcal{A}_a$ runs in time $q(\eta)$. So by Definition 1, $\max_a p_a(\eta)$ is negligible, where

$$p_a(\eta) = \Pr\begin{bmatrix} r \stackrel{R}{\leftarrow} I_\eta(T_{mr}); k \leftarrow \mathrm{mkgen}_\eta(r); (m,t) \leftarrow \mathcal{A}_a : \\ \mathrm{check}_\eta(m,k,t) \end{bmatrix}$$

Since $I_\eta(n'')$ is polynomial in η, $\sum_{a\in[1,I_\eta(n'')]} p_a(\eta) \leq \max_a p_a(\eta) \times I_\eta(n'')$ is also negligible.

On the other hand, let c be a channel and a' be a bitstring. We need to evaluate $|\Pr[C[\![L]\!]] \rightsquigarrow_\eta \overline{c}\langle a'\rangle] - \Pr[C[\![R]\!]] \rightsquigarrow_\eta \overline{c}\langle a'\rangle]|$. We consider three categories of pairs of traces $(\mathcal{T}, \mathcal{T}')$ where $\mathcal{T}$ and $\mathcal{T}'$ are traces of $C[\![L]\!]]$ and $C[\![R]\!]]$ respectively:

1. Traces $\mathcal{T}$ and $\mathcal{T}'$ have the same configurations except for the replacement of L with R in processes, they terminate, and none of their configurations executes $\overline{c}\langle a'\rangle$ immediately.

2. Traces $\mathcal{T}$ and $\mathcal{T}'$ have the same configurations except for the replacement of L with R in processes up to a point at which their corresponding configurations both execute $\overline{c}\langle a'\rangle$ immediately.

3. Traces $\mathcal{T}$ and $\mathcal{T}'$ have the same configurations except for the replacement of L with R in processes up to a point at which their configurations differ because for some $a \in [1, I_\eta(n'')]$, for some messages m, ma received on channel $c_2[a]$ (where c_2 is the channel used in $[\![L]\!]$ and $[\![R]\!]$ for the second parallel process of L and R), the result returned by $[\![L]\!]$ differs from the one returned by $[\![R]\!]$. In this case, the simulating Turing machine that runs $r \stackrel{R}{\leftarrow} I_\eta(T_{mr}); k \leftarrow \mathrm{mkgen}_\eta(r)$ and executes $\mathcal{A}_a$ will return (m, ma), by construction.

All traces of $C[\![L]\!]]$ fall in one of the above categories, and similarly for traces of $C[\![R]\!]]$. Traces of the first category have no contribution to $\Pr[C[\![L]\!]] \rightsquigarrow_\eta \overline{c}\langle a'\rangle]$ and to $\Pr[C[\![R]\!]] \rightsquigarrow_\eta \overline{c}\langle a'\rangle]$; traces of the second category cancel out when computing $\Pr[C[\![L]\!]] \rightsquigarrow_\eta \overline{c}\langle a'\rangle] - \Pr[C[\![R]\!]] \rightsquigarrow_\eta \overline{c}\langle a'\rangle]$. So

$$\begin{aligned}
&|\Pr[C[\![L]\!]] \rightsquigarrow_\eta \overline{c}\langle a'\rangle] - \Pr[C[\![R]\!]] \rightsquigarrow_\eta \overline{c}\langle a'\rangle]| \\
&\leq \Pr[(\mathcal{T}, \mathcal{T}') \text{ is in the third category}] \\
&\leq \sum_{a\in[1,I_\eta(n'')]} \Pr[r \stackrel{R}{\leftarrow} I_\eta(T_{mr}); k \leftarrow \mathrm{mkgen}_\eta(r); (m,t) \leftarrow \mathcal{A}_a] \\
&\leq \sum_{a\in[1,I_\eta(n'')]} p_a(\eta)
\end{aligned}$$

Hence $|\Pr[C[\![L]\!]] \rightsquigarrow_\eta \overline{c}\langle a'\rangle] - \Pr[C[\![R]\!]] \rightsquigarrow_\eta \overline{c}\langle a'\rangle]|$ is negligible, so $[\![L]\!] \approx [\![R]\!]$. □

E.4 Proof of Proposition 3

Let us first introduce some notations. We denote by $L_{j_0,\dots,j_k}$ the subtrees of L defined as follows by induction on k. We define $L_1, \dots, L_{m'}$ such that $L = (L_1, \dots, L_{m'})$. The functional process $L_{j_0,\dots,j_k}$ being defined, we define $L_{j_0,\dots,j_k,1}, \dots, L_{j_0,\dots,j_k,m'}$ to be the immediate sub-functional-processes of $L_{j_0,\dots,j_k}$, so that $L_{j_0,\dots,j_k}$ is of the form $!^{i\leq n}\mathsf{new}\ y_1 : T_1; \dots; \mathsf{new}\ y_m : T_m; (L_{j_0,\dots,j_k,1}, \dots, L_{j_0,\dots,j_k,m'})$.

When $L_{j_0,\dots,j_k} = !^{i\leq n}\mathsf{new}\ y_1 : T_1; \dots; \mathsf{new}\ y_m : T_m; (L_{j_0,\dots,j_k,1}, \dots, L_{j_0,\dots,j_k,m'})$, we define $i_{j_0,\dots,j_k} = i$, $n_{j_0,\dots,j_k} = n$, $y_{(j_0,\dots,j_k),k'} = y_{k'}$, and $\mathrm{nNew}_{j_0,\dots,j_k} = m$.

When $L_{j_0,\dots,j_l} = (x_1 : T_1, \dots, x_m : T_m) \rightarrow FP$, we say that $L_{j_0,\dots,j_l}$ is a leaf of L, and we define $x_{(j_0,\dots,j_l),k'} = x_{k'}$, $T_{(j_0,\dots,j_l),k'} = T_{k'}$, and $\mathrm{nInput}_{j_0,\dots,j_l} = m$.

In order to prove Proposition 3, we define a context C such that $Q_0 \approx_0^V C[\![L]\!]]$ and $C[\![R]\!]] \approx_0^V Q_0'$. While Q_0 evaluates the terms in $\mathcal{M}$ directly, the context C will send messages to $[\![L]\!]$ in order to evaluate these terms in $C[\![L]\!]]$. Similarly, the process Q_0' contains inlined versions of the functional processes in R, while $C[\![R]\!]]$ computes the same result by sending messages to $[\![R]\!]$.

In order to define C, we first define a process $\mathrm{relay}(L)$ as follows:

$$\begin{aligned}
&\mathrm{relay}((G_1, \dots, G_m)) = \mathrm{relay}(G_1)^1 \mid \dots \mid \mathrm{relay}(G_m)^m \\
&\mathrm{relay}(!^{i\leq n}\mathsf{new}\ y_1 : T_1; \dots; \mathsf{new}\ y_l : T_l; (G_1, \dots, G_m))^{\widetilde{j}}_{\widetilde{i}} = \\
&\quad !^{i\leq n} d_{\widetilde{j}}[\widetilde{i}, i](); \overline{c_{\widetilde{j}}[\widetilde{i}, i]}\langle\rangle; c_{\widetilde{j}}[\widetilde{i}, i](); \overline{d_{\widetilde{j}}[\widetilde{i}, i]}\langle\rangle; \\
&\qquad (\mathrm{relay}(G_1)^{\widetilde{j},1}_{\widetilde{i},i} \mid \dots \mid \mathrm{relay}(G_m)^{\widetilde{j},m}_{\widetilde{i},i} \mid \\
&\qquad\ !^{i'\leq n'} d_{\widetilde{j}}[\widetilde{i}, i](); \overline{d_{\widetilde{j}}[\widetilde{i}, i]}\langle\rangle) \\
&\mathrm{relay}((x_1 : T_1, \dots, x_l : T_l) \rightarrow FP)^{\widetilde{j}}_{\widetilde{i}} = \\
&\quad d_{\widetilde{j}}[\widetilde{i}](x_1 : T_1, \dots, x_l : T_l); \overline{c_{\widetilde{j}}[\widetilde{i}]}\langle x_1, \dots, x_l\rangle; \\
&\quad c_{\widetilde{j}}[\widetilde{i}](r : \mathit{bitstring}); \overline{d_{\widetilde{j}}[\widetilde{i}]}\langle r\rangle; \\
&\quad !^{i'\leq n'} d_{\widetilde{j}}[\widetilde{i}](x_1 : T_1, \dots, x_l : T_l); \overline{d_{\widetilde{j}}[\widetilde{i}]}\langle r\rangle
\end{aligned}$$

where $\widetilde{i} = i_1, \dots, i_{l'}$ and $\widetilde{j} = j_0, \dots, j_{l'}$. The relay process corresponding to replicated restrictions relays messages sent on channel $d_{\widetilde{j}}$ to channel $c_{\widetilde{j}}$ (used in $[\![L]\!]$ and $[\![R]\!]$) so that the corresponding random numbers $y_1, \dots, y_l$ are chosen by $[\![L]\!]$. When those random numbers have already been chosen, the process accepts messages on $d_{\widetilde{j}}$ but yields control back to the sending process without executing anything by outputting on $d_{\widetilde{j}}$. Thus, the caller of the relay process can harmlessly ask several times for choosing the same random numbers. Similarly, the relay process corresponding to a function relays the arguments of the function received on channel $d_{\widetilde{j}}$ to channel $c_{\widetilde{j}}$, so that $[\![L]\!]$ replies on channel $c_{\widetilde{j}}$ with the result r of the function, which is forwarded to channel $d_{\widetilde{j}}$. The relay process also allows calling several times the same function with the same values of $\widetilde{j}$ and $\widetilde{i}$, in which case it always returns the same result r. (We make sure in the following that when a function is called several times, the calls all use the same arguments.) Since L and R are required to have the same structure by Hypothesis H2, $\mathrm{relay}(L) = \mathrm{relay}(R)$.

We introduce the following auxiliary definitions, which allow us to define the correspondence mapIdx_M from replication indices at M in Q_0 to replication indices at N_M in L:

- For each $M \in \mathcal{M}$ and $k \leq \mathrm{nNewSeq}_M$, we define $\mathrm{count}_\eta(k, M)$ as follows. Let $n_1, \dots, n_l$ be the sequence of bounds of replications above the definition of $z_{kk',M}$ for any k'. Let l' be the length of the longest common prefix of $\mathrm{im\ index}_k(M)$ and $\mathrm{im\ index}_{k_0}(M)$ for $k_0 < k$. We define $\mathrm{count}_\eta(k, M) = I_\eta(n_{l'+1}) \times \dots \times I_\eta(n_l)$. We define parameters $\mathrm{count}_{k,M}$ such that $I_\eta(\mathrm{count}_{k,M}) = \mathrm{count}_\eta(k, M)$.

  We define function symbols $\mathrm{num}_{k,M} : [1, n_1] \times \dots \times [1, n_l] \to [1, \mathrm{count}_{k,M}]$ such that $I_\eta(\mathrm{num}_{k,M})(a_1, \dots, a_l) = 1 + (a_{l'+1} - 1) + I_\eta(n_{l'+1}) \times ((a_{l'+2} - 1) + I_\eta(n_{l'+2}) \times \dots + I_\eta(n_{l-1}) \times (a_l - 1))$. Then $\mathrm{num}_{k,M}$ establishes a bijection between the last $l - l'$ components of its argument and its result.

- We define $\mathrm{tot\_count}_\eta(j_0, \dots, j_k)$ as the sum of $\mathrm{count}_\eta(k+1, M'')$ for all M'' such that the first $k+1$ elements of $BL(M'')$ are equal to $j_0, \dots, j_k$, counting only once terms M'' that share the first $k+1$ sequences of random variables.

  We set $I_\eta(n_{j_0,\dots,j_k}) = \mathrm{tot\_count}_\eta(j_0, \dots, j_k)$, where $n_{j_0,\dots,j_k}$ is the bound of the replication at the root of $L_{j_0,\dots,j_k}$ in L. The value of $I_\eta(n_{j_0,\dots,j_k})$ is then large enough so that there is always an available copy of the desired replicated process when we need to execute one.

  The replication at the root of $\mathrm{relay}(L_{j_0,\dots,j_k})^{j_0,\dots,j_k}_{i_1,\dots,i_k}$ is also bounded by $n_{j_0,\dots,j_k}$. The other replication of $\mathrm{relay}(L_{j_0,\dots,j_k})^{j_0,\dots,j_k}_{i_1,\dots,i_k}$ is bounded by n', where $I_\eta(n')$ is the sum for all $M \in \mathcal{M}$ of $I_\eta(n_1) \times \dots \times I_\eta(n_l)$ where $n_1, \dots, n_l$ is the sequence of bounds of replications above M in Q_0.

- We order the term occurrences in $\mathcal{M}$ arbitrarily, with a total ordering. Let $\mathrm{start}_\eta(k, M)$ be defined as follows. Let M' the smallest (in the chosen ordering of $\mathcal{M}$) term occurrence of $\mathcal{M}$ that shares the first k sequences of random variables with M. Then $\mathrm{start}_\eta(k, M)$ is the sum of $\mathrm{count}_\eta(k, M'')$ for all M'' smaller than M' such that the first k elements of $BL(M'')$ are equal to the first k elements of $BL(M')$, counting only once terms M'' that share the first k sequences of random variables.

  We define function symbols $\mathrm{addstart}_{k,M} : [1, \mathrm{count}_{k,M}] \to [1, n_{j_0,\dots,j_k}]$ where $BL(M) = (j_0, \dots, j_k, \dots)$, such that $I_\eta(\mathrm{addstart}_{k,M})(a) = \mathrm{start}_\eta(k, M) + a$.

- Let us define $\mathrm{convindex}(k, M)$ as the sequence of terms

$$\begin{aligned}&\mathrm{convindex}(k, M) = \\ &\quad(\mathrm{addstart}_{1,M}(\mathrm{num}_{1,M}(\mathrm{im\ index}_1(M))), \\ &\qquad \dots, \mathrm{addstart}_{k,M}(\mathrm{num}_{k,M}(\mathrm{im\ index}_k(M))))\end{aligned}$$

  This sequence of terms implements the function mapIdx_M mentioned in the explanation of the transformation, in Section 3.2. More precisely, $\mathrm{mapIdx}_M(\widetilde{a}) = \mathrm{convindex}(l, M)\{\widetilde{a}/\widetilde{i}\}$, where $\widetilde{i}$ is the sequence of current replication indices at M and $l = \mathrm{nNewSeq}_M$.

Then we define $C = (\mathsf{newChannel}\ c_{\widetilde{j}}; \mathsf{newChannel}\ d_{\widetilde{j}};)_{\widetilde{j}}([\,] \mid \mathrm{relay}(L) \mid Q''_0)$ where the process Q''_0 is defined from Q_0 as follows:

- When $x \in S$, we replace its definition $\mathsf{new}\ x : T; Q$ with $\mathsf{let}\ x : T = cst\ \mathsf{in}\ Q$ for some constant cst.

- For each $M \in \mathcal{M}$, let $P_M = C_M[M]$ be the smallest subprocess of Q_0 containing M. Let $l = \mathrm{nNewSeq}_M$ and $m = \mathrm{nInput}_M$. Let $BL(M) = (j_0, \dots, j_l)$. Let $d_M = d_{j_0,\dots,j_l}[\mathrm{convindex}(l, M)]$ and for all $k \leq l$, $d_{M,k} = d_{j_0,\dots,j_{k-1}}[\mathrm{convindex}(k, M)]$. We replace P_M with $\overline{d_{M,1}}\langle\rangle; d_{M,1}(); \dots \overline{d_{M,l}}\langle\rangle; d_{M,l}(); \overline{d_M}\langle\sigma_M x_{1,M}, \dots, \sigma_M x_{m,M}\rangle; d_M(y : \mathit{bitstring}); C_M[y]$ where y is a fresh variable.

Instead of evaluating the terms $M \in \mathcal{M}$ directly as in Q_0, Q''_0 sends messages to the relay process $\mathrm{relay}(L)$, which will then forward them to $[\![L]\!]$ in $C[[\![L]\!]]$ and to $[\![R]\!]$ in $C[[\![R]\!]]$.

Lemma 11 $Q_0 \approx^V_0 C[[\![L]\!]]$

Proof The bounds of replications of $[\![L]\!]$ and $\mathrm{relay}(L)$ have been defined above. As outlined in the proof of Proposition 6, the length of all bitstrings manipulated by Q_0 is polynomial in η. We can therefore define $\mathrm{maxlen}_\eta(c_{\widetilde{j}})$ to be a polynomial large enough so that messages sent on $c_{\widetilde{j}}$ by $C[[\![L]\!]]$ are never truncated. We define $\mathrm{maxlen}_\eta(d_{\widetilde{j}}) = \mathrm{maxlen}_\eta(c_{\widetilde{j}})$; then messages on $d_{\widetilde{j}}$ are never truncated.

Let C' be any evaluation context acceptable for Q_0, $C[[\![L]\!]]$, V. We relate traces of $C'[Q_0]$ and of $C'[C[[\![L]\!]]]$ as follows.

We assume that the channels $c_{\widetilde{j}}$ and $d_{\widetilde{j}}$ do not occur in C' and Q_0, and that during reductions (NewChannel), these channels are substituted by themselves. (This is easy to guarantee by renaming; this assumption simplifies notations in the proof.)

We write $M =_E M'$ when $E, M \Downarrow a$ and $E, M' \Downarrow a$ for some bitstring a. We denote by $k\text{-th}(\widetilde{i})$ the k-th component of the tuple $\widetilde{i}$, and by $|\widetilde{i}|$ the number of elements of the tuple $\widetilde{i}$.

We define a relation between variables of S in Q_0 and variables y defined by new in $[\![L]\!]$: we say that $y[a_1, \dots, a_j] \xrightarrow{\mathrm{var}}_E \mathrm{varImL}(y, M)[\widetilde{a'}]$ when for all $k' \leq j$, $E, \mathrm{addstart}_{k',M}(\mathrm{num}_{k',M}(\mathrm{im}\ (\rho_{j-1}(M) \circ \dots \circ \rho_{k'}(M))\{\widetilde{a'}/\widetilde{i}\})) \Downarrow a_{k'}$, where $\widetilde{i} \leq \widetilde{n}$ are the current replication indices at the definition of $\mathrm{varImL}(y, M)$ with their associated bounds, and for all $l \leq |\widetilde{i}|$, $l\text{-th}(\widetilde{a'}) \in [1, I_\eta(l\text{-th}(\widetilde{n}))]$. (Note that $\xrightarrow{\mathrm{var}}$ depends on η.)

We show that the relation $\xrightarrow{\mathrm{var}}_E$ is a (partial) function, that is, if $y[a_1, \dots, a_j] \xrightarrow{\mathrm{var}}_E M_V$ and $y[a_1, \dots, a_j] \xrightarrow{\mathrm{var}}_E M'_V$ then $M_V = M'_V$. Assume that $y[a_1, \dots, a_j] \xrightarrow{\mathrm{var}}_E z'[\widetilde{a'}]$ and $y[a_1, \dots, a_j] \xrightarrow{\mathrm{var}}_E z''[\widetilde{a''}]$. Then

- we have $z' = \mathrm{varImL}(y, M')$ and

$$\begin{aligned}&E, \mathrm{addstart}_{k',M'}(\mathrm{num}_{k',M'}(\mathrm{im}\ (\rho_{j-1}(M') \circ \\ &\quad \dots \circ \rho_{k'}(M'))\{\widetilde{a'}/\widetilde{i'}\})) \Downarrow a_{k'} \text{ for all } k' \leq j\end{aligned}$$

  where $\widetilde{i'} \leq \widetilde{n'}$ are the current replication indices at the definition of z' with their associated bounds, and for all $l \leq |\widetilde{i'}|$, $l\text{-th}(\widetilde{a'}) \in [1, I_\eta(l\text{-th}(\widetilde{n'}))]$,

- we have $z'' = \mathrm{varImL}(y, M'')$ and

$$\begin{aligned}&E, \mathrm{addstart}_{k',M''}(\mathrm{num}_{k',M''}(\mathrm{im}\ (\rho_{j-1}(M'') \circ \\ &\quad \dots \circ \rho_{k'}(M''))\{\widetilde{a''}/\widetilde{i''}\})) \Downarrow a_{k'} \text{ for all } k' \leq j\end{aligned}$$

where $\widetilde{i''} \leq \widetilde{n''}$ are the current replication indices at the definition of z'' with their associated bounds, and for all $l \leq |\widetilde{i''}|$, $l\text{-th}(\widetilde{a''}) \in [1, I_\eta(l\text{-th}(\widetilde{n''}))]$.

For all terms M'', we have either $\mathrm{start}_\eta(k', M'') \leq \mathrm{start}_\eta(k', M')$ or $\mathrm{start}_\eta(k', M'') \geq \mathrm{start}_\eta(k', M') + \mathrm{count}_\eta(k', M')$ since $\mathrm{start}_\eta(k', M'')$ is computed by adding $\mathrm{count}_\eta(k', M_3)$ for some terms M_3 in a fixed order. Moreover, $\mathrm{num}_{k',M'}(\dots)$ evaluates to a bitstring in $[1, \mathrm{count}_\eta(k', M')]$. Therefore, $\mathrm{start}_\eta(k', M'') \leq \mathrm{start}_\eta(k', M')$. By symmetry, $\mathrm{start}_\eta(k', M'') \geq \mathrm{start}_\eta(k', M')$. So we have for all $k' \leq j$, $\mathrm{start}_\eta(k', M') = \mathrm{start}_\eta(k', M'')$ and $\mathrm{num}_{k',M'}(\mathrm{im}\ (\rho_{j-1}(M') \circ \ldots \circ \rho_{k'}(M'))\{\widetilde{a'}/\widetilde{i'}\}) =_E \mathrm{num}_{k',M''}(\mathrm{im}\ (\rho_{j-1}(M'') \circ \ldots \circ \rho_{k'}(M''))\{\widetilde{a''}/\widetilde{i''}\})$. Since $\mathrm{start}_\eta(j, M') = \mathrm{start}_\eta(j, M'')$, by definition of start_η, M' shares the first j sequences of random variables with M''. Since y has j indices, y is defined under j replications in L, so $\mathrm{varImL}(y, M') = \mathrm{varImL}(y, M'')$, that is, $z' = z''$. So $|\widetilde{a'}| = |\widetilde{a''}|$. By Hypothesis H′4.2, $\rho_{k'}(M') = \rho_{k'}(M'')$ for all $k' < j$. By definition of num, $I_\eta(\mathrm{num}_{k',M'}) = I_\eta(\mathrm{num}_{k',M''})$ for all $k' \leq j$.

We show by induction on k' that if for all $k'' \leq k'$, $\mathrm{num}_{k'',M'}(\mathrm{im}\ (\rho_{k'-1}(M') \circ \ldots \circ \rho_{k''}(M'))\{\widetilde{a'}/\widetilde{i'}\}) =_E \mathrm{num}_{k'',M'}(\mathrm{im}\ (\rho_{k'-1}(M') \circ \ldots \circ \rho_{k''}(M'))\{\widetilde{a''}/\widetilde{i'}\})$, where $\widetilde{i'} \leq \widetilde{n'}$ are the current replication indices at the definition of $z_{k'\text{-},M'}$ with their associated bounds, and $l\text{-th}(\widetilde{a'}), l\text{-th}(\widetilde{a''}) \in [1, I_\eta(l\text{-th}(\widetilde{n'}))]$, then $\widetilde{a'} = \widetilde{a''}$.

- For $k' = 1$, we assume $\mathrm{num}_{1,M'}(\widetilde{a'}) =_E \mathrm{num}_{1,M'}(\widetilde{a''})$. The longest common prefix of $\mathrm{index}_1(M')$ and $\mathrm{index}_{j''}(M')$ for $j'' < 1$ is empty, since $\mathrm{index}_{j''}(M')$ is defined only for $j'' \geq 1$. So $\mathrm{num}_{1,M'}$ establishes a bijection between the tuples $\widetilde{a'}$ smaller than the current replication bounds at definition of $z_{1\text{-},M'}$ and the interval $[1, \mathrm{count}_\eta(1, M')]$. So $\widetilde{a'} = \widetilde{a''}$.

- For $k' > 1$, we assume that $\mathrm{num}_{k'',M'}(\mathrm{im}\ (\rho_{k'-1}(M') \circ \ldots \circ \rho_{k''}(M'))\{\widetilde{a'}/\widetilde{i'}\}) =_E \mathrm{num}_{k'',M'}(\mathrm{im}\ (\rho_{k'-1}(M') \circ \ldots \circ \rho_{k''}(M'))\{\widetilde{a''}/\widetilde{i'}\})$ for all $k'' \leq k'$. Let $k'_{\mathrm{ind}} < k'$. Let $E, \mathrm{im}\ (\rho_{k'-1}(M') \circ \ldots \circ \rho_{k'_{\mathrm{ind}}}(M'))\{\widetilde{a'}/\widetilde{i'}\} \Downarrow \widetilde{a'}_{\mathrm{ind}}$ and $E, \mathrm{im}\ (\rho_{k'-1}(M') \circ \ldots \circ \rho_{k'_{\mathrm{ind}}}(M'))\{\widetilde{a''}/\widetilde{i'}\} \Downarrow \widetilde{a''}_{\mathrm{ind}}$. By hypothesis, we have for all $k'' \leq k'_{\mathrm{ind}}$, $\mathrm{num}_{k'',M'}(\mathrm{im}\ (\rho_{k'_{\mathrm{ind}}-1}(M') \circ \ldots \circ \rho_{k''}(M'))\{\widetilde{a'}_{\mathrm{ind}}/\widetilde{i'}_{\mathrm{ind}}\}) =_E \mathrm{num}_{k'',M'}(\mathrm{im}\ (\rho_{k'_{\mathrm{ind}}-1}(M') \circ \ldots \circ \rho_{k''}(M'))\{\widetilde{a''}_{\mathrm{ind}}/\widetilde{i'}_{\mathrm{ind}}\})$ where $\widetilde{i'}_{\mathrm{ind}} \leq \widetilde{n'}_{\mathrm{ind}}$ are the current replication indices at the definition of $z_{k'_{\mathrm{ind}}\text{-},M'}$ with their associated bounds. By induction hypothesis, $\widetilde{a'}_{\mathrm{ind}} = \widetilde{a''}_{\mathrm{ind}}$, so for all $k'' < k'$, $\mathrm{im}\ (\rho_{k'-1}(M') \circ \ldots \circ \rho_{k''}(M'))\{\widetilde{a'}/\widetilde{i'}\} =_E \mathrm{im}\ (\rho_{k'-1}(M') \circ \ldots \circ \rho_{k''}(M'))\{\widetilde{a''}/\widetilde{i'}\}$. For $k'' = k'$, we have $\mathrm{num}_{k',M'}(\widetilde{a'}) =_E \mathrm{num}_{k',M'}(\widetilde{a''})$.

  Let l be the length of the longest common prefix of $\mathrm{im\ index}_{k'}(M')$ and $\mathrm{im\ index}_{k''_0}(M')$ for $k''_0 < k'$. Since $\mathrm{index}_{k''_0}(M') = \mathrm{index}_{k'}(M') \circ \rho_{k'-1}(M') \circ \ldots \circ \rho_{k''_0}(M')$, the first l components of $\mathrm{im}\ (\rho_{k'-1}(M') \circ \ldots \circ \rho_{k''_0}(M'))$ are then the first l components of $\widetilde{i'}$, so the first l components of $\widetilde{a'}$ and $\widetilde{a''}$ are equal. Moreover $\mathrm{num}_{k',M'}$ establishes a bijection between the last $|\widetilde{a'}| - l$ components of its argument and the interval $[1, \mathrm{count}_\eta(k', M')]$. So the last $|\widetilde{a'}| - l$ components of $\widetilde{a'}$ and $\widetilde{a''}$ are equal. Hence $\widetilde{a'} = \widetilde{a''}$.

Therefore, we conclude that $\widetilde{a'} = \widetilde{a''}$, so $z'[\widetilde{a'}] = z''[\widetilde{a''}]$.

Next, we show that the function $\xrightarrow{\mathrm{var}}_E$ is injective. If $y'[a'_1, \ldots, a'_{j'}] \xrightarrow{\mathrm{var}}_E z[a_1, \ldots, a_j]$ and $y''[a''_1, \ldots, a''_{j''}] \xrightarrow{\mathrm{var}}_E z[a_1, \ldots, a_j]$, then $z = \mathrm{varImL}(y', M')$ and $z = \mathrm{varImL}(y'', M'')$. By Hypothesis H′4.1, M' and M'' share at least the first $j' = j''$ sequences of random variables and $y' = y''$. By Hypothesis H′4.2, $\rho_{k'}(M') = \rho_{k'}(M'')$ for all $k' < j' = j''$. By definition of addstart and num, $\mathrm{start}_\eta(k', M') = \mathrm{start}_\eta(k', M'')$ and $I_\eta(\mathrm{num}_{k',M'}) = I_\eta(\mathrm{num}_{k',M''})$ for all $k' \leq j' = j''$. Hence $a'_{k'} = a''_{k'}$ for all $k' \leq j' = j''$. So $y'[a'_1, \ldots, a'_{j'}] = y''[a''_1, \ldots, a''_{j''}]$.

For each trace $\mathrm{initConfig}(C'[Q_0]) \to \ldots \to E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m$ of $C'[Q_0]$ of probability p_m, we show that there exists a trace $\mathrm{initConfig}(C'[C[\llbracket L \rrbracket]]) \to \ldots \to E'_{m'}, P'_{m'}, \mathcal{Q}'_{m'}, \mathcal{C}'_{m'}$ of $C'[C[\llbracket L \rrbracket]]$ of probability $p'_{m'}$ such that

- For all $z \notin S$, $E'_{m'}(z[a'_1, \ldots, a'_{j'}]) = E_m(z[a'_1, \ldots, a'_{j'}])$; for all $z \in S$, $z[a'_1, \ldots, a'_{j'}]$ is in $\mathrm{Dom}(E_m)$ if and only if it is in $\mathrm{Dom}(E'_{m'})$; if y is defined by new in L and $y[a_1, \ldots, a_j] \in \mathrm{Dom}(E'_{m'})$ then there exists M_V such that $y[a_1, \ldots, a_k] \xrightarrow{\mathrm{var}}_{E_m} M_V$ and $M_V \in \mathrm{Dom}(E_m)$ and for all such M_V, $E'_{m'}(y[a_1, \ldots, a_j]) = E_m(M_V)$.

- $P'_{m'}$ is obtained from P_m as Q''_0 from Q_0 (transforming only the occurrences that appear in P_m), $\mathcal{Q}'_{m'} = \mathcal{Q}^1_{m'} \uplus \mathcal{Q}^2_{m'} \uplus \mathcal{Q}^3_{m'}$, where $\mathcal{Q}^1_{m'}$ is obtained from $\mathcal{Q}_m$ as Q''_0 from Q_0 (transforming only the occurrences that appear in $\mathcal{Q}_m$), $\mathcal{Q}^2_{m'}$ is what remains of $\mathrm{relay}(L)$ after partial execution, and $\mathcal{Q}^3_{m'}$ is what remains of $\llbracket L \rrbracket$ after partial execution. More precisely, let

$$\mathrm{relay}(L^{a_1,\ldots,a_k}_{j_0,\ldots,j_k}) = \mathrm{relay}(L_{j_0,\ldots,j_k})^{j_0,\ldots,j_k}_{i_1,\ldots,i_k}\{a_1/i_1, \ldots, a_k/i_k\}$$
$$\llbracket L^{a_1,\ldots,a_k}_{j_0,\ldots,j_k} \rrbracket = \llbracket L_{j_0,\ldots,j_k} \rrbracket^{j_0,\ldots,j_k}_{i_1,\ldots,i_k}\{a_1/i_1, \ldots, a_k/i_k\}$$

  where $i_1, \ldots, i_k$ are the replications indices of L above $L_{j_0,\ldots,j_k}$. These processes correspond respectively to the relay process and to the translation of the subtree $L_{j_0,\ldots,j_k}$ of L, for the value of the replication indices $a_1, \ldots, a_k$. Let $\mathrm{redRepl}(a, !^{i \leq n} P) = P\{a/i\}$. Then $\mathcal{Q}^2_{m'}$ and $\mathcal{Q}^3_{m'}$ are formed as follows:

  - for each $j_0, \ldots, j_{k-1}, a_1, \ldots, a_k$ such that
  $$y_{(j_0,\ldots,j_{k-1}),k'}[a_1, \ldots, a_k] \in \mathrm{Dom}(E'_{m'}),$$
  $\mathcal{Q}^2_{m'}$ contains
  $$d_{j_0,\ldots,j_{k-1}}[a_1, \ldots, a_k](); \overline{d_{j_0,\ldots,j_{k-1}}[a_1, \ldots, a_k]}\langle\rangle$$
  possibly several times.
  - for each $j_0, \ldots, j_{k-1}, a_1, \ldots, a_k$ such that
  $$y_{(j_0,\ldots,j_{k-2}),k''}[a_1, \ldots, a_{k-1}] \in \mathrm{Dom}(E'_{m'}) \text{ and}$$
  $$y_{(j_0,\ldots,j_{k-1}),k'}[a_1, \ldots, a_k] \notin \mathrm{Dom}(E'_{m'}),$$

$\mathcal{Q}^2_{m'}$ contains $\mathrm{redRepl}(a_k, \mathrm{relay}(L^{a_1,\dots,a_{k-1}}_{j_0,\dots,j_{k-1}}))$ and $\mathcal{Q}^3_{m'}$ contains $\mathrm{redRepl}(a_k, [\![L^{a_1,\dots,a_{k-1}}_{j_0,\dots,j_{k-1}}]\!])$.

– for each $j_0,\dots,j_l, a_1,\dots,a_l$ such that

$$y_{(j_0,\dots,j_{l-1}),k'}[a_1,\dots,a_l] \in \mathrm{Dom}(E'_{m'})$$

and $L_{j_0,\dots,j_l}$ is a leaf of L, either $\mathcal{Q}^2_{m'}$ contains $\mathrm{relay}(L^{a_1,\dots,a_l}_{j_0,\dots,j_l})$ and $\mathcal{Q}^3_{m'}$ contains $[\![L^{a_1,\dots,a_l}_{j_0,\dots,j_l}]\!]$, or $\mathcal{Q}^2_{m'}$ contains

$$d_{j_0,\dots,j_l}[a_1,\dots,a_l](x_{(j_0,\dots,j_l),1} : T_{(j_0,\dots,j_l),1},\dots,$$
$$x_{(j_0,\dots,j_l),l'} : T_{(j_0,\dots,j_l),l'}); \overline{d_{j_0,\dots,j_l}[a_1,\dots,a_l]}\langle r\rangle$$

with $l' = \mathrm{nInput}_{j_0,\dots,j_l}$, possibly several times, and there exist $M' \in \mathcal{M}$ and $\widetilde{a'}$ such that $E_m, \mathrm{convindex}(l, M')\{\widetilde{a'}/\widetilde{i'}\} \Downarrow a_1,\dots,a_l$, $E_m, M'\{\widetilde{a'}/\widetilde{i'}\} \Downarrow r$, and $BL(M') = (j_0,\dots,j_l)$, where $\widetilde{i'}$ is the sequence of replication indices at M'.

where for each k, a_k is a bitstring in $[1, \mathrm{tot\_count}_\eta(j_0,\dots, j_{k-1})]$.

- $\mathcal{C}'_{m'} = \mathcal{C}_m \cup \{c_{\widetilde{j}}, d_{\widetilde{j}} \mid \widetilde{j}\}$.
- $p'_{m'} = p_m \times \prod_{z,a'_1,\dots,a'_{j'}} |I_\eta(T)|$ where T is the type of z and $z \in S, a'_1,\dots,a'_{j'}$ are such that $z[a'_1,\dots,a'_{j'}] \in \mathrm{Dom}(E_m)$ and there exists no $y[a_1,\dots,a_j] \in \mathrm{Dom}(E'_{m'})$ such that $y[a_1,\dots,a_j] \xrightarrow{\mathrm{var}}_{E_m} z[a'_1,\dots,a'_{j'}]$.

Note that the same trace of $C'[C[[\![L]\!]]]$ corresponds to $\prod_{z,a'_1,\dots,a'_{j'}} |I_\eta(T)|$ traces of $C'[Q_0]$ that differ only by the values of $E_m(z[a'_1,\dots,a'_{j'}])$ for $z \in S, a'_1,\dots,a'_{j'}$ as defined in the last item above.

The proof proceeds by induction on the length m of the trace of $C'[Q_0]$. For the induction step, we distinguish cases depending on the last reduction step of the trace.

- For the initial case, we show by induction on C'' that for all $C'', \mathcal{Q}, \mathcal{C}, \sigma$ such that σ substitutes channel names for channel names without touching $c_{\widetilde{j}}$ and $d_{\widetilde{j}}$, there exist $\mathcal{Q}', \mathcal{C}', \sigma'$ such that σ' substitutes channel names for channel names without touching $c_{\widetilde{j}}$ and $d_{\widetilde{j}}$, $\emptyset, \{C''[\sigma Q_0]\} \uplus \mathcal{Q}, \mathcal{C} \rightsquigarrow^* \emptyset, \{\sigma' Q_0\} \uplus \mathcal{Q}', \mathcal{C}'$, and $\emptyset, \{C''[\sigma C[[\![L]\!]]]\} \uplus \mathcal{Q}, \mathcal{C} \rightsquigarrow^* \emptyset, \{\sigma' C[[\![L]\!]]\} \uplus \mathcal{Q}', \mathcal{C}'$. This is obvious when $C'' = [\,]$, with $\sigma' = \sigma$, $\mathcal{Q}' = \mathcal{Q}$, and $\mathcal{C}' = \mathcal{C}$. We show this result by applying (Par) when $C'' = C_1 \mid Q_1$ or $C'' = Q_1 \mid C_1$, and (NewChannel) when $C'' = \mathsf{newChannel}\ c; C_1$.

  So we can apply this result to $C'' = C'$, $\sigma = \mathrm{Id}$, $\mathcal{Q} = \emptyset$, and $\mathcal{C} = \mathrm{fc}(C'[Q_0])$. We have $\mathrm{fc}(C'[Q_0]) = \mathrm{fc}(C'[C[[\![L]\!]]])$, since $\mathrm{fc}(Q_0) = \mathrm{fc}(Q''_0) = \mathrm{fc}(C[[\![L]\!]])$. Therefore, there exist $\mathcal{Q}, \mathcal{C}, \sigma$ such that σ substitutes channel names for channel names without touching $c_{\widetilde{j}}$ and $d_{\widetilde{j}}$, $\emptyset, \{C'[Q_0]\}, \mathrm{fc}(C'[Q_0]) \rightsquigarrow^* \emptyset, \{\sigma Q_0\} \uplus \mathcal{Q}, \mathcal{C}$, and

$$\emptyset, \{C'[C[[\![L]\!]]]\}, \mathrm{fc}(C'[C[[\![L]\!]]]) \rightsquigarrow^* \emptyset, \{\sigma C[[\![L]\!]]\} \uplus \mathcal{Q}, \mathcal{C}$$
$$\rightsquigarrow^* \emptyset, \{\sigma Q''_0, \mathrm{relay}(L), [\![L]\!]\} \uplus \mathcal{Q}, \mathcal{C} \cup \{c_{\widetilde{j}}, d_{\widetilde{j}} \mid \widetilde{j}\}$$
by (NewChannel) and (Par)
$$\rightsquigarrow^* \emptyset, \{\sigma Q''_0\} \uplus \mathcal{Q}^2_0 \uplus \mathcal{Q}^3_0 \uplus \mathcal{Q}, \mathcal{C} \cup \{c_{\widetilde{j}}, d_{\widetilde{j}} \mid \widetilde{j}\}$$
by (Par) and (Repl)

  where $\mathcal{Q}^2_0 = \{\mathrm{redRepl}(a, \mathrm{relay}(L_{j_0})^{j_0}) \mid j_0, a \in [1, \mathrm{tot\_count}_\eta(j_0)]\}$ is what remains from $\mathrm{relay}(L)$ after expansion of parallel compositions and replications and $\mathcal{Q}^3_0 = \{\mathrm{redRepl}(a, [\![L_{j_0}]\!]^{j_0}) \mid j_0, a \in [1, \mathrm{tot\_count}_\eta(j_0)]\}$ is what remains of $[\![L]\!]$ after expansion of parallel compositions and replications.

  Moreover, $\sigma Q''_0$ is obtained from σQ_0 as Q''_0 from Q_0, and $\mathcal{Q}$ does not contain any occurrence modified when transforming Q_0 into Q''_0, so $\{\sigma Q''_0\} \uplus \mathcal{Q}$ is obtained from $\{\sigma Q_0\} \uplus \mathcal{Q}$ as Q''_0 from Q_0.

  Reducing $\{\sigma Q''_0\} \uplus \mathcal{Q}$ and $\{\sigma Q_0\} \uplus \mathcal{Q}$ by $\rightsquigarrow$ until they are in normal form, we obtain that $\mathrm{reduce}(\emptyset, \{C'[Q_0]\}, \mathrm{fc}(C'[Q_0])) = (\emptyset, \mathcal{Q}_0, \mathcal{C}')$ and $\mathrm{reduce}(\emptyset, \{C'[C[[\![L]\!]]]\}, \mathrm{fc}(C'[C[[\![L]\!]]])) = (\emptyset, \mathcal{Q}^1_0 \uplus \mathcal{Q}^2_0 \uplus \mathcal{Q}^3_0, \mathcal{C}' \cup \{c_{\widetilde{j}}, d_{\widetilde{j}} \mid \widetilde{j}\})$, where $\mathcal{Q}^1_0$ is obtained from $\mathcal{Q}_0$ as Q''_0 from Q_0. Therefore, $\mathrm{initConfig}(C'[Q_0])$ and $\mathrm{initConfig}(C'[C[[\![L]\!]]])$ satisfy the desired invariant.

- When the trace of $C'[Q_0]$ executes $\mathsf{new}\ x[a_1,\dots,a_l] : T$ by (New) for $x \in S$ at step m, the corresponding trace of $C'[C[[\![L]\!]]]$ executes $\mathsf{let}\ x[a_1,\dots,a_l] : T = cst\ \mathsf{in}$ by (Let) at step m'. This yields $|I_\eta(T)|$ traces of $C'[Q_0]$, one for each value of $E_m(x[a_1,\dots,a_l])$, each with probability $p_m = p_{m-1}/|I_\eta(T)|$. In contrast, this yields a single trace of $C'[C[[\![L]\!]]]$, with probability $p'_{m'} = p'_{m'-1}$.

  Moreover, there exists no $y[a'_1,\dots,a'_{l'}] \in \mathrm{Dom}(E'_{m'})$ such that $y[a'_1,\dots,a'_{l'}] \xrightarrow{\mathrm{var}}_{E_m} x[a_1,\dots,a_l]$. Otherwise, by the first point of the invariant, before the definition of $x[a_1,\dots,a_l]$, there would exist M_V such that $y[a'_1,\dots,a'_{l'}] \xrightarrow{\mathrm{var}}_{E_{m-1}} M_V$ and $M_V \in \mathrm{Dom}(E_{m-1})$. Since E_m is an extension of E_{m-1}, $y[a'_1,\dots,a'_{l'}] \xrightarrow{\mathrm{var}}_{E_m} M_V$. Since $\xrightarrow{\mathrm{var}}_{E_m}$ is injective, $M_V = x[a_1,\dots,a_l]$. This yields a contradiction, since $M_V \in \mathrm{Dom}(E_{m-1})$ but $x[a_1,\dots,a_l] \notin \mathrm{Dom}(E_{m-1})$ by Invariant 4. (The array cell $x[a_1,\dots,a_l]$ cannot be defined several times in a trace.)

  It is then easy to see that the invariant is satisfied.

- When the trace of $C'[Q_0]$ executes $\sigma_{\mathrm{i}} P_M$ for $M \in \mathcal{M}$, the corresponding trace of $C'[C[[\![L]\!]]]$ executes

$$\sigma_{\mathrm{i}}(\overline{d_{M,1}}\langle\rangle; d_{M,1}(); \dots \overline{d_{M,l}}\langle\rangle; d_{M,l}();$$
$$\overline{d_M}\langle \sigma_M x_{1,M},\dots,\sigma_M x_{m,M}\rangle; d_M(y : \mathit{bitstring}); C_M[y])$$

  where $\sigma_{\mathrm{i}} = \{\widetilde{a}/\widetilde{i}\}$, $\widetilde{i}$ is the sequence of current replication indices at P_M, and $BL(M) = (j_0,\dots,j_l)$.

  For $k \le l$, let a_k be such that

$$E_m, \mathrm{addstart}_{k,M}(\mathrm{num}_{k,M}(\sigma_{\mathrm{i}}(\mathrm{im\,index}_k(M))))) \Downarrow a_k$$

  and let $\widetilde{b_k}$ be such that $E_m, \sigma_{\mathrm{i}}(\mathrm{im\,index}_k(M)) \Downarrow \widetilde{b_k}$.

  Let m'_k be the step of the trace of $C'[C[[\![L]\!]]]$ after executing $\overline{\sigma_{\mathrm{i}} d_{M,k}}\langle\rangle; \sigma_{\mathrm{i}} d_{M,k}()$, where $d_{M,k} = d_{j_0,\dots,j_{k-1}}[\mathrm{convindex}(k, M)]$. We show by induction on k that for all k', $y_{(j_0,\dots,j_{k-1}),k'}[a_1,\dots,a_k] \in \mathrm{Dom}(E'_{m'_k})$ and that the invariant is satisfied at step m'_k except that

$\sigma_{\mathrm{i}}(\overline{d_{M,1}}\langle\rangle; d_{M,1}(); \ldots; \overline{d_{M,k}}\langle\rangle; d_{M,k}())$ has been removed from $P'_{m'_k}$. Let $z_{kk'} = \mathrm{varImL}(y_{((j_0,\ldots,j_{k-1}),k'}, M)$. We have $y_{(j_0,\ldots,j_{k-1}),k'}[a_1,\ldots,a_k] \xrightarrow{\mathrm{var}}_{E_m} z_{kk'}[\widetilde{b_k}]$. Moreover, $z_{kk'}[\widetilde{b_k}] \in \mathrm{Dom}(E_m)$ since $z_{kk'}[\sigma_{\mathrm{i}}(\mathrm{im\ index}_k(M))]$ occurs in $\sigma_{\mathrm{i}} M$, and $\sigma_{\mathrm{i}} M$ is successfully evaluated in the trace of $C'[Q_0]$. We distinguish two cases:

- 1) $y_{((j_0,\ldots,j_{k-1}),k'}[a_1,\ldots,a_k] \in \mathrm{Dom}(E'_{m'_{k-1}})$.
  By the invariant at step m'_{k-1}, $Q^2_{m'_{k-1}}$ contains $d_{j_0,\ldots,j_{k-1}}[a_1,\ldots,a_k](); \overline{d_{j_0,\ldots,j_{k-1}}[a_1,\ldots,a_k]}\langle\rangle$.
  So we can execute $\overline{\sigma_{\mathrm{i}} d_{M,k}}\langle\rangle; \sigma_{\mathrm{i}} d_{M,k}()$ by two (Output) steps, without changing the environment, so $y_{(j_0,\ldots,j_{k-1}),k'}[a_1,\ldots,a_k] \in \mathrm{Dom}(E'_{m'_k})$ and the invariant is satisfied at step m'_k except that $\sigma_{\mathrm{i}}(\overline{d_{M,1}}\langle\rangle; d_{M,1}(); \ldots \overline{d_{M,k}}\langle\rangle; d_{M,k}())$ is removed from $P'_{m'_k}$.
- 2) $y_{(j_0,\ldots,j_{k-1}),k'}[a_1,\ldots,a_k] \notin \mathrm{Dom}(E'_{m'_{k-1}})$. By induction hypothesis, $y_{(j_0,\ldots,j_{k-2}),k'}[a_1,\ldots,a_{k-1}] \in \mathrm{Dom}(E'_{m'_{k-1}})$. By the invariant at step m'_{k-1},
  $$\mathrm{redRepl}(a_k, \mathrm{relay}(L^{a_1,\ldots,a_{k-1}}_{j_0,\ldots,j_{k-1}})) \in \mathcal{Q}^2_{m'_{k-1}} \text{ and}$$
  $$\mathrm{redRepl}(a_k, [\![L^{a_1,\ldots,a_{k-1}}_{j_0,\ldots,j_{k-1}}]\!]) \in \mathcal{Q}^3_{m'_{k-1}}.$$
  By (Output) twice, we send an empty message on $d_{j_0,\ldots,j_{k-1}}[a_1,\ldots,a_k]$ and on $c_{j_0,\ldots,j_{k-1}}[a_1,\ldots,a_k]$. By (New), we define $y_{(j_0,\ldots,j_{k-1}),k'}[a_1,\ldots,a_k]$ for each k'. We choose $E_m(z_{kk'}[\widetilde{b_k}])$ as value of $y_{(j_0,\ldots,j_{k-1}),k'}[a_1,\ldots,a_k]$ (with probability $\frac{1}{|I_\eta(T)|}$ where T is the type of $y_{(j_0,\ldots,j_{k-1}),k'}$). Finally, by (Output) twice, we send an empty message on $c_{j_0,\ldots,j_{k-1}}[a_1,\ldots,a_k]$ and on $d_{j_0,\ldots,j_{k-1}}[a_1,\ldots,a_k]$. Then the invariant is satisfied at step m'_k except that $\sigma_{\mathrm{i}}(\overline{d_{M,1}}\langle\rangle; d_{M,1}(); \ldots \overline{d_{M,k}}\langle\rangle; d_{M,k}())$ is removed from $P'_{m'_k}$. (Note that the probability of the trace of $C'[C[[\![L]\!]]]$ is divided by $\prod_{k'} |I_\eta(T_{(j_0,\ldots,j_{k-1}),k'})|$ where $T_{(j_0,\ldots,j_{k-1}),k'}$ is the type of $y_{(j_0,\ldots,j_{k-1}),k'}[a_1, \ldots, a_k]$. This is what is required by the invariant since $y_{(j_0,\ldots,j_{k-1}),k'}[a_1,\ldots,a_k]$ is defined at step m'_k but was not at step m'_{k-1}.)

So $y_{(j_0,\ldots,j_{k-1}),k'}[a_1,\ldots,a_k] \in \mathrm{Dom}(E'_{m'_l})$ for all $k \le l$ and k', and the invariant is satisfied at step m'_l except that $\sigma_{\mathrm{i}}(\overline{d_{M,1}}\langle\rangle; d_{M,1}(); \ldots \overline{d_{M,l}}\langle\rangle; d_{M,l}())$ is removed from $P'_{m'_l}$. Let a be such that $E_m, \sigma_{\mathrm{i}} M \Downarrow a$. Let m'' be the step of the trace of $C'[C[[\![L]\!]]]$ after executing $\sigma_{\mathrm{i}}(\overline{d_M}\langle\sigma_M x_{1,M}, \ldots, \sigma_M x_{l',M}\rangle; d_M(y : \mathit{bitstring}))$ with $l' = \mathrm{nInput}_M$. By the invariant, we have two cases:

- 1) $\mathrm{relay}(L^{a_1,\ldots,a_l}_{j_0,\ldots,j_l}) \in \mathcal{Q}^2_{m'_l}$ and $[\![L^{a_1,\ldots,a_l}_{j_0,\ldots,j_l}]\!] \in \mathcal{Q}^3_{m'_l}$.
  The process $\sigma_{\mathrm{i}}\overline{d_M}\langle\sigma_M x_{1,M}, \ldots, \sigma_M x_{l',M}\rangle$ sends the value of $\sigma_{\mathrm{i}}\sigma_M x_{k',M}$ for $k' \le l'$ on channel $d_{j_0,\ldots,j_l}[a_1,\ldots,a_l]$. By (Output), this message is received by $\mathrm{relay}(L^{a_1,\ldots,a_l}_{j_0,\ldots,j_l})$, which forwards it by (Output) again to $[\![L^{a_1,\ldots,a_l}_{j_0,\ldots,j_l}]\!]$ on channel $c_{j_0,\ldots,j_l}[a_1,\ldots,a_l]$. On reception of this message by $[\![L^{a_1,\ldots,a_l}_{j_0,\ldots,j_l}]\!]$, $E'_{m''}(x_{(j_0,\ldots,j_l),k'}[a_1,\ldots,a_l])$ is set to the received value, so $E_m, \sigma_{\mathrm{i}}\sigma_M x_{k',M} \Downarrow E'_{m''}(x_{(j_0,\ldots,j_l),k'}[a_1,\ldots,a_l])$ for each $k' \le l'$. For all $k \le l$ and k', since $y_{(j_0,\ldots,j_{k-1}),k'}[a_1,\ldots, a_k] \xrightarrow{\mathrm{var}}_{E_m} z_{kk'}[\widetilde{b_k}]$, by the invariant we have $E'_{m'_l}(y_{(j_0,\ldots,j_{k-1}),k'}[a_1,\ldots,a_k]) = E_m(z_{kk'}[\widetilde{b_k}])$, so $E'_{m''}(y_{(j_0,\ldots,j_{k-1}),k'}[a_1,\ldots,a_k]) = E_m(z_{kk'}[\widetilde{b_k}])$.
  Moreover, $\sigma_M y_{kk',M} = z_{kk'}[\mathrm{im\ index}_k(M)]$, so
  $$E_m, \sigma_{\mathrm{i}}\sigma_M y_{kk',M} \Downarrow E'_{m''}(y_{(j_0,\ldots,j_{k-1}),k'}[a_1,\ldots,a_k])$$
  Therefore, for all variables x of N_M defined under k replications, $E_m, \sigma_{\mathrm{i}}\sigma_M x \Downarrow E'_{m''}(x[a_1,\ldots,a_k])$. Since $M = \sigma_M N_M$, we have $E_m, \sigma_{\mathrm{i}}\sigma_M N_M \Downarrow a$, so $E'_{m''}, N_M\{a_1/i_1,\ldots,a_l/i_l\} \Downarrow a$, where $i_1,\ldots,i_l$ are the replication indices of L above $L_{j_0,\ldots,j_l}$. Hence $[\![L^{a_1,\ldots,a_l}_{j_0,\ldots,j_l}]\!]$ sends back a on channel $c_{j_0,\ldots,j_l}[a_1,\ldots,a_l]$ by (Output), which is forwarded on channel $d_{j_0,\ldots,j_l}[a_1,\ldots,a_l]$ by $\mathrm{relay}(L^{a_1,\ldots,a_l}_{j_0,\ldots,j_l})$ by (Output) again, so a is stored in $y[\widetilde{a}]$ by Q''. Thus $E'_{m''}(y[\widetilde{a}]) = a$.
  In order to show that the invariant still holds after this step, we remark that, after these outputs, the relay process makes available the following process
  $$d_{j_0,\ldots,j_l}[a_1,\ldots,a_l](x_{(j_0,\ldots,j_l),1} : T_{(j_0,\ldots,j_l),1}, \ldots, x_{(j_0,\ldots,j_l),l'} : T_{(j_0,\ldots,j_l),l'}); \overline{d_{j_0,\ldots,j_l}[a_1,\ldots,a_l]}\langle a\rangle$$
  and we have $E_m, \mathrm{convindex}(l, M)\{\widetilde{a}/\widetilde{i}\} \Downarrow a_1,\ldots, a_l$, $E_m, M\{\widetilde{a}/\widetilde{i}\} \Downarrow a$, and $BL(M) = (j_0,\ldots,j_l)$.
- 2) $d_{j_0,\ldots,j_l}[a_1,\ldots,a_l](x_{(j_0,\ldots,j_l),1} : T_{(j_0,\ldots,j_l),1}, \ldots, x_{(j_0,\ldots,j_l),l'} : T_{(j_0,\ldots,j_l),l'}); \overline{d_{j_0,\ldots,j_l}[a_1,\ldots,a_l]}\langle r\rangle \in \mathcal{Q}^2_{m'_l}$ and there exist $M' \in \mathcal{M}$ and $\widetilde{a'}$ such that $E_m, \mathrm{convindex}(l, M')\{\widetilde{a'}/\widetilde{i'}\} \Downarrow a_1,\ldots,a_l$, $E_m, M'\{\widetilde{a'}/\widetilde{i'}\} \Downarrow r$, and $BL(M') = (j_0,\ldots,j_l)$, where $\widetilde{i'}$ is the sequence of current replication indices at M'.
  We have $E_m, \mathrm{convindex}(l, M)\{\widetilde{a}/\widetilde{i}\} \Downarrow a_1,\ldots,a_l$ by definition of $a_1,\ldots,a_l$. So
  $$\mathrm{convindex}(l, M')\{\widetilde{a'}/\widetilde{i'}\} =_{E_m} \mathrm{convindex}(l, M)\{\widetilde{a}/\widetilde{i}\}$$
  so, as shown in the proof that $\xrightarrow{\mathrm{var}}_E$ is a function, $\mathrm{index}_l(M')\{\widetilde{a'}/\widetilde{i'}\} =_{E_m} \mathrm{index}_l(M)\{\widetilde{a}/\widetilde{i}\} =_{E_m} \widetilde{b_l}$ and M' and M share the first l sequences of random variables, that is, all sequences of random variables, or $m_l = 0$ and $M = M'$. Moreover, $BL(M) = BL(M') = (j_0,\ldots,j_l)$, so $N_M = N_{M'}$.
  If $m_l = 0$ and $M = M'$, $\widetilde{a'} = \widetilde{a}$, so $E_m, \sigma_{\mathrm{i}} M \Downarrow r$, so $r = a$.
  Otherwise, by Hypothesis H′4.3, there exists a term M_0 such that $M = (\mathrm{index}_l(M))M_0$, $M' = (\mathrm{index}_l(M'))M_0$, and M_0 does not contain the current replication indices at M or M'. Then $a =_{E_m} M\{\widetilde{a}/\widetilde{i}\} =_{E_m} M_0\{\widetilde{b_l}/\widetilde{i''}\} =_{E_m} M'\{\widetilde{a'}/\widetilde{i'}\} =_{E_m} r$ where $\widetilde{i''}$ is the sequence of current replication indices at definition of $z_{lk',M}$ for any k'.

Therefore, in all cases, we obtain $E'_{m''}(y[\widetilde{a}]) = a$, so $\sigma_{\mathrm{i}} C_M[y]$ in the trace of $C'[C[\llbracket L \rrbracket]]$ executes in the same way as $\sigma_{\mathrm{i}} C_M[M]$ in the trace of $C'[Q_0]$, which yields the desired invariant.

- The other cases are easy: both sides reduce in the same way.

Conversely, we show that all traces of $C'[C[\llbracket L \rrbracket]]$ correspond to a trace of $C'[Q_0]$ with the same relation as above. The proof follows a technique similar to the previous proof.

So $\prod_{z,a'_1,\dots,a'_{j'}} |I_\eta(T)|$ traces of $C'[Q_0]$, each of probability p_m, correspond to one trace of $C'[C[\llbracket L \rrbracket]]$ with probability $p'_{m'} = p_m \times \prod_{z,a'_1,\dots,a'_{j'}} |I_\eta(T)|$. Moreover, for all channels c and bitstrings a, $E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m$ executes $\overline{c}\langle a \rangle$ immediately if and only if $E'_{m'}, P'_{m'}, \mathcal{Q}'_{m'}, \mathcal{C}'_{m'}$ executes $\overline{c}\langle a \rangle$ immediately. So $\Pr[C'[Q_0] \rightsquigarrow_\eta \overline{c}\langle a \rangle] = \Pr[C'[C[\llbracket L \rrbracket]] \rightsquigarrow_\eta \overline{c}\langle a \rangle]$. Hence $Q_0 \approx_0^V C[\llbracket L \rrbracket]$. □

Lemma 12 $Q'_0 \approx_0^V C[\llbracket R \rrbracket]$

Proof sketch The proof uses the same technique as the proof of Lemma 11. The main addition is that, in contrast to L, R may contain functional processes that are more complex than just terms. In order to handle them, we need to define a relation between variables of Q'_0 and variables of R defined by let or new in functional processes: when y is such a variable, $y[a_1,\dots,a_l] \xrightarrow{\mathrm{var}}_E \mathrm{varImR}(y,M)[\widetilde{a'}]$ where for all $k \le l$, $E, \mathrm{addstart}_{k,M}(\mathrm{num}_{k,M}(\mathrm{im\ index}_k(M)\{\widetilde{a'}/\widetilde{i}\})) \Downarrow a_k$ and $\widetilde{i}$ is the sequence of current replication indices at M. The relation $\xrightarrow{\mathrm{var}}_E$ is not a function for these variables, but we can show that when $y[a_1,\dots,a_l]$ is related to several variables, these variables hold the same value at runtime.

The most delicate case is that of find functional processes

$$FP = \mathsf{find}\ (\textstyle\bigoplus_{j=1}^m \widetilde{u_j} \le \widetilde{n_j}\ \mathsf{suchthat\ defined}(z_{j1}[\widetilde{u_{j1}}],\dots,$$
$$z_{jl_j}[\widetilde{u_{jl_j}}]) \wedge M_j\ \mathsf{then}\ FP_j)\ \mathsf{else}\ FP'$$

where for each k, $\widetilde{u_{jk}}$ is the concatenation of the prefix of the current replication indices of length l'_0 and of a non-empty prefix of $\widetilde{u_j}$. When executing such a find process, $\llbracket R \rrbracket$ tests the value of $z_{jk}[a_1,\dots,a_{l'_1}]$ for all indices of $a_1,\dots,a_{l'_1}$ such that $a_1,\dots,a_{l'_0}$ correspond to a prefix of the current replication indices. Correspondingly, $\mathrm{transf}_{\phi,C_M}(FP)$ tests the values of all variables that are related to $z_{jk}[a_1,\dots,a_{l'_1}]$ by $\xrightarrow{\mathrm{var}}$. □

Lemma 13 *Process Q'_0 satisfies Invariant 1.*

Proof Process Q'_0 satisfies Invariant 1 since all newly created definitions concern fresh variables; for variables of Q'_0 that correspond to variables defined by new or by an input in R, there is a single definition for each of them in Q'_0; for variables of Q'_0 that correspond to variables defined by let in R, there are several definitions only when there are several definitions of these variables in R, and since $\llbracket R \rrbracket$ satisfies Invariant 1, these definitions are in different branches of find (or if) in R, so also in Q'_0. □

Lemma 14 *Process Q'_0 satisfies Invariant 2.*

Proof The only variable accesses created in Q'_0 come from $\mathrm{transf}_{\phi_0,C_M}(FP)$. We show by induction on FP that the only variable accesses created by $\mathrm{transf}_{\phi,C_M}(FP)$ and not guarded by a corresponding find are in $\mathrm{im}\ \phi$. (We do not consider variable accesses in C_M, which already existed in Q_0.) So the only variable accesses created by $\mathrm{transf}_{\phi_0,C_M}(FP_M)$ and not guarded by a corresponding find are in $\mathrm{im}\ \phi_0$. Moreover, variable accesses in $\mathrm{im}\ \phi_0$ are of three kinds:

1. $\mathrm{varImR}(x_{j,M},M)[i'_1,\dots,i'_{l'}]$ which are defined in P'_M, just above $\mathrm{transf}_{\phi_0,C_M}(FP_M)$.
2. $\mathrm{varImR}(y'_{jk,M},M)[\mathrm{im\ index}_j(M)]$ where
   (a) either $\mathrm{nNew}_{j,M} > 0$ and $z_{j1,M}[\mathrm{im\ index}_j(M)]$ is guaranteed to be defined, since it occurs at this point in the initial process Q_0 which satisfies Invariant 2. By the addition of defined conditions in find and the fact that $z'_{jk,M} = \mathrm{varImR}(y'_{jk,M},M)$ is defined in Q'_0 where $z_{j1,M}$ was defined in Q_0, this implies that $\mathrm{varImR}(y'_{jk,M},M)[\mathrm{im\ index}_j(M)]$ is also defined.
   (b) or $\mathrm{nNew}_{j,M} = 0$, then $\mathrm{im\ index}_j(M)$ is the sequence of current replication indices at M, and $\mathrm{varImR}(y'_{jk,M},M)[\mathrm{im\ index}_j(M)]$ is defined just above P'_M.
3. $\mathrm{varImR}(z,M)[i'_1,\dots,i'_{l'}]$ where z is defined by let in FP_M. Since $\llbracket R \rrbracket$ satisfies Invariant 2, accesses to $z[i_1,\dots,i_l]$ in FP_M occur under the definition of $z[i_1,\dots,i_l]$ in FP_M, so accesses to $\mathrm{varImR}(z,M)[i'_1,\dots,i'_{l'}] = \phi_0(z[i_1,\dots,i_l])$ also occur under their definition in $\mathrm{transf}_{\phi_0,C_M}(FP_M)$.

Therefore, Q'_0 satisfies Invariant 2. □

Lemma 15 *Process Q'_0 satisfies Invariant 3.*

Proof The only newly added variable definitions are let $\mathrm{varImR}(x_{j,M},M) : T_{j,M} = \sigma_M x_{j,M}$ and new $z'_{jk,M} : T'_{jk,M}$. Each variable $\mathrm{varImR}(x_{j,M},M)$ has at most one definition in Q'_0. For variables $z'_{jk,M}$, when several of these definitions are added for the same variable $z'_{jk,M}$, they are added in place of the definition(s) of $z_{j1,M}$, so by Hypothesis H′3.1.1, they occur under the same replications, so they all have the same type. Therefore, the type environment for Q'_0 is well-defined.

Assume that $M \in \mathcal{M}$ and $P_M = C_M[M]$ is the smallest process containing M. Let $\mathcal{E}_L$ be the type environment at $P_M = C_M[M]$ in Q_0; let $\mathcal{E}_R$ be the type environment at P'_M in Q'_0; let $\mathcal{E}'_L$ be the type environment at N_M in L; let $\mathcal{E}'_R$ be the type environment at FP_M in R. We know that $\mathcal{E}_L \vdash P_M$, and show that $\mathcal{E}_R \vdash P'_M$. It is then easy to see that Q'_0 is well-typed knowing that Q_0 is well-typed. We note that $\mathcal{E}_R$ is an extension of $\mathcal{E}_L$ with types for variables $\mathrm{varImR}(y'_{jk,M'},M')$, $\mathrm{varImR}(x_{j,M'},M')$, and $\mathrm{varImR}(z,M')$ when z is defined by let in $FP_{M'}$, for each $M' \in \mathcal{M}$. By Hypothesis H′3.2, $\mathcal{E}_L \vdash \sigma_M x_{j,M} : T_{j,M}$, so $\mathcal{E}_R \vdash \sigma_M x_{j,M} : T_{j,M}$, since $\mathcal{E}_R$ is an extension of $\mathcal{E}_L$. Then, in order to show $\mathcal{E}_R \vdash P'_M$, it is enough to show $\mathcal{E}_R \vdash \mathrm{transf}_{\phi_0,C_M}(FP_M)$.

We say that ϕ is well-typed when $z[\widetilde{M}] \in \mathrm{Dom}(\phi)$ and $\mathcal{E}'_R \vdash z[\widetilde{M}] : T'$ implies $\mathcal{E}_R \vdash \phi(z[\widetilde{M}]) : T'$.

First, it is easy to show by induction on M' that for all well-typed ϕ, for all M' such that $\mathcal{E}'_R \vdash M' : T$, we have $\mathcal{E}_R \vdash \phi(M') : T$.

Next, we show that for all well-typed ϕ, if $\mathcal{E}'_R \vdash [\![FP']\!]^{\widetilde{j}}_{\widetilde{i}}$ and the type of the result of FP' is the type of N_M, then $\mathcal{E}_R \vdash \mathrm{transf}_{\phi,C_M}(FP')$, by induction on FP'.

- If $FP' = M'$, we have to show that $\mathcal{E}_R \vdash C_M[\phi(M')]$. Let T such that $\mathcal{E}_L \vdash M : T$.

  We have $M = \sigma_M N_M$, so if N_M contains a function symbol, $\mathcal{E}'_L \vdash N_M : T$. If $N_M = x_{j,M}$, $M = \sigma_M x_{j,M}$ is of type $T_{j,M}$ by Hypothesis H′3.2, so $T = T_{j,M}$, hence we also have $\mathcal{E}'_L \vdash N_M : T$. If $N_M = y_{jk,M}$, $M = \sigma_M y_{jk,M} = z_{jk,M}[\mathrm{im\ index}_j(M)]$ is of type $T_{jk,M}$ by Hypothesis H′3.1.1, so $T = T_{jk,M}$ and we also have $\mathcal{E}'_L \vdash N_M : T$.

  By hypothesis, we have then $\mathcal{E}'_R \vdash M' : T$, so $\mathcal{E}_R \vdash \phi(M') : T$. Since $\mathcal{E}_L \vdash C_M[M]$ with $\mathcal{E}_L \vdash M : T$, by a substitution lemma, we conclude that $\mathcal{E}_R \vdash C_M[\phi(M')]$.

- The inductive cases follow easily using $\mathcal{E}'_R \vdash [\![FP']\!]^{\widetilde{j}}_{\widetilde{i}}$ and the property proved above to type terms.

  In the case of a find branch with non-empty defined conditions, we extend ϕ into ϕ' as follows. Let $\widetilde{i'}$ be the sequence of current replication indices at M' and $\widetilde{u'}$ be a sequence formed with a fresh variable for each variable in $\widetilde{i'}$.

  - If $z_k = y'_{jk',M'}$ for some k', then

    $$\phi'(z_k[M_{k1}, \dots, M_{kl'_k}]) = \mathrm{varImR}(z_k, M')[\mathrm{im\ index}_j(M')\{\widetilde{u'}/\widetilde{i'}\}].$$

    Since $\mathrm{varImR}(z_k, M')$ is defined where $z_{j1,M'}$ is defined, the indices of $\mathrm{varImR}(z_k, M')$ are the indices of $z_{j1,M'}$, so $\mathrm{im\ index}_j(M')$ is of the suitable type. Moreover, $\widetilde{u'}$ and $\widetilde{i'}$ have the same types, so by a substitution lemma, $\mathrm{im\ index}_j(M')\{\widetilde{u'}/\widetilde{i'}\}$ is of the suitable type. Moreover z_k in R and $\mathrm{varImR}(z_k, M')$ in Q'_0 are both declared of type $T'_{jk',M'}$, so $\mathcal{E}'_R \vdash z_k[M_{k1}, \dots, M_{kl'_k}] : T'_{jk',M'}$ and $\mathcal{E}_R \vdash \mathrm{varImR}(z_k, M')[\mathrm{im\ index}_j(M')\{\widetilde{u'}/\widetilde{i'}\}] : T'_{jk',M'}$.
  - If z_k is defined by let or by a function input, $\phi'(z_k[M_{k1}, \dots, M_{kl'_k}]) = \mathrm{varImR}(z_k, M')[\widetilde{u'}]$. $\mathrm{varImR}(z_k, M')$ is declared under the same replications as M', so $\widetilde{u'}$ is of the suitable type. The variables z_k in R and $\mathrm{varImR}(z_k, M')$ in Q'_0 are declared of the same type, so if $\mathcal{E}'_R \vdash z_k[M_{k1}, \dots, M_{kl'_k}] : T'$ then $\mathcal{E}_R \vdash \mathrm{varImR}(z_k, M')[\widetilde{u'}] : T'$.

  So ϕ' is well-typed.

  Moreover, we show that $\mathcal{E}_R \vdash \mathrm{im\ index}_{j_1}(M')\{\widetilde{u'}/\widetilde{i'}\} = \mathrm{im\ index}_{j_1}(M) : \mathit{bool}$. We have $z_{j_1k,M} = z_{j_1k,M'}$ since M and M' share the j_1 first sequences of random variables, so $\mathrm{im\ index}_{j_1}(M')$ and $\mathrm{im\ index}_{j_1}(M)$ are of the same type, since they are both used as indices of $z_{j_1k,M}$. Since $\widetilde{u'}$ and $\widetilde{i'}$ are of the same type, by a substitution lemma, $\mathrm{im\ index}_{j_1}(M')\{\widetilde{u'}/\widetilde{i'}\}$ and $\mathrm{im\ index}_{j_1}(M)$ are of the same type, which yields the desired result.

It is easy to see that ϕ_0 is well-typed. Moreover $\mathcal{E}'_R \vdash [\![FP_M]\!]^{\widetilde{j}}_{\widetilde{i}}$ and the type of the result of FP_M is the type of N_M by Hypothesis H0, so $\mathcal{E}_R \vdash \mathrm{transf}_{\phi_0,C_M}(FP_M)$. □

Proof of Proposition 3 Invariants 1, 2, and 3 have been proved in Lemmas 13, 14, and 15 respectively. Finally, we show that $Q_0 \approx^V Q'_0$. After renaming variables so that V and C do not contain variables of L and R, by Lemmas 1, 11, and 12, $Q_0 \approx^V_0 C[[\![L]\!]] \approx^V C[[\![R]\!]] \approx^V_0 Q'_0$, so by transitivity $Q_0 \approx^V Q'_0$. □

E.5 Proofs for Section 4

Proof of Proposition 4 Let C be an acceptable context for $Q \mid Q_x$, $Q \mid Q'_x$, $\emptyset$. We relate the traces of $C[Q \mid Q_x]$ and $C[Q \mid Q'_x]$ as follows:

- If a trace of $C[Q \mid Q_x]$ never executes the subprocess $\overline{c}\langle x[u_1, \dots, u_m]\rangle$ of Q_x, then we obtain a trace of $C[Q \mid Q'_x]$ with the same probability, by just replacing Q_x with Q'_x and subprocesses of Q_x with the corresponding subprocess of Q'_x.

- Otherwise, the considered trace of $C[Q \mid Q_x]$ executes the subprocess $\overline{c}\langle x[u_1, \dots, u_m]\rangle$ of Q_x exactly once, with $E(u_1) = a_1, \dots, E(u_m) = a_m$, and $E(x[a_1, \dots, a_m]) = a$, where E is the environment when $\overline{c}\langle x[u_1, \dots, u_m]\rangle$ is executed. By hypothesis, the definition of $x[a_1, \dots, a_m]$ in this trace is either a restriction new $x[a_1, \dots, a_m] : T$, or an assignment let $x[a_1, \dots, a_m] : T = z[M_1, \dots, M_l]$ with $E, M_k \Downarrow a'_k$ for all $k \leq l$, and the definition of $z[a'_1, \dots, a'_l]$ in this trace is new $z[a'_1, \dots, a'_l] : T$.

  We build $|I_\eta(T)|$ traces of $C[Q \mid Q'_x]$ from this trace, by choosing any value of $I_\eta(T)$ for the restriction new $x[a_1, \dots, a_m] : T$ or new $z[a'_1, \dots, a'_l] : T$ defined above, and the value a for the restriction new $y : T$ of Q'_x. By definition of S, these traces are the same as the trace of $C[Q \mid Q_x]$ except perhaps for values of variables in S, and for the process Q'_x instead of Q_x. The probability of each of these traces is $1/|I_\eta(T)|$ times the probability of the considered trace of $C[Q \mid Q_x]$, since these traces choose one more random number in $I_\eta(T)$ than the trace of $C[Q \mid Q_x]$.

Moreover, all traces of $C[Q \mid Q'_x]$ are obtained by the previous construction. (To show that, we rebuild a trace of $C[Q \mid Q_x]$ from the trace of $C[Q \mid Q'_x]$ by the reverse construction of the one detailed above.)

For each configuration $E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m$ of the trace of $C[Q \mid Q_x]$, and corresponding configuration $E'_{m'}, P'_{m'}, \mathcal{Q}'_{m'}, \mathcal{C}'_{m'}$ of the trace of $C[Q \mid Q'_x]$, for all channels c and bitstrings a, $E_m, P_m, \mathcal{Q}_m, \mathcal{C}_m$ executes $\overline{c}\langle a\rangle$ immediately if and only if $E'_{m'}, P'_{m'}, \mathcal{Q}'_{m'}, \mathcal{C}'_{m'}$ executes $\overline{c}\langle a\rangle$ immediately.

Therefore, $\Pr[C[Q \mid Q_x] \rightsquigarrow_\eta \overline{c}\langle a\rangle] = \Pr[C[Q \mid Q'_x] \rightsquigarrow_\eta \overline{c}\langle a\rangle]$, so $Q \mid Q_x \approx_0 Q \mid Q'_x$. □

Proof sketch of Proposition 5 Let C be an acceptable context for $Q \mid Q_x, Q \mid Q'_x, \emptyset$.

We first exclude traces $\mathcal{T}$ such that $\mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a}]) = \mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a'}])$ and $\widetilde{a} \neq \widetilde{a'}$. These traces have negligible probability by hypothesis, since $C[\_ \mid Q_x]$ is an acceptable context for Q, 0, $\{x\}$. So this removal does not change the result.

For the remaining traces, when $\widetilde{a} \neq \widetilde{a'}$, $\mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a}]) \neq \mathrm{defRestr}_{\mathcal{T}}(x[\widetilde{a'}])$, so the definitions of $x[\widetilde{a}]$ and $x[\widetilde{a'}]$ do not come from a single execution of the same restriction. (So $x[\widetilde{a}]$ and $x[\widetilde{a'}]$ are independent random numbers.) Then we can apply a proof similar to that of Proposition 4, except that we replace each tested value of $x[\widetilde{a'}]$ with independent random numbers instead of single one. $\square$

Proof of Lemma 2 Let us prove the result for one-session secrecy. (The proof is essentially the same for secrecy.) The contexts $[\,] \mid Q_x$ and $[\,] \mid Q'_x$ are acceptable contexts for Q, Q', $\{x\}$ (after renaming $u_1, \dots, u_m, y$ so that they do not occur in Q and Q'). We have $Q \approx^{\{x\}} Q'$. So, by Lemma 1, $Q \mid Q_x \approx Q' \mid Q_x$ and $Q \mid Q'_x \approx Q' \mid Q'_x$. Since Q preserves the one-session secrecy of x, $Q \mid Q_x \approx Q \mid Q'_x$. So, by transitivity of $\approx$, $Q' \mid Q_x \approx Q' \mid Q'_x$. Therefore, Q' preserves the one-session secrecy of x. $\square$