CryptoVerif: a Computationally-Sound Security Protocol Verifier (Initial Version with Communications on Channels)

Bruno Blanchet

CryptoVerif: a Computationally-Sound Security Protocol Verifier (Initial Version with Communications on Channels)

Bruno Blanchet

RESEARCH REPORT
N° RR-9525
October 2023
Project-Team Prosecco

ISSN 0249-6399 ISRN INRIA/RR--RR-9525--FR+ENG

CryptoVerif: a Computationally-Sound Security Protocol Verifier
(Initial Version with Communications on Channels)

Bruno Blanchet[*]

Project-Team Prosecco

Research Report n° RR-9525 — October 2023 — 166 pages

Abstract: This document presents the security protocol verifier CryptoVerif. CryptoVerif does not rely on the symbolic, Dolev-Yao model, but on the computational model. It can verify secrecy, correspondence (which include authentication), and indistinguishability properties. It produces proofs presented as sequences of games, like those manually written by cryptographers; these games are formalized in a probabilistic process calculus. CryptoVerif provides a generic method for specifying security properties of the cryptographic primitives. It produces proofs valid for any number of sessions of the protocol, and provides an upper bound on the probability of success of an attack against the protocol as a function of the probability of breaking each primitive and of the number of sessions. It can work automatically, or the user can guide it with manual proof indications.

Key-words: security protocols, verification, computational model

[*] Inria

CryptoVerif: un vérificateur de protocoles cryptographiques sûr dans le modèle calculatoire
(Version initiale avec communication sur des canaux)

Résumé : Ce document présente le vérificateur de protocoles cryptographiques CryptoVerif. CryptoVerif ne s'appuie pas sur le modèle symbolique de Dolev-Yao, mais sur le modèle calculatoire. Il peut vérifier le secret, les correspondances (qui comprennent l'authentification) et les propriétés d'indistinguabilité. Il produit des preuves par suites de jeux, comme celles écrites manuellement par les cryptographes ; ces jeux sont formalisés dans un calcul de processus probabiliste. CryptoVerif fournit une méthode générique pour spécifier les propriétés de sécurité des primitives cryptographiques. Il produit des preuves valables pour un nombre quelconque de sessions du protocole, et fournit une borne supérieure sur la probabilité de succès d'une attaque contre le protocole en fonction de la probabilité de casser chaque primitive et du nombre de sessions. Il peut fonctionner automatiquement, ou l'utilisateur peut guider la preuve manuellement.

Mots-clés : protocoles cryptographiques, vérification, modèle calculatoire

Contents

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 symbolic, 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 only these blackboxes. This abstract model makes it easier to build automatic verification tools, but the security proofs are in general not sound with respect to the computational model.

In contrast to most previous protocol verifiers, CryptoVerif works directly in the computational model, without considering the Dolev-Yao model. It produces proofs valid for any number of sessions of the protocol, in the presence of an active adversary. These proofs are presented as sequences of games, as used by cryptographers [20, 61, 62]: the initial game represents the protocol to prove; the goal is to bound the probability of breaking a certain security property in this game; intermediate games are obtained each from the previous one by transformations such that the difference of probability between consecutive games can easily be bounded; the final game is such that the desired probability is obviously bounded from the form of the game. (In general, it is simply 0 in that game.) The desired probability can then be easily bounded in the initial game.

We represent games in a process calculus. This calculus is inspired by the pi-calculus and by the calculi of [54] and of [50]. In this calculus, messages are bitstrings, and cryptographic primitives are functions from bitstrings to bitstrings. The calculus has a probabilistic semantics. The main tool for specifying security properties is indistinguishability: Q is indistinguishable from Q' up to probability p, $Q \approx_p Q'$, when the adversary has probability at most p 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 standard security assumption on 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 (*i.e.*, function). 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.

CryptoVerif 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 security assumptions on cryptographic primitives in order to obtain a simpler game. As described in Section 5.2, these transformations can be specified in a generic way: we represent the security assumption of each cryptographic primitive by an observational equivalence $L \approx_p R$, where the processes L and R encode oracles: they input the arguments of the oracle and send its result back. Then, the prover can automatically transform a process Q that calls the oracles of L (more precisely, contains as subterms terms that perform the same computations as oracles of L) into a process Q' that calls the oracles of R instead. We have used this technique to specify several variants of shared-key and public-key encryption, signature, message authentication codes, hash functions, Diffie-Hellman key agreement, simply by giving the appropriate equivalence $L \approx_p R$ to the prover. Other game transformations are syntactic transformations, used in order to be able to apply an assumption on a cryptographic primitive, or to simplify the game obtained after applying such an assumption.

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, simple protocols can often be proved in a fully automatic way. For delicate cases, CryptoVerif has an interactive mode, in which the user can manually specify the transformations to apply. It is often sufficient to specify a few well-chosen case distinctions and transformations coming from the security assumptions 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 instance for proving some public-key protocols, in which several security assumptions on primitives can be applied, but only one leads to a proof of the protocol. Importantly, CryptoVerif 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 assumptions on the cryptographic primitives.

CryptoVerif has been implemented in OCaml (more than 60000 lines of code) and is available at `http://cryptoverif.inria.fr/`.

Related Work Various methods have been proposed for verifying security protocols in the computational model. Following the seminal paper by Abadi and Rogaway [1], many results show the soundness of the Dolev-Yao model with respect to the computational model, which makes it possible to use Dolev-Yao provers in order to prove protocols in the computational model (see, e.g., [5,33,37,38,46] and the survey [36]). 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). A tool [35] was developed based on [37] to obtain computational proofs using the formal verifier AVISPA, for protocols that rely on public-key encryption and signatures.

Several frameworks exist for formalizing proofs of protocols in the computational model. Backes, Pfitzmann, and Waidner [7,8] designed an abstract cryptographic library and showed its soundness with respect to computational primitives, under arbitrary active attacks. This framework has been used for a computationally-sound machine-checked proof of the Needham-Schroeder-Lowe protocol [65,66]. Canetti [30] introduced the notion of universal composability. With Herzog [32], 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 [22] for verifying protocols in this framework. Process calculi have been designed for representing cryptographic games, such as the probabilistic polynomial-time calculus of [54] and the cryptographic lambda-calculus of [57]. Logics have also been designed for proving security protocols in the computational model, such as the computational variant of PCL (Protocol Composition Logic) [42,43] and CIL (Computational Indistinguishability Logic) [11]. Canetti *et al.* [31] use the framework of time-bounded task-PIOAs (Probabilistic Input/Output Automata) to prove security protocols in the computational model. This framework makes it possible to combine probabilistic and non-deterministic behaviors. These frameworks can be used to prove security properties of protocols in the computational sense, but except for [32] which relies on a Dolev-Yao prover, they have not been automated up to now, as far as we know.

Several techniques have been used for directly mechanizing proofs in the computational model. Type systems [41,50,52,64] provide computational security guarantees. For instance, [50] handles shared-key and public-key encryption, with an unbounded number of sessions, by relying on the Backes-Pfitzmann-Waidner library. A type inference algorithm is given in [6]. The recent tool OWL [44] also relies on a type system that provides computational security guarantees. It supports MACs, public-key signatures, authenticated symmetric and public key encryption, random oracles, and the gap Diffie-Hellman assumption [58]. It can prove secrecy and integrity properties. In another line of research, a specialized Hoare logic was designed for proving asymmetric encryption schemes in the random oracle model [39,40].

The tool CertiCrypt [12,13,15,17,18] enables the machine-checked construction and verification of cryptographic proofs by sequences of games [20,63]. It relies on the general-purpose proof assistant Coq, which is widely believed to be correct. Nowak *et al.* [3, 55, 56] follow a similar idea by providing Coq proofs for several cryptographic primitives. More recently, frameworks for cryptographic proofs in Coq, FCF [59], and in Isabelle, CryptHOL [16], have been designed and used for proving cryptographic schemes. EasyCrypt [14], the successor of CertiCrypt, no longer generates Coq proofs, but provides a higher automation level by relying on SMT solvers, which makes the tool easier to use. Even if it focuses more on cryptographic primitives and schemes than on protocols, it has been used for proving some protocols such as one-round key exchange [10], e-voting [34], AWS key management [4], and distance bounding [27]. These frameworks and tools can perform more subtle reasoning than CryptoVerif, at the cost of more user effort: the user has to give all games and guide the proof that the games are indistinguishable. That becomes tedious for large protocols, which require many large games.

The tool Squirrel [9] relies on a computationally sound logic that allows to write interactive proofs of, e.g., stateful protocols. Still, it currently proves a security notion weaker than the standard one: the number of sessions of the protocol must be bounded independently of the security parameter (instead of being polynomial in the security parameter).

Independently, we have built the tool CryptoVerif [24] to help cryptographers, not only for the verification, but also by generating the proofs by sequences of games [20, 63], automatically or with little user interaction. In particular, CryptoVerif generates the games, possibly using the indications of which transformations to perform. This tool extends considerably early work by Laud [48,49] which was limited either to passive adversaries or to a single session of the protocol. More recently, Tšahhirov and Laud [51, 67] developed a tool similar to CryptoVerif but that represents games by dependency graphs. It handles public-key and shared-key encryption and proves secrecy properties; it does not provide bounds on the probability of success of an attack.

Outline The next section presents our process calculus for representing games, with its syntax, type system, formal semantics, as well as the definition of security properties. Section 3 collects information about games and reasons using it. Section 4 gives criteria for proving security properties of protocols. Section 5 describes the game transformations that we use for proving protocols. Section 6 explains how the prover chooses which transformation to apply at each point.

Notations We recall the following notations. We denote by $\{M_1/x_1, \dots, M_m/x_m\}$ the substitution that replaces x_j with M_j for each $j \in \{1, \dots, m\}$. The cardinal of a set or multiset S is denoted by $|S|$. Multisets S are represented by functions that map each element x of S to the number of occurrences of x in S, that is, when S is a multiset, $S(x)$ is the number of elements of S equal to x. We use $\uplus$ for multiset union, defined by $(S_1 \uplus S_2)(x) = S_1(x) + S_2(x)$. When

S and S' are multisets, $\max(S, S')$ is the multiset such that $\max(S, S')(x) = \max(S(x), S'(x))$. The notation $\{x_1 \mapsto a_1, \dots, x_m \mapsto a_m\}$ designates the function that maps x_j for a_j for each $j \in \{1, \dots, m\}$ and is undefined for other inputs. When f is a function, $f[x \mapsto a]$ is the function that maps x to a and all other elements as f. If S is a finite set, $x \overset{R}{\leftarrow} S$ 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. If D is a discrete probability distribution, we denote by $D(a)$ the probability that $X = a$, $\Pr[X = a]$, where X is a random variable with probability distribution D.

2 A Calculus for Cryptographic Games

2.1 Syntax and Informal Semantics

CryptoVerif represents games in the syntax of Figure 1. This calculus assumes a countable set of channel names, denoted by c. It uses parameters, denoted by n, which are integers that bound the number of executions of processes.

It also uses types, denoted by T, which are non-empty, countable sets of values. We assume that there exists an efficient injection from each type to the set of bitstrings, and that its inverse is also efficiently computable. A type is *fixed* when it is the set of all bitstrings of a certain length; a type is *bounded* when it is a finite set. Particular types are predefined: $\mathit{bool} = \{\mathrm{true}, \mathrm{false}\}$, where false is 0 and true is 1; *bitstring* is the set of all bitstrings; $\mathit{bitstring}_\perp = \mathit{bitstring} \cup \{\perp\}$ where $\perp$ is a special symbol; $[1, n]$ is the set of integers $\{1, \dots, n\}$, where n is a parameter. (We consider integers as bitstrings without leading zeroes.)

The calculus also uses function symbols f. Each function symbol comes with a type declaration $f : T_1 \times \ldots \times T_m \to T$, and represents an efficiently computable, deterministic function that maps each tuple in $T_1 \times \ldots \times T_m$ to an element of T. 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*), tuples $(M_1, \dots, M_m)$ (taking values of any types and returning values of type *bitstring*; tuples are assumed to provide unambiguous concatenation, with tags for the types of $M_1, \dots, M_m$ so that tuples of different types are always different); test $\mathrm{if\_fun}(M_1, M_2, M_3)$ (with a first argument of type *bool* and the last two arguments of the same type T; it returns a value of type T: M_2 when M_1 is true and M_3 when M_1 is false).

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 $x[M_1, \dots, M_m]$ returns the content of the cell of indices $M_1, \dots, M_m$ of the m-dimensional array variable x. We use x, y, z, u as variable names. The function application $f(M_1, \dots, M_m)$ returns the result of applying function f to $M_1, \dots, M_m$ Terms contain additional constructs which are very similar to those also included in output processes and explained below. These constructs conclude by evaluating a term, instead of executing a process. The construct $\mathsf{event\_abort}$ e executes event e (without argument) and aborts the game.

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 $i \in [1, n]$;

$M, N ::=$	terms
i	replication index
$x[M_1, \dots, M_m]$	variable access
$f(M_1, \dots, M_m)$	function application
$\mathsf{new}\ x[\widetilde{i}] : T; N$	random number
$\mathsf{let}\ p = M\ \mathsf{in}\ N\ \mathsf{else}\ N'$	assignment (pattern-matching)
$\mathsf{let}\ x[\widetilde{i}] : T = M\ \mathsf{in}\ N$	assignment
$\mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ N'$	conditional
$\mathsf{find}[\mathit{unique?}]\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}$ $\mathsf{suchthat}$ $\mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M'_j\ \mathsf{then}\ N_j)\ \mathsf{else}\ N'$	array lookup
$\mathsf{insert}\ \mathit{Tbl}(M_1, \dots, M_l); N$	insert in table
$\mathsf{get}[\mathit{unique?}]\ \mathit{Tbl}(p_1, \dots, p_l)\ \mathsf{suchthat}\ M\ \mathsf{in}\ N\ \mathsf{else}\ N'$	get from table
$\mathsf{event}\ e(M_1, \dots, M_l); N$	event
$\mathsf{event\_abort}\ e$	event e and abort
$p ::=$	pattern
$x[\widetilde{i}] : T$	variable
$f(p_1, \dots, p_m)$	function application
$=M$	comparison with a term
$Q ::=$	input process
0	nil
$Q \mid Q'$	parallel composition
$!^{i \leq n} Q$	replication n times
$\mathsf{newChannel}\ c; Q$	channel restriction
$c[M_1, \dots, M_l](p); P$	input
$P ::=$	output process
$\overline{c[M_1, \dots, M_l]}\langle N \rangle; Q$	output
$\mathsf{new}\ x[\widetilde{i}] : T; P$	random number
$\mathsf{let}\ p = M\ \mathsf{in}\ P\ \mathsf{else}\ P'$	assignment
$\mathsf{if}\ M\ \mathsf{then}\ P\ \mathsf{else}\ P'$	conditional
$\mathsf{find}[\mathit{unique?}]\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}$ $\mathsf{suchthat}$ $\mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P$	array lookup
$\mathsf{insert}\ \mathit{Tbl}(M_1, \dots, M_l); P$	insert in table
$\mathsf{get}[\mathit{unique?}]\ \mathit{Tbl}(p_1, \dots, p_l)\ \mathsf{suchthat}\ M\ \mathsf{in}\ P\ \mathsf{else}\ P'$	get from table
$\mathsf{event}\ e(M_1, \dots, M_l); P$	event
$\mathsf{event\_abort}\ e$	event e and abort
yield	end

Figure 1: Syntax of the process calculus

newChannel $c; Q$ creates a new private channel c and executes Q; this construct is useful in proofs, but does not occur in games manipulated by CryptoVerif. The semantics of the input $c[M_1, \dots, M_l](p); P$ will be explained below together with the semantics of the output.

The output process new $x[\widetilde{i}] : T; P$ chooses a new random value in T, stores it in $x[\widetilde{i}]$, and executes P. The abbreviation $\widetilde{i}$ stands for a sequence of replication indices $i_1, \dots, i_m$. The random value is chosen according to the default distribution D_T for type T, which is determined as follows:

- When the type T is declared with option *nonuniform*, the default probability distribution D_T for type T may be non-uniform. It is left unspecified.
- Otherwise, if T is *fixed*, T consists of all bitstrings of a certain length, and the default distribution is the uniform distribution. The probability of each element of T is $1/|T|$.
- If T is *bounded* but not *fixed*, T is finite, and the default distribution is an approximately uniform distribution, such that its distance to the uniform distribution is at most ϵ_T. The distance between two probability distributions D_1 and D_2 for type T is

  $$d(D_1, D_2) = \frac{1}{2} \sum_{a \in T} |D_1(a) - D_2(a)|$$

  Indeed, probabilistic Turing machines that run in bounded time cannot choose random elements exactly uniformly in sets whose cardinal is not a power of 2.

  For example, a possible algorithm to obtain a random integer in $[0, m-1]$ is to choose a random integer x' uniformly among $[0, 2^k - 1]$ for a certain k large enough and return $x' \bmod m$. By euclidean division, we have $2^k = qm + r$ with $r \in [0, m-1]$. With this algorithm

  $$D(a) = \begin{cases} \frac{q+1}{2^k} & \text{if } a \in [0, r-1] \\ \frac{q}{2^k} & \text{if } a \in [r, m-1] \end{cases}$$

  so

  $$\left| D(a) - \frac{1}{m} \right| = \begin{cases} \frac{q+1}{2^k} - \frac{1}{m} & \text{if } a \in [0, r-1] \\ \frac{1}{m} - \frac{q}{2^k} & \text{if } a \in [r, m-1] \end{cases}$$

  Therefore

  $$\begin{aligned} d(D_T, \mathit{uniform}) &= \frac{1}{2} \sum_{a \in T} \left| D(a) - \frac{1}{m} \right| = \frac{1}{2} r \left(\frac{q+1}{2^k} - \frac{1}{m} \right) - \frac{1}{2}(m-r) \left(\frac{1}{m} - \frac{q}{2^k} \right) \\ &= \frac{r(m-r)}{m.2^k} \leq \frac{m}{2^{k+1}} \end{aligned}$$

  so we can take $\epsilon_T = \frac{m}{2^{k+1}}$. A given precision of $\epsilon_T = \frac{1}{2^{k'}}$ can be obtained by choosing $k = (k' + \text{number of bits of } m)$ random bits.

  By default, CryptoVerif does not display ϵ_T in probability formulas, to make them more readable.

When T is not declared with any of the options *nonuniform*, *fixed*, or *bounded*, CryptoVerif rejects the construct new $x[\widetilde{i}] : T; P$. Function symbols represent deterministic functions, so all random numbers must be chosen by new $x[\widetilde{i}] : T$. Deterministic functions make automatic syntactic manipulations easier: we can duplicate a term without changing its value.

The process $\mathsf{let}\ x[\widetilde{i}] : T = M\ \mathsf{in}\ P$ stores the value of M (which must be in T) in $x[\widetilde{i}]$ and executes P. Furthermore, we say that a function $f : T_1 \times \ldots \times T_m \rightarrow T$ is *efficiently injective* when it is injective and its inverses are efficiently computable, that is, there exist functions $f_j^{-1} : T \rightarrow T_j$ $(1 \leq j \leq m)$ such that $f_j^{-1}(f(x_1, \ldots, x_m)) = x_j$ and f_j^{-1} is efficiently computable. When f is efficiently injective, we define a pattern matching construct $\mathsf{let}\ f(x_1, \ldots, x_m) = M\ \mathsf{in}\ P\ \mathsf{else}\ P'$ as an abbreviation for $\mathsf{let}\ y : T = M\ \mathsf{in}\ \mathsf{let}\ x'_1 : T_1 = f_1^{-1}(y)\ \mathsf{in}\ \ldots \mathsf{let}\ x'_m : T_m = f_m^{-1}(y)\ \mathsf{in}\ \mathsf{if}\ f(x'_1, \ldots, x'_m) = y\ \mathsf{then}\ (\mathsf{let}\ x_1 : T_1 = x'_1\ \mathsf{in}\ \ldots \mathsf{let}\ x_m : T_m = x'_m\ \mathsf{in}\ P)\ \mathsf{else}\ P'$ where $y, x'_1, \ldots, x'_m$ are fresh variables. (The variables $x'_1, \ldots, x'_m$ are introduced to make sure that none of the variables $x_1, \ldots, x_m$ is defined when the pattern-matching fails.) We naturally generalize this construct to $\mathsf{let}\ p = M\ \mathsf{in}\ P\ \mathsf{else}\ P'$ where p is built from variables, efficiently injective functions, and equality tests. When p is simply a variable, the pattern-matching always succeeds, so the else branch of the assignment is never executed and can be omitted.

The process $\mathsf{event}\ e(M_1, \ldots, M_l); P$ executes the event $e(M_1, \ldots, M_l)$, then runs P. This event records that a certain program point has been reached with certain values of $M_1, \ldots, M_l$, but otherwise does not affect the execution of the process. Events are used in particular for specifying security properties.

The process $\mathsf{event\_abort}\ e$ executes event e (without argument) and aborts the game.

Next, we explain the process $\mathsf{find}[\mathit{unique}?]\ (\bigoplus_{j=1}^m u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \ldots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}\ \mathsf{suchthat}\ \mathsf{defined}(M_{j1}, \ldots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P$. The order and array indices on tuples are taken component-wise, so for instance, $u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \ldots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}$ can be further abbreviated $\widetilde{u_j}[\widetilde{i}] = \widetilde{i_j} \leq \widetilde{n_j}$. A simple example is the following: $\mathsf{find}\ u = i \leq n\ \mathsf{suchthat}\ \mathsf{defined}(x[i]) \wedge x[i] = a\ \mathsf{then}\ P'\ \mathsf{else}\ P$ tries to find an index i such that $x[i]$ is defined and $x[i] = a$, and when such an i is found, it stores it in u and 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, $\mathsf{find}\ u_1[\widetilde{i}] = i_1 \leq n_1, \ldots, u_m[\widetilde{i}] = i_m \leq n_m\ \mathsf{suchthat}\ \mathsf{defined}(M_1, \ldots, M_l) \wedge M\ \mathsf{then}\ P'\ \mathsf{else}\ P$ tries to find values of $i_1, \ldots, i_m$ for which $M_1, \ldots, M_l$ are defined and M is true. In case of success, it stores the obtained values in $u_1[\widetilde{i}], \ldots, u_m[\widetilde{i}]$ and executes P'. In case of failure, it executes P. This is further generalized to m branches: $\mathsf{find}\ (\bigoplus_{j=1}^m u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \ldots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}\ \mathsf{suchthat}\ \mathsf{defined}(M_{j1}, \ldots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P$ tries to find a branch j in $[1, m]$ such that there are values of $i_{j1}, \ldots, i_{jm_j}$ for which $M_{j1}, \ldots, M_{jl_j}$ are defined and M_j is true. In case of success, it stores them in $u_{j1}[\widetilde{i}], \ldots, u_{jm}[\widetilde{i}]$ and 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 $i_{j1}, \ldots, i_{jm_j}$ 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 one j and one value of $i_{j1}, \ldots, i_{jm_j}$ such that the corresponding condition is true, according to the distribution $D_{\mathsf{find}}(S)$ where S is the set of possible solutions $j, i_{j1}, \ldots, i_{jm_j}$, stores it in $u_{j1}[\widetilde{i}], \ldots, u_{jm_j}[\widetilde{i}]$, and executes P_j. The distribution $D_{\mathsf{find}}(S)$ is almost uniform: formally, the distance between $D_{\mathsf{find}}(S)$ and the uniform distribution is at most $\epsilon_{\mathsf{find}}/2$, that is, $d(D_{\mathsf{find}}(S), \mathit{uniform}) \leq \epsilon_{\mathsf{find}}/2$, that is, $\frac{1}{2} \sum_{v \in S} \left| D_{\mathsf{find}}(S)(v) - \frac{1}{|S|} \right| \leq \frac{\epsilon_{\mathsf{find}}}{2}$, so that, when $|S| = |S'|$, for any bijection $\phi : S \rightarrow S'$, $\frac{1}{2} \sum_{v \in S} |D_{\mathsf{find}}(S)(v) - D_{\mathsf{find}}(S')(\phi(v))| \leq \epsilon_{\mathsf{find}}$. Moreover $D_{\mathsf{find}}(S)(v_i) = D_{|S|}(i)$ where $S = \{v_1, \ldots, v_{|S|}\}$ with the values v_i ordered in increasing order lexicographically, for some distribution $D_{|S|}$ that depends only on the cardinal of S. In other words, the probability of a value v_i in the distribution $D_{\mathsf{find}}(S)$ does not depend on the values in the set S but only on the number $|S|$ of elements of S and on the position i of the value v_i in S ordered in increasing order lexicographically. Therefore, transformations that do not modify the number of successful values nor their order, that is, transformations that map elements v of S to elements $\phi(v)$ of S' at the same position i, preserve the probabilities exactly and we do not need to add ϵ_{find} to the probability when we apply such a transformation. This is true for instance when we remove a branch of find that is never taken. By default, CryptoVerif does not display ϵ_{find} in probability formulas, to make them more readable. We cannot take the first element found because the game transformations made by CryptoVerif may reorder the elements. For these transformations to preserve the behavior of the game, the distribution of the chosen element must be invariant by reordering, up to a small probability ϵ_{find}. In this definition, the variables $i_{j1}, \dots, i_{jm_j}$ are considered as replication indices, while $u_{j1}[\widetilde{i}], \dots, u_{jm_j}[\widetilde{i}]$ are considered as array variables. The indication $[\mathit{unique}?]$ stands for either $[\mathsf{unique}_e]$ or empty. The empty case has just been explained. When the find is marked $[\mathsf{unique}_e]$ and there are several solutions that make the condition of the find evaluate to true, we execute the event e and abort the game. When there is zero or one solution, the find is executed as when $[\mathit{unique}?]$ is empty. This semantics allows us to perform game transformations that require the find to have a single solution.

The conditional if M then P else P' executes P if M evaluates to true. Otherwise, it executes P'. CryptoVerif also supports the conditional if defined$(M_1, \dots, M_l) \wedge M$ then P else P', which executes P if $M_1, \dots, M_l$ are defined and M evaluates to true. Otherwise, it executes P'. This conditional is internally encoded as find suchthat defined$(M_1, \dots, M_l) \wedge M$ then P else P'. The conjunct M can be omitted when it is true, writing if defined$(M_1, \dots, M_l)$ then P else P'.

The constructs insert and get handle tables, used for instance to store the keys of the protocol participants. A table can be represented as a list of tuples; insert $\mathit{Tbl}(M_1, \dots, M_l); P$ inserts the element $M_1, \dots, M_l$ in the table Tbl; get $\mathit{Tbl}(x_1 : T_1, \dots, x_l : T_l)$ suchthat M in P else P' tries to retrieve an element $(x_1, \dots, x_l)$ in the table Tbl such that M is true. When such an element is found, it executes P with $x_1, \dots, x_l$ bound to that element. (When several such elements are found, one of them is chosen randomly according to distribution $D_{\mathsf{get}}(S)$ where S is the set of indices of suitable elements, with $d(D_{\mathsf{get}}(S), \mathit{uniform}) \leq \epsilon_{\mathsf{find}}/2$.) When no such element is found, P' is executed. We can generalize this construct to patterns instead of variables similarly to the let case. As in the case of find, the indication $[\mathit{unique}?]$ stands for either $[\mathsf{unique}_e]$ or empty. The empty case has just been explained. When the get is marked $[\mathsf{unique}_e]$ and there are several solutions, we execute the event e and abort the game. When there is zero or one solution, the get is executed as when $[\mathit{unique}?]$ is empty. CryptoVerif internally translates the insert and get constructs into find.

Let us explain the output $\overline{c[M_1, \dots, M_l]}\langle N\rangle; Q$. A channel $c[M_1, \dots, M_l]$ consists of both a channel name c and a tuple of terms $M_1, \dots, M_l$. Channel names c can be declared private by newChannel c; the adversary can never have access to channel $c[M_1, \dots, M_l]$ when c is private. (This is useful in the proofs, although all channels of protocols are often public.) Terms $M_1, \dots, M_l$ 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 $\overline{c[M_1, \dots, M_l]}\langle N\rangle; Q$, one looks for an input on channel $c[M'_l \dots, M'_l]$, where $M'_1, \dots, M'_l$ evaluate to the same bitstrings as $M_1, \dots, M_l$, in the available input processes. If no such input process is found, the process blocks. Otherwise, one such input process $c[M'_1, \dots, M'_l](x[\widetilde{i}] : T); P$ is chosen randomly according to the probability distribution $D_{\mathsf{in}}(S)$ where S is the multiset of suitable input processes. The communication is then executed: the output message N is evaluated and stored in $x[\widetilde{i}]$ if it is in T (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. The input construct can be generalized to patterns instead of variables similarly to the let case; when pattern-matching fails, the input process executes yield. The syntax requires an output to be followed by an input process, as in [50]. 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 $!^{i\leq n}c[i](x[i]:T)\dots\overline{c'[i]}\langle M\rangle\dots$ 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.

The yield construct is an abbreviation for $\overline{\mathit{yield}}\langle()\rangle$. By performing an output, this construct returns control to the adversary, which is going to receive the message. An else branch of find, if, get, or let may be omitted when it is $\mathsf{else\ yield}$. (Note that "else 0" would not be syntactically correct.) Similarly, $;\mathsf{yield}$ may be omitted after event, new, or insert and $\mathsf{in\ yield}$ may be omitted after let. A trailing 0 after an output may be omitted.

The *current replication indices* at a certain program point in a process are the replication indices $i_1,\dots,i_m$ bound by replications and find above that program point. The replication $!^{i\leq n}Q$ binds the replication index i in Q. The find construct $\mathsf{find}[\mathit{unique}?]\ (\bigoplus_{j=1}^{m}u_{j1}[\widetilde{i}]=i_{j1}\leq n_{j1},\dots,u_{jm_j}[\widetilde{i}]=i_{jm_j}\leq n_{jm_j}\ \mathsf{suchthat\ defined}(M_{j1},\dots,M_{jl_j})\wedge M_j\ \mathsf{then}\ \dots)\ \mathsf{else}\ \dots$ binds the replication indices $i_{j1},\dots,i_{jm_j}$ in $\mathsf{defined}(M_{j1},\dots,M_{jl_j})\wedge M_j$. We often abbreviate $x[i_1,\dots,i_m]$ by x when $i_1,\dots,i_m$ are the current replication indices at the definition of x, but it should be kept in mind that this is only an abbreviation. Variables defined under a replication must be arrays: for example $!^{i_1\leq n_1}\dots!^{i_m\leq n_m}\mathsf{let}\ x[i_1,\dots,i_m]:T=M\ \mathsf{in}\ \dots$ 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 $x[i_1,\dots,i_m]$ in Q_0, the indices $i_1,\dots,i_m$ 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, if (or let), or get.

   In a let with pattern-matching, $\mathsf{let}\ p=M\ \mathsf{in}\ P\ \mathsf{else}\ P'$, the variables bound by p are considered to be defined in the in branch; however, the variables defined in M and in terms included in the pattern p are defined before the branching.

   In $\mathsf{get}[\mathit{unique}?]\ \mathit{Tbl}(p_1,\dots,p_l)\ \mathsf{suchthat}\ M\ \mathsf{in}\ P\ \mathsf{else}\ P'$, the variables bound by p_j for $j\leq l$ and the variables defined in M and in terms included in the patterns p_j are (temporarily) defined before the branching.

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.) A definition of $x[\widetilde{i}]$ can be $\mathsf{new}\ x[\widetilde{i}]:T$, a let, get, or input that contains the pattern $x[\widetilde{i}]:T$, or $\mathsf{find}\dots x[\widetilde{i}]=i\leq n\dots$.

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

- syntactically under the definition of $x[M_1,\dots,M_m]$ (in which case $M_1,\dots,M_m$ are in fact the current replication indices at the definition of x);

- or in a $\mathsf{defined}$ condition in a find process or term;

- or in M'_j in a process or term of the form $\mathsf{find}\ (\bigoplus_{j=1}^{m''}\widetilde{u_j}[\widetilde{i}]=\widetilde{i_j}\leq\widetilde{n_j}\ \mathsf{suchthat\ defined}(M'_{j1},\dots,M'_{jl_j})\wedge M'_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P$ where for some $k\leq l_j$, $x[M_1,\dots,M_m]$ is a subterm of M'_{jk}.

- or in P_j in a process or term of the form $\mathsf{find}\ (\bigoplus_{j=1}^{m''} \widetilde{u_j}[\widetilde{i}] = \widetilde{i_j} \leq \widetilde{n_j}\ \mathsf{suchthat}\ \mathsf{defined}(M'_{j1}, \ldots, M'_{jl_j}) \wedge M'_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P$ where for some $k \leq l_j$, there is a subterm N of M'_{jk} such that $N\{\widetilde{u_j}[\widetilde{i}]/\widetilde{i_j}\} = x[M_1, \ldots, M_m]$.

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 two items). The scope of variable definitions is defined as follows: $x[\widetilde{i}]$ is syntactically under its definition when it is

- inside P in $\mathsf{new}\ x[\widetilde{i}] : T; P$;
- inside N in $\mathsf{new}\ x[\widetilde{i}] : T; N$;
- inside P in $\mathsf{let}\ p = M\ \mathsf{in}\ P\ \mathsf{else}\ P'$ when $x[\widetilde{i}] : T$ is bound in the pattern p;
- inside N in $\mathsf{let}\ p = M\ \mathsf{in}\ N\ \mathsf{else}\ N'$ when $x[\widetilde{i}] : T$ is bound in the pattern p;
- inside N in $\mathsf{let}\ x[\widetilde{i}] : T = M\ \mathsf{in}\ N$;
- inside P_j in $\mathsf{find}[\mathit{unique}?]\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \ldots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}\ \mathsf{suchthat}\ \mathsf{defined}(M_{j1}, \ldots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P$ when x is u_{jk} for some $k \leq m_j$;
- inside N_j in $\mathsf{find}[\mathit{unique}?]\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \ldots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}\ \mathsf{suchthat}\ \mathsf{defined}(M_{j1}, \ldots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ N_j)\ \mathsf{else}\ N$ when x is u_{jk} for some $k \leq m_j$;
- inside M or P in $\mathsf{get}[\mathit{unique}?]\ \mathit{Tbl}(p_1, \ldots, p_l)\ \mathsf{suchthat}\ M\ \mathsf{in}\ P\ \mathsf{else}\ P'$ when $x[\widetilde{i}] : T$ is bound in one of the patterns $p_1, \ldots, p_l$;
- inside M or N in $\mathsf{get}[\mathit{unique}?]\ \mathit{Tbl}(p_1, \ldots, p_l)\ \mathsf{suchthat}\ M\ \mathsf{in}\ N\ \mathsf{else}\ N'$ when $x[\widetilde{i}] : T$ is bound in one of the patterns $p_1, \ldots, p_l$;
- inside P in $c[M_1, \ldots, M_l](p); P$ when $x[\widetilde{i}] : T$ is bound in the pattern p.

A variable access that does not correspond to the first item of Invariant 2 is called an *array access*. We furthermore require the following invariant.

Invariant 3 (Variables defined in find and get conditions) The process Q_0 satisfies Invariant 3 with public variables V if and only if the variables defined in conditions of find and the variables defined in patterns and in conditions of get have no array accesses and are not in the set of variables V.

These conditions are needed for variables of get, because they will be transformed into variables defined in conditions of find by the transformation of get into find.

Invariant 4 (Terms in find and get conditions) The process Q_0 satisfies Invariant 4 if and only if event and insert do not occur in conditions of find and get.

Invariant 4 guarantees that evaluating the condition of a find or get does not change the state of the system.

Definition 1 A term is *simple* when it contains only replication indices, variables, and function applications.

Invariant 5 (Terms in input channels and defined conditions) The process Q_0 satisfies Invariant 5 if and only if all terms in input channels $c[M_1, \dots, M_l]$ and in conditions $\mathsf{defined}(M_1, \dots, M_l)$ in find are simple.

Terms that are not simple are handled by expanding them into their corresponding processes. Invariant 5 is needed because terms in input channels cannot be expanded, as we need an output process to put the computations coming from expanded terms, and similarly terms in $\mathsf{defined}$ conditions cannot be expanded (see the transformation **expand** in Section 5.1.3). By combining this invariant with Invariant 2, we see that the terms of all variable accesses $x[M_1, \dots, M_m]$ are simple.

The last 3 invariants did not appear in previous versions of the calculus because all terms were simple.

Invariant 6 (Events) We distinguish three disjoint sets of events e, Shoup events, non-unique events, and other events. The process Q_0 satisfies Invariant 6 if and only if

- Shoup events occur only in processes of the form $\mathsf{event\_abort}\ e$ in Q_0,
- non-unique events occur only in $\mathsf{find}[\mathsf{unique}_e]$ or $\mathsf{get}[\mathsf{unique}_e]$ in Q_0, and
- other events occur in $\mathsf{event}\ e(M_1, \dots, M_l)$ or in $\mathsf{event\_abort}\ e$ in Q_0.

The name "Shoup events" is used because these events are introduced when applying Shoup's lemma [63] (see Section 5.1.12). The non-unique events are those triggered when a $\mathsf{find}[\mathsf{unique}_e]$ or $\mathsf{get}[\mathsf{unique}_e]$ actually has several solutions.

All these invariants are checked by the prover for the initial game and preserved by all game transformations.

We denote by $\mathrm{var}(P)$ the set of variables that occur in P, $\mathrm{vardef}(P)$ the set of variables defined in P ($\mathrm{var}(P)$ may contain more variables than $\mathrm{vardef}(P)$ in case some variables are read using find but never defined), 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 2 Let T_{mk} and T_{ms} be types that correspond intuitively to keys and message authentication codes, respectively; T_{mk} is a fixed-length type. A message authentication code scheme MAC [19] consists of two function symbols:

- $\mathrm{mac} : \mathit{bitstring} \times T_{mk} \to T_{ms}$ is the MAC algorithm taking as arguments 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 random coins as an additional argument.)
- $\mathrm{verify} : \mathit{bitstring} \times T_{mk} \times T_{ms} \to \mathit{bool}$ is a verification algorithm such that $\mathrm{verify}(m, k, t) = \mathrm{true}$ if and only if t is a valid MAC of message m under key k. (Since mac is deterministic, $\mathrm{verify}(m, k, t)$ is typically $\mathrm{mac}(m, k) = t$.)

We have $\forall m \in \mathit{bitstring}, \forall k \in T_{mk}, \mathrm{verify}(m, k, \mathrm{mac}(m, k)) = \mathrm{true}$.

The advantage of an adversary against unforgeability under chosen message attacks (UF-CMA) is

$$\mathsf{Succ}^{\mathsf{uf-cma}}_{\mathsf{MAC}}(t, q_m, q_v, l) = \max_{\mathcal{A}} \Pr\begin{bmatrix} k \overset{R}{\leftarrow} T_{mk}; (m, s) \leftarrow \mathcal{A}^{\mathrm{mac}(.,k),\mathrm{verify}(.,k,.)} : \mathrm{verify}(m, k, s) \\ \wedge\, m \text{ was never queried to the oracle } \mathrm{mac}(., k) \end{bmatrix}$$

where the adversary $\mathcal{A}$ is any probabilistic Turing machine that runs in time at most t, calls $\mathrm{mac}(.,k)$ at most q_m times with messages of length at most l, and calls $\mathrm{verify}(.,k,.)$ at most q_v times with messages of length at most l.

$\mathsf{Succ}_{\mathsf{MAC}}^{\mathsf{uf-cma}}(t, q_m, q_v, l)$ is the probability that an adversary forges a MAC, that is, returns a pair (m, s) where s is a correct MAC for m, without having queried the MAC oracle $\mathrm{mac}(.,k)$ on m. Intuitively, when the MAC is secure, this probability is small: the adversary has little chance of forging a MAC. Hence, the MAC guarantees the integrity of the MACed message because one cannot compute the MAC without the secret key.

Two frameworks exist for expressing security properties. In the asymptotic framework, used in [23, 24], the length of keys is determined by a security parameter η, and a MAC is UF-CMA when $\mathsf{Succ}_{\mathsf{MAC}}^{\mathsf{uf-cma}}(t, q_m, q_v, l)$ is a negligible function of η when t is polynomial in η. ($f(\eta)$ is *negligible* when for all polynomials q, there exists $\eta_o \in \mathbb{N}$ such that for all $\eta > \eta_0$, $f(\eta) \le \frac{1}{q(\eta)}$.) The assumption that functions are efficiently computable means that they are computable in time polynomial in η and in the length of their arguments. The goal is to show that the probability of success of an attack against the protocol is negligible, assuming the parameters n are polynomial in η and the network messages are of length polynomial in η. In contrast, in the exact security framework, on which we focus in this report, one computes the probability of success of an attack against the protocol as a function of the probability of breaking the primitives such as $\mathsf{Succ}_{\mathsf{MAC}}^{\mathsf{uf-cma}}(t, q_m, q_v, l)$, of the runtime of functions, of the parameters n, and of the length of messages, thus providing a more precise security result. Intuitively, the probability $\mathsf{Succ}_{\mathsf{MAC}}^{\mathsf{uf-cma}}(t, q_m, q_v, l)$ is assumed to be small (otherwise, the computed probability of attack will be large), but no formal assumption on this probability is needed to establish the security theorem.

Definition 3 Let T_k, T_r, and T_e be types for random coins, keys, and ciphertexts respectively. T_k and T_r are fixed-length types. A symmetric encryption scheme SE [19] consists of two function symbols:

- $\mathrm{enc} : \mathit{bitstring} \times T_k \times T_r \to T_e$ is the encryption algorithm taking as arguments the cleartext, the key, and random coins, and returning the ciphertext,
- $\mathrm{dec} : T_e \times T_k \to \mathit{bitstring}_\perp$ is the decryption algorithm taking as arguments the ciphertext and the key, and returning either the cleartext when decryption succeeds or $\perp$ when decryption fails,

such that $\forall k \in T_k, \forall m \in \mathit{bitstring}, \forall r \in T_r$, $\mathrm{dec}(\mathrm{enc}(m,k,r),k) = m$.

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. The advantage of an adversary against indistinguishability under chosen plaintext attacks (IND-CPA) is

$$\mathsf{Succ}_{\mathsf{SE}}^{\mathsf{ind-cpa}}(t, q_e, l) = \max_{\mathcal{A}} 2 \Pr\begin{bmatrix} b \stackrel{R}{\leftarrow} \{0,1\}; k \stackrel{R}{\leftarrow} T_k; \\ b' \leftarrow \mathcal{A}^{r \stackrel{R}{\leftarrow} T_r; \mathrm{enc}(LR(.,.,b),k,r)} : b' = b \end{bmatrix} - 1$$

where $\mathcal{A}$ is any probabilistic Turing machine that runs in time at most t and calls $r \stackrel{R}{\leftarrow} T_r$; $\mathit{enc}(LR(.,.,b),k,r)$ at most q_e times on messages of length at most l.

Given two bitstrings a_0 and a_1 of the same length, the left-right encryption oracle $r \stackrel{R}{\leftarrow} T_r$; $\mathit{enc}(LR(.,.,b),k,r)$ returns $r \stackrel{R}{\leftarrow} T_r; \mathrm{enc}(LR(a_0,a_1,b),k,r)$, that is, encrypts a_0 when $b = 0$ and a_1 when $b = 1$. $\mathsf{Succ}_{\mathsf{SE}}^{\mathsf{ind-cpa}}(t, q_e, l)$ is the probability that the adversary distinguishes the encryption of the messages a_0 given as first arguments to the left-right encryption oracle from the encryption of the messages a_1 given as second arguments. Intuitively, when the encryption scheme is IND-CPA secure, this probability is small: the ciphertext gives almost no information on what the cleartext is (one cannot determine whether it is a_0 or a_1 without having the secret key).

Example 1 Let us consider the following trivial protocol:

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

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. In other words, the protocol sends the key x'_k encrypted using an encrypt-then-MAC scheme [19]. 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:

$$\begin{aligned} Q_0 &= \mathit{start}(); \mathsf{new}\ x_k : T_l; \mathsf{new}\ x_{mk} : T_{mk}; \overline{c}\langle\rangle; (Q_A \mid Q_B) \\ Q_A &= !^{i \leq n} c_A[i](); \mathsf{new}\ x'_k : T_k; \mathsf{new}\ x_r : T_r; \\ &\qquad \mathsf{let}\ x_m : \mathit{bitstring} = \mathrm{enc}(\mathrm{k2b}(x'_k), x_k, x_r)\ \mathsf{in}\ \overline{c_A[i]}\langle x_m, \mathrm{mac}(x_m, x_{mk})\rangle \\ Q_B &= !^{i' \leq n} c_B[i'](x'_m, x_{ma}); \mathsf{if}\ \mathrm{verify}(x'_m, x_{mk}, x_{ma})\ \mathsf{then} \\ &\qquad \mathsf{let}\ \mathrm{i}_\perp(\mathrm{k2b}(x''_k)) = \mathrm{dec}(x'_m, x_k)\ \mathsf{in}\ \overline{c_B[i']}\langle\rangle \end{aligned}$$

When Q_0 receives a message on channel *start*, it begins execution: it generates the keys x_k and x_{mk} randomly. Then it yields control to 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 adversary outputs on channels $c_A[i]$ or $c_B[i]$ respectively. In a session that runs as expected, the adversary 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 $\mathrm{k2b}(x) = x$; it is needed only for type conversion. The adversary is then expected to forward this message on $c_B[i]$. When Q_B receives this message, it verifies 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 adversary is responsible for forwarding messages from A to B. It can send messages in unexpected ways in order to mount an attack.

This very small example is sufficient to illustrate the main features of CryptoVerif.

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 or find that bind $i_1, \ldots, i_m$ with declaration $i_j \leq n_j$. (For instance, the definition of $x[i_1, \ldots, i_m]$ occurs under $!^{i_1 \leq n_1}, \ldots, !^{i_m \leq n_m}$ or it occurs in the condition of find $u_1 = i_1 \leq n_1, \ldots, u_m = i_m \leq n_m$ under no replication.

The type T is the one given in the definition of x in $\mathsf{new}\ x[\widetilde{i}] : T$ or in a pattern $x[\widetilde{i}] : T$ in an assignment, an input, or a get. In the find construct, $\mathsf{find} \dots x[\widetilde{i}] = i \leq n$, the type T of x is $T = [1, n]$.) 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 $\mathcal{E}$ using the rules of Figures 2 and 3. These figures defines four judgments:

- $\mathcal{E} \vdash M : T$ means that the term M has type T in environment $\mathcal{E}$.
- $\mathcal{E} \vdash p : T$ means that the pattern p 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 $x[M_1, \dots, M_m]$, $M_1, \dots, M_m$ must be of the suitable interval type. When $f(M_1, \dots, M_m)$ is called and $f : T_1 \times \dots \times T_m \to T$, M_j must be of type T_j, and $f(M_1, \dots, M_m)$ is then of type T.

The term $\mathsf{new}\ x[\widetilde{i}] : T; N$ is accepted only when T is declared *fixed*, *bounded*, or *nonuniform*. We check that $x[\widetilde{i}]$ is of type T (which is in fact always true when the construction of $\mathcal{E}$ succeeds). N must well-typed, and its type is also the type of $\mathsf{new}\ x[\widetilde{i}] : T; N$.

In $\mathsf{let}\ p = M\ \mathsf{in}\ N\ \mathsf{else}\ N'$, p must have the same type as M, and N and N' must have the same type, which is also the type of $\mathsf{let}\ p = M\ \mathsf{in}\ N\ \mathsf{else}\ N'$. The typing rules for patterns p are found at the bottom of Figure 2. The pattern $x[\widetilde{i}] : T$ has type T, provided $x[\widetilde{i}]$ has type T (which is in fact always true when the construction of $\mathcal{E}$ succeeds). The other typing rules for patterns are straightforward. The particular case $\mathsf{let}\ x[\widetilde{i}] : T = M\ \mathsf{in}\ N$ is typed similarly, except that the else branch is omitted.

In $\mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ N'$, M must be of type *bool* and N and N' must have the same type, which is also the type of $\mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ N'$.

In

$$\begin{aligned}&\mathsf{find}[\mathit{unique}?]\ (\textstyle\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}\ \mathsf{suchthat}\\ &\quad \mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ N_j)\ \mathsf{else}\ N\end{aligned}$$

the replication indices $i_{j1}, \dots, i_{jm_j}$ are bound in $M_{j1}, \dots, M_{jl_j}, M_j$, of types $[1, n_{j1}], \dots, [1, n_{jm_j}]$ respectively; M_j is of type *bool* for all $j \leq m$; N_j for all $j \leq m$ and N all have the same type, which is also the type of the find term.

In $\mathsf{insert}\ \mathit{Tbl}(M_1, \dots, M_l); N$, $M_1, \dots, M_l$ must be of the type declared for the elements of the table Tbl, and the type of N is the type of the insert term.

In $\mathsf{get}[\mathit{unique}?]\ \mathit{Tbl}(p_1, \dots, p_l)\ \mathsf{suchthat}\ M\ \mathsf{in}\ N\ \mathsf{else}\ N'$, $p_1, \dots, p_l$ must be of the type declared for the elements of the table Tbl and M must be of type *bool*. The terms N and N' must have the same type, which is also the type of the get term.

In $\mathsf{event}\ e(M_1, \dots, M_l); N$, $M_1, \dots, M_l$ must be of the type declared for the arguments of event e, and the type of N is the type of the event term.

The term $\mathsf{event\_abort}\ e$ can have any type (because it aborts the game); the event e must be declared without argument, which we denote by $e : ()$.

The type system for processes requires each subterm to be well-typed. In $!^{i \leq n} Q$, i is of type $[1, n]$ in Q. The processes new, let, if, find, insert, get, event, and $\mathsf{event\_abort}$ are typed similarly to the corresponding terms.

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

Typing rules for terms:

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

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

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

$$\frac{T \textit{ fixed}, \textit{bounded}, \text{ or } \textit{nonuniform} \qquad \mathcal{E} \vdash x[\widetilde{i}] : T \qquad \mathcal{E} \vdash N : T'}{\mathcal{E} \vdash \mathsf{new}\ x[\widetilde{i}] : T; N : T'} \qquad \text{(TNewT)}$$

$$\frac{\mathcal{E} \vdash M : T \qquad \mathcal{E} \vdash p : T \qquad \mathcal{E} \vdash N : T' \qquad \mathcal{E} \vdash N' : T'}{\mathcal{E} \vdash \mathsf{let}\ p = M\ \mathsf{in}\ N\ \mathsf{else}\ N' : T'} \qquad \text{(TLetT)}$$

$$\frac{\mathcal{E} \vdash M : T \qquad \mathcal{E} \vdash x[\widetilde{i}] : T \qquad \mathcal{E} \vdash N : T'}{\mathcal{E} \vdash \mathsf{let}\ x[\widetilde{i}] : T = M\ \mathsf{in}\ N : T'} \qquad \text{(TLetT2)}$$

$$\frac{\mathcal{E} \vdash M : \textit{bool} \qquad \mathcal{E} \vdash N : T \qquad \mathcal{E} \vdash N' : T}{\mathcal{E} \vdash \mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ N' : T} \qquad \text{(TIfT)}$$

$$\frac{\begin{array}{c}\forall j \leq m, \forall k \leq m_j, \mathcal{E} \vdash u_{jk}[\widetilde{i}] : [1, n_{jk}] \\ \forall j \leq m, \forall k \leq l_j, \mathcal{E}[i_{j1} \mapsto [1, n_{j1}], \ldots, i_{jm_j} \mapsto [1, n_{jm_j}]] \vdash M_{jk} : T_{jk} \\ \forall j \leq m, \mathcal{E}[i_{j1} \mapsto [1, n_{j1}], \ldots, i_{jm_j} \mapsto [1, n_{jm_j}]] \vdash M_j : \textit{bool} \\ \forall j \leq m, \mathcal{E} \vdash N_j : T \qquad \mathcal{E} \vdash N : T\end{array}}{\begin{array}{c}\mathcal{E} \vdash \mathsf{find}[\textit{unique}?]\ (\bigoplus_{j=1}^m u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \ldots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}\ \mathsf{suchthat} \\ \mathsf{defined}(M_{j1}, \ldots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ N_j)\ \mathsf{else}\ N : T\end{array}} \qquad \text{(TFindT)}$$

$$\frac{\textit{Tbl} : T_1 \times \ldots \times T_l \qquad \forall j \leq l, \mathcal{E} \vdash M_j : T_j \qquad \mathcal{E} \vdash N : T}{\mathcal{E} \vdash \mathsf{insert}\ \textit{Tbl}(M_1, \ldots, M_l); N : T} \qquad \text{(TInsertT)}$$

$$\frac{\textit{Tbl} : T_1 \times \ldots \times T_l \qquad \forall j \leq l, \mathcal{E} \vdash p_j : T_j \qquad \mathcal{E} \vdash M : \textit{bool} \qquad \mathcal{E} \vdash N : T \qquad \mathcal{E} \vdash N' : T}{\mathsf{get}[\textit{unique}?]\ \textit{Tbl}(p_1, \ldots, p_l)\ \mathsf{suchthat}\ M\ \mathsf{in}\ N\ \mathsf{else}\ N' : T} \qquad \text{(TGetT)}$$

$$\frac{e : T_1 \times \ldots \times T_l \qquad \forall j \leq l, \mathcal{E} \vdash M_j : T_j \qquad \mathcal{E} \vdash N : T}{\mathcal{E} \vdash \mathsf{event}\ e(M_1, \ldots, M_l); N : T} \qquad \text{(TEventT)}$$

$$\frac{e : ()}{\mathcal{E} \vdash \mathsf{event\_abort}\ e : T'} \qquad \text{(TEventAbortT)}$$

Typing rules for patterns:

$$\frac{\mathcal{E} \vdash x[\widetilde{i}] : T}{\mathcal{E} \vdash (x[\widetilde{i}] : T) : T} \qquad \text{(TVarP)}$$

$$\frac{f : T_1 \times \ldots \times T_m \to T \qquad \forall j \leq m, \mathcal{E} \vdash p_j : T_j}{\mathcal{E} \vdash f(p_1, \ldots, p_m) : T} \qquad \text{(TFunP)}$$

$$\frac{\mathcal{E} \vdash M : T}{\mathcal{E} \vdash =M : T} \qquad \text{(TEqP)}$$

Figure 2: Typing rules (1)

Typing rules for input processes:

$$\mathcal{E} \vdash 0 \tag{TNil}$$

$$\frac{\mathcal{E} \vdash Q \qquad \mathcal{E} \vdash Q'}{\mathcal{E} \vdash Q \mid Q'} \tag{TPar}$$

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

$$\frac{\mathcal{E} \vdash Q}{\mathcal{E} \vdash \mathsf{newChannel}\ c; Q} \tag{TNewChannel}$$

$$\frac{\forall j \leq l, \mathcal{E} \vdash M_j : T'_j \qquad \mathcal{E} \vdash p : T \qquad \mathcal{E} \vdash P}{\mathcal{E} \vdash c[M_1, \dots, M_l](p); P} \tag{TIn}$$

Typing rules for output processes:

$$\frac{\forall j \leq l, \mathcal{E} \vdash M_j : T'_j \qquad \mathcal{E} \vdash N : T \qquad \mathcal{E} \vdash Q}{\mathcal{E} \vdash \overline{c[M_1, \dots, M_l]}\langle N \rangle; Q} \tag{TOut}$$

$$\frac{T \textit{ fixed}, \textit{ bounded}, \text{ or } \textit{nonuniform} \qquad \mathcal{E} \vdash x[\widetilde{i}] : T \qquad \mathcal{E} \vdash P}{\mathcal{E} \vdash \mathsf{new}\ x[\widetilde{i}] : T; P} \tag{TNew}$$

$$\frac{\mathcal{E} \vdash M : T \qquad \mathcal{E} \vdash p : T \qquad \mathcal{E} \vdash P \qquad \mathcal{E} \vdash P'}{\mathcal{E} \vdash \mathsf{let}\ p = M\ \mathsf{in}\ P\ \mathsf{else}\ P'} \tag{TLet}$$

$$\frac{\mathcal{E} \vdash M : \textit{bool} \qquad \mathcal{E} \vdash P \qquad \mathcal{E} \vdash P'}{\mathcal{E} \vdash \mathsf{if}\ M\ \mathsf{then}\ P\ \mathsf{else}\ P'} \tag{TIf}$$

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

$$\frac{\textit{Tbl} : T_1 \times \ldots \times T_l \qquad \forall j \leq l, \mathcal{E} \vdash M_j : T_j \qquad \mathcal{E} \vdash P}{\mathcal{E} \vdash \mathsf{insert}\ \textit{Tbl}(M_1, \dots, M_l); P} \tag{TInsert}$$

$$\frac{\textit{Tbl} : T_1 \times \ldots \times T_l \qquad \forall j \leq l, \mathcal{E} \vdash p_j : T_j \qquad \mathcal{E} \vdash M : \textit{bool} \qquad \mathcal{E} \vdash P \qquad \mathcal{E} \vdash P'}{\mathsf{get}[\textit{unique?}]\ \textit{Tbl}(p_1, \dots, p_l)\ \mathsf{suchthat}\ M\ \mathsf{in}\ P\ \mathsf{else}\ P'} \tag{TGet}$$

$$\frac{e : T_1 \times \ldots \times T_l \qquad \forall j \leq l, \mathcal{E} \vdash M_j : T_j \qquad \mathcal{E} \vdash P}{\mathcal{E} \vdash \mathsf{event}\ e(M_1, \dots, M_l); P} \tag{TEvent}$$

$$\frac{e : ()}{\mathcal{E} \vdash \mathsf{event\_abort}\ e} \tag{TEventAbort}$$

$$\mathcal{E} \vdash \mathsf{yield} \tag{TYield}$$

Figure 3: Typing rules (2)

in Q and with $i_{j1} \mapsto [1, n_{j1}], \dots, i_{jm_j} \mapsto [1, n_{jm_j}]$ for each $\mathsf{find}[\mathit{unique}?]$ $(\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}$ $\mathsf{suchthat}$ $\mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j$ then $P_j)$ else P such that the considered occurrence of M is in the condition $\mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j$.

Invariant 7 (Typing) The process Q_0 satisfies Invariant 7 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 values may appear at each point of the protocol.

2.4 Formal Semantics

2.4.1 Definition of the Semantics

The formal semantics of our calculus is presented in Figures 4, 5, 7, 8, and 9.

In this semantics, each term M or process P or Q is labeled by a program point μ, replacing M with ${}^{\mu}M$ and similarly for P and Q. We still use the notations M, P, Q for terms and processes tagged with program points. The program points are used in order to track from where each term or process comes from in the initial process. These program points are simply constant tags, and the initial process is tagged with a distinct program point at each subterm and subprocess.

A semantic configuration is a sextuple $E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$, where

- E is an environment mapping array cells to values.
- (σ, P) is the output process P currently scheduled, with the associated mapping sequence σ which gives values of replication indices.

  The mapping sequence $\sigma = [i_1 \mapsto a_1, \dots, i_m \mapsto a_m]$ is a sequence of mappings $i_j \mapsto a_j$, which can also be interpreted as a function: $\sigma(i_j) = a_j$ for all $j \leq m$, and $\sigma(\widetilde{i})$ is defined by the natural extension to sequences. However, using a sequence allows us to define $\mathrm{Dom}(\sigma) = [i_1, \dots, i_m]$ to be the sequence of current replication indices and $\mathrm{Im}(\sigma) = [a_1, \dots, a_m]$ to be the sequence of their values. When $\sigma = [i_1 \mapsto a_1, \dots, i_m \mapsto a_m]$, $\sigma[i_{m+1} \mapsto a_{m+1}, \dots, i_l \mapsto a_l] = [i_1 \mapsto a_1, \dots, i_l \mapsto a_l]$.
- $\mathcal{Q}$ is the multiset of input processes running in parallel with P, with their associated mapping sequences giving values of replication indices.
- $\mathcal{C}h$ is the set of channels already created.
- $\mathcal{T}$ defines the contents of tables. It is a list of $\mathit{Tbl}(a_1, \dots, a_m)$ indicating that table Tbl contains the element $(a_1, \dots, a_m)$.
- $\mu\mathcal{E}v$ is a sequence representing the events executed so far. Each element of the sequence is of the form $(\mu, \widetilde{a}) : e(a_1, \dots, a_m)$, meaning that the event $e(a_1, \dots, a_m)$ has been executed at program point μ with replication indices evaluating to $\widetilde{a}$.

  We define $\mathcal{E}v = \mathrm{removepp}(\mu\mathcal{E}v) = [e(a_1, \dots, a_m) \mid (\mu, \widetilde{a}) : e(a_1, \dots, a_m) \in \mu\mathcal{E}v]$ to be the sequence of events $\mu\mathcal{E}v$ without their associated program points and replication indices.

In addition to the grammar given in Figure 1, the terms M of the semantics can be values a and abort event values $\mathsf{event\_abort}\ (\mu, \widetilde{a}) : e$, and the processes P can be abort, corresponding to the situation in which the game has been aborted. These additional terms and processes are not tagged with program points. (They do not occur in the initial process.)

The semantics is defined by reduction rules of the form $E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', (\sigma', P'), \mathcal{Q}', \mathcal{C}h', \mathcal{T}', \mu\mathcal{E}v'$ meaning that $E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ reduces to $E', (\sigma', P'), \mathcal{Q}', \mathcal{C}h', \mathcal{T}', \mu\mathcal{E}v'$ with probability p. 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:

$$\Pr[E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v \to E', (\sigma', P'), \mathcal{Q}', \mathcal{C}h', \mathcal{T}', \mu\mathcal{E}v'] = \sum_{E,(\sigma,P),\mathcal{Q},\mathcal{C}h,\mathcal{T},\mu\mathcal{E}v \xrightarrow{p}_t E',(\sigma',P'),\mathcal{Q}',\mathcal{C}h',\mathcal{T}',\mu\mathcal{E}v'} p$$

The probability of a trace $Tr = E_1, (\sigma_1, P_1), \mathcal{Q}_1, \mathcal{C}h_1, \mathcal{T}_1, \mu\mathcal{E}v_1 \xrightarrow{p_1}_{t_1} \dots \xrightarrow{p_{m-1}}_{t_{m-1}} E_m, (\sigma_m, P_m), \mathcal{Q}_m, \mathcal{C}h_m, \mathcal{T}_m, \mu\mathcal{E}v_m$ is $\Pr[Tr] = p_1 \times \dots \times p_{m-1}$. We define the semantics only for patterns $x[\widetilde{i}] : T$, the other patterns can be encoded as outlined in Section 2.1.

In Figures 4 and 5, we define an auxiliary relation for evaluating terms: $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', \sigma, M', \mathcal{T}', \mu\mathcal{E}v'$ means that the term M reduces to M' in environment E with the replication indices defined by σ, the table contents $\mathcal{T}$, and the sequence of events $\mu\mathcal{E}v$, with probability p. Rule (ReplIndex) evaluates replication indices using the function σ. Rule (Var) looks for the value of the variable in the environment E. Rule (Fun) evaluates the function call. Rule (NewT) chooses a random $a \in T$ according to distribution D_T, and stores it in $x[\sigma(\widetilde{i})]$ by extending the environment E accordingly. Similarly, Rule (LetT) extends the environment E with the value of $x[\sigma(\widetilde{i})]$. Rule (IfT1) evaluates the then branch of if when the condition is true, and Rule (IfT2) evaluates the else branch otherwise.

Rules (FindTE) to (FindT3) define the semantics of find. First, they all evaluate the conditions for all branches j and all values of the indices $i_{j1}, \dots, i_{jm_j}$. If one of these evaluations executes an event (which can happen in case the condition contains an $\mathsf{event\_abort}\ e$ or a $\mathsf{find}[\mathsf{unique}_e]$), the whole find executes the same event; in case the evaluations of the conditions execute several different events, one of them is chosen randomly, according to distribution $D_{\mathsf{find}}(S)$, that is, almost uniformly over the choices of branches and indices (Rule (FindTE)). Otherwise, the branch and indices for which the condition is true are collected in a set S. If S is empty, the else branch of the find is executed (Rule (FindT2)). When S is not empty, two cases can happen. Either the find is not marked $[\mathsf{unique}_e]$, and we choose an element $v_0 = (j', a'_1, \dots, a'_{m_{j'}})$ of S randomly according to the distribution $D_{\mathsf{find}}(S)$, store the corresponding indices $a'_1, \dots, a'_{m_{j'}}$ in $u_{j'1}[\sigma(\widetilde{i})], \dots, u_{j'm_{j'}}[\sigma(\widetilde{i})]$ by extending the environment accordingly, and we continue with the selected branch N'_j. If the find is marked $[\mathsf{unique}_e]$ and S has a single element, we do the same. If the find is marked $[\mathsf{unique}_e]$ and S has several elements, we execute the event e (Rule (FindT3)). We recall that $D_{\mathsf{find}}(S)(v_0)$ denotes the probability of choosing v_0 in the distribution $D_{\mathsf{find}}(S)$. The terms in conditions of find may define variables, included in the environment E''; we ignore these additional variables and compute the final environment from the initial environment E, because these variables have no array accesses by Invariant 3, so the values of these variables are not used after the evaluation of the condition. The conditions of find, $D \wedge M$, are evaluated using Rules (DefinedNo) and (DefinedYes). If an element of the $\mathsf{defined}$ condition D is not defined, then the condition is false (Rule (DefinedNo)); when all elements of the $\mathsf{defined}$ condition are defined, we evaluate M (Rule (DefinedYes)). Since terms in conditions of find do not contain insert nor event (Invariant 4), the table contents and the sequence of events are left unchanged by the evaluation of the condition of find.

$$E, \sigma, {}^{\mu}i, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1} E, \sigma, \sigma(i), \mathcal{T}, \mu\mathcal{E}v \qquad \text{(ReplIndex)}$$

$$\frac{x[a_1, \dots, a_m] \in \mathrm{Dom}(E)}{E, \sigma, {}^{\mu}x[a_1, \dots, a_m], \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1} E, \sigma, E(x[a_1, \dots, a_m]), \mathcal{T}, \mu\mathcal{E}v} \qquad \text{(Var)}$$

$$\frac{f : T_1 \times \ldots \times T_m \to T \qquad \forall j \leq m, a_j \in T_j \qquad f(a_1, \dots, a_m) = a}{E, \sigma, {}^{\mu}f(a_1, \dots, a_m), \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1} E, \sigma, a, \mathcal{T}, \mu\mathcal{E}v} \qquad \text{(Fun)}$$

$$\frac{a \in T \qquad E' = E[x[\sigma(\widetilde{i})] \mapsto a]}{E, \sigma, {}^{\mu}\mathsf{new}\ x[\widetilde{i}] : T; N, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{D_T(a)}_{N(a)} E', \sigma, N, \mathcal{T}, \mu\mathcal{E}v} \qquad \text{(NewT)}$$

$$\frac{a \in T \qquad E' = E[x[\sigma(\widetilde{i})] \mapsto a]}{E, \sigma, {}^{\mu}\mathsf{let}\ x[\widetilde{i}] : T = a\ \mathsf{in}\ N, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1} E', \sigma, N, \mathcal{T}, \mu\mathcal{E}v} \qquad \text{(LetT)}$$

$$E, \sigma, {}^{\mu}\mathsf{if}\ \mathrm{true}\ \mathsf{then}\ N\ \mathsf{else}\ N', \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1} E, \sigma, N, \mathcal{T}, \mu\mathcal{E}v \qquad \text{(IfT1)}$$

$$\frac{a \neq \mathrm{true}}{E, \sigma, {}^{\mu}\mathsf{if}\ a\ \mathsf{then}\ N\ \mathsf{else}\ N', \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1} E, \sigma, N', \mathcal{T}, \mu\mathcal{E}v} \qquad \text{(IfT2)}$$

$$\frac{\begin{array}{c}(v_k)_{1 \leq k \leq l} \text{ is the sequence of } (j, a_1, \dots, a_{m_j}) \text{ for } a_1 \in [1, n_{j1}], \dots, a_{m_j} \in [1, n_{jm_j}]\\ \text{ordered in increasing lexicographic order}\\ \forall k \in [1, l], E, \sigma[i_{j1} \mapsto a_1, \dots, i_{jm_j} \mapsto a_{m_j}], D_j \wedge M_j, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p_k}{}^{*}_{t_k} E_k, \sigma_k, r_k, \mathcal{T}, \mu\mathcal{E}v\\ \text{where } v_k = (j, a_1, \dots, a_{m_j}) \text{ and } r_k \text{ is a value or } \mathsf{event\_abort}\ (\mu', \widetilde{a}) : e\\ S = \{k \mid \exists(\mu', \widetilde{a}, e), r_k = \mathsf{event\_abort}\ (\mu', \widetilde{a}) : e\} \qquad k_0 \in S\end{array}}{\begin{array}{c}E, \sigma, {}^{\mu}\mathsf{find}[\mathit{unique}?]\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}\ \mathsf{suchthat}\\ D_j \wedge M_j\ \mathsf{then}\ N_j)\ \mathsf{else}\ N, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p_1 \dots p_l D_{\mathsf{find}}(S)(k_0)}_{t_1 \dots t_l FE(k_0)} E_{k_0}, \sigma_{k_0}, r_{k_0}, \mathcal{T}, \mu\mathcal{E}v\end{array}}$$

(FindTE)

$$\frac{\begin{array}{c}(v_k)_{1 \leq k \leq l} \text{ is the sequence of } (j, a_1, \dots, a_{m_j}) \text{ for } a_1 \in [1, n_{j1}], \dots, a_{m_j} \in [1, n_{jm_j}]\\ \text{ordered in increasing lexicographic order}\\ \forall k \in [1, l], E, \sigma[i_{j1} \mapsto a_1, \dots, i_{jm_j} \mapsto a_{m_j}], D_j \wedge M_j, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p_k}{}^{*}_{t_k} E'', \sigma', r_k, \mathcal{T}, \mu\mathcal{E}v\\ \text{where } v_k = (j, a_1, \dots, a_{m_j}) \text{ and } r_k \text{ is a value}\\ S = \{v_k \mid r_k = \mathrm{true}\} \qquad |S| = 1 \text{ or } [\mathit{unique}?] \text{ is empty}\\ v_0 = (j', a'_1, \dots, a'_{m_{j'}}) \in S \qquad E' = E[u_{j'1}[\sigma(\widetilde{i})] \mapsto a'_1, \dots, u_{j'm_{j'}}[\sigma(\widetilde{i})] \mapsto a'_{m_{j'}}]\end{array}}{\begin{array}{c}E, \sigma, {}^{\mu}\mathsf{find}[\mathit{unique}?]\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}\ \mathsf{suchthat}\\ D_j \wedge M_j\ \mathsf{then}\ N_j)\ \mathsf{else}\ N, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p_1 \dots p_l D_{\mathsf{find}}(S)(v_0)}_{t_1 \dots t_l F1(v_0)} E', \sigma, N_{j'}, \mathcal{T}, \mu\mathcal{E}v\end{array}}$$

(FindT1)

$$\frac{\text{First four lines as in (FindT1)} \qquad S = \{v_k \mid r_k = \mathrm{true}\} = \emptyset}{\begin{array}{c}E, \sigma, {}^{\mu}\mathsf{find}[\mathit{unique}?]\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}\ \mathsf{suchthat}\\ D_j \wedge M_j\ \mathsf{then}\ N_j)\ \mathsf{else}\ N, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p_1 \dots p_l}_{t_1 \dots t_l F2} E, \sigma, N, \mathcal{T}, \mu\mathcal{E}v\end{array}} \qquad \text{(FindT2)}$$

$$\frac{\text{First four lines as in (FindT1)} \qquad S = \{v_k \mid r_k = \mathrm{true}\} \qquad |S| > 1}{\begin{array}{c}E, \sigma, {}^{\mu}\mathsf{find}[\mathsf{unique}_e]\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}\ \mathsf{suchthat}\\ D_j \wedge M_j\ \mathsf{then}\ N_j)\ \mathsf{else}\ N, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p_1 \dots p_l}_{t_1 \dots t_l F3} E, \sigma, \mathsf{event\_abort}\ (\mu, \mathrm{Im}(\sigma)) : e, \mathcal{T}, \mu\mathcal{E}v\end{array}}$$

(FindT3)

Figure 4: Semantics (1): terms, first part

$$E, \sigma, {}^{\mu}\mathsf{insert}\ \mathit{Tbl}(a_1, \dots, a_l); N, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1} E, \sigma, N, (\mathcal{T}, \mathit{Tbl}(a_1, \dots, a_l)), \mu\mathcal{E}v \qquad \text{(InsertT)}$$

$$\frac{\begin{array}{c}[v_1, \dots, v_m] = [x \in \mathcal{T} \mid \exists a_1, \dots, \exists a_l, x = \mathit{Tbl}(a_1, \dots, a_l)] \\ \forall k \in [1, m], E[x_1[\sigma(\widetilde{i})] \mapsto a_1, \dots, x_l[\sigma(\widetilde{i})] \mapsto a_l], \sigma, M, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p_k}{}^{*}_{t_k} E_k, \sigma_k, r_k, \mathcal{T}, \mu\mathcal{E}v \\ \text{where } v_k = \mathit{Tbl}(a_1, \dots, a_l) \text{ and } r_k \text{ is a value or } \mathsf{event\_abort}\ (\mu', \widetilde{a}) : e \\ S = \{k \in [1, m] \mid \exists (\mu', \widetilde{a}, e), r_k = \mathsf{event\_abort}\ (\mu', \widetilde{a}) : e\} \qquad k_0 \in S\end{array}}{\begin{array}{c}E, \sigma, {}^{\mu}\mathsf{get}[\mathit{unique}?]\ \mathit{Tbl}(x_1[\widetilde{i}] : T_1, \dots, x_l[\widetilde{i}] : T_l)\ \mathsf{suchthat}\ M\ \mathsf{in}\ N\ \mathsf{else}\ N', \mathcal{T}, \mu\mathcal{E}v \\ \xrightarrow{p_1 \dots p_m D_{\mathsf{get}}(S)(k_0)}{}_{t_1 \dots t_m GE(k_0)} E_{k_0}, \sigma_{k_0}, r_{k_0}, \mathcal{T}, \mu\mathcal{E}v\end{array}} \qquad \text{(GetTE)}$$

$$\frac{\begin{array}{c}[v_1, \dots, v_m] = [x \in \mathcal{T} \mid \exists a_1, \dots, \exists a_l, x = \mathit{Tbl}(a_1, \dots, a_l)] \\ \forall k \in [1, m], E[x_1[\sigma(\widetilde{i})] \mapsto a_1, \dots, x_l[\sigma(\widetilde{i})] \mapsto a_l], \sigma, M, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p_k}{}^{*}_{t_k} E'', \sigma, r_k, \mathcal{T}, \mu\mathcal{E}v \\ \text{where } v_k = \mathit{Tbl}(a_1, \dots, a_l) \text{ and } r_k \text{ is a value} \\ S = \{k \in [1, m] \mid r_k = \text{true}\} \\ |S| = 1 \text{ or } [\mathit{unique}?] \text{ is empty} \\ k_0 \in S \qquad \mathit{Tbl}(a_1, \dots, a_l) = v_{k_0} \qquad E' = E[x_1[\sigma(\widetilde{i})] \mapsto a_1, \dots, x_l[\sigma(\widetilde{i})] \mapsto a_l]\end{array}}{\begin{array}{c}E, \sigma, {}^{\mu}\mathsf{get}[\mathit{unique}?]\ \mathit{Tbl}(x_1[\widetilde{i}] : T_1, \dots, x_l[\widetilde{i}] : T_l)\ \mathsf{suchthat}\ M\ \mathsf{in}\ N\ \mathsf{else}\ N', \mathcal{T}, \mu\mathcal{E}v \\ \xrightarrow{p_1 \dots p_m D_{\mathsf{get}}(S)(k_0)}{}_{t_1 \dots t_m G1(k_0)} E', \sigma, N, \mathcal{T}, \mu\mathcal{E}v\end{array}} \qquad \text{(GetT1)}$$

$$\frac{\text{First four lines as in (GetT1)} \qquad S = \emptyset}{\begin{array}{c}E, \sigma, {}^{\mu}\mathsf{get}[\mathit{unique}?]\ \mathit{Tbl}(x_1[\widetilde{i}] : T_1, \dots, x_l[\widetilde{i}] : T_l)\ \mathsf{suchthat}\ M\ \mathsf{in}\ N\ \mathsf{else}\ N', \mathcal{T}, \mu\mathcal{E}v \\ \xrightarrow{p_1 \dots p_m}{}_{t_1 \dots t_m G2} E, \sigma, N', \mathcal{T}, \mu\mathcal{E}v\end{array}} \qquad \text{(GetT2)}$$

$$\frac{\text{First four lines as in (GetT1)} \qquad |S| > 1}{\begin{array}{c}E, \sigma, {}^{\mu}\mathsf{get}[\mathsf{unique}_e]\ \mathit{Tbl}(x_1[\widetilde{i}] : T_1, \dots, x_l[\widetilde{i}] : T_l)\ \mathsf{suchthat}\ M\ \mathsf{in}\ N\ \mathsf{else}\ N', \mathcal{T}, \mu\mathcal{E}v \\ \xrightarrow{p_1 \dots p_m}{}_{t_1 \dots t_m G3} E, \sigma, \mathsf{event\_abort}\ (\mu, \mathrm{Im}(\sigma)) : e, \mathcal{T}, \mu\mathcal{E}v\end{array}} \qquad \text{(GetT3)}$$

$$E, \sigma, {}^{\mu}\mathsf{event}\ e(a_1, \dots, a_l); N, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1} E, \sigma, N, \mathcal{T}, (\mu\mathcal{E}v, (\mu, \mathrm{Im}(\sigma)) : e(a_1, \dots, a_l)) \qquad \text{(EventT)}$$

$$E, \sigma, {}^{\mu}\mathsf{event\_abort}\ e, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1} E, \sigma, \mathsf{event\_abort}\ (\mu, \mathrm{Im}(\sigma)) : e, \mathcal{T}, \mu\mathcal{E}v \qquad \text{(EventAbortT)}$$

$$\frac{E, \sigma, N, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', \sigma', N', \mathcal{T}', \mu\mathcal{E}v'}{E, \sigma, C[N], \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', \sigma', C[N'], \mathcal{T}', \mu\mathcal{E}v'} \qquad \text{(CtxT)}$$

$$E, \sigma, C[\mathsf{event\_abort}\ (\mu, \widetilde{a}) : e], \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1} E, \sigma, \mathsf{event\_abort}\ (\mu, \widetilde{a}) : e, \mathcal{T}, \mu\mathcal{E}v \qquad \text{(CtxEventT)}$$

$$\frac{\neg \forall j \leq l, \exists a_j, E, \sigma, M_j, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1}{}^{*} E, \sigma, a_j, \mathcal{T}, \mu\mathcal{E}v}{E, \sigma, \mathsf{defined}(M_1, \dots, M_l) \wedge M, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1} E, \sigma, \text{false}, \mathcal{T}, \mu\mathcal{E}v} \qquad \text{(DefinedNo)}$$

$$\frac{\forall j \leq l, \exists a_j, E, \sigma, M_j, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1}{}^{*} E, \sigma, a_j, \mathcal{T}, \mu\mathcal{E}v}{E, \sigma, \mathsf{defined}(M_1, \dots, M_l) \wedge M, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1} E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v} \qquad \text{(DefinedYes)}$$

Figure 5: Semantics (2): terms, second part, and defined conditions

$$
\begin{aligned}
C ::= {} & {}^{\mu}x[a_1, \dots, a_{k-1}, [\,], M_{k+1}, \dots, M_m] \\
& {}^{\mu}f(a_1, \dots, a_{k-1}, [\,], M_{k+1}, \dots, M_m) \\
& {}^{\mu}\mathsf{let}\ x[\widetilde{i}] : T = [\,]\ \mathsf{in}\ N \\
& {}^{\mu}\mathsf{if}\ [\,]\ \mathsf{then}\ N\ \mathsf{else}\ N' \\
& {}^{\mu}\mathsf{event}\ e(a_1, \dots, a_{k-1}, [\,], M_{k+1}, \dots, M_l); N \\
& {}^{\mu}\mathsf{insert}\ \mathit{Tbl}(a_1, \dots, a_{k-1}, [\,], M_{k+1}, \dots, M_l); N
\end{aligned}
$$

Figure 6: Term contexts

Rule (InsertT) inserts the new table element in $\mathcal{T}$. Rules (GetTE) to (GetT3) define the semantics of get. We denote by $[x \in L \mid f(x)]$ the list of all elements x of the list L that satisfy $f(x)$, in the same order as in L. We denote by $|L|$ the length of list L. We denote by $\mathsf{nth}(L, j)$ the j-th element of the list L. Rule (GetTE) executes event_abort e when the evaluation of the condition M executes event_abort e for some element of the table Tbl; when several events may be executed, one of them is chosen randomly according to distribution $D_{\mathsf{get}}(S)$, that is, almost uniformly in the elements of the table Tbl. Rules (GetT1), (GetT2), and (GetT3) compute the set S of elements of indices of elements of table Tbl in $\mathcal{T}$ that satisfy condition M. If S is empty, we execute N' (Rule (GetT2)). When S is not empty, two cases can happen. If the get is not marked $[\mathsf{unique}_e]$ or S has a single element, then one of its elements is chosen randomly according to distribution $D_{\mathsf{get}}(S)$, we store this element in $x_1[\sigma(\widetilde{i})], \dots, x_l[\sigma(\widetilde{i})]$ by extending the environment E, and continue by executing N (Rule (GetT1)). If the get is marked $[\mathsf{unique}_e]$ and S contains several elements, then we execute event e and abort (Rule (GetT3)). Since terms in conditions of get do not contain insert nor event (Invariant 4), the table contents and the sequence of events are left unchanged by the evaluation of M. The modified environment E'' obtained after evaluating a condition M can be ignored because there are no array accesses to the variables defined in conditions of get, by Invariant 3, so the values of these variables are not used after the evaluation of the condition.

Remark 1 Another way of defining the semantics of tables would be to consider two distinct calculi, one with tables (used for the initial game), and one without tables (used for the other games). The semantics of the calculus without tables can be defined without the component $\mathcal{T}$. We then need to relate the two semantics.

Rule (EventT) adds the executed event to $\mu\mathcal{E}v$. Rule (EventAbortT) executes event_abort e. The event e is not immediately added to $\mu\mathcal{E}v$, because for terms that occur in conditions of find, in case several branches execute event_abort, we may need to choose randomly which event will be added to $\mu\mathcal{E}v$. Hence, we use the result event_abort $(\mu, \widetilde{a}) : e$ instead.

Rules (CtxT) and (CtxEventT) allow evaluating terms under a context. In these rules, C is an elementary context, of one of the forms defined in Figure 6. When the term N reduces to some other term N', Rule (CtxT) allows one to reduce it in the same way under a context C. When the term N is an event, $C[N]$ also executes the same event by Rule (CtxEventT).

These rules define a small-step semantics for terms. We consider the reflexive and transitive closure $\xrightarrow{p}{}_t^{*}$ of the relation $\xrightarrow{p}{}_t$ to reach directly the normal form of the term, which can be either a value a or an abort event value event_abort $(\mu, \widetilde{a}) : e$. We have $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1}{}^{*} E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v$ and, if $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}{}_t E', \sigma', M', \mathcal{T}', \mu\mathcal{E}v'$ and $E', \sigma', M', \mathcal{T}', \mu\mathcal{E}v' \xrightarrow{p'}{}_{t'}^{*} E'', \sigma'', M'', \mathcal{T}'', \mu\mathcal{E}v''$,

$$E, \{(\sigma, {}^{\mu}0)\} \uplus \mathcal{Q}, \mathcal{C}h \rightsquigarrow E, \mathcal{Q}, \mathcal{C}h \tag{Nil}$$

$$E, \{(\sigma, {}^{\mu}(Q_1 \mid Q_2))\} \uplus \mathcal{Q}, \mathcal{C}h \rightsquigarrow E, \{(\sigma, Q_1), (\sigma, Q_2)\} \uplus \mathcal{Q}, \mathcal{C}h \tag{Par}$$

$$E, \{(\sigma, {}^{\mu}!^{i \le n} Q)\} \uplus \mathcal{Q}, \mathcal{C}h \rightsquigarrow E, \{(\sigma[i \mapsto a], Q) \mid a \in [1, n]\} \uplus \mathcal{Q}, \mathcal{C}h \tag{Repl}$$

$$\frac{c' \notin \mathcal{C}h}{E, \{(\sigma, {}^{\mu}\mathsf{newChannel}\ c; Q)\} \uplus \mathcal{Q}, \mathcal{C}h \rightsquigarrow E, \{(\sigma, Q\{c'/c\})\} \uplus \mathcal{Q}, \mathcal{C}h \cup \{c'\}} \tag{NewChannel}$$

$$\frac{E, \sigma, M_j, \emptyset, \emptyset \xrightarrow{1} E, \sigma, M'_j, \emptyset, \emptyset}{E, \{(\sigma, C[M_j])\} \uplus \mathcal{Q}, \mathcal{C}h \rightsquigarrow E, \{(\sigma, C[M'_j])\} \uplus \mathcal{Q}, \mathcal{C}h} \tag{Input}$$

$$\text{where } C = {}^{\mu}c[a_1, \dots, a_{j-1}, [\,], M_{j+1}, \dots, M_l](x[\widetilde{i}] : T); P$$

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

Figure 7: Semantics (3): input processes

then $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p \times p'}{}^{*}_{t,t'} E'', \sigma'', M'', \mathcal{T}'', \mu\mathcal{E}v''$: we take the product of the probabilities to have the probability of a sequence of reductions, and we specify which sequence was taken by a list of indices t, t'.

Figure 7 defines the semantics of input processes. We use an auxiliary reduction relation $\rightsquigarrow$, for reducing input processes. This relation transforms configurations of the form $E, \mathcal{Q}, \mathcal{C}h$. 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 $\mathcal{C}h$. Semantic configurations are considered equivalent modulo renaming of channels in $\mathcal{C}h$, 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. In the (Input) rule, the terms $M_1, \dots, M_l$ are simple by Invariant 5, so their evaluation is deterministic (the unique result is obtained with probability 1), the environment E, the contents of tables $\mathcal{T}$, and the sequence of events $\mu\mathcal{E}v$ are unchanged, and $\mathcal{T}$ and $\mu\mathcal{E}v$ are unused, that is why we can write $E, \sigma, M_j, \emptyset, \emptyset \xrightarrow{1} E, \sigma, M'_j, \emptyset, \emptyset$ using empty $\mathcal{T}$ and $\mu\mathcal{E}v$, written $\emptyset$. The relation $\rightsquigarrow$ is convergent (confluent and terminating), so it has normal forms. Processes in $\mathcal{Q}$ in configurations $E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ are always in normal form by $\rightsquigarrow$, so they always start with an input.

Finally, Figures 8 and 9 define the semantics of output processes. Most of these rules are very similar to those for terms: they just use processes instead of terms as continuations, and include a whole semantic configuration. Rule (EventAbort) executes event e and aborts the game, by reducing to the configuration with process abort. Similarly to the case of terms, Rules (Ctx) and (CtxEvent) allow evaluating terms under a context inside a process. In these rules, C is an elementary context, of one of the forms defined in Figure 10.

Rule (Output) performs communications: it selects an input on the desired channel randomly, and immediately executes the communication. (The process blocks if no suitable input is available.) The scheduled process after this rule is the receiving process. The input processes that follow the output are stored in the available input processes, after reducing them by rules of Figure 7. In this rule, S is a multiset. When we take probabilities over multisets, we consider that $D_{\mathsf{in}}(S)(\sigma', Q_0)$ is the probability of choosing *one* of the elements equal to σ', Q_0 in S according to the distribution $D_{\mathsf{in}}(S)$, so that the probability of choosing any element equal to (σ', Q_0) is in fact $S(\sigma', Q_0) \times D_{\mathsf{in}}(S)(\sigma', Q_0)$.

After finishing execution of a process, the system produces the sequence of executed events

$$\frac{\text{Same assumption as in (FindTE)} \qquad r_{k_0} = \mathsf{event\_abort}\ (\mu', \widetilde{a}) : e}{\begin{array}{c}E, (\sigma, {}^{\mu}\mathsf{find}[\mathit{unique}?]\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}\ \mathsf{suchthat}\\ D_j \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v\\ \xrightarrow{p_1 \dots p_{l_0} D_{\mathsf{find}}(S)(k_0)}_{t_1 \dots t_{l_0} FE(k_0)} E_{k_0}, (\sigma_{k_0}, \mathsf{abort}), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, (\mu\mathcal{E}v, (\mu', \widetilde{a}) : e)\end{array}} \quad \text{(FindE)}$$

$$\frac{\begin{array}{c}\text{First four lines as in (FindT1)} \qquad S = \{v_k \mid r_k = \text{true}\} \qquad |S| = 1 \text{ or } [\mathit{unique}?] \text{ is empty}\\ v_0 = (j', a'_1, \dots, a'_{m_{j'}}) \in S \qquad E' = E[u_{j'1}[\sigma(\widetilde{i})] \mapsto a'_1, \dots, u_{j'm_{j'}}[\sigma(\widetilde{i})] \mapsto a'_{m_{j'}}]\end{array}}{\begin{array}{c}E, (\sigma, {}^{\mu}\mathsf{find}[\mathit{unique}?]\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}\ \mathsf{suchthat}\\ D_j \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p_1 \dots p_l D_{\mathsf{find}}(S)(v_0)}_{t_1 \dots t_l F1(v_0)} E', (\sigma, P_{j'}), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v\end{array}}$$
(Find1)

$$\frac{\text{First four lines as in (FindT1)} \qquad S = \{v_k \mid r_k = \text{true}\} = \emptyset}{\begin{array}{c}E, (\sigma, {}^{\mu}\mathsf{find}[\mathit{unique}?]\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}\ \mathsf{suchthat}\\ D_j \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p_1 \dots p_l}_{t_1 \dots t_l F2} E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v\end{array}} \quad \text{(Find2)}$$

$$\frac{\text{First four lines as in (FindT1)} \qquad S = \{v_k \mid r_k = \text{true}\} \qquad |S| > 1}{\begin{array}{c}E, (\sigma, {}^{\mu}\mathsf{find}[\mathsf{unique}_e]\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}\ \mathsf{suchthat} D_j \wedge\\ M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p_1 \dots p_l}_{t_1 \dots t_l F3} E, (\sigma, \mathsf{abort}), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, (\mu\mathcal{E}v, (\mu, \mathrm{Im}(\sigma)) : e)\end{array}}$$
(Find3)

$$E, (\sigma, {}^{\mu}\mathsf{insert}\ \mathit{Tbl}(a_1, \dots, a_l); P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1} E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, (\mathcal{T}, \mathit{Tbl}(a_1, \dots, a_l)), \mu\mathcal{E}v$$
(Insert)

$$\frac{\text{Same assumption as in (GetTE)} \qquad r_{k_0} = \mathsf{event\_abort}\ (\mu', \widetilde{a}) : e}{\begin{array}{c}E, (\sigma, {}^{\mu}\mathsf{get}[\mathit{unique}?]\ \mathit{Tbl}(x_1[\widetilde{i}] : T_1, \dots, x_l[\widetilde{i}] : T_l)\ \mathsf{suchthat}\ M\ \mathsf{in}\ P\ \mathsf{else}\ P'), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v\\ \xrightarrow{p_1 \dots p_m D_{\mathsf{get}}(S)(k_0)}_{t_1 \dots t_m GE(k_0)} E_{k_0}, (\sigma_{k_0}, \mathsf{abort}), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, (\mu\mathcal{E}v, (\mu', \widetilde{a}) : e)\end{array}}$$
(GetE)

$$\frac{\begin{array}{c}\text{First four lines as in (GetT1)} \qquad |S| = 1 \text{ or } [\mathit{unique}?] \text{ is empty}\\ k_0 \in S \qquad \mathit{Tbl}(a_1, \dots, a_l) = v_{k_0} \qquad E' = E[x_1[\sigma(\widetilde{i})] \mapsto a_1, \dots, x_l[\sigma(\widetilde{i})] \mapsto a_l]\end{array}}{\begin{array}{c}E, (\sigma, {}^{\mu}\mathsf{get}[\mathit{unique}?]\ \mathit{Tbl}(x_1[\widetilde{i}] : T_1, \dots, x_l[\widetilde{i}] : T_l)\ \mathsf{suchthat}\ M\ \mathsf{in}\ P\ \mathsf{else}\ P'), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v\\ \xrightarrow{p_1 \dots p_m D_{\mathsf{get}}(S)(k_0)}_{t_1 \dots t_m G1(k_0)} E', (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v\end{array}}$$
(Get1)

$$\frac{\text{First four lines as in (GetT1)} \qquad S = \emptyset}{\begin{array}{c}E, (\sigma, {}^{\mu}\mathsf{get}[\mathit{unique}?]\ \mathit{Tbl}(x_1[\widetilde{i}] : T_1, \dots, x_l[\widetilde{i}] : T_l)\ \mathsf{suchthat}\ M\ \mathsf{in}\ P\ \mathsf{else}\ P'), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v\\ \xrightarrow{p_1 \dots p_m}_{t_1 \dots t_m G2} E, (\sigma, P'), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v\end{array}}$$
(Get2)

$$\frac{\text{First four lines as in (GetT1)} \qquad |S| > 1}{\begin{array}{c}E, (\sigma, {}^{\mu}\mathsf{get}[\mathsf{unique}_e]\ \mathit{Tbl}(x_1[\widetilde{i}] : T_1, \dots, x_l[\widetilde{i}] : T_l)\ \mathsf{suchthat}\ M\ \mathsf{in}\ P\ \mathsf{else}\ P'), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v\\ \xrightarrow{p_1 \dots p_m}_{t_1 \dots t_m G3} E, (\sigma, \mathsf{abort}), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, (\mu\mathcal{E}v, (\mu, \mathrm{Im}(\sigma)) : e)\end{array}}$$
(Get3)

Figure 8: Semantics (4): output processes, first part

$$\frac{a \in T \qquad E' = E[x[\sigma(\widetilde{i})] \mapsto a]}{E, (\sigma, {}^{\mu}\mathsf{new}\ x[\widetilde{i}] : T; P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{D_T(a)}_{N(a)} E', (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v} \tag{New}$$

$$\frac{a \in T \qquad E' = E[x[\sigma(\widetilde{i})] \mapsto a]}{E, (\sigma, {}^{\mu}\mathsf{let}\ x[\widetilde{i}] : T = a\ \mathsf{in}\ P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1} E', (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v} \tag{Let}$$

$$E, (\sigma, {}^{\mu}\mathsf{if}\ \mathrm{true}\ \mathsf{then}\ P\ \mathsf{else}\ P'), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1} E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v \tag{If1}$$

$$\frac{a \neq \mathrm{true}}{E, (\sigma, {}^{\mu}\mathsf{if}\ a\ \mathsf{then}\ P\ \mathsf{else}\ P'), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1} E, (\sigma, P'), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v} \tag{If2}$$

$$\begin{aligned}&E, (\sigma, {}^{\mu}\mathsf{event}\ e(a_1, \dots, a_l); P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1} \\ &\qquad E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, (\mu\mathcal{E}v, (\mu, \mathrm{Im}(\sigma)) : e(a_1, \dots, a_l))\end{aligned} \tag{Event}$$

$$E, (\sigma, {}^{\mu}\mathsf{event\_abort}\ e), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1} E, (\sigma, \mathsf{abort}), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, (\mu\mathcal{E}v, (\mu, \mathrm{Im}(\sigma)) : e) \tag{EventAbort}$$

$$\frac{E, \sigma, N, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', \sigma', N', \mathcal{T}', \mu\mathcal{E}v'}{E, (\sigma, C[N]), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', (\sigma', C[N']), \mathcal{Q}, \mathcal{C}h, \mathcal{T}', \mu\mathcal{E}v'} \tag{Ctx}$$

$$E, (\sigma, C[\mathsf{event\_abort}\ (\mu, \widetilde{a}) : e]), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1} E, (\sigma, \mathsf{abort}), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, (\mu\mathcal{E}v, (\mu, \widetilde{a}) : e) \tag{CtxEvent}$$

$$\frac{\begin{gathered}E, \mathcal{Q}', \mathcal{C}h' = \mathrm{reduce}(E, \{(\sigma, Q'')\}, \mathcal{C}h)\\ S = \{(\sigma', Q) \in \mathcal{Q} \mid Q = {}^{\mu''}c[a_1, \dots, a_l](x'[\widetilde{i}] : T').P' \text{ and } b \in T' \text{ for some } \mu'', \sigma', x', T', P'\}\\ (\sigma', Q_0) \in S \qquad Q_0 = {}^{\mu'}c[a_1, \dots, a_l](x[\widetilde{i}] : T).P\end{gathered}}{\begin{gathered}E, (\sigma, {}^{\mu}\overline{c[a_1, \dots, a_l]}\langle b\rangle.Q''), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{S(\sigma', Q_0) \times D_{\mathsf{in}}(S)(\sigma', Q_0)}_{O(\sigma', Q_0)}\\ E[x[\sigma'(\widetilde{i})] \mapsto b], (\sigma', P), \mathcal{Q} \uplus \mathcal{Q}' \setminus \{(\sigma', Q_0)\}, \mathcal{C}h', \mathcal{T}, \mu\mathcal{E}v\end{gathered}} \tag{Output}$$

Figure 9: Semantics (5): output processes, second part

$\mathcal{E}v$. These events can be used to distinguish games, so we introduce an additional algorithm, a *distinguisher* D that takes as input a sequence of events $\mathcal{E}v$ (without program points and replication indices) and returns true or false.

An example of distinguisher is D_e defined by $D_e(\mathcal{E}v) = \mathrm{true}$ if and only if $e \in \mathcal{E}v$: this distinguisher detects the execution of event e. We will denote the distinguisher D_e simply by e. More generally, distinguishers can detect various properties of the sequence of events $\mathcal{E}v$ executed by the game and of its result a. We denote by $D \vee D'$, $D \wedge D'$, and $\neg D$ the distinguishers such that $(D \vee D')(\mathcal{E}v) = D(\mathcal{E}v) \vee D'(\mathcal{E}v)$, $(D \wedge D')(\mathcal{E}v) = D(\mathcal{E}v) \wedge D'(\mathcal{E}v)$, and $(\neg D)(\mathcal{E}v) = \neg D(\mathcal{E}v)$. We denote by $\Pr[Q : D]$ the probability that Q executes a sequence of events $\mathcal{E}v$ such that $D(\mathcal{E}v) = \mathrm{true}$. This is formally defined as follows.

Definition 4 The initial configuration for running process Q is $\mathrm{initConfig}(Q) = \emptyset, (\sigma_0, {}^{\mu}\overline{\mathit{start}}\langle\rangle), \mathcal{Q}, \mathcal{C}h, \emptyset, \emptyset$ where $\emptyset, \mathcal{Q}, \mathcal{C}h = \mathrm{reduce}(\emptyset, \{(\sigma_0, Q)\}, \mathrm{fc}(Q))$ and σ_0 is the empty mapping sequence.

A *trace of* Q is a trace that starts from $\mathrm{initConfig}(Q)$: $\mathcal{T}r = \mathrm{initConfig}(Q) \xrightarrow{p_1}_{t_1} \dots \xrightarrow{p_{m-1}}_{t_{m-1}} \mathit{Conf}_m$. Let $\mathcal{T}r$ be the set of all traces of Q.

Let $\mathcal{T}r_{\mathrm{full}}$ be the set of *full traces* of Q, that is, the set of traces of $\mathcal{T}r$ whose last configuration

$$\begin{aligned} C ::= {} & {}^{\mu}\mathsf{let}\ x[\widetilde{i}] : T = [\,]\ \mathsf{in}\ P \\ & {}^{\mu}\mathsf{if}\ [\,]\ \mathsf{then}\ P\ \mathsf{else}\ P' \\ & {}^{\mu}\overline{c[a_1, \dots, a_{k-1}, [\,], M_{k+1}, \dots, M_l]}\langle N \rangle.Q \\ & {}^{\mu}\overline{c[a_1, \dots, a_l]}\langle [\,] \rangle.Q \\ & {}^{\mu}\mathsf{insert}\ \mathit{Tbl}(a_1, \dots, a_{k-1}, [\,], M_{k+1}, \dots, M_l); P \\ & {}^{\mu}\mathsf{event}\ e(a_1, \dots, a_{k-1}, [\,], M_{k+1}, \dots, M_l); P \end{aligned}$$

Figure 10: Process contexts

Conf_m cannot be reduced.

A trace Tr' is an *extension* of Tr when Tr' is obtained by continuing execution from the last configuration of Tr. Equivalently, Tr is a *prefix* of Tr'.

Let φ be a property of traces, that is, a function from traces to $\{\text{true}, \text{false}\}$. We say that Tr satisfies φ, and we write $\mathit{Tr} \vdash \varphi$, when $\varphi(\mathit{Tr}) = \text{true}$.

A property φ is *preserved by extension* when for all traces Tr such that $\mathit{Tr} \vdash \varphi$, for all extensions Tr' of Tr, $\mathit{Tr}' \vdash \varphi$.

Given a trace $\mathit{Tr} = \text{initConfig}(Q) \xrightarrow{p_1}_{t_1} \dots \xrightarrow{p_{m-1}}_{t_{m-1}} \mathit{Conf}_m$, recall that $\Pr[\mathit{Tr}] = p_1 \times \dots \times p_{m-1}$. We define

$$\begin{aligned} \Pr[Q : \varphi] &= \sum_{\mathit{Tr} \in \mathcal{Tr}_{\text{full}},\, \mathit{Tr} \vdash \varphi} \Pr[\mathit{Tr}]\,, \\ \Pr[Q \preceq \varphi] &= \sum_{\mathit{Tr} \in \mathcal{Tr}_{\text{full}}, \exists \mathit{Tr}' \text{ prefix of } \mathit{Tr}, \mathit{Tr}' \vdash \varphi} \Pr[\mathit{Tr}] \\ &= \sum_{\mathit{Tr} \in \mathcal{Tr}, \mathit{Tr} \vdash \varphi, \text{for any strict prefix } \mathit{Tr}' \text{ of } \mathit{Tr}, \mathit{Tr}' \not\vdash \varphi} \Pr[\mathit{Tr}]\,. \end{aligned}$$

Given a distinguisher D, we can consider it as a property of traces by defining $\mathit{Tr} \vdash D$ if and only if $D(\text{removepp}(\mu\mathcal{E}v_m)) = \text{true}$ where the last configuration of Tr is $\mathit{Conf}_m = E_m, (\sigma_m, P_m), \mathcal{Q}_m, \mathcal{C}h_m, \mathcal{T}_m, \mu\mathcal{E}v_m$. The function removepp guarantees that the pair (program point, replication indices) is not used in the evaluation of the distinguisher. Actually, this pair could be removed from the semantics. It is useful for the proof of injective correspondences (Section 4.2.4).

Lemma 1 *1.* $\Pr[Q : \varphi] \leq \Pr[Q \preceq \varphi]$.

2. If φ is preserved by extension, then $\Pr[Q : \varphi] = \Pr[Q \preceq \varphi]$.

Proof The first property holds because, when $\mathit{Tr} \vdash \varphi$, there exists a prefix Tr' of Tr (take $\mathit{Tr}' = \mathit{Tr}$) such that $\mathit{Tr}' \vdash \varphi$.

The second property holds because, if φ is preserved by extension and there exists a prefix Tr' of Tr such that $\mathit{Tr}' \vdash \varphi$, then $\mathit{Tr} \vdash \varphi$. □

For simple terms M, the evaluation can be defined without tables and events. We define $E, \rho, M \Downarrow a$ if and only if $E, \rho, M, \emptyset, \emptyset \xrightarrow{1}{}^{*} E, \rho, a, \emptyset, \emptyset$, where the environment E gives the values of process variables (in particular arrays), and the environment ρ gives the values of replication indices and other variables (e.g., those used in correspondences, see Section 2.7.3). The evaluation relation $E, \rho, M \Downarrow a$ can also be defined by induction as follows:

$$E, \rho, i \Downarrow \rho(i)$$

$$\frac{\forall j \leq m, E, \rho, M_j \Downarrow a_j}{E, \rho, x[M_1, \dots, M_m] \Downarrow E(x[a_1, \dots, a_m])}$$

$$\frac{\forall j \leq m, E, \rho, M_j \Downarrow a_j}{E, \rho, f(M_1, \dots, M_m) \Downarrow f(a_1, \dots, a_m)}$$

For terms that do not contain process variables, the environment E can be omitted, and we write $\rho, M \Downarrow a$. It can also be defined by induction as follows:

$$\rho, i \Downarrow \rho(i)$$

$$\frac{\forall j \leq m, \rho, M_j \Downarrow a_j}{\rho, f(M_1, \dots, M_m) \Downarrow f(a_1, \dots, a_m)}$$

2.4.2 Properties

Given a process Q_0, we write I_μ for the current replication indices at μ in Q_0.

Lemma 2 *Let Tr be a trace of Q_0. In the derivation of Tr, for all configurations $E, \sigma, {}^\mu M, \mathcal{T}$, $\mu\mathcal{E}v$ or $E, (\sigma, {}^\mu P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$, $\mathrm{Dom}(\sigma) = I_\mu$ is the sequence of current replication indices at μ in Q_0 (or $\mathrm{Dom}(\sigma) = \emptyset$ is the sequence of current replication indices at ${}^\mu\overline{start}\langle\rangle$ in the initial configuration) and for all configurations $E, \mathcal{Q}, \mathcal{C}h$ or $E, (\sigma, {}^\mu P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$, for all $(\sigma', {}^{\mu'} Q) \in \mathcal{Q}$, $\mathrm{Dom}(\sigma') = I_{\mu'}$.*

Proof sketch We say that

- a configuration $E, \sigma, {}^\mu M, \mathcal{T}, \mu\mathcal{E}v$ is ok when $\mathrm{Dom}(\sigma) = I_\mu$;
- a configuration $E, \mathcal{Q}, \mathcal{C}h$ is ok when for all $(\sigma', {}^{\mu'} Q) \in \mathcal{Q}$, $\mathrm{Dom}(\sigma') = I_{\mu'}$;
- a configuration $E, (\sigma, {}^\mu P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ is ok when $\mathrm{Dom}(\sigma) = I_\mu$ (or $\mathrm{Dom}(\sigma) = \emptyset$ is the sequence of current replication indices at ${}^\mu\overline{start}\langle\rangle$ in the initial configuration) and for all $(\sigma', {}^{\mu'} Q) \in \mathcal{Q}$, $\mathrm{Dom}(\sigma') = I_{\mu'}$.

We show by induction on the derivations that

1. if $E_1, \sigma_1, M_1, \mathcal{T}_1, \mu\mathcal{E}v_1$ is ok, then all configurations $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v$ in the derivation of E_1, $\sigma_1, M_1, \mathcal{T}_1, \mu\mathcal{E}v_1 \xrightarrow{p}_t E_2, \sigma_2, M_2, \mathcal{T}_2, \mu\mathcal{E}v_2$ are ok;
2. if $E_1, \mathcal{Q}_1, \mathcal{C}h_1$ is ok, then all configurations $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v$ in the derivation of $E_1, \mathcal{Q}_1$, $\mathcal{C}h_1 \rightsquigarrow E_2, \mathcal{Q}_2, \mathcal{C}h_2$ are ok and $E_2, \mathcal{Q}_2, \mathcal{C}h_2$ is ok;
3. if $E_1, \mathcal{Q}_1, \mathcal{C}h_1$ is ok, then $\mathrm{reduce}(E_1, \mathcal{Q}_1, \mathcal{C}h_1)$ is ok and all configurations $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v$ or $E, \mathcal{Q}, \mathcal{C}h$ and in the derivation of $E_1, \mathcal{Q}_1, \mathcal{C}h_1 \rightsquigarrow^* \mathrm{reduce}(E_1, \mathcal{Q}_1, \mathcal{C}h_1)$ are ok;
4. if $E_1, (\sigma_1, P_1), \mathcal{Q}_1, \mathcal{C}h_1, \mathcal{T}_1, \mu\mathcal{E}v_1$ is ok, then all configurations $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v$ or $E, \mathcal{Q}, \mathcal{C}h$ or $E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ in the derivation of $E_1, (\sigma_1, P_1), \mathcal{Q}_1, \mathcal{C}h_1, \mathcal{T}_1, \mu\mathcal{E}v_1 \xrightarrow{p}_t E_2, (\sigma_2, P_2)$, $\mathcal{Q}_2, \mathcal{C}h_2, \mathcal{T}_2, \mu\mathcal{E}v_2$ are ok.

Indeed, the changes in σ match the definition of replication indices.

For Property 1, in rules for find, the indices of the find condition are added to the domain of σ when evaluating the condition and in all other cases, σ is unchanged.

In the proof of Property 2, in (Repl), the replication index i is added to σ and in all other cases, σ is unchanged. We use Property 1 in the case of input (Input).

Property 3 follows immediately from Property 2 by induction.

For Property 4, in rules for find, the indices of the find condition are added to the domain of σ when evaluating the condition, as in Property 1; in (Output), we use Property 3. In all other cases, σ is unchanged. We use Property 1 when evaluating terms, in conditions of find and get and in (Ctx).

Moreover, in the computation of $\mathrm{initConfig}(Q_0)$, the configuration $\emptyset, \{(\sigma_0, Q_0)\}, \mathrm{fc}(Q)$ is ok (the current replication indices at the root of Q_0 are empty), so by Property 3, $\mathrm{initConfig}(Q_0)$ is ok. $\square$

Lemma 3 *If $E, \sigma, N, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', \sigma', N', \mathcal{T}', \mu\mathcal{E}v'$, then E' is an extension of E, $\mathcal{T}$ is a prefix of $\mathcal{T}'$, $\mu\mathcal{E}v$ is a prefix of $\mu\mathcal{E}v'$, σ' is an extension of σ, and if the term N' is not of the form $C_1[\dots C_k[$event_abort $(\mu, \widetilde{a}) : e]\dots]$ for some $k \in \mathbb{N}$ and $C_1, \dots, C_k$ contexts defined in Figure 6, then $\sigma' = \sigma$. For all configurations $E'', \sigma'', N'', \mathcal{T}'', \mu\mathcal{E}v''$ in the derivation of $E, \sigma, N, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', \sigma', N', \mathcal{T}', \mu\mathcal{E}v'$, we have that E'' is an extension of E and, if $E'', \sigma'', N'', \mathcal{T}'', \mu\mathcal{E}v''$ is not in the derivation of an hypothesis of a rule for* find *or* get*, then E' is an extension of E''; σ'' is an extension of σ; $\mathcal{T}$ is a prefix of $\mathcal{T}''$, which is a prefix of $\mathcal{T}'$; and $\mu\mathcal{E}v$ is a prefix of $\mu\mathcal{E}v''$, which is a prefix of $\mu\mathcal{E}v'$.*

If $E, \mathcal{Q}, \mathcal{C}h \rightsquigarrow E', \mathcal{Q}', \mathcal{C}h'$, then $E' = E$ and $\mathcal{C}h \subseteq \mathcal{C}h'$. For all configurations $E'', \sigma'', N'', \mathcal{T}''$, $\mu\mathcal{E}v''$ in the derivation of $E, \mathcal{Q}, \mathcal{C}h \rightsquigarrow E', \mathcal{Q}', \mathcal{C}h'$, we have $E'' = E$, $\mathcal{T}'' = \emptyset$, and $\mu\mathcal{E}v'' = \emptyset$.

If $E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', (\sigma', P'), \mathcal{Q}', \mathcal{C}h', \mathcal{T}', \mu\mathcal{E}v'$, then E' is an extension of E, $\mathcal{T}$ is a prefix of $\mathcal{T}'$, $\mu\mathcal{E}v$ is a prefix of $\mu\mathcal{E}v'$, and $\mathcal{C}h \subseteq \mathcal{C}h'$.

- *If $E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', (\sigma', P'), \mathcal{Q}', \mathcal{C}h', \mathcal{T}', \mu\mathcal{E}v'$ is derived by* (Output)*, then $\mathcal{T}' = \mathcal{T}$, $\mu\mathcal{E}v' = \mu\mathcal{E}v$, for all configurations $E'', \mathcal{Q}'', \mathcal{C}h''$ in that derivation, $E'' = E$ and $\mathcal{C}h \subseteq \mathcal{C}h'' \subseteq \mathcal{C}h'$, and for all configurations $E'', \sigma'', N'', \mathcal{T}'', \mu\mathcal{E}v''$ in that derivation, $E'' = E$, σ'' is an extension of σ, $\mathcal{T}'' = \emptyset$, and $\mu\mathcal{E}v'' = \emptyset$.*

- *In all other cases, $\mathcal{C}h' = \mathcal{C}h$, σ' is an extension of σ, and if the process P' is not* abort *or of the form $C_0[C_1[\dots C_k[$event_abort $(\mu, \widetilde{a}) : e]\dots]]$ for some C_0 context defined in Figure 10, $k \in \mathbb{N}$, and $C_1, \dots, C_k$ contexts defined in Figure 6, then $\sigma' = \sigma$. For all configurations $E'', \sigma'', N'', \mathcal{T}'', \mu\mathcal{E}v''$ in the derivation of $E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', (\sigma', P'), \mathcal{Q}', \mathcal{C}h'$, $\mathcal{T}', \mu\mathcal{E}v'$, we have that E'' is an extension of E and, if $E'', \sigma'', N'', \mathcal{T}'', \mu\mathcal{E}v''$ is not in the derivation of an hypothesis of a rule for* find *or* get*, then E' is an extension of E''; σ'' is an extension of σ; $\mathcal{T}$ is a prefix of $\mathcal{T}''$, which is a prefix of $\mathcal{T}'$; and $\mu\mathcal{E}v$ is a prefix of $\mu\mathcal{E}v''$, which is a prefix of $\mu\mathcal{E}v'$.*

Proof sketch By induction on the derivation of $E, \sigma, N, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', \sigma', N', \mathcal{T}', \mu\mathcal{E}v'$ and by cases on the reductions $E, \mathcal{Q}, \mathcal{C}h \rightsquigarrow E', \mathcal{Q}', \mathcal{C}h'$ and $E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', (\sigma', P'), \mathcal{Q}'$, $\mathcal{C}h', \mathcal{T}', \mu\mathcal{E}v'$. $\square$

We say that a term N is *in evaluation position* in a configuration $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v$ when $M = C_1[\dots C_k[N]\dots]$ for some $k \in \mathbb{N}$ and $C_1, \dots, C_k$ contexts defined in Figure 6.

An *input context* is a context of the form ${}^{\mu}c[a_1, \dots, a_{j-1}, [\,], M_{j+1}, \dots, M_m](x[\widetilde{i}] : T); P$.

We say that a term N is *in evaluation position* in a configuration $E, \mathcal{Q}, \mathcal{C}h$ when $(\sigma', Q) \in \mathcal{Q}$ and $Q = C_0[C_1[\dots C_k[N]\dots]]$ for some C_0 input context, $k \in \mathbb{N}$ and $C_1, \dots, C_k$ contexts defined in Figure 6.

We say that a term N is *in evaluation position* in a configuration $E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ when $P = C_0[C_1[\dots C_k[N]\dots]]$ for some C_0 context defined in Figure 10, $k \in \mathbb{N}$, and $C_1, \dots, C_k$ contexts defined in Figure 6 or $(\sigma', Q) \in \mathcal{Q}$ and $Q = C_0[C_1[\dots C_k[N]\dots]]$ for some C_0 input context, $k \in \mathbb{N}$ and $C_1, \dots, C_k$ contexts defined in Figure 6.

The output process P is *in evaluation position* in configuration $E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$. The input processes Q such that $(\sigma', Q) \in \mathcal{Q}$ for some σ' are *in evaluation position* in configurations $E, \mathcal{Q}, \mathcal{C}h$ and $E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$.

Lemma 4 *Consider a trace Tr of Q_0.*

All subterms of M that occur in non-evaluation position in a configuration $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v$ in the derivation of Tr are subterms of Q_0.

All subterms and subprocesses of processes in $\mathcal{Q}$ that occur in non-evaluation position in a configuration $E, \mathcal{Q}, \mathcal{C}h$ in the derivation of Tr are subterms, resp. subprocesses, of Q_0 up to renaming of channels.

All subterms and subprocesses of P and of processes in $\mathcal{Q}$ that occur in non-evaluation position in a configuration $E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ in the derivation of Tr are subterms, resp. subprocesses, of Q_0 up to renaming of channels (except for the process 0 that follows $\overline{start}\langle\rangle$ in initConfig(Q_0)*).*

Proof We say that

- a configuration $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v$ is ok when all subterms of M that are not in evaluation position are subterms of Q_0;
- a configuration $E, \mathcal{Q}, \mathcal{C}h$ is ok when all subterms and subprocesses of processes in $\mathcal{Q}$ that are not in evaluation position are subterms, resp. subprocesses, of Q_0 up to renaming of channels;
- a configuration $E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ is ok when all subterms and subprocesses of P and of processes in $\mathcal{Q}$ that are not in evaluation position are subterms, resp. subprocesses, of Q_0 up to renaming of channels.

We show by induction on the derivations that

1. if $E_1, \sigma_1, M_1, \mathcal{T}_1, \mu\mathcal{E}v_1$ is ok, then all configurations $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v$ in the derivation of E_1, $\sigma_1, M_1, \mathcal{T}_1, \mu\mathcal{E}v_1 \xrightarrow{p}_t E_2, \sigma_2, M_2, \mathcal{T}_2, \mu\mathcal{E}v_2$ are ok;
2. if $E_1, \mathcal{Q}_1, \mathcal{C}h_1$ is ok, then all configurations $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v$ in the derivation of $E_1, \mathcal{Q}_1$, $\mathcal{C}h_1 \rightsquigarrow E_2, \mathcal{Q}_2, \mathcal{C}h_2$ are ok and $E_2, \mathcal{Q}_2, \mathcal{C}h_2$ is ok;
3. if $E_1, \mathcal{Q}_1, \mathcal{C}h_1$ is ok, then reduce$(E_1, \mathcal{Q}_1, \mathcal{C}h_1)$ is ok and all configurations $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v$ or $E, \mathcal{Q}, \mathcal{C}h$ and in the derivation of $E_1, \mathcal{Q}_1, \mathcal{C}h_1 \rightsquigarrow^*$ reduce$(E_1, \mathcal{Q}_1, \mathcal{C}h_1)$ are ok;
4. if $E_1, (\sigma_1, P_1), \mathcal{Q}_1, \mathcal{C}h_1, \mathcal{T}_1, \mu\mathcal{E}v_1$ is ok, then all configurations $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v$ or $E, \mathcal{Q}, \mathcal{C}h$ or $E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ in the derivation of $E_1, (\sigma_1, P_1), \mathcal{Q}_1, \mathcal{C}h_1, \mathcal{T}_1, \mu\mathcal{E}v_1 \xrightarrow{p}_t E_2, (\sigma_2, P_2)$, $\mathcal{Q}_2, \mathcal{C}h_2, \mathcal{T}_2, \mu\mathcal{E}v_2$ are ok.

Property 1: In (ReplIndex), (Var), (Fun), and (EventAbortT), all terms are in evaluation position, so all configurations are ok. In (NewT), (LetT), (IfT1), (IfT2), (InsertT), and (EventT),

N (resp. N') is not in evaluation position, so by hypothesis it is a subterm of Q_0. Therefore, the target configuration is ok. In the rules for find, the recursive calls are on $D_j \wedge M_j$ which is not in evaluation position, so it is a subterm of Q_0. Therefore, the initial configurations of the recursive calls are ok, and we conclude for the configurations inside the recursive calls by induction hypothesis. The target configuration is ok because in (FindTE) and (FindT3), the resulting term is in evaluation position (it has no subterm), and in (FindT1) and (FindT2), the resulting term is a term that is not evaluation position in the initial configuration, so it is a subterm of Q_0. In (DefinedNo) and (DefinedYes), the terms $M_1, \dots, M_l$ are not in evaluation position in the initial configuration, so they are subterms of Q_0. Hence the initial configurations of the recursive calls are ok, and we conclude for the configurations inside the recursive calls by induction hypothesis. The target configuration is ok because in (DefinedNo), false is in evaluation position (it has no subterm) and in (DefinedYes), the resulting term M is a term that is not evaluation position in the initial configuration, so it is a subterm of Q_0. The case of get is similar to the one of find. In (CtxT), the initial configuration of the recursive call is ok because terms that are not in evaluation position in N are also not in evaluation position in $C[N]$, so they are subterms of Q_0. We conclude for the configurations inside the recursive call by induction hypothesis. The target configuration is ok because terms that are not in evaluation position in $C[N']$ are either not in evaluation position inside C, in which case they are subterms of Q_0 because the initial configuration is ok, or they are not in evaluation position in N', in which case they are also subterms of Q_0 because the target configuration of the recursive call is ok. In (CtxEventT), the target configuration is ok because the term is in evaluation position (it has no subterm).

Property 2: The desired property is preserved for unmodified processes in $\mathcal{Q}$. This is enough to conclude for (Nil). For (Par) and (Repl), the resulting processes Q_1, Q_2, Q are not in evaluation position in the initial configuration, so they are subprocesses of Q_0 up to renaming of channels. For (NewChannel), Q is not in evaluation position in the initial configuration, so it is a subprocesses of Q_0 up to renaming of channels, and so is $Q\{c'/c\}$. For (Input), the subterms of M_j that are not in evaluation position are not in evaluation position in $C[M_j]$, so they are subterms of Q_0. We can then apply Property 2 for the recursive call, so all configurations in the derivation of the recursive call are ok. The target configuration is ok because the subterms or subprocesses of $C[M'_j]$ not in evaluation position are either subterms of M'_j not in evaluation position, which are subterms of Q_0 since the target configuration is ok, or subterms of subprocesses of C not in evaluation position, which are subterms or subprocesses of Q_0 up to renaming of channels.

Property 3 follows immediately from Property 2 by induction.

Property 4: In (Output), Q'' is not in evaluation position in the initial configuration, so it is a subprocess of Q_0 up to renaming of channels. Therefore, the configuration $E, \{(\sigma, Q'')\}, \mathcal{C}h$ is ok. By Property 3, all configurations in the computation of $E, \mathcal{Q}', \mathcal{C}h'$ and $E, \mathcal{Q}', \mathcal{C}h'$ itself are ok. Moreover, P is not in evaluation position in the initial configuration, so it is a subprocess of Q_0 up to renaming of channels. We can then conclude that the target configuration is ok. All other cases can be treated similarly to terms in Property 1.

Moreover, in the computation of $\mathrm{initConfig}(Q_0)$, the configuration $\emptyset, \{(\sigma_0, Q_0)\}, \mathrm{fc}(Q)$ is ok, so by Property 3, $\mathrm{initConfig}(Q_0)$ is ok except for the process 0 that follows $\overline{start}\langle\rangle$ in $\mathrm{initConfig}(Q_0)$. That process disappears in the first reduction, which is by (Output). □

Corollary 1 *Consider a trace Tr of* Q_0.

In Tr, the target process of rules (New), (Let), (If1), (If2), (Find1), (Find2), (Insert), (Get1), (Get2), (Output), (Event) *is a subprocess of* Q_0 *up to renaming of channels.*

In Tr, the target term of rules (NewT), (LetT), (IfT1), (IfT2), (FindT1), (FindT2), (InsertT), (GetT1), (GetT2), (EventT), (DefinedYes) *is a subterm* Q_0.

Proof The target term or process of these rules appears in non-evaluation position in the initial configuration of these rules, so by Lemma 4, it is a subterm of Q_0 or a subprocess of Q_0 up to renaming of channels. $\square$

We say that a configuration *Conf is at program point* μ in a trace *Tr* of Q_0 when *Conf* occurs in the derivation of *Tr*, and either $\mathit{Conf} = E, \sigma, {}^{\mu}N, \mathcal{T}, \mu\mathcal{E}v$ for some subterm ${}^{\mu}N$ of Q_0 or $\mathit{Conf} = E, (\sigma, {}^{\mu}P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ for some subprocess ${}^{\mu}P$ of Q_0 up to renaming of channels.

Lemma 5 *Let Tr be a trace of* Q_0*. Let Conf be a configuration at program point* μ *in Tr. If* μ *is not inside a condition of* find *or* get*, then Conf is not in the derivation of an hypothesis of a rule for* find *or* get *inside the derivation of Tr.*

Proof The only rules that can conclude with a term find or get are (NewT), (LetT), (IfT1), (IfT2), (FindT1), (FindT2), (InsertT), (GetT1), (GetT2), (EventT), (DefinedYes) and, by Corollary 1, their target term is a subterm of Q_0. The situation is similar find and get processes. So the executed term or process in the initial configuration of rules for find and get is always a subterm or subprocess of Q_0 up to renaming of channels. When a configuration *Conf* is in the derivation of an hypothesis of a rule for find or get, it therefore always deals with program points syntactically in conditions of find or get in Q_0. (The semantic rules do not create program points.) Since μ is not inside a condition of find or get, we conclude that *Conf* is not in the derivation of an hypothesis of a rule for find or get. $\square$

Given a trace *Tr*, we define a partial ordering relation $\preceq_{\mathit{Tr}}$ (reflexive, transitive, antisymmetric) on the occurrences of configurations in the derivation of *Tr*: if $\mathit{Conf}_1 \xrightarrow{p}_t \mathit{Conf}_2$ occurs in the derivation of *Tr*, then $\mathit{Conf}_1 \preceq_{\mathit{Tr}} \mathit{Conf}_2$ and for all *Conf* that occur in the derivation of the assumptions of $\mathit{Conf}_1 \xrightarrow{p}_t \mathit{Conf}_2$, $\mathit{Conf}_1 \preceq_{\mathit{Tr}} \mathit{Conf} \preceq_{\mathit{Tr}} \mathit{Conf}_2$, and similarly for $\rightsquigarrow$ instead of $\xrightarrow{p}_t$. If *Tr* is a trace of Q_0, then for all *Conf* that occur in the derivation of $\emptyset, \{(\sigma_0, Q_0)\}, \mathrm{fc}(Q_0) \rightsquigarrow^* \emptyset, \mathcal{Q}, \mathcal{C}h$, we have $\mathit{Conf} \preceq_{\mathit{Tr}} \mathrm{initConfig}(Q_0)$. When $\mathit{Conf}_1 \preceq_{\mathit{Tr}} \mathit{Conf}_2$, we say that Conf_1 *occurs before* Conf_2 in *Tr*, or equivalently, that Conf_2 *occurs after* Conf_1 in *Tr*.

We say that that a configuration $\mathit{Conf} = E, \sigma, {}^{\mu}N, \mathcal{T}, \mu\mathcal{E}v$, $\mathit{Conf} = E, (\sigma, {}^{\mu}P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$, $\mathit{Conf} = E, (\sigma, {}^{\mu'}P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ with $(\sigma', {}^{\mu}Q) \in \mathcal{Q}$, or $\mathit{Conf} = E, \mathcal{Q}, \mathcal{C}h$ with $(\sigma', {}^{\mu}Q) \in \mathcal{Q}$ *is inside program point* μ. A configuration may be inside several program points: $\mathit{Conf} = E, \mathcal{Q}, \mathcal{C}h$ is inside the program points of the input processes in $\mathcal{C}h$, $\mathit{Conf} = E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ is inside the program points of one output process (P) as well as the input processes in $\mathcal{C}h$.

We say that the program point μ is immediately above the program points μ_j in a process Q_0 when Q_0 contains one of the following constructs:

$${}^{\mu}x[{}^{\mu_1}M_1, \dots, {}^{\mu_m}M_m]$$
$${}^{\mu}f({}^{\mu_1}M_1, \dots, {}^{\mu_m}M_m)$$
$${}^{\mu}\mathsf{new}\ x[\widetilde{i}] : T; {}^{\mu_1}N$$
$${}^{\mu}\mathsf{let}\ x[\widetilde{i}] : T = {}^{\mu_1}M\ \mathsf{in}\ {}^{\mu_2}N$$
$${}^{\mu}\mathsf{if}\ {}^{\mu_1}M\ \mathsf{then}\ {}^{\mu_2}N\ \mathsf{else}\ {}^{\mu_3}N'$$
$${}^{\mu}\mathsf{find}[\mathit{unique}?]\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}\ \mathsf{suchthat}$$
$$\mathsf{defined}({}^{\mu_{j,1}}M_{j1}, \dots, {}^{\mu_{j,l_j}}M_{jl_j}) \wedge {}^{\mu_{j,l_j+1}}M'_j\ \mathsf{then}\ {}^{\mu_{j,l_j+2}}N_j)\ \mathsf{else}\ {}^{\mu_0}N'$$
$${}^{\mu}\mathsf{insert}\ \mathit{Tbl}({}^{\mu_1}M_1, \dots, {}^{\mu_l}M_l); {}^{\mu_0}N$$
$${}^{\mu}\mathsf{get}[\mathit{unique}?]\ \mathit{Tbl}(x_1[\widetilde{i}] : T_1, \dots, x_l[\widetilde{i}] : T_l)\ \mathsf{suchthat}\ {}^{\mu_1}M\ \mathsf{in}\ {}^{\mu_2}N\ \mathsf{else}\ {}^{\mu_3}N'$$

$^{\mu}\mathsf{event}\ e(^{\mu_1}M_1,\dots,{}^{\mu_l}M_l);{}^{\mu_0}N$
$^{\mu}(^{\mu_1}Q \mid {}^{\mu_2}Q')$
$^{\mu}!^{i\leq n\,\mu_1}Q$
$^{\mu}\mathsf{newChannel}\ c;{}^{\mu_1}Q$
$^{\mu}c[^{\mu_1}M_1,\dots,{}^{\mu_l}M_l](p);{}^{\mu_0}P$
$^{\mu}\overline{c[^{\mu_1}M_1,\dots,{}^{\mu_l}M_l]}\langle^{\mu_0}N\rangle;Q$
$^{\mu}\mathsf{new}\ x[\widetilde{i}] : T;{}^{\mu_1}P$
$^{\mu}\mathsf{let}\ x[\widetilde{i}] = {}^{\mu_1}M\ \mathsf{in}\ {}^{\mu_2}P$
$^{\mu}\mathsf{if}\ {}^{\mu_1}M\ \mathsf{then}\ {}^{\mu_2}P\ \mathsf{else}\ {}^{\mu_3}P'$
$^{\mu}\mathsf{find}[\mathit{unique}?]\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1},\dots,u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}\ \mathsf{suchthat}$
  $\mathsf{defined}(^{\mu_{j,1}}M_{j1},\dots,{}^{\mu_{j,l_j}}M_{jl_j}) \wedge {}^{\mu_{j,l_j+1}}M_j\ \mathsf{then}\ {}^{\mu_{j,l_j+2}}P_j)\ \mathsf{else}\ {}^{\mu_0}P$
$^{\mu}\mathsf{insert}\ \mathit{Tbl}(^{\mu_1}M_1,\dots,{}^{\mu_l}M_l);{}^{\mu_0}P$
$^{\mu}\mathsf{get}[\mathit{unique}?]\ \mathit{Tbl}(x_1[\widetilde{i}] : T_1,\dots,x_l[\widetilde{i}] : T_l)\ \mathsf{suchthat}\ {}^{\mu_1}M\ \mathsf{in}\ {}^{\mu_2}P\ \mathsf{else}\ {}^{\mu_3}P'$
$^{\mu}\mathsf{event}\ e(^{\mu_1}M_1,\dots,{}^{\mu_l}M_l);{}^{\mu_0}P$

The relation "μ is above μ'" is the reflexive and transitive closure of "μ is immediately above μ'".

Lemma 6 *Let Tr be a trace of Q_0. If Conf is a configuration in Tr inside program point μ and μ is a program point in Q_0, then either Conf is at the program point μ at the top of Q_0, or there exists a configuration $\mathit{Conf}' \neq \mathit{Conf}$ such that $\mathit{Conf}' \preceq_{\mathit{Tr}} \mathit{Conf}$ and Conf' is inside program point μ or inside the program point μ' immediately above μ in Q_0.*

As a consequence, if a configuration Conf inside program point μ in Q_0 is in Tr, then there are configurations inside all program points above μ in Q_0 before Conf in Tr.

Proof First property. Since *Conf* is a configuration in *Tr*, we are in one of the following cases:

- $\mathit{Conf} = \emptyset, \{(\sigma_0, Q_0)\}, \mathrm{fc}(Q_0)$, the very first configuration of *Tr*. The configuration *Conf* is at the program point μ at the top of Q_0.
- $\mathit{Conf} = \mathrm{initConfig}(Q_0) = \emptyset, (\sigma_0, {}^{\mu''}\overline{\mathit{start}}\langle\rangle), \mathcal{Q}, \mathcal{C}h, \emptyset, \emptyset$. Since μ'' is not in Q_0, μ is the program point of a process in $\mathcal{Q}$. Then $\mathit{Conf}' = \emptyset, \mathcal{Q}, \mathcal{C}h$ is also inside μ and $\mathit{Conf}' \preceq_{\mathit{Tr}} \mathit{Conf}$, by definition of $\preceq_{\mathit{Tr}}$.
- *Conf* is the initial configuration of an assumption of a semantic rule. In rules for find, the initial configuration of assumption is not inside a program point (because it evaluates the defined condition, not a term). In rules for get, (CtxT), (Ctx), and (Input), the initial configuration of the conclusion is inside the program point μ' immediately above μ. In rules (DefinedNo) and (DefinedYes), these rules are used to conclude assumptions of find, and the initial configuration of the find rule is inside the program point μ' immediately above μ. In rule (Output), $\mathit{Conf} = E, \{(\sigma, Q'')\}, \mathcal{C}h$, and the initial configuration of the conclusion of the rule (Output) is inside the program point μ' immediately above μ.
- *Conf* is the target configuration of a semantic rule. In rules (NewT), (LetT), (IfT1), (IfT2), (FindT1), (FindT2), (InsertT), (GetT1), (GetT2), and (EventT), the initial configuration

of the rule is inside the program point μ' immediately above μ. In rule (CtxT), the initial configuration of the rule is inside the same program point μ. In rule (DefinedYes), this rule is used to conclude assumptions of find, and the initial configuration of the find rule is inside the program point μ' immediately above μ. In the rules for input processes, if μ is the program point of an unchanged element of $\mathcal{Q}$, the initial configuration of the rule is also inside μ. This is sufficient for (Nil). If μ is the program point of a modified process, then for rules (Par), (Repl), and (NewChannel), the initial configuration of the rule is inside the program point μ' immediately above μ, and for rule (Input), the initial configuration of the rule is inside the same program point μ. In the rules for output processes other than (Output), $\mathcal{Q}$ is unchanged, so if μ is the program point of a process in $\mathcal{Q}$, then the initial configuration of the rule is inside the same program point μ. In rules (New), (Let), (If1), (If2), (Find1), (Find2), (Insert), (Get1), (Get2), and (Event), if μ is the program point of an output process, then the initial configuration of the rule is inside the program point μ' immediately above μ. In rule (Ctx), the initial configuration of the rule is inside the same program point μ. In rule (Output), if μ is the program point of a process in $\mathcal{Q}$, then the initial configuration of the rule is inside the same program point μ. If μ is the program point of a process in $\mathcal{Q}'$, then the configuration $E, \mathcal{Q}', \mathcal{C}h'$ in the assumption of the rule is inside the same program point μ. If μ is the program point of P, then the initial configuration of the rule is inside the program point μ' immediately above μ, because $(\sigma', Q_0) \in S \subseteq \mathcal{Q}$ and μ' is the program point of Q_0.

Second property. Suppose that *Conf* is inside μ and there is a program point μ' immediately above μ in Q_0. Let us show that there exists $\mathit{Conf}' \preceq_{\mathit{Tr}} \mathit{Conf}$ such that Conf' is inside μ'. The proof proceeds by well-founded induction on $\preceq_{\mathit{Tr}}$. The program point μ is not at the top of Q_0, so by the first property, there exists $\mathit{Conf}' \neq \mathit{Conf}$ such that $\mathit{Conf}' \preceq_{\mathit{Tr}} \mathit{Conf}$ and Conf' is inside μ or inside μ'. If Conf' is inside μ, we conclude by applying the induction hypothesis on Conf'. If Conf' is inside μ', we have the result. By applying this property repeatedly, we obtain the second property. □

Lemma 7 *Let Tr be a trace of* Q_0.

1. *If* $\mathit{Conf} = E, \sigma, C_m[\dots C_1[{}^{\mu}M]\dots], \mathcal{T}, \mu\mathcal{E}v$ *is the target configuration of a semantic rule in Tr, where* μ *is a program point in* Q_0, $C_1, \dots, C_m$ *are term contexts defined in Figure 6, and* ${}^{\mu}M$ *is not a subterm of* Q_0*, then the reduction that yields Conf is obtained by* m *applications of* (CtxT) *with contexts* $C_m, \dots, C_1$ *from a reduction with target configuration* $E, \sigma, {}^{\mu}M, \mathcal{T}, \mu\mathcal{E}v$*, itself proved by* (CtxT).

2. *If* $\mathit{Conf} = E, (\sigma, C_0[C_m[\dots C_1[{}^{\mu}M]\dots]]), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ *is the target configuration of a semantic rule in Tr, where* μ *is a program point in* Q_0, C_0 *is a process context defined in Figure 10,* $C_1, \dots, C_m$ *are term contexts defined in Figure 6, and* ${}^{\mu}M$ *is not a subterm of* Q_0*, then the reduction that yields Conf is obtained by one application of* (Ctx) *with context* C_0 *and* m *applications of* (CtxT) *with contexts* $C_m, \dots, C_1$ *from a reduction with target configuration* $E, \sigma, {}^{\mu}M, \mathcal{T}, \mu\mathcal{E}v$*, itself proved by* (CtxT).

3. *Let* $Q = C_0[C_m[\dots C_1[{}^{\mu}M]\dots]]$ *where* μ *is a program point in* Q_0, C_0 *is an input context,* $C_1, \dots, C_m$ *are term contexts defined in Figure 6, and* ${}^{\mu}M$ *is not a subterm of* Q_0*. If* $\mathit{Conf} = E, \{(\sigma, Q\} \uplus \mathcal{Q}, \mathcal{C}h$ *is target configuration of a semantic rule that affects* Q *in Tr, then the reduction that yields Conf is obtained by one application of* (Input) *with context* C_0 *and* m *applications of* (CtxT) *with contexts* $C_m, \dots, C_1$ *from a reduction with target configuration* $E, \sigma, {}^{\mu}M, \mathcal{T}, \mu\mathcal{E}v$*, itself proved by* (CtxT).

Proof Property 1. This property is proved by induction of m. The reduction that yields *Conf* cannot be obtained by (NewT), (LetT), (IfT1), (IfT2), (FindT1), (FindT2), (InsertT), (GetT1), (GetT2), (EventT), (DefinedYes), because in this case, by Corollary 1, $C_m[\dots C_1[{}^{\mu}M]\dots]$ would be a subterm of Q_0, so ${}^{\mu}M$ would be a subterm of Q_0. So it is obtained by (CtxT). For $m = 0$, this is enough to conclude. For $m > 0$, the rule (CtxT) is applied with context C_m. Indeed, since ${}^{\mu}M$ is not a value, the hole of the context cannot be after $C_{m-1}[\dots C_1[{}^{\mu}M]\dots]$ and, since ${}^{\mu}M$ is not a subterm of Q_0, the hole of the context cannot be before $C_{m-1}[\dots C_1[{}^{\mu}M]\dots]$. (If it were before, $C_{m-1}[\dots C_1[{}^{\mu}M]\dots]$ would not be in evaluation position in the initial configuration of rule (CtxT), so by Lemma 4, $C_{m-1}[\dots C_1[{}^{\mu}M]\dots]$ would be a subterm of Q_0.) Hence the reduction that yields *Conf* is obtained from a reduction that yields $E, \sigma, C_{m-1}[\dots C_1[{}^{\mu}M]\dots], \mathcal{T}, \mu\mathcal{E}v$ by applying (CtxT) with context C_m. We conclude by applying the induction hypothesis.

Property 2. The reduction that yields *Conf* cannot be obtained by (New), (Let), (If1), (If2), (Find1), (Find2), (Insert), (Get1), (Get2), (Output), or (Event), because in this case, by Corollary 1, the process of *Conf* would be a subprocess of Q_0 up to renaming of channels, so ${}^{\mu}M$ would be a subterm of Q_0. Therefore, *Conf* is the target configuration of (Ctx). Furthermore, by the same reasoning as in Property 1, this rule is applied with context C_0. Hence the reduction that yields *Conf* is obtained from a reduction that yields $E, \sigma, C_m[\dots C_1[{}^{\mu}M]\dots], \mathcal{T}, \mu\mathcal{E}v$ by applying (Ctx) with context C_0. We conclude by Property 1.

Property 3. The reduction that yields *Conf* cannot be obtained by (Par), (Repl), or (NewChannel), because in this case, by Lemma 4, since Q occurs in non-evaluation position in the initial configuration of the rule, Q would be a subprocess of Q_0 up to renaming of channels, so ${}^{\mu}M$ would be a subterm of Q_0. Therefore, *Conf* is the target configuration of (Input). Furthermore, by the same reasoning as in Property 1, this rule is applied with context C_0. Hence the reduction that yields *Conf* is obtained from a reduction that yields $E, \sigma, C_m[\dots C_1[{}^{\mu}M]\dots], \mathcal{T}, \mu\mathcal{E}v$ by applying (Input) with context C_0. We conclude by Property 1. □

Lemma 8 *Let Tr be a trace of Q_0.*

1. *If Conf* $= E, (\sigma, {}^{\mu}P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ *is a configuration in Tr where μ is a program point in Q_0, then this configuration is derived in Tr from a configuration Conf*$' = E', (\sigma, {}^{\mu}P'), \mathcal{Q}, \mathcal{C}h, \mathcal{T}', \mu\mathcal{E}v'$ *where ${}^{\mu}P'$ is a subprocess of Q_0 up to renaming of channels, by* (Ctx) *any number of times.*

2. *If Conf* $= E, \{(\sigma, {}^{\mu}Q)\} \uplus \mathcal{Q}, \mathcal{C}h$ *is a configuration in Tr where μ is a program point in Q_0, then, possibly after swapping reductions in Tr, this configuration is derived from a configuration Conf*$' = E, \{(\sigma, {}^{\mu}Q')\} \uplus \mathcal{Q}, \mathcal{C}h$ *where ${}^{\mu}Q'$ is a subprocess of Q_0 up to renaming of channels, by* (Input) *any number of times.*

3. *If Conf* $= E, (\sigma, C_0[C_m[\dots C_1[{}^{\mu}M]\dots]]), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ *is a configuration in Tr where μ is a program point in Q_0, C_0 is a process context defined in Figure 10, and $C_1, \dots, C_m$ are term contexts defined in Figure 6, then we have*

$$E', \sigma, {}^{\mu}M', \mathcal{T}', \mu\mathcal{E}v' \xrightarrow{p}_t \dots \xrightarrow{p'}_{t'} E, \sigma, {}^{\mu}M, \mathcal{T}, \mu\mathcal{E}v$$

by (CtxT) *any number of times, where ${}^{\mu}M'$ is a subterm of Q_0 and in Tr, these reductions are in fact performed starting from Conf*$' = E', (\sigma, C_0[C_m[\dots C_1[{}^{\mu}M']\dots]]), \mathcal{Q}, \mathcal{C}h, \mathcal{T}', \mu\mathcal{E}v'$ *under one application of* (Ctx) *with context C_0 and m applications of* (CtxT) *with contexts $C_m, \dots, C_1$.*

4. *If Conf* $= E, \{(\sigma, C_0[C_m[\dots C_1[{}^{\mu}M]\dots]])\} \uplus \mathcal{Q}, \mathcal{C}h$ *is a configuration in Tr, where* μ *is a program point in* Q_0*,* C_0 *is an input context, and* $C_1, \dots, C_m$ *are term contexts defined in Figure 6, then we have*

$$E, \sigma, {}^{\mu}M', \emptyset, \emptyset \xrightarrow{p}_t \dots \xrightarrow{p'}_{t'} E, \sigma, {}^{\mu}M, \emptyset, \emptyset$$

by (CtxT) *any number of times, where* ${}^{\mu}M'$ *is a subterm of* Q_0 *and possibly after swapping reductions in Tr, these reductions are in fact performed starting from a configuration Conf'* $= E, \{(\sigma, C_0[C_m[\dots C_1[{}^{\mu}M']\dots]])\} \uplus \mathcal{Q}, \mathcal{C}h$ *under one application of* (Input) *with context* C_0 *and* m *applications of* (CtxT) *with contexts* $C_m, \dots, C_1$*.*

5. *If Conf* $= E, \sigma, C_l[\dots C_1[{}^{\mu}M]\dots], \mathcal{T}, \mu\mathcal{E}v$ *is a configuration in Tr and* μ *is a program point in* Q_0 *where* $C_1, \dots, C_l$ *are term contexts defined in Figure 6, then we have*

$$E', \sigma, {}^{\mu}M', \mathcal{T}', \mu\mathcal{E}v' \xrightarrow{p}_t \dots \xrightarrow{p'}_{t'} E, \sigma, {}^{\mu}M, \mathcal{T}, \mu\mathcal{E}v$$

by (CtxT) *any number of times, where* ${}^{\mu}M'$ *is a subterm of* Q_0 *and in Tr, these reductions are in fact performed starting*

- *from a configuration Conf'* $= E', \sigma, C_m[\dots C_1[{}^{\mu}M']\dots], \mathcal{T}', \mu\mathcal{E}v'$ *where* $m \geq l$*,* $C_1, \dots, C_m$ *are term contexts defined in Figure 6, under* m *applications of* (CtxT) *with contexts* $C_m, \dots, C_1$*.*
- *or from a configuration Conf'* $= E', (\sigma, C_0[C_m[\dots C_1[{}^{\mu}M']\dots]]), \mathcal{Q}, \mathcal{C}h, \mathcal{T}', \mu\mathcal{E}v'$ *where* $m \geq l$*,* C_0 *is a process context defined in Figure 10, and* $C_1, \dots, C_l$ *are term contexts defined in Figure 6, under one application of* (Ctx) *with context* C_0 *and* m *applications of* (CtxT) *with contexts* $C_m, \dots, C_1$*.*
- *or, possibly after swapping reductions in Tr, from a configuration Conf'* $= E', \{(\sigma, C_0[C_m[\dots C_1[{}^{\mu}M']\dots]])\} \uplus \mathcal{Q}, \mathcal{C}h$ *where* $m \geq l$*,* C_0 *is an input context and* $C_1, \dots, C_m$ *are term contexts defined in Figure 6, under one application of* (Input) *with context* C_0 *and* m *applications of* (CtxT) *with contexts* $C_m, \dots, C_1$*.*

Proof Property 1. The proof proceeds by well-founded induction on $\preceq_{Tr}$. The configuration *Conf* cannot be initConfig(Q_0) because μ is a program point in Q_0. Then, *Conf* is the target configuration of some semantic rule. If *Conf* is the target configuration of (New), (Let), (If1), (If2), (Find1), (Find2), (Insert), (Get1), (Get2), (Output), or (Event), then by Corollary 1, ${}^{\mu}P$ is a subprocess of Q_0 up to renaming of channels, so the property holds with *Conf'* = *Conf*. If it is the target configuration of (Ctx), we conclude by applying the induction hypothesis to the initial configuration of this rule.

Property 2. The proof proceeds by well-founded induction on $\preceq_{Tr}$. If *Conf* $= \emptyset, \{(\sigma_0, Q_0)\}$, fc($Q_0$), then the property holds with *Conf'* = *Conf*, since Q_0 is a subprocess of Q_0. If *Conf* is the initial configuration of the assumption of (Output), then by Lemma 4, Q is a subprocess of Q_0 up to renaming of channels, since Q occurs in non-evaluation position in the initial configuration of (Output). So the property holds with *Conf'* = *Conf*. Otherwise, *Conf* is the target configuration of a semantic rule.

- If that rule does not affect Q, then it is of the form $E, \{(\sigma, {}^{\mu}Q)\} \uplus \mathcal{Q}', \mathcal{C}h' \rightsquigarrow E, \{(\sigma, {}^{\mu}Q)\} \uplus \mathcal{Q}, \mathcal{C}h$. We apply the induction hypothesis to $E, \{(\sigma, {}^{\mu}Q)\} \uplus \mathcal{Q}', \mathcal{C}h'$, so possibly after swapping reductions in *Tr*, we have $E, \{(\sigma, {}^{\mu}Q')\} \uplus \mathcal{Q}', \mathcal{C}h' \rightsquigarrow^* E, \{(\sigma, {}^{\mu}Q)\} \uplus \mathcal{Q}', \mathcal{C}h'$ by (Input) any number of times, where ${}^{\mu}Q'$ is a subprocess of Q_0 up to renaming of channels. By swapping reductions, we have $E, \{(\sigma, {}^{\mu}Q')\} \uplus \mathcal{Q}', \mathcal{C}h' \rightsquigarrow E, \{(\sigma, {}^{\mu}Q')\} \uplus \mathcal{Q}, \mathcal{C}h \rightsquigarrow^* E, \{(\sigma, {}^{\mu}Q)\} \uplus \mathcal{Q}, \mathcal{C}h$, so we obtain the desired property with *Conf'* $= E, \{(\sigma, {}^{\mu}Q')\} \uplus \mathcal{Q}, \mathcal{C}h$.

- Otherwise, that rule affects Q. If that rule is (Par), (Repl), or (NewChannel), then by Lemma 4, Q is a subprocess of Q_0 up to renaming of channels, since Q occurs in non-evaluation position in the initial configuration of the rule. So the property holds with $\mathit{Conf}' = \mathit{Conf}$. If that rule is (Input), then we obtain the result by induction hypothesis applied to the initial configuration of the rule.

Property 3. The proof proceeds by well-founded induction on $\preceq_{Tr}$. The configuration Conf cannot be initConfig(Q_0) because μ is a program point in Q_0. Then, Conf is the target configuration of some semantic rule. If ${}^{\mu}M$ is a subterm of Q_0, then the property holds with $\mathit{Conf}' = \mathit{Conf}$. Otherwise, by Lemma 7, Property 2, the reduction that yields Conf is obtained by one application of (Ctx) with context C_0 and m applications of (CtxT) with contexts C_m, $\dots$, C_1 from a reduction with target configuration $E, \sigma, {}^{\mu}M, \mathcal{T}, \mu\mathcal{E}v$, itself proved by (CtxT). We obtain the desired property by applying the induction hypothesis to the configuration before this reduction step by (Ctx).

Property 4. Let $Q = C_0[C_m[\dots C_1[{}^{\mu}M]\dots]]$. The proof proceeds by well-founded induction on $\preceq_{Tr}$. If $\mathit{Conf} = \emptyset, \{(\sigma_0, Q_0)\}, \mathrm{fc}(Q_0)$, then the property holds with $\mathit{Conf}' = \mathit{Conf}$, since Q_0 is a subprocess of Q_0, so ${}^{\mu}M$ is a subterm of Q_0. If Conf is the initial configuration of the assumption of (Output), then by Lemma 4, Q is a subprocess of Q_0 up to renaming of channels, since Q occurs in non-evaluation position in the initial configuration of (Output). So ${}^{\mu}M$ is a subterm of Q_0 and the property holds with $\mathit{Conf}' = \mathit{Conf}$. Otherwise, Conf is the target configuration of a semantic rule. If that rule does not affect Q, then we swap reductions as in the proof of Property 2. Otherwise, if ${}^{\mu}M$ is a subterm of Q_0, then the property holds with $\mathit{Conf}' = \mathit{Conf}$. Otherwise, by Lemma 7, Property 3, the reduction that yields Conf is obtained by one application of (Input) with context C_0 and m applications of (CtxT) with contexts C_m, $\dots$, C_1 from a reduction with target configuration $E, \sigma, {}^{\mu}M, \mathcal{T}, \mu\mathcal{E}v$, itself proved by (CtxT). We obtain the desired property by applying the induction hypothesis to the configuration before this reduction step by (Input).

Property 5. The proof proceeds by well-founded induction on $\preceq_{Tr}$. First case: Conf is the initial configuration of an assumption of a semantic rule. This rule cannot be a rule for find (in rules for find, the initial configuration of assumption is not inside a program point, because it evaluates the defined condition, not a term). In rules for get, (DefinedNo), and (DefinedYes), by Lemma 4, since the term in Conf occurs in non-evaluation position in the initial configuration of the rule, it is a subterm of Q_0, so ${}^{\mu}M$ is a subterm of Q_0. The property holds with $\mathit{Conf}' = \mathit{Conf}$. In (CtxT), we let C_{l+1} be the context used in this rule, and we conclude by induction hypothesis applied to the initial configuration of this rule: $E, \sigma, C_{l+1}[C_l[\dots C_1[{}^{\mu}M]\dots]], \mathcal{T}, \mu\mathcal{E}v$. In (Input), we let $m = l$. The initial configuration of the rule is of the form $E, \{C_0[C_l[\dots C_1[{}^{\mu}M]\dots]]\} \uplus \mathcal{Q}, \mathcal{C}h$ for some input context C_0. We conclude by applying Property 4 to that configuration. In (Ctx), we let $m = l$. The initial configuration of the rule is of the form $E, (\sigma, C_0[C_l[\dots C_1[{}^{\mu}M]\dots]]), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ for some process context C_0 defined in Figure 10. We conclude by applying Property 3 to that configuration.

Second case: Conf is the target configuration of a semantic rule. If ${}^{\mu}M$ is a subterm of Q_0, then the property holds with $\mathit{Conf}' = \mathit{Conf}$. Otherwise, by Lemma 7, Property 1, the reduction that yields Conf is obtained by l applications of (CtxT) with contexts C_l, $\dots$, C_1 from a reduction with target configuration $E, \sigma, {}^{\mu}M, \mathcal{T}, \mu\mathcal{E}v$, itself proved by (CtxT). We apply the induction hypothesis to the initial configuration of the reduction that yields Conf, and we continue the reduction one more step by applying possibly (Ctx) or (Input) with context C_0, (CtxT) m times with contexts C_m, $\dots$, C_1 under the reduction with target configuration $E, \sigma, {}^{\mu}M, \mathcal{T}, \mu\mathcal{E}v$. Up to swapping of reductions in the case of input processes, this is also what happens in Tr. (By inspection of the rules, only the rule (Input) with context C_0 can reduce an input process $C_0[N]$, where C_0 is an input context; only the rule (Ctx) with context C_0 can reduce an output process $C_0[N]$, where C_0 is a process context defined in Figure 10; only the rule (CtxT) with context C_j can reduce a term $C_j[N]$, where C_j is a term context defined in Figure 6.) $\square$

2.4.3 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.

We define the multiset of variable accesses that may be defined by a term or a process (given the replication indices fixed by a mapping sequence σ) as follows:

$$\begin{aligned}
&\mathit{Defined}(\sigma, {}^{\mu}i) = \emptyset\\
&\mathit{Defined}(\sigma, {}^{\mu}x[M_1,\dots,M_m]) = \biguplus_{j=1}^{m} \mathit{Defined}(\sigma, M_j)\\
&\mathit{Defined}(\sigma, {}^{\mu}f(M_1,\dots,M_m)) = \biguplus_{j=1}^{m} \mathit{Defined}(\sigma, M_j)\\
&\mathit{Defined}(\sigma, {}^{\mu}\mathsf{new}\ x[\widetilde{i}] : T; N) = \{x[\sigma(\widetilde{i})]\} \uplus \mathit{Defined}(\sigma, N)\\
&\mathit{Defined}(\sigma, {}^{\mu}\mathsf{let}\ x[\widetilde{i}] : T = M\ \mathsf{in}\ N) = \{x[\sigma(\widetilde{i})]\} \uplus \mathit{Defined}(\sigma, M) \uplus \mathit{Defined}(\sigma, N)\\
&\mathit{Defined}(\sigma, {}^{\mu}\mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ N') = \mathit{Defined}(M) \uplus \max(\mathit{Defined}(N), \mathit{Defined}(N'))\\
&\mathit{Defined}(\sigma, {}^{\mu}\mathsf{find}[\mathit{unique}?]\ (\textstyle\bigoplus_{j=1}^{m} \widetilde{u_j}[\widetilde{i}] = \widetilde{i_j} \leq \widetilde{n_j}\ \mathsf{suchthat}\ \mathsf{defined}(\widetilde{M_j}) \wedge M_j\ \mathsf{then}\ N_j)\ \mathsf{else}\ N) =\\
&\qquad \max(\max_{j=1}^{m} \max_{\widetilde{a} \leq \widetilde{n_j}} \mathit{Defined}(\sigma[\widetilde{i_j} \mapsto \widetilde{a}], M_j), \max_{j=1}^{m}\{\widetilde{u_j}[\sigma(\widetilde{i})]\} \uplus \mathit{Defined}(\sigma, N_j), \mathit{Defined}(\sigma, N))\\
&\mathit{Defined}(\sigma, {}^{\mu}\mathsf{insert}\ \mathit{Tbl}(M_1,\dots,M_l); N) = \biguplus_{j=1}^{l} \mathit{Defined}(\sigma, M_j) \uplus \mathit{Defined}(\sigma, N)\\
&\mathit{Defined}(\sigma, {}^{\mu}\mathsf{get}[\mathit{unique}?]\ \mathit{Tbl}(x_1[\widetilde{i}] : T_1,\dots,x_l[\widetilde{i}] : T_l)\ \mathsf{suchthat}\ M\ \mathsf{in}\ N\ \mathsf{else}\ N') =\\
&\qquad \max(\{x_j[\sigma(\widetilde{i})] \mid j \leq l\} \uplus \max(\mathit{Defined}(\sigma, M), \mathit{Defined}(\sigma, N)), \mathit{Defined}(\sigma, N'))\\
&\mathit{Defined}(\sigma, {}^{\mu}\mathsf{event}\ e(M_1,\dots,M_l); N) = \biguplus_{j=1}^{l} \mathit{Defined}(\sigma, M_j) \uplus \mathit{Defined}(\sigma, N)\\
&\mathit{Defined}(\sigma, {}^{\mu}\mathsf{event\_abort}\ e) = \emptyset\\
&\mathit{Defined}(\sigma, a) = \mathit{Defined}(\sigma, \mathsf{event\_abort}\ (\mu, \widetilde{a}) : e) = \emptyset\\
&\mathit{Defined}(\sigma, {}^{\mu}0) = \emptyset\\
&\mathit{Defined}(\sigma, {}^{\mu}(Q_1 \mid Q_2)) = \mathit{Defined}(\sigma, Q_1) \uplus \mathit{Defined}(\sigma, Q_2)\\
&\mathit{Defined}(\sigma, {}^{\mu}!^{i \leq n} Q) = \biguplus_{a \in [1,n]} \mathit{Defined}(\sigma[i \mapsto a], Q)\\
&\mathit{Defined}(\sigma, {}^{\mu}\mathsf{newChannel}\ c; Q) = \mathit{Defined}(\sigma, Q)\\
&\mathit{Defined}(\sigma, {}^{\mu}c[M_1,\dots,M_l](x[\widetilde{i}] : T); P) = \{x[\sigma(\widetilde{i})]\} \uplus \mathit{Defined}(\sigma, P)\\
&\mathit{Defined}(\sigma, {}^{\mu}\overline{c[M_1,\dots,M_l]}\langle N\rangle; Q) = \biguplus_{j=1}^{l} \mathit{Defined}(\sigma, M_j) \uplus \mathit{Defined}(\sigma, N) \uplus \mathit{Defined}(\sigma, Q)\\
&\mathit{Defined}(\sigma, {}^{\mu}\mathsf{new}\ x[\widetilde{i}] : T; P) = \{x[\sigma(\widetilde{i})]\} \uplus \mathit{Defined}(\sigma, P)
\end{aligned}$$

$$\begin{aligned}
&\mathit{Defined}(\sigma, {}^{\mu}\mathsf{let}\ x[\widetilde{i}] : T = M\ \mathsf{in}\ P) = \{x[\sigma(\widetilde{i})]\} \uplus \mathit{Defined}(\sigma, M) \uplus \mathit{Defined}(\sigma, P)\\
&\mathit{Defined}(\sigma, {}^{\mu}\mathsf{if}\ M\ \mathsf{then}\ P\ \mathsf{else}\ P') = \mathit{Defined}(M) \uplus \max(\mathit{Defined}(P), \mathit{Defined}(P'))\\
&\mathit{Defined}(\sigma, {}^{\mu}\mathsf{find}[\mathit{unique}?]\ (\textstyle\bigoplus_{j=1}^{m} \widetilde{u_j}[\widetilde{i}] = \widetilde{i_j} \leq \widetilde{n_j}\ \mathsf{suchthat\ defined}(\widetilde{M_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P) =\\
&\qquad \max(\max_{j=1}^{m} \max_{\widetilde{a} \leq \widetilde{n_j}} \mathit{Defined}(\sigma[\widetilde{i_j} \mapsto \widetilde{a}], M_j), \max_{j=1}^{m}\{\widetilde{u_j}[\sigma(\widetilde{i})]\} \uplus \mathit{Defined}(\sigma, P_j), \mathit{Defined}(\sigma, P))\\
&\mathit{Defined}(\sigma, {}^{\mu}\mathsf{insert}\ \mathit{Tbl}(M_1, \dots, M_l); P) = \biguplus_{j=1}^{l} \mathit{Defined}(\sigma, M_j) \uplus \mathit{Defined}(\sigma, P)\\
&\mathit{Defined}(\sigma, {}^{\mu}\mathsf{get}[\mathit{unique}?]\ \mathit{Tbl}(x_1[\widetilde{i}] : T_1, \dots, x_l[\widetilde{i}] : T_l)\ \mathsf{suchthat}\ M\ \mathsf{in}\ P\ \mathsf{else}\ P') =\\
&\qquad \max(\{x_j[\sigma(\widetilde{i})] \mid j \leq l\} \uplus \max(\mathit{Defined}(\sigma, M), \mathit{Defined}(\sigma, P)), \mathit{Defined}(\sigma, P'))\\
&\mathit{Defined}(\sigma, {}^{\mu}\mathsf{event}\ e(M_1, \dots, M_l); P) = \biguplus_{j=1}^{l} \mathit{Defined}(\sigma, M_j) \uplus \mathit{Defined}(\sigma, P)\\
&\mathit{Defined}(\sigma, {}^{\mu}\mathsf{event\_abort}\ e) = \emptyset\\
&\mathit{Defined}(\sigma, \mathsf{abort}) = \emptyset
\end{aligned}$$

Notice that, by Invariant 5, the terms M_j in channels of inputs and the terms $\widetilde{M_j}$ in defined conditions of find do not define any variable. By Invariant 3, the variables defined in conditions of find and get can be considered as defined temporarily only during the evaluation of the considered condition. Given a configuration $\mathit{Conf} = E, \sigma, N, \mathcal{T}, \mu\mathcal{E}v$ or $\mathit{Conf} = E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ or $\mathit{Conf} = E, \mathcal{Q}, \mathcal{C}h$, we denote by E_{Conf} the environment E in configuration Conf. We define

$$\begin{aligned}
&\mathit{Defined}^{\mathrm{Fut}}(E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v) = \mathit{Defined}(\sigma, M)\\
&\mathit{Defined}^{\mathrm{Fut}}(E, \mathcal{Q}, \mathcal{C}h) = \biguplus_{(\sigma,Q)\in\mathcal{Q}} \mathit{Defined}(\sigma, Q)\\
&\mathit{Defined}^{\mathrm{Fut}}(E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v) = \mathit{Defined}(\sigma, P) \uplus \biguplus_{(\sigma,Q)\in\mathcal{Q}} \mathit{Defined}(\sigma, Q)\\
&\mathit{Defined}(\mathit{Conf}) = \mathrm{Dom}(E_{\mathit{Conf}}) \uplus \mathit{Defined}^{\mathrm{Fut}}(\mathit{Conf})\,.
\end{aligned}$$

Invariant 8 (Single definition, for executing games) The semantic configuration Conf (which can be $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v$ or $E, \mathcal{Q}, \mathcal{C}h$ or $E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$) satisfies Invariant 8 if and only if $\mathit{Defined}(\mathit{Conf})$ does not contain duplicate elements.

Lemma 9 *Let Tr be trace of Q_0. If Q_0 satisfies Invariant 1, then all semantic configurations in the derivation of Tr satisfy Invariant 8.*

Proof sketch We first show that, for all program points μ in Q_0, if $\mathrm{Dom}(\sigma) = I_\mu$ are the current replication indices at μ and the process or term Q at μ satisfies Invariant 1, then all elements of $\mathit{Defined}(\sigma, Q)$ are of the form $x[\widetilde{a}]$ where $x \in \mathrm{vardef}(Q)$ and $\mathrm{Im}(\sigma)$ is a prefix of $\widetilde{a}$. The proof proceeds by induction on Q. At the definition of a variable $x[\widetilde{i}]$, $x[\sigma(\widetilde{i})]$ is added to $\mathit{Defined}(\sigma, Q)$ and we have $x \in \mathrm{vardef}(Q)$; by Invariant 1, $\widetilde{i}$ are the current replication indices at that definition, so $\sigma(\widetilde{i}) = \mathrm{Im}(\sigma)$. All recursive calls $\mathit{Defined}(\sigma', {}^{\mu'}Q')$ consider an extension σ' of σ and a subprocess or subterm Q' of Q (so $\mathrm{vardef}(Q') \subseteq \mathrm{vardef}(Q)$) such that $\mathrm{Dom}(\sigma') = I_{\mu'}$ are the current replication indices at μ'.

Next, we show that, for all program points μ, if $\mathrm{Dom}(\sigma) = I_\mu$ are the current replication indices at μ and the process or term Q at μ satisfies Invariant 1, then $\mathit{Defined}(\sigma, Q)$ does not contain duplicate elements. The proof proceeds by induction on Q. All multiset unions in the computation of $\mathit{Defined}(\sigma, Q)$ are disjoint unions by the property above, because either they use different extensions of σ (case of replication) or they use disjoint variable definitions or subprocesses or subterms in the same branch of find, if, or get, which must define different variables by Invariant 1.

We show by induction on the derivations that, if $\mathit{Conf} \xrightarrow{p}_t \mathit{Conf}'$, then $\mathit{Defined}(\mathit{Conf}) \supseteq \mathit{Defined}(\mathit{Conf}')$ and for all semantic configurations Conf'' in the derivation of $\mathit{Conf} \xrightarrow{p}_t \mathit{Conf}'$, $\mathit{Defined}(\mathit{Conf}) \supseteq \mathit{Defined}(\mathit{Conf}'')$, and similarly with $\rightsquigarrow$ instead of $\xrightarrow{p}_t$.

The result follows: since Q_0 satisfies Invariant 1, $\mathit{Defined}(\sigma_0, Q_0)$ does not contain duplicate elements, where σ_0 is the empty mapping sequence. Then $\emptyset, \{(\sigma_0, Q_0)\}, \mathrm{fc}(Q_0)$ satisfies Invariant 8 and so do $\mathrm{reduce}(\emptyset, \{(\sigma_0, Q_0)\}, \mathrm{fc}(Q_0))$, $\mathrm{initConfig}(Q_0)$, and the other configurations of Tr. □

Corollary 2 *If Q_0 satisfies Invariant 1, then each variable that is not defined in a condition of* find *or* get *is defined at most once for each value of its array indices in a trace of Q_0.*

Proof Let x be the considered variable and Tr be the considered trace of Q_0. The only semantic rules that can add $x[\widetilde{a}]$ to the environment E are (NewT), (LetT), (FindT1), (GetT1), (New), (Let), (Find1), (Get1), and (Output). By Corollary 1, the target term or process of these rules is a subterm or subprocess of Q_0 up to renaming of channels. Hence, the target configuration Conf' of these rules is at some program point μ in Tr. By hypothesis, x is not defined in conditions of find or get, so μ is not inside the condition of find or get in Q_0, so by Lemma 5, the configuration Conf' is not in the derivation of an assumption of a rule for find or get.

In order to derive a contradiction, assume that two transitions $\mathit{Conf}_1 \xrightarrow{p_1}_{t_1} \mathit{Conf}'_1$ and $\mathit{Conf}_2 \xrightarrow{p_2}_{t_2} \mathit{Conf}'_2$ inside Tr define the same variable $x[\widetilde{a}]$.

- First case: one transition happens before the other, for instance $\mathit{Conf}'_1 \preceq_{\mathit{Tr}} \mathit{Conf}_2$. (The case $\mathit{Conf}'_2 \preceq_{\mathit{Tr}} \mathit{Conf}_1$ is symmetric.) Since $\mathit{Conf}_1 \xrightarrow{p_1}_{t_1} \mathit{Conf}'_1$ defines $x[\widetilde{a}]$, we have $x[\widetilde{a}] \in \mathrm{Dom}(E_{\mathit{Conf}'_1})$. Since Conf'_1 is not in the derivation of an assumption of a rule for find or get, by Lemma 3, E_{Conf_2} extends $E_{\mathit{Conf}'_1}$, so $x[\widetilde{a}] \in \mathrm{Dom}(E_{\mathit{Conf}_2})$. Moreover, since $\mathit{Conf}_2 \xrightarrow{p_2}_{t_2} \mathit{Conf}'_2$ defines $x[\widetilde{a}]$, we have $x[\widetilde{a}] \in \mathit{Defined}^{\mathrm{Fut}}(\mathit{Conf}_2)$, by inspecting all rules that add elements to the environment. Therefore $\mathit{Defined}(\mathit{Conf}_2) = \mathrm{Dom}(E_{\mathit{Conf}_2}) \uplus \mathit{Defined}^{\mathrm{Fut}}(\mathit{Conf}_2)$ contains twice $x[\widetilde{a}]$. Contradiction with Invariant 8.

- Second case: the transitions cannot be ordered. By definition of $\preceq_{\mathit{Tr}}$, this can happen only when a semantic rule uses several derivations for its assumptions, which happens only in rules for find or get. Contradiction.

That concludes the proof. □

Variables defined in conditions of get may be defined several times, once for each element of the table that is tested. Variables defined in conditions of find may be defined several times in case the same variable is used in several branches of the same find. (We use the indices of the find as indices of the variables defined in the condition, so when we evaluate several times the condition of a certain branch of a find, we use variables with different indices.) In Section 2.6, we define properties that exclude these situations (Properties 4 and 5), and we prove in Lemma 28 that every variable is defined at most once for each value of its indices when these properties are satisfied.

2.4.4 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 9 (Defined variables, for executing games) The semantic configuration $E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ satisfies Invariant 9 if and only if every occurrence of a variable access $x[M_1, \dots, M_m]$ in (σ, P) or $\mathcal{Q}$ is either

1. present in $\mathrm{Dom}(E)$: if $x[M_1, \dots, M_m]$ occurs in a process P' for $(\sigma', P') \in \{(\sigma, P)\} \cup \mathcal{Q}$, then for all $j \leq m$, $E, \sigma', M_j \Downarrow a_j$ and $x[a_1, \dots, a_m] \in \mathrm{Dom}(E)$;
2. or syntactically under the definition of $x[M_1, \dots, M_m]$ (in which case for all $j \leq m$, M_j is a constant or variable replication index);
3. or in a $\mathsf{defined}$ condition in a find process or term;
4. or in M'_j in a process or term of the form $\mathsf{find}\ (\bigoplus_{j=1}^{m''} \widetilde{u_j}[\widetilde{i}] = \widetilde{i_j} \leq \widetilde{n_j}\ \mathsf{suchthat}\ \mathsf{defined}(M'_{j1}, \dots, M'_{jl_j}) \wedge M'_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P$ where for some $k \leq l_j$, $x[M_1, \dots, M_m]$ is a subterm of M'_{jk}.
5. or in P_j in a process or term of the form $\mathsf{find}\ (\bigoplus_{j=1}^{m''} \widetilde{u_j}[\widetilde{i}] = \widetilde{i_j} \leq \widetilde{n_j}\ \mathsf{suchthat}\ \mathsf{defined}(M'_{j1}, \dots, M'_{jl_j}) \wedge M'_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P$ where for some $k \leq l_j$, there is a subterm N of M'_{jk} such that $N\{\widetilde{u_j}[\widetilde{i}]/\widetilde{i_j}\} = x[M_1, \dots, M_m]$.

Similarly, $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v$ satisfies Invariant 9 if and only if every occurrence of a variable access $x[M_1, \dots, M_m]$ in M either is present in $\mathrm{Dom}(E)$ (for all $j \leq m$, $E, \sigma, M_j \Downarrow a_j$ and $x[a_1, \dots, a_m] \in \mathrm{Dom}(E)$) or satisfies one of the last four conditions above.

$E, \sigma, \mathsf{defined}(M'_1, \dots, M'_l) \wedge M, \mathcal{T}, \mu\mathcal{E}v$ satisfies Invariant 9 if and only if every occurrence of a variable access $x[M_1, \dots, M_m]$ in M either is a subterm of $M'_1, \dots, M'_l$, or is present in $\mathrm{Dom}(E)$ (for all $j \leq m$, $E, \sigma, M_j \Downarrow a_j$ and $x[a_1, \dots, a_m] \in \mathrm{Dom}(E)$) or satisfies one of the last four conditions above.

Recall that, by Invariants 2 and 5, the terms of all variable accesses $x[M_1, \dots, M_m]$ are simple. That is why we can evaluate them by $E, \sigma', M_j \Downarrow a_j$.

Lemma 10 *If Q_0 satisfies Invariant 2, then* $\mathrm{initConfig}(Q_0)$ *satisfies Invariant 9.*

Lemma 11 *Let M be a simple term. If $E, \sigma, M \Downarrow a$, then for all subterms $x[M_1, \dots, M_m]$ of M, for all $j' \leq m$, $E, \sigma, M_{j'} \Downarrow a_{j'}$ and $x[a_1, \dots, a_m]$ is in* $\mathrm{Dom}(E)$*.*

Proof sketch By induction on M. □

Lemma 12 *Let N, M be simple terms. If $E, \sigma[i \mapsto a'], N \Downarrow a$ and $E, \sigma, M \Downarrow a'$, then we have $E, \sigma, N\{M/i\} \Downarrow a$.*

Proof sketch By induction on N. □

Lemma 13 *If $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', \sigma', M', \mathcal{T}', \mu\mathcal{E}v'$ and $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v$ satisfies Invariant 9, then so does $E', \sigma', M', \mathcal{T}', \mu\mathcal{E}v'$.*

If $E, \sigma, \mathsf{defined}(M_1, \dots, M_m) \wedge M, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', \sigma', M', \mathcal{T}', \mu\mathcal{E}v'$ and $E, \sigma, \mathsf{defined}(M_1, \dots, M_m) \wedge M, \mathcal{T}, \mu\mathcal{E}v$ satisfies Invariant 9, then so does $E', \sigma', M', \mathcal{T}', \mu\mathcal{E}v'$.

If $E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', (\sigma', P'), \mathcal{Q}', \mathcal{C}h', \mathcal{T}', \mu\mathcal{E}v'$ and $E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ satisfies Invariant 9, then so does $E', (\sigma', P'), \mathcal{Q}', \mathcal{C}h', \mathcal{T}', \mu\mathcal{E}v'$.

Moreover, if the rules that define $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', \sigma', M', \mathcal{T}', \mu\mathcal{E}v'$ (resp. $E, \sigma, \mathsf{defined}(M_1, \dots, M_m) \wedge M, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', \sigma', M', \mathcal{T}', \mu\mathcal{E}v'$ or $E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', (\sigma', P'), \mathcal{Q}', \mathcal{C}h', \mathcal{T}', \mu\mathcal{E}v'$) require as assumption $E'', \sigma'', M'', \mathcal{T}'', \mu\mathcal{E}v'' \xrightarrow{p}_t \dots$ or $E'', \sigma'', \mathsf{defined}(M''_1, \dots, M''_m) \wedge M'', \mathcal{T}'', \mu\mathcal{E}v'' \xrightarrow{p}_t \dots$, and the initial configuration $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v$ (resp. $E, \sigma, \mathsf{defined}(M_1, \dots, M_m) \wedge M, \mathcal{T}, \mu\mathcal{E}v$ or $E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$) satisfies Invariant 9, then so does the initial configuration of the assumption, $E'', \sigma'', M'', \mathcal{T}'', \mu\mathcal{E}v''$ or $E'', \sigma'', \mathsf{defined}(M''_1, \dots, M''_m) \wedge M'', \mathcal{T}'', \mu\mathcal{E}v''$.

Proof sketch The proof proceeds by induction following the definition of $\xrightarrow{p}_t$. We just sketch the main arguments.

If $x[M_1, \dots, M_m]$ is in the second case of Invariant 9, and we execute the definition of $x[M_1, \dots, M_m]$, then for all $j \leq m$, M_j is a variable replication index and $x[\sigma(M_1), \dots, \sigma(M_m)]$ is added to $\mathrm{Dom}(E)$ by rules (NewT), (LetT), (FindT1), (GetT1), (New), (Let), (Find1), (Output), or (Get1) so it moves to the first case of Invariant 9.

If $x[M_1, \dots, M_m]$ is in the third case of Invariant 9, and we execute the corresponding find, this access to x simply disappears.

If $x[M_1, \dots, M_m]$ is in the fourth case of Invariant 9, and we execute the find, then $x[M_1, \dots, M_m]$ is a subterm of M'_{jk} for some $j \leq m''$ and $k \leq l_j$. Therefore, the initial configuration of the assumption $E, \sigma[\widetilde{i_j} \mapsto \widetilde{a}], D_j \wedge M'_j, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p_{k'}}{}^{*}_{t_{k'}} E'', \sigma', r_{k'}, \mathcal{T}, \mu\mathcal{E}v$ with $D_j \wedge M'_j = \mathsf{defined}(M'_{j1}, \dots, M'_{jl_j}) \wedge M'_j$ and $\sigma' = \sigma[\widetilde{i_j} \mapsto \widetilde{a}]$ also satisfies Invariant 9. In case this assumption is reduced by (DefinedYes), we have $E, \sigma', M'_{jk}, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1}{}^{*} E, \sigma', a_{jk}, \mathcal{T}, \mu\mathcal{E}v$, that is, $E, \sigma', M'_{jk} \Downarrow a_{jk}$. Therefore, by Lemma 11, for all $j' \leq m$, $E, \sigma', M_{j'} \Downarrow a_{j'}$ and $x[a_1, \dots, a_m]$ is in $\mathrm{Dom}(E)$. So $x[M_1, \dots, M_m]$ moves to the first case of Invariant 9 in $E, \sigma', M'_j, \mathcal{T}, \mu\mathcal{E}v$ after reduction by (DefinedYes).

If $x[M_1, \dots, M_m]$ is in the last case of Invariant 9, and we execute the find selecting branch j by (FindT1) or (Find1), then there is a subterm N of M'_{jk} for some $k \leq l_j$ such that $N\{\widetilde{u_j}[\widetilde{i}]/\widetilde{i_j}\} = x[M_1, \dots, M_m]$. By hypothesis of (FindT1) or (Find1), we have $E, \sigma[\widetilde{i_j} \mapsto \widetilde{a'}], D_j \wedge M'_j, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p_{k'}}{}^{*}_{t_{k'}} E, \sigma', r_{k'}, \mathcal{T}, \mu\mathcal{E}v$ where $r_{k'} = \mathrm{true}$, $v_0 = (j, \widetilde{a'}) \in S$, $D_j \wedge M'_j = \mathsf{defined}(M'_{j1}, \dots, M'_{jl_j}) \wedge M'_j$, and $\sigma' = \sigma[\widetilde{i_j} \mapsto \widetilde{a'}]$. This assumption cannot reduce by (DefinedNo) because the result is true, so it reduces by (DefinedYes). Therefore, we have $E, \sigma', M'_{jk}, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{1}{}^{*} E, \sigma', a, \mathcal{T}, \mu\mathcal{E}v$ for some a, that is, $E, \sigma', M'_{jk} \Downarrow a$. The term $N = x[N_1, \dots, N_m]$ is a subterm of M'_{jk}. Therefore, by Lemma 11, for all $j' \leq m$, $E, \sigma', N_{j'} \Downarrow a_{j'}$ and $x[a_1, \dots, a_m]$ is in $\mathrm{Dom}(E)$. Moreover, the resulting environment E' is an extension of E, so a fortiori for all $j' \leq m$, $E', \sigma', N_{j'} \Downarrow a_{j'}$ and $x[a_1, \dots, a_m]$ is in $\mathrm{Dom}(E')$. We have for all $j' \leq m$, $M_{j'} = N_{j'}\{\widetilde{u_j}[\widetilde{i}]/\widetilde{i_j}\}$, $E'(\widetilde{u_j}[\widetilde{i}]) = \widetilde{a'}$, and $\sigma'(\widetilde{i_j}) = \widetilde{a'}$, so by Lemma 12, for all $j' \leq m$, $E', \sigma, M_{j'} \Downarrow a_{j'}$ 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 9.

In all other cases, the situation remains unchanged. For context rules, this is because, in the allowed contexts, the hole is never under a defined condition. □

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 it is present in Dom(E): for all $j \leq m$, $E, \sigma, M_j \Downarrow a_j$ 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.

2.4.5 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 use the following definitions:

- $\mathcal{E} \vdash E$ if and only if $E(x[a_1, \dots, a_m]) = a$ implies $\mathcal{E}(x) = T_1 \times \dots \times T_m \rightarrow T$ with for all $j \leq m$, $a_j \in T_j$ and $a \in T$.
- We define $\mathcal{E} \vdash P$, $\mathcal{E} \vdash Q$, and $\mathcal{E} \vdash M : T$ as in Section 2.3, with the additional rules $\mathcal{E} \vdash a : T$ if and only if $a \in T$, $\mathcal{E} \vdash \mathsf{event\_abort}\ (\mu, \widetilde{a}) : e : T$ for all T, and $\mathcal{E} \vdash \mathsf{abort}$. (These rules are useful to type evaluated terms and processes.)
- $\mathcal{E} \vdash (\sigma, P)$ if and only if $\mathcal{E}[i_1 \mapsto [1, n_1], \dots, i_m \mapsto [1, n_m]] \vdash P$ and for all $j \leq m$, $\sigma(i_j) \in [1, n_j]$ for some $n_1, \dots, n_m$, where $\mathrm{Dom}(\sigma) = [i_1, \dots, i_m]$. The judgments $\mathcal{E} \vdash (\sigma, Q)$ and $\mathcal{E} \vdash (\sigma, M) : T$ are defined in the same way.
- $\mathcal{E} \vdash \mathcal{T}$ if and only if $\mathit{Tbl}(a_1, \dots, a_m) \in \mathcal{T}$ implies $\mathit{Tbl} : T_1 \times \dots \times T_m$ with for all $j \leq m$, $a_j \in T_j$.
- $\mathcal{E} \vdash \mu\mathcal{E}v$ if and only if $(\mu, \widetilde{a}) : e(a_1, \dots, a_m) \in \mu\mathcal{E}v$ implies $e : T_1 \times \dots \times T_m$ with for all $j \leq m$, $a_j \in T_j$.
- $\mathcal{E} \vdash E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ if and only if $\mathcal{E} \vdash E$, $\mathcal{E} \vdash (\sigma, P)$, $\mathcal{E} \vdash \mathcal{T}$, $\mathcal{E} \vdash \mu\mathcal{E}v$, and for all $(\sigma', Q) \in \mathcal{Q}$, $\mathcal{E} \vdash (\sigma', Q)$.
- $\mathcal{E} \vdash E, \mathcal{Q}, \mathcal{C}h$ if and only if $\mathcal{E} \vdash E$ and for all $(\sigma', Q) \in \mathcal{Q}$, $\mathcal{E} \vdash (\sigma', Q)$.
- $\mathcal{E} \vdash E, \sigma, M : T, \mathcal{T}, \mu\mathcal{E}v$ if and only if $\mathcal{E} \vdash E$, $\mathcal{E} \vdash (\sigma, M) : T$, $\mathcal{E} \vdash \mathcal{T}$, and $\mathcal{E} \vdash \mu\mathcal{E}v$.

Lemma 14 *If $\mathcal{E} \vdash E, \sigma, M : T, \mathcal{T}, \mu\mathcal{E}v$ and $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', \sigma', M', \mathcal{T}', \mu\mathcal{E}v'$, then $\mathcal{E} \vdash E', \sigma', M' : T, \mathcal{T}', \mu\mathcal{E}v'$.*

So, $\mathcal{E} \vdash E, \sigma, M : T, \mathcal{T}, \mu\mathcal{E}v$ and $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}{}^*_t E', \sigma', a, \mathcal{T}', \mu\mathcal{E}v'$, then $\mathcal{E} \vdash E', \sigma', a : T, \mathcal{T}', \mu\mathcal{E}v'$.

Proof sketch By induction on the derivation of $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', \sigma', M', \mathcal{T}', \mu\mathcal{E}v'$. □

Lemma 15 *If $\mathcal{E} \vdash E, \mathcal{Q}, \mathcal{C}h$ and $E, \mathcal{Q}, \mathcal{C}h \rightsquigarrow E', \mathcal{Q}', \mathcal{C}h'$, then $\mathcal{E} \vdash E', \mathcal{Q}', \mathcal{C}h'$.*

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

Proof sketch By cases on the derivation of $E, \mathcal{Q}, \mathcal{C}h \rightsquigarrow E', \mathcal{Q}', \mathcal{C}h'$. In the case of the replication, we have $\mathcal{E} \vdash (\sigma, !^{i \leq n} Q)$, so $\mathcal{E}[i_1 \mapsto [1, n_1], \dots, i_m \mapsto [1, n_m]] \vdash !^{i \leq n} Q$ and for all $j \leq m$, $\sigma(i_j) \in [1, n_j]$ for some $n_1, \dots, n_m$, where $\mathrm{Dom}(\sigma) = [i_1, \dots, i_m]$. By (TRepl), $\mathcal{E}[i_1 \mapsto [1, n_1], \dots, i_m \mapsto [1, n_m], i \mapsto [1, n]] \vdash Q$, so $\mathcal{E} \vdash (\sigma[i \mapsto a], Q)$ for $a \in [1, n]$. In the case of the input, we use Lemma 14. □

Lemma 16 *If $\mathcal{E} \vdash Q_0$, then $\mathcal{E} \vdash \mathrm{initConfig}(Q_0)$.*

Proof sketch By Lemma 15 and the previous definitions. □

Lemma 17 (Subject reduction) *If $\mathcal{E} \vdash E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ and $E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', (\sigma', P'), \mathcal{Q}', \mathcal{C}h', \mathcal{T}', \mu\mathcal{E}v'$, then $\mathcal{E} \vdash E', (\sigma', P'), \mathcal{Q}', \mathcal{C}h', \mathcal{T}', \mu\mathcal{E}v'$.*

Proof sketch By cases on the derivation of $E, P, \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', (\sigma', P'), \mathcal{Q}', \mathcal{C}h', \mathcal{T}', \mu\mathcal{E}v'$, using Lemmas 14 and 15. □

Moreover, if the rules that define $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', \sigma', M', \mathcal{T}', \mu\mathcal{E}v'$ (resp. $E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', (\sigma', P'), \mathcal{Q}', \mathcal{C}h', \mathcal{T}', \mu\mathcal{E}v'$) require as assumption $E'', \sigma'', M'', \mathcal{T}'', \mu\mathcal{E}v'' \xrightarrow{p}_t \dots$ and the initial configuration is well-typed $\mathcal{E} \vdash E, \sigma, M : T, \mathcal{T}, \mu\mathcal{E}v$ (resp. $\mathcal{E} \vdash E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$) then so is the initial configuration of the assumption, that is, there exists T'' such that $\mathcal{E} \vdash E'', \sigma'', M'' : T'', \mathcal{T}'', \mu\mathcal{E}v''$.

As an immediate consequence of Lemmas 16, 17, and 14 and the observation above, we obtain: if Q_0 satisfies Invariant 7, then in traces of Q_0, the tests $a \in T$ in rules (LetT) and (Let) and $\forall j \leq m, a_j \in T_j$ in rule (Fun) always succeed. Moreover, in rules (NewT) and (New), we always have that T is *fixed*, *bounded*, or *nonuniform*. In rules (IfT1), (IfT2), (If1), and (If2), the condition is in *bool* $= \{\text{false}, \text{true}\}$ (when it is a value, not an abort event value), so the condition $a \neq \text{true}$ is equivalent to $a = \text{false}$. In the rules for find, we have $r_k \in \{\text{false}, \text{true}\}$ when r_k is a value (not an abort event value). In the rules (InsertT), (GetTE), (GetT1), (GetT2), (Insert), (GetE), (Get1), and (Get2), we have $a_j \in T_j$ for $j \leq l$, where $\mathit{Tbl} : T_1 \times \ldots \times T_l$. In the rules (EventT) and (Event), we have $a_j \in T_j$ for $j \leq l$, where $e : T_1 \times \ldots \times T_l$.

2.5 Subset for the Initial Game

The variables are always defined with the current replication indices $\widetilde{i}$, so we omit them, writing x for $x[\widetilde{i}]$; they are implicitly added by CryptoVerif. When a variable is used with the current replication indices at its definition, we can also omit the indices.

Along similar lines, the channels c are used without indices, and the current replication indices are implicitly added by CryptoVerif. This allows the adversary the select to which copy of processes it sends messages. The construct newChannel cannot occur in games manipulated by CryptoVerif. It is used only inside proofs. The grammar of the resulting calculus is summarized in Figure 11.

We recommend using the constructs get and insert to manage key tables, instead of find or if with defined conditions. When no find nor if with defined conditions occurs in the game, by Invariant 2, all accesses to variable x are of the form $x[\widetilde{i}]$ where $\widetilde{i}$ are the current replication indices at the definition of x. Such accesses are simply abbreviated as x. Variables can then be considered as ordinary variables instead of arrays, since we only access the array cell at the current replication indices. This choice has several other advantages:

- Tables with get/insert are closer to lists usually used by cryptographers than find, they should be easier to understand for the user.
- Tables are supported by the symbolic protocol verifier ProVerif while find is not. Similarly, ProVerif does not support channels with indices. So avoiding find and channels with indices allows us to have a language compatible with ProVerif.
- Our compiler that translates CryptoVerif specifications into OCaml implementations [28] does not support find, because tables with get/insert are also much easier to implement than find.

$M, N ::=$	terms
$\quad i$	replication index
$\quad x[M_1, \dots, M_m]$	variable access
$\quad f(M_1, \dots, M_m)$	function application
$\quad \mathsf{new}\ x : T; N$	random number
$\quad \mathsf{let}\ p = M\ \mathsf{in}\ N\ \mathsf{else}\ N'$	assignment (pattern-matching)
$\quad \mathsf{let}\ x : T = M\ \mathsf{in}\ N$	assignment
$\quad \mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ N'$	conditional
$\quad \mathsf{find}[\mathit{unique}?]\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}\ \mathsf{suchthat}$	
$\quad\quad \mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M'_j\ \mathsf{then}\ N_j)\ \mathsf{else}\ N'$	array lookup
$\quad \mathsf{insert}\ \mathit{Tbl}(M_1, \dots, M_l); N$	insert in table
$\quad \mathsf{get}[\mathit{unique}?]\ \mathit{Tbl}(p_1, \dots, p_l)\ \mathsf{suchthat}\ M\ \mathsf{in}\ N\ \mathsf{else}\ N'$	get from table
$\quad \mathsf{event}\ e(M_1, \dots, M_l); N$	event
$\quad \mathsf{event\_abort}\ e$	event e and abort
$p ::=$	pattern
$\quad x : T$	variable
$\quad f(p_1, \dots, p_m)$	function application
$\quad =M$	comparison with a term
$Q ::=$	input process
$\quad 0$	nil
$\quad Q \mid Q'$	parallel composition
$\quad !^{i \leq n} Q$	replication n times
$\quad c(p); P$	input
$P ::=$	output process
$\quad \overline{c}\langle N \rangle; Q$	output
$\quad \mathsf{new}\ x : T; P$	random number
$\quad \mathsf{let}\ p = M\ \mathsf{in}\ P\ \mathsf{else}\ P'$	assignment
$\quad \mathsf{if}\ M\ \mathsf{then}\ P\ \mathsf{else}\ P'$	conditional
$\quad \mathsf{find}[\mathit{unique}?]\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}\ \mathsf{suchthat}$	
$\quad\quad \mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P$	array lookup
$\quad \mathsf{insert}\ \mathit{Tbl}(M_1, \dots, M_l); P$	insert in table
$\quad \mathsf{get}[\mathit{unique}?]\ \mathit{Tbl}(p_1, \dots, p_l)\ \mathsf{suchthat}\ M\ \mathsf{in}\ P\ \mathsf{else}\ P'$	get from table
$\quad \mathsf{event}\ e(M_1, \dots, M_l); P$	event
$\quad \mathsf{event\_abort}\ e$	event e and abort
$\quad \mathsf{yield}$	end

Figure 11: Subset of the calculus for the initial game

We can define processes by macros: let $pid(x_1 : T_1, \dots, x_m : T_m) = P$ or let $qid(x_1 : T_1, \dots, x_m : T_m) = Q$. If a process $pid(M_1, \dots, M_m)$ occurs in the initial game, CryptoVerif verifies that $M_1, \dots, M_m$ are of types $T_1, \dots, T_m$ respectively, and replaces $pid(M_1, \dots, M_m)$ with the expansion $P\{M_1/x_1, \dots, M_m/x_m\}$.

We can also define functions by macros: letfun $f(x_1 : T_1, \dots, x_m : T_m) = M$. If a term $f(M_1, \dots, M_m)$ occurs in the initial game, CryptoVerif verifies that $M_1, \dots, M_m$ are of types $T_1, \dots, T_m$ respectively, and replaces $f(M_1, \dots, M_m)$ with the expansion $M\{M_1/x_1, \dots, M_m/x_m\}$.

In the initial game, all bound variables with several incompatible definitions (different indices, different types, or variables defined in the same branch of a test) as well as variables declared without an explicit type are not allowed to occur in V nor in defined conditions of find or if and are renamed to distinct names, so that Invariant 1 is satisfied for these variables. The condition on input channels in Invariant 5 is always satisfied by definition of the language. CryptoVerif checks the rest of the invariants.

2.6 Subsets used inside the Sequence of Games

During the computation of the sequence of games, several properties are used by CryptoVerif, either required by some game transformations or guaranteed by others. We summarize them in this section.

Property 1 No function returns values of interval types. The types of values chosen by new $x[\widetilde{i}] : T$ are not interval types. The type T of the sent message in the (TOut) rule and of the receiving pattern in the (TIn) rule are not interval types.

This property is satisfied by all games manipulated by CryptoVerif, but not by processes that model the adversary. Combined with Invariants 7, 2, and 5, it implies that the terms of variable accesses $x[M_1, \dots, M_m]$ contain only replication indices and variables. (Tuples, events, and tables can take interval types as arguments. The constraint on inputs and outputs could probably be relaxed.)

For processes that model security assumptions on primitives, the receiving variable can be of an interval type. (This is used for instance to specify the computational Diffie-Hellman assumption; see Section 5.2.)

Property 2 The newChannel c construct does not appear in games.

Property 3 The indices of channels are always the current replication indices.

These properties are also satisfied by all games manipulated by CryptoVerif, but not by processes that model the adversary.

Property 4 The constructs insert and get do not occur in the game.

This property is not valid in the initial game, but it is in all other games of the sequence produced by CryptoVerif. The very first game transformation applied by CryptoVerif, **expand_tables**, encodes insert and get using find (see Section 5.1.2). The constructs insert and get are never introduced by subsequent game transformations, so this property remains valid in the rest of the sequence.

Property 5 The variables defined in conditions of find have pairwise distinct names.

$M, N ::=$	terms
$\quad i$	replication index
$\quad x[M_1, \dots, M_m]$	variable access
$\quad f(M_1, \dots, M_m)$	function application
$FC ::=$	find condition
$\quad M$	term
$\quad \mathsf{new}\ x[\widetilde{i}] : T; FC$	random number
$\quad \mathsf{let}\ p = M\ \mathsf{in}\ FC\ \mathsf{else}\ FC'$	assignment (pattern-matching)
$\quad \mathsf{let}\ x[\widetilde{i}] : T = M\ \mathsf{in}\ N$	assignment
$\quad \mathsf{if}\ M\ \mathsf{then}\ FC\ \mathsf{else}\ FC'$	conditional
$\quad \mathsf{find}[\mathit{unique}?]\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}\ \mathsf{suchthat}$	
$\qquad \mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge FC'_j\ \mathsf{then}\ FC_j)\ \mathsf{else}\ FC''$	array lookup
$\quad \mathsf{event\_abort}\ e$	event e and abort
$p ::=$	pattern
$\quad x[\widetilde{i}] : T$	variable
$\quad f(p_1, \dots, p_m)$	function application
$\quad {=}M$	comparison with a term
$Q ::=$	input process
$\quad 0$	nil
$\quad Q \mid Q'$	parallel composition
$\quad !^{i \leq n} Q$	replication n times
$\quad c[M_1, \dots, M_l](p); P$	input
$P ::=$	output process
$\quad \overline{c[M_1, \dots, M_l]}\langle N \rangle; Q$	output
$\quad \mathsf{new}\ x[\widetilde{i}] : T; P$	random number
$\quad \mathsf{let}\ p = M\ \mathsf{in}\ P\ \mathsf{else}\ P'$	assignment
$\quad \mathsf{if}\ M\ \mathsf{then}\ P\ \mathsf{else}\ P'$	conditional
$\quad \mathsf{find}[\mathit{unique}?]\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}\ \mathsf{suchthat}$	
$\qquad \mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge FC_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P$	array lookup
$\quad \mathsf{event}\ e(M_1, \dots, M_l); P$	event
$\quad \mathsf{event\_abort}\ e$	event e and abort
$\quad \mathsf{yield}$	end

Figure 12: Subset after game expansion

This property is enforced by the transformation **auto_SArename** (see Section 5.1.2) by renaming variables defined in conditions of find to distinct names. (This is easy since these variables do not have array accesses by Invariant 3.) Property 5 is required as a precondition by many game transformations, and may be broken by game transformations that duplicate code. Therefore, we apply **auto_SArename** after these game transformations.

Property 6 The terms M are simple except for conditions of find.

The grammar of the language taking into account this property as well as Properties 2 and 4 is shown in Figure 12. By Invariant 4, event does not occur in conditions of find, so event never occurs as term. Property 6 is enforced by the transformation **expand** (see Section 5.1.3) by converting other terms into processes. This transformation is applied on the initial game after **expand_tables**. Property 6 is broken by the cryptographic transformation of Section 5.2, so by default **expand** is called again after this transformation. Many game transformations require Property 6 as a precondition.

2.7 Security Properties, Indistinguishability

A context is a process containing a hole $[\,]$. An evaluation context C is a context built from $[\,]$, 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.

We write $\mathrm{event}(D)$ for the set of events that occur in the distinguisher D (i.e. are used by the distinguisher D). We write $\mathrm{event}(Q)$ for the set of events that occur in the process Q. We use similar notations for output processes, contexts, ... We write $\mathrm{event}(Q, Q')$ for $\mathrm{event}(Q) \cup \mathrm{event}(Q')$.

Definition 5 (Indistinguishability) Let Q and Q' be two processes, V a set of variables, and $\mathcal{E}$ a set of events. Assume that Q and Q' satisfy Invariants 1 to 7 with public variables V, and the variables of V are defined in Q and Q', with the same types.

An evaluation context C is said to be *acceptable* for Q with public variables V if and only if $\mathrm{var}(C) \cap \mathrm{var}(Q) \subseteq V$, $\mathrm{vardef}(C) \cap V = \emptyset$, C and Q do not use any common table, and $C[Q]$ satisfies Invariants 1 to 7 with public variables V.

We write $Q \approx_p^{V,\mathcal{E}} Q'$ when, for all evaluation contexts C acceptable for Q and Q' with public variables V and all distinguishers D that run in time at most t_D and such that $\mathrm{event}(D) \cap \mathrm{event}(Q, Q') \subseteq \mathcal{E}$, $|\Pr[C[Q] : D] - \Pr[C[Q'] : D]| \leq p(C, t_D)$.

This definition formalizes that the probability that algorithms C and D distinguish the games Q and Q' is at most $p(C, t_D)$. The probability p typically depends on the runtime of C and D, but may also depend on other parameters, such as the number of queries to each oracle made by C. That is why p takes as arguments the whole algorithm C and the runtime of D. More specifically:

Property 7 All probabilities computed by CryptoVerif are built from the following components by mathematical operations:

- the runtime of the context;
- the maximum number of outputs made by the context on each channel;
- the value of replication bounds, which is also determined from the number of outputs performed by the context on channels;

- the maximum length of the bitstring represented by a term, in particular a variable; this length may depend on messages output by the context; it is used only for unbounded types; for bounded types, we use the maximum length of the type instead;
- the maximum length of bitstring of a type T;
- the length of the result of a function, expressed as a function of the length of its arguments;
- the time of some action, expressed as a function of other elements of the formula;
- probability functions, used in particular to express the probability of breaking each primitive from other elements of the formula;
- the cardinal $|T|$ of a type T;
- the probability of collision between two random values of a type T, or between a random value and a value independent from that random value; these probabilities depend on the default distribution on the type D_T;
- ϵ_T, the distance between the default distribution D_T of type T and the uniform distribution;
- ϵ_{find}, where the distance between $D_{\mathsf{find}}(S)$ and the uniform distribution is $\epsilon_{\mathsf{find}}/2$.

Among the elements above, the first four depend on the context. In particular, probability formulas output by CryptoVerif do not depend on the variable, table, event names in the context. They also do not depend on the values of variables, but may depend on their length. For variables of bounded types, the probabilities do not depend at all on the values.

The set of events $\mathcal{E}$ corresponds to events that the adversary is allowed to observe. When $\mathcal{E} = \mathrm{event}(Q, Q')$, we omit it and write $Q \approx_p^V Q'$.

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'.) When V is empty, we omit it and write $Q \approx_p^{\mathcal{E}} Q'$.

When C is acceptable for Q with public variables V, and we transform Q into Q', we can rename the fresh variables of Q' (introduced by the game transformation) so that they do not occur in C. Then C is also acceptable for Q' with public variables V. (To establish this property, we use that the variables of V are defined in Q and Q', with the same types, so that, if $C[Q]$ is well-typed, then so is $C[Q']$.)

When C is acceptable for Q with public variables V, we have that $\mathrm{vardef}(C) \cap \mathrm{var}(Q) = \emptyset$, because $\mathrm{vardef}(C) \cap \mathrm{var}(Q) = \mathrm{vardef}(C) \cap \mathrm{var}(C) \cap \mathrm{var}(Q) \subseteq \mathrm{vardef}(C) \cap V = \emptyset$.

The following lemma is a straightforward consequence of Definition 5:

Lemma 18 *1. Reflexivity: $Q \approx_0^{V,\mathcal{E}} Q$.*

2. Symmetry: If $Q \approx_p^{V,\mathcal{E}} Q'$, then $Q' \approx_p^{V,\mathcal{E}} Q$.

3. Transitivity: If $Q \approx_p^{V,\mathcal{E}} Q'$ and $Q' \approx_{p'}^{V,\mathcal{E}} Q''$, then $Q \approx_{p+p'}^{V,\mathcal{E}} Q''$.

4. Application of a context: If $Q \approx_p^{V,\mathcal{E}} Q'$ and C is an evaluation context acceptable for Q and Q' with public variables V, then $C[Q] \approx_{p'}^{V',\mathcal{E}'} C[Q']$, where $p'(C', t_D) = p(C'[C[\,]], t_D)$, $V' \subseteq V \cup \mathrm{var}(C)$, and $\mathcal{E}' = \mathcal{E} \cup (\mathrm{event}(C) \setminus \mathrm{event}(Q, Q'))$.

Next, we introduce a notion related to indistinguishability that treats Shoup and non-unique events specially.

Definition 6 (Property preservation with introduction of events) Let Q and Q' be two processes and V a set of variables. Assume that Q and Q' satisfy Invariants 1 to 7 with public variables V, and the variables of V are defined in Q and Q', with the same types.

Let $D_{\text{false}}(\mathcal{E}v) = \text{false}$ for all $\mathcal{E}v$. Let $\mathsf{NonUnique}_Q = \bigvee\{e \mid [\mathsf{unique}_e] \text{ occurs in } Q\}$ and $\mathsf{NonUnique}_{Q,D} = \bigvee\{e \mid [\mathsf{unique}_e] \text{ occurs in } Q, e \notin D\}$, where D is a distinguisher consisting of a disjunction of Shoup and non-unique events, and we write $e \notin D$ to say that e does not occur in this disjunction. We have $\mathsf{NonUnique}_{Q,D} = \mathsf{NonUnique}_Q \wedge \neg D$.

We write $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : Q, D, \mathit{EvUsed} \xrightarrow{V}_p Q', D', \mathit{EvUsed}'$ when $\mathcal{D}$ is a set of distinguishers, $\mathcal{D}_{\mathsf{SNU}}$ is a set of Shoup and non-unique events in EvUsed, D and D' are distinguishers consisting of a disjunction of Shoup and non-unique events, the events that occur in Q or in D are in EvUsed, $\mathit{EvUsed} \subseteq \mathit{EvUsed}'$, the events that occur in Q' but not in Q are in EvUsed' but not in EvUsed, the events that occur in D' but not in D are in EvUsed' but not in EvUsed, and, for all evaluation contexts C acceptable for Q and Q' with public variables V that do not contain events in EvUsed', all distinguishers $D_0 \in \mathcal{D} \cup \{D_{\text{false}}\}$ that run in time at most t_{D_0}, all distinguishers D_1 that are disjunctions of events in $\mathcal{D}_{\mathsf{SNU}}$,

$$\begin{aligned}\Pr[C[Q] : (D_0 \vee D_1 \vee D) \wedge \neg\mathsf{NonUnique}_{Q,D_1\vee D}]\\ \leq \Pr[C[Q'] : (D_0 \vee D_1 \vee D') \wedge \neg\mathsf{NonUnique}_{Q',D_1\vee D'}] + p(C, t_{D_0})\end{aligned} \tag{1}$$

Intuitively, the events EvUsed are those used by CryptoVerif in the sequence of games until the game Q included, while the events EvUsed' are those used until Q'. Hence, EvUsed contains the events that occur in Q; EvUsed' contains EvUsed and the events that occur in Q'. The formula $D_0 \in \mathcal{D} \cup \{D_{\text{false}}\}$ corresponds to the initial query to prove: it is

- a correspondence distinguisher $\neg\varphi$ for a correspondence property (see Section 2.7.3);
- S or $\neg\overline{\mathsf{S}}$ for (one-session) secrecy (see Section 2.7.1) and bit secrecy (see Section 2.7.2);
- any distinguisher for indistinguishability;
- D_{false} when the initial query has already been proved, and only Shoup and non-unique events remain to be proved.

We need to specify precisely the distinguishers needed for the queries we want to prove, because some game transformations of CryptoVerif rely on that. For instance, **simplify** (Section 5.1.21) removes events that are not used by the queries.

The formula D_1 is a disjunction of Shoup and non-unique events that remain to be proved, both in Q and in Q'. These events are in $\mathcal{D}_{\mathsf{SNU}}$ and in EvUsed. The formula D is a disjunction of Shoup and non-unique events that remain to be proved in Q, while the formula D' is a disjunction of Shoup and non-unique events that remain to be proved in Q'. Hence, the events that occur in D and not in D' are events proved while transforming Q into Q'. ("Proving" an event means proving that this event has a negligible probability of occurring, and adding that probability to p.) In contrast, the events that occur in D' and not in D are fresh Shoup and non-unique events introduced during the transformation of Q into Q', and that will need to be proved later; hence these events are in EvUsed' but not in EvUsed. More generally, all events that occur in Q' but not in Q are fresh events introduced in the transformation of Q into Q', so they are in EvUsed' but not in EvUsed.

When there are no Shoup nor non-unique events, we have $D_1 = D = D' = D_{\text{false}}$ and $\mathsf{NonUnique}_{Q,D_1 \vee D} = \mathsf{NonUnique}_{Q',D_1 \vee D'} = D_{\text{false}}$, so the inequality (1) reduces to

$$\Pr[C[Q] : D_0] \leq \Pr[C[Q'] : D_0] + p(C, t_{D_0})$$

Using $\neg D_0$ instead of D_0, we obtain

$$1 - \Pr[C[Q] : D_0] \leq 1 - \Pr[C[Q'] : D_0] + p(C, t_{D_0})$$

so by combining the two, $|\Pr[C[Q] : D_0] - \Pr[C[Q'] : D_0]| \leq p(C, t_{D_0})$ as in the definition of indistinguishability. In the general case, the inequality (1) differs from this formula because (1) always counts the traces that execute Shoup and non-unique events that remain to be proved (these traces are always included in the probability by $D_1 \vee D$, resp. $D_1 \vee D'$; the probability of these events needs to be bounded), and never counts the traces that execute proved non-unique events (these traces are excluded by $\neg\mathsf{NonUnique}_{Q,D_1 \vee D}$, resp. $\neg\mathsf{NonUnique}_{Q',D_1 \vee D'}$; the probability of these events has already been bounded). We could also exclude traces that execute proved Shoup events, though it is less essential: Shoup events often simply disappear when they are proved, while non-unique events remain in the game. Excluding traces that execute proved non-unique events allows us to exploit that the corresponding find or get is unique in the transformation from Q to Q': intuitively, the traces in which the find or get is not unique are not counted, so they can be ignored. (Obviously, this point needs to be proved more precisely for each game transformation.)

We need to introduce a distinct event for each [unique_e] because not all find[unique_e] and get[unique_e] may be proved unique in the current process and because we need to distinguish the non-unique events that occur in Q from those that occur in the context C. However, once a [unique_e] is proved, its name does not matter: all such events are counted in $\mathsf{NonUnique}_{Q,D_1 \vee D}$, and that remains true in future game transformations, since events are never re-added to D_1 or D. Therefore, we can rename all such events to the same name. So we simply abbreviate [unique_e] by [unique] when event e is proved. The context C must not contain the events used in Q or Q' and more generally events in EvUsed'.

The formula (1) could also be written

$$\begin{aligned}\Pr[C[Q] &: (D_0 \wedge \neg\mathsf{NonUnique}_Q) \vee D_1 \vee D] \\ &\leq \Pr[C[Q'] : (D_0 \wedge \neg\mathsf{NonUnique}_{Q'}) \vee D_1 \vee D'] + p(C, t_{D_0})\end{aligned}$$

since $\mathsf{NonUnique}_{Q,D_1 \vee D} = \mathsf{NonUnique}_Q \wedge \neg(D_1 \vee D)$. The advantage of the latter formulation is that the dependency in D_1, D, D' is simpler, which we sometimes exploit in the proofs. However, its drawback is that it is less clear for which events the traces are always counted and for which ones they are never counted, because some events appear both in $\mathsf{NonUnique}_Q$ and in $D_1 \vee D$. That is why we chose formula (1), to make it clear that the traces that execute events in $D_1 \vee D$ are always counted and the traces that execute events in $\mathsf{NonUnique}_{Q,D_1 \vee D}$ are never counted.

When f is a function from sequences of events to sequences of events and D is a distinguisher, we define the distinguisher $D \circ f$ by $(D \circ f)(\mathcal{E}v) = D(f(\mathcal{E}v))$. In particular, when σ is a renaming of events, the distinguisher $D \circ \sigma^{-1}$ is defined by $(D \circ \sigma^{-1})(\mathcal{E}v) = D(\sigma^{-1}\mathcal{E}v)$. When D is defined as a logical formula, that corresponds to renaming the events in D. For instance, if $D = e_1 \vee \ldots \vee e_m$, then $D \circ \sigma^{-1} = \sigma e_1 \vee \ldots \vee \sigma e_m$. When $\mathcal{D}$ is a set of distinguishers, $\sigma\mathcal{D} = \{D \circ \sigma^{-1} \mid D \in \mathcal{D}\}$.

We write $\mathcal{D}_{\neg\mathcal{E}}$ for the set of distinguishers that do not use events in $\mathcal{E}$.

Lemma 19 *1. Link with indistinguishability:*

(a) *Suppose that $\mathcal{D}_{\mathsf{SNU}}$ is a set of Shoup events, D is a distinguisher consisting of a disjunction of Shoup events, Q, Q' do not contain non-unique events, the events that occur in Q, D, $\mathcal{D}$, or $\mathcal{D}_{\mathsf{SNU}}$ are in EvUsed, and the events that occur in Q' also occur in Q. If $Q \approx_p^{V,\mathcal{E}} Q'$, then $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : Q, D, \mathit{EvUsed} \xrightarrow{V}_p Q', D, \mathit{EvUsed}$, for all $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}}, D$ such that $\mathrm{event}(\mathcal{D}) \cap \mathit{EvUsed} \subseteq \mathcal{E}$, $\mathcal{D}_{\mathsf{SNU}} \subseteq \mathcal{E}$, $\mathrm{event}(D) \subseteq \mathcal{E}$.*

(b) *Suppose that Q and Q' do not contain non-unique events, the events that occur in Q are in EvUsed, and the events that occur in Q' also occur in Q. Then $\mathcal{D}_{\neg(\mathit{EvUsed}\setminus\mathcal{E})}, \emptyset : Q, D_{\mathrm{false}}, \mathit{EvUsed} \xrightarrow{V}_p Q', D_{\mathrm{false}}, \mathit{EvUsed}$ if and only if $Q \approx_p^{V,\mathcal{E}} Q'$.*

2. *Reflexivity: If $\mathcal{D}_{\mathsf{SNU}}$ is a set of Shoup and non-unique events in EvUsed, D is a distinguisher consisting of a disjunction of Shoup and non-unique events, and the events that occur in Q or in D are in EvUsed, then $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : Q, D, \mathit{EvUsed} \xrightarrow{V}_0 Q, D, \mathit{EvUsed}$.*

3. *Transitivity: If $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : Q, D, \mathit{EvUsed} \xrightarrow{V}_p Q', D', \mathit{EvUsed}'$ and $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : Q', D', \mathit{EvUsed}' \xrightarrow{V}_{p'} Q'', D'', \mathit{EvUsed}''$, then we have $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : Q, D, \mathit{EvUsed} \xrightarrow{V}_{p''} Q'', D'', \mathit{EvUsed}''$, where $p''(C, t_{D_0}) = p(C, t_{D_0}) + p'(C, t_{D_0})$.*

4. *Application of a context: If $\mathcal{D}_{\neg \mathit{EvUsed}'}, \mathcal{D}_{\mathsf{SNU}} : Q, D, \mathit{EvUsed} \xrightarrow{V}_p Q', D', \mathit{EvUsed}'$, σ is a renaming of the events in EvUsed' to events not in EvUsed^+, C is a context acceptable for σQ and $\sigma Q'$ with public variables V such that $\mathrm{event}(C) \subseteq \mathit{EvUsed}^+$, and $\mathcal{D}'_{\mathsf{SNU}}$ is a set of Shoup and non-unique events in EvUsed^+, then we have $\mathcal{D}_{\neg\sigma \mathit{EvUsed}'}, \sigma\mathcal{D}_{\mathsf{SNU}} \cup \mathcal{D}'_{\mathsf{SNU}} : C[\sigma Q], D \circ \sigma^{-1}, \sigma \mathit{EvUsed} \cup \mathit{EvUsed}^+ \xrightarrow{V'}_{p'} C[\sigma Q'], D' \circ \sigma^{-1}, \sigma \mathit{EvUsed}' \cup \mathit{EvUsed}^+$, where $p'(C', t_{D_0}) = p(\sigma^{-1}(C'[C[\,]]), t_{D_0})$, and $V' \subseteq V \cup \mathrm{var}(C)$.*

5. *Adding distinguishers: If $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : Q, D, \mathit{EvUsed} \xrightarrow{V}_p Q', D', \mathit{EvUsed}'$ and $e \in \mathcal{D}_{\mathsf{SNU}}$, then $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : Q, D \vee e, \mathit{EvUsed} \xrightarrow{V}_p Q', D' \vee e, \mathit{EvUsed}'$.*

6. *Removing distinguishers: If $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : Q, D, \mathit{EvUsed} \xrightarrow{V}_p Q', D', \mathit{EvUsed}'$, $\mathcal{D}' \subseteq \mathcal{D}$, and $\mathcal{D}'_{\mathsf{SNU}} \subseteq \mathcal{D}_{\mathsf{SNU}}$, then $\mathcal{D}', \mathcal{D}'_{\mathsf{SNU}} : Q, D, \mathit{EvUsed} \xrightarrow{V}_p Q', D', \mathit{EvUsed}'$*

Proof Property 1a: Given the hypothesis, $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : Q, D, \mathit{EvUsed} \xrightarrow{V}_p Q', D, \mathit{EvUsed}$ reduces to: for all evaluation contexts C acceptable for Q and Q' with public variables V that do not contain events in EvUsed, all distinguishers $D_0 \in \mathcal{D} \cup \{D_{\mathrm{false}}\}$ that run in time at most t_{D_0}, and all distinguishers D_1 that are disjunctions of events in $\mathcal{D}_{\mathsf{SNU}}$,

$$\Pr[C[Q] : D_0 \vee D_1 \vee D] \le \Pr[C[Q'] : D_0 \vee D_1 \vee D] + p(C, t_{D_0})$$

We have $\mathrm{event}(D_0 \vee D_1 \vee D) \cap \mathrm{event}(Q, Q') \subseteq \mathrm{event}(D_0 \vee D_1 \vee D) \cap \mathit{EvUsed} \subseteq \mathcal{E}$. Moreover, $D_0 \vee D_1 \vee D$ can be implemented in the same time as D_0 since evaluating $D_0 \vee D_1 \vee D$ can be done by setting the final result to true as soon as an event in $D_1 \vee D$ is executed, and evaluating D_0 otherwise. This does not take more time than evaluating D_0. So this inequality is a consequence of $Q \approx_p^{V,\mathcal{E}} Q'$.

Property 1b: Given the hypothesis, $\mathcal{D}_{\neg(\mathit{EvUsed}\setminus\mathcal{E})}, \emptyset : Q, D_{\mathrm{false}}, \mathit{EvUsed} \xrightarrow{V}_p Q', D_{\mathrm{false}}, \mathit{EvUsed}$ reduces to: for all evaluation contexts C acceptable for Q and Q' with public variables V that do not contain events in EvUsed, and all distinguishers $D_0 \in \mathcal{D}_{\neg(\mathit{EvUsed}\setminus\mathcal{E})}$ that run in time at most t_{D_0},

$$\Pr[C[Q] : D_0] \le \Pr[C[Q'] : D_0] + p(C, t_{D_0}),$$

that is,

$$\Pr[C[Q] : D_0] - \Pr[C[Q'] : D_0] \leq p(C, t_{D_0})\,. \tag{2}$$

We have $\mathrm{event}(D_0) \cap \mathrm{event}(Q, Q') \subseteq \mathrm{event}(D_0) \cap \mathit{EvUsed} \subseteq \mathcal{E}$. Therefore, $Q \approx_p^{V,\mathcal{E}} Q'$ implies $\mathcal{D}_{\neg(\mathit{EvUsed}\setminus\mathcal{E})}, \emptyset : Q, D_{\mathrm{false}}, \mathit{EvUsed} \xrightarrow{V}_p Q', D_{\mathrm{false}}, \mathit{EvUsed}$.

Conversely, assume $\mathcal{D}_{\neg(\mathit{EvUsed}\setminus\mathcal{E})}, \emptyset : Q, D_{\mathrm{false}}, \mathit{EvUsed} \xrightarrow{V}_p Q', D_{\mathrm{false}}, \mathit{EvUsed}$. Let C be an evaluation context C acceptable for Q and Q' with public variables V and D be a distinguisher such that $\mathrm{event}(D) \cap \mathrm{event}(Q, Q') \subseteq \mathcal{E}$. Let σ be a renaming of events in *EvUsed* to fresh events. Then σC does not contain events in *EvUsed*. Given a sequence of events $\mathcal{E}v$, let $f(\mathcal{E}v)$ be obtained by removing all events in $\mathit{EvUsed}\setminus\mathcal{E}$ from $\mathcal{E}v$ and, in the remaining sequence, renaming the events e in $\sigma\mathit{EvUsed}$ to $\sigma^{-1}(e)$. Let $D_0 = D \circ f$. By construction, $D_0 \in \mathcal{D}_{\neg(\mathit{EvUsed}\setminus\mathcal{E})}$. So by (2), we get

$$\Pr[(\sigma C)[Q] : D_0] - \Pr[(\sigma C)[Q'] : D_0] \leq p(C, t_{D_0})\,.$$

Since $\neg D_0 \in \mathcal{D}_{\neg(\mathit{EvUsed}\setminus\mathcal{E})}$ and runs in the same time as D_0,

$$1 - \Pr[(\sigma C)[Q] : D_0] - 1 + \Pr[(\sigma C)[Q'] : D_0] \leq p(C, t_{D_0})$$

so

$$|\Pr[(\sigma C)[Q] : D_0] - \Pr[(\sigma C)[Q'] : D_0|] \leq p(C, t_{D_0})\,.$$

Furthermore, each trace of $(\sigma C)[Q]$ with sequence of events $\mathcal{E}v$ corresponds to a trace of $C[Q]$ with the same probability and a sequence of events $\mathcal{E}v'$ equal to $f(\mathcal{E}v)$ plus some events not used by D, so $D_0(\mathcal{E}v) = D(f(\mathcal{E}v)) = D(\mathcal{E}v')$. Indeed, in a trace of $(\sigma C)[Q]$, if an event $e(\dots)$ is executed in Q, then it is executed by $C[Q]$ as well. If it is in $\mathcal{E}$, then it is left unchanged by f. If it is not in $\mathcal{E}$, then it is in $\mathrm{event}(Q) \setminus \mathcal{E} \subseteq \mathit{EvUsed} \setminus \mathcal{E}$, so it is removed by f; furthermore, this event is not used by D since $\mathrm{event}(D) \cap \mathrm{event}(Q) \subseteq \mathcal{E}$. If an event $e(\dots)$ is executed in σC, then either $e \in \sigma\mathit{EvUsed}$, $e'(\dots)$ is executed by $C[Q]$ with $e' = \sigma^{-1}(e)$, and f maps e to e'; or $e \notin \sigma\mathit{EvUsed}$, $e \notin \mathit{EvUsed}$, $e(\dots)$ is executed by $C[Q]$, and f leaves e unchanged. Therefore, $\Pr[(\sigma C)[Q] : D_0] = \Pr[C[Q] : D]$. We have a similar situation for Q' instead of Q, and D can be implemented in the same time as D_0, so

$$|\Pr[C[Q] : D] - \Pr[C[Q'] : D]| \leq p(C, t_D)$$

so $Q \approx_p^{V,\mathcal{E}} Q'$.

Property 2: Obvious.

Property 3: The events in Q or D are in *EvUsed*. We have $\mathit{EvUsed} \subseteq \mathit{EvUsed}' \subseteq \mathit{EvUsed}''$. The events that occur in Q'' but not in Q occur either in Q'' but not in Q' or in Q' but not in Q; in the former case, they are in $\mathit{EvUsed}'' \setminus \mathit{EvUsed}'$; in the latter case, they are in $\mathit{EvUsed}' \setminus \mathit{EvUsed}$; so in both cases they are in $\mathit{EvUsed}'' \setminus \mathit{EvUsed}$. The same reasoning applies for the events that occur in D'' but not in D.

Let C be any evaluation context acceptable for Q and Q'' with public variables V that does not contain events in EvUsed''. After renaming the variables of C that do not occur in Q and Q'' and the tables of C that do not occur in Q and Q'' so that they do not occur in Q', C is also acceptable for Q' with public variables V. Furthermore, by Property 7, this renaming does not change the probabilities. Since $\mathit{EvUsed}' \subseteq \mathit{EvUsed}''$, C does not contain events in EvUsed'.

Let $D_0 \in \mathcal{D} \cup \{D_{\mathrm{false}}\}$ that runs in time at most t_{D_0}. Let D_1 be a disjunction of events of

$\mathcal{D}_{\mathsf{SNU}}$. Then we have:

$$\begin{aligned}
&\Pr[C[Q] : (D_0 \vee D_1 \vee D)) \wedge \neg\mathsf{NonUnique}_{Q,D_1 \vee D}] \\
&\quad \leq \Pr[C[Q'] : (D_0 \vee D_1 \vee D') \wedge \neg\mathsf{NonUnique}_{Q',D_1 \vee D'}] + p(C, t_{D_0}) \\
&\quad \leq \Pr[C[Q''] : (D_0 \vee D_1 \vee D'') \wedge \neg\mathsf{NonUnique}_{Q'',D_1 \vee D''}] + p(C, t_{D_0}) + p'(C, t_{D_0}) \\
&\quad \leq \Pr[C[Q''] : (D_0 \vee D_1 \vee D'')) \wedge \neg\mathsf{NonUnique}_{Q'',D_1 \vee D''}] + p''(C, t_{D_0})
\end{aligned}$$

by definition of p''. Therefore, we have $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : Q, D, \mathit{EvUsed} \xrightarrow{V}_{p''} Q'', D'', \mathit{EvUsed}''$.

Property 4: $\sigma\mathcal{D}_{\mathsf{SNU}}$ is a set of Shoup and non-unique events in $\sigma\mathit{EvUsed} \subseteq \sigma\mathit{EvUsed} \cup \mathit{EvUsed}^+$, so $\sigma\mathcal{D}_{\mathsf{SNU}} \cup \mathcal{D}'_{\mathsf{SNU}}$ is a set of Shoup and non-unique events in $\sigma\mathit{EvUsed} \cup \mathit{EvUsed}^+$. The events that occur in $C[\sigma Q]$ are either in C or in σQ; the former case, they are in EvUsed^+ by hypothesis; in the latter case, they are also in $\sigma\mathit{EvUsed}$, since the events of Q are in EvUsed. So the events that occur in $C[\sigma Q]$ are in $\sigma\mathit{EvUsed} \cup \mathit{EvUsed}^+$. The events in $D \circ \sigma^{-1}$ are in $\sigma\mathit{EvUsed} \subseteq \sigma\mathit{EvUsed} \cup \mathit{EvUsed}^+$. We have $\mathit{EvUsed} \subseteq \mathit{EvUsed}'$, so $\sigma\mathit{EvUsed} \cup \mathit{EvUsed}^+ \subseteq \sigma\mathit{EvUsed}' \cup \mathit{EvUsed}^+$. The events that occur in $C[\sigma Q']$ but not in $C[\sigma Q]$ are in $\sigma Q'$ but not in σQ, so they are in $\sigma\mathit{EvUsed}' \setminus \sigma\mathit{EvUsed}$, so in $(\sigma\mathit{EvUsed}' \cup \mathit{EvUsed}^+) \setminus (\sigma\mathit{EvUsed} \cup \mathit{EvUsed}^+)$. Similarly, the events that occur in $D' \circ \sigma^{-1}$ but not in $D \circ \sigma^{-1}$ are in $\sigma\mathit{EvUsed}' \setminus \sigma\mathit{EvUsed}$, so they are in $(\sigma\mathit{EvUsed}' \cup \mathit{EvUsed}^+) \setminus (\sigma\mathit{EvUsed} \cup \mathit{EvUsed}^+)$.

Let C' be any evaluation context acceptable for $C[\sigma Q]$ and $C[\sigma Q']$ with public variables V' that does not contain events in $\sigma\mathit{EvUsed}' \cup \mathit{EvUsed}^+$. We rename the variables of C' not in V' so that they are not in V; by Property 7, this renaming does not change the probabilities. Then $\sigma^{-1}(C'[C[\,]])$ is an evaluation context acceptable for Q and Q' with public variables V. Indeed,

$$\begin{aligned}
\mathrm{var}(\sigma^{-1}(C'[C[\,]])) \cap \mathrm{var}(Q) &= (\mathrm{var}(C') \cup \mathrm{var}(C)) \cap \mathrm{var}(Q) \\
&\subseteq (V' \cup \mathrm{var}(C)) \cap \mathrm{var}(Q) \\
&\qquad \text{since } \mathrm{var}(C') \cap \mathrm{var}(Q) \subseteq \mathrm{var}(C') \cap \mathrm{var}(C[Q]) \subseteq V' \\
&\subseteq (V \cup \mathrm{var}(C)) \cap \mathrm{var}(Q) \qquad \text{since } V' \subseteq V \cup \mathrm{var}(C) \\
&\subseteq V \qquad \text{since } \mathrm{var}(C) \cap \mathrm{var}(Q) \subseteq V
\end{aligned}$$

We have similarly $\mathrm{var}(\sigma^{-1}(C'[C[\,]])) \cap \mathrm{var}(Q') \subseteq V$. We also have $\mathrm{vardef}(\sigma^{-1}(C'[C[\,]])) \cap V = (\mathrm{vardef}(C') \cap V) \cup (\mathrm{vardef}(C) \cap V) = \emptyset$ since $\mathrm{vardef}(C) \cap V = \emptyset$ because C is an acceptable evaluation context for σQ with public variables V and $\mathrm{vardef}(C') \cap V \subseteq \mathrm{vardef}(C') \cap V' = \emptyset$ because we have renamed the variables of C' not in V' so that they are not in V and C' is an acceptable evaluation context for $C[\sigma Q]$ and with public variables V'. Moreover, C and σQ do not use any common table, and C' and $C[\sigma Q]$ do not use any common table so a fortiori C' and σQ do not use any common table. Therefore, $C'[C[\,]]$ and σQ do not use any common table, so $\sigma^{-1}(C'[C[\,]])$ and Q do not use any common table. Similarly, $\sigma^{-1}(C'[C[\,]])$ and Q' do not use any common table. The context $\sigma^{-1}(C'[C[\,]])$ does not contain events in EvUsed', since C' and C do not contain events in $\sigma\mathit{EvUsed}'$, because C' does not contain events in $\sigma\mathit{EvUsed}' \cup \mathit{EvUsed}^+$ and the events of C are in EvUsed^+ which is disjoint from $\sigma\mathit{EvUsed}'$. (The renamings σ and σ^{-1} are bijections, so for instance σ maps the fresh events introduced by σ to EvUsed' and σ^{-1} maps EvUsed' to the fresh events introduced by σ.)

By using the property $\mathcal{D}_{\neg\mathit{EvUsed}'}, \mathcal{D}_{\mathsf{SNU}} : Q, D, \mathit{EvUsed} \xrightarrow{V}_p Q', D', \mathit{EvUsed}'$ with the context $\sigma^{-1}(C'[C[\,]])$, we get for any distinguishers $D_0 \in \mathcal{D}_{\neg\mathit{EvUsed}'} \cup \{D_{\mathrm{false}}\}$ that runs in time t_{D_0} and D_1 disjunction of events in $\mathcal{D}_{\mathsf{SNU}}$:

$$\begin{aligned}
&\Pr[\sigma^{-1}C'[\sigma^{-1}C[Q]] : (D_0 \vee D_1 \vee D)) \wedge \neg\mathsf{NonUnique}_{Q,D_1 \vee D}] \\
&\quad \leq \Pr[\sigma^{-1}C'[\sigma^{-1}C[Q']] : (D_0 \vee D_1 \vee D')) \wedge \neg\mathsf{NonUnique}_{Q',D_1 \vee D'}] + p(\sigma^{-1}(C'[C[\,]]), t_{D_0})
\end{aligned} \tag{3}$$

Let $D'_0 \in \mathcal{D}_{\neg\sigma EvUsed'} \cup \{D_{\text{false}}\}$ that runs in time at most t_{D_0}, Let D'_1 be a disjunction of events in $\sigma\mathcal{D}_{\mathsf{SNU}} \cup \mathcal{D}'_{\mathsf{SNU}}$. We can write D'_1 under the form $D'_1 = D'_2 \vee D'_3$ where D'_2 is a disjunction of events in $\sigma\mathcal{D}_{\mathsf{SNU}}$ and D'_3 is a disjunction of events in $\mathcal{D}'_{\mathsf{SNU}}$.

By applying (3) to $D_0 = (D'_0 \circ \sigma \vee D'_3 \circ \sigma) \wedge \neg\mathsf{NonUnique}_{\sigma^{-1}C, D'_3\circ\sigma}$, which uses events not in $EvUsed'$, and to $D_1 = D'_2 \circ \sigma$, we get

$$\begin{aligned}
&\Pr[\sigma^{-1}C'[\sigma^{-1}C[Q]] : (((D'_0 \circ \sigma \vee D'_3 \circ \sigma) \wedge \neg\mathsf{NonUnique}_{\sigma^{-1}C, D'_3\circ\sigma}) \vee D'_2 \circ \sigma \vee D))\\
&\quad \wedge \neg\mathsf{NonUnique}_{Q, D'_2\circ\sigma\vee D}]\\
&\quad \leq \Pr[\sigma^{-1}C'[\sigma^{-1}C[Q']] : (((D'_0 \circ \sigma \vee D'_3 \circ \sigma) \wedge \neg\mathsf{NonUnique}_{\sigma^{-1}C, D'_3\circ\sigma}) \vee D'_2 \circ \sigma \vee D'))\\
&\quad \wedge \neg\mathsf{NonUnique}_{Q', D'_2\circ\sigma\vee D'}] + p(\sigma^{-1}(C'[C[\,]]), t_{D'_0})
\end{aligned}$$

since D_0 can be implemented to run in the same time as D'_0. By applying σ, we have

$$\begin{aligned}
&\Pr[C'[C[\sigma Q]] : (((D'_0 \vee D'_3) \wedge \neg\mathsf{NonUnique}_{C, D'_3}) \vee D'_2 \vee D \circ \sigma^{-1})) \wedge \neg\mathsf{NonUnique}_{\sigma Q, D'_2\vee D\circ\sigma^{-1}}]\\
&\quad \leq \Pr[C'[C[\sigma Q']] : (((D'_0 \vee D'_3) \wedge \neg\mathsf{NonUnique}_{C, D'_3}) \vee D'_2 \vee D' \circ \sigma^{-1}))\\
&\quad \wedge \neg\mathsf{NonUnique}_{\sigma Q', D'_2\vee D'\circ\sigma^{-1}}] + p(\sigma^{-1}(C'[C[\,]]), t_{D'_0})
\end{aligned}$$

Since the events of C are in $EvUsed^+$, the events of $\mathsf{NonUnique}_{C, D'_3}$ are disjoint from those in $(D'_2 \vee D \circ \sigma^{-1})$, so $(\neg\mathsf{NonUnique}_{C, D'_3}) \vee (D'_2 \vee D \circ \sigma^{-1}) = \neg\mathsf{NonUnique}_{C, D'_3}$ and similarly $(\neg\mathsf{NonUnique}_{C, D'_3}) \vee (D'_2 \vee D' \circ \sigma^{-1}) = \neg\mathsf{NonUnique}_{C, D'_3}$. So we have

$$\begin{aligned}
&\Pr[C'[C[\sigma Q]] : (D'_0 \vee D'_3 \vee D'_2 \vee D \circ \sigma^{-1})) \wedge \neg\mathsf{NonUnique}_{C, D'_3} \wedge \neg\mathsf{NonUnique}_{\sigma Q, D'_2\vee D\circ\sigma^{-1}}]\\
&\quad \leq \Pr[C'[C[\sigma Q']] : (D'_0 \vee D'_3 \vee D'_2 \vee D' \circ \sigma^{-1})) \wedge \neg\mathsf{NonUnique}_{C, D'_3}\\
&\quad \wedge \neg\mathsf{NonUnique}_{\sigma Q', D'_2\vee D'\circ\sigma^{-1}}] + p(\sigma^{-1}(C'[C[\,]]), t_{D'_0})
\end{aligned}$$

that is

$$\begin{aligned}
&\Pr[C'[C[\sigma Q]] : (D'_0 \vee D'_1 \vee D \circ \sigma^{-1})) \wedge \neg\mathsf{NonUnique}_{C[\sigma Q], D'_1\vee D\circ\sigma^{-1}}]\\
&\quad \leq \Pr[C'[C[\sigma Q']] : (D'_0 \vee D'_1 \vee D' \circ \sigma^{-1})) \wedge \neg\mathsf{NonUnique}_{C[\sigma Q'], D'_1\vee D'\circ\sigma^{-1}}]\\
&\quad + p(\sigma^{-1}(C'[C[\,]]), t_{D'_0})
\end{aligned}$$

Properties 5 and 6: Obvious. □

When CryptoVerif transforms a game G into a game G', in most cases, we have $G \approx_p^{V,\mathcal{E}} G'$, where p is the probability difference coming from the transformation, and computed by CryptoVerif, which implies $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : G, D, EvUsed \xrightarrow{V}_p G', D, EvUsed$, for all $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}}, D$ such that $\text{event}(\mathcal{D}) \cap EvUsed \subseteq \mathcal{E}$, $\mathcal{D}_{\mathsf{SNU}} \subseteq \mathcal{E}$, $\text{event}(D) \subseteq \mathcal{E}$ by Lemma 19, Property 1a. However, there are exceptions to this situation:

- transformations that exploit the uniqueness of $\mathsf{find}[\mathsf{unique}_e]$ or $\mathsf{get}[\mathsf{unique}_e]$, which are valid only when event e is not executed. These events are taken into account by $\mathsf{NonUnique}_{Q,D}$.
- transformations that insert events using Shoup's lemma. This is the case of the transformations **insert_event** (see Section 5.1.12) and **insert** (see Section 5.1.13). In this case, we have $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : G, D_{\text{false}}, EvUsed \xrightarrow{V}_p G', e, EvUsed \cup \{e\}$ where e is the introduced event.

The addition of Shoup events may also be combined with the cryptographic transformation of Section 5.2, for example for specifying the decisional Diffie-Hellman assumption. In general, the cryptographic axioms are of the form

$$\mathcal{D}_{\neg \mathit{EvUsed}_R}, \emptyset : L, D_{\text{false}}, \emptyset \rightarrow_p R, D_R, \mathit{EvUsed}_R$$

where L does not contain events, $D_R = \bigvee\{e \mid \mathsf{event\_abort}\ e \text{ occurs in } R\}$ and $\mathit{EvUsed}_R = \{e \text{ that occur in } R\}$ (both $\mathsf{event\_abort}\ e$ and $[\mathsf{unique}_e]$). By Lemma 19, Property 4, we infer

$$\mathcal{D}_{\neg \sigma \mathit{EvUsed}_R}, \mathcal{D}_{\mathsf{SNU}} : C[L], D_{\text{false}}, \mathit{EvUsed} \xrightarrow{V}_{p'} C[\sigma R], D_R \circ \sigma^{-1}, \mathit{EvUsed} \cup \sigma \mathit{EvUsed}_R$$

where σ is a renaming of the events in EvUsed_R to events not in EvUsed, C is a context acceptable for L and R such that the events that occur in C are in EvUsed, $\mathcal{D}_{\mathsf{SNU}}$ is a set of Shoup and non-unique events in EvUsed, $p'(C', t_{D_0}) = p(\sigma^{-1}(C'[C[\,]]), t_{D_0})$, and $V \subseteq \mathrm{var}(C)$.

Distinguishers in $\mathcal{D}_{\neg \sigma \mathit{EvUsed}_R}$ include distinguishers that use events in EvUsed, in particular distinguishers for correspondences in G, as well as distinguishers in $\mathcal{D}_{\neg(\mathit{EvUsed} \cup \sigma \mathit{EvUsed}_R)}$ used for secrecy and indistinguishability.

- transformations that prove the absence of some events (up to some probability). For such transformations, we have $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : G, e, \mathit{EvUsed} \xrightarrow{V}_p G, D_{\text{false}}, \mathit{EvUsed}$ where p is an upper bound of the probability of event e in G. For Shoup events, this generally happens when G does not contain e and $p(C, t_{D_0}) = 0$. For non-unique events, p is the probability that the $\mathsf{find}[\mathsf{unique}_e]$ or $\mathsf{get}[\mathsf{unique}_e]$ yields several possible choices; after this step, the event e is in $\mathsf{NonUnique}_{G,D}$, so we can exploit uniqueness of $\mathsf{find}[\mathsf{unique}_e]$ or $\mathsf{get}[\mathsf{unique}_e]$.

That is why, in general, when CryptoVerif transforms a game G into a game G', we have $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : G, D, \mathit{EvUsed} \xrightarrow{V}_p G', D', \mathit{EvUsed}'$.

There are still transformations that do not fit in this framework (**guess** and **guess_branch**, because they multiply probabilities, as shown in Sections 5.1.17, 5.1.18, and 5.1.19; **success simplify** because it needs to compensate probabilities of traces that execute S with those that execute $\overline{\mathsf{S}}$ to show soundness for secrecy, as shown in Section 5.1.23).

2.7.1 Secrecy

Let us now define the secrecy properties that are proved by CryptoVerif.

Definition 7 ((One-session) secrecy) Let Q be a process, x a variable, and V a set of variables. Let

$$\begin{aligned}
Q_{\mathsf{1\text{-}ses.secr.}(x)} = {} & c_{s0}(); \mathsf{new}\ b : \mathit{bool}; \overline{c_{s0}}\langle\rangle; \\
& (c_s(u_1 : [1, n_1], \dots, u_m : [1, n_m]); \mathsf{if\ defined}(x[u_1, \dots, u_m])\ \mathsf{then} \\
& \ \mathsf{if}\ b\ \mathsf{then}\ \overline{c_s}\langle x[u_1, \dots, u_m]\rangle\ \mathsf{else\ new}\ y : T; \overline{c_s}\langle y\rangle \\
& \mid c'_s(b' : \mathit{bool}); \mathsf{if}\ b = b'\ \mathsf{then\ event\_abort}\ \mathsf{S}\ \mathsf{else\ event\_abort}\ \overline{\mathsf{S}})
\end{aligned}$$

```
$Q_{\mathsf{Secrecy}(x)} = c_{s0}(); \mathsf{new}\ b : \mathit{bool}; \overline{c_{s0}}\langle\rangle;$
    $(!^{i_s \leq n_s}\ c_s[i_s](u_1 : [1, n_1], \dots, u_m : [1, n_m]); \mathsf{if\ defined}(x[u_1, \dots, u_m])\ \mathsf{then}$
      $\mathsf{if}\ b\ \mathsf{then}\ \overline{c_s[i_s]}\langle x[u_1, \dots, u_m]\rangle\ \mathsf{else}$
      $\mathsf{find}\ u'_s = i'_s \leq n_s\ \mathsf{suchthat\ defined}(y[i'_s], u_1[i'_s], \dots, u_m[i'_s]) \wedge$
          $u_1[i'_s] = u_1 \wedge \dots \wedge u_m[i'_s] = u_m$
      $\mathsf{then}\ \overline{c_s[i_s]}\langle y[u'_s]\rangle$
      $\mathsf{else\ new}\ y : T; \overline{c_s[i_s]}\langle y\rangle$
    $\mid c'_s(b' : \mathit{bool}); \mathsf{if}\ b = b'\ \mathsf{then\ event\_abort\ S\ else\ event\_abort}\ \overline{\mathsf{S}})$
```

where $c_{s0}, c_s, c'_s \notin \mathrm{fc}(Q)$, $u_1, \dots, u_m, u'_s, y, b, b' \notin \mathrm{var}(Q) \cup V$, S, $\overline{\mathsf{S}}$ do not occur in Q, and $\mathcal{E}(x) = [1, n_1] \times \dots \times [1, n_m] \to T$.

Let *sp* be $\mathsf{1\text{-}ses.secr.}(x)$ (*one-session secrecy of* x) or $\mathsf{Secrecy}(x)$ (*secrecy of* x). The events used by *sp* are S and $\overline{\mathsf{S}}$. Let $C_{sp} = [\,] \mid Q_{sp}$.

Let C be an evaluation context acceptable for $C_{sp}[Q]$ with public variables V ($x \notin V$) that does not contain the events used by *sp*. The advantage of the adversary C against *sp* in process Q is

$$\mathsf{Adv}_Q^{sp}(C) = \Pr[C[C_{sp}[Q]] : \mathsf{S}] - \Pr[C[C_{sp}[Q]] : \overline{\mathsf{S}}]$$

The process Q *satisfies sp* with public variables V ($x \notin V$) up to probability p when, for all evaluation contexts C acceptable for $C_{sp}[Q]$ with public variables V that do not contain the events used by *sp*, $\mathsf{Adv}_Q^{sp}(C) \leq p(C)$.

Intuitively, when Q satisfies sp, the adversary cannot guess the random bit b, that is, it cannot distinguish whether the test process Q_{sp} outputs the value of the secret ($b = \text{true}$) or outputs a random number ($b = \text{false}$).

For one-session secrecy, the adversary performs a single test query, modeled by $Q_{\mathsf{1\text{-}ses.secr.}(x)}$. In more detail, in $Q_{\mathsf{1\text{-}ses.secr.}(x)}$, we choose a random bit b; the adversary sends the indices $(u_1, \dots, u_m)$ on channel c_s to perform a test query on $x[u_1, \dots, u_m]$: if $b = \text{true}$, the test query sends back $x[u_1, \dots, u_m]$; if $b = \text{false}$, it sends back a random value y. Finally, the adversary should guess the bit b: it sends its guess b' on channel c'_s and, if the guess is correct, then event S is executed, and otherwise, event $\overline{\mathsf{S}}$ is executed. The probability of getting some information on the secret is the difference between the probability of S and the probability of $\overline{\mathsf{S}}$. (When the adversary always sends a guess on channel c'_s, we have $\Pr[C[C_{sp}[Q]] : \overline{\mathsf{S}}] = 1 - \Pr[C[C_{sp}[Q]] : \mathsf{S}]$, so the advantage of the adversary is $\mathsf{Adv}_Q^{sp}(C) = \Pr[C[C_{sp}[Q]] : \mathsf{S}] - \Pr[C[C_{sp}[Q]] : \overline{\mathsf{S}}] = 2\Pr[C[C_{sp}[Q]] : \mathsf{S}] - 1$, which is a more standard formula. By flipping a coin, the adversary can execute events S and $\overline{\mathsf{S}}$ with the same probability, that is why the probability that the adversary really guesses b is the difference between the probability of these two events. We need not take the absolute value of $\Pr[C[C_{sp}[Q]] : \mathsf{S}] - \Pr[C[C_{sp}[Q]] : \overline{\mathsf{S}}]$ because, when it is negative, we can obtain the opposite, positive value by considering an adversary that sends the guess $\neg b'$ instead of b'.)

For secrecy, the adversary can perform several test queries, modeled by $Q_{\mathsf{Secrecy}(x)}$. This corresponds to the "real-or-random" definition of security [2]. (As shown in [2], 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 $x[u_1, \dots, u_m]$.) The replication bound n_s in $Q_{\mathsf{Secrecy}(x)}$ is chosen large enough so that it does not prevent communications that would otherwise occur, so n_s does not actually limit the number of test queries. When we return a random value ($b = \text{false}$) and several tests queries are performed on the same indices $u_1, \dots, u_m$, we must return the same random value. That is why, in this case, we look for previous test queries ($\mathsf{find}\ u'_s \dots$) and return the previous value of y in case a previous test query was performed with the same indices. For different indices $u_1, \dots, u_m$, the returned random values are independent of each other, so the secrecy of x requires that the cells of array x are indistinguishable from independent random values. In contrast, the one-session secrecy of x only requires that all array cells of x are indistinguishable from random values, not that they are independent of each other.

By Invariant 3, the variables defined in conditions of find and in patterns and in conditions of get have no array accesses. Therefore, the definition above applies only to variables x that are not defined in conditions of find nor in patterns nor in conditions of get.

Lemma 20 *Let sp be $\mathsf{1\text{-}ses.secr.}(x)$ or $\mathsf{Secrecy}(x)$.*

If Q satisfies sp with public variables V up to probability p and C is an acceptable evaluation context for Q with public variables V, then for all $V' \subseteq V \cup \mathrm{var}(C)$, $C[Q]$ satisfies sp with public variables V' up to probability p' such that $p'(C') = p(C'[C])$.

If $Q \approx_p^{V\cup\{x\},\mathcal{E}} Q'$ and Q satisfies sp with public variables V up to probability p', then Q' satisfies sp with public variables V up to probability p'' such that $p''(C) = p'(C) + 2 \times p(C[C_{sp}[\,]], t_{\mathsf{S}})$.

Proof Suppose that Q satisfies sp with public variables V ($x \notin V$) and C is an acceptable evaluation context for Q with public variables V. Let $V' \subseteq V \cup \mathrm{var}(C)$. Choose channels c_{s0}, c_s, c'_s, variables $u_1, \dots, u_m, u'_s, y, b, b'$, and events S, $\overline{\mathsf{S}}$ such that they do not occur in $C[Q]$. Let C' be an acceptable evaluation context for $C_{sp}[C[Q]]$ with public variables V' that does not contain S nor $\overline{\mathsf{S}}$. Then we have

$$\begin{aligned}\mathsf{Adv}^{sp}_{C[Q]}(C') &= \Pr[C'[C_{sp}[C[Q]]] : \mathsf{S}] - \Pr[C'[C_{sp}[C[Q]]] : \overline{\mathsf{S}}]\\ &= \Pr[C'[C[C_{sp}[Q]]] : \mathsf{S}] - \Pr[C'[C[C_{sp}[Q]]] : \overline{\mathsf{S}}]\\ &\le p(C'[C])\end{aligned}$$

We can commute the contexts C and C_{sp} because the context C does not bind the channels of Q_{sp}. The context $C'[C]$ is an acceptable evaluation context for $C_{sp}[Q]$ with public variables V that does not contain S nor $\overline{\mathsf{S}}$: there is no common table between C and Q, and between C' and $C_{sp}[C[Q]]$, so a fortiori between C' and Q and Q_{sp} does not use tables, so there is no common table between $C'[C]$ and $C_{sp}[Q]$; moreover

$$\begin{aligned}&\mathrm{var}(C'[C]) \cap \mathrm{var}(C_{sp}[Q])\\ &\quad = ((\mathrm{var}(C') \cap \mathrm{var}(C_{sp}[Q])) \cup \mathrm{var}(C)) \cap \mathrm{var}(C_{sp}[Q])\\ &\quad \subseteq (V' \cup \mathrm{var}(C)) \cap \mathrm{var}(C_{sp}[Q]) && \text{since } \mathrm{var}(C') \cap \mathrm{var}(C_{sp}[C[Q]]) \subseteq V'\\ &\quad \subseteq (V \cup \mathrm{var}(C)) \cap \mathrm{var}(C_{sp}[Q]) && \text{since } V' \subseteq V \cup \mathrm{var}(C)\\ &\quad \subseteq V && \text{since } \mathrm{var}(C) \cap \mathrm{var}(Q) \subseteq V \text{ and } \mathrm{var}(C) \cap \mathrm{var}(Q_{sp}) = \emptyset\end{aligned}$$

Suppose that $Q \approx_p^{V\cup\{x\},\mathcal{E}} Q'$ and Q satisfies sp with public variables V up to probability p'. Let C be an acceptable evaluation context for $C_{sp}[Q']$ with public variables V that does not contain S nor $\overline{\mathsf{S}}$.

$$\begin{aligned}\mathsf{Adv}^{sp}_{Q'}(C) &= \Pr[C[C_{sp}[Q']] : \mathsf{S}] - \Pr[C[C_{sp}[Q']] : \overline{\mathsf{S}}]\\ &\le \Pr[C[C_{sp}[Q]] : \mathsf{S}] - \Pr[C[C_{sp}[Q]] : \overline{\mathsf{S}}] +\\ &\quad |\Pr[C[C_{sp}[Q']] : \mathsf{S}] - \Pr[C[C_{sp}[Q]] : \mathsf{S}]| +\\ &\quad |\Pr[C[C_{sp}[Q]] : \overline{\mathsf{S}}] - \Pr[C[C_{sp}[Q']] : \overline{\mathsf{S}}]|\\ &\le p'(C) + 2 \times p(C[C_{sp}[\,]], t_{\mathsf{S}})\end{aligned}$$

since $t_{\mathsf{S}} = t_{\overline{\mathsf{S}}}$. Indeed, by renaming the variables and tables of C that do not appear in Q' to variables and tables that also do not occur in Q, C is also an acceptable evaluation context for

$C_{sp}[Q]$ with public variables V. Furthermore, by Property 7, this renaming does not change the probabilities. $\square$

2.7.2 Secrecy for a Bit

Definition 8 (Bit secrecy) Let Q be a process, x a boolean variable defined under no replication, and V a set of variables. Let

$$Q_{\mathsf{bit\ secr.}(x)} = c''_s(b' : \mathit{bool}); \mathsf{if\ defined}(x)\ \mathsf{then\ if}\ x = b'\ \mathsf{then\ event\_abort\ S\ else\ event\_abort}\ \overline{\mathsf{S}}$$

where $c''_s \notin \mathrm{fc}(Q)$, $b' \notin \mathrm{var}(Q) \cup V$, S, $\overline{\mathsf{S}}$ do not occur in Q, and $\mathcal{E}(x) = \mathit{bool}$.

Let sp be $\mathsf{bit\ secr.}(x)$ (*bit secrecy of* x). The events used by sp are S and $\overline{\mathsf{S}}$. Let $C_{sp} = [\,] \mid Q_{sp}$. The definitions of $\mathsf{Adv}^{sp}_{Q}(C)$ and "*Q satisfies sp*" are as in Definition 7.

Intuitively, when Q satisfies sp, the adversary cannot guess the boolean x, that is, it cannot distinguish whether $x = \text{true}$ or $x = \text{false}$. The adversary performs a single test query, modeled by $Q_{\mathsf{bit\ secr.}(x)}$. This definition is simpler than the definition of (one-session) secrecy for x, because we do not introduce an additional random bit b.

By Invariant 3, the variables defined in conditions of find and in patterns and in conditions of get have no array accesses. Therefore, the definition above applies only to variables x that are not defined in conditions of find nor in patterns nor in conditions of get.

Lemma 20 is also valid when $sp = \mathsf{bit\ secr.}(x)$, with the same statement and proof.

Lemma 21 *If b_0 is a boolean variable defined under no replication and Q preserves the one-session secrecy of b_0 with public variables V up to probability p, then Q preserves the bit secrecy of b_0 with public variables V up to probability $2p$.*

Proof Let C be any acceptable evaluation context for $Q \mid Q_{\mathsf{bit\ secr.}(b_0)}$ with public variables V. Let

$$C' = C[\_ \mid c''_s(b'_0 : \mathit{bool}); \overline{c_{s0}}\langle\rangle; c_{s0}(); \overline{c_s}\langle\rangle; c_s(b''_0 : \mathit{bool}); \overline{c'_s}\langle b'_0 = b''_0\rangle]\,.$$

We execute $C'[Q \mid Q_{\mathsf{1\text{-}ses.secr.}(b_0)}]$.

When $b'_0 = b_0$ ($C[Q \mid Q_{\mathsf{bit\ secr.}(b_0)}]$ executes S),

- if $b = \text{true}$ (probability 1/2), then $b''_0 = b_0$, so $b' = (b'_0 = b''_0) = \text{true}$ and S is executed with probability 1/2
- if $b = \text{false}$ (probability 1/2), then b''_0 is random, so
  - $b''_0 = b'_0$ with probability 1/4, so $b' = (b'_0 = b''_0) = \text{true}$ and $\overline{\mathsf{S}}$ is executed;
  - $b''_0 = \neg b'_0$ with probability 1/4, so $b' = (b'_0 = b''_0) = \text{false}$ and S is executed.

When $b'_0 = \neg b_0$ ($C[Q \mid Q_{\mathsf{bit\ secr.}(b_0)}]$ executes $\overline{\mathsf{S}}$),

- if $b = \text{true}$ (probability 1/2), then $b''_0 = b_0$, so b' is false and $\overline{\mathsf{S}}$ is executed with probability 1/2;
- if $b = \text{false}$ (probability 1/2), then b''_0 is random, so
  - $b''_0 = b'_0$ with probability 1/4, so $b' = \text{true}$ and $\overline{\mathsf{S}}$ is executed;
  - $b''_0 = \neg b'_0$ with probability 1/4, so $b' = \text{false}$ and S is executed.

So

$$\Pr[C'[Q \mid Q_{\text{1-ses.secr.}(b_0)}] : \mathsf{S}] = \frac{3}{4}\Pr[C[Q \mid Q_{\text{bit secr.}(b_0)}] : \mathsf{S}] + \frac{1}{4}\Pr[C[Q \mid Q_{\text{bit secr.}(b_0)}] : \overline{\mathsf{S}}]$$
$$\Pr[C'[Q \mid Q_{\text{1-ses.secr.}(b_0)}] : \overline{\mathsf{S}}] = \frac{1}{4}\Pr[C[Q \mid Q_{\text{bit secr.}(b_0)}] : \mathsf{S}] + \frac{3}{4}\Pr[C[Q \mid Q_{\text{bit secr.}(b_0)}] : \overline{\mathsf{S}}]$$

Finally, we obtain

$$\begin{aligned}&\Pr[C'[Q \mid Q_{\text{1-ses.secr.}(b_0)}] : \mathsf{S}] - \Pr[C'[Q \mid Q_{\text{1-ses.secr.}(b_0)}] : \overline{\mathsf{S}}]\\ &= \frac{1}{2}(\Pr[C[Q \mid Q_{\text{bit secr.}(b_0)}] : \mathsf{S}] - \Pr[C[Q \mid Q_{\text{bit secr.}(b_0)}] : \overline{\mathsf{S}}])\end{aligned}$$

so $\Pr[C[Q \mid Q_{\text{bit secr.}(b_0)}] : \mathsf{S}] - \Pr[C[Q \mid Q_{\text{bit secr.}(b_0)}] : \overline{\mathsf{S}}] = 2(\Pr[C'[Q \mid Q_{\text{1-ses.secr.}(b_0)}] : \mathsf{S}] - \Pr[C'[Q \mid Q_{\text{1-ses.secr.}(b_0)}] : \overline{\mathsf{S}}]) \leq 2p(C') = 2p(C)$, neglecting the additional runtime of C'. $\square$

Intuitively, the factor 2 is necessary, because in the definition of one-session secrecy, even if the adversary knows the secret bit b_0 perfectly, it will not be able to distinguish b_0 from the random bit y in half of the cases, because b_0 and y have the same value.

In the rest of Section 2.7.2, we consider a process

$$Q = c(); \mathsf{new}\ b_0 : \mathit{bool}; \overline{c}\langle\rangle; Q'$$

and let $Q_L = Q'\{\text{true}/b_0\}$ and $Q_R = Q'\{\text{false}/b_0\}$, so that Q chooses a random bit b_0 and runs as Q_L when b_0 is true and as Q_R when b_0 is false. We assume that Q_L and Q_R never abort, that is, they contain neither $\mathsf{event\_abort}$ nor $\mathsf{find}[\mathsf{unique}_e]$. Moreover, they do not use the variable b_0. We assume that the channels of the inputs at the root of Q_L and Q_R are not used elsewhere in Q_L or Q_R. We have the following lemmas.

Lemma 22 *If Q preserves the bit secrecy of b_0 with public variables V up to probability p, then $Q_L \approx_{p'}^{V,\emptyset} Q_R$ where $p'(C, t_D) = p(C + t_D)$ and the context $C + t_D$ runs in time $t_C + t_D$ and its other parameters (replication bounds, lengths of bitstrings) are the same as for C.*

The processes Q_L and Q_R can execute different events without breaking the bit secrecy of b_0, because the adversary for the bit secrecy of b_0 does not have access to the events executed by Q. Hence, $Q_L \approx_{p'}^{V} Q_R$ would not hold in general.

Proof Let C be any acceptable evaluation context for Q_L and Q_R with public variables V, and D a distinguisher such that $\text{event}(D) \cap \text{event}(Q_L, Q_R) = \emptyset$. Let c and c''_s be a channel that C does not use.

Let C' be a context that outputs on channel c, inputs on channel c, runs C but stores events executed by C in its internal state instead of actually executing events, computes D on the stored sequence of events executed by C and stores the result in b'_0, and sends b'_0 on channel c''_s. Such a context C' exists because it can be encoded as a probabilistic Turing machine adversary, which can itself be encoded as a context in CryptoVerif, as shown in Section 2.8.

When b_0 is true, $C'[Q \mid Q_{\text{bit secr.}(x)}]$ stores in b'_0 the result of $C[Q_L] : D$. When $b'_0 = \text{true}$, S is executed. When $b'_0 = \text{false}$, $\overline{\mathsf{S}}$ is executed. So

$$\Pr[C'[Q \mid Q_{\text{bit secr.}(x)}] : \mathsf{S}/b_0 = \text{true}] = \Pr[C[Q_L] : D]$$
$$\Pr[C'[Q \mid Q_{\text{bit secr.}(x)}] : \overline{\mathsf{S}}/b_0 = \text{true}] = 1 - \Pr[C[Q_L] : D]$$

When b_0 is false, $C'[Q \mid Q_{\mathsf{bit\ secr.}(x)}]$ stores in b'_0 the result of $C[Q_R] : D$. When $b'_0 = \text{true}$, $\overline{\mathsf{S}}$ is executed. When $b'_0 = \text{false}$, S is executed. So

$$\Pr[C'[Q \mid Q_{\mathsf{bit\ secr.}(x)}] : \mathsf{S}/b_0 = \text{false}] = 1 - \Pr[C[Q_R] : D]$$
$$\Pr[C'[Q \mid Q_{\mathsf{bit\ secr.}(x)}] : \overline{\mathsf{S}}/b_0 = \text{false}] = \Pr[C[Q_R] : D]$$

Finally, we obtain

$$\begin{aligned}
&\Pr[C'[Q \mid Q_{\mathsf{bit\ secr.}(x)}] : \mathsf{S}] - \Pr[C'[Q \mid Q_{\mathsf{bit\ secr.}(x)}] : \overline{\mathsf{S}}] \\
&\quad = \frac{1}{2}(\Pr[C[Q_L] : D] + 1 - \Pr[C[Q_R] : D] - (1 - \Pr[C[Q_L] : D]) - \Pr[C[Q_R] : D]) \\
&\quad = \Pr[C[Q_L] : D] - \Pr[C[Q_R] : D]
\end{aligned}$$

so $\Pr[C[Q_L] : D] - \Pr[C[Q_R] : D] \leq p'(C, t_D)$. By negating the bit b'_0, we swap the events S and $\overline{\mathsf{S}}$ without changing the probability, so $\Pr[C[Q_R] : D] - \Pr[C[Q_L] : D] \leq p'(C, t_D)$. Therefore, $|\Pr[C[Q_L] : D] - \Pr[C[Q_R] : D]| \leq p'(C, t_D)$. So $Q_L \approx_{p'}^{V,\emptyset} Q_R$. □

Lemma 22 is the main motivation for the notion of secrecy for a bit: it allows proving indistinguishability between two processes by showing secrecy of bit b_0. Using this notion instead of one-session secrecy of b_0 avoids losing a factor 2, as shown by Lemma 21.

We use this idea to encode the diff construct originally introduced in ProVerif [26]: given a process Q_1 that contains terms $\text{diff}[M, M']$ and processes $\text{diff}[P, P']$, we define $\text{fst}(Q_1)$ as Q_1 with $\text{diff}[M, M']$ replaced with M and $\text{snd}(Q_1)$ as Q_1 with $\text{diff}[M, M']$ replaced with M', and similarly for $\text{diff}[P, P']$; the goal is to show that $\text{fst}(Q_1) \approx_p^{V,\emptyset} \text{snd}(Q_1)$ for some p and determine p. In order to do that, we define Q' as Q_1 with $\text{diff}[M, M']$ replaced with $\text{if\_fun}(b_0, M, M')$ when M and M' are simple and with $\mathsf{if}\ b_0\ \mathsf{then}\ M\ \mathsf{else}\ M'$ otherwise[1], $\text{diff}[P, P']$ replaced with $\mathsf{if}\ b_0\ \mathsf{then}\ P\ \mathsf{else}\ P'$, and $Q = c(); \mathsf{new}\ b_0 : \mathit{bool}; \overline{c}\langle\rangle; Q'$. We have $Q_L = Q'\{\text{true}/b_0\} \approx_0^{V,\emptyset} \text{fst}(Q_1)$ and $Q_R = Q'\{\text{false}/b_0\} \approx_0^{V,\emptyset} \text{snd}(Q_1)$, so by Lemma 22, if Q preserves the bit secrecy of b_0 with public variables V up to probability p, then $\text{fst}(Q_1) \approx_{p'}^{V,\emptyset} \text{snd}(Q_1)$ where $p'(C, t_D) = p(C + t_D)$.

Lemma 23 *If $Q_L \approx_p^{V,\emptyset} Q_R$ then $Q \approx_{p/2}^{V\cup\{b_0\},\emptyset} c(); \mathsf{new}\ b_0 : \mathit{bool}; \overline{c}\langle\rangle; Q_R$.*

Proof Let C be an evaluation context acceptable for Q and $c(); \mathsf{new}\ b_0 : \mathit{bool}; \overline{c}\langle\rangle; Q_R$ with public variables $V \cup \{b_0\}$ and D be a distinguisher. We have

$$\begin{aligned}
&|\Pr[C[Q] : D] - \Pr[C[c(); \mathsf{new}\ b_0 : \mathit{bool}; \overline{c}\langle\rangle; Q_R] : D| \\
&\quad \leq \frac{1}{2}|\Pr[C[Q] : D/b_0 = \text{true}] - \Pr[C[c(); \mathsf{new}\ b_0 : \mathit{bool}; \overline{c}\langle\rangle; Q_R] : D/b_0 = \text{true}]| \\
&\qquad + \frac{1}{2}|\Pr[C[Q] : D/b_0 = \text{false}] - \Pr[C[c(); \mathsf{new}\ b_0 : \mathit{bool}; \overline{c}\langle\rangle; Q_R] : D/b_0 = \text{false}]| \\
&\quad \leq \frac{1}{2}|\Pr[C[c(); \mathsf{let}\ b_0 = \text{true}\ \mathsf{in}\ \overline{c}\langle\rangle; Q_L] : D] - \Pr[C[c(); \mathsf{let}\ b_0 = \text{true}\ \mathsf{in}\ \overline{c}\langle\rangle; Q_R] : D]| \\
&\quad \leq \frac{1}{2}p(C, t_D)
\end{aligned}$$

[1]$\text{if\_fun}(b_0, M, M')$ differs from $\mathsf{if}\ b_0\ \mathsf{then}\ M\ \mathsf{else}\ M'$ in that it evaluates both M and M'. Since $\text{diff}[M, M']$ evaluates either M or M' but not both, we translate it into $\mathsf{if}\ b_0\ \mathsf{then}\ M\ \mathsf{else}\ M'$ when the evaluation of M or M' may modify the semantic state, e.g. by executing an event or by defining a variable. The evaluation of simple terms does not modify the semantic state.

Indeed, $\Pr[C[c(); \mathsf{let}\ b_0 = \mathrm{true}\ \mathsf{in}\ \overline{c}\langle\rangle; Q_L] : D] = \Pr[C''[Q_L] : D]$ (and similarly for Q_R), where $C'' = C[\mathsf{newChannel}\ \widetilde{c'}; (F_{\widetilde{c},\widetilde{c'}} \mid \mathsf{newChannel}\ \widetilde{c}; (c(); \mathsf{let}\ b_0 = \mathrm{true}\ \mathsf{in}\ \overline{c}\langle\rangle; F_{\widetilde{c'},\widetilde{c}}) \mid [\,])]$, $\widetilde{c}$ are the channels of the inputs at the root of Q_L and Q_R (which we assume not to be used elsewhere in Q_L, Q_R), $\widetilde{c'}$ are fresh channels corresponding to channels in $\widetilde{c}$, and $F_{\widetilde{c'},\widetilde{c}}$ forwards all messages sent on a channel in $\widetilde{c'}$ to the corresponding channel in $\widetilde{c}$, with replication bounds corresponding to the maximum of the replication bounds in Q_L and Q_R. (All messages on channels in $\widetilde{c}$ are forwarded to channels in $\widetilde{c'}$ and then back on channels in $\widetilde{c}$ provided the code $c(); \mathsf{let}\ b_0 = \mathrm{true}\ \mathsf{in}\ \overline{c}\langle\rangle$ has already been executed. That prevents executing Q_L or Q_R before $c(); \mathsf{let}\ b_0 = \mathrm{true}\ \mathsf{in}\ \overline{c}\langle\rangle$.) By Property 7, replacing C'' with C as argument of $p(C, t_D)$ does not affect the probability. (We neglect the additional runtime of C''.) $\square$

Lemma 24 is the converse of Lemma 22.

Lemma 24 *If $Q_L \approx_p^{V,\emptyset} Q_R$, then Q preserves the bit secrecy of b_0 with public variables V up to probability p' where $p'(C) = p(C[C_{\mathsf{bit\ secr.}(b_0)}], t_{\mathsf{S}})$.*

Proof If $Q_L \approx_p^{V,\emptyset} Q_R$, then by Lemma 23, $Q \approx_{p/2}^{V\cup\{b_0\},\emptyset} c(); \mathsf{new}\ b_0 : \mathit{bool}; \overline{c}\langle\rangle; Q_R$. Moreover, $c(); \mathsf{new}\ b_0 : \mathit{bool}; \overline{c}\langle\rangle; Q_R$ preserves the bit secrecy of b_0 with public variables V up to probability 0. (Since Q_R does not use b_0, the variable b' in $Q_{\mathsf{bit\ secr.}(b_0)}$ is independent of b_0, so a trace that executes S corresponds to a trace of the same probability and that executes $\overline{\mathsf{S}}$ by changing the value of b_0, so $\Pr[C[c(); \mathsf{new}\ b_0 : \mathit{bool}; \overline{c}\langle\rangle; Q_R \mid Q_{\mathsf{bit\ secr.}(b_0)}] : \mathsf{S}] = \Pr[C[c(); \mathsf{new}\ b_0 : \mathit{bool}; \overline{c}\langle\rangle; Q_R \mid Q_{\mathsf{bit\ secr.}(b_0)}] : \overline{\mathsf{S}}]$.) So by Lemma 20 (version for bit secrecy), Q preserves the bit secrecy of b_0 with public variables V up to probability p'. $\square$

Lemma 25 provides a converse of Lemma 21 when Q has the particular form given above. There is no probability loss in this case.

Lemma 25 *If Q preserves the bit secrecy of b_0 with public variables V up to probability p, then Q preserves the one-session secrecy of b_0 with public variables V up to probability p.*

Proof If Q preserves the bit secrecy of b_0 with public variables V up to probability p, then by Lemma 22, $Q_L \approx_{p'}^{V,\emptyset} Q_R$ where $p'(C, t_D) = p(C + t_D)$. By Lemma 23, $Q \approx_{p'/2}^{V\cup\{b_0\},\emptyset} c(); \mathsf{new}\ b_0 : \mathit{bool}; \overline{c}\langle\rangle; Q_R$.

Moreover, $c(); \mathsf{new}\ b_0 : \mathit{bool}; \overline{c}\langle\rangle; Q_R$ preserves the one-session secrecy of b_0 with public variables V up to probability 0. Indeed, since Q_R does not use b_0, b_0 can in fact be chosen in the test query in $Q_{\mathsf{1\text{-}ses.secr.}(b_0)}$, so that test query always returns a random boolean, independently of the value of the variable b of $Q_{\mathsf{1\text{-}ses.secr.}(b_0)}$. Therefore, the variable b' is independent of b, so a trace that executes S corresponds to a trace of the same probability and that executes $\overline{\mathsf{S}}$ by changing the value of b, so $\Pr[C[c(); \mathsf{new}\ b_0 : \mathit{bool}; \overline{c}\langle\rangle; Q_R \mid Q_{\mathsf{1\text{-}ses.secr.}(b_0)}] : \mathsf{S}] = \Pr[C[c(); \mathsf{new}\ b_0 : \mathit{bool}; \overline{c}\langle\rangle; Q_R \mid Q_{\mathsf{1\text{-}ses.secr.}(b_0)}] : \overline{\mathsf{S}}]$.

So by Lemma 20 (version for one-session secrecy), Q preserves the one-session secrecy of b_0 with public variables V up to probability p'' such that $p''(C) = p'(C[C_{\mathsf{1\text{-}ses.secr.}(b_0)}], t_{\mathsf{S}}) = p(C[C_{\mathsf{1\text{-}ses.secr.}(b_0)}] + t_{\mathsf{S}})$ which is about $p(C)$ by Property 7, neglecting the additional runtime of the context. $\square$

2.7.3 Correspondences

In this section, we define non-injective and injective correspondences.

Non-injective Correspondences A non-injective correspondence is a property of the form "if some events have been executed, then some other events have been executed at least once". Here, we generalize these correspondences to implications between logical formulas $\psi \Rightarrow \phi$, which may contain events. We use the following logical formulas:

$\phi ::=$	formula
$\quad M$	term
$\quad \mathsf{event}(e(M_1, \dots, M_m))$	event
$\quad \phi_1 \wedge \phi_2$	conjunction
$\quad \phi_1 \vee \phi_2$	disjunction

Terms $M, M_1, \dots, M_m$ in formulas must contain only variables x without array indices and function applications, and their variables are assumed to be distinct from variables of processes. Formulas denoted by ψ are conjunctions of events. In a correspondence $\psi \Rightarrow \phi$, the variables of ψ are universally quantified; those of ϕ that do not occur in ψ are existentially quantified. Formally:

Definition 9 The semantics of the correspondence $\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi$, also written $\widetilde{x} : \widetilde{T}, \widetilde{y} : \widetilde{T'}; \psi \Rightarrow \phi$ in a less explicit syntax, is $[\![\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi]\!] = [\![\widetilde{x} : \widetilde{T}, \widetilde{y} : \widetilde{T'}; \psi \Rightarrow \phi]\!] = \forall \widetilde{x} \in \widetilde{T}, (\psi \Rightarrow \exists \widetilde{y} \in \widetilde{T'}, \phi)$, where $\widetilde{x} = \mathrm{var}(\psi)$ and $\widetilde{y} = \mathrm{var}(\phi) \setminus \mathrm{var}(\psi)$.

The formula M holds when M evaluates to true. The formula $\mathsf{event}(e(M_1, \dots, M_n))$ holds when the event $e(M_1, \dots, M_n)$ has been executed. Conjunction, disjunction, implication, existential and universal quantifications are defined as usual. More formally, we write $\rho, \mathcal{E}v \vdash \varphi$ when the sequence of events $\mathcal{E}v$ satisfies the formula φ, in the environment ρ that maps variables to their values. We define $\rho, \mathcal{E}v \vdash \varphi$ as follows:

$\rho, \mathcal{E}v \vdash M$ if and only if $\rho, M \Downarrow \mathrm{true}$
$\rho, \mathcal{E}v \vdash \mathsf{event}(e(M_1, \dots, M_m))$ if and only if
 for all $j \leq m$, $\rho, M_j \Downarrow a_j$ and $e(a_1, \dots, a_m) \in \mathcal{E}v$
$\rho, \mathcal{E}v \vdash \varphi_1 \wedge \varphi_2$ if and only if $\rho, \mathcal{E}v \vdash \varphi_1$ and $\rho, \mathcal{E}v \vdash \varphi_2$
$\rho, \mathcal{E}v \vdash \varphi_1 \vee \varphi_2$ if and only if $\rho, \mathcal{E}v \vdash \varphi_1$ or $\rho, \mathcal{E}v \vdash \varphi_2$
$\rho, \mathcal{E}v \vdash \varphi_1 \Rightarrow \varphi_2$ if and only if $\rho, \mathcal{E}v \vdash \varphi_1$ implies $\rho, \mathcal{E}v \vdash \varphi_2$
$\rho, \mathcal{E}v \vdash \exists x \in T, \varphi$ if and only if there exists $a \in T$ such that $\rho[x \mapsto a], \mathcal{E}v \vdash \varphi$
$\rho, \mathcal{E}v \vdash \forall x \in T, \varphi$ if and only if for every $a \in T$, we have $\rho[x \mapsto a], \mathcal{E}v \vdash \varphi$

When φ is a closed formula, we write $\mathcal{E}v \vdash \varphi$ for $\rho, \mathcal{E}v \vdash \varphi$ where ρ is the empty function.

Definition 10 The sequence of events $\mathcal{E}v$ *satisfies the correspondence* φ if and only if $\mathcal{E}v \vdash \varphi$.

Definition 11 We define a distinguisher $D(\mathcal{E}v) = \mathrm{true}$ if and only if $\mathcal{E}v \vdash \varphi$, and we denote this distinguisher D simply by φ.

The advantage of the adversary C against the *correspondence* φ in process Q is $\mathsf{Adv}^{\varphi}_{Q}(C) = \Pr[C[Q] : \neg\varphi]$, where C is an evaluation context acceptable for Q with any public variables that does not contain events used by φ.

The process Q *satisfies the correspondence* φ with public variables V up to probability p if and only if for all evaluation contexts C acceptable for Q with public variables V that do not contain events used by φ, $\mathsf{Adv}^{\varphi}_{Q}(C) \leq p(C)$.

When sp is a correspondence φ, we define $C_{sp} = [\,]$ and the events used by sp are the events that occur in the formula φ. Therefore, the definition of "Q satisfies the correspondence φ" matches the definition of "Q satisfies sp" given in Definition 7.

A process satisfies φ up to probability p when the probability that it generates a sequence of events $\mathcal{E}v$ that does not satisfy φ is at most $p(C)$, in the presence of an adversary represented by the context C.

Example 2 The semantics of the correspondence

$$\forall x : pkey, y : host, z : nonce; \mathsf{event}(e_B(x,y,z)) \Rightarrow \mathsf{event}(e_A(x,y,z)) \tag{4}$$

is

$$\forall x \in pkey, \forall y \in host, \forall z \in nonce, \mathsf{event}(e_B(x,y,z)) \Rightarrow \mathsf{event}(e_A(x,y,z)) \tag{5}$$

It means that, with overwhelming probability, for all x, y, z, if $e_B(x,y,z)$ has been executed, then $e_A(x,y,z)$ has been executed.

The semantics of the correspondence

$$\begin{aligned}&\forall x : T; \mathsf{event}(e_1(x)) \wedge \mathsf{event}(e_2(x)) \Rightarrow\\ &\qquad \exists y : T'; \mathsf{event}(e_3(x)) \vee (\mathsf{event}(e_4(x,y)) \wedge \mathsf{event}(e_5(x,y)))\end{aligned}$$

is

$$\begin{aligned}&\forall x \in T, \mathsf{event}(e_1(x)) \wedge \mathsf{event}(e_2(x)) \Rightarrow\\ &\quad \exists y \in T', \mathsf{event}(e_3(x)) \vee (\mathsf{event}(e_4(x,y)) \wedge \mathsf{event}(e_5(x,y)))\end{aligned}$$

It means that, with overwhelming probability, for all x, if $e_1(x)$ and $e_2(x)$ have been executed, then $e_3(x)$ has been executed or there exists y such that both $e_4(x,y)$ and $e_5(x,y)$ have been executed.

Injective Correspondences Injective correspondences are properties of the form "if some event has been executed n times, then some other events have been executed at least n times". In order to model them in our logical formulas, we extend the grammar of formulas ϕ with injective events $\mathsf{inj\text{-}event}(e(M_1, \dots, M_m))$. The formula ψ is a conjunction of (injective or non-injective) events. The conditions on the number of executions of events apply only to injective events.

The definition of formula satisfaction is also extended, to be able to indicate at which step an event has been executed (that is, at which index it appears in $\mathcal{E}v$): $\mathsf{event}(e(\widetilde{M}))@\tau$ means that event $e(\widetilde{M})$ has been executed at step τ. Formally:

$\rho, \mathcal{E}v \vdash \mathsf{event}(e(M_1, \dots, M_m))@M_0$ if and only if
for all $j \leq m$, $\rho, M_j \Downarrow a_j$, $a_0 \neq \bot$, and $e(a_1, \dots, a_m) = \mathcal{E}v(a_0)$

With this definition, we have:

Definition 12 The semantics of the correspondence $\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi$, also written $\widetilde{x} : \widetilde{T}, \widetilde{y} : \widetilde{T'}; \psi \Rightarrow \phi$ in a less explicit syntax, is

$$\begin{aligned}&[\![\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi]\!] = [\![\widetilde{x} : \widetilde{T}, \widetilde{y} : \widetilde{T'}; \psi \Rightarrow \phi]\!] = \exists f_1, \dots, f_k \in \mathbb{N}^m \times \prod \widetilde{T} \to \mathbb{N} \cup \{\bot\},\\ &\quad \mathrm{Inj}(I, f_1) \wedge \dots \wedge \mathrm{Inj}(I, f_k) \wedge \forall \tau_1, \dots, \tau_m \in \mathbb{N}, \forall \widetilde{x} \in \widetilde{T}, (\psi^\tau \Rightarrow \exists \widetilde{y} \in \widetilde{T'}, \phi^\tau),\end{aligned}$$

where $\widetilde{x} = \mathrm{var}(\psi)$, $\widetilde{y} = \mathrm{var}(\phi) \setminus \mathrm{var}(\psi)$, $\psi = F_1 \wedge \dots \wedge F_m$, $\psi^\tau = F_1^\tau \wedge \dots \wedge F_m^\tau$, $F_j^\tau = \mathsf{event}(e(\widetilde{M}))@\tau_j$ if $F_j = \mathsf{event}(e(\widetilde{M}))$ or $F_j = \mathsf{inj\text{-}event}(e(\widetilde{M}))$, $I = \{j \mid F_j = \mathsf{inj\text{-}event}(\dots)\}$, ϕ^τ is obtained from ϕ by replacing each injective event $\mathsf{inj\text{-}event}(e(\widetilde{M}))$ with $\mathsf{event}(e(\widetilde{M}))@f_j(\tau_1, \dots, \tau_m, \widetilde{x})$ using a distinct function f_j for each injective event in ϕ^τ, and $\mathrm{Inj}(I, f)$ if and only if $f(\tau_1, \dots, \tau_m, \widetilde{x}) = f(\tau'_1, \dots, \tau'_m, \widetilde{x}') \neq \bot \Rightarrow \forall j \in I, \tau_j = \tau'_j$.

In ψ^τ and ϕ^τ, events are labeled with their associated execution step, τ_j for the events in ψ^τ and $f_j(\tau_1, \dots, \tau_m, \widetilde{x})$ for the injective events in ϕ^τ. Therefore, the functions f_j map the execution steps of events in ψ, τ_1, ..., τ_m, and the values of the variables in ψ, $\widetilde{x}$, to the associated execution steps of injective events in ϕ. (The result $\perp$ corresponds to the case in which the event in ϕ is not executed: in case of disjunctions, not all events in ϕ are required to be executed.) The correspondence is injective when these functions f_j are injective in their arguments that correspond to injective events in ψ. The indices of injective events in ψ are collected in the set I, and injectivity is guaranteed by $\mathrm{Inj}(I, f_j)$, which means that, ignoring the result $\perp$, f_j is injective in its arguments of indices in I.

Definition 11 is unchanged for injective correspondences.

Example 3 The semantics of the correspondence

$$\forall x : \mathit{pkey}, y : \mathit{host}, z : \mathit{nonce}; \textsf{inj-event}(e_B(x, y, z)) \Rightarrow \textsf{inj-event}(e_A(x, y, z)) \tag{6}$$

is

$$\begin{aligned}&\exists f \in \mathbb{N} \times \mathit{pkey} \times \mathit{host} \times \mathit{nonce} \to \mathbb{N} \cup \{\perp\}, \mathrm{Inj}(\{1\}, f) \land \\ &\quad \forall \tau \in \mathbb{N}, \forall x \in \mathit{pkey}, \forall y \in \mathit{host}, \forall z \in \mathit{nonce}, \\ &\qquad \textsf{event}(e_B(x, y, z))@\tau \Rightarrow \textsf{event}(e_A(x, y, z))@f(\tau, x, y, z)\end{aligned} \tag{7}$$

It means that, with overwhelming probability, each execution of $e_B(x, y, z)$ corresponds to a distinct execution of $e_A(x, y, z)$. In this case, f is a function that maps the execution step τ of $e_B(x, y, z)$ and the variables x, y, z to the execution step of $e_A(x, y, z)$. (This step is never $\perp$.) This function is injective in its first argument, the step τ, so if there are n executions of $e_B(x, y, z)$, at steps τ_1, ..., τ_n, then there are at least n executions of $e_A(x, y, z)$, at steps $f(\tau_1, x, y, z), \dots, f(\tau_n, x, y, z)$ and these steps are distinct by injectivity of f in its first argument.

The semantics of the correspondence

$$\begin{aligned}&\forall x : T; \textsf{event}(e_1(x)) \land \textsf{inj-event}(e_2(x)) \Rightarrow \exists y : T'; \textsf{inj-event}(e_3(x)) \lor \\ &\quad (\textsf{inj-event}(e_4(x, y)) \land \textsf{inj-event}(e_5(x, y)))\end{aligned}$$

is

$$\begin{aligned}&\exists f_1, f_2, f_3 \in \mathbb{N}^2 \times T \to \mathbb{N} \cup \{\perp\}, \mathrm{Inj}(\{2\}, f_1) \land \mathrm{Inj}(\{2\}, f_2) \land \mathrm{Inj}(\{2\}, f_3) \land \\ &\forall \tau_1, \tau_2 \in \mathbb{N}, \forall x \in T, \textsf{event}(e_1(x))@\tau_1 \land \textsf{event}(e_2(x))@\tau_2 \Rightarrow \exists y \in T', \textsf{event}(e_3(x))@f_1(\tau_1, \tau_2, x) \lor \\ &\quad (\textsf{event}(e_4(x, y))@f_2(\tau_1, \tau_2, x) \land \textsf{event}(e_5(x, y))@f_3(\tau_1, \tau_2, x))\end{aligned}$$

It means that, with overwhelming probability, for all x, if $e_1(x)$ has been executed, then each execution of $e_2(x)$ corresponds to distinct executions of $e_3(x)$ or to distinct executions of $e_4(x, y)$ and $e_5(x, y)$. The functions f_1, f_2, and f_3 map the execution steps τ_1 and τ_2 of e_1 and e_2 and the variable x to the execution steps of e_3, e_4, and e_5 respectively. Ignoring the result $\perp$, they are injective in their second argument, which corresponds to the execution step of the injective event e_2.

When no injective event occurs in $\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi$, Definition 12 reduces to the definition of non-injective correspondences: there are no functions f_j, $\phi^\tau = \phi$, and $\textsf{event}(e(\widetilde{M}))@\tau$ holds for some τ if and only if $\textsf{event}(e(\widetilde{M}))$ holds, so ψ^τ holds for some τ_1, ..., τ_m if and only if ψ holds.

Well-formedness condition When we consider a correspondence $\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T}'; \phi$, with $\widetilde{x} = \mathrm{var}(\psi)$ and $\widetilde{y} = \mathrm{var}(\phi) \setminus \mathrm{var}(\psi)$, we should have

$$\forall \widetilde{x} \in \widetilde{T}, \forall \widetilde{x}' \in \widetilde{T}, \forall \widetilde{y} \in \widetilde{T}', \psi = \psi\{\widetilde{x}'/\widetilde{x}\} \Rightarrow \phi = \phi\{\widetilde{x}'/\widetilde{x}\} \tag{8}$$

where $\widetilde{x}'$ are fresh variables, and the equality of terms is the equality of their values, but disjunctions, conjunctions, and events are considered syntactically. This condition guarantees that, given an execution of events in ψ, the formula to verify $\exists \widetilde{y} \in \widetilde{T}', \phi^\tau$ is uniquely determined. It avoids pathological correspondences such as

$$\forall x : T; \mathsf{event}(e(f(x)) \Rightarrow \mathsf{event}(e'(x)) \tag{9}$$

with $f(a) = f(b) = c$, for which $\mathsf{event}(e(c))$ corresponds to both $\mathsf{event}(e(f(a))$ and $\mathsf{event}(e(f(b))$, so when event $e(c)$ is executed, x can take both values a and b, so (9) requires the execution of events $e'(a)$ and $e'(b)$. An even more pathological case is when $f(x) = c$ for all x: in this case, when event $e(c)$ is executed, (9) requires the execution of event $e'(x)$ for all $x \in T$, which is impossible when T is infinite. However, the condition allows the correspondence (9) when f is injective, so x is uniquely determined, and it also allows the correspondences

$$\forall x : T; \mathsf{event}(e(f(x)) \Rightarrow \text{false} \tag{10}$$

and

$$\forall x : T; \mathsf{event}(e(f(x)) \Rightarrow \mathsf{event}(e'(f(x))) \tag{11}$$

for any function f: (10) requires that event $e(y)$ is never executed with y in the image of f, and (11) requires that $e'(y)$ is executed when $e(y)$ is executed with y in the image of f.

CryptoVerif displays a warning when it does not manage to prove the well-formedness condition (8).

Property

Lemma 26 *If Q satisfies a correspondence φ with public variables V up to probability p and C is an acceptable evaluation context for Q with public variables V that does not contain events used in φ, then for all $V' \subseteq V \cup \mathrm{var}(C)$, $C[Q]$ satisfies a correspondence φ with public variables V' up to probability p' such that $p'(C') = p(C'[C])$.*

If $Q \approx_p^{V,\mathcal{E}} Q'$, Q satisfies a correspondence φ with public variables V up to probability p', and $\mathcal{E}$ contains all events in φ, then Q' satisfies φ with public variables V up to probability p'' such that $p''(C) = p'(C) + p(C, t_\varphi)$.

Proof Suppose that Q satisfies a correspondence φ with public variables V and C is an acceptable evaluation context for Q with public variables V that does not contain events used in φ. Let $V' \subseteq V \cup \mathrm{var}(C)$. Let C' be an evaluation context acceptable for $C[Q]$ with public variables V' that does not contain events used by φ. We rename the variables of C' not in V' so that they are not in V; by Property 7, this renaming does not change the probabilities. We have

$$\mathsf{Adv}^{\varphi}_{C[Q]}(C') = \Pr[C'[C[Q]] : \neg\varphi] \leq p(C'[C])$$

because $C'[C]$ is an evaluation context acceptable for Q with public variables V: there is no common table between C and Q, and between C' and $C[Q]$, so a fortiori between C' and Q, so there is no common table between $C'[C]$ and Q; moreover

$$\begin{aligned}\mathrm{var}(C'[C]) \cap \mathrm{var}(Q) &= ((\mathrm{var}(C') \cap \mathrm{var}(Q)) \cup \mathrm{var}(C)) \cap \mathrm{var}(Q) \\ &\subseteq (V' \cup \mathrm{var}(C)) \cap \mathrm{var}(Q) && \text{since } \mathrm{var}(C') \cap \mathrm{var}(C[Q]) \subseteq V' \\ &\subseteq (V \cup \mathrm{var}(C)) \cap \mathrm{var}(Q) && \text{since } V' \subseteq V \cup \mathrm{var}(C) \\ &\subseteq V && \text{since } \mathrm{var}(C) \cap \mathrm{var}(Q) \subseteq V\end{aligned}$$

We also have $\mathrm{vardef}(C'[C[\,]]) \cap V = (\mathrm{vardef}(C') \cap V) \cup (\mathrm{vardef}(C) \cap V) = \emptyset$ since $\mathrm{vardef}(C) \cap V = \emptyset$ because C is an acceptable evaluation context for Q with public variables V and $\mathrm{vardef}(C') \cap V \subseteq \mathrm{vardef}(C') \cap V' = \emptyset$ because we have renamed the variables of C' not in V' so that they are not in V and C' is an acceptable evaluation context for $C[Q]$ and with public variables V'.

Suppose that $Q \approx_p^{V,\mathcal{E}} Q'$, Q satisfies a correspondence φ with public variables V up to probability p', and $\mathcal{E}$ contains all events in φ. Let C be an evaluation context acceptable for Q' with public variables V that does not contain events used by φ. We have

$$\begin{aligned}\mathsf{Adv}_{Q'}^{\varphi}(C) &= \Pr[C[Q'] : \neg\varphi] \\ &\leq \Pr[C[Q] : \neg\varphi] + |\Pr[C[Q'] : \neg\varphi] - \Pr[C[Q] : \neg\varphi]| \\ &\leq p'(C) + p(C, t_\varphi)\end{aligned}$$

Indeed, by renaming the variables and tables of C that do not appear in Q' to variables and tables that also do not occur in Q, C is also an acceptable evaluation context for Q with public variables V. Furthermore, by Property 7, this renaming does not change the probabilities. $\square$

Reachability secrecy Reachability secrecy aims to show that the adversary cannot compute the secret value. This notion is standard in the symbolic model, but less common than the notion of secrecy as "the adversary cannot distinguish the secret from a random value" (Section 2.7.1) in the computational model. It is still used, e.g. in the property of one-wayness or in the computational Diffie-Hellman assumption.

This notion makes sense only when the secret value is of a large type. Otherwise, the adversary would have a non-negligible probability of finding the secret value just by random guessing.

This notion is in fact encoded as a correspondence property. We distinguish two variants. One-session reachability secrecy of x means that the adversary cannot compute any cell of array x, even if it has access to the public variables in V. Reachability secrecy of x means that the adversary cannot compute any cell of array x, even if it has access to the other cells of x and to the public variables in V.

Definition 13 ((One-session) reachability secrecy) Let Q be a process, x a variable, and V a set of variables. Let

$$\begin{aligned}Q_{\mathsf{1\text{-}ses.reach.secr.}(x)} = {} & !^{i_t \leq n_t} c_s[i_t](x' : T, u_1 : [1, n_1], \dots, u_m : [1, n_m]); \\ & \mathsf{if\ defined}(x[u_1, \dots, u_m]) \wedge x' = x[u_1, \dots, u_m] \mathsf{\ then\ event\ } \mathrm{adv\_has\_x} \\ Q_{\mathsf{Reach.secr.}(x)} = {} & !^{i_r \leq n_r} c_r[i_r](u'_1 : [1, n_1], \dots, u'_m : [1, n_m]); \mathsf{if\ defined}(x[u'_1, \dots, u'_m]) \mathsf{\ then} \\ & \mathsf{let\ } \mathit{reveal} : \mathit{bool} = \mathrm{true} \mathsf{\ in\ } \overline{c_r[i_r]}\langle x[u'_1, \dots, u'_m] \rangle \\ | \ & !^{i_t \leq n_t} c_s(x' : T, u_1 : [1, n_1], \dots, u_m : [1, n_m]); \\ & \mathsf{find\ } i \leq i_r \mathsf{\ suchthat\ defined}(\mathit{reveal}[i], u'_1[i], \dots, u'_m[i]) \wedge \\ & \qquad u'_1[i] = u_1 \wedge \dots \wedge u'_m[i] = u_m \mathsf{\ then\ yield\ else} \\ & \mathsf{if\ defined}(x[u_1, \dots, u_m]) \wedge x' = x[u_1, \dots, u_m] \mathsf{\ then\ event\ } \mathrm{adv\_has\_x}\end{aligned}$$

where $c_s, c_r \notin \mathrm{fc}(Q)$, $x', u_1, \dots, u_m, u'_1, \dots, u'_m, \mathit{reveal} \notin \mathrm{var}(Q) \cup V$, adv_has_x does not occur in Q, and $\mathcal{E}(x) = [1, n_1] \times \ldots \times [1, n_m] \to T$.

The process Q *satisfies one-session reachability secrecy of* x with public variables V ($x \notin V$) up to probability p if and only if the process $Q \mid Q_{\textsf{1-ses.reach.secr.}(x)}$ satisfies the correspondence $\mathsf{event}(\mathrm{adv\_has\_x}) \Rightarrow \mathrm{false}$ with public variables V up to probability p.

The process Q *satisfies reachability secrecy of* x with public variables V ($x \notin V$) up to probability p if and only if $Q \mid Q_{\textsf{Reach.secr.}(x)}$ satisfies the correspondence $\mathsf{event}(\mathrm{adv\_has\_x}) \Rightarrow \mathrm{false}$ with public variables V up to probability p.

The process $Q_{\textsf{1-ses.reach.secr.}(x)}$ waits on channel $c_s[i_t]$ for a candidate value x' and indices $u_1, \dots, u_m$. If $x[u_1, \dots, u_m]$ is defined and equal to x', the adversary managed to compute $x[u_1, \dots, u_m]$, hence to break one-session reachability secrecy. In this case, we execute event adv_has_x, and our goal will be to bound the probability of this event, by showing the correspondence $\mathsf{event}(\mathrm{adv\_has\_x}) \Rightarrow \mathrm{false}$.

The process $Q_{\textsf{Reach.secr.}(x)}$ additionally provides a reveal query: by sending indices $u'_1, \dots, u'_m$ on channel $c_r[i_r]$, the adversary can obtain the value of $x[u'_1, \dots, u'_m]$ if it is defined. Obviously, the adversary breaks reachability secrecy if it computes $x[u_1, \dots, u_m]$ without having first made a successful reveal query on the indices $u_1, \dots, u_m$. The absence of such a reveal query is verified by $\mathsf{find}\ i \leq i_r \ldots$ before executing event adv_has_x.

The bounds on the number of queries (n_t, n_r) are chosen large enough that they do not limit the adversary.

2.7.4 Computation of Advantages

Definition 14 Let sp be a security property: sp is $\textsf{1-ses.secr.}(x)$, $\textsf{Secrecy}(x)$, $\textsf{bit secr.}(x)$, or a trace property, represented by any distinguisher that does not use S, $\overline{\mathsf{S}}$, nor non-unique events. (Trace properties include correspondences φ, as well as true, the property that is always true.) Let D be a disjunction of Shoup and non-unique events that does not contain S nor $\overline{\mathsf{S}}$. Let C be an evaluation context acceptable for Q with any public variables.

When sp is a trace property, we define $V_{sp} = \emptyset$ and

$$\mathsf{Adv}_Q(C, sp, D) = \Pr[C[Q] : (\neg sp \vee D) \wedge \neg\mathsf{NonUnique}_{Q,D}] \tag{12}$$

When sp is $\textsf{1-ses.secr.}(x)$, $\textsf{Secrecy}(x)$, or $\textsf{bit secr.}(x)$, we define $V_{sp} = \{x\}$ and

$$\mathsf{Adv}_Q(C, sp, D) = \Pr[C[Q] : \mathsf{S} \vee D] - \Pr[C[Q] : \overline{\mathsf{S}} \vee \mathsf{NonUnique}_{Q,D}] \tag{13}$$

We write $\mathsf{Bound}_Q(V, sp, D, p)$ when $V_{sp} \subseteq V$ and for all evaluation contexts C' acceptable for $C_{sp}[Q]$ with public variables $V \setminus V_{sp}$ that does not contain events used by sp or D nor non-unique events in Q, we have $\mathsf{Adv}_Q(C, sp, D) \leq p(C)$ for $C = C'[C_{sp}[\,]]$.

The events S and $\overline{\mathsf{S}}$ are only executed by $\mathsf{event\_abort}$. In (13), we could write $(\mathsf{S} \vee D) \wedge \neg\mathsf{NonUnique}_{Q,D}$ instead of $\mathsf{S} \vee D$, to be more similar to (12). That would be equivalent because the game immediately aborts after executing S as well as events in D and in $\mathsf{NonUnique}_{Q,D}$, so only one of these events is executed. In (13), we expect that $C = C'[C_{sp}[\,]]$ for some context C'. This is what happens in the definition of $\mathsf{Bound}_Q(V, sp, D, p)$. In that definition, C is a context acceptable for Q with public variables V.

Lemma 27 *1. In the initial game Q, let $D_U = \bigvee\{e \mid [\mathsf{unique}_e]$ occurs in $Q\} = \mathsf{NonUnique}_Q$. If $\mathsf{Bound}_Q(V, sp, D_U, p)$, then Q satisfies property sp with public variables $V \setminus V_{sp}$ up to probability p' where $p'(C') = p(C'[C_{sp}[\,]])$.*

2. *If* $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : Q, D, \mathit{EvUsed} \xrightarrow{V}_p Q', D', \mathit{EvUsed}'$*, the events* S *and* $\overline{\mathsf{S}}$ *are not in* EvUsed'*,* $\mathsf{Bound}_{Q'}(V, sp, D', p')$*, and*
   - *either* sp *is a trace property,* $\neg sp \in \mathcal{D}$*, and* $p''(C) = p(C, t_{\neg sp}) + p'(C)$*;*
   - *or* sp *is* $\mathsf{1\text{-}ses.secr.}(x)$*,* $\mathsf{Secrecy}(x)$*, or* $\mathsf{bit\ secr.}(x)$*,* $\{\mathsf{S}, \neg\overline{\mathsf{S}}\} \subseteq \mathcal{D}$*, and* $p''(C) = 2p(C, t_{\mathsf{S}}) + p'(C)$

   then $\mathsf{Bound}_Q(V, sp, D, p'')$*.*
3. *Let* D *and* D' *be disjunctions of Shoup and non-unique events that do not contain* S *nor* $\overline{\mathsf{S}}$*.*
   - *If* sp *is a trace property,* $\mathsf{Bound}_Q(V, sp, D, p)$*, and* $\mathsf{Bound}_Q(V, \mathrm{true}, D', p')$*, then we have* $\mathsf{Bound}_Q(V, sp, D \vee D', p + p')$*.*
   - *If* sp *is* $\mathsf{1\text{-}ses.secr.}(x)$*,* $\mathsf{Secrecy}(x)$*, or* $\mathsf{bit\ secr.}(x)$*,* $D'_{\mathsf{NU}} = \bigvee\{e \mid e$ *occurs in* D' *and* e *is a non-unique event*$\}$*,* $\mathsf{Bound}_Q(V, sp, D, p)$*,* $\mathsf{Bound}_Q(V, \mathrm{true}, D', p')$*, and* $\mathsf{Bound}_Q(V, \mathrm{true}, D'_{\mathsf{NU}}, p'')$*, then* $\mathsf{Bound}_Q(V, sp, D \vee D', p + p' + p'')$*.*
4. *If* $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : Q, D, \mathit{EvUsed} \xrightarrow{V}_p Q', D', \mathit{EvUsed}'$*, the distinguisher* D'' *is a disjunction of Shoup and non-unique events in* EvUsed *that does not contain* S *nor* $\overline{\mathsf{S}}$*, and* $\mathsf{Bound}_Q(V, \mathrm{true}, D'', p')$*, then* $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : Q, D \vee D'', \mathit{EvUsed} \xrightarrow{V}_{p+p'} Q', D', \mathit{EvUsed}'$*.*

Proof Property 1: We have $\mathsf{NonUnique}_{Q,D_U} = D_{\mathrm{false}}$. Let C' be an evaluation context acceptable for $C_{sp}[Q]$ with public variables $V \setminus V_{sp}$ that does not contain events used by sp, and $C = C'[C_{sp}[\,]]$. Since $\mathsf{Bound}_Q(V, sp, D_U, p)$, we have $\mathsf{Adv}_Q(C, sp, D_U) \le p(C)$.

- In case sp is a trace property, $\mathsf{Adv}^{sp}_Q(C') = \Pr[C'[Q] : \neg sp] \le \Pr[C'[Q] : \neg sp \vee D_U] = \Pr[C'[C_{sp}[Q]] : \neg sp \vee D_U] = \mathsf{Adv}_Q(C'[C_{sp}[\,]], sp, D_U)$.
- In case sp is $\mathsf{1\text{-}ses.secr.}(x)$, $\mathsf{Secrecy}(x)$, or $\mathsf{bit\ secr.}(x)$, $\mathsf{Adv}^{sp}_Q(C') = \Pr[C'[C_{sp}[Q]] : \mathsf{S}] - \Pr[C'[C_{sp}[Q]] : \overline{\mathsf{S}}] \le \Pr[C'[C_{sp}[Q]] : \mathsf{S} \vee D_U] - \Pr[C'[C_{sp}[Q]] : \overline{\mathsf{S}}] = \mathsf{Adv}_Q(C'[C_{sp}[\,]], sp, D_U)$.

In both cases, $\mathsf{Adv}^{sp}_Q(C') \le \mathsf{Adv}_Q(C'[C_{sp}[\,]], sp, D_U) = \mathsf{Adv}_Q(C, sp, D_U) \le p(C) = p'(C')$, so Q satisfies property sp with public variables $V \setminus V_{sp}$ up to probability p'.

Property 2, case sp is a trace property: Let C be an evaluation context acceptable for Q with public variables V that does not contain events used by sp nor D nor non-unique events of Q. Let C' be obtained by renaming the variables and tables of C that do not occur in Q to variables and tables that also do not occur in Q', and by renaming the events of C so that they are not in EvUsed'. The context C' is then acceptable for Q and Q' with public variables V and does not contain events in EvUsed'. Since $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : Q, D, \mathit{EvUsed} \xrightarrow{V}_p Q', D', \mathit{EvUsed}'$ and $\neg sp \in \mathcal{D}$, we have by taking $D_1 = D_{\mathrm{false}}$,

$$\begin{aligned}\mathsf{Adv}_Q(C, sp, D) &= \Pr[C[Q] : (\neg sp \vee D) \wedge \neg\mathsf{NonUnique}_{Q,D}]\\ &= \Pr[C'[Q] : (\neg sp \vee D) \wedge \neg\mathsf{NonUnique}_{Q,D}]\\ &\qquad\text{since the renaming of events does not affect the events of the distinguisher}\\ &\le p(C', t_{\neg sp}) + \Pr[C'[Q'] : (\neg sp \vee D') \wedge \neg\mathsf{NonUnique}_{Q',D'}]\\ &\le p(C', t_{\neg sp}) + \mathsf{Adv}_{Q'}(C', sp, D')\\ &\le p(C', t_{\neg sp}) + p'(C') \qquad\qquad \text{since } \mathsf{Bound}_{Q'}(V, sp, D', p')\\ &\le p(C, t_{\neg sp}) + p'(C)\\ &\qquad\text{since the renaming does not modify the probability formulas by Property 7}\\ &\le p''(C)\end{aligned}$$

so $\mathsf{Bound}_Q(V, sp, D, p'')$.

Property 2, case sp is $\mathsf{1\text{-}ses.secr.}(x)$, $\mathsf{Secrecy}(x)$, or $\mathsf{bit\ secr.}(x)$: Let C' be an evaluation context acceptable for $C_{sp}[Q]$ with public variables $V \setminus V_{sp}$ that does not contain S, $\overline{\mathsf{S}}$, events used by D, nor non-unique events of Q. Let $C = C'[C_{sp}[\,]]$. Let C'' be obtained by renaming the variables and tables of C that do not occur in Q to variables and tables that also do not occur in Q', and by renaming the events of C so that they are not in EvUsed'. (The events S and $\overline{\mathsf{S}}$ are left unchanged by this renaming; they are not in EvUsed'.) The context C'' is then acceptable for Q and Q' with public variables V and does not contain events in EvUsed'. Then we have

$$\begin{aligned}
\mathsf{Adv}_Q(C, sp, D) &= \Pr[C[Q] : \mathsf{S} \vee D] - \Pr[C[Q] : \overline{\mathsf{S}} \vee \mathsf{NonUnique}_{Q,D}] \\
&= \Pr[C''[Q] : \mathsf{S} \vee D] - \Pr[C''[Q] : \overline{\mathsf{S}} \vee \mathsf{NonUnique}_{Q,D}] \\
&\quad \text{since the renaming of events does not affect the events of the distinguisher} \\
&= \Pr[C''[Q] : (\mathsf{S} \vee D) \wedge \neg\mathsf{NonUnique}_{Q,D}] \\
&\quad - \Pr[C''[Q] : (\overline{\mathsf{S}} \wedge \neg D) \vee \mathsf{NonUnique}_{Q,D}]
\end{aligned}$$

since $\mathsf{S} \vee D$ is equivalent to $(\mathsf{S} \vee D) \wedge \neg\mathsf{NonUnique}_{Q,D}$ and $\overline{\mathsf{S}}$ is equivalent to $\overline{\mathsf{S}} \wedge \neg D$: the game aborts immediately after executing S, $\overline{\mathsf{S}}$, and the events in D and $\mathsf{NonUnique}_{Q,D}$ so only one of them can be executed

$$\begin{aligned}
&= \Pr[C''[Q] : (\mathsf{S} \vee D) \wedge \neg\mathsf{NonUnique}_{Q,D}] \\
&\quad + \Pr[C''[Q] : (\neg\overline{\mathsf{S}} \vee D) \wedge \neg\mathsf{NonUnique}_{Q,D}] - 1 \\
&\leq \Pr[C''[Q'] : (\mathsf{S} \vee D') \wedge \neg\mathsf{NonUnique}_{Q',D'}] + p(C'', t_{\mathsf{S}}) \\
&\quad + \Pr[C''[Q'] : (\neg\overline{\mathsf{S}} \vee D') \wedge \neg\mathsf{NonUnique}_{Q',D'}] + p(C'', t_{\mathsf{S}}) - 1 \\
&\qquad \text{since } \neg\overline{\mathsf{S}} \text{ can be implemented in the same time as } \mathsf{S} \\
&\leq \Pr[C''[Q'] : (\mathsf{S} \vee D') \wedge \neg\mathsf{NonUnique}_{Q',D'}] + p(C'', t_{\mathsf{S}}) \\
&\quad - \Pr[C''[Q'] : (\overline{\mathsf{S}} \wedge \neg D') \vee \mathsf{NonUnique}_{Q',D'}] + p(C'', t_{\mathsf{S}}) \\
&\leq \Pr[C''[Q'] : \mathsf{S} \vee D'] - \Pr[C''[Q'] : \overline{\mathsf{S}} \vee \mathsf{NonUnique}_{Q',D'}] + 2p(C'', t_{\mathsf{S}})
\end{aligned}$$

since $\mathsf{S} \vee D'$ is also equivalent to $(\mathsf{S} \vee D') \wedge \neg\mathsf{NonUnique}_{Q,D'}$ and $\overline{\mathsf{S}}$ is also equivalent to $\overline{\mathsf{S}} \wedge \neg D'$

$$\begin{aligned}
&\leq \mathsf{Adv}_{Q'}(C'', sp, D') + 2p(C'', t_{\mathsf{S}}) \\
&\leq p'(C'') + 2p(C'', t_{\mathsf{S}}) \qquad\qquad \text{since } \mathsf{Bound}_{Q'}(V, sp, D', p') \\
&\leq p'(C) + 2p(C, t_{\mathsf{S}}) \\
&\quad \text{since the renaming does not modify the probability formulas by Property 7} \\
&\leq p''(C)
\end{aligned}$$

so $\mathsf{Bound}_Q(V, sp, D, p'')$.

Property 3, case sp is a trace property: Let C be an evaluation context acceptable for Q with public variables V that does not contain events used by sp, D, D', nor non-unique events of Q. We have $\mathsf{NonUnique}_{Q,D} = \mathsf{NonUnique}_Q \wedge \neg D$. Therefore

$$\begin{aligned}
\mathsf{Adv}_Q(C, sp, D) &= \Pr[C[Q] : (\neg sp \vee D) \wedge \neg\mathsf{NonUnique}_{Q,D}] \\
&= \Pr[C[Q] : (\neg sp \vee D) \wedge \neg(\mathsf{NonUnique}_Q \wedge \neg D)] \\
&= \Pr[C[Q] : (\neg sp \vee D) \wedge (\neg\mathsf{NonUnique}_Q \vee D)] \\
&= \Pr[C[Q] : (\neg sp \wedge \neg\mathsf{NonUnique}_Q) \vee D]
\end{aligned}$$

In particular, $\mathsf{Adv}_Q(C, \text{true}, D) = \Pr[C[Q] : D]$. Hence

$$\begin{aligned}\mathsf{Adv}_Q(C, sp, D \vee D') &= \Pr[C[Q] : (\neg sp \wedge \neg \mathsf{NonUnique}_Q) \vee D \vee D']\\ &\leq \Pr[C[Q] : (\neg sp \wedge \neg \mathsf{NonUnique}_Q) \vee D] + \Pr[C[Q] : D']\\ &\leq \mathsf{Adv}_Q(C, sp, D) + \mathsf{Adv}_Q(C, \text{true}, D')\\ &\leq p(C) + p'(C)\end{aligned}$$

since $\mathsf{Bound}_Q(V, sp, D, p)$ and $\mathsf{Bound}_Q(V, \text{true}, D', p')$. So $\mathsf{Bound}_Q(V, sp, D \vee D', p + p')$.

Property 3, case sp is $\mathsf{1\text{-}ses.secr.}(x)$, $\mathsf{Secrecy}(x)$, or $\mathsf{bit\ secr.}(x)$: Let C' be an evaluation context acceptable for $C_{sp}[Q]$ with public variables $V \setminus V_{sp}$ that does not contain S, $\overline{\mathsf{S}}$, events used by D, D', nor non-unique events of Q. Let $C = C'[C_{sp}[\,]]$. Since $\mathsf{NonUnique}_{Q,D} = \mathsf{NonUnique}_Q \wedge \neg D$, we have

$$\mathsf{Adv}_Q(C, sp, D) = \Pr[C[Q] : \mathsf{S} \vee D] - \Pr[C[Q] : \overline{\mathsf{S}} \vee (\mathsf{NonUnique}_Q \wedge \neg D)]$$

Hence

$$\begin{aligned}\mathsf{Adv}_Q(C, sp, D \vee D') &= \Pr[C[Q] : \mathsf{S} \vee D \vee D']\\ &\quad - \Pr[C[Q] : \overline{\mathsf{S}} \vee (\mathsf{NonUnique}_Q \wedge \neg D \wedge \neg D')]\\ &\leq \Pr[C[Q] : \mathsf{S} \vee D] + \Pr[C[Q] : D']\\ &\quad - \Pr[C[Q] : \overline{\mathsf{S}} \vee (\mathsf{NonUnique}_Q \wedge \neg D)] + \Pr[C[Q] : D'_{\mathsf{NU}}]\\ &\leq \mathsf{Adv}_Q(C, sp, D) + \mathsf{Adv}_Q(C, \text{true}, D') + \mathsf{Adv}_Q(C, \text{true}, D'_{\mathsf{NU}})\\ &\leq p(C) + p'(C) + p''(C)\end{aligned}$$

since $\mathsf{Bound}_Q(V, sp, D, p)$, $\mathsf{Bound}_Q(V, \text{true}, D', p')$, $\mathsf{Bound}_Q(V, \text{true}, D'_{\mathsf{NU}}, p'')$, and C is also an evaluation context acceptable for Q with public variables V. So $\mathsf{Bound}_Q(V, sp, D \vee D', p + p' + p'')$.

The inequality $- \Pr[C[Q] : \overline{\mathsf{S}} \vee (\mathsf{NonUnique}_Q \wedge \neg D \wedge \neg D')] \leq - \Pr[C[Q] : \overline{\mathsf{S}} \vee (\mathsf{NonUnique}_Q \wedge \neg D)] + \Pr[C[Q] : D'_{\mathsf{NU}}]$ used above is justified as follows:

$$\begin{aligned}&\Pr[C[Q] : \overline{\mathsf{S}} \vee (\mathsf{NonUnique}_Q \wedge \neg D)]\\ &\quad\leq \Pr[C[Q] : \overline{\mathsf{S}} \vee (\mathsf{NonUnique}_Q \wedge \neg D \wedge \neg D') \vee (\mathsf{NonUnique}_Q \wedge D')]\\ &\quad\leq \Pr[C[Q] : \overline{\mathsf{S}} \vee (\mathsf{NonUnique}_Q \wedge \neg D \wedge \neg D')] + \Pr[C[Q] : \mathsf{NonUnique}_Q \wedge D']\\ &\quad\leq \Pr[C[Q] : \overline{\mathsf{S}} \vee (\mathsf{NonUnique}_Q \wedge \neg D \wedge \neg D')] + \Pr[C[Q] : D'_{\mathsf{NU}}]\end{aligned}$$

Property 4: The distinguisher $D \vee D''$ is a disjunction of Shoup and non-unique events in *EvUsed*. Let C be any evaluation context acceptable for Q with public variables V that does not contain events in *EvUsed'*. Let $D_0 \in \mathcal{D} \cup \{D_{\text{false}}\}$. Let D_1 be a disjunction of events in $\mathcal{D}_{\mathsf{SNU}}$. We have

$$\begin{aligned}&\Pr[C[Q] : (D_0 \vee D_1 \vee D \vee D'') \wedge \neg \mathsf{NonUnique}_{Q,D_1 \vee D \vee D''}]\\ &\quad= \Pr[C[Q] : (D_0 \wedge \neg \mathsf{NonUnique}_Q) \vee D_1 \vee D \vee D'']\\ &\hspace{14em}\text{as in Property 3, case } sp \text{ is a trace property}\\ &\quad\leq \Pr[C[Q] : (D_0 \wedge \neg \mathsf{NonUnique}_Q) \vee D_1 \vee D] + \Pr[C[Q] : D'']\\ &\quad\leq \Pr[C[Q] : (D_0 \vee D_1 \vee D) \wedge \neg \mathsf{NonUnique}_{Q,D_1 \vee D}] + \mathsf{Adv}_Q(C, \text{true}, D'')\\ &\text{since } \mathsf{Adv}_Q(C, \text{true}, D'') = \Pr[C[Q] : D''] \text{ (see Property 3, case } sp \text{ is a trace property)}\\ &\quad\leq \Pr[C[Q'] : (D_0 \vee D_1 \vee D') \wedge \neg \mathsf{NonUnique}_{Q,D_1 \vee D'}] + p(C, t_{D_0}) + p'(C)\end{aligned}$$

since $\mathsf{Bound}_Q(V, \text{true}, D'', p')$. (Note that the context C does not contain events used by D'' nor non-unique events of Q.) □

This lemma allows one to bound the advantage of the adversary against secrecy and correspondences. Property 1 is used in the initial game, to express the desired probability from $\mathsf{Bound}_Q(V, sp, D_U, p)$. (Using the distinguisher D_U can also be understood by saying that we consider that the adversary wins if some non-unique event is executed, that is, if a find or get declared unique by the user actually has several possible choices. That allows the implementation to make any choice when a find[unique_e] or get[unique_e] has several possible choices: the security proof remains valid. In particular, a find[unique_e] or get[unique_e] can be implemented by always choosing the first found element.) Property 2 is used when a game Q is transformed into a game Q' during the proof. It allows one to bound the probability in Q from a bound in Q'. Property 3 is useful when distinct sequences of games are used for bounding the probabilities of breaking sp and of D on one side and of D' on the other side. We bound these two probabilities by $\mathsf{Bound}_Q(V, sp, D, p)$ and $\mathsf{Bound}_Q(V, \text{true}, D', p')$ separately, then obtain a bound $\mathsf{Bound}_Q(V, sp, D \vee D', p''')$ by computing a sum. (When we deal with secrecy and $D'_{\mathsf{NU}} \neq D_{\text{false}}$, the probability of D'_{NU} can be bounded by looking at the proof for D'.)

More formally, consider the following cases, using Lemma 27, Property 1:

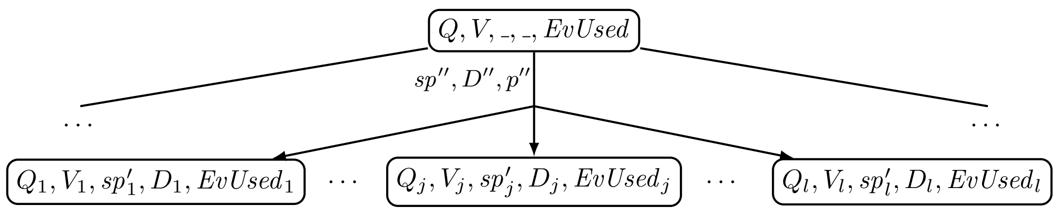
- If we want to prove that Q_0 satisfies the correspondence φ with public variables V, then we let $sp = \varphi$.
- If we want prove that Q_0 satisfies the (one-session or bit) secrecy of x with public variables V' ($x \notin V'$), then we let sp be $\mathsf{1\text{-}ses.secr.}(x)$, $\mathsf{Secrecy}(x)$, or $\mathsf{bit\ secr.}(x)$ and $V = V' \cup \{x\}$.

In both cases, we show $\mathsf{Bound}_{Q_0}(V, sp, D_U, p)$. The proof produced by CryptoVerif can be represented as a tree whose nodes are labeled with quintuples $(Q, V, sp', D, \mathit{EvUsed})$ and whose edges are labeled with triples (sp'', D'', p''), where Q is the current game, V is the set of public variables, sp' and sp'' are either the initial property to prove sp or true, D and D'' are disjunctions of Shoup and non-unique events, EvUsed is the set of events used so far, and p'' is a probability formula. The edges have a single source node, but may have 0, 1, or several target nodes. We associate to each node labeled with $(Q, V, sp', D, \mathit{EvUsed})$ a property $\mathsf{Bound}_Q(V, sp', D, p)$, and to each edge labeled with (sp'', D'', p'') with source node labeled with $(Q, V, sp', D, \mathit{EvUsed})$ a property $\mathsf{Bound}_Q(V, sp'', D'', p')$. CryptoVerif computes the probabilities p, p' such that these properties hold from the leaves of the tree to its root, as explained next.

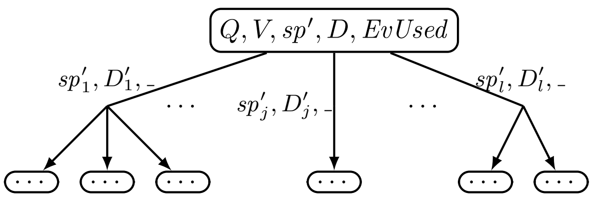
The root of the tree is labeled with $(Q_0, V, sp, D_U, \mathit{EvUsed}_0)$ such that EvUsed_0 is the set containing the events used by correspondences to prove or that occur in Q_0.

When an edge labeled with (sp'', D'', p'') has a source node labeled with $(Q, V, \_, \_, \mathit{EvUsed})$ and target nodes labeled with $(Q_j, V_j, sp'_j, D_j, \mathit{EvUsed}_j)$ for $j \in \{1, \dots, l\}$ (Figure 13(a)), the bound associated to the edge can be computed from the bound associated to the target nodes by the probability formula p'' that labels the edge: if $\mathsf{Bound}_{Q_j}(V_j, sp'_j, D_j, p_j)$ for $j \in \{1, \dots, l\}$, then $\mathsf{Bound}_Q(V, sp'', D'', p)$ where $p(C) = p''(C, p_1(C), \dots, p_l(C))$. This situation corresponds to a game transformation that transforms game Q into games Q_j ($j \in \{1, \dots, l\}$). We can distinguish several cases depending on where the edge comes from:

- For most game transformations, the edge has a single target node ($l = 1$), $V_1 = V$, $sp'_1 = sp''$, and the game transformation transforms Q into Q_1 and satisfies $\mathcal{D}_1, \emptyset : Q, D'', \mathit{EvUsed} \xrightarrow{V}_{p'} Q_1, D_1, \mathit{EvUsed}_1$ where $\mathcal{D}_1 = \{\neg sp''\}$ when sp'' is a trace property, $\mathcal{D}_1 = \{\mathsf{S}, \neg\overline{\mathsf{S}}\}$ when sp'' is $\mathsf{1\text{-}ses.secr.}(x)$, $\mathsf{Secrecy}(x)$, or $\mathsf{bit\ secr.}(x)$, and $\mathcal{D}_1 = \emptyset$ when $sp'' = \text{true}$. (During the building of the proof tree, we have $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : Q, D, \mathit{EvUsed} \xrightarrow{V}_{p'} Q_1, D', \mathit{EvUsed}_1$, where the distinguishers $\mathcal{D}$ correspond to the active queries not for introduced Shoup and non-unique events, so $\mathcal{D}_1 \subseteq \mathcal{D}$, $\mathcal{D}_{\mathsf{SNU}}$ are the active queries for Shoup and non-unique events both before and after this step so D'' and D_1 are disjunctions of events in


$Q, V, \_, \_, \mathit{EvUsed}$
sp'', D'', p''
$\dots$
$\dots$
$Q_1, V_1, sp'_1, D_1, \mathit{EvUsed}_1$
$\dots$
$Q_j, V_j, sp'_j, D_j, \mathit{EvUsed}_j$
$\dots$
$Q_l, V_l, sp'_l, D_l, \mathit{EvUsed}_l$

(a) edge, source, and target nodes



(b) node and outgoing edges

Figure 13: Structure of a proof tree

$\mathcal{D}_{\mathsf{SNU}}$, $D = D'' \wedge \neg D_1$ are the Shoup/non-unique events proved at this step, $D' = D_1 \wedge \neg D''$ are the Shoup/non-unique events introduced at this step. By applying several times Lemma 19, Property 5, we obtain $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : Q, D'', \mathit{EvUsed} \xrightarrow{V}_{p'} Q_1, D_1, \mathit{EvUsed}_1$. By Lemma 19, Property 6, we obtain $\mathcal{D}_1, \emptyset : Q, D'', \mathit{EvUsed} \xrightarrow{V}_{p'} Q_1, D_1, \mathit{EvUsed}_1$.) The bound is inferred by Lemma 27, Property 2.

If sp'' is a trace property and $\mathsf{Bound}_{Q_1}(V, sp'', D_1, p_1)$, then $\mathsf{Bound}_Q(V, sp'', D'', p)$ where $p(C) = p'(C, t_{\neg sp''}) + p_1(C)$, so we can define $p''(C, p_t) = p'(C, t_{\neg sp''}) + p_t$.

If sp'' is $\mathsf{1\text{-}ses.secr.}(x)$, $\mathsf{Secrecy}(x)$, or $\mathsf{bit\ secr.}(x)$ and $\mathsf{Bound}_{Q_1}(V, sp'', D_1, p_1)$, then we have $\mathsf{Bound}_Q(V, sp'', D'', p)$ where $p(C) = 2p'(C, t_{\mathsf{S}}) + p_1(C)$, so we can define $p''(C, p_t) = 2p'(C, t_{\mathsf{S}}) + p_t$.

To unify these two cases, we define

$$\mathrm{pstd}_{p'}^{sp''}(C, p_t) = \begin{cases} p'(C, t_{\neg sp''}) + p_t & \text{when } sp'' \text{ is a trace property} \\ 2p'(C, t_{\mathsf{S}}) + p_t & \text{when } sp'' \text{ is } \mathsf{1\text{-}ses.secr.}(x),\ \mathsf{Secrecy}(x), \text{ or } \mathsf{bit\ secr.}(x) \end{cases}$$

so that, when an edge comes from a transformation $\mathcal{D}_1, \emptyset : Q, D'', \mathit{EvUsed} \xrightarrow{V}_{p'} Q_1, D_1, \mathit{EvUsed}_1$ and the considered security property is sp'', the edge can be labeled with the probability formula $\mathrm{pstd}_{p'}^{sp''}$. If $\mathsf{Bound}_{Q_1}(V, sp'', D_1, p_1)$, then $\mathsf{Bound}_Q(V, sp'', D'', p)$ where $p(C) = \mathrm{pstd}_{p'}^{sp''}(C, p_1(C))$.

- When a query is proved (by the command **success**, Section 4), the edge has no target node ($l = 0$), and we simply obtain $\mathsf{Bound}_Q(V, sp'', D'', p'')$. The probability p'' that labels the edge is determined by the **success** command (Proposition 1, 2, or 3).

  These propositions require that $D'' = D_{\mathrm{false}}$. This is obtained by splitting the properties to prove one property at a time (with one edge for each property starting from the source node), yielding bounds of the form $\mathsf{Bound}_Q(V, sp'', D_{\mathrm{false}}, p'')$ or $\mathsf{Bound}_Q(V, \mathrm{true}, e, p'')$. For the first form, p'' can immediately be computed from Proposition 1, 2, or 3. For the second form, when e is a non-unique event, we use Section 4.2.2 and when e is a Shoup event, we notice that $\mathsf{Adv}_Q(C, \mathrm{true}, e) = \Pr[C[Q] : e \wedge \neg\mathsf{NonUnique}_{Q,e}] = \Pr[C[Q] : e \wedge \neg\mathsf{NonUnique}_Q] = \mathsf{Adv}_Q(C, [\![\mathsf{event}(e) \Rightarrow \mathrm{false}]\!], D_{\mathrm{false}})$, so we can use $\mathsf{Bound}_Q(V, [\![\mathsf{event}(e) \Rightarrow \mathrm{false}]\!], D_{\mathrm{false}}, p'')$ instead.

The **success** command is the only one that removes an event from D'', which then happens only when we evaluate $\mathsf{Bound}_Q(V, \mathrm{true}, e, p'')$, so $D'' = e$, $sp'' = \mathrm{true}$, $l = 0$.

- In case of other transformations such as the **guess** transformation, the relation between bounds that defines p'' is given directly in the soundness lemma for the transformation (Lemma 58, 59, or 60). In particular, for the transformation **guess_branch**, the edge has as many target nodes as there are branches in the guessed instruction. For the transformation **guess** i, Lemma 58 requires $D'' = D_{\mathrm{false}}$, which can be achieved as for **success** above.

When a node labeled with $(Q, V, sp', D, \mathit{EvUsed})$ has outgoing edges labeled respectively $(sp'_1, D'_1, \_)$, $\dots$, $(sp'_l, D'_l, \_)$ (Figure 13(b)), then $D = D'_1 \vee \dots \vee D'_l$ (D is a disjunction of the form $e_1 \vee \dots \vee e_m$, $D'_1, \dots, D'_l$ are disjunctions that form a partition of the disjuncts of D), there exists $j_0 \leq l$ such that $sp'_{j_0} = sp'$ and for all $j \neq j_0$, $sp'_j = \mathrm{true}$. The bound associated to the node is computed from the bound associated to the edges by Lemma 27, Property 3:

- If sp' is a trace property, then we have $\mathsf{Bound}_Q(V, sp', D, p'_1 + \dots + p'_l)$, where $\mathsf{Bound}_Q(V, sp'_j, D'_j, p'_j)$ for all $j \in \{1, \dots, l\}$.

- If sp' is $\mathsf{1\text{-}ses.secr.}(x)$, $\mathsf{Secrecy}(x)$, or $\mathsf{bit\ secr.}(x)$, then we have $\mathsf{Bound}_Q(V, sp', D, \sum_{j=1}^{l} p'_j + \sum_{j=1,\dots,l; j \neq j_0} p''_j)$, where $\mathsf{Bound}_Q(V, sp'_j, D'_j, p'_j)$ for all $j \in \{1, \dots, l\}$, $\mathsf{Bound}_Q(V, sp'_j, D'_{\mathsf{NU}j}, p''_j)$ for all $j \in \{1, \dots, l\}$ with $j \neq j_0$, and $D'_{\mathsf{NU}j}$ is obtained from D'_j by keeping only the non-unique events.

  To prove $\mathsf{Bound}_Q(V, sp'_j, D'_{\mathsf{NU}j}, p''_j)$, we build a proof tree with root $Q, V, sp'_j, D'_{\mathsf{NU}j}, \mathit{EvUsed}$ from the subtree $Q, V, sp', D, \mathit{EvUsed} \xrightarrow{sp'_j, D'_j, p_j} \dots$ by

  - replacing the root $Q, V, sp', D, \mathit{EvUsed}$ with $Q, V, sp'_j, D'_j, \mathit{EvUsed}$.
  - removing all Shoup events of D'_j from distinguishers that label nodes and edges. In particular, the root $Q, V, sp'_j, D'_j, \mathit{EvUsed}$ then becomes $Q, V, sp'_j, D'_{\mathsf{NU}j}, \mathit{EvUsed}$.
  - removing subtrees that start with an edge labeled $sp'_j, D_{\mathrm{false}}, p$ for some p.

  The proof steps remain valid: all proof steps are a fortiori valid when we ignore some Shoup events. That can be verified for each transformation using its soundness lemma. For instance, for proof steps that come from usual transformations that satisfy property preservation with introduction of events and have a Shoup event e of D'_j both before and after, we can avoid adding that event e when we derive $\mathcal{D}_1, \emptyset : Q, D'_j, \mathit{EvUsed} \xrightarrow{V}_{p'} Q_1, D_1, \mathit{EvUsed}_1$ from $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : Q, D, \mathit{EvUsed} \xrightarrow{V}_{p'} Q_1, D', \mathit{EvUsed}_1$. For proof steps that prove a Shoup event e of D'_j (via **success**), that proof step starts with just event e, so it is simply removed. We can then apply the previous reasoning to that proof tree.

When the proof is a basic sequence of games, each node has one son, which is the next game in the sequence, except the last game of the sequence which has no son. Only the final proof step is distinct for each query. However, it may happen that distinct sequences of games are used to bound several events occurring in the game; in this case, there is a branching in the proof and a node has several sons. Examples of proof trees can be found in Figure 14; they are explained below.

$$\begin{tikzcd}
 & \boxed{G_0, V, sp, D_{\text{false}}, \{e_0\}} \arrow[d, "{sp, D_{\text{false}}, \text{pstd}_0^{sp}}"] & \\
 & \boxed{G_1, V, sp, e, \{e_0, e\}} \arrow[d, "{sp, e, \text{pstd}_p^{sp}}"] & \\
 & \boxed{G_2, V, sp, e, \{e_0, e\}} \arrow[dl, "{sp, D_{\text{false}}, 0}"'] \arrow[dr, "{\text{true}, e, 0}"] & \\
 {} & & {}
\end{tikzcd}$$

(a) linear sequence until final proof

$$\begin{tikzcd}
 & \boxed{G_0, V, sp, D_{\text{false}}, \{e_0\}} \arrow[d, "{sp, D_{\text{false}}, \text{pstd}_0^{sp}}"] & \\
 & \boxed{G_1, V, sp, e, \{e_0, e\}} \arrow[d, "{sp, e, \text{pstd}_p^{sp}}"] & \\
 & \boxed{G_2, V, sp, e, \{e_0, e\}} \arrow[dl, "{sp, D_{\text{false}}, \text{pstd}_{p_3}^{sp}}"'] \arrow[dr, "{\text{true}, e, \text{pstd}_{p_4}^{sp}}"] & \\
 \boxed{G_3, V, sp, D_{\text{false}}, \{e_0, e\}} \arrow[d, "{sp, D_{\text{false}}, 0}"'] & & \boxed{G_4, V, \text{true}, e, \{e_0, e\}} \arrow[d, "{\text{true}, e, 0}"] \\
 {} & & {}
\end{tikzcd}$$

(b) example with branching

Figure 14: Examples of proof trees

The bound associated to the leaves of the tree is computed by **success**; the bound associated to an edge is computed from the bounds associated to its target nodes, and the bound associated to a node is computed from the bounds associated to its outgoing edges. We can then compute the bounds associated to all nodes of the tree, by induction from the leaves to the root. At the root, we obtain a bound $\mathsf{Bound}_{Q_0}(V, sp, D_U, p)$ that yields the desired result.

Lemma 27 allows us to obtain more precise probability bounds than the standard computation of probabilities generally done by cryptographers, when we use Shoup's lemma [63]. By Shoup's lemma, if G' is obtained from G by inserting an event e and modifying the code executed after e, the probability of distinguishing G' from G is bounded by the probability of executing e: for all contexts C acceptable for G and G' (with any public variables) and all distinguishers D, $|\Pr[C[G] : D] - \Pr[C[G'] : D]| \leq \Pr[C[G'] : e]$. Hence,

$$\Pr[C[G] : D] \leq \Pr[C[G'] : e] + \Pr[C[G'] : D].$$

We improve over this computation of probabilities by considering e and D simultaneously instead of making the sum of the two probabilities:

$$\Pr[C[G] : D] \leq \Pr[C[G'] : D \vee e].$$

For example, suppose that we want to bound the probability of event e_0 in G_0: we define $sp = [\![\mathsf{event}(e_0) \Rightarrow \text{false}]\!] = \neg e_0$. We transform G_0 into G_1 using Shoup's lemma, so that G_1 differs from G_0 only when G_1 executes event e, and we have $\{e_0\}, \emptyset : G_0, D_{\text{false}}, \mathit{EvUsed}_0 \rightarrow_0 G_1, e, \mathit{EvUsed}_1$; then we transform G_1 into G_2, so that $G_1 \approx_p^{\{e_0,e\}} G_2$, so we have $\{e_0\}, \emptyset : G_1, e, \mathit{EvUsed}_1 \rightarrow_p G_2, e, \mathit{EvUsed}_1$ by Lemma 19, Property 1a; and G_2 executes neither e_0 nor e. We suppose for simplicity that no $[\mathsf{unique}_{e'}]$ occurs, so that $\mathsf{NonUnique}_{G_i,D}$ is always false. The corresponding proof tree is given in Figure 14(a).

- Since e_0 does not occur in G_2, we have $\mathsf{Adv}_{G_2}(C, sp, D_{\text{false}}) = 0$ for all evaluation contexts C acceptable for G_2 with public variables V that do not contain event e_0. So $\mathsf{Bound}_{G_2}(V,$

$sp, D_{\mathrm{false}}, 0)$. Similarly, $\mathsf{Bound}_{G_2}(V, \mathrm{true}, e, 0)$. These two properties are represented in the proof tree by the two edges outgoing from node $G_2, V, sp, e, \{e_0, e\}$.

- By Lemma 27, Property 3, $\mathsf{Bound}_{G_2}(V, sp, e, p'_3)$ where $p'_3(C) = 0$.
- Since $\{e_0\}, \emptyset : G_1, e, \mathit{EvUsed}_1 \to_p G_2, e, \mathit{EvUsed}_1$, by Lemma 27, Property 2, we obtain $\mathsf{Bound}_{G_1}(V, sp, e, p'_2)$ where $p'_2(C) = p(C, t_{\neg sp}) + p'_3(C)$. This is represented in the proof tree by the edge from node $G_1, V, sp, e, \{e_0, e\}$ to node $G_2, V, sp, e, \{e_0, e\}$ labeled with $sp, e, \mathrm{pstd}^{sp}_p$.
- Since $\{e_0\}, \emptyset : G_0, D_{\mathrm{false}}, \mathit{EvUsed}_0 \to_0 G_1, e, \mathit{EvUsed}_1$, by Lemma 27, Property 2, we obtain $\mathsf{Bound}_{G_0}(V, sp, D_{\mathrm{false}}, p'_1)$ where $p'_1(C) = 0 + p'_2(C)$. This is represented in the proof tree by the edge from node $G_0, V, sp, D_{\mathrm{false}}, \{e_0\}$ to node $G_1, V, sp, e, \{e_0, e\}$ labeled with $sp, D_{\mathrm{false}}, \mathrm{pstd}^{sp}_0$.
- Finally, by Lemma 27, Property 1, we conclude that G_0 satisfies sp with public variables V up to probability p'_1, where $p'_1(C) = p'_2(C) = p(C, t_{\neg sp}) + p'_3(C) = p(C, t_{e_0})$, which means that $\Pr[C[G_0] : e_0] \leq p(C, t_{e_0})$ for all evaluation contexts C acceptable for G_0 with public variables V that do not contain event e_0.

Let C be an evaluation context acceptable for G_0 with public variables V that does not contain event e_0. Since we suppose for simplicity that no $[\mathsf{unique}_{e'}]$ occurs, so that $\mathsf{NonUnique}_{G_i,D}$ is always false, we have $\mathsf{Adv}_{G_i}(C, sp, D) = \Pr[C[G_i] : \neg sp \vee D]$, so we can write the previous computation simply using probabilities:

$$\begin{aligned}
\Pr[C[G_0] : e_0] &\leq \Pr[C[G_1] : e_0 \vee e] && \text{since } \{e_0\}, \emptyset : G_0, D_{\mathrm{false}}, \mathit{EvUsed}_0 \to_0 G_1, e, \mathit{EvUsed}_1\\
&\leq p(C, t_{e_0}) + \Pr[C[G_2] : e_0 \vee e] && \text{since } \{e_0\}, \emptyset : G_1, e, \mathit{EvUsed}_1 \to_p G_2, e, \mathit{EvUsed}_1\\
&\leq p(C, t_{e_0}) && \text{since } G_2 \text{ executes neither } e_0 \text{ nor } e.
\end{aligned}$$

In contrast, the standard computation of probabilities yields

$$\Pr[C[G_0] : e_0] \leq \Pr[C[G_1] : e_0] + \Pr[C[G_1] : e] \leq p(C, t_{e_0}) + p(C, t_e).$$

The runtime t_D of D is essentially the same for e_0, e, and $e_0 \vee e$, so $\Pr[C[G_0] : e_0] \leq p(C, t_D)$ by Lemma 27, while $\Pr[C[G_0] : e_0] \leq 2p(C, t_D)$ by the standard computation, so we have gained a factor 2. The probability that comes from the transformation of G_1 into G_2 is counted once (for distinguisher $e_0 \vee e$) instead of counting it twice (once for e_0 and once for e).

The standard computation of probabilities corresponds to applying point 3 of Lemma 27 to bound each probability separately and compute the sum, as soon as the considered distinguisher D has several disjuncts. Instead, we use point 3 of Lemma 27 only when the proof uses different sequences of games to bound the probabilities of the events, as in Figure 14(b).

Consider a proof tree that consists of a main branch that is a sequence of applications of transformations that satisfy property preservation with introduction of events (properties of the form $\mathcal{D}, \emptyset : Q_i, D'_i, \mathit{EvUsed}_i \xrightarrow{V}_{p_i} Q_{i+1}, D_{i+1}, \mathit{EvUsed}_{i+1}$), and side branches that may use any transformation to bound the probability of Shoup and non-unique events. All nodes on the main branch use the same security property sp, while nodes on side branches use true as security property. Such a proof tree happens when we prove indistinguishability properties, where sp is any distinguisher used in the definition of indistinguishability (see Section 2.7.5). In particular, the transformations **guess**, **guess_branch**, and **success simplify** are not allowed in the main branch but may be used in side branches. Lemma 27, Property 4 allows one to transform such a proof tree into a single property of the form $\mathcal{D}, \emptyset : Q, D, \mathit{EvUsed} \xrightarrow{V}_p Q', D', \mathit{EvUsed}'$.

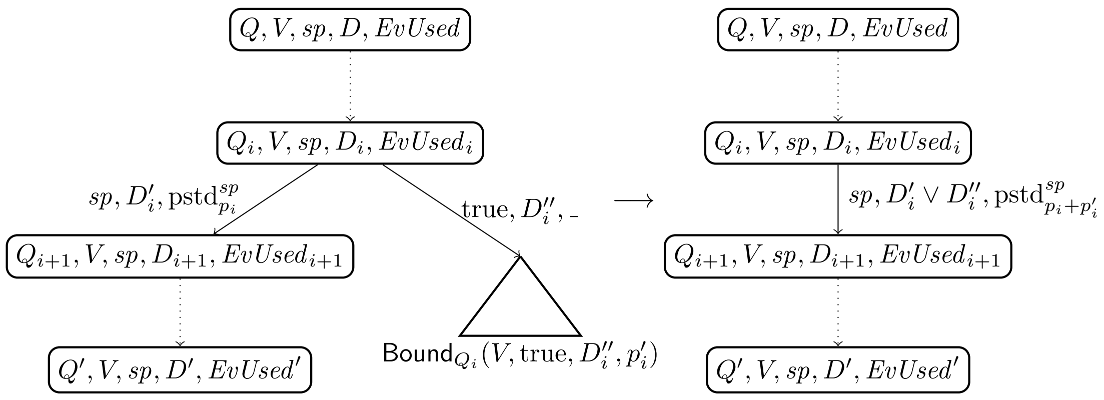

$Q, V, sp, D, EvUsed$
$Q_i, V, sp, D_i, EvUsed_i$
$sp, D'_i, \mathrm{pstd}^{sp}_{p_i}$
$\mathrm{true}, D''_i, \_$
$Q_{i+1}, V, sp, D_{i+1}, EvUsed_{i+1}$
$Q', V, sp, D', EvUsed'$
$\mathsf{Bound}_{Q_i}(V, \mathrm{true}, D''_i, p'_i)$
$\longrightarrow$
$Q, V, sp, D, EvUsed$
$Q_i, V, sp, D_i, EvUsed_i$
$sp, D'_i \vee D''_i, \mathrm{pstd}^{sp}_{p_i+p'_i}$
$Q_{i+1}, V, sp, D_{i+1}, EvUsed_{i+1}$
$Q', V, sp, D', EvUsed'$

Figure 15: Removing subtrees

Indeed, the main branch starts from the root $Q, V, sp, D, EvUsed$ to a leaf $Q', V, sp, D', EvUsed$, with additional subtrees starting from various nodes on this branch. Each node on the main branch is labeled $Q_i, V, sp, D_i, EvUsed_i$, and the edge of the main branch that starts from $Q_i, V, sp, D_i, EvUsed_i$ is labeled $sp, D'_i, \mathrm{pstd}^{sp}_{p_i}$ (see Figure 15). Suppose an additional subtree starts from a node $Q_i, V, sp, D_i, EvUsed_i$ with an edge labeled $\mathrm{true}, D''_i, \_$. This subtree yields a bound $\mathsf{Bound}_{Q_i}(V, \mathrm{true}, D''_i, p'_i)$. The edge of the main branch from $Q_i, V, sp, D_i, EvUsed_i$ yields a property $\mathcal{D}, \emptyset : Q_i, D'_i, EvUsed_i \xrightarrow{V}_{p_i} Q_{i+1}, D_{i+1}, EvUsed_{i+1}$. By Lemma 27, Property 4, the additional subtree can then be removed from the proof tree by replacing $sp, D'_i, \mathrm{pstd}^{sp}_{p_i}$ with $sp, D'_i \vee D''_i, \mathrm{pstd}^{sp}_{p_i+p'_i}$ as label of the edge of the main branch starting from the node $Q_i, V, sp, D_i, EvUsed_i$. By repeating this operation, we remove all additional subtrees, obtaining a proof tree that consists of a single branch with nodes labeled $Q_i, V, sp, D_i, EvUsed_i$, such that the edge that starts from $Q_i, V, sp, D_i, EvUsed_i$ is labeled $sp, D_i, \mathrm{pstd}^{sp}_{p_i}$. This yields a sequence of properties $\mathcal{D}, \emptyset : Q_i, D_i, EvUsed_i \xrightarrow{V}_{p_i} Q_{i+1}, D_{i+1}, EvUsed_{i+1}$, which yields a single such property $\mathcal{D}, \emptyset : Q, D, EvUsed \xrightarrow{V}_{p} Q', D', EvUsed'$ by transitivity (Lemma 19, Property 3).

2.7.5 Proof of Indistinguishability

To prove indistinguishability between two games G_0 and G_1, CryptoVerif finds a game G_2 such that $\mathcal{D}, \emptyset : G_0, D_{U0}, EvUsed \xrightarrow{V}_{p} G_2, D_2, EvUsed'$ and $\mathcal{D}, \emptyset : G_1, D_{U1}, EvUsed_1 \xrightarrow{V}_{p', D^+} G_2, D_2, EvUsed'_1$ where $\mathcal{D}$ is the set of all distinguishers, $EvUsed = \mathrm{event}(G_0)$, $EvUsed_1 = \mathrm{event}(G_1)$, $D_{U0} = \bigvee\{e \mid [\mathsf{unique}_e] \text{ occurs in } G_0\}$ and $D_{U1} = \bigvee\{e \mid [\mathsf{unique}_e] \text{ occurs in } G_1\}$. The active queries D_2 are also required to be the same in both sequences of games. (In general, CryptoVerif builds proof trees; they can be transformed into the properties above by Lemma 27, Property 4 as explained above. Only transformations that satisfy property preservation with introduction of events are allowed in the sequence of games that proves indistinguishability. The transformations **guess**, **guess_branch**, and **success simplify** are not allowed in that sequence, but are allowed in side branches that bound the probability of introduced events.) So for all evaluation contexts C acceptable for G_0 and G_2 with public variables V that do not contain events $EvUsed'$, and all distinguishers $D_0 \in \mathcal{D}$ that run in time at most t_{D_0},

$$\Pr[C[G_0] : D_0 \vee D_{U0}] \leq \Pr[C[G_2] : (D_0 \vee D_2) \wedge \neg\mathsf{NonUnique}_{G_2, D_2}] + p(C, t_{D_0}) \qquad (14)$$

since $\mathsf{NonUnique}_{G_0,D_{U0}} = D_{\mathrm{false}}$, and for all evaluation contexts C acceptable for G_1 and G_2 with public variables V that do not contain events in EvUsed'_1, and all distinguishers $D_0 \in \mathcal{D}$ that run in time at most t_{D_0},

$$\Pr[C[G_1] : D_0 \vee D_{U1}] \leq \Pr[C[G_2] : (D_0 \vee D_2) \wedge \neg\mathsf{NonUnique}_{G_2,D_2}] + p'(C, t_{D_0})\,. \tag{15}$$

Let C be an evaluation context acceptable for G_0 and G_1 with public variables V. After renaming the variables of C that do not occur in G_0 and G_1 and the tables of C that do not occur in G_0 and G_1 so that they do not occur in G_2, C is also acceptable for G_2 with public variables V. Furthermore, by Property 7, this renaming does not change the probabilities. Let $D_0 \in \mathcal{D}$ be a distinguisher that runs in time at most t_{D_0}. We rename the events of C in EvUsed' or EvUsed'_1 to some fresh events, and modify D_0 so that it considers the renamed events as if they were the original events. That does not change the probability $|\Pr[C[G_0] : D_0] - \Pr[C[G_1] : D_0]|$, and guarantees that C does not contain events in EvUsed' nor in EvUsed'_1. So

$$\begin{aligned}
\Pr[C[G_0] : D_0] &\leq \Pr[C[G_0] : D_0 \vee D_{U0}] \\
&\leq \Pr[C[G_2] : (D_0 \vee D_2) \wedge \neg\mathsf{NonUnique}_{G_2,D_2}] + p(C, t_{D_0}) \\
&\leq \Pr[C[G_2] : ((D_0 \wedge \neg D_2) \vee D_2) \wedge \neg\mathsf{NonUnique}_{G_2,D_2}] + p(C, t_{D_0}) \\
&\leq \Pr[C[G_2] : (D_0 \wedge \neg D_2) \wedge \neg\mathsf{NonUnique}_{G_2,D_2}] \\
&\qquad + \Pr[C[G_2] : D_2 \wedge \neg\mathsf{NonUnique}_{G_2,D_2}] + p(C, t_{D_0}) \\
&\leq \Pr[C[G_2] : D_0 \wedge \neg D_2] + \Pr[C[G_2] : D_2 \wedge \neg\mathsf{NonUnique}_{G_2,D_2}] + p(C, t_{D_0})
\end{aligned}$$

By applying (15) to $\neg D_0$, which is also in $\mathcal{D}$ and runs in the same time as D_0, we have

$$\begin{aligned}
1 - \Pr[C[G_1] : D_0] &= \Pr[C[G_1] : \neg D_0] \\
&\leq \Pr[C[G_1] : \neg D_0 \vee D_{U1}] \\
&\leq \Pr[C[G_2] : (\neg D_0 \vee D_2) \wedge \neg\mathsf{NonUnique}_{G_2,D_2}] + p'(C, t_{D_0}) \\
&\leq \Pr[C[G_2] : \neg D_0 \vee D_2] + p'(C, t_{D_0}) \\
&\leq 1 - \Pr[C[G_2] : D_0 \wedge \neg D_2] + p'(C, t_{D_0})
\end{aligned}$$

so

$$-\Pr[C[G_1] : D_0] \leq -\Pr[C[G_2] : D_0 \wedge \neg D_2] + p'(C, t_{D_0})$$

so

$$\Pr[C[G_0] : D_0] - \Pr[C[G_1] : D_0] \leq \Pr[C[G_2] : D_2 \wedge \neg\mathsf{NonUnique}_{G_2,D_2}] + p(C, t_{D_0}) + p'(C, t_{D_0})$$

By applying the formula above to $\neg D_0$, which runs in the same time as D_0, we have

$$\Pr[C[G_1] : D_0] - \Pr[C[G_0] : D_0] \leq \Pr[C[G_2] : D_2 \wedge \neg\mathsf{NonUnique}_{G_2,D_2}] + p(C, t_{D_0}) + p'(C, t_{D_0})$$

so

$$|\Pr[C[G_0] : D_0] - \Pr[C[G_1] : D_0]| \leq \Pr[C[G_2] : D_2 \wedge \neg\mathsf{NonUnique}_{G_2,D_2}] + p(C, t_{D_0}) + p'(C, t_{D_0})$$

so $G_0 \approx^V_{p''} G_1$ where $p''(C, t_{D_0}) = \Pr[C[G_2] : D_2 \wedge \neg\mathsf{NonUnique}_{G_2,D_2}] + p(C, t_{D_0}) + p'(C, t_{D_0})$.

2.7.6 Proof of query_equiv

Decisional case (query_equiv without [computational] **annotation)** The situation is similar to the proof of indistinguishability, but the property we want to prove is $\mathcal{D}_{\neg EvUsed_1}, \emptyset : G_0, D_{\text{false}}, \emptyset \xrightarrow{V}_p G_1, D_1, EvUsed_1$ where G_0 contains no events, $D_1 = e_1 \vee \ldots \vee e_m$, $e_1, \ldots, e_m$ are the Shoup events occurring in G_1, $e'_1, \ldots, e'_l$ are the non-unique events occurring in G_1, $EvUsed_1 = \{e_1, \ldots, e_m, e'_1, \ldots, e'_l\}$, $V = \emptyset$.

We need to show that, for all evaluation contexts C acceptable for G_0 and G_1 without public variables that do not contain events in $EvUsed_1$ and all distinguishers $D_0 \in \mathcal{D}_{\neg EvUsed_1}$,

$$\Pr[C[G_0] : D_0] \leq \Pr[C[G_1] : (D_0 \vee D_1) \wedge \neg \mathsf{NonUnique}_{G_1,D_1}] + p(C, t_{D_0})$$

CryptoVerif finds a game G_2 such that $\mathcal{D}_{\neg EvUsed'_0}, \emptyset : G_0, D_{\text{false}}, \emptyset \xrightarrow{V}_{p_0} G_2, D_2, EvUsed'_0$ and $\mathcal{D}_{\neg EvUsed''_1}, \emptyset : G_1, D_{U1}, EvUsed_1 \xrightarrow{V}_{p_1} G_2, D'_2, EvUsed'_1$ where $EvUsed''_1 = EvUsed'_1 \setminus \{e_1, \ldots, e_m\}$, $D_{U1} = e'_1 \vee \ldots \vee e'_l = \mathsf{NonUnique}_{G_1,D_1}$, and $D_2 = D_1 \vee D'_2$. (The events $e_1, \ldots, e_m$ must be preserved by the second proof, hence we allow distinguishers in $\mathcal{D}_{\neg EvUsed''_1}$ to use these events. The events $e_1, \ldots, e_m$ will be introduced in the first proof, and the active queries in G_2 are also required to match so $D_2 = D_1 \vee D'_2$.) So for all evaluation contexts C acceptable for G_0 and G_2 without public variables that do not contain events in $EvUsed'_0$, and all distinguishers $D_0 \in \mathcal{D}_{\neg EvUsed'_0}$ that run in time at most t_{D_0},

$$\Pr[C[G_0] : D_0] \leq \Pr[C[G_2] : (D_0 \vee D_2) \wedge \neg \mathsf{NonUnique}_{G_2,D_2}] + p_0(C, t_{D_0})$$

and for all evaluation contexts C acceptable for G_1 and G_2 without public variables that do not contain events in $EvUsed'_1$, and all distinguishers $D_0 \in \mathcal{D}_{\neg EvUsed''_1}$ that run in time at most t_{D_0},

$$\Pr[C[G_1] : D_0 \vee D_{U1}] \leq \Pr[C[G_2] : (D_0 \vee D'_2) \wedge \neg \mathsf{NonUnique}_{G_2,D'_2}] + p_1(C, t_{D_0})$$

In the last equation, we replace D_0 with $\neg D_0 \wedge \neg D_1$ for $D_0 \in \mathcal{D}_{\neg EvUsed'_1}$, yielding

$$\begin{aligned}&\Pr[C[G_1] : (\neg D_0 \wedge \neg D_1) \vee D_{U1}] \\ &\quad \leq \Pr[C[G_2] : ((\neg D_0 \wedge \neg D_1) \vee D'_2) \wedge \neg \mathsf{NonUnique}_{G_2,D'_2}] + p_1(C, t_{D_0})\end{aligned}$$

so

$$\begin{aligned}&\Pr[C[G_2] : ((D_0 \vee D_1) \wedge \neg D'_2) \vee \mathsf{NonUnique}_{G_2,D'_2}] \\ &\quad \leq \Pr[C[G_1] : (D_0 \vee D_1) \wedge \neg D_{U1}] + p_1(C, t_{D_0})\end{aligned}$$

Then we get

$$\begin{aligned}\Pr[C[G_0] : D_0] &\leq \Pr[C[G_2] : (D_0 \vee D_2) \wedge \neg \mathsf{NonUnique}_{G_2,D_2}] + p_0(C, t_{D_0}) \\ &\leq \Pr[C[G_2] : (D_0 \vee D_1 \vee D'_2) \wedge \neg \mathsf{NonUnique}_{G_2,D'_2}] + p_0(C, t_{D_0}) \\ &\leq \Pr[C[G_2] : ((D_0 \vee D_1) \wedge \neg D'_2) \vee \mathsf{NonUnique}_{G_2,D'_2}] \\ &\qquad + \Pr[C[G_2] : D'_2 \wedge \neg \mathsf{NonUnique}_{G_2,D'_2}] + p_0(C, t_{D_0}) \\ &\leq \Pr[C[G_1] : (D_0 \vee D_1) \wedge \neg D_{U1}] \\ &\qquad + p_1(C, t_{D_0}) + \Pr[C[G_2] : D'_2 \wedge \neg \mathsf{NonUnique}_{G_2,D'_2}] + p_0(C, t_{D_0}) \\ &\leq \Pr[C[G_1] : (D_0 \vee D_1) \wedge \neg \mathsf{NonUnique}_{G_1,D_1}] \\ &\qquad + p_1(C, t_{D_0}) + \Pr[C[G_2] : D'_2 \wedge \neg \mathsf{NonUnique}_{G_2,D'_2}] + p_0(C, t_{D_0})\end{aligned}$$

so we get the desired result with $p(C, t_{D_0}) = p_1(C, t_{D_0}) + \Pr[C[G_2] : D'_2 \wedge \neg \mathsf{NonUnique}_{G_2,D'_2}] + p_0(C, t_{D_0})$.

Computational case ($\mathsf{query\_equiv}$ with [computational] **annotation)** As in the decisional case, we want to prove $\mathcal{D}_{\neg \mathit{EvUsed}_1}, \emptyset : G_0, D_{\mathrm{false}}, \emptyset \xrightarrow{V}_p G_1, D_1, \mathit{EvUsed}_1$ where G_0 contains no events, $D_1 = e_1 \vee \ldots \vee e_m$, $e_1, \ldots, e_m$ are the Shoup events occurring in G_1, $e'_1, \ldots, e'_l$ are the non-unique events occurring in G_1, $\mathit{EvUsed}_1 = \{e_1, \ldots, e_m, e'_1, \ldots, e'_l\}$, $V = \emptyset$. Additionally, we want to show that the random values of G_0 and G_1 marked [unchanged] can be used in events (different from $e_1, \ldots, e_m$ since $e_1, \ldots, e_m$ have no arguments) in the game transformed using this assumption. That corresponds to adding oracles that execute the same arbitrary events using [unchanged] random values to both G_0 and G_1. We write G'_0 and G'_1 for the games G_0 and G_1 respectively with additional events and show $\mathcal{D}_{\neg \mathit{EvUsed}_1}, \emptyset : G'_0, D_{\mathrm{false}}, \mathit{EvUsed}_0 \xrightarrow{V}_p G'_1, D_1, \mathit{EvUsed}_1 \cup \mathit{EvUsed}_0$ where EvUsed_0 contains these additional events. These additional events can be observed by the adversary, so they are allowed in distinguishers in $\mathcal{D}_{\neg \mathit{EvUsed}_1}$.

Let $D_{U1} = e'_1 \vee \ldots \vee e'_l = \mathsf{NonUnique}_{G_1, D_1}$.

Let us write $O_0(\widetilde{r}, \widetilde{args})$ (resp. $O_1(\widetilde{r}, \widetilde{args})$) for the result of oracle O in game G_0 (resp. G_1) with randomness $\widetilde{r}$ and arguments $\widetilde{args}$.

In order to establish this property, we show that there exists a mapping ϕ of the randomness, such that if random value variable r has value v in G_0, then it has value $\phi_r(v)$ in G_1, ϕ_r is the identity when the variable r is marked [unchanged], ϕ_r preserves the probability distribution of variable r, and we define a game G_2 in which oracle O with randomness $\widetilde{r}$ and arguments $\widetilde{args}$ returns

$$\begin{aligned}
&\mathsf{let}\ x_0 = O_0(\widetilde{r}, \widetilde{args})\ \mathsf{in}\\
&\mathsf{let}\ x_1 = O_1(\phi(\widetilde{r}), \widetilde{args})\ \mathsf{in}\\
&\mathsf{if}\ x_0 = x_1\ \mathsf{then}\ x_0\ \mathsf{else}\ \mathsf{event\_abort}\ \mathrm{distinguish}
\end{aligned}$$

We bound

$$\begin{aligned}
p(C) &= \Pr[C[G_2] : \mathrm{distinguish} \vee \mathsf{NonUnique}_{G_2, D_{\mathrm{false}}}]\\
&= \Pr[C[G_2] : \mathrm{distinguish} \vee D_{U1}]\\
&= \mathsf{Adv}_{G_2}(C, \mathrm{distinguish} \Rightarrow \mathrm{false}, D_{U1})
\end{aligned}$$

From this bound, we infer the desired property.

We consider a game G_3 in which oracle O returns

$$\begin{aligned}
&\mathsf{let}\ x_0 = O_0(\widetilde{r}, \widetilde{args})\ \mathsf{in}\\
&\mathsf{let}\ x_1 = O_1(\phi(\widetilde{r}), \widetilde{args})\ \mathsf{in}\\
&x_0
\end{aligned}$$

We define G'_2 as G_2 with the same additional events as in G'_0 and G'_1, and G'_3 as G_3 with the same additional events as in G'_0 and G'_1. The game G'_3 behaves as G'_0 except that it executes a Shoup event e_i or a non-unique event when G_1 does, so we have, for any evaluation context C acceptable for G'_0 and G'_3 without public variables, and any distinguisher D_0,

$$\Pr[C[G'_0] : D_0] \leq \Pr[C[G'_3] : D_0 \vee D_1 \vee D_{U1}]$$

Moreover, G'_3 behaves as G'_1 except when G'_2 executes event distinguish, that is, when G_2 executes event distinguish (the additional events introduced in G'_2 are not needed to evaluate the probability of distinguish since we do not consider their probability), so for all D,

$$|\Pr[C[G'_3] : D] - \Pr[C[G'_1] : D]| \leq \Pr[C[G_2] : \mathrm{distinguish}] \tag{16}$$

Let C be any evaluation context acceptable for G'_0 and G'_2 without public variables. By renaming x_0 and x_1 to variables not in C, C is also acceptable for G'_0 and G'_3 without public variables. Let D_0 be any distinguisher. With $D = D_0 \vee D_1 \vee D_{U1}$ in (16), we obtain

$$\begin{aligned}\Pr[C[G'_0] : D_0] &\leq \Pr[C[G'_3] : D_0 \vee D_1 \vee D_{U1}]\\ &\leq \Pr[C[G'_1] : D_0 \vee D_1 \vee D_{U1}] + \Pr[C[G_2] : \text{distinguish}]\\ &\leq \Pr[C[G'_1] : (D_0 \vee D_1) \wedge \neg\mathsf{NonUnique}_{G_1,D_1}] + \Pr[C[G'_1] : D_{U1}]\\ &\quad\quad + \Pr[C[G_2] : \text{distinguish}]\\ &\leq \Pr[C[G'_1] : (D_0 \vee D_1) \wedge \neg\mathsf{NonUnique}_{G_1,D_1}] + p(C)\end{aligned}$$

since G'_1 and G_2 execute events D_{U1} in the same cases, and D_{U1} and distinguish are mutually exclusive. Therefore, we have $\mathcal{D}_{\neg EvUsed_1}, \emptyset : G'_0, D_{\text{false}}, EvUsed_0 \xrightarrow{V}_p G'_1, D_1, EvUsed_1 \cup EvUsed_0$.

Currently, CryptoVerif can prove query_equiv *only when the mapping ϕ is the identity for all variables. Other cases can be proved manually and used as assumptions in* equiv *statements.*

2.8 Turing Machine Adversary

In CryptoVerif, the adversary is modeled as an evaluation context. However, usually, in cryptographic results, an adversary is a bounded-time probabilistic Turing machine. In this section, we explain how any bounded-time probabilistic Turing machine that communicates on channels can be represented as a CryptoVerif evaluation context.

Let Q_0 be the initial game that interacts with an adversary. Let $c_1, \ldots, c_k$ be the channels used in Q_0. Let T_{all} be the union of all types that occur in Q_0. Let T'_{all} be the type of pairs containing the encoding a channel as first component and an element of T_{all} as second component. The encoding of a channel is either the constant *yield* or a tuple of integers $(j, i_1, \ldots, i_{k'})$ with $1 \leq j \leq k$. (We assume that unambiguous tuples can be encoded as CryptoVerif values, and that the constant *yield* is different from a tuple.) Let d_0, d_1, and d_2 be channels that do not occur in Q_0.

Let Q_1 be a process that contains the parallel composition of processes

$$!^{i_1 \leq n_1} \ldots !^{i_{k'} \leq n_{k'}} c_j[i_1, \ldots, i_{k'}](x : T_{\mathsf{all}}).\overline{d_0}\langle((j, i_1, \ldots, i_{k'}), x)\rangle$$

for each output $\overline{c_j[i'_1, \ldots, i'_{k'}]}\langle N\rangle$ that occurs under $!^{i'_1 \leq n_1} \ldots !^{i'_{k'} \leq n_{k'}}$ in Q_0. Since, in the initial game Q_0, the channels of all outputs use the current replication indices as channel indices, as in $c_j[i'_1, \ldots, i'_{k'}]$, a single output is executed for each value of the indices and for each syntactic occurrence of the output, so the inputs in Q_1 can receive all outputs made by Q_0. The process Q_1 forwards all these outputs to the same channel d_0, with a message that specifies both the channel $c_j[i_1, \ldots, i_{k'}]$ on which Q_0 emitted (encoded as a bitstring) and the message x sent by Q_0.

In addition, Q_1 also contains the parallel composition of processes

$$!^{i_1 \leq n_1} \ldots !^{i_{k'} \leq n_{k'}} \mathit{yield}().\overline{d_0}\langle(\mathit{yield}, ())\rangle$$

for each occurrence of yield that occurs under $!^{i'_1 \leq n_1} \ldots !^{i'_{k'} \leq n_{k'}}$ in Q_0, to receive all outputs that come from the yield construct.

Let $C =$ newChannel d_0; newChannel d_1; newChannel d_2; $(\mathit{start}().\overline{d_1[1]}\langle s_0\rangle \mid Q_1 \mid Q_2 \mid [\,])$, where the process Q_2 is defined in Figure 16. Let us explain how the context C can simulate any Turing machine interacting with the process Q_0.

The current state of the Turing machine is sent on channel $d_1[i]$ where i is a loop index that starts at 1 and increases during execution. As shown in the semantics of CryptoVerif, upon

```
1   $Q_2 = !^{i \leq n} d_1[i](s : \mathit{bitstring});$
2       let $(s', o, v) = f(s)$ in
            *Lines 3–6 are repeated for each $j \leq k$ and each $k'$*
            *such that there is an input on channel $c_j[i'_1, \dots, i'_{k'}]$ in $Q_0$.*
3       if $o = (j, k')$ then
4           let $(a_1, \dots, a_{k'}, b) = v$ in $\overline{c_j[a_1, \dots, a_{k'}]}\langle b \rangle;$
5           $d_0(s'' : T'_{\mathsf{all}}); \overline{d_1[i+1]}\langle f'(s', s'') \rangle$
6       else

7       if $o = \mathit{random}$ then
8           new $x : \mathit{bool}; \overline{d_1[i+1]}\langle f''(s', x) \rangle$
9       else
10      if $o = \mathit{abort}$ then
11          event_abort $e$
12      else
13          $\overline{d_2}\langle\rangle$
```

Figure 16: Looping process

startup, a message is sent on channel *start*. When C receives that message, it sends the initial state of the Turing machine s_0 on channel $d_1[1]$. This message is received by process Q_2 (line 1). Then Q_2 calls the function f on the current state s of the Turing machine (line 2). This function executes the Turing machine, until one of the following situations happens:

- The Turing machine sends a message b on a channel $c_j[a_1, \dots, a_{k'}]$; in this case, f returns $(s', (j, k'), (a_1, \dots, a_{k'}, b))$, where s' is the new state of the Turing machine. The test at line 3 is then going to succeed for the appropriate value of j, k', and the desired message is going to be sent at line 4. After receiving a message, the process Q_0 always replies by sending a message (except if it aborts). This message is going to be received by Q_1, which is going to forward on d_0 the channel and the received message. These channel and message are then received as s'' at line 5. Then $f'(s', s'')$ is the new state of the Turing machine after receiving that message. This state is sent on channel $d_1[i+1]$, which restarts a new iteration of Q_2.

- The Turing machine generates a fresh random bit; in this case, f returns $(s', \mathit{random}, ())$ where s' is the new state of the Turing machine. The test at line 7 is then going to succeed. At line 8, a random bit x is chosen. Then $f''(s', x)$ is the new state of the Turing machine with that random bit. This state is sent on channel $d_1[i+1]$, which restarts a new iteration of Q_2 as in the previous case.

- The Turing machine aborts; in this case, f returns $(s', \mathit{abort}, ())$. The test at line 10 is then going to succeed, and the process aborts at line 11. (The event e is any event not used elsewhere; the event is not really useful, it is present because the CryptoVerif language always executes an event before aborting.)

- The Turing machine stops; in this case, f returns $(s', \mathit{stop}, ())$. No test succeeds, so line 13 is executed. The process tries to send a message on channel d_2, but there is no input on this channel, so the process blocks.

The constants *random*, *abort*, and *stop* are assumed to be pairwise distinct, and distinct from all pairs.

The function f is a CryptoVerif primitive, because it can be implemented by a deterministic bounded-time Turing machine. (Recall that f stops when the initial probabilistic Turing machine makes a random choice, and the random choice is performed by CryptoVerif at lines 7–8.) Similarly, the function f' that computes the new state of the Turing machine from the old state and the received message, and the function f'' that computes the new state of the Turing machine from the old state and a random bit are CryptoVerif primitives.

The replication bound n (used in Q_2, line 1) is chosen large enough so that the loop never stops due to that bound: the Turing machine aborts or stops before the bound is reached. This is possible since the Turing machine runs in bounded time, so sends a bounded number of messages and chooses a bounded number of random bits.

Notice that, if Q_0 sends and receives messages on the same channels, it may happen that a message sent by Q_0 is immediately received by Q_0 without being intercepted by the adversary. In this case, since both Q_0 and Q_1 are going to listen on the same channels, the destination of the message (either the honest process Q_0 or the adversary Q_1) is chosen randomly with uniform probability, depending on the number of available receivers. Therefore, adding more copies of the receiving processes in Q_1 increases the probability that the adversary receives the message. Moreover, when the same channel is used for both inputs and outputs, the messages sent by Q_2 at line 4 may be received back by the adversary via Q_1, instead of being received by Q_0. We recommend avoiding this strange situation, by using distinct channels for inputs on the one hand and outputs on the other hand. More generally, we recommend using distinct channels for each input and output, so that the adversary gets full control of the network, as already mentioned page 13.

As a slight extension, it would still be possible to allow Q_0 to output on $c_j[i_1, \dots, i_{k'}]$ after receiving a message on the same channel $c_j[i_1, \dots, i_{k'}]$. In this case, a message sent by Q_0 on $c_j[i_1, \dots, i_{k'}]$ cannot be received by Q_0, because the input on $c_j[i_1, \dots, i_{k'}]$ is no longer available when the output on $c_j[i_1, \dots, i_{k'}]$ is performed by Q_0. Moreover, the problem that messages sent by Q_2 at line 4 may be received back by the adversary via Q_1, instead of being received by Q_0, can be avoided by putting the receiver process

$$c_j[a_1, \dots, a_{k'}](x : T_{\mathsf{all}}).\overline{d_0}\langle((j, a_1, \dots, a_{k'}), x)\rangle$$

after $\overline{c_j[a_1, \dots, a_{k'}]}\langle b\rangle$ in parallel with $d_0(s'' : T'_{\mathsf{all}}); \overline{d_1[i+1]}\langle f'(s', s'')\rangle$ in Q_2, instead of including

$$!^{i_1 \leq n_1} \dots !^{i_{k'} \leq n_{k'}} c_j[i_1, \dots, i_{k'}](x : T_{\mathsf{all}}).\overline{d_0}\langle((j, i_1, \dots, i_{k'}), x)\rangle$$

in Q_1.

The context C does not allow the Turing machine to execute events of its choice, while a CryptoVerif context can execute events. We could obviously extend the model to allow the Turing machine to execute events, but this is not needed for the cases we consider. Indeed, if the adversary represented as a CryptoVerif context executes events, these events can be deleted without changing the final result returned by the distinguisher: for correspondences, by Definition 11, the context is not allowed to contain events used by φ, and all other events are ignored by the distinguisher $\neg\varphi$; for one-session secrecy, secrecy, and bit secrecy, by Definitions 7 and 8, the context is not allowed to contain S nor $\overline{\mathsf{S}}$, and all other events are ignored by the distinguishers S and $\overline{\mathsf{S}}$.

To sum up, the context given in this section allows us to run any probabilistic bounded-time Turing machine as a CryptoVerif context, so CryptoVerif contexts are powerful enough to represent the adversaries usually considered by cryptographers.

3 Collecting True Facts

In this section, we consider only processes that satisfy Properties 4 and 5. We can assume without loss of generality that the adversary also satisfies these properties: the Turing machine adversary encoded in Section 2.8 satisfies them and tables (insert and get) can be removed by encoding them using find by transformation **expand_tables** (Section 5.1.2) and variables defined in conditions of find can be renamed to have distinct names by transformation **auto_SArename** (Section 5.1.1).

Given a configuration $\mathit{Conf} = E, \sigma, N, \mathcal{T}, \mu\mathcal{E}v$ or $\mathit{Conf} = E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ or $\mathit{Conf} = E, \mathcal{Q}, \mathcal{C}h$, we denote by E_{Conf} the environment E in configuration Conf. We denote by $E_{\mathit{Tr} \preceq \mathit{Conf}}$ the union of $E_{\mathit{Conf}'}$ for all configurations $\mathit{Conf}' \preceq_{\mathit{Tr}} \mathit{Conf}$ in Tr. It is a set of mappings $x[\widetilde{a}] \mapsto b$. At this stage, it may include conflicting mappings $x[\widetilde{a}] \mapsto b$ and $x[\widetilde{a}] \mapsto b'$ with $b \neq b'$. We prove below (Lemma 28) that this situation never happens. The notation $E_{\mathit{Tr} \preceq \mathit{Conf}}$ is useful because the environment computed in the semantics does not keep the values of variables defined in conditions of find after these conditions are evaluated. Considering the union of all environments of previous configurations allows us to recover the values of these variables, and to use them in the facts that we collect. Given a configuration $\mathit{Conf} = E, \sigma, N, \mathcal{T}, \mu\mathcal{E}v$ or $\mathit{Conf} = E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$, we denote by σ_{Conf} the mapping sequence for replication indices σ in the configuration Conf and by $\mu\mathcal{E}v_{\mathit{Conf}}$ the sequence of events $\mu\mathcal{E}v$ in configuration Conf.

Let us define *Defined* as in Section 2.4.3, except that

$$\mathit{Defined}(\sigma, {}^{\mu}\mathsf{find}[\mathit{unique?}]\ (\textstyle\bigoplus_{j=1}^{m} \widetilde{u_j}[\widetilde{i}] = \widetilde{i_j} \leq \widetilde{n_j}\ \mathsf{suchthat}\ \mathsf{defined}(\widetilde{M_j}) \wedge M_j\ \mathsf{then}\ N_j)\ \mathsf{else}\ N) =$$
$$\left(\biguplus_{j=1}^{m} \biguplus_{\widetilde{a} \leq \widetilde{n_j}} \mathit{Defined}(\sigma[\widetilde{i_j} \mapsto \widetilde{a}], M_j)\right) \uplus \max\left(\max_{j=1}^{m}\left(\{\widetilde{u_j}[\sigma(\widetilde{i})]\} \uplus \mathit{Defined}(\sigma, N_j)\right), \mathit{Defined}(\sigma, N)\right)$$
$$\mathit{Defined}(\sigma, {}^{\mu}\mathsf{find}[\mathit{unique?}]\ (\textstyle\bigoplus_{j=1}^{m} \widetilde{u_j}[\widetilde{i}] = \widetilde{i_j} \leq \widetilde{n_j}\ \mathsf{suchthat}\ \mathsf{defined}(\widetilde{M_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P) =$$
$$\left(\biguplus_{j=1}^{m} \biguplus_{\widetilde{a} \leq \widetilde{n_j}} \mathit{Defined}(\sigma[\widetilde{i_j} \mapsto \widetilde{a}], M_j)\right) \uplus \max\left(\max_{j=1}^{m}\left(\{\widetilde{u_j}[\sigma(\widetilde{i})]\} \uplus \mathit{Defined}(\sigma, P_j)\right), \mathit{Defined}(\sigma, P)\right)$$

so that the variables defined in conditions of find are now considered as defined forever, and not temporarily during the evaluation of the considered condition. We also define

$$\mathit{Defined}(\mathit{Tr} \preceq \mathit{Conf}) = \mathrm{Dom}(E_{\mathit{Tr} \preceq \mathit{Conf}}) \uplus \mathit{Defined}^{\mathrm{Fut}}(\mathit{Conf})\,.$$

Lemma 28 *Let Q_0 be a process that satisfies Properties 4 and 5. Let Tr be a trace of Q_0 and Conf be a configuration in the derivation of Tr. Then the following properties hold:*

1. *$\mathit{Defined}(\mathit{Tr} \preceq \mathit{Conf})$ does not contain duplicate elements.*

2. *Each variable is defined at most once for each value of its array indices in Tr.*

3. *$E_{\mathit{Tr} \preceq \mathit{Conf}}$ contains at most one binding for each $x[\widetilde{a}]$.*

Proof sketch The proof is similar to the proof of Lemma 9. We first show as in Lemma 9 that, for all program points μ in Q_0, if $\mathrm{Dom}(\sigma) = I_\mu$ are the current replication indices at μ and the process or term Q at μ satisfies Invariant 1, then all elements of $\mathit{Defined}(\sigma, Q)$ are of the form $x[\widetilde{a}]$ where $x \in \mathrm{vardef}(Q)$ and $\mathrm{Im}(\sigma)$ is a prefix of $\widetilde{a}$.

Next, we show that, for all program points μ, if $\mathrm{Dom}(\sigma) = I_\mu$ are the current replication indices at μ and the process or term Q at μ satisfies Invariant 1, then $\mathit{Defined}(\sigma, Q)$ does not contain duplicate elements. The proof proceeds by induction on Q. All multiset unions in the computation of $\mathit{Defined}(\sigma, Q)$ are disjoint unions by the property above, because either they use different extensions of σ (cases of replication and of conditions of find) or they use disjoint variable definitions or subprocesses or subterms in the same branch of find or if, which must define different variables by Invariant 1 and by Property 5.

We show by induction on the derivations that, if $\mathit{Conf} \xrightarrow{p}_t \mathit{Conf}'$, then $\mathit{Defined}(\mathit{Tr} \preceq \mathit{Conf}) \supseteq \mathit{Defined}(\mathit{Tr} \preceq \mathit{Conf}')$ and for all semantic configurations Conf'' in the derivation of $\mathit{Conf} \xrightarrow{p}_t \mathit{Conf}'$, $\mathit{Defined}(\mathit{Tr} \preceq \mathit{Conf}) \supseteq \mathit{Defined}(\mathit{Tr} \preceq \mathit{Conf}'')$, and similarly with $\rightsquigarrow$ instead of $\xrightarrow{p}_t$.

The first result follows: since Q_0 satisfies Invariant 1, $\mathit{Defined}(\sigma_0, Q_0)$ does not contain duplicate elements, where σ_0 is the empty mapping sequence. Let $\mathit{Conf}_0 = \emptyset, \{(\sigma_0, Q_0)\}, \mathrm{fc}(Q_0)$, $\mathit{Conf}_1 = \mathrm{reduce}(\emptyset, \{(\sigma_0, Q_0)\}, \mathrm{fc}(Q_0))$, $\mathit{Conf}_2 = \mathrm{initConfig}(Q_0)$, and Conf_3 be any other configuration of Tr. Then $\mathit{Defined}(\mathit{Tr} \preceq \mathit{Conf}_0)$, $\mathit{Defined}(\mathit{Tr} \preceq \mathit{Conf}_1)$, $\mathit{Defined}(\mathit{Tr} \preceq \mathit{Conf}_2)$, and therefore $\mathit{Defined}(\mathit{Tr} \preceq \mathit{Conf}_3)$ do not contain duplicate elements.

Let us prove the second result. In order to derive a contradiction, assume that two transitions $\mathit{Conf}_1 \xrightarrow{p_1}_{t_1} \mathit{Conf}'_1$ and $\mathit{Conf}_2 \xrightarrow{p_2}_{t_2} \mathit{Conf}'_2$ inside Tr define the same variable $x[\widetilde{a}]$.

- First case: one transition happens before the other, for instance $\mathit{Conf}'_1 \preceq_{\mathit{Tr}} \mathit{Conf}_2$. (The case $\mathit{Conf}'_2 \preceq_{\mathit{Tr}} \mathit{Conf}_1$ is symmetric.) Since $\mathit{Conf}_1 \xrightarrow{p_1}_{t_1} \mathit{Conf}'_1$ defines $x[\widetilde{a}]$, we have $x[\widetilde{a}] \in \mathrm{Dom}(E_{\mathit{Conf}'_1})$, so $x[\widetilde{a}] \in \mathrm{Dom}(E_{\mathit{Tr} \preceq \mathit{Conf}_2})$. Moreover, since $\mathit{Conf}_2 \xrightarrow{p_2}_{t_2} \mathit{Conf}'_2$ defines $x[\widetilde{a}]$, we have $x[\widetilde{a}] \in \mathit{Defined}^{\mathrm{Fut}}(\mathit{Conf}_2)$, by inspecting all rules that add elements to the environment. Therefore $\mathit{Defined}(\mathit{Tr} \preceq \mathit{Conf}_2) = \mathrm{Dom}(E_{\mathit{Tr} \preceq \mathit{Conf}_2}) \uplus \mathit{Defined}^{\mathrm{Fut}}(\mathit{Conf}_2)$ contains twice $x[\widetilde{a}]$. Contradiction.

- Second case: the transitions cannot be ordered. By definition of $\preceq_{\mathit{Tr}}$, this can happen only when a semantic rule uses several derivations for its assumptions, which happens only in rules for find. (Recall that get is excluded by Property 4.) Therefore, there exists k_1 and k_2 such that $\mathit{Conf}_1 \xrightarrow{p_1}_{t_1} \mathit{Conf}'_1$ is in the derivation of $\mathit{Conf}_{0,k} = E, \sigma[\widetilde{i_{j_k}} \mapsto \widetilde{a}_k], D_{j_k} \wedge M_{j_k}, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p_k}{}^{*}_{t_k} \mathit{Conf}'_{0,k} = E_k, \sigma_k, r_k, \mathcal{T}, \mu\mathcal{E}v$ with $v_k = (j_k, \widetilde{a}_k)$ for $k = k_1$ and $\mathit{Conf}_2 \xrightarrow{p_2}_{t_2} \mathit{Conf}'_2$ is in that derivation for $k = k_2$, with $k_1 \neq k_2$. We have $x[\widetilde{a}] \in \mathit{Defined}(\mathit{Tr} \preceq \mathit{Conf}'_{0,k_1}) \subseteq \mathit{Defined}(\mathit{Tr} \preceq \mathit{Conf}_{0,k_1}) = \mathrm{Dom}(E_{\mathit{Tr} \preceq \mathit{Conf}_{0,k_1}}) \cup \mathit{Defined}(\sigma[\widetilde{i_{j_{k_1}}} \mapsto \widetilde{a}_{k_1}], M_{j_{k_1}})$. Moreover, $\mathit{Conf}_{0,k} \preceq \mathit{Conf}_1$, so $\mathrm{Dom}(E_{\mathit{Tr} \preceq \mathit{Conf}_{0,k_1}}) \subseteq \mathrm{Dom}(E_{\mathit{Tr} \preceq \mathit{Conf}_1})$. Since $\mathit{Conf}_1 \xrightarrow{p_1}_{t_1} \mathit{Conf}'_1$ defines $x[\widetilde{a}]$, we have $x[\widetilde{a}] \in \mathit{Defined}^{\mathrm{Fut}}(\mathit{Conf}_1)$, by inspecting all rules that add elements to the environment. Since $\mathit{Defined}(\mathit{Tr} \preceq \mathit{Conf}_1) = \mathrm{Dom}(E_{\mathit{Tr} \preceq \mathit{Conf}_1}) \uplus \mathit{Defined}^{\mathrm{Fut}}(\mathit{Conf}_1)$ does not contain duplicate elements, we have $x[\widetilde{a}] \notin \mathrm{Dom}(E_{\mathit{Tr} \preceq \mathit{Conf}_1})$, so $x[\widetilde{a}] \notin \mathrm{Dom}(E_{\mathit{Tr} \preceq \mathit{Conf}_{0,k_1}})$. Hence we have $x[\widetilde{a}] \in \mathit{Defined}(\sigma[\widetilde{i_{j_{k_1}}} \mapsto \widetilde{a}_{k_1}], M_{j_{k_1}})$. Similarly, $x[\widetilde{a}] \in \mathit{Defined}(\sigma[\widetilde{i_{j_{k_2}}} \mapsto \widetilde{a}_{k_2}], M_{j_{k_2}})$. Let us show that the sets $\mathit{Defined}(\sigma[\widetilde{i_{j_{k_1}}} \mapsto \widetilde{a}_{k_1}], M_{j_{k_1}})$ and $\mathit{Defined}(\sigma[\widetilde{i_{j_{k_2}}} \mapsto \widetilde{a}_{k_2}], M_{j_{k_2}})$ are disjoint. We have $v_{k_1} \neq v_{k_2}$, so either $j_{k_1} \neq j_{k_2}$ and in this case these sets are disjoint because $M_{j_{k_1}}$ and $M_{j_{k_2}}$ define different variables by Property 5, or $j_{k_1} = j_{k_2}$ and $\widetilde{a}_{k_1} \neq \widetilde{a}_{k_2}$ and in this case these sets are disjoint because they use different extensions of σ. Since these sets are disjoint, they cannot both contain $x[\widetilde{a}]$. Contradiction.

The last result is an immediate consequence of the second one. □

Lemma 29 *Let Q_0 be a process that satisfies Properties 4 and 5. Let Tr be a trace of Q_0 and Conf be a configuration in the derivation of Tr. We have $E_{Tr \preceq Conf} = E_{Conf}[x[\widetilde{a}] \mapsto b$ for some variables x defined in a condition of* find *and some indices $\widetilde{a}$ and values $b]$.*

Proof sketch By induction on the derivation. □

The previous lemma shows that the only difference between E_{Conf} and $E_{Tr \preceq Conf}$ is that variables defined in conditions of find are added to $E_{Tr \preceq Conf}$. These variables have no array accesses, so they do not appear in defined conditions of find. Therefore, these defined conditions yield the same result whether they are evaluated in E_{Conf} or in $E_{Tr \preceq Conf}$.

We use *facts* the represent properties that hold at certain program points in processes. We consider the following facts:

- The boolean term M means that M evaluates to true.
- $\mathsf{defined}(M)$ means that M is defined (all array accesses in M are defined).
- $\mathsf{event}(e(\widetilde{M}))$ means that event $e(\widetilde{M})$ has been executed.
- $\mathsf{event}(e(\widetilde{M}))@\tau$ means that event $e(\widetilde{M})$ has been executed at step τ (index in the sequence of events $\mu\mathcal{E}v$).
- $M_1 : \mathsf{event}(e(\widetilde{M}))$ means that event $e(\widetilde{M})$ has been executed with pair (program point, replication indices) equal to M_1.
- $M_1 : \mathsf{event}(e(\widetilde{M}))@\tau$ means that event $e(\widetilde{M})$ has been executed at step τ with pair (program point, replication indices) equal to M_1.
- $\mathit{programpoint}(\mu, \widetilde{M})$ means that program point μ has been executed with replication indices equal to $\widetilde{M}$.
- $\mathit{programpoint}(\mathcal{S}_1, \widetilde{M_1}) \preceq \ldots \preceq \mathit{programpoint}(\mathcal{S}_m, \widetilde{M_m})$ means that, for $j \leq m$, some program point $\mu_j \in \mathcal{S}_j$ has been executed with replication indices equal to $\widetilde{M_j}$, and furthermore these program points have been executed in the order of increasing j.
- $\mathit{lastdefprogrampoint}(\mu, \widetilde{M})$ means that program point μ has been executed with replication indices equal to $\widetilde{M}$ and the values of variables and replication indices are unchanged since that program point (that is, no variable definition nor output that changes the replication indices was executed since that program point).

Given an environment E mapping process variables to their values, an environment ρ mapping replication indices and non-process variables of the formula φ to their values, and a sequence of events $\mu\mathcal{E}v$, we define $E, \rho, \mu\mathcal{E}v \vdash \varphi$, meaning that $E, \rho, \mu\mathcal{E}v$ satisfy φ, as follows:

- $E, \rho, \mu\mathcal{E}v \vdash M$ if and only if $E, \rho, M \Downarrow \text{true}$.
- $E, \rho, \mu\mathcal{E}v \vdash \mathsf{defined}(M)$ if and only if $E, \rho, M \Downarrow a$ for some a.
- $E, \rho, \mu\mathcal{E}v \vdash \mathsf{event}(e(\widetilde{M}))$ if and only if $E, \rho, \widetilde{M} \Downarrow \widetilde{a}$ and $(\mu, \widetilde{a'}) : e(\widetilde{a}) \in \mu\mathcal{E}v$ for some μ and $\widetilde{a'}$.

- $E, \rho, \mu\mathcal{E}v \vdash \mathsf{event}(e(\widetilde{M}))@M_0$ if and only if $E, \rho, \widetilde{M} \Downarrow \widetilde{a}$, $E, \rho, M_0 \Downarrow a_0$, and $\mu\mathcal{E}v(a_0) = (\mu, \widetilde{a'}) : e(\widetilde{a})$ for some μ and $\widetilde{a'}$.

- $E, \rho, \mu\mathcal{E}v \vdash M_1 : \mathsf{event}(e(\widetilde{M}))$ if and only if $E, \rho, \widetilde{M} \Downarrow \widetilde{a}$, $E, \rho, M_1 \Downarrow (\mu, \widetilde{a'})$ and $(\mu, \widetilde{a'}) : e(\widetilde{a}) \in \mu\mathcal{E}v$.

- $E, \rho, \mu\mathcal{E}v \vdash M_1 : \mathsf{event}(e(\widetilde{M}))@M_0$ if and only if $E, \rho, \widetilde{M} \Downarrow \widetilde{a}$, $E, \rho, M_0 \Downarrow a_0$, $E, \rho, M_1 \Downarrow (\mu, \widetilde{a'})$, and $\mu\mathcal{E}v(a_0) = (\mu, \widetilde{a'}) : e(\widetilde{a})$.

Logical connectives are defined as usual. When φ does not contain events, $\mu\mathcal{E}v$ can be omitted, writing $E, \rho \vdash \varphi$.

Let Tr be a trace of Q_0. Let $Conf = E, \sigma, N, \mathcal{T}, \mu\mathcal{E}v$ or $Conf = E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ be a configuration that occurs in the derivation of Tr. We define $Tr \preceq Conf, \rho \vdash \varphi$, meaning that the prefix of Tr until $Conf$ satisfies the formula φ with environment ρ (giving values of non-process variables of φ) as follows:

- $Tr \preceq Conf, \rho \vdash F$ if and only if $E_{Tr \preceq Conf}, \sigma_{Conf} \cup \rho, \mu\mathcal{E}v_{Conf} \vdash F$, when F is a term M, a defined fact $\mathsf{defined}(M)$, or an event $\mathsf{event}(e(\widetilde{M}))$, $\mathsf{event}(e(\widetilde{M}))@M_0$, $M_1 : \mathsf{event}(e(\widetilde{M}))$, or $M_1 : \mathsf{event}(e(\widetilde{M}))@M_0$.

- $Tr \preceq Conf, \rho \vdash \mathit{programpoint}(\mathcal{S}_1, \widetilde{M_1}) \preceq \dots \preceq \mathit{programpoint}(\mathcal{S}_m, \widetilde{M_m})$ if and only if, for all $j \in \{1, \dots, m\}$, there exists $Conf_j$ at program point $\mu_j \in \mathcal{S}_j$ in Tr such that $E_{Tr \preceq Conf}, \sigma_{Conf} \cup \rho, \widetilde{M_j} \Downarrow \mathrm{Im}(\sigma_{Conf_j})$ and $Conf_1 \preceq_{Tr} \dots \preceq_{Tr} Conf_m \preceq_{Tr} Conf$.

  The fact $\mathit{programpoint}(\mu, \widetilde{M})$ is actually a particular case of $\mathit{programpoint}(\mathcal{S}_1, \widetilde{M_1}) \preceq \dots \preceq \mathit{programpoint}(\mathcal{S}_m, \widetilde{M_m})$ with $m = 1$ and $\mathcal{S}_1 = \{\mu\}$. By specializing the definition above, we have $Tr \preceq Conf, \rho \vdash \mathit{programpoint}(\mu, \widetilde{M})$ if and only if there is a configuration $Conf'$ at program point μ in Tr such that $Conf' \preceq_{Tr} Conf$ and $E_{Tr \preceq Conf}, \sigma_{Conf} \cup \rho, \widetilde{M} \Downarrow \mathrm{Im}(\sigma_{Conf'})$.

- $Tr \preceq Conf, \rho \vdash \mathit{lastdefprogrampoint}(\mu, \widetilde{M})$ if and only if there is a configuration $Conf'$ at program point μ in Tr such that $Conf' \preceq_{Tr} Conf$, $E_{Tr \preceq Conf'} = E_{Tr \preceq Conf}$, $\sigma_{Conf'} = \sigma_{Conf}$, and $E_{Tr \preceq Conf}, \sigma_{Conf} \cup \rho, \widetilde{M} \Downarrow \mathrm{Im}(\sigma_{Conf'})$.

Logical connectives are defined as usual. Most facts are evaluated in the environment $E_{Tr \preceq Conf}$ and the mapping sequence σ_{Conf}. Events are evaluated using the sequence of events $\mu\mathcal{E}v_{Conf}$ in $Conf$, but correspond to an execution of the event at some point before $Conf$ in the trace.

We define that a trace Tr satisfies a logical formula φ with environment ρ (giving values of non-process variables of φ), denoted $Tr, \rho \vdash \varphi$ as $Tr \preceq Conf, \rho \vdash \varphi$, where Tr ends with $Conf$. Along the same line, we define $E_{Tr} = E_{Tr \preceq Conf}$ and $\mu\mathcal{E}v_{Tr} = \mu\mathcal{E}v_{Conf}$ where Tr ends with $Conf$.

When the formula φ does not contain free non-process variables, we may write $Tr \vdash \varphi$ instead of $Tr, \rho \vdash \varphi$ since the environment ρ is useless. When $\mathcal{F}$ is a set of formulas (in particular, of facts), we write $\bigwedge \mathcal{F}$ for $\bigwedge_{F \in \mathcal{F}} F$ and $\bigvee \mathcal{F}$ for $\bigvee_{F \in \mathcal{F}} F$. We also write $Tr, \rho \vdash \mathcal{F}$ when for all $\varphi \in \mathcal{F}$, $Tr, \rho \vdash \varphi$. This is equivalent to $Tr, \rho \vdash \bigwedge \mathcal{F}$. We use similar notations for prefixes $Tr \preceq Conf$ instead of traces Tr.

Additionally, we define the following facts:

- $[\![\mathit{elsefind}((i_1 \leq n_1, \dots, i_m \leq n_m), (M_1, \dots, M_l), M)]\!] = \forall i_1 \in [1, n_1], \dots, \forall i_m \in [1, n_m], \neg(\mathsf{defined}(M_1) \wedge \dots \wedge \mathsf{defined}(M_l) \wedge M)$.

- $[\![\mathit{elselet}(\widetilde{x} : \widetilde{T}, N, M)]\!] = \forall \widetilde{x} \in \widetilde{T}, N \neq M$.

3.1 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 $j \leq m$, $E(x_j) \in T_j$, then $E, M \Downarrow \mathrm{true}$.

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

  The term M reduces into M' by the rewrite rule $\forall x_1 : T_1, \dots, \forall x_m : T_m, M_1 \to M_2$ 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 $\mathsf{new}\ y_1 : T'_1, \dots, \mathsf{new}\ y_l : T'_l, \forall x_1 : T_1, \dots, \forall x_m : T_m, M_1 \approx_p M_2$ with $\mathrm{vardef}(M_2) \subseteq \mathrm{vardef}(M_1)$. Informally, these claims mean that M_1 and M_2 evaluate to the same bitstring except in cases of probability at most p, 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:

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

  where $\mathcal{A}$ is a probabilistic Turing machine.

  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 $\mathsf{new}\ y_1 : T'_1, \dots, \mathsf{new}\ y_l : T'_l, \forall x_1 : T_1, \dots, \forall x_m : T_m, M_1 \to M_2$.

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

$$\neg\mathrm{true} \to \mathrm{false} \qquad \neg\mathrm{false} \to \mathrm{true} \qquad \forall x : \mathit{bool}, \neg(\neg x) \to x$$
$$\forall x : T, \forall y : T, \neg(x = y) \to x \neq y$$
$$\forall x : T, \forall y : T, \neg(x \neq y) \to x = y$$
$$\forall x : T, x = x \to \mathrm{true} \qquad \forall x : T, x \neq x \to \mathrm{false}$$
$$\forall x : \mathit{bool}, \forall y : \mathit{bool}, \neg(x \wedge y) \to (\neg x) \vee (\neg y)$$
$$\forall x : \mathit{bool}, \forall y : \mathit{bool}, \neg(x \vee y) \to (\neg x) \wedge (\neg y)$$
$$\forall x : \mathit{bool}, x \wedge \mathrm{true} \to x \qquad \forall x : \mathit{bool}, x \wedge \mathrm{false} \to \mathrm{false}$$
$$\forall x : \mathit{bool}, x \vee \mathrm{true} \to \mathrm{true} \qquad \forall x : \mathit{bool}, x \vee \mathrm{false} \to x$$
$$\forall x : T, \forall y : T, \mathrm{if\_fun}(\mathrm{true}, x, y) \to x \qquad \forall x : T, \forall y : T, \mathrm{if\_fun}(\mathrm{false}, x, y) \to y$$

$$\begin{aligned}
&\forall x : \mathit{bool}, \forall y : T, \text{if\_fun}(x, y, y) \to y\\
&\forall x_1 : T_1, \dots, \forall x_m : T_m, \forall x : \mathit{bool}, \forall y : T_k, \forall z : T_k,\\
&\quad f(x_1, \dots, x_{k-1}, \text{if\_fun}(x, y, z), x_{k+1}, \dots, x_m) \to\\
&\qquad \text{if\_fun}(x, f(x_1, \dots, x_{k-1}, y, x_{k+1}, \dots, x_m), f(x_1, \dots, x_{k-1}, z, x_{k+1}, \dots, x_m))\\
&\quad \text{when } f : T_1 \times \dots \times T_m \to T \text{ has option } \textbf{autoSwapIf}
\end{aligned}$$

The prover also has support for commutative function symbols, that is, binary function symbols $f : T \times T \to T'$ such that for all $x, y \in T$, $f(x, y) = f(y, x)$. For such symbols, all equality and matching tests are performed modulo commutativity. The functions $\wedge$, $\vee$, $=$, and $\neq$ are commutative. So, for instance, the 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.

Example 4 For example, considering MAC and encryption schemes as in Definitions 2 and 3 respectively, we have:

$$\begin{aligned}
&\forall k : T_{mk}, \forall m : \mathit{bitstring},\\
&\quad \text{verify}(m, k, \text{mac}(m, k)) = \text{true}
\end{aligned} \tag{mac}$$

$$\begin{aligned}
&\forall m : \mathit{bitstring}; \forall k : T_k, \forall r : T_r,\\
&\quad \text{dec}(\text{enc}(m, k, r), k) = \text{i}_\perp(m)
\end{aligned} \tag{enc}$$

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 $\text{i}_\perp$.

3.2 Collecting True Facts from a Game

CryptoVerif collects a set of facts $\mathcal{F}_\mu$ that hold at each program point μ in the current game Q_0. Additionally, CryptoVerif also collects facts $\mathcal{F}_\mu^{\text{Fut}}$ (future facts at μ), which hold at the end of the block of code that contains μ and ends with an output or an $\mathsf{event\_abort}$ instruction that aborts the end. For instance, $\mathcal{F}_\mu^{\text{Fut}}$ may contain equalities that come from assignments performed after μ in the same block of code. (However, the facts in $\mathcal{F}_\mu^{\text{Fut}}$ m ay not hold in case a $\mathsf{find}[\mathsf{unique}_e]$ aborts because several choices make theconditions of that find succeed.) These sets of facts may contain facts M, $\mathsf{defined}(M)$, $M_1 : \mathsf{event}(e(\widetilde{M}))$, and $\mathit{programpoint}(\mu, \widetilde{M})$. In these sets of facts, all terms M must be simple.

Previous versions of the algorithm that collects facts were presented in [24, Appendix C.2] and [23, Appendix B.2]. The current algorithm is an extension that relies on the same principles. The facts $\mathit{programpoint}(\mu, \widetilde{M})$ is new. In particular, we have $\mathit{programpoint}(\mu, I_\mu) \in \mathcal{F}_\mu$. The algorithm that collects facts satisfies the following properties.

Lemma 30 *Let C be an evaluation context acceptable for Q_0, Tr be a trace of $C[Q_0]$, μ be a program point in Q_0, and $\mathcal{F}_\mu$ be computed in Q_0. If a configuration $Conf$ is at program point μ in Tr, then $Tr \preceq Conf \vdash \mathcal{F}_\mu$.*

Additionally, there is a more precise version of $\mathcal{F}_\mu$ that distinguishes cases depending on the program points at which the various variables are defined, generating several $\mathcal{F}_{\mu,c}$ for the various cases c. For this version, we have:

Lemma 31 *Let C be an evaluation context acceptable for Q_0, Tr be a trace of $C[Q_0]$, μ be a program point in Q_0, and $\mathcal{F}_{\mu,c}$ be computed in Q_0. If a configuration $Conf$ is at program point μ in Tr, then there exists c such that $Tr \preceq Conf \vdash \mathcal{F}_{\mu,c}$.*

$\mathcal{F}_\mu$ can be seen as a particular case of $\mathcal{F}_{\mu,c}$ by considering a single case c.

Corollary 3 *Let C be an evaluation context acceptable for Q_0, Tr be a trace of $C[Q_0]$, μ be a program point in Q_0, and $\mathcal{F}_\mu$ (resp. $\mathcal{F}_{\mu,c}$) be computed in Q_0. Let $Conf$ be a configuration at program point μ in Tr. Let θ be a renaming of I_μ to fresh replication indices and $\rho = \{\theta I_\mu \mapsto \sigma_{Conf} I_\mu\}$. Let $Conf'$ be a term or output process configuration in Tr such that $Conf \preceq_{Tr} Conf'$.*

We have $Tr \preceq Conf', \rho \vdash \theta\mathcal{F}_\mu$ and there exists c such that $Tr \preceq Conf', \rho \vdash \theta\mathcal{F}_{\mu,c}$.

In particular, $Tr, \rho \vdash \theta\mathcal{F}_\mu$ and there exists c such that $Tr, \rho \vdash \theta\mathcal{F}_{\mu,c}$.

Proof By Lemma 30, $Tr \preceq Conf \vdash \mathcal{F}_\mu$. By Lemma 31, there exists c such that $Tr \preceq Conf \vdash \mathcal{F}_{\mu,c}$. Let $\mathcal{F} = \mathcal{F}_\mu$ (resp. $\mathcal{F} = \mathcal{F}_{\mu,c}$) such that $Tr \preceq Conf \vdash \mathcal{F}$. By definition of ρ, we have $Tr \preceq Conf, \rho \vdash \theta\mathcal{F}$. Since $Conf \preceq_{Tr} Conf'$, the environment $E_{Tr \preceq Conf'}$ is an extension of $E_{Tr \preceq Conf}$, so the terms and defined facts in $\theta\mathcal{F}$ are preserved when considering $Tr \preceq Conf'$ instead of $Tr \preceq Conf$. (They do not use σ_{Conf}, resp. $\sigma_{Conf'}$, by the renaming θ.) Moreover, by Lemma 3, $\mu\mathcal{E}v_{Conf'}$ is an extension of $\mu\mathcal{E}v_{Conf}$, so the events are also preserved. By definition of $Tr \preceq Conf, \rho \vdash \mathit{programpoint}(\mathcal{S}_1, \widetilde{M_1}) \preceq \ldots \preceq \mathit{programpoint}(\mathcal{S}_m, \widetilde{M_m})$, the sequences of program points $\mathit{programpoint}(\mathcal{S}_1, \widetilde{M_1}) \preceq \ldots \preceq \mathit{programpoint}(\mathcal{S}_m, \widetilde{M_m})$ are also preserved. Since $\mathcal{F}$ is a set of facts containing only terms, defined facts, events, and sequences of program points, we conclude that $Tr \preceq Conf', \rho \vdash \theta\mathcal{F}$.

The last point is obtained by choosing $Conf'$ to be the last configuration of Tr. □

Lemma 32 *Let C be an evaluation context acceptable for Q_0, $Tr = \mathrm{initConfig}(C[Q_0]) \xrightarrow{p}_t \ldots \xrightarrow{p'}_{t'} E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ be a trace that does not execute any non-unique event of Q_0 with $P = \overline{c[\widetilde{a}]}\langle a\rangle; Q$ for some c, $\widetilde{a}$, a, and Q or $P = \mathsf{abort}$, μ be a program point in Q_0, and $\mathcal{F}_\mu^{\mathrm{Fut}}$ be computed in Q_0. If the configuration $Conf$ is at program point μ in Tr and no executed process in the configurations between the configuration at the end of reduction step that contains $Conf$ (included) and $E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ (excluded) is of the form $\overline{c[\widetilde{a}]}\langle a\rangle; Q$ for some $\widetilde{a}$, a, and Q (when $Conf$ is a process configuration with process $\overline{c[\widetilde{a}]}\langle a\rangle; Q$ for some c, $\widetilde{a}$, a, and Q, we have $Conf = E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$), then $Tr \vdash \mathcal{F}_\mu^{\mathrm{Fut}}$.*

3.3 Equational Prover

In order to reason on facts, CryptoVerif uses an equational prover: from a set of facts $\mathcal{F}$, this equational prover tries to derive a contradiction by rewriting terms, using an algorithm inspired by Knuth-Bendix completion. It also eliminates collisions between independent random values, thus the contradiction is obtained up to the probability of the eliminated collisions, that is, the probability that $\mathcal{F}$ holds is bounded by the probability of these collisions. When this algorithm succeeds, we say that "$\mathcal{F}$ yields a contradiction in game Q_0", and CryptoVerif computes the probbaility of the eliminated collisions. (We may omit the current game Q_0 when it is clear from the context.) Previous versions of this algorithm were presented in [24, Appendix C.5] and [23, Appendix B.3]. Those versions did not evaluate the probability because they considered asymptotic security: they showed that the probability was negligible in the security parameter. Here, we use exact security: we compute the value of the probabilities, so the soundness of this algorithm can be expressed by the following lemma, adapted from [23, Proposition 7].

Lemma 33 *If for all $j \in J$, $\mathcal{F}_j$ yields a contradiction in a game Q_0, then CryptoVerif returns a probability p such that for all evaluation contexts C acceptable for Q_0 with any public variables,* $\Pr[C[Q_0] \preceq \bigvee_{j\in J} \exists \widetilde{x}_j \in \widetilde{T}_j, \bigwedge \mathcal{F}_j] \leq p(C)$*, where $\widetilde{x}_j$ are the replication indices and non-process variables that occur in $\mathcal{F}_j$ and $\widetilde{T}_j$ are their types.*

In particular, the lemma states that, when several sets of facts $\mathcal{F}_j$ yield a contradiction in the same game, CryptoVerif counts only once in the probability p the collisions that are eliminated in proofs that $\mathcal{F}_j$ yields a contradiction for several j.

More generally, let us consider an algorithm φ built from the following grammar:

$\varphi ::=$	algorithm
$\mathcal{F}$ yields a contradiction	equational proof
$\varphi_1 \wedge \varphi_2$	conjunction
$\varphi_1 \vee \varphi_2$	disjunction
ψ	mathematical formula
if ψ then φ_1 else φ_2	test

The mathematical formulas ψ in such algorithms must not depend on the executed trace. (They may depend on the syntax of the game Q_0 or on the set of public variables V, for instance.)

We translate such algorithms into logical formulas on traces:

$$\{\![\mathcal{F} \text{ yields a contradiction}]\!\} = \neg\exists \widetilde{x} \in \widetilde{T}, \bigwedge \mathcal{F} \quad \text{where } \widetilde{x} \text{ are the replication indices and}$$
$$\text{non-process variables that occur in } \mathcal{F} \text{ and } \widetilde{T} \text{ are their types.}$$
$$\{\![\varphi_1 \wedge \varphi_2]\!\} = \{\![\varphi_1]\!\} \wedge \{\![\varphi_2]\!\}$$
$$\{\![\varphi_1 \vee \varphi_2]\!\} = \begin{cases} \{\![\varphi_1]\!\} & \text{if } \varphi_1 \\ \{\![\varphi_2]\!\} & \text{otherwise} \end{cases}$$
$$\{\![\psi]\!\} = \psi$$
$$\{\![\text{if } \psi \text{ then } \varphi_1 \text{ else } \varphi_2]\!\} = \text{if } \psi \text{ then } \{\![\varphi_1]\!\} \text{ else } \{\![\varphi_2]\!\}$$

Intuitively, when algorithm φ returns true, CryptoVerif shows that the formula $\{\![\varphi]\!\}$ holds for most traces. It bounds the probability of the traces for which this formula does not hold, as shown by the following lemma.

Lemma 34 *If algorithm φ returns true in a game Q_0, then CryptoVerif returns a probability p such that for all evaluation contexts C acceptable for Q_0 with any public variables,* $\Pr[C[Q_0] \preceq \neg\{\![\varphi]\!\}] \leq p(C)$.

Proof We show by induction on the definition of φ that, if φ returns true and $Tr \vdash \neg\{\![\varphi]\!\}$, then there exists $\mathcal{F}$ such that "$\mathcal{F}$ yields a contradiction" has been called in the evaluation of φ and returned true, and $Tr \vdash \exists \widetilde{x} \in \widetilde{T}, \bigwedge \mathcal{F}$ where $\widetilde{x}$ are the replication indices and non-process variables that occur in $\mathcal{F}$ and $\widetilde{T}$ are their types.

- Case $\varphi = (\mathcal{F}$ yields a contradiction): obvious.

- Case $\varphi = \varphi_1 \wedge \varphi_2$: Since φ returns true, φ_1 and φ_2 both return true. Since $\mathit{Tr} \vdash \neg(\{\![\varphi_1]\!\} \wedge \{\![\varphi_2]\!\})$, we have either $\mathit{Tr} \vdash \neg\{\![\varphi_1]\!\}$ or $\mathit{Tr} \vdash \neg\{\![\varphi_2]\!\}$. In the first case, by induction hypothesis on φ_1, there exists $\mathcal{F}$ such that "$\mathcal{F}$ yields a contradiction" has been called in the evaluation of φ_1 and returned true, and $\mathit{Tr} \vdash \exists \widetilde{x} \in \widetilde{T}, \bigwedge \mathcal{F}$ where $\widetilde{x}$ are the replication indices and non-process variables that occur in $\mathcal{F}$ and $\widetilde{T}$ are their types. Moreover, "$\mathcal{F}$ yields a contradiction" has been called in the evaluation of φ. The second case is symmetric.

- Case $\varphi = \varphi_1 \vee \varphi_2$: If φ_1 returns true, then $\{\![\varphi]\!\} = \{\![\varphi_1]\!\}$, so $\mathit{Tr} \vdash \neg\{\![\varphi_1]\!\}$. We conclude by induction hypothesis on φ_1, as above. If φ_1 returns false, then φ_2 returns true, since φ returns true. Hence $\{\![\varphi]\!\} = \{\![\varphi_2]\!\}$, so $\mathit{Tr} \vdash \neg\{\![\varphi_2]\!\}$. We conclude by induction hypothesis on φ_2.

- Case $\varphi = \psi$: since ψ evaluates to true, there is no trace Tr such that $\mathit{Tr} \vdash \neg\psi$, so the property holds trivially.

- Case $\varphi =$ if ψ then φ_1 else φ_2: if ψ evaluates to true, then we conclude by induction hypothesis on φ_1. Indeed, since φ returns true, φ_1 returns true. Since $\mathit{Tr} \vdash \neg\{\![\varphi]\!\}$, we have $\mathit{Tr} \vdash \neg\{\![\varphi_1]\!\}$. By induction hypothesis, there exists $\mathcal{F}$ such that "$\mathcal{F}$ yields a contradiction" has been called in the evaluation of φ_1 and returned true, and $\mathit{Tr} \vdash \exists \widetilde{x} \in \widetilde{T}, \bigwedge \mathcal{F}$ where $\widetilde{x}$ are the replication indices and non-process variables that occur in $\mathcal{F}$ and $\widetilde{T}$ are their types. Then "$\mathcal{F}$ yields a contradiction" has also been called in the evaluation of φ. Similarly, if ψ evaluates to false, then we conclude by induction hypothesis on φ_2.

We conclude by Lemma 33. □

4 success: Criteria for Proving Security Properties

The command **success** tries to prove the active queries, as explained below. We consider a process Q_0 that satisfies Properties 4 and 5, and prove secrecy and correspondence properties for Q_0.

4.1 Secrecy

Let us now define syntactic criteria that allow us to prove secrecy properties of protocols. We first define the function noleak in Figure 17 and explain it below. This function implicitly depends on the current game Q_0 and the public variables V. The function call noleak$(x[\widetilde{M'}], \mathcal{I}, \mathcal{F})$ shows that $x[\widetilde{M'}]$ does not leak to the adversary, assuming $\mathcal{F}$ holds. The set $\mathcal{I}$ contains all replication indices that appear in $\mathcal{F}$. If $\mathcal{F}$ yields a contradiction, noleak is true, since it shows the absence of leak *assuming* $\mathcal{F}$ *holds.* Otherwise, $x[\widetilde{M'}]$ may leak either because $x \in V$ so x is a public variable, or because of an occurrence of a term $x[\widetilde{M}]$ in the game that reads $x[\widetilde{M'}]$ (so $\widetilde{M} = \widetilde{M'}$ holds) and such that the result of $x[\widetilde{M}]$ leaks.

- In case $x[\widetilde{M}]$ occurs in the term M in an assignment $\mathsf{let}\ y[\widetilde{i}] = M$, the function noleak recursively tries to prove that $y[\widetilde{i}]$ does not leak, when $y[\widetilde{i}]$ may use $x[\widetilde{M'}]$, that is, when $\widetilde{M'} = \widetilde{M}$. The fact $\widetilde{M'} = \widetilde{M}$ and the facts $\mathcal{F}_\mu$ that hold at the program point μ of $x[\widetilde{M}]$ are added to the known facts $\mathcal{F}$ in the recursive call. Indeed, these facts are known to hold in this case. The replication indices are renamed to fresh indices in order to avoid using the same index variable for indices that can actually take different values.

$$\text{noleak}(x[\widetilde{M'}], \mathcal{I}, \mathcal{F}) = (\mathcal{F} \text{ yields a contradiction}) \vee ((x \notin V) \wedge \bigwedge_{{}^{\mu}x[\widetilde{M}] \text{ in } Q_0}$$

– if ${}^{\mu}x[\widetilde{M}]$ is in M in an assignment $\mathsf{let}\ y[\widetilde{i}] = M$ in Q_0, M is built from replication indices, variables, function applications, and conditionals, and the current call is not inside a call to noleak($y[\_], \_, \_$), then

noleak($y[\widetilde{\theta i}], \mathcal{I} \cup \{\widetilde{\theta i}\}, \mathcal{F} \cup \theta\mathcal{F}_{\mu} \cup \{\theta\widetilde{M} = \widetilde{M'}\}$)
where θ is a renaming of $\widetilde{i}$ to fresh replication indices

– if ${}^{\mu}x[\widetilde{M}]$ is in $\mathsf{event}\ e(M_1, \dots, M_{k-1}, C[{}^{\mu}x[\widetilde{M}]], M_{k+1}, \dots, M_m)$ in Q_0 for C defined in Figure 18, then true

– otherwise, $\mathcal{F} \cup \theta\mathcal{F}_{\mu} \cup \{\theta\widetilde{M} = \widetilde{M'}\}$ yields a contradiction
where θ is a renaming of I_{μ} to fresh replication indices)

Figure 17: Function noleak

- In case $x[\widetilde{M}]$ occurs in the arguments of an event, the arguments of the event do not leak to the adversary, so this occurrence of $x[\widetilde{M}]$ does not make $x[\widetilde{M'}]$ leak.
- In all other cases, we consider that the result of $x[\widetilde{M}]$ may leak, so, in order to prove that $x[\widetilde{M'}]$ does not leak, we show that the occurrence of $x[\widetilde{M}]$ at μ cannot read $x[\widetilde{M'}]$, by showing that $\widetilde{M} = \widetilde{M'}$, $\mathcal{F}_{\mu}$, and $\mathcal{F}$ together yield a contradiction.

Definition 15 (μ follows a definition of x) We say that μ *follows a definition of* x when $\mathsf{new}\ x[\widetilde{i}] : T; {}^{\mu} \dots$, $\mathsf{let}\ x[\widetilde{i}] = M\ \mathsf{in}\ {}^{\mu} \dots$, $\mathsf{find}[unique?]\ (\bigoplus_{j=1}^{m} \widetilde{u_j}[\widetilde{i}] = \widetilde{i_j} \leq \widetilde{n_j}\ \mathsf{suchthat}\ \mathsf{defined}(\widetilde{M_j}) \wedge M_j\ \mathsf{then}\ {}^{\mu_j} \dots)\ \mathsf{else}\ \dots$ with $\mu_j = \mu$ and x in $\widetilde{u_j}$ for some $j \leq m$, or $c[\widetilde{M}](x[\widetilde{i}] : T); {}^{\mu}P$ occurs in Q_0.

We do not mention get in the previous definition, because it is excluded by Property 4. For each μ that follows a definition of x in Q_0, we define defRand$_{\mu}(x)$ as follows:

$$\text{defRand}_{\mu}(x) = \begin{cases} x[\widetilde{i}] & \text{if } \mathsf{new}\ x[\widetilde{i}] : T; {}^{\mu} \dots \text{ occurs in } Q_0 \\ y[\widetilde{M}] & \text{if } \mathsf{let}\ x[\widetilde{i}] : T = y[\widetilde{M}]\ \mathsf{in}\ {}^{\mu} \dots \text{ occurs in } Q_0 \text{ and} \\ & y \text{ is defined only by random choices in } Q_0 \end{cases}$$

In all other cases, defRand$_{\mu}(x)$ is not defined. The variable defRand$_{\mu}(x)$ is the random variable that defines x just before program point μ. When x itself is chosen randomly at that point, defRand$_{\mu}(x)$ is simply $x[\widetilde{i}]$, where $\widetilde{i}$ are the current replication indices. When x is defined by an assignment of a variable $y[\widetilde{M}]$ that is random, defRand$_{\mu}(x)$ is that variable. Otherwise, we give up and do not define defRand$_{\mu}(x)$.

$$\text{prove}^{\text{1-ses.secr.}(x)}(\mu) = \text{defRand}_{\mu}(x) \text{ is defined and}$$
$$\text{noleak}(\theta\text{defRand}_{\mu}(x), \{\theta I_{\mu}\}, \theta\mathcal{F}_{\mu})$$
$$\text{where } \theta \text{ is a renaming of } I_{\mu} \text{ to fresh replication indices}$$
$$\text{prove}^{\text{1-ses.secr.}(x)}(\mathcal{S}) = \bigwedge_{\mu \in \mathcal{S}} \text{prove}^{\text{1-ses.secr.}(x)}(\mu)$$

$$
\begin{aligned}
C ::= {} & [\,] \\
& y[M_1, \dots, M_{k-1}, C, M_{k+1}, \dots, M_m] \\
& f(M_1, \dots, M_{k-1}, C, M_{k+1}, \dots, M_m) \\
& \mathsf{new}\ y[\widetilde{i}] : T; C \\
& \mathsf{let}\ y[\widetilde{i}] = M\ \mathsf{in}\ C \\
& \mathsf{if}\ M\ \mathsf{then}\ C\ \mathsf{else}\ N' \\
& \mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ C \\
& \mathsf{find}[unique?]\ (\textstyle\bigoplus_{j=1,\dots,m; j \neq k} \widetilde{u_j}[\widetilde{i}] = \widetilde{i_j} \leq \widetilde{n_j}\ \mathsf{suchthat}\ \mathsf{defined}(\widetilde{M_j}) \wedge M_j\ \mathsf{then}\ N_j) \\
& \qquad \oplus \widetilde{u_k}[\widetilde{i}] = \widetilde{i_k} \leq \widetilde{n_k}\ \mathsf{suchthat}\ \mathsf{defined}(\widetilde{M_k}) \wedge M_k\ \mathsf{then}\ C\ \mathsf{else}\ N \\
& \mathsf{find}[unique?]\ (\textstyle\bigoplus_{j=1}^{m} \widetilde{u_j}[\widetilde{i}] = \widetilde{i_j} \leq \widetilde{n_j}\ \mathsf{suchthat}\ \mathsf{defined}(\widetilde{M_j}) \wedge M_j\ \mathsf{then}\ N_j)\ \mathsf{else}\ C \\
& \mathsf{event}\ e(\widetilde{M}); C
\end{aligned}
$$

Figure 18: Event contexts

The function call $\mathrm{prove}^{\mathsf{1\text{-}ses.secr.}(x)}(\mu)$ proves one-session secrecy for the definition of x just before program point μ. It considers only the cases in which x is defined either by a random choice or by an assignment from a random choice. In other cases, the proof fails. (These other cases can typically be handled by first removing assignments as needed.) Intuitively, $\mathrm{prove}^{\mathsf{1\text{-}ses.secr.}(x)}(\mu)$ guarantees that, when $x[\widetilde{i}]$ is defined just before program point μ, the random variable $\mathrm{defRand}_\mu(x)$ that defines $x[\widetilde{i}]$ does not leak, knowing that the facts $\mathcal{F}_\mu$ hold. Only events and variables $y[\widetilde{i'}]$ that do not leak depend on the random choice that defines $x[\widetilde{i}]$; the sent messages and the control flow of the process are independent of $x[\widetilde{i}]$, so the adversary obtains no information on $x[\widetilde{i}]$. That guarantees the one-session secrecy of $x[\widetilde{i}]$ when it is defined just before μ. This is verified for all program points in $\mathcal{S}$ by $\mathrm{prove}^{\mathsf{1\text{-}ses.secr.}(x)}(\mathcal{S})$. When x is defined by assignment of $z[\widetilde{M}]$, this proof of one-session secrecy allows some array cells of z to leak, provided the array cells $z[\widetilde{M}]$ used to define x do not leak.

In order to prove secrecy, we also define $\mathrm{prove}^{\mathsf{distinct}(x)}(\mu_1, \mu_2) = (z_1 \neq z_2) \vee (\theta_1\mathcal{F}_{\mu_1} \cup \theta_2\mathcal{F}_{\mu_2} \cup \{\theta_1\widetilde{M_1} = \theta_2\widetilde{M_2}, \widetilde{i}_1 \neq \widetilde{i}_2\}$ yields a contradiction), where $\mathrm{defRand}_{\mu_1}(x) = z_1[\widetilde{M_1}]$, $\mathrm{defRand}_{\mu_2}(x) = z_2[\widetilde{M_2}]$, $\widetilde{i}$ are the current replication indices at the definition of x, θ_1 and θ_2 are two distinct renamings of $\widetilde{i}$ to fresh replication indices, $\widetilde{i}_1 = \theta_1\widetilde{i}$, and $\widetilde{i}_2 = \theta_2\widetilde{i}$. Intuitively, $\mathrm{prove}^{\mathsf{distinct}(x)}(\mu_1, \mu_2)$ guarantees that, if $x[\widetilde{i}_1]$ is defined at μ_1, so $x[\widetilde{i}_1] = z_1[\theta_1\widetilde{M_1}]$, and $x[\widetilde{i}_2]$ is defined at μ_2, so $x[\widetilde{i}_2] = z_2[\theta_2\widetilde{M_2}]$, with $\widetilde{i}_1 \neq \widetilde{i}_2$, then the random variables that define x in these two cases, $z_1[\theta_1\widetilde{M_1}]$ and $z_2[\theta_2\widetilde{M_2}]$, are different, that is, $z_1 \neq z_2$ or $\theta_1\widetilde{M_1} \neq \theta_2\widetilde{M_2}$. Therefore, $z_1[\theta_1\widetilde{M_1}]$ is independent of $z_2[\theta_2\widetilde{M_2}]$, so $x[\widetilde{i}_1]$ is independent of $x[\widetilde{i}_2]$. Combining this information with the proof of one-session secrecy, we can prove secrecy of x: we define

$$\mathrm{prove}^{\mathsf{Secrecy}(x)}(\mathcal{S}) = \mathrm{prove}^{\mathsf{1\text{-}ses.secr.}(x)}(\mathcal{S}) \wedge \bigwedge_{\mu_1, \mu_2 \in \mathcal{S}} \mathrm{prove}^{\mathsf{distinct}(x)}(\mu_1, \mu_2)$$

The proof of bit secrecy is the same as for one-session secrecy:

$$\mathrm{prove}^{\mathsf{bit\ secr.}(x)}(\mathcal{S}) = \mathrm{prove}^{\mathsf{1\text{-}ses.secr.}(x)}(\mathcal{S})$$

The proof of (one-session or bit) secrecy is justified by the following proposition.

Proposition 1 ((One-session or bit) secrecy) *Consider a process Q_0 that satisfies Properties 4 and 5. Let sp be* $\mathsf{1\text{-}ses.secr.}(x)$*,* $\mathsf{Secrecy}(x)$*, or* $\mathsf{bit\ secr.}(x)$*. Let $\mathcal{S} = \{\mu \mid \mu$ follows a definition of $x\}$. If $\mathrm{prove}^{sp}(\mathcal{S})$ and for all evaluation contexts C acceptable for Q_0, $\Pr[C[Q_0] \preceq \neg\{\![\mathrm{prove}^{sp}(\mathcal{S})]\!\}] \leq p(C)$, then Q_0 satisfies sp with public variables V ($x \notin V$) up to probability p' such that $p'(C) = p(C[C_{sp}[\,]])$ and $\mathsf{Bound}_{Q_0}(V \cup \{x\}, sp, D_{\mathrm{false}}, p)$.*

The proof of Proposition 1 relies on the following definitions and lemma. We have

$$\{\![\mathrm{noleak}(x[\widetilde{M'}], \mathcal{I}, \mathcal{F})]\!\} = (\forall \mathcal{I}, \neg \bigwedge \mathcal{F}) \vee ((x \notin V) \wedge \bigwedge_{{}^{\mu}x[\widetilde{M}] \text{ in } Q_0}$$

– if ${}^{\mu}x[\widetilde{M}]$ is in M in an assignment $\mathsf{let}\ y[\widetilde{i}] = M$, M is built from replication indices, variables, function applications, and conditionals, and the current call is not inside a call to $\mathrm{noleak}(y, \_, \_)$, then

$\{\![\mathrm{noleak}(y[\theta\widetilde{i}], \mathcal{I} \cup \{\theta\widetilde{i}\}, \mathcal{F} \cup \theta\mathcal{F}_\mu \cup \{\theta\widetilde{M} = \widetilde{M'}\})]\!\}$
where θ is a renaming of $\widetilde{i}$ to fresh replication indices

– if ${}^{\mu}x[\widetilde{M}]$ is in $\mathsf{event}\ e(M_1, \dots, M_{k-1}, C[{}^{\mu}x[\widetilde{M}]], M_{k+1}, \dots, M_m)$ for C defined in Figure 18, then true

– otherwise, $\forall(\mathcal{I} \cup \theta I_\mu), \neg \bigwedge(\mathcal{F} \cup \theta\mathcal{F}_\mu \cup \{\theta\widetilde{M} = \widetilde{M'}\})$
where θ is a renaming of I_μ to fresh replication indices)

$\{\![\mathrm{noleak}(x[\widetilde{M'}], \mathcal{I}, \mathcal{F})]\!\}$ is the logical formula that is guaranteed when $\mathrm{noleak}(x[\widetilde{M'}], \mathcal{I}, \mathcal{F})$ succeeds, up to a small probability computed by the equational prover and that bounds $\Pr[C[Q_0] \preceq \neg\{\![\mathrm{noleak}(x[\widetilde{M'}], \mathcal{I}, \mathcal{F})]\!\}]$. It is obtained by collecting formulas guaranteed by each call to "$\mathcal{F}$ yields a contradiction": such a call guarantees $\forall \widetilde{z} \in \widetilde{T''}, \neg \bigwedge \mathcal{F}$ up to a small probability that it evaluates, where $\widetilde{z}$ are the non-process variables in $\mathcal{F}$ and $\widetilde{T''}$ are their types. In the definition of $\{\![\mathrm{noleak}(x[\widetilde{M'}], \mathcal{I}, \mathcal{F})]\!\}$, the notation $\forall \mathcal{I}$ means that all variables in $\mathcal{I}$ are universally quantified in their respective types. The notation $\forall(\mathcal{I} \cup \theta I_\mu)$ is similar. We have similarly

$$\{\![\mathrm{prove}^{\mathsf{1\text{-}ses.secr.}(x)}(\mu)]\!\} = \begin{cases} \{\![\mathrm{noleak}(\theta\mathrm{defRand}_\mu(x), \{\theta I_\mu\}, \theta\mathcal{F}_\mu)]\!\} \\ \qquad \text{if } \mathrm{defRand}_\mu(x) \text{ is defined,} \\ \qquad \text{where } \theta \text{ is a renaming of } I_\mu \text{ to fresh replication indices} \\ \text{false} \quad \text{otherwise} \end{cases}$$

$$\{\![\mathrm{prove}^{\mathsf{1\text{-}ses.secr.}(x)}(\mathcal{S})]\!\} = \bigwedge_{\mu \in \mathcal{S}} \{\![\mathrm{prove}^{\mathsf{1\text{-}ses.secr.}(x)}(\mu)]\!\}$$

$$\{\![\mathrm{prove}^{\mathsf{distinct}(x)}(\mu_1, \mu_2)]\!\} = (z_1 \neq z_2) \vee (\forall \widetilde{i_1}, \forall \widetilde{i_2}, \neg \bigwedge \theta_1\mathcal{F}_{\mu_1} \cup \theta_2\mathcal{F}_{\mu_2} \cup \{\theta_1\widetilde{M_1} = \theta_2\widetilde{M_2}, \widetilde{i_1} \neq \widetilde{i_2}\})$$

$$\{\![\mathrm{prove}^{\mathsf{Secrecy}(x)}(\mathcal{S})]\!\} = \{\![\mathrm{prove}^{\mathsf{1\text{-}ses.secr.}(x)}(\mathcal{S})]\!\} \wedge \bigwedge_{\mu_1, \mu_2 \in \mathcal{S}} \{\![\mathrm{prove}^{\mathsf{distinct}(x)}(\mu_1, \mu_2)]\!\}$$

$$\{\![\mathrm{prove}^{\mathsf{bit\ secr.}(x)}(\mathcal{S})]\!\} = \{\![\mathrm{prove}^{\mathsf{1\text{-}ses.secr.}(x)}(\mathcal{S})]\!\}$$

The only semantic rules that can add $x[\widetilde{a}]$ to the environment E are (NewT), (LetT), (FindT1), (New), (Let), (Find1), and (Output). (get is excluded by Property 4.) By Corollary 1, the target term or process of these rules is a subterm or subprocess of Q_0 up to renaming of channels. Hence, the target configuration *Conf* of these rules is at some program point μ in

Tr. In this case, we say that $x[\widetilde{a}]$ *is defined just before* μ in a trace Tr. Furthermore, given $x[\widetilde{a}]$ and Tr, there is at most one program point μ such that $x[\widetilde{a}]$ is defined just before μ in Tr, by Lemma 28.

Let sp be $\mathsf{1\text{-}ses.secr.}(x)$, $\mathsf{Secrecy}(x)$, or $\mathsf{bit\ secr.}(x)$. Let Tr be a trace of $C[C_{sp}[Q_0]]$. Let $E = E_{Tr}$. We define the set $\mathrm{Tidx}(Tr)$ of indices of successful test queries as follows:

- When sp is $\mathsf{1\text{-}ses.secr.}(x)$: Let μ_t be the program point of the input that performs the test query in $Q_{\mathsf{1\text{-}ses.secr.}(x)}$: ${}^{\mu_t}c_s(u_1 : [1,n_1], \dots, u_m : [1,n_m])$. If there is an (Output) reduction in Tr with $\mu' = \mu_t$, we define E' to be the environment after that reduction. If $x[E(u_1),\dots,E(u_m)] \in \mathrm{Dom}(E')$, we let $\mathrm{Tidx}(Tr) = \{\epsilon\}$ (the empty sequence of indices). Otherwise, $\mathrm{Tidx}(Tr) = \emptyset$.

- When sp is $\mathsf{Secrecy}(x)$: Let μ_t be the program point of the input that performs the test query in $Q_{\mathsf{Secrecy}(x)}$: ${}^{\mu_t}c_s(u_1 : [1,n_1], \dots, u_m : [1,n_m])$. Let $\mathrm{Tidx}(Tr) = \{a \in [1,n_s] \mid$ there is an (Output) reduction in Tr with $\mu' = \mu_t$, $\sigma'(\widetilde{i}) = a$, and $x[E(u_1[a]),\dots,E(u_m[a])] \in \mathrm{Dom}(E')$ where E' is the environment after that reduction, and for all (Output) reductions in Tr before the latter reduction, with $\mu' = \mu_t$, $\sigma'(\widetilde{i}) = a'$, $E(u_1[a']) = E(u_1[a])$, …, and $E(u_m[a']) = E(u_m[a])$, we have $x[E(u_1[a]),\dots,E(u_m[a])] \notin \mathrm{Dom}(E'')$ where E'' is the environment after that reduction$\}$.

  The test query with index a is the first successful test query for $x[E(u_1[a]),\dots,E(u_m[a])]$, so $x[E(u_1[a]),\dots,E(u_m[a])]$ is defined at that test query, that is, $x[E(u_1[a]),\dots,E(u_m[a])] \in \mathrm{Dom}(E')$. For all previous test queries on the same indices, $x[E(u_1[a]),\dots,E(u_m[a])]$ was not defined, that is, $x[E(u_1[a]),\dots,E(u_m[a])] \notin \mathrm{Dom}(E'')$. The bound n_s and the variables u_1, …, u_m come from $Q_{\mathsf{Secrecy}(x)}$.

- When sp is $\mathsf{bit\ secr.}(x)$: Let μ_t be the program point of the input in $Q_{\mathsf{bit\ secr.}(x)}$: ${}^{\mu_t}c''_s(b' : bool)$. If there is an (Output) reduction in Tr with $\mu' = \mu_t$, we define E' to be the environment after that reduction. If $x \in \mathrm{Dom}(E')$, we let $\mathrm{Tidx}(Tr) = \{\epsilon\}$. Otherwise, $\mathrm{Tidx}(Tr) = \emptyset$.

Let $\mathrm{Tpp}(Tr) = \{\mu \mid \exists a \in \mathrm{Tidx}(Tr), x[E(u_1[a]),\dots,E(u_m[a])]$ is defined just before μ in $Tr\}$. When sp is $\mathsf{bit\ secr.}(x)$, $m = 0$, so this definition reduces to $\mathrm{Tpp}(Tr) = \emptyset$ if $\mathrm{Tidx}(Tr) = \emptyset$ and $\mathrm{Tpp}(Tr) = \{\mu_\epsilon\}$ where x is defined just before μ_ϵ in Tr otherwise. We write $Tr \vdash sp$ when $Tr \vdash \{\![\mathrm{prove}^{sp}(\mathrm{Tpp}(Tr))]\!\}$.

Lemma 35 *Consider a process Q_0 that satisfies Properties 4 and 5. Let sp be $\mathsf{1\text{-}ses.secr.}(x)$, $\mathsf{Secrecy}(x)$, or $\mathsf{bit\ secr.}(x)$. Let C be an evaluation context acceptable for $C_{sp}[Q_0]$ with any public variables V ($x \notin V$) that does not contain S nor $\overline{\mathsf{S}}$. We have*

$$\Pr[C[C_{sp}[Q_0]] : \mathsf{S} \wedge sp] = \Pr[C[C_{sp}[Q_0]] : \overline{\mathsf{S}} \wedge sp]\,.$$

Proof of the cases $sp = \mathsf{1\text{-}ses.secr.}(x)$ and $sp = \mathsf{Secrecy}(x)$ Let Tr be a full trace of $C[C_{sp}[Q_0]]$ such that $Tr \vdash sp$ and b is defined in Tr, that is, $b \in \mathrm{Dom}(E_{Tr})$.

Let $E = E_{Tr}$. Let $\widetilde{i}$ be the current replication indices at the definition of x in Q_0. For $j \in \mathrm{Tidx}(Tr)$, let $\widetilde{a}_j = E(u_1[j]),\dots,E(u_m[j])$, so that the test query at index j tests $x[\widetilde{a}_j]$. Let $Conf_j$ be the target configuration of the semantic rule that adds $x[\widetilde{a}_j]$ to E, and μ_j be such that $x[\widetilde{a}_j]$ is defined just before μ_j in Tr. So $Conf_j$ is at program point μ_j in Tr. Let $z_j[\widetilde{M_j}] = \mathrm{defRand}_{\mu_j}(x)$ (which is always defined since $\mu_j \in \mathrm{Tpp}(Tr)$ and $Tr \vdash sp$, so $Tr \vdash \{\![\mathrm{prove}^{\mathsf{1\text{-}ses.secr.}(x)}(\mu_j)]\!\}$). Let $\widetilde{b}_j$ be such that $E, \{\widetilde{i} \mapsto \widetilde{a}_j\}, \widetilde{M}_j \Downarrow \widetilde{b}_j$. Then $x[\widetilde{a}]$ is added to the environment E by (NewT), (LetT), (New), or (Let), we have $E(x[\widetilde{a}_j]) = E(z_j[\widetilde{b}_j])$, and $z_j[\widetilde{b}_j]$ is chosen at random by (NewT) or (New) in Tr by definition of $\text{defRand}_{\mu_j}(x)$. Let us prove that, for all $j_1 \neq j_2$ in $\text{Tidx}(Tr)$, we have $z_{j_1} \neq z_{j_2}$ or $\widetilde{b}_{j_1} \neq \widetilde{b}_{j_2}$.

- When sp is $\mathsf{1\text{-}ses.secr.}(x)$, this is trivially true since $\text{Tidx}(Tr)$ contains at most one element.
- When sp is $\mathsf{Secrecy}(x)$, we have $\mu_{j_1}, \mu_{j_2} \in \text{Tpp}(Tr)$ and $Tr \vdash \mathsf{Secrecy}(x)$, so we have $Tr \vdash \{\![\text{prove}^{\mathsf{distinct}(x)}(\mu_{j_1}, \mu_{j_2})]\!\}$, so $z_{j_1} \neq z_{j_2}$ or $Tr \vdash \forall \widetilde{i}_1, \forall \widetilde{i}_2, \neg \bigwedge \theta_1 \mathcal{F}_{\mu_{j_1}} \cup \theta_2 \mathcal{F}_{\mu_{j_2}} \cup \{\theta_1 \widetilde{M}_{j_1} = \theta_2 \widetilde{M}_{j_2}, \widetilde{i}_1 \neq \widetilde{i}_2\}$ where θ_1 and θ_2 are two distinct renamings of $\widetilde{i}$ to fresh replication indices, $\widetilde{i}_1 = \theta_1 \widetilde{i}$, and $\widetilde{i}_2 = \theta_2 \widetilde{i}$. In the latter case, let $\rho = \{\widetilde{i}_1 \mapsto \widetilde{a}_{j_1}, \widetilde{i}_2 \mapsto \widetilde{a}_{j_2}\}$. We have $\sigma_{Conf_{j_1}} = [\widetilde{i} \mapsto \widetilde{a}_{j_1}]$. By Corollary 3, $Tr, \rho \vdash \theta_1 \mathcal{F}_{\mu_{j_1}}$. Similarly, $Tr, \rho \vdash \theta_2 \mathcal{F}_{\mu_{j_2}}$. Since $j_1 \neq j_2$, $\widetilde{a}_{j_1} \neq \widetilde{a}_{j_2}$. (By construction of $\text{Tidx}(Tr)$, we consider only the first successful test query for a certain $x[\widetilde{a}_j]$.) So $Tr, \rho \vdash \widetilde{i}_1 \neq \widetilde{i}_2$. Therefore, $Tr, \rho \vdash \theta_1 \widetilde{M}_{j_1} \neq \theta_2 \widetilde{M}_{j_2}$, so $\widetilde{b}_{j_1} \neq \widetilde{b}_{j_2}$.

For $j \in \text{Tidx}(Tr)$, let us choose elements v_j in T, where T is the type of x. Let us consider the following two sets of traces:

1. Tr modified by choosing $b = \text{true}$ and $z_j[\widetilde{b}_j] = v_j$ for all $j \in \text{Tidx}(Tr)$. (The variable b is chosen and used in Q_{sp}. Note that the variable y of Q_{sp} is not defined when $b = \text{true}$. This set contains a single trace.)
2. Tr modified by choosing $b = \text{false}$ and $y[j] = v_j$ and $z_j[\widetilde{b}_j] = v'_j$ for any $v'_j \in T$, for all $j \in \text{Tidx}(Tr)$. (This set contains $|T|^{|\text{Tidx}(Tr)|}$ traces.)

The trace Tr is one of the traces in these two sets: just choose the values of b, $z_j[\widetilde{b}_j]$, and $y[j]$ if b is false that are used in Tr.

We show by induction on the derivation of these traces Tr_s (Tr_1 is in set 1 and Tr_2 is in set 2) that they have matching configurations $Conf'_s = E_s, \sigma, M_s, \mathcal{T}, \mu\mathcal{E}v_s$, $Conf'_s = E_s, \mathcal{Q}_s, \mathcal{C}h$, or $Conf'_s = E_s, (\sigma, P_s), \mathcal{Q}_s, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v_s$ for $s \in \{1, 2\}$ that differ as follows:

- $E_1(b) = \text{true}$ while $E_2(b) = \text{false}$.
- $E_1(u'_s[j])$ when sp is $\mathsf{Secrecy}(x)$ and $E_1(y[j])$ are undefined even when $E_2(u'_s[j])$ and $E_2(y[j])$ are defined for some indices $j \in \text{Tidx}(Tr)$.
- $E_1(z[\widetilde{a}])$ differs from $E_2(z[\widetilde{a}])$ for some z and $\widetilde{a}$ such that for some $\widetilde{M}$, $\mathcal{I}$, $\mathcal{F}$, we have $Tr \vdash \{\![\text{noleak}(z[\widetilde{M}], \mathcal{I}, \mathcal{F})]\!\}$ and $Tr \vdash \exists \mathcal{I}, (\widetilde{M} = \widetilde{a}) \wedge \bigwedge \mathcal{F}$.
- Values inside M_1 and M_2 may differ when $Conf'_s$ (for $s \in \{1, 2\}$) occurs in the derivation of
$$E_s, \sigma, \mathsf{let}\ y[\widetilde{i}] = M\ \mathsf{in}\ M', \mathcal{T}, \mu\mathcal{E}v_s \xrightarrow{1}{}^{*} E_s[y[\sigma\widetilde{i}] \mapsto a_s], \sigma, M', \mathcal{T}, \mu\mathcal{E}v_s$$
$$\text{or of } E_s, (\sigma, \mathsf{let}\ y[\widetilde{i}] = M\ \mathsf{in}\ P), \mathcal{Q}_s, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v_s \xrightarrow{1}{}^{*} E_s[y[\sigma\widetilde{i}] \mapsto a_s], (\sigma, P), \mathcal{Q}_s, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v_s$$
where M is built from replication indices, variables, function applications, and conditionals and for some $\widetilde{M}$, $\mathcal{I}$, $\mathcal{F}$, we have $Tr \vdash \{\![\text{noleak}(y[\widetilde{M}], \mathcal{I}, \mathcal{F})]\!\}$ and $Tr \vdash \exists \mathcal{I}, (\widetilde{M} = \sigma\widetilde{i}) \wedge \bigwedge \mathcal{F}$.
- Terms M_1 and M_2 may differ when $Conf'_s$ (for $s \in \{1, 2\}$) occurs in the derivation of
$$E_s, \sigma, \mathsf{event}\ e(\widetilde{M}_s); M', \mathcal{T}, \mu\mathcal{E}v_s \xrightarrow{1} E_s, \sigma, \mathsf{event}\ e(\widetilde{M}'_s); M', \mathcal{T}, \mu\mathcal{E}v_s$$
$$\text{or of } E_s, (\sigma, \mathsf{event}\ e(\widetilde{M}_s); P), \mathcal{Q}_s, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v_s \xrightarrow{1} E_s, (\sigma, \mathsf{event}\ e(\widetilde{M}'_s); P), \mathcal{Q}_s, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v_s$$

where the terms M_s match and the only rules above this reduction and under Conf'_s are (CtxT) with matching simple contexts any number of times followed by (CtxT) with context $\mathsf{event}\ e(a_{s,1}, \dots, a_{s,k-1}, [\,], N_{k+1}, \dots, N_l); N$ or (Ctx) with context $\mathsf{event}\ e(a_{s,1}, \dots, a_{s,k-1}, [\,], N_{k+1}, \dots, N_l); P$ once, where

- a *simple context* is a context of the form $x[a_1, \dots, a_{k-1}, [\,], N_{k+1}, \dots, N_m]$ or $f(a_1, \dots, a_{k-1}, [\,], N_{k+1}, \dots, N_m)$ for some x, f, k, m, a_1, ..., a_{k-1}, N_{k+1}, ..., N_m,
- simple contexts C_s (for $s \in \{1, 2\}$) *match* when $C_s = x[a_{s,1}, \dots, a_{s,k-1}, [\,], N_{k+1}, \dots, N_m]$ or $C_s = f(a_{s,1}, \dots, a_{s,k-1}, [\,], N_{k+1}, \dots, N_m)$ for some x, f, k, m, $a_{s,1}$, ..., $a_{s,k-1}$, N_{k+1}, ..., N_m, and
- terms M_s (for $s \in \{1, 2\}$) *match* when $M_s = a_{s,0}$, or $M_s = x[a_{s,1}, \dots, a_{s,k-1}, M'_s, N_{k+1}, \dots, N_m]$ or $M_s = f(a_{s,1}, \dots, a_{s,k-1}, M'_s, N_{k+1}, \dots, N_m)$ for some x, f, k, m, $a_{s,0}$, $a_{s,1}$, ..., $a_{s,k-1}$, N_{k+1}, ..., N_m and matching M'_s.

- Terms M_1 and M_2 (resp. processes P_1 and P_2) may differ when

$$\begin{aligned} M_s &= C_1[\dots C_k[\mathsf{event}\ e(\widetilde{M_s}); M'] \dots]\,, \\ P_s &= C_0[C_1[\dots C_k[\mathsf{event}\ e(\widetilde{M_s}); M'] \dots]]\,, \\ \text{or } P_s &= \mathsf{event}\ e(\widetilde{M_s}); P \end{aligned}$$

  where $k \in \mathbb{N}$, C_1, ..., C_k are term contexts defined in Figure 6, C_0 is a process context defined in Figure 10, and the terms $\widetilde{M_s}$ match, for $s \in \{1, 2\}$.

- Arguments of events in $\mu\mathcal{E}v_1$ and $\mu\mathcal{E}v_2$ may differ.
- The events S and $\overline{\mathsf{S}}$ are swapped: when $\mu\mathcal{E}v_1$ contains S, $\mu\mathcal{E}v_2$ contains $\overline{\mathsf{S}}$, and conversely.
- Some additional configurations corresponding to the execution Q_{sp} differ.

The proof can be sketched as follows. The different choice of b leads to $E_1(b) = \text{true}$ and $E_2(b) = \text{false}$. The only semantic rule that reads the environment is (Var), when it evaluates an occurrence of the variable in question. By Definition 7, $b \notin \mathrm{var}(Q_0) \cup V$, so the only occurrences of b are in Q_{sp}.

If the adversary sends $\widetilde{a}$ on channel c_s and $x[\widetilde{a}]$ is not defined, then Q_{sp} simply yields. If the adversary sends $\widetilde{a}$ on channel c_s and $x[\widetilde{a}]$ is defined, then $\widetilde{a} = \widetilde{a}_j$ for some $j \in \mathrm{Tidx}(\mathit{Tr})$. In set 1, $E_1(b) = \text{true}$, so Q_{sp} outputs $E_1(x[\widetilde{a}_j]) = E_1(z_j[\widetilde{b}_j]) = v_j$. In set 2, $E_2(b) = \text{false}$, so when it is the first time that the adversary sends $\widetilde{a}$ on channel c_s and $x[\widetilde{a}]$ is defined, Q_{sp} chooses a fresh $y[j]$ equal to v_j, and outputs $E_2(y[j]) = v_j$; when the adversary sends again $\widetilde{a}$ on channel c_s, Q_{sp} finds $u'_s = j$ (by construction of $\mathrm{Tidx}(\mathit{Tr})$), and outputs $E_2(y[j]) = v_j$. So in both sets, Q_{sp} outputs the same value.

If the adversary sends b' to c'_s, then the result of the test $b' = b$ differs between set 1 and set 2, since $E_1(b) \neq E_2(b)$, so if set 1 executes S, then set 2 executes $\overline{\mathsf{S}}$ and conversely. That is why the events S and $\overline{\mathsf{S}}$ are swapped.

By Definition 7, $u'_s, y \notin \mathrm{var}(Q_0) \cup V$, so the only occurrences of u'_s and y are in Q_{sp}. Therefore, the changes that come from differences in the definition of u'_s and y are already taken into account above.

The value of $E_1(z_j[\widetilde{b_j}])$ is different from the one of $E_2(z_j[\widetilde{b_j}])$. Since $\mathit{Tr} \vdash sp$, we have $\mathit{Tr} \vdash \{\![\mathrm{prove}^{\mathsf{1\text{-}ses.secr.}(x)}(\mu_j)]\!\}$, so $\mathit{Tr} \vdash \{\![\mathrm{noleak}(\theta z_j[\widetilde{M_j}], \{\theta I_{\mu_j}\}, \theta\mathcal{F}_{\mu_j})]\!\}$ where θ is a renaming of I_{μ_j} to fresh replication indices. We have $\sigma_{\mathit{Conf}_j} = [I_{\mu_j} \mapsto \widetilde{a}_j]$. Let $\rho = \{\theta I_{\mu_j} \mapsto \widetilde{a}_j\}$. By Corollary 3, $\mathit{Tr}, \rho \vdash \theta\mathcal{F}_{\mu_j}$. Moreover, $\mathit{Tr}, \rho \vdash \theta\widetilde{M_j} = \widetilde{b}_j$. So $\mathit{Tr} \vdash \exists \theta I_{\mu_j}, (\theta\widetilde{M_j} = \widetilde{b}_j) \wedge \bigwedge \theta\mathcal{F}_{\mu_j}$.

Hence, for $\widetilde{M} = \theta\widetilde{M_j}$, $\mathcal{I} = \{\theta I_{\mu_j}\}$, and $\mathcal{F} = \theta\mathcal{F}_{\mu_j}$, we have $Tr \vdash \{\![\mathrm{noleak}(z_j[\widetilde{M}], \mathcal{I}, \mathcal{F})]\!\}$ and $Tr \vdash \exists\mathcal{I}, (\widetilde{M} = \widetilde{b_j}) \wedge \bigwedge\mathcal{F}$.

The difference between $E_1(z[\widetilde{a}])$ and $E_2(z[\widetilde{a}])$ for z and $\widetilde{a}$ such that for some $\widetilde{M}$, $\mathcal{I}$, $\mathcal{F}$, we have $Tr \vdash \{\![\mathrm{noleak}(z[\widetilde{M}], \mathcal{I}, \mathcal{F})]\!\}$ and $Tr \vdash \exists\mathcal{I}, (\widetilde{M} = \widetilde{a}) \wedge \bigwedge\mathcal{F}$ has consequences when (Var) evaluates $z[\widetilde{a}]$:

$$E_s, \sigma, {}^{\mu}z[\widetilde{a}], \mathcal{T}, \mu\mathcal{E}v_s \xrightarrow{1} E_s, \sigma, E_s(z[\widetilde{a}]), \mathcal{T}, \mu\mathcal{E}v_s\,.$$

By Lemma 8, Property 5 applied to the configuration $Conf = E_s, \sigma, {}^{\mu}z[\widetilde{a}], \mathcal{T}, \mu\mathcal{E}v_s$ with $l = 0$, we have

$$E'_s, \sigma, {}^{\mu}z[\widetilde{M'}], \mathcal{T}', \mu\mathcal{E}v'_s \xrightarrow{1}{}^{*} E_s, \sigma, {}^{\mu}z[\widetilde{a}], \mathcal{T}, \mu\mathcal{E}v_s$$

by any number of applications of (CtxT), where ${}^{\mu}z[\widetilde{M'}]$ is a subterm of $C[C_{sp}[Q_0]]$. Furthermore, if the evaluation of $\widetilde{M'}$ itself uses (Var) that evaluates a $z[\widetilde{a}]$ that differs, we replace $z[\widetilde{M'}]$ by the smallest subterm of $\widetilde{M'}$ that evaluates a $z[\widetilde{a}]$ that differs. By this replacement, we guarantee that the evaluation of $\widetilde{M'}$ proceeds in the same way in set 1 and set 2 and yields the same $\widetilde{a}$. Recall that $\widetilde{M'}$ are simple terms by Invariants 2 and 5, so the evaluation of $\widetilde{M'}$ does not change E_s, $\mathcal{T}$, $\mu\mathcal{E}v_s$. So we have

$$E_s, \sigma, {}^{\mu}z[\widetilde{M'}], \mathcal{T}, \mu\mathcal{E}v_s \xrightarrow{1}{}^{*} E_s, \sigma, {}^{\mu}z[\widetilde{a}], \mathcal{T}, \mu\mathcal{E}v_s \xrightarrow{1} E_s, \sigma, E_s(z[\widetilde{a}]), \mathcal{T}, \mu\mathcal{E}v_s$$

for $s \in \{1, 2\}$, where ${}^{\mu}z[\widetilde{M'}]$ is a subterm of $C[C_{sp}[Q_0]]$, by any number of applications of (CtxT) followed by one application of (Var). Let θ be a renaming of I_μ to fresh replication indices and $\rho = \{\theta I_\mu \mapsto \sigma I_\mu\}$. By Corollary 3, $Tr_s, \rho \vdash \theta\mathcal{F}_\mu$. Moreover, $E_s, \sigma, \widetilde{M'} \Downarrow \widetilde{a}$, so $E_s, \rho, \theta\widetilde{M'} \Downarrow \widetilde{a}$. Since E_{Tr_s} extends E_s, we have $Tr_s, \rho \vdash \theta\widetilde{M'} = \widetilde{a}$. Since Tr is among the traces Tr_s, we have $Tr, \rho \vdash \theta\mathcal{F}_\mu$ and $Tr, \rho \vdash \theta\widetilde{M'} = \widetilde{a}$. There exists ρ' with domain $\mathcal{I}$ such that $Tr, \rho' \vdash (\widetilde{M} = \widetilde{a}) \wedge \bigwedge\mathcal{F}$. So $Tr, \rho \cup \rho' \vdash \bigwedge(\mathcal{F} \cup \theta\mathcal{F}_\mu \cup \{\theta\widetilde{M'} = \widetilde{a}, \widetilde{M} = \widetilde{a}\})$. Since $Tr \vdash \{\![\mathrm{noleak}(z[\widetilde{M}], \mathcal{I}, \mathcal{F})]\!\}$ and $Tr \vdash \exists\mathcal{I}, (\widetilde{M} = \widetilde{a}) \wedge \bigwedge\mathcal{F}$, we have $Tr \vdash \neg(\forall\mathcal{I}, \neg\bigwedge\mathcal{F})$, so Tr satisfies the second disjunct of $\{\![\mathrm{noleak}(z[\widetilde{M}], \mathcal{I}, \mathcal{F})]\!\}$. Therefore, $z \notin V$, so the occurrence of ${}^{\mu}z[\widetilde{M'}]$ evaluated above is either in Q_{sp}, and in this case z is actually x and this case has already been studied above, or in Q_0. In the latter situation, we are in one of the following three cases:

- ${}^{\mu}z[\widetilde{M'}]$ is in M in an assignment ${}^{\mu'}\mathsf{let}\ y[\widetilde{i}] = M$ in Q_0, M is built from replication indices, variables, function applications, and conditionals and $Tr \vdash \{\![\mathrm{noleak}(y[\theta\widetilde{i}], \mathcal{I} \cup \{\theta\widetilde{i}\}, \mathcal{F} \cup \theta\mathcal{F}_\mu \cup \{\theta\widetilde{M'} = \widetilde{M}\})]\!\}$ where θ is a renaming of $\widetilde{i}$ to fresh replication indices. Let $\widetilde{M''} = \theta\widetilde{i}$, $\mathcal{I}' = \mathcal{I} \cup \{\theta\widetilde{i}\}$, and $\mathcal{F}' = \mathcal{F} \cup \theta\mathcal{F}_\mu \cup \{\theta\widetilde{M'} = \widetilde{M}\}$. Since μ' is above μ, by Lemma 6, there is a configuration inside μ' before $Conf$ in Tr. By Lemma 8 applied to that configuration (Property 1 when the assignment is a process, Property 5 with $l = 0$ when it is a term), the assignment ${}^{\mu'}\mathsf{let}\ y[\widetilde{i}] = M$ is evaluated by

$$E_s, \sigma, {}^{\mu'}\mathsf{let}\ y[\widetilde{i}] = M\ \mathsf{in}\ M', \mathcal{T}, \mu\mathcal{E}v_s \xrightarrow{1}{}^{*} E_s[y[\sigma\widetilde{i}] \mapsto a_s], \sigma, M', \mathcal{T}, \mu\mathcal{E}v_s$$

$$\text{or } E_s, (\sigma, {}^{\mu'}\mathsf{let}\ y[\widetilde{i}] = M\ \mathsf{in}\ P), \mathcal{Q}_s, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v_s \xrightarrow{1}{}^{*} E_s[y[\sigma\widetilde{i}] \mapsto a_s], (\sigma, P), \mathcal{Q}_s, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v_s$$

  for $s \in \{1, 2\}$. We have $Tr \vdash \{\![\mathrm{noleak}(y[\widetilde{M''}], \mathcal{I}', \mathcal{F}')]\!\}$. Moreover $\widetilde{i} = I_\mu$ and $Tr, \rho \cup \rho' \vdash \theta\widetilde{i} = \sigma\widetilde{i}$ by definition of ρ. Hence $Tr, \rho \cup \rho' \vdash \widetilde{M''} = \sigma\widetilde{i}$ and $Tr, \rho \cup \rho' \vdash \mathcal{F}'$, so $Tr \vdash \exists\mathcal{I}', (\widetilde{M''} = \sigma\widetilde{i}) \wedge \bigwedge\mathcal{F}'$. Hence we are in a case in which different values inside terms M_1 and M_2 are allowed. Furthermore, the added values $E_s(y[\sigma\widetilde{i}]) = a_s$ may differ. Let $\widetilde{a'} = \sigma\widetilde{i}$. We have $Tr \vdash \{\![\mathrm{noleak}(y[\widetilde{M''}], \mathcal{I}', \mathcal{F}')]\!\}$ and $Tr \vdash \exists\mathcal{I}', (\widetilde{M''} = \widetilde{a'}) \wedge \bigwedge\mathcal{F}'$, so $E_1(y[\widetilde{a'}])$ is indeed allowed to differ from $E_2(y[\widetilde{a'}])$.

- ${}^{\mu}z[\widetilde{M'}]$ is in ${}^{\mu'}\mathsf{event}\ e(M_1, \dots, M_{k-1}, C[{}^{\mu}z[\widetilde{M'}]], M_{k+1}, \dots, M_m)$ in Q_0, for C defined in Figure 18. Since μ' is above μ, by Lemma 6, there is a configuration inside μ' before *Conf* in *Tr*. By Lemma 8 applied to that configuration (Property 1 when the event is a process, Property 5 with $l = 0$ when it is a term), the evaluation of the event starts from a configuration at μ' in *Tr*. The evaluation of $\mathsf{event}\ e(M_1, \dots, M_{k-1}, C[{}^{\mu}z[\widetilde{M'}]], M_{k+1}, \dots, M_m)$ first evaluates M_1, …, M_{k-1} to values using (CtxT) or (Ctx) with an event context. (If they evaluated to abort event values, $C[{}^{\mu}z[\widetilde{M'}]]$ would not be evaluated.) Then if evaluates the context C: $\mathsf{new}\ y[\widetilde{i}] : T; C$ is evaluated by (NewT), $\mathsf{let}\ y[\widetilde{i}] = M\ \mathsf{in}\ C$ is evaluated by (LetT), $\mathsf{if}\ M\ \mathsf{then}\ C\ \mathsf{else}\ N'$ is evaluated by (IfT1) (M must evaluate to true because otherwise, ${}^{\mu}z[\widetilde{M'}]$ would not be evaluated), $\mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ C$ is evaluated by (IfT2) (M must not evaluate to true because otherwise, ${}^{\mu}z[\widetilde{M'}]$ would not be evaluated), $\mathsf{event}\ e(\widetilde{M}); C$ is evaluated by (EventT), and find contexts are evaluated by rules for find, until we reach

$$\mathsf{event}\ e(a_{s,1}, \dots, a_{s,k-1}, C_{s,1}[\dots C_{s,l}[{}^{\mu}z[\widetilde{M'}]] \dots], M_{k+1}, \dots, M_m)$$

for $s \in \{1, 2\}$, where $C_{s,1}$, …, $C_{s,l}$ are matching simple contexts. (Values may differ in case M_1, …, M_{k-1}, or terms in C contain other occurrences of variables whose value differs.) At this point, the reduction proceeds as follows:

$$E_s, \sigma, \mathsf{event}\ e(\widetilde{M_s}); M', \mathcal{T}, \mu\mathcal{E}v_s \xrightarrow{1} E_s, \sigma, \mathsf{event}\ e(\widetilde{M'_s}); M', \mathcal{T}, \mu\mathcal{E}v_s$$
$$\text{or } E_s, (\sigma, \mathsf{event}\ e(\widetilde{M_s}); P), \mathcal{Q}_s, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v_s \xrightarrow{1} E_s, (\sigma, \mathsf{event}\ e(\widetilde{M'_s}); P), \mathcal{Q}_s, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v_s$$

by (Var), (CtxT) with matching simple contexts any number of times followed by (CtxT) or (Ctx) with context $\mathsf{event}\ e(a_{s,1}, \dots, a_{s,k-1}, [\,], M_{k+1}, \dots, M_m); \dots$ once, where

$$\widetilde{M_s} = a_{s,1}, \dots, a_{s,k-1}, C_{s,1}[\dots C_{s,l}[{}^{\mu}z[\widetilde{M'}]] \dots], M_{k+1}, \dots, M_m$$
$$\text{and } \widetilde{M'_s} = a_{s,1}, \dots, a_{s,k-1}, C_{s,1}[\dots C_{s,l}[a_s] \dots], M_{k+1}, \dots, M_m$$

for $s \in \{1, 2\}$. Further reductions still manipulate configurations of the same form until the event itself is executed by (EventT) or (Event), which adds the event e with possibly different arguments to $\mu\mathcal{E}v_s$.

- $Tr \vdash \forall(\mathcal{I} \cup \theta I_\mu), \neg \bigwedge(\mathcal{F} \cup \theta\mathcal{F}_\mu \cup \{\theta\widetilde{M'} = \widetilde{M}\})$ where θ is a renaming of I_μ to fresh replication indices. We have $Tr, \rho \cup \rho' \vdash \neg \bigwedge(\mathcal{F} \cup \theta\mathcal{F}_\mu \cup \{\theta\widetilde{M'} = \widetilde{M}\})$. That yields a contradiction, so this case does not happen.

The sequence of events $\mu\mathcal{E}v_1$ (resp. $\mu\mathcal{E}v_2$) is never read by the semantic rules. It is only read by the distinguisher. Therefore, changes in this sequence of events do not modify the rest of the trace.

That concludes the proof that traces in set 1 and set 2 match.

Furthermore, the trace in set 1 and the traces in set 2 have the same probability. All full traces of $C[C_{sp}[Q_0]]$ that define b and that satisfy *sp* belong to set 1 or to set 2 for some *Tr* (for instance using the trace in question as *Tr*). Therefore, these sets form a partition of the full traces of $C[C_{sp}[Q_0]]$ that define b and that satisfy *sp*, and the sets that execute S have the same probability as the sets that execute $\overline{\mathsf{S}}$. Moreover, the traces of $C[C_{sp}[Q_0]]$ that do not define b execute neither S nor $\overline{\mathsf{S}}$. So $\Pr[C[C_{sp}[Q_0]] : \mathsf{S} \wedge sp] = \Pr[C[C_{sp}[Q_0]] : \overline{\mathsf{S}} \wedge sp]$. □

Proof of the case $sp = \mathsf{bit\ secr.}(x)$ Let *Tr* be a full trace of $C[C_{sp}[Q_0]]$ such that $Tr \vdash sp$. Let $E = E_{Tr}$.

If $\mathrm{Tidx}(\mathit{Tr}) = \emptyset$, then Tr executes neither S nor $\overline{\mathsf{S}}$.

Otherwise, $\mathrm{Tidx}(\mathit{Tr}) = \{\epsilon\}$ and x is defined in Tr, that is, $x \in \mathrm{Dom}(E)$. Let Conf_ϵ be the target configuration of the semantic rule that adds x to E, and μ_ϵ be such that x is defined just before μ_ϵ in Tr. So Conf_ϵ is at program point μ_ϵ in Tr. Let $z_\epsilon[\widetilde{M_\epsilon}] = \mathrm{defRand}_{\mu_\epsilon}(x)$ (which is always defined since $\mu_\epsilon \in \mathrm{Tpp}(\mathit{Tr})$ and $\mathit{Tr} \vdash \mathit{sp}$, so $\mathit{Tr} \vdash \{\![\mathrm{prove}^{\mathsf{1\text{-}ses.secr.}(x)}(\mu_\epsilon)]\!\}$). Let $\widetilde{b_\epsilon}$ be such that $E, \emptyset, \widetilde{M_\epsilon} \Downarrow \widetilde{b_\epsilon}$. Then x is added to the environment E by (NewT), (LetT), (New), or (Let), we have $E(x) = E(z_\epsilon[\widetilde{b_\epsilon}])$, and $z_\epsilon[\widetilde{b_\epsilon}]$ is chosen at random by (NewT) or (New) in Tr by definition of $\mathrm{defRand}_{\mu_\epsilon}(x)$.

Let us consider the following two traces:

1. Tr_1 is Tr modified by choosing $z_\epsilon[\widetilde{b_\epsilon}] = \mathrm{true}$.

2. Tr_2 is Tr modified by choosing $z_\epsilon[\widetilde{b_\epsilon}] = \mathrm{false}$.

The trace Tr is one of these two traces: just choose the value $z_\epsilon[\widetilde{b_\epsilon}]$ that is used in Tr.

We show by induction on the derivation of these traces Tr_s that they have matching configurations $\mathit{Conf}'_s = E_s, \sigma, M_s, \mathcal{T}, \mu\mathcal{E}v_s$, $\mathit{Conf}'_s = E_s, \mathcal{Q}_s, \mathcal{C}h$, or $\mathit{Conf}'_s = E_s, (\sigma, P_s), \mathcal{Q}_s, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v_s$ for $s \in \{1, 2\}$ that differ as follows:

- $E_1(z[\widetilde{a}])$ differs from $E_2(z[\widetilde{a}])$ for some z and $\widetilde{a}$ such that for some $\widetilde{M}$, $\mathcal{I}$, $\mathcal{F}$, we have $\mathit{Tr} \vdash \{\![\mathrm{noleak}(z[\widetilde{M}], \mathcal{I}, \mathcal{F})]\!\}$ and $\mathit{Tr} \vdash \exists \mathcal{I}, (\widetilde{M} = \widetilde{a}) \wedge \bigwedge \mathcal{F}$.

- Values inside M_1 and M_2 may differ when Conf'_s (for $s \in \{1, 2\}$) occurs in the derivation of

$$E_s, \sigma, \mathsf{let}\ y[\widetilde{i}] = M\ \mathsf{in}\ M', \mathcal{T}, \mu\mathcal{E}v_s \xrightarrow{1}{}^{*} E_s[y[\sigma\widetilde{i}] \mapsto a_s], \sigma, M', \mathcal{T}, \mu\mathcal{E}v_s$$
$$\text{or of } E_s, (\sigma, \mathsf{let}\ y[\widetilde{i}] = M\ \mathsf{in}\ P), \mathcal{Q}_s, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v_s \xrightarrow{1}{}^{*} E_s[y[\sigma\widetilde{i}] \mapsto a_s], (\sigma, P), \mathcal{Q}_s, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v_s$$

  where M is built from replication indices, variables, function applications, and conditionals and for some $\widetilde{M}$, $\mathcal{I}$, $\mathcal{F}$, we have $\mathit{Tr} \vdash \{\![\mathrm{noleak}(y[\widetilde{M}], \mathcal{I}, \mathcal{F})]\!\}$ and $\mathit{Tr} \vdash \exists \mathcal{I}, (\widetilde{M} = \sigma\widetilde{i}) \wedge \bigwedge \mathcal{F}$.

- Terms M_1 and M_2 may differ when Conf'_s (for $s \in \{1, 2\}$) occurs in the derivation of

$$E_s, \sigma, \mathsf{event}\ e(\widetilde{M_s}); M', \mathcal{T}, \mu\mathcal{E}v_s \xrightarrow{1} E_s, \sigma, \mathsf{event}\ e(\widetilde{M'_s}); M', \mathcal{T}, \mu\mathcal{E}v_s$$
$$\text{or of } E_s, (\sigma, \mathsf{event}\ e(\widetilde{M_s}); P), \mathcal{Q}_s, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v_s \xrightarrow{1} E_s, (\sigma, \mathsf{event}\ e(\widetilde{M'_s}); P), \mathcal{Q}_s, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v_s$$

  where the terms M_s match and the only rules above this reduction and under Conf'_s are (CtxT) with matching simple contexts any number of times followed by (CtxT) with context $\mathsf{event}\ e(a_{s,1}, \dots, a_{s,k-1}, [\,], N_{k+1}, \dots, N_l); N$ or (Ctx) with context $\mathsf{event}\ e(a_{s,1}, \dots, a_{s,k-1}, [\,], N_{k+1}, \dots, N_l); P$ once, where simple contexts and matching are defined as in the cases $\mathit{sp} = \mathsf{1\text{-}ses.secr.}(x)$ and $\mathit{sp} = \mathsf{Secrecy}(x)$.

- Terms M_1 and M_2 (resp. processes P_1 and P_2) may differ when

$$\begin{aligned} M_s &= C_1[\dots C_k[\mathsf{event}\ e(\widetilde{M_s}); M'] \dots]\,, \\ P_s &= C_0[C_1[\dots C_k[\mathsf{event}\ e(\widetilde{M_s}); M'] \dots]]\,, \\ \text{or } P_s &= \mathsf{event}\ e(\widetilde{M_s}); P \end{aligned}$$

  where $k \in \mathbb{N}$, $C_1, \dots, C_k$ are term contexts defined in Figure 6, C_0 is a process context defined in Figure 10, and the terms $\widetilde{M_s}$ match, for $s \in \{1, 2\}$.

- Arguments of events in $\mu\mathcal{E}v_1$ and $\mu\mathcal{E}v_2$ may differ.
- The events S and $\overline{\mathsf{S}}$ are swapped: when $\mu\mathcal{E}v_1$ contains S, $\mu\mathcal{E}v_2$ contains $\overline{\mathsf{S}}$, and conversely.
- Some additional configurations corresponding to the execution Q_{sp} differ.

The proof can be sketched as follows.

If the adversary sends b' to c''_s, then x is defined (since $\mathrm{Tidx}(\mathit{Tr}) = \{\epsilon\}$) and the result of the test $x = b'$ differs between Tr_1 and Tr_2, since $E_1(x) = E_1(z_\epsilon[\widetilde{b_\epsilon}]) = \mathrm{true} \neq E_2(x) = E_2(z_\epsilon[\widetilde{b_\epsilon}]) = \mathrm{false}$, so if Tr_1 executes S, then Tr_2 executes $\overline{\mathsf{S}}$ and conversely. That is why the events S and $\overline{\mathsf{S}}$ are swapped.

The value of $E_1(z_\epsilon[\widetilde{b_\epsilon}])$ is different from the one of $E_2(z_\epsilon[\widetilde{b_\epsilon}])$. Since $\mathit{Tr} \vdash \mathit{sp}$, we have $\mathit{Tr} \vdash \{\![\mathrm{prove}^{\mathsf{1\text{-}ses.secr.}(x)}(\mu_\epsilon)]\!\}$, so $\mathit{Tr} \vdash \{\![\mathrm{noleak}(\theta z_\epsilon[\widetilde{M_\epsilon}], \{\theta I_{\mu_\epsilon}\}, \theta\mathcal{F}_{\mu_\epsilon})]\!\}$ where θ is a renaming of I_{μ_ϵ} to fresh replication indices. Here, I_{μ_ϵ} is empty since x is defined under no replication. We have $\sigma_{\mathit{Conf}_\epsilon} = []$. Let $\rho = \emptyset$. By Corollary 3, $\mathit{Tr}, \rho \vdash \theta\mathcal{F}_{\mu_\epsilon}$. Moreover, $\mathit{Tr}, \rho \vdash \theta\widetilde{M_\epsilon} = \widetilde{b_\epsilon}$. So $\mathit{Tr} \vdash \exists\theta I_{\mu_\epsilon}, (\theta\widetilde{M_\epsilon} = \widetilde{b_\epsilon}) \wedge \bigwedge \theta\mathcal{F}_{\mu_\epsilon}$. Hence, for $\widetilde{M} = \theta\widetilde{M_\epsilon}$, $\mathcal{I} = \{\theta I_{\mu_\epsilon}\}$, and $\mathcal{F} = \theta\mathcal{F}_{\mu_\epsilon}$, we have $\mathit{Tr} \vdash \{\![\mathrm{noleak}(z_\epsilon[\widetilde{M}], \mathcal{I}, \mathcal{F})]\!\}$ and $\mathit{Tr} \vdash \exists\mathcal{I}, (\widetilde{M} = \widetilde{b_\epsilon}) \wedge \bigwedge \mathcal{F}$.

The difference between $E_1(z[\widetilde{a}])$ and $E_2(z[\widetilde{a}])$ for z and $\widetilde{a}$ such that for some $\widetilde{M}$, $\mathcal{I}$, $\mathcal{F}$, we have $\mathit{Tr} \vdash \{\![\mathrm{noleak}(z[\widetilde{M}], \mathcal{I}, \mathcal{F})]\!\}$ and $\mathit{Tr} \vdash \exists\mathcal{I}, (\widetilde{M} = \widetilde{a}) \wedge \bigwedge \mathcal{F}$ has consequences when (Var) evaluates $z[\widetilde{a}]$, as in the cases $\mathit{sp} = \mathsf{1\text{-}ses.secr.}(x)$ and $\mathit{sp} = \mathsf{Secrecy}(x)$. That concludes the proof that traces Tr_1 and Tr_2 match.

Furthermore, the traces Tr_1 and Tr_2 have the same probability. All full traces of $C[C_{sp}[Q_0]]$ such that Tidx is non-empty and that satisfy sp are Tr_1 or Tr_2 for some Tr (for instance using the trace in question as Tr). Therefore, Tr_1 and Tr_2 form a partition of the full traces of $C[C_{sp}[Q_0]]$ such that Tidx is non-empty and that satisfy sp, and half of these traces execute S, the other half execute $\overline{\mathsf{S}}$. Moreover, the traces of $C[C_{sp}[Q_0]]$ such that Tidx is empty execute neither S nor $\overline{\mathsf{S}}$. So $\Pr[C[C_{sp}[Q_0]] : \mathsf{S} \wedge \mathit{sp}] = \Pr[C[C_{sp}[Q_0]] : \overline{\mathsf{S}} \wedge \mathit{sp}]$. □

Proof of Proposition 1 Let sp be $\mathsf{1\text{-}ses.secr.}(x)$, $\mathsf{Secrecy}(x)$, or $\mathsf{bit\ secr.}(x)$. Let C be an evaluation context acceptable for $C_{sp}[Q_0]$ with public variables V ($x \notin V$) that does not contain S nor $\overline{\mathsf{S}}$. We have

$$\begin{aligned}
\mathsf{Adv}^{sp}_{Q_0}(C) &= \Pr[C[C_{sp}[Q_0]] : \mathsf{S}] - \Pr[C[C_{sp}[Q_0]] : \overline{\mathsf{S}}] \\
&= \Pr[C[C_{sp}[Q_0]] : \mathsf{S} \wedge \mathit{sp}] + \Pr[C[C_{sp}[Q_0]] : \mathsf{S} \wedge \neg\mathit{sp}] \\
&\quad - \Pr[C[C_{sp}[Q_0]] : \overline{\mathsf{S}} \wedge \mathit{sp}] - \Pr[C[C_{sp}[Q_0]] : \overline{\mathsf{S}} \wedge \neg\mathit{sp}] \\
&= \Pr[C[C_{sp}[Q_0]] : \mathsf{S} \wedge \neg\mathit{sp}] - \Pr[C[C_{sp}[Q_0]] : \overline{\mathsf{S}} \wedge \neg\mathit{sp}] && \text{by Lemma 35} \\
&\leq \Pr[C[C_{sp}[Q_0]] : \neg\mathit{sp}] \\
&\leq \Pr[C[C_{sp}[Q_0]] : \neg\{\![\mathrm{prove}^{sp}(\{\mu \mid \mu \text{ follows a definition of } x\})]\!\}] \\
&\qquad\qquad \text{because, for all } \mathit{Tr},\ \mathrm{Tpp}(\mathit{Tr}) \subseteq \{\mu \mid \mu \text{ follows a definition of } x\} \\
&\leq \Pr[C[C_{sp}[Q_0]] \preceq \neg\{\![\mathrm{prove}^{sp}(\{\mu \mid \mu \text{ follows a definition of } x\})]\!\}] && \text{by Lemma 1} \\
&\leq p(C[C_{sp}[\,]]) = p'(C)
\end{aligned}$$

So Q_0 satisfies sp with public variables V up to probability p'. Moreover,

$$\begin{aligned}
\mathsf{Adv}_{Q_0}(C[C_{sp}[\,]], \mathit{sp}, D_{\mathrm{false}}) &= \Pr[C[C_{sp}[Q_0]] : \mathsf{S}] - \Pr[C[C_{sp}[Q_0]] : \overline{\mathsf{S}} \vee \mathsf{NonUnique}_{Q_0}] \\
&\leq \mathsf{Adv}^{sp}_{Q_0}(C) \leq p(C[C_{sp}[\,]])
\end{aligned}$$

so $\mathsf{Bound}_{Q_0}(V \cup \{x\}, \mathit{sp}, D_{\mathrm{false}}, p)$. □

Example 5 Using assumptions on cryptographic primitives, the process Q_0 of Example 1 can be transformed into the following process Q_0'':

$$\begin{aligned}
&Q_0'' = \mathit{start}(); \mathsf{new}\ x_k : T_k; \mathsf{new}\ x_{mk} : T_{mk}; \overline{c}\langle\rangle; (Q_A'' \mid Q_B'') \\
&Q_A'' = !^{i \leq n} c_A[i](); \mathsf{new}\ x_k' : T_k; \mathsf{new}\ x_r : T_r; \\
&\quad \mathsf{let}\ x_m : \mathit{bitstring} = \mathrm{enc}'(\mathrm{Z}_k, k, x_r')\ \mathsf{in} \\
&\quad \overline{c_A[i]}\langle x_m, \mathrm{mac}'(x_m, x_{mk})\rangle \\
&Q_B'' = !^{i' \leq n} c_B[i'](x_m', x_{ma}); \\
&\quad \mathsf{find}\ u \leq n\ \mathsf{suchthat}\ \mathsf{defined}(x_m[u], x_k'[u]) \wedge \\
&\qquad x_m' = x_m[u] \wedge \mathrm{verify}'(x_m', x_{mk}, x_{ma})\ \mathsf{then} \\
&\quad \mathsf{let}\ x_k'' : T_k = x_k'[u]\ \mathsf{in}\ \overline{c_B[i']}\langle\rangle
\end{aligned}$$

and $Q_0 \approx_p^{x_k''} Q_0''$. In order to prove the one-session secrecy of x_k'', we notice that x_k'' is defined by $\mathsf{let}\ x_k'' : T_k = x_k'[u]$, the only variable access to x_k' in Q_0'' is $\mathsf{let}\ x_k'' : T_k = x_k'[u]$, and x_k'' is not used in Q_0''. So by Proposition 1, Q_0'' satisfies the one-session secrecy of x_k'' without public variables up to probability 0. (We have $\mathrm{defRand}_\mu(x_k'') = x_k'[u]$ and $\{\![\mathrm{noleak}(x_k'[u], \mathcal{I}, \mathcal{F})]\!\} = (\forall \mathcal{I}, \neg \bigwedge \mathcal{F}) \vee ((x_k' \notin V) \wedge \{\![\mathrm{noleak}(x_k''[\widetilde{\theta i}], \mathcal{I}', \mathcal{F}')]\!\}) = \mathrm{true}$ since $x_k' \notin V$, $x_k'' \notin V$, and $\{\![\mathrm{noleak}(x_k''[\widetilde{\theta i}], \mathcal{I}', \mathcal{F}')]\!\} = (\forall \mathcal{I}', \neg \bigwedge \mathcal{F}') \vee ((x_k'' \notin V) \wedge \mathrm{true}) = \mathrm{true}$. So $\Pr[C[Q_0''] \preceq \neg\{\![\mathrm{prove}^{\mathsf{1\text{-}ses.secr.}(x_k'')}(\mathcal{S})]\!\}] = 0$.) By Lemma 20, the process Q_0 of Example 1 also satisfies the one-session secrecy of x_k'' without public variables up to probability $p'(C) = 2p(C[C_{\mathsf{1\text{-}ses.secr.}(x_k'')}[\,]], t_{\mathsf{S}})$. 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, $\mathrm{prove}^{\mathsf{Secrecy}(x)}(\mathcal{S})$ 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.

4.2 Correspondences

4.2.1 Example

We illustrate the proof of correspondences on the following example, inspired by the corrected Woo-Lam public key protocol [68]:

$$\begin{aligned}
&B \to A : (N, B) \\
&A \to B : \{pk_A, B, N\}_{sk_A}
\end{aligned}$$

This protocol is a simple nonce challenge: B sends to A a fresh nonce N and its identity. A replies by signing the nonce N, B's identity, and A's public key (which we use here instead of A's identity for simplicity: this avoids having to relate identities and keys; CryptoVerif can obviously also handle the version with A's identity). The signatures are assumed to be (existentially) unforgeable under chosen message attacks (UF-CMA) [45], so, when B receives the signature, B is convinced that A is present. The signature cannot be a replay because the nonce N is signed.

In our calculus, this protocol is encoded by the following process G_0, explained below:

$$\begin{aligned}
G_0 = c_0(); \mathsf{new}\ rk_A : \mathit{keyseed}; \mathsf{let}\ pk_A = \mathrm{pkgen}(rk_A)\ \mathsf{in} \\
\mathsf{let}\ sk_A = \mathrm{skgen}(rk_A)\ \mathsf{in}\ \overline{c_1}\langle pk_A\rangle; (Q_A \mid Q_B)
\end{aligned}$$

$$\begin{aligned}
Q_A = {} & !^{i_A \leq n} c_2[i_A](x_N : \mathit{nonce}, x_B : \mathit{host});\\
& \mathsf{event}\ e_A(pk_A, x_B, x_N); \mathsf{new}\ r : \mathit{seed};\\
& \overline{c_3[i_A]}\langle \mathrm{sign}(\mathrm{concat}(pk_A, x_B, x_N), sk_A, r)\rangle\\
Q_B = {} & !^{i_B \leq n} c_4[i_B](x_{pk_A} : \mathit{pkey}); \mathsf{new}\ N : \mathit{nonce};\\
& \overline{c_5[i_B]}\langle N, B\rangle; c_6[i_B](s : \mathit{signature});\\
& \mathsf{if}\ \mathrm{verify}(\mathrm{concat}(x_{pk_A}, B, N), x_{pk_A}, s)\ \mathsf{then}\\
& \mathsf{if}\ x_{pk_A} = pk_A\ \mathsf{then}\ \mathsf{event}\ e_B(x_{pk_A}, B, N)
\end{aligned}$$

The process G_0 is assumed to run in interaction with an adversary, which also models the network. G_0 first receives an empty message on channel c_0, sent by the adversary. Then, it chooses randomly with uniform probability a bitstring rk_A in the type *keyseed*, by the construct new rk_A : *keyseed*. Then, G_0 generates the public key pk_A corresponding to the coins rk_A, by calling the public-key generation algorithm pkgen. Similarly, G_0 generates the secret key sk_A by calling skgen. It outputs the public key pk_A on channel c_1, so that the adversary has this public key.

After outputting this message, the control passes to the receiving process, which is part of the adversary. Several processes are then made available, which represent the roles of A and B in the protocol: the process $Q_A \mid Q_B$ is the parallel composition of Q_A and Q_B; it makes simultaneously available the processes defined in Q_A and Q_B. Let Q'_A and Q'_B be such that $Q_A = !^{i_A \leq n} Q'_A$ and $Q_B = !^{i_B \leq n} Q'_B$. The replication $!^{i_A \leq n} Q'_A$ represents n copies of the process Q'_A, indexed by the replication index i_A. The process Q'_A begins with an input on channel $c_2[i_A]$; the channel is indexed with i_A so that the adversary can choose which copy of the process Q'_A receives the message by sending it on channel $c_2[i_A]$ for the appropriate value of i_A. The situation is similar for Q'_B, which expects a message on channel $c_4[i_B]$. The adversary can then run each copy of Q'_A or Q'_B simply by sending a message on the appropriate channel $c_2[i_A]$ or $c_4[i_B]$.

The process Q'_B first expects on channel $c_4[i_B]$ a message x_{pk_A} in the type *pkey* of public keys. This message is not really part of the protocol. It serves for starting a new session of the protocol, in which B interacts with the participant of public key x_{pk_A}. For starting a session between A and B, this message should be pk_A. Then, Q'_B chooses randomly with uniform probability a nonce N in the type *nonce*. The type *nonce* is *large*: collisions between independent random numbers chosen uniformly in a large type are eliminated by CryptoVerif. Q'_B sends the message (N, B) on channel $c_5[i_B]$. The control then passes to the receiving process, included in the adversary. This process is expected to forward this message (N, B) on channel $c_2[i_A]$, but may proceed differently in order to mount an attack against the protocol.

Upon receiving a message (x_N, x_B) on channel $c_2[i_A]$, where the bitstring x_N is in the type *nonce* and x_B in the type *host*, the process Q'_A executes the event $e_A(pk_A, x_B, x_N)$. This event does not change the state of the system. Events just record that a certain program point has been reached, with certain values of the arguments of the event. Then, Q'_A chooses randomly with uniform probability a bitstring r in the type *seed*; this random bitstring is next used as coins for the signature algorithm. Finally, Q'_A outputs the signed message $\{pk_A, x_B, x_N\}_{sk_A}$. (The function concat concatenates its arguments, with information on the length of these arguments, so that the arguments can be recovered from the concatenation.) The control then passes to the receiving process, which should forward this message on channel $c_6[i_B]$ if it wishes to run the protocol correctly.

Upon receiving a message s on $c_6[i_B]$, Q'_B verifies that the signature s is correct and, if $x_{pk_A} = pk_A$, that is, if B runs a session with A, it executes the event $e_B(x_{pk_A}, B, N)$. Our goal is to prove that, if event e_B is executed, then event e_A has also been executed. However, when B runs a session with a participant other than A, it is perfectly correct that B terminates without event e_A being executed; that is why event e_B is executed only when B runs a session with A.

By the unforgeability of signatures, the signature verification with pk_A succeeds only for signatures generated with sk_A. So, when we verify that the signature is correct, we can furthermore check that it has been generated using sk_A. So, after game transformations explained below, we obtain the following final game:

$$\begin{aligned}
G_1 = {} & c_0(); \mathsf{new}\ rk_A : \mathit{keyseed};\\
& \mathsf{let}\ pk_A = \mathrm{pkgen}'(rk_A)\ \mathsf{in}\ \overline{c_1}\langle pk_A\rangle; (Q_{1A} \mid Q_{1B})\\
Q_{1A} = {} & !^{i_A \leq n} c_2[i_A](x_N : \mathit{nonce}, x_B : \mathit{host});\\
& \mathsf{event}\ e_A(pk_A, x_B, x_N);\\
& \mathsf{let}\ m = \mathrm{concat}(pk_A, x_B, x_N)\ \mathsf{in}\\
& \mathsf{new}\ r : \mathit{seed}; \overline{c_3[i_A]}\langle \mathrm{sign}'(m, \mathrm{skgen}'(rk_A), r)\rangle\\
Q_{1B} = {} & !^{i_B \leq n} c_4[i_B](x_{pk_A} : \mathit{pkey}); \mathsf{new}\ N : \mathit{nonce};\\
& \overline{c_5[i_B]}\langle N, B\rangle; c_6[i_B](s : \mathit{signature});\\
& \mathsf{find}\ u \leq n\ \mathsf{suchthat}\ \mathsf{defined}(m[u], x_B[u], x_N[u])\\
& \quad \wedge (x_{pk_A} = pk_A) \wedge (B = x_B[u]) \wedge (N = x_N[u])\\
& \quad \wedge \mathrm{verify}'(\mathrm{concat}(x_{pk_A}, B, N), x_{pk_A}, s)\ \mathsf{then}\\
& \mathsf{event}\ e_B(x_{pk_A}, B, N))
\end{aligned}$$

The assignment $sk_A = \mathrm{skgen}(rk_A)$ has been removed and $\mathrm{skgen}(rk_A)$ has been substituted for sk_A, in order to make the term $\mathrm{sign}(m, \mathrm{skgen}(rk_A), r)$ appear. This term is needed for the security of the signature scheme to apply.

In Q_{1A}, the signed message is stored in variable m, and this variable is used when computing the signature.

Finally, using the unforgeability of signatures, the signature verification has been replaced with an array lookup: the signature verification can succeed only when $\mathrm{concat}(x_{pk_A}, B, N)$ has been signed with sk_A, so we look for the message $\mathrm{concat}(x_{pk_A}, B, N)$ in the array m and the event e_B is executed only when this message is found. In other words, we look for an index $u \leq n$ such that $m[u]$ is defined and $m[u] = \mathrm{concat}(x_{pk_A}, B, N)$. By definition of m, $m[u] = \mathrm{concat}(pk_A, x_B[u], x_N[u])$, so the equality $m[u] = \mathrm{concat}(x_{pk_A}, B, N)$ can be replaced with $(x_{pk_A} = pk_A) \wedge (B = x_B[u]) \wedge (N = x_N[u])$. (Recall that the result of the concat function contains enough information to recover its arguments.) This transformation replaces the function symbols pkgen, skgen, sign, and verify with primed function symbols pkgen$'$, skgen$'$, sign$'$, and verify$'$ respectively, to avoid repeated applications of the unforgeability of signatures with the same key. (The unforgeability of signatures is applied only to unprimed symbols.)

The soundness of the game transformations shows that $G_0 \approx G_1$. We will prove that G_1 satisfies the correspondences (4) and (6) with any public variables V, in particular with $V = \emptyset$. By Lemma 26, G_0 also satisfies these correspondences with public variables $V = \emptyset$. Let us sketch how the proof of correspondence (4) for the game G_1 will proceed. Let Q'_{1A} and Q'_{1B} such that $Q_{1A} = !^{i_A \leq n} Q'_{1A}$ and $Q_{1B} = !^{i_B \leq n} Q'_{1B}$. Assume that event e_B is executed in the copy of Q'_{1B} of index i_B, that is, $e_B(x_{pk_A}[i_B], B, N[i_B])$ is executed. (Recall that the variables x_{pk_A}, N, u, ... are implicitly arrays.) Then the condition of the find above e_B holds, that is, $m[u[i_B]]$, $x_B[u[i_B]]$, and $x_N[u[i_B]]$ are defined, $x_{pk_A}[i_B] = pk_A$, $B = x_B[u[i_B]]$, and $N[i_B] = x_N[u[i_B]]$. Moreover, since $m[u[i_B]]$ is defined, the assignment that defines m has been executed in the copy of Q'_{1A} of index $i_A = u[i_B]$. Then the event $e_A(pk_A, x_B, x_N)$, located above the definition of m, must have been executed in that copy of Q'_{1A}, that is, $e_A(pk_A, x_B[u[i_B]], x_N[u[i_B]])$ has been executed.

The equalities in the condition of the find imply that this event is also $e_A(x_{pk_A}[i_B], B, N[i_B])$. To sum up, if $e_B(x_{pk_A}[i_B], B, N[i_B])$ has been executed, then $e_A(x_{pk_A}[i_B], B, N[i_B])$ has been executed, so we have the correspondence (4). This reasoning is typical of the way the prover shows correspondences. In particular, the conditions of array lookups are key in these proofs, because they allow us to relate values in processes that run in parallel (here, the processes that represent A and B), and interesting correspondences relate events that occur in such processes. Next, we detail and formalize this reasoning, both for non-injective and injective correspondences.

4.2.2 Non-unique Events

The only correspondence that involves a non-unique event e is $\mathsf{event}(e) \Rightarrow \text{false}$, and it is simply proved by noticing that the event e no longer occurs in the game after the transformation **prove_unique** (Section 5.1.4). Therefore, non-unique events are not concerned by the proofs of Sections 4.2.3 and 4.2.4.

4.2.3 Non-injective Correspondences

Intuitively, in order to prove that Q_0 satisfies a non-injective correspondence $\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi$, with $\widetilde{x} = \text{var}(\psi)$ and $\widetilde{y} = \text{var}(\phi) \setminus \text{var}(\psi)$, we collect all facts that hold at events in ψ and show that these facts imply ϕ using the equational prover.

When **casesInCorresp = false**, CryptoVerif uses $\mathcal{F}_\mu$ to collect these facts. When **casesInCorresp = true** (the default), it uses $\mathcal{F}_{\mu,c}$ for more precision. In this section, we detail the proof with $\mathcal{F}_{\mu,c}$. The usage of $\mathcal{F}_\mu$ can be considered as using a single case c, relying on Lemma 30 instead of Lemma 31. Formally, we collect facts that hold when the event F in ψ has been executed, as follows.

Definition 16 (μ executes F, $\mathcal{F}_{F,\mu,c}$) When $F = \mathsf{event}(e(M_1, \dots, M_m))$ and ${}^{\mu}\mathsf{event}\ e(M'_1, \dots, M'_m); \dots$ occurs in Q_0 or, for $m = 0$, ${}^{\mu}\mathsf{event\_abort}\ e$ or ${}^{\mu}\mathsf{find}[\mathsf{unique}_e] \dots$ occurs in Q_0, we say that μ *executes* F.

If μ executes F and for all ${}^{\mu}\mathsf{event}\ e(M'_1, \dots, M'_m); \dots$ in Q_0, $M'_1, \dots, M'_m$ are simple terms, then we define $\mathcal{F}^0_{F,\mu,c} = \mathcal{F}_{\mu,c} \cup \{M'_j = M_j \mid j \leq m\} \cup \{\mathit{lastdefprogrampoint}(\mu, I_\mu)$ if ${}^{\mu}\mathsf{event\_abort}\ e$ or ${}^{\mu}\mathsf{find}[\mathsf{unique}_e] \dots$ occurs in $Q_0\}$. If additionally F is not a non-unique event, then we define $\mathcal{F}_{F,\mu,c} = \mathcal{F}^0_{F,\mu,c} \cup \mathcal{F}^{\text{Fut}}_\mu$.

Intuitively, when the event F in ψ has been executed, it has been executed by some subterm or subprocess of Q_0, so there exists a subterm or subprocess ${}^{\mu}\mathsf{event}\ e(M'_1, \dots, M'_m); \dots$ or, for $m = 0$, ${}^{\mu}\mathsf{event\_abort}\ e$ or ${}^{\mu}\mathsf{find}[\mathsf{unique}_e] \dots$ in Q_0 such that, the event $e(M'_1, \dots, M'_m)$ has been executed and it is equal to the event F, hence $M'_j = M_j$ holds for $j \leq m$. Moreover, since the program point μ, which executes F, has been reached, $\mathcal{F}_{\mu,c}$ holds for some case c (Lemma 31). Furthermore, when the event aborts, it is the last step of the trace, so $\mathit{lastdefprogrampoint}(\mu, I_\mu)$ also holds. Hence $\mathcal{F}^0_{F,\mu,c}$ holds for some case c. Additionally, assuming we consider traces that do not execute non-unique events, since the adversary cannot stop execution of the process until the next output or $\mathsf{event\_abort}\ e$, $\mathcal{F}^{\text{Fut}}_\mu$ also holds (Lemma 32), so $\mathcal{F}_{F,\mu,c}$ holds for some case c. This is proved more formally in Lemma 37 below. (The case of $\mathsf{get}[\mathsf{unique}_e]$ is not mentioned in Definition 16 because it is excluded by Property 4.)

We restrict ourselves to the case in which $M'_1, \dots, M'_m$ are simple terms because only simple terms allowed in sets of facts.

Let θ be a substitution equal to the identity on the variables $\widetilde{x}$ of ψ. This substitution gives values to existentially quantified variables $\widetilde{y}$ of ϕ. We say that $\mathcal{F} \models\!\!\Rightarrow_\theta \phi$ when we can show that $\mathcal{F}$ implies $\theta\phi$. Formally, we define:

$\mathcal{F} \models\!\!\Rightarrow_\theta M$ if and only if $\mathcal{F} \cup \{\neg\theta M\}$ yields a contradiction

$\mathcal{F} \models\!\!\Rightarrow_\theta \mathsf{event}(e(M_1, \dots, M_m))$ if and only if there exist
$M'_0, \dots, M'_m$ such that $M'_0 : \mathsf{event}(e(M'_1, \dots, M'_m)) \in \mathcal{F}$
and $\mathcal{F} \cup \{\bigvee_{j=1}^m \theta M_j \neq M'_j\}$ yields a contradiction

$\mathcal{F} \models\!\!\Rightarrow_\theta \phi_1 \wedge \phi_2$ if and only if $\mathcal{F} \models\!\!\Rightarrow_\theta \phi_1$ and $\mathcal{F} \models\!\!\Rightarrow_\theta \phi_2$

$\mathcal{F} \models\!\!\Rightarrow_\theta \phi_1 \vee \phi_2$ if and only if $\mathcal{F} \models\!\!\Rightarrow_\theta \phi_1$ or $\mathcal{F} \models\!\!\Rightarrow_\theta \phi_2$

Terms θM are proved by contradiction, using the equational prover. Events θF are proved by looking for some event F' in $\mathcal{F}$ and showing by contradiction that $\theta F = F'$, using the equational prover.

Let $\varphi = [\![\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi]\!]$ be a non-injective correspondence that does not use non-unique events, with $\psi = F_1 \wedge \dots \wedge F_m$, $\widetilde{x} = \mathrm{var}(\psi)$, and $\widetilde{y} = \mathrm{var}(\phi) \setminus \mathrm{var}(\psi)$. Suppose that, in Q_0, the arguments of the events that occur in ψ are always simple terms. Suppose that, for all $j \leq m$, μ_j that executes F_j and c_j is a case for $\mathcal{F}_{\mu_j,c_j}$. For $j \leq m$, let θ_j be a renaming of I_{μ_j} to fresh replication indices. (The renamings θ_j have pairwise disjoint images.) Let θ be a family parameterized by $\mu_1, c_1, \dots, \mu_m, c_m$ of substitutions equal to the identity on $\widetilde{x}$. We define $\mathrm{prove}^\varphi(\theta, \mu_1, c_1, \dots, \mu_m, c_m) = (\theta_1 \mathcal{F}_{F_1,\mu_1,c_1} \cup \dots \cup \theta_m \mathcal{F}_{F_m,\mu_m,c_m} \models\!\!\Rightarrow_{\theta(\mu_1,c_1,\dots,\mu_m,c_m)} \phi)$. This function defines the algorithm that we use to prove the correspondence φ assuming for all $j \leq m$, F_j is executed in μ_j and we are in case c_j. We also define $\mathrm{prove}^\varphi(\theta, \mathcal{S}) = \bigwedge_{(\mu_1,c_1,\dots,\mu_m,c_m) \in \mathcal{S}} \mathrm{prove}^\varphi(\theta, \mu_1, c_1, \dots, \mu_m, c_m)$.

Non-injective correspondences are proved as follows.

Proposition 2 *Let $\varphi = [\![\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi]\!]$ be a non-injective correspondence that does not use non-unique events, with $\psi = F_1 \wedge \dots \wedge F_m$, $\widetilde{x} = \mathrm{var}(\psi)$, and $\widetilde{y} = \mathrm{var}(\phi) \setminus \mathrm{var}(\psi)$. Let Q_0 be a process that satisfies Properties 4 and 5. Suppose that, in Q_0, the arguments of the events that occur in ψ are always simple terms. Let $\mathcal{S} = \{(\mu_1, c_1, \dots, \mu_m, c_m) \mid \forall j \leq m, \mu_j$ executes F_j and c_j is a case for $\mathcal{F}_{\mu_j,c_j}\}$. If there exists a family of substitutions θ equal to the identity on $\widetilde{x}$ such that $\mathrm{prove}^\varphi(\theta, \mathcal{S})$ and for all evaluation contexts C acceptable for Q_0, $\Pr[C[Q_0] \preceq \neg\{\![\mathrm{prove}^\varphi(\theta, \mathcal{S})]\!\}] \leq p(C)$, then $\mathsf{Bound}_{Q_0}(V, \varphi, D_{\mathrm{false}}, p)$ for any V.*

Intuitively, when $\psi = F_1 \wedge \dots \wedge F_m$ holds, $\theta_1 \mathcal{F}_{F_1,\mu_1,c_1} \cup \dots \cup \theta_m \mathcal{F}_{F_m,\mu_m,c_m}$ hold for some μ_1, c_1, ..., μ_m, c_m. For some θ equal to the identity on ψ, $\theta_1 \mathcal{F}_{F_1,\mu_1,c_1} \cup \dots \cup \theta_m \mathcal{F}_{F_m,\mu_m,c_m}$ implies $\theta\phi$, so $\theta\phi$ holds. Hence the correspondence is satisfied. The proof of Proposition 2 relies on the following properties and lemmas. We have

$$\{\![\mathcal{F} \models\!\!\Rightarrow_\theta M]\!\} = \forall \widetilde{z} \in \widetilde{T''}, \neg\left(\bigwedge \mathcal{F} \wedge \neg\theta M\right)$$

$$\{\![\mathcal{F} \models\!\!\Rightarrow_\theta \mathsf{event}(e(M_1, \dots, M_m))]\!\} = \forall \widetilde{z} \in \widetilde{T''}, \neg\left(\bigwedge \mathcal{F} \wedge \bigvee_{j=1}^m \theta M_j \neq M'_j\right)$$
$$\text{for some } M'_0 : \mathsf{event}(e(M'_1, \dots, M'_m)) \in \mathcal{F}$$

$$\{\![\mathcal{F} \models\!\!\Rightarrow_\theta \phi_1 \wedge \phi_2]\!\} = \{\![\mathcal{F} \models\!\!\Rightarrow_\theta \phi_1]\!\} \wedge \{\![\mathcal{F} \models\!\!\Rightarrow_\theta \phi_2]\!\}$$

$$\{\![\mathcal{F} \models\!\!\Rightarrow_\theta \phi_1 \vee \phi_2]\!\} = \begin{cases} \{\![\mathcal{F} \models\!\!\Rightarrow_\theta \phi_1]\!\} & \text{if } \mathcal{F} \models\!\!\Rightarrow_\theta \phi_1 \\ \{\![\mathcal{F} \models\!\!\Rightarrow_\theta \phi_2]\!\} & \text{otherwise} \end{cases}$$

where $\widetilde{z}$ are the non-process variables in $\mathcal{F}$, in the image of θ, and in $\widetilde{x}$, and $\widetilde{T''}$ are their types.

Lemma 36 $\{\![\mathcal{F} \models\!\!\Rightarrow_\theta \phi]\!\} \Rightarrow \forall \widetilde{z} \in \widetilde{T''}, \neg(\bigwedge \mathcal{F} \wedge \neg\theta\phi)$*, where $\widetilde{z}$ are the non-process variables in $\mathcal{F}$, in the image of θ, and in $\widetilde{x}$, and $\widetilde{T''}$ are their types.*

Proof By induction on ϕ. $\square$

Lemma 37 *Let Q_0 be a process that satisfies Properties 4 and 5. Let Tr be a full trace of $C[Q_0]$. Let $\mu\mathcal{E}v$ be the sequence of events in the last configuration of Tr. Let $F = \mathsf{event}(e(\widetilde{M}))$ where $\widetilde{M}$ is a tuple of terms and e is an event that does not occur in C. Suppose that the arguments of e in Q_0 are always simple terms. Let ρ be a mapping of the variables of $\widetilde{M}$ and τ to their values. Suppose that $Tr, \rho \vdash F@\tau$.*

Then there exist a program point μ (in Q_0) that executes F and a case c such that, for any θ' renaming of I_μ to fresh replication indices, there exists a mapping σ with domain $\theta' I_\mu$ such that $\mu\mathcal{E}v(\rho(\tau)) = (\mu, \sigma(\theta' I_\mu)) : e(\dots)$ and $Tr, \sigma \cup \rho \vdash \theta' \mathcal{F}^0_{F,\mu,c}$. If additionally, Tr does not execute a non-unique event of Q_0, then $Tr, \sigma \cup \rho \vdash \theta' \mathcal{F}_{F,\mu,c}$.

Proof Let $E = E_{Tr}$. Since $Tr, \rho \vdash F@\tau$ and the variables of $\widetilde{M}$ and τ are defined in ρ, there exists $\widetilde{a}$ such that $\rho, \widetilde{M} \Downarrow \widetilde{a}$ and $\mu\mathcal{E}v(\rho(\tau)) = (\mu, \widetilde{a}_0) : e(\widetilde{a})$ for some μ and $\widetilde{a}_0$. The rule of the semantics that may have added this element to $\mu\mathcal{E}v$ is (Event), (EventAbort), (CtxEvent), (FindE), (Find3), (GetE), (Get3), or (EventT).

- In case (Event), the initial configuration of rule (Event) is of the form $E_1, (\sigma_1, {}^\mu\mathsf{event}\ e(\widetilde{a}); P), \mathcal{Q}_0, \mathcal{C}h_0, \mathcal{T}_1, \mu\mathcal{E}v_1$. By Lemma 8, Property 1 applied to this configuration, we have reductions

$$\begin{aligned} \mathit{Conf} &= E_0, (\sigma_0, {}^\mu\mathsf{event}\ e(\widetilde{M'}); P), \mathcal{Q}_0, \mathcal{C}h_0, \mathcal{T}_0, \mu\mathcal{E}v_0 \\ &\xrightarrow{p_0}_{t_0} \dots \xrightarrow{p_1}_{t_1} E_1, (\sigma_1, {}^\mu\mathsf{event}\ e(\widetilde{a}); P), \mathcal{Q}_0, \mathcal{C}h_0, \mathcal{T}_1, \mu\mathcal{E}v_1 \\ &\xrightarrow{1} E_1, (\sigma_1, P), \mathcal{Q}_0, \mathcal{C}h_0, \mathcal{T}_1, (\mu\mathcal{E}v_1, (\mu, \mathrm{Im}(\sigma_1)) : e(\widetilde{a})) \end{aligned}$$

  where ${}^\mu\mathsf{event}\ e(\widetilde{M'}); P$ is a subprocess of $C[Q_0]$ up to renaming of channels, by any number of applications of (Ctx) and a final application of (Event).

- In case (EventAbort), (Find3), or (Get3), the rules that can conclude with a process ${}^\mu P$ with $P = \mathsf{event\_abort}\ e$, $P = \mathsf{find}[\mathsf{unique}_e] \dots$, or $P = \mathsf{get}[\mathsf{unique}_e] \dots$ are (New), (Let), (If1), (If2), (Find1), (Find2), (Insert), (Get1), (Get2), (Output), (Event) and by Corollary 1, their target process is a subprocess of $C[Q_0]$ up to renaming of channels. So we have a reduction

$$\begin{aligned} \mathit{Conf} &= E_0, (\sigma_0, {}^\mu P), \mathcal{Q}_0, \mathcal{C}h_0, \mathcal{T}_0, \mu\mathcal{E}v_0 \\ &\xrightarrow{p}_t E_0, (\sigma_0, \mathsf{abort}), \mathcal{Q}_0, \mathcal{C}h_0, \mathcal{T}_0, (\mu\mathcal{E}v_0, (\mu, \mathrm{Im}(\sigma_0)) : e) \end{aligned}$$

  where ${}^\mu P$ is a subprocess of $C[Q_0]$ up to renaming of channels, by (EventAbort), (Find3), or (Get3).

- In case (EventT), the initial configuration of the rule (EventT) is of the form $E_1, \sigma_1, {}^\mu\mathsf{event}\ e(\widetilde{a}); N, \mathcal{T}_1, \mu\mathcal{E}v_1$. By Lemma 8, Property 5 applied to this configuration with $l = 0$, we have reductions

$$\begin{aligned} \mathit{Conf} &= E_0, \sigma_0, {}^\mu\mathsf{event}\ e(\widetilde{M'}); N, \mathcal{T}_0, \mu\mathcal{E}v_0 \\ &\xrightarrow{p_0}_{t_0} \dots \xrightarrow{p_1}_{t_1} E_1, \sigma_1, {}^\mu\mathsf{event}\ e(\widetilde{a}); N, \mathcal{T}_1, \mu\mathcal{E}v_1 \\ &\xrightarrow{1} E_1, \sigma_1, N, \mathcal{T}_1, (\mu\mathcal{E}v_1, (\mu, \mathrm{Im}(\sigma_1)) : e(\widetilde{a})) \end{aligned}$$

  where ${}^\mu\mathsf{event}\ e(\widetilde{M'}); N$ is a subterm of $C[Q_0]$, by any number of applications of (CtxT) and a final application of (EventT).

- In case (CtxEvent), the only rule that can conclude with a process $C[\textsf{event\_abort}\ (\mu, \widetilde{a}_0) : e]$ is (Ctx). (The rules (New), (Let), (If1), (If2), (Find1), (Find2), (Insert), (Get1), (Get2), (Output), (Event) cannot conclude with $C[\textsf{event\_abort}\ (\mu, \widetilde{a}_0) : e]$ because, by Corollary 1, their target process is a subprocess of $C[Q_0]$ up to renaming of channels, and the initial process $C[Q_0]$ does not contain the abort event value $\textsf{event\_abort}\ (\mu, \widetilde{a}_0) : e$.) Hence, there is a rule that concludes with a term $\textsf{event\_abort}\ (\mu, \widetilde{a}_0) : e$.

  In cases (FindE) and (GetE), there is also a rule that concludes with a term $\textsf{event\_abort}\ (\mu, \widetilde{a}_0) : e$.

  The only rules that conclude with a term $\textsf{event\_abort}\ (\mu, \widetilde{a}_0) : e$ are (FindTE), (FindT3), (GetTE), (GetT3), (EventAbortT), and (CtxEventT). (It cannot be (NewT), (LetT), (IfT1), (IfT2), (FindT1), (FindT2), (InsertT), (GetT1), (GetT2), (EventT), (DefinedYes) because, by Corollary 1, their target term is a subterm of $C[Q_0]$, and the initial process $C[Q_0]$ does not contain the abort event value $\textsf{event\_abort}\ (\mu, \widetilde{a}_0) : e$.) In cases (FindTE) and (GetTE), there is recursively another rule that concludes with $\textsf{event\_abort}\ (\mu, \widetilde{a}_0) : e$. In case (CtxEventT), the only rule that can conclude with $C[\textsf{event\_abort}\ (\mu, \widetilde{a}_0) : e]$ is (CtxT), so there is recursively another rule that concludes with $\textsf{event\_abort}\ (\mu, \widetilde{a}_0) : e$. Therefore, $\textsf{event\_abort}\ (\mu, \widetilde{a}_0) : e$ ultimately comes from an application of (EventAbortT), (FindT3), or (GetT3):

$$\mathit{Conf} = E_0, \sigma_0, {}^{\mu}N, \mathcal{T}_0, \mu\mathcal{E}v_0 \xrightarrow{p}_t E_0, \sigma_0, \textsf{event\_abort}\ (\mu, \mathrm{Im}(\sigma_0)) : e, \mathcal{T}_0, \mu\mathcal{E}v_0$$

  where $N = \textsf{event\_abort}\ e$, $N = \textsf{find}[\textsf{unique}_e] \dots$, or $N = \textsf{get}[\textsf{unique}_e] \dots$ Furthermore, the only rules that can conclude with such a term ${}^{\mu}N$ are (NewT), (IfT1), (IfT2), (LetT), (FindT1), (FindT2), (InsertT), (GetT1), (GetT2), (EventT), (DefinedYes) and, by Corollary 1, their target term is a subterm of $C[Q_0]$.

In all cases, since e does not occur in C, μ is in fact a program point of Q_0. Therefore, the process or term at program point μ in Q_0 is of the form ${}^{\mu}\textsf{event}\ e(\widetilde{M'}); \dots$, ${}^{\mu}\textsf{event\_abort}\ e$, ${}^{\mu}\textsf{find}[\textsf{unique}_e] \dots$, or ${}^{\mu}\textsf{get}[\textsf{unique}_e] \dots$. In cases (Event) and (EventT), we have $\sigma_0 = \sigma_1$ by Lemma 3. In all cases, $\mathrm{Dom}(\sigma_0) = I_\mu$ are the current replication indices at program point μ by Lemma 2, and $\mathrm{Im}(\sigma_0) = \widetilde{a}_0$, so $\sigma_0 = [I_\mu \mapsto \widetilde{a}_0]$. Moreover, $\widetilde{M'}$ are simple terms, so their evaluation can be written $E_0, \{I_\mu \mapsto \widetilde{a}_0\}, \widetilde{M'} \Downarrow \widetilde{a}$. Let $\sigma = \{\theta' I_\mu \mapsto \widetilde{a}_0\}$. We have $\mu\mathcal{E}v(\rho(\tau)) = (\mu, \sigma(\theta' I_\mu)) : e(\widetilde{a})$.

We have $E_0, \sigma_0, \widetilde{M'} \Downarrow \widetilde{a}$ so $E_0, \sigma, \theta'\widetilde{M'} \Downarrow \widetilde{a}$. The environment E_{Tr} extends E_0, so $E_{\mathit{Tr}}, \sigma, \theta'\widetilde{M'} \Downarrow \widetilde{a}$, so $\mathit{Tr}, \sigma \cup \rho \vdash \theta'\widetilde{M'} = \widetilde{M}$.

The configuration Conf is at program point μ in Tr, so by Corollary 3, we have $\mathit{Tr}, \sigma \vdash \theta'\mathcal{F}_{\mu,c}$.

When the process or term at μ is ${}^{\mu}\textsf{event\_abort}\ e$, ${}^{\mu}\textsf{find}[\textsf{unique}_e] \dots$, or ${}^{\mu}\textsf{get}[\textsf{unique}_e] \dots$, the environment and replication indices in Conf are the same as at the end of the trace, since the execution after Conf applies (EventAbort), (Find3), or (Get3), which terminate the trace keeping the same environment and replication indices as in Conf or (EventAbortT), (FindT3), or (GetT3) which build an abort event value keeping the same environment and replication indices as in Conf, followed by some rules among (FindTE), (GetTE), (CtxT), (CtxEventT), (FindE), (GetE), (Ctx), and (CtxEvent), which preserve the environment and replication indices that come with the abort event value. So $\mathit{Tr}, \sigma \vdash \mathit{lastdefprogrampoint}(\mu, \theta' I_\mu)$.

Therefore, μ executes F and $\mathit{Tr}, \sigma \cup \rho \vdash \theta'\mathcal{F}^0_{F,\mu,c}$ for some c.

Suppose additionally that Tr does not execute any non-unique event of Q_0. Let Tr'' be the prefix of Tr that stops at the first evaluated output $c[\widetilde{a'}]\langle b\rangle; Q$ that follows Conf, or $\mathit{Tr}'' = \mathit{Tr}$ if Tr contains no output after Conf. By Lemma 32, we have $\mathit{Tr}'' \vdash \mathcal{F}^{\mathrm{Fut}}_{\mu}$, so $\mathit{Tr}'', \sigma \vdash \theta'\mathcal{F}^{\mathrm{Fut}}_{\mu}$.

Moreover, E extends $E_{Tr''}$ and by Lemma 3, $\mu\mathcal{E}v_{Tr}$ extends $\mu\mathcal{E}v_{Tr''}$, so $Tr, \sigma \vdash \theta'\mathcal{F}^{\mathrm{Fut}}_{\mu}$. Therefore, $Tr, \sigma \cup \rho \vdash \theta'\mathcal{F}_{F,\mu,c}$ for some c. □

Lemma 38 *Let $\varphi = [\![\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi]\!]$ be a non-injective correspondence that does not use non-unique events, with $\psi = F_1 \wedge \ldots \wedge F_m$, $\widetilde{x} = \mathrm{var}(\psi)$, and $\widetilde{y} = \mathrm{var}(\phi) \setminus \mathrm{var}(\psi)$. Let Q_0 be a process that satisfies Properties 4 and 5. Suppose that, in Q_0, the arguments of the events that occur in ψ are always simple terms.*

Let $\mathcal{S} = \{(\mu_1, c_1, \ldots, \mu_m, c_m) \mid \forall j \le m, \mu_j$ executes F_j and c_j is a case for $\mathcal{F}_{\mu_j,c_j}\}$. Let C be an evaluation context acceptable for Q_0 with public variables V that does not contain events used by φ. Let Tr be a full trace of $C[Q_0]$ that does not execute any non-unique event of Q_0. If $Tr \vdash \neg\varphi$, then for any θ family of substitutions equal to the identity on $\widetilde{x}$, $Tr \vdash \neg\{\![\mathrm{prove}^{\varphi}(\theta, \mathcal{S})]\!\}$.

Proof Since $Tr \vdash \neg\varphi$, we have $Tr \vdash \exists \widetilde{x} \in \widetilde{T}, F_1 \wedge \cdots \wedge F_m \wedge \forall \widetilde{y} \in \widetilde{T'}, \neg\phi$. So there exists ρ that maps $\widetilde{x}$ to elements of $\widetilde{T}$ such that $Tr, \rho \vdash F_1 \wedge \cdots \wedge F_m \wedge \forall \widetilde{y} \in \widetilde{T'}, \neg\phi$. By Lemma 37, for all $j \le m$, there exists a program point μ_j (in Q_0) that executes F_j and a case c_j such that, for any θ_j renaming of I_{μ_j} to fresh replication indices, there exists a mapping σ_j with domain $\theta_j I_{\mu_j}$ such that $Tr, \sigma_j \cup \rho \vdash \theta_j \mathcal{F}_{F_j,\mu_j,c_j}$. Since $Tr, \rho \vdash \forall \widetilde{y} \in \widetilde{T'}, \neg\phi$, we have $Tr, \rho \vdash \neg\theta(\mu_1, c_1, \ldots, \mu_m, c_m)\phi$. Therefore, $Tr, \sigma_1 \cup \cdots \cup \sigma_m \cup \rho \vdash \theta_1 \mathcal{F}_{F_1,\mu_1,c_1} \cup \cdots \cup \theta_m \mathcal{F}_{F_m,\mu_m,c_m} \cup \{\neg\theta(\mu_1, c_1, \ldots, \mu_m, c_m)\phi\}$, so $Tr \vdash \exists \theta_1 I_{\mu_1}, \ldots, \exists \theta_m I_{\mu_m}, \exists \widetilde{x} \in \widetilde{T}, \bigwedge \mathcal{F}_0(\mu_1, c_1, \ldots, \mu_m, c_m) \wedge \neg\theta(\mu_1, c_1, \ldots, \mu_m, c_m)\phi$ where $\mathcal{F}_0(\mu_1, c_1, \ldots, \mu_m, c_m) = \theta_1 \mathcal{F}_{F_1,\mu_1,c_1} \cup \cdots \cup \theta_m \mathcal{F}_{F_m,\mu_m,c_m}$. We have

$$\begin{aligned}
&\{\![\mathrm{prove}^{\varphi}(\theta, \mu_1, c_1, \ldots, \mu_m, c_m)]\!\}\\
&\quad\Rightarrow \forall \theta_1 I_{\mu_1}, \ldots, \forall \theta_m I_{\mu_m}, \forall \widetilde{x} \in \widetilde{T}, \neg\left(\bigwedge \mathcal{F}_0(\mu_1, c_1, \ldots, \mu_m, c_m) \wedge \neg\theta(\mu_1, c_1, \ldots, \mu_m, c_m)\phi\right)
\end{aligned}$$

by Lemma 36. (The non-process variables in $\mathcal{F}_0(\mu_1, c_1, \ldots, \mu_m, c_m)$, in the image of $\theta(\mu_1, c_1, \ldots, \mu_m, c_m)$, and in $\widetilde{x}$ are in $\theta_1 I_{\mu_1}, \ldots, \theta_m I_{\mu_m}, \widetilde{x}$.) So $Tr \vdash \neg\{\![\mathrm{prove}^{\varphi}(\theta, \mu_1, c_1, \ldots, \mu_m, c_m)]\!\}$, so $Tr \vdash \neg\{\![\mathrm{prove}^{\varphi}(\theta, \mathcal{S})]\!\}$. □

Proof of Proposition 2 Let C be an evaluation context acceptable for Q_0 with public variables V that does not contain events used by φ. We have

$$\begin{aligned}
\mathsf{Adv}_{Q_0}(C, \varphi, D_{\mathrm{false}}) &= \Pr[C[Q_0] : \neg\varphi \wedge \neg\mathsf{NonUnique}_{Q_0}] \\
&\le \Pr[C[Q_0] : \neg\{\![\mathrm{prove}^{\varphi}(\theta, \mathcal{S})]\!\}] && \text{by Lemma 38}\\
&\le \Pr[C[Q_0] \preceq \neg\{\![\mathrm{prove}^{\varphi}(\theta, \mathcal{S})]\!\}] && \text{by Lemma 1}\\
&\le p(C)
\end{aligned}$$

So $\mathsf{Bound}_{Q_0}(V, \varphi, D_{\mathrm{false}}, p)$. □

Example 6 Let us prove that the example G_1 satisfies (4). We first study the facts $\mathcal{F}_{\mu_B}$ that hold at the program point μ_B that executes event e_B. $\mathcal{F}_{\mu_B}$ contains $\mathsf{defined}(m[u[i_B]])$, $\mathsf{defined}(x_B[u[i_B]])$, $\mathsf{defined}(x_N[u[i_B]])$, $x_{pk_A}[i_B] = pk_A$, $B = x_B[u[i_B]]$, and $N[i_B] = x_N[u[i_B]]$, because the condition of find holds at μ_B. Moreover, we have $\mathsf{defined}(m[u[i_B]]) \in \mathcal{F}_{\mu_B}$, and, when $m[i_A]$ is defined, $(\mu_A, i_A) : \mathsf{event}(e_A(pk_A, x_B[i_A], x_N[i_A]))$ holds, so $(\mu_A, i_A) : \mathsf{event}(e_A(pk_A, x_B[i_A], x_N[i_A]))\{u[i_B]/i_A\} \in \mathcal{F}_{\mu_B}$, that is, $(\mu_A, u[i_B]) : \mathsf{event}(e_A(pk_A, x_B[u[i_B]], x_N[u[i_B]])) \in \mathcal{F}_{\mu_B}$. In other words, since m is defined at index $u[i_B]$, event e_A has been executed in the copy of Q'_{1A} of index $u[i_B]$. ($\mathcal{F}_{\mu_B}$ also contains other facts, which are useless for proving the desired correspondences, so we do not list them.)

For $\psi = F = \mathsf{event}(e_B(x, y, z))$, μ_B is the only program point that executes F, so this event has been executed in some copy of Q'_{1B} of index i'_B, with $x_{pk_A}[i'_B] = x, B = y, N[i'_B] = z$. Then, when ψ holds, the facts $\theta'\mathcal{F}_{F,\mu_B} \supseteq \theta'\mathcal{F}_{\mu_B} \cup \{x_{pk_A}[i'_B] = x, B = y, N[i'_B] = z\}$ hold for some value of i'_B, with $\theta' = \{i'_B/i_B\}$. (We consider a single case c here, so we can simply omit the case c.)

Furthermore, the substitution $\theta(\mu_B)$ is the identity since all variables of ϕ also occur in ψ. Then we just have to show that $\theta'\mathcal{F}_{F,\mu_B}$ implies $\phi = \mathsf{event}(e_A(x, y, z))$, that is, $\theta'\mathcal{F}_{F,\mu_B} \models\!\!\!\Rightarrow_{\theta(\mu_B)} \mathsf{event}(e_A(x, y, z))$. Since $(\mu_A, u[i_B]) : \mathsf{event}(e_A(pk_A, x_B[u[i_B]], x_N[u[i_B]])) \in \mathcal{F}_{\mu_B}$, we have $(\mu_A, u[i'_B]) : \mathsf{event}(e_A(pk_A, x_B[u[i'_B]], x_N[u[i'_B]])) \in \theta'\mathcal{F}_{F,\mu_B}$, so the equational prover just has to prove by contradiction that $e_A(pk_A, x_B[u[i'_B]], x_N[u[i'_B]]) = e_A(x, y, z)$, that is, $pk_A = x$, $x_B[u[i'_B]] = y$, and $x_N[u[i'_B]] = z$. The proof succeeds using the following equalities of $\theta'\mathcal{F}_{F,\mu_B}$: $x_{pk_A}[i'_B] = x$, $B = y$, $N[i'_B] = z$, $x_{pk_A}[i'_B] = pk_A$, $B = x_B[u[i'_B]]$, and $N[i'_B] = x_N[u[i'_B]]$.

Hence, G_1 satisfies (4) with any public variables V: if $\psi = \mathsf{event}(e_B(x, y, z))$ has been executed, then $\phi = \mathsf{event}(e_A(x, y, z))$ has been executed.

In the implementation, the substitution θ is initially defined as the identity on $\widetilde{x} = \mathrm{var}(\psi)$. It is defined on other variables when checking $\mathcal{F} \models\!\!\!\Rightarrow_\theta M$ by trying to find θ such that $\theta M \in \mathcal{F}$, and when checking $\mathcal{F} \models\!\!\!\Rightarrow_\theta \mathsf{event}(e(M_1, \dots, M_m))$ by trying to find θ such that $\theta\mathsf{event}(e(M_1, \dots, M_m)) \in \mathcal{F}$. When we do not manage to find the image by θ of all variables of M, resp. $M_1, \dots, M_m$, the check fails. When there are several suitable facts $\theta M \in \mathcal{F}$ or $\theta\mathsf{event}(e(M_1, \dots, M_m)) \in \mathcal{F}$, the system tries all possibilities.

4.2.4 Injective Correspondences

Injective correspondences are more difficult to check than non-injective ones, because they require distinguishing between several executions of the same event. We achieve that by relying on the pair (program points, replication indices) that is recorded in the sequence $\mu\mathcal{E}v$ together with each event: distinct executions of events either occur at different program points or have different values of replication indices.

We extend Definition 16 to injective events, with exactly the same definition as for non-injective events.

The proof of injective correspondences extends that for non-injective correspondences: for a correspondence $\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T}'; \phi$, we additionally prove that distinct executions of the injective events of ψ correspond to distinct executions of each injective event of ϕ, that is, if the injective events of ψ have different pairs (program point, replication indices), then each injective event of ϕ has a different pair (program point, replication indices). In order to achieve this proof, we collect the following information for each injective event of ϕ:

- the set of facts $\mathcal{F}$ that are known to hold, which will be used to reason on replication indices of events;
- the program point and replication indices of the considered injective event of ϕ, stored in a pair M_0; these program point and indices are computed when we prove that this event is executed;
- the program point and replication indices of the injective events of ψ, stored as a mapping $\mathcal{I} = \{j \mapsto (\mu_j, \theta_j I_{\mu_j}) \mid F_j \text{ is an injective event}\}$, where $\psi = F_1 \wedge \ldots \wedge F_m$, μ_j is the program point that executes F_j, and θ_j is a renaming of I_{μ_j} to fresh replication indices, for $j \le m$;
- the set $\mathcal{V}$ containing the replication indices in $\mathcal{F}$ and the variables $\widetilde{x}$ of ψ; these variables will be renamed to fresh variables in order to avoid conflicts of variable names between different events.

This information is stored in a set $\mathcal{S}$, which contains quadruples $(\mathcal{F}, M_0, \mathcal{I}, \mathcal{V})$. We will show that, if the pair (program point, replication indices) of two executions of the injective events of

ψ are different, then the pair (program, replication indices) of the corresponding executions of the considered injective event of ϕ are also different. The equality between pairs (program point, replication indices) is obviously defined as the equality between program points and between replication indices. Formally, we consider $(\mathcal{F}, M_0, \mathcal{I}, \mathcal{V})$ and $(\mathcal{F}', M_0', \mathcal{I}', \mathcal{V}')$ in $\mathcal{S}$. We rename the variables $\mathcal{V}'$ of the second element to fresh variables by a substitution θ'' and show that, if $\mathcal{I} \neq \theta''\mathcal{I}'$, then $M_0 \neq \theta'' M_0'$ (knowing $\mathcal{F}$ and $\theta''\mathcal{F}'$). This property implies injectivity.

Since this reasoning is done for each injective event in ϕ, we collect the associated sets $\mathcal{S}$ in a pseudo-formula $\mathcal{C}$, obtained by replacing each injective event of ϕ with a set $\mathcal{S}$ and all other leaves of ϕ with $\bot$.

We say that $\vdash \mathcal{C}$ when for all non-bottom leaves $\mathcal{S}$ of $\mathcal{C}$, for all $(\mathcal{F}, M_0, \mathcal{I}, \mathcal{V})$, $(\mathcal{F}', M_0', \mathcal{I}', \mathcal{V}')$ in $\mathcal{S}$, $\mathcal{F} \cup \theta''\mathcal{F}' \cup \{\bigvee_{j \in \mathrm{Dom}(\mathcal{I})} \mathcal{I}(j) \neq \theta''\mathcal{I}'(j), M_0 = \theta'' M_0'\}$ yields a contradiction, where the substitution θ'' is a renaming of variables in $\mathcal{V}'$ to distinct fresh variables. As explained above, the condition $\vdash \mathcal{C}$ guarantees injectivity.

We extend the definition of $\mathcal{F} \models\!\Rightarrow_\theta \phi$ used for non-injective correspondences to $\mathcal{F} \models\!\Rightarrow_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi$, which means that $\mathcal{F}$ implies $\theta\phi$ and $\mathcal{C}$ correctly collects the tuples $(\mathcal{F}, M_0, \mathcal{I}, \mathcal{V})$ associated to this proof. Formally, we define:

$\mathcal{F} \models\!\Rightarrow_\theta^{\mathcal{I},\mathcal{V},\bot} M$ if and only if $\mathcal{F} \cup \{\neg\theta M\}$ yields a contradiction

$\mathcal{F} \models\!\Rightarrow_\theta^{\mathcal{I},\mathcal{V},\bot} \mathsf{event}(e(M_1, \dots, M_m))$ if and only if
there exist $M_0', \dots, M_m'$ such that $M_0' : \mathsf{event}(e(M_1', \dots, M_m')) \in \mathcal{F}$ and
$\mathcal{F} \cup \{\bigvee_{j=1}^m \theta M_j \neq M_j'\}$ yields a contradiction

$\mathcal{F} \models\!\Rightarrow_\theta^{\mathcal{I},\mathcal{V},\mathcal{S}} \mathsf{inj\text{-}event}(e(M_1, \dots, M_m))$ if and only if
there exist $M_0', \dots, M_m'$ such that $M_0' : \mathsf{event}(e(M_1', \dots, M_m')) \in \mathcal{F}$,
$\mathcal{F} \cup \{\bigvee_{j=1}^m \theta M_j \neq M_j'\}$ yields a contradiction, and $(\mathcal{F}, M_0', \mathcal{I}, \mathcal{V}) \in \mathcal{S}$.

$\mathcal{F} \models\!\Rightarrow_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}_1 \wedge \mathcal{C}_2} \phi_1 \wedge \phi_2$ if and only if $\mathcal{F} \models\!\Rightarrow_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}_1} \phi_1$ and $\mathcal{F} \models\!\Rightarrow_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}_2} \phi_2$

$\mathcal{F} \models\!\Rightarrow_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}_1 \vee \mathcal{C}_2} \phi_1 \vee \phi_2$ if and only if $\mathcal{F} \models\!\Rightarrow_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}_1} \phi_1$ or $\mathcal{F} \models\!\Rightarrow_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}_2} \phi_2$

These formulas differ from the non-injective case in that we propagate $\mathcal{I}$, $\mathcal{V}$, $\mathcal{C}$ and, in the case of injective events, we make sure that quadruples $(\mathcal{F}, M_0', \mathcal{I}, \mathcal{V})$ are collected correctly by requiring that $(\mathcal{F}, M_0', \mathcal{I}, \mathcal{V}) \in \mathcal{S}$.

Let $\varphi = [\![\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T}'; \phi]\!]$ be a correspondence that does not use non-unique events, with $\psi = F_1 \wedge \dots \wedge F_m$, $\widetilde{x} = \mathrm{var}(\psi)$, and $\widetilde{y} = \mathrm{var}(\phi) \setminus \mathrm{var}(\psi)$. Suppose that, in Q_0, the arguments of the events that occur in ψ are always simple terms. Suppose that, for all $j \leq m$, μ_j executes F_j and c_j is a case for $\mathcal{F}_{\mu_j,c_j}$. For $j \leq m$, let θ_j be a renaming of I_{μ_j} to fresh replication indices. (The renamings θ_j have pairwise disjoint images.) Let $\mathcal{C}$ be a pseudo formula and θ be a family parameterized by $\mu_1, c_1, \dots, c_m, \mu_m$ of substitutions equal to the identity on $\widetilde{x}$. We define $\mathrm{prove}^\varphi(\mathcal{C}, \theta, \mu_1, c_1, \dots, \mu_m, c_m) = (\mathcal{F} \models\!\Rightarrow_{\theta(\mu_1, c_1, \dots, \mu_m, c_m)}^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi)$ where $\mathcal{F} = \theta_1 \mathcal{F}_{F_1,\mu_1,c_1} \cup \dots \cup \theta_m \mathcal{F}_{F_m,\mu_m,c_m}$, $\mathcal{I} = \{j \mapsto (\mu_j, \theta_j I_{\mu_j}) \mid F_j \text{ is an injective event}\}$, and $\mathcal{V} = \mathrm{var}(\theta_1 I_{\mu_1}) \cup \dots \cup \mathrm{var}(\theta_m I_{\mu_m}) \cup \{\widetilde{x}\}$. The algorithm $\mathrm{prove}^\varphi(\mathcal{C}, \theta, \mu_1, \dots, \mu_m)$ shows that the non-injective version of the correspondence φ holds assuming the events in $\psi = F_1 \wedge \dots \wedge F_m$ are executed at program points $\mu_1, \dots, \mu_m$ respectively. Indeed, in this case, the facts $\mathcal{F} = \theta_1 \mathcal{F}_{F_1,\mu_1} \cup \dots \cup \theta_m \mathcal{F}_{F_m,\mu_m}$ hold and the formula $\mathcal{F} \models\!\Rightarrow_{\theta(\mu_1, \dots, \mu_m)}^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi$ shows that this implies $\theta(\mu_1, \dots, \mu_m)\phi$. (The substitution $\theta(\mu_1, \dots, \mu_m)\phi$ determines the values of $\widetilde{y}$.) Additionally, $\mathrm{prove}^\varphi(\mathcal{C}, \theta, \mu_1, \dots, \mu_m)$ makes sure that $\mathcal{C}$ correctly collects the information needed to prove injectivity. We also define $\mathrm{prove}^\varphi(\mathcal{C}, \theta, \mathcal{S}) = (\vdash \mathcal{C}) \wedge \bigwedge_{(\mu_1, c_1, \dots, \mu_m, c_m) \in \mathcal{S}} \mathrm{prove}^\varphi(\mathcal{C}, \theta, \mu_1, c_1, \dots, \mu_m, c_m)$. This algorithm proves the correspondence φ assuming the events in ψ are executed at program points in $\mathcal{S}$. It verifies injectivity via $\vdash \mathcal{C}$.

The following proposition shows the soundness of the proof of injective correspondences based on this algorithm.

Proposition 3 *Let $\varphi = [\![\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T}'; \phi]\!]$ be a correspondence that does not use non-unique events, with $\psi = F_1 \wedge \ldots \wedge F_m$, $\widetilde{x} = \mathrm{var}(\psi)$, and $\widetilde{y} = \mathrm{var}(\phi) \setminus \mathrm{var}(\psi)$. Let Q_0 be a process that satisfies Properties 4 and 5. Suppose that, in Q_0, the arguments of the events that occur in ψ are always simple terms.*

Let $\mathcal{S} = \{(\mu_1, c_1, \ldots, \mu_m, c_m) \mid \forall j \leq m, \mu_j$ executes F_j and c_j is a case for $\mathcal{F}_{\mu_j,c_j}\}$. Assume that there exist a pseudo-formula $\mathcal{C}$ and a family of substitutions θ equal to the identity on $\widetilde{x}$ such that $\mathrm{prove}^{\varphi}(\mathcal{C}, \theta, \mathcal{S})$. Assume that for all evaluation contexts C acceptable for Q_0, $\Pr[C[Q_0] \preceq \neg\{\![\mathrm{prove}^{\varphi}(\mathcal{C}, \theta, \mathcal{S})]\!\}] \leq p(C)$.

Then $\mathsf{Bound}_{Q_0}(V, \varphi, D_{\mathrm{false}}, p)$ for any V.

In the implementation, the value of $\mathcal{C}$ is computed by adding $(\mathcal{F}, M'_0, \mathcal{I}, \mathcal{V})$ to $\mathcal{S}$ when handling injective events during the checking of $\mathrm{prove}^{\varphi}(\mathcal{C}, \theta, \mathcal{S})$. We check $\vdash \mathcal{C}$ incrementally, after each addition of an element to $\mathcal{C}$. The proof of Proposition 3 relies on the following definitions and lemmas. We have

$$\{\![\mathcal{F} \models\!\!\Rightarrow_{\theta}^{\mathcal{I},\mathcal{V},\perp} M]\!\} = \forall \widetilde{z} \in \widetilde{T}'', \neg\left(\bigwedge \mathcal{F} \wedge \neg\theta M\right)$$

$$\{\![\mathcal{F} \models\!\!\Rightarrow_{\theta}^{\mathcal{I},\mathcal{V},\perp} \mathsf{event}(e(M_1, \ldots, M_m))]\!\} = \forall \widetilde{z} \in \widetilde{T}'', \neg\left(\bigwedge \mathcal{F} \wedge \bigvee_{j=1}^{m} \theta M_j \neq M'_j\right)$$
$$\text{for some } M'_0 : \mathsf{event}(e(M'_1, \ldots, M'_m)) \in \mathcal{F}$$

$$\{\![\mathcal{F} \models\!\!\Rightarrow_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{S}} \mathsf{inj\text{-}event}(e(M_1, \ldots, M_m))]\!\} = \forall \widetilde{z} \in \widetilde{T}'', \neg\left(\bigwedge \mathcal{F} \wedge \bigvee_{j=1}^{m} \theta M_j \neq M'_j\right)$$
$$\text{for some } M'_0 : \mathsf{event}(e(M'_1, \ldots, M'_m)) \in \mathcal{F} \text{ and } (\mathcal{F}, M'_0, \mathcal{I}, \mathcal{V}) \in \mathcal{S}$$

$$\{\![\mathcal{F} \models\!\!\Rightarrow_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}_1\wedge\mathcal{C}_2} \phi_1 \wedge \phi_2]\!\} = \{\![\mathcal{F} \models\!\!\Rightarrow_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}_1} \phi_1]\!\} \wedge \{\![\mathcal{F} \models\!\!\Rightarrow_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}_2} \phi_2]\!\}$$

$$\{\![\mathcal{F} \models\!\!\Rightarrow_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}_1\vee\mathcal{C}_2} \phi_1 \vee \phi_2]\!\} = \begin{cases} \{\![\mathcal{F} \models\!\!\Rightarrow_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}_1} \phi_1]\!\} & \text{if } \mathcal{F} \models\!\!\Rightarrow_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}_1} \phi_1 \\ \{\![\mathcal{F} \models\!\!\Rightarrow_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}_2} \phi_2]\!\} & \text{otherwise} \end{cases}$$

where $\widetilde{z} = \mathcal{V}$ and $\widetilde{T}''$ are the types of these variables. We also have

$$\{\![\vdash \mathcal{C}]\!\} = \bigwedge_{\mathcal{S}\neq\perp \text{ leaf of } \mathcal{C}} \; \bigwedge_{(\mathcal{F},M_0,\mathcal{I},\mathcal{V})\in\mathcal{S}} \; \bigwedge_{(\mathcal{F}',M'_0,\mathcal{I}',\mathcal{V}')\in\mathcal{S}} \forall \widetilde{z} \in \widetilde{T}'',$$
$$\neg\left(\bigwedge \mathcal{F} \wedge \bigwedge \theta''\mathcal{F}' \wedge \left(\bigvee_{j\in\mathrm{Dom}(\mathcal{I})} \mathcal{I}(j) \neq \theta''\mathcal{I}'(j)\right) \wedge M_0 = \theta'' M'_0\right)$$

where the substitution θ'' is a renaming of variables in $\mathcal{V}'$ to distinct fresh variables, $\widetilde{z} = \mathcal{V} \cup \theta''\mathcal{V}'$, and $\widetilde{T}''$ are the types of these variables.

We define $\mathrm{formula}(\mathcal{F} \models\!\!\Rightarrow_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi)$ as follows:

$$\mathrm{formula}(\mathcal{F} \models\!\!\Rightarrow_{\theta}^{\mathcal{I},\mathcal{V},\perp} M) = \theta M$$

$$\mathrm{formula}(\mathcal{F} \models\!\!\Rightarrow_{\theta}^{\mathcal{I},\mathcal{V},\perp} \mathsf{event}(e(M_1, \ldots, M_m))) = \theta\mathsf{event}(e(M_1, \ldots, M_m))$$

$$\mathrm{formula}(\mathcal{F} \models\!\!\Rightarrow_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{S}} \mathsf{inj\text{-}event}(e(M_1, \ldots, M_m))) =$$
$$\bigvee_{(\mathcal{F},N_0,\mathcal{I},\mathcal{V})\in\mathcal{S}} \exists \tau \in \mathbb{N}, N_0 : \theta\mathsf{event}(e(M_1, \ldots, M_m))@\tau$$

$$\mathrm{formula}(\mathcal{F} \models\!\!\Rightarrow_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}_1\wedge\mathcal{C}_2} \phi_1 \wedge \phi_2) = \mathrm{formula}(\mathcal{F} \models\!\!\Rightarrow_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}_1} \phi_1) \wedge \mathrm{formula}(\mathcal{F} \models\!\!\Rightarrow_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}_2} \phi_2)$$

$$\text{formula}(\mathcal{F} \models\!\!\Rightarrow_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}_1 \vee \mathcal{C}_2} \phi_1 \vee \phi_2) = \begin{cases} \text{formula}(\mathcal{F} \models\!\!\Rightarrow_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}_1} \phi_1) & \text{if } \mathcal{F} \models\!\!\Rightarrow_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}_1} \phi_1 \\ \text{formula}(\mathcal{F} \models\!\!\Rightarrow_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}_2} \phi_2) & \text{otherwise} \end{cases}$$

The formula formula$(\mathcal{F} \models\!\!\Rightarrow_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi)$ generalizes $\theta\phi$ to the case of injective events.

Lemma 39 $\{\![\mathcal{F} \models\!\!\Rightarrow_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi]\!\} \Rightarrow \forall \widetilde{z} \in \widetilde{T}'', \neg \left(\bigwedge \mathcal{F} \wedge \neg\text{formula}(\mathcal{F} \models\!\!\Rightarrow_\theta^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi)\right)$ *where* $\widetilde{z} = \mathcal{V}$ *and* $\widetilde{T}''$ *are the types of these variables.*

Proof By induction on ϕ. This result is similar to Lemma 36. The case of injective events is new. The case of disjunction differs, but is straightforward by induction hypothesis. □

The next lemma shows that, for events e in the considered correspondence, two distinct executions of event e have distinct pairs (program point, replication indices). When the term M contains no array accesses, we define $\sigma(M)$ by $\sigma, M \Downarrow \sigma(M)$.

Lemma 40 *Assume that the event* e *is used in the correspondence* φ. *Let* Q_0 *be a process that satisfies Property 4. Let* C *be an evaluation context acceptable for* Q_0 *with public variables* V *that does not contain events used by* φ. *If* $\text{initConfig}(C[Q_0]) \xrightarrow{p_1}_{t_1} \dots \xrightarrow{p_{m-1}}_{t_{m-1}} E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$, $\mu\mathcal{E}v(\tau) = (\mu, \widetilde{a}) : e(a_1, \dots, a_m)$, $\mu\mathcal{E}v(\tau') = (\mu', \widetilde{a}') : e(a'_1, \dots, a'_m)$, *and* $\tau \neq \tau'$, *then* $(\mu, \widetilde{a}) \neq (\mu', \widetilde{a}')$.

Proof Let us fix the event symbol e. We define the multisets $\mathit{Events}(\widetilde{a}, M)$, $\mathit{Events}(\widetilde{a}, P)$, and $\mathit{Events}(\widetilde{a}, Q)$ by

$$\mathit{Events}(\widetilde{a}, {}^\mu i) = \emptyset$$

$$\mathit{Events}(\widetilde{a}, {}^\mu x[M_1, \dots, M_m]) = \biguplus_{j \in \{1,\dots,m\}} \mathit{Events}(\widetilde{a}, M_j)$$

$$\mathit{Events}(\widetilde{a}, {}^\mu f(M_1, \dots, M_m)) = \biguplus_{j \in \{1,\dots,m\}} \mathit{Events}(\widetilde{a}, M_j)$$

$$\mathit{Events}(\widetilde{a}, {}^\mu \mathsf{new}\ x[\widetilde{i}] : T; N) = \mathit{Events}(\widetilde{a}, N)$$

$$\mathit{Events}(\widetilde{a}, {}^\mu \mathsf{let}\ x[\widetilde{i}] : T = M\ \mathsf{in}\ N) = \mathit{Events}(\widetilde{a}, M) \uplus \mathit{Events}(\widetilde{a}, N)$$

$$\mathit{Events}(\widetilde{a}, {}^\mu \mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ N') = \mathit{Events}(\widetilde{a}, M) \uplus \max(\mathit{Events}(\widetilde{a}, N), \mathit{Events}(\widetilde{a}, N'))$$

$$\mathit{Events}(\widetilde{a}, {}^\mu \mathsf{find}[\mathit{unique}?]\ (\textstyle\bigoplus_{j=1}^m \widetilde{u_j}[\widetilde{i}] = \widetilde{i_j} \leq \widetilde{n_j}\ \mathsf{suchthat}\ \mathsf{defined}(\widetilde{M_j}) \wedge M'_j\ \mathsf{then}\ N_j)\ \mathsf{else}\ N') =$$
$$\biguplus_{j=1}^m \biguplus_{\widetilde{a_j} \leq \widetilde{n_j}} \mathit{Events}((\widetilde{a}, \widetilde{a_j}), M'_j) \uplus \max(\max_{j=1}^m \mathit{Events}(\widetilde{a}, N_j), \mathit{Events}(\widetilde{a}, N'))$$

$$\mathit{Events}(\widetilde{a}, {}^\mu \mathsf{event}\ e'(M_1, \dots, M_m); M) = \biguplus_{j \in \{1,\dots,m\}} \mathit{Events}(\widetilde{a}, M_j) \uplus \mathit{Events}(\widetilde{a}, M) \text{ if } e' \neq e$$

$$\mathit{Events}(\widetilde{a}, {}^\mu \mathsf{event}\ e(M_1, \dots, M_m); M) = \{(\mu, \widetilde{a})\} \uplus \biguplus_{j \in \{1,\dots,m\}} \mathit{Events}(\widetilde{a}, M_j) \uplus \mathit{Events}(\widetilde{a}, M)$$

$$\mathit{Events}(\widetilde{a}, {}^\mu \mathsf{event\_abort}\ e') = \emptyset \text{ if } e' \neq e$$

$$\mathit{Events}(\widetilde{a}, {}^\mu \mathsf{event\_abort}\ e) = \{(\mu, \widetilde{a})\}$$

$$\mathit{Events}(\widetilde{a}, {}^\mu 0) = \emptyset$$

$$\mathit{Events}(\widetilde{a}, {}^\mu (Q_1 \mid Q_2)) = \mathit{Events}(\widetilde{a}, Q_1) \uplus \mathit{Events}(\widetilde{a}, Q_2)$$

$$\mathit{Events}(\widetilde{a}, {}^{\mu}!^{i \leq n} Q) = \biguplus_{a \in [1,n]} \mathit{Events}((\widetilde{a}, a), Q)$$

$$\mathit{Events}(\widetilde{a}, {}^{\mu}\mathsf{newChannel}\ c; Q) = \mathit{Events}(\widetilde{a}, Q)$$

$$\mathit{Events}(\widetilde{a}, {}^{\mu}c[M_1, \dots, M_l](x[\widetilde{i}] : T); P) = \biguplus_{j \in \{1,\dots,l\}} \mathit{Events}(\widetilde{a}, M_j) \uplus \mathit{Events}(\widetilde{a}, P)$$

$$\mathit{Events}(\widetilde{a}, {}^{\mu}\overline{c[M_1, \dots, M_l]}\langle N \rangle; Q) = \biguplus_{j \in \{1,\dots,l\}} \mathit{Events}(\widetilde{a}, M_j) \uplus \mathit{Events}(\widetilde{a}, N) \uplus \mathit{Events}(\widetilde{a}, Q)$$

$$\mathit{Events}(\widetilde{a}, {}^{\mu}\mathsf{new}\ x[\widetilde{i}] : T; P) = \mathit{Events}(\widetilde{a}, P)$$

$$\mathit{Events}(\widetilde{a}, {}^{\mu}\mathsf{let}\ x[\widetilde{i}] : T = M\ \mathsf{in}\ P) = \mathit{Events}(\widetilde{a}, M) \uplus \mathit{Events}(\widetilde{a}, P)$$

$$\mathit{Events}(\widetilde{a}, {}^{\mu}\mathsf{if}\ M\ \mathsf{then}\ P\ \mathsf{else}\ P') = \mathit{Events}(\widetilde{a}, M) \uplus \max(\mathit{Events}(\widetilde{a}, P), \mathit{Events}(\widetilde{a}, P'))$$

$$\mathit{Events}(\widetilde{a}, {}^{\mu}\mathsf{find}[\mathit{unique}?]\ (\textstyle\bigoplus_{j=1}^{m} \widetilde{u_j}[\widetilde{i}] = \widetilde{i_j} \leq \widetilde{n_j}\ \mathsf{suchthat}\ \mathsf{defined}(\widetilde{M_j}) \wedge M'_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P) =$$
$$\biguplus_{j=1}^{m} \biguplus_{\widetilde{a_j} \leq \widetilde{n_j}} \mathit{Events}((\widetilde{a}, \widetilde{a_j}), M'_j) \uplus \max(\max_{j=1}^{m} \mathit{Events}(\widetilde{a}, P_j), \mathit{Events}(\widetilde{a}, P))$$

$$\mathit{Events}(\widetilde{a}, {}^{\mu}\mathsf{event}\ e'(M_0, \dots, M_m); P) = \biguplus_{j \in \{1,\dots,m\}} \mathit{Events}(\widetilde{a}, M_j) \uplus \mathit{Events}(\widetilde{a}, P) \text{ if } e' \neq e$$

$$\mathit{Events}(\widetilde{a}, {}^{\mu}\mathsf{event}\ e(M_0, \dots, M_m); P) = \{(\mu, \widetilde{a})\} \uplus \biguplus_{j \in \{1,\dots,m\}} \mathit{Events}(\widetilde{a}, M_j) \uplus \mathit{Events}(\widetilde{a}, P)$$

$$\mathit{Events}(\widetilde{a}, {}^{\mu}\mathsf{event\_abort}\ e') = \emptyset \text{ if } e' \neq e$$

$$\mathit{Events}(\widetilde{a}, {}^{\mu}\mathsf{event\_abort}\ e) = \{(\mu, \widetilde{a})\}$$

$$\mathit{Events}(\widetilde{a}, {}^{\mu}\mathsf{yield}) = \emptyset$$

(get and insert are omitted because they do not occur in the game by Property 4.)

We define the multisets

$$\mathit{Events}(\mu\mathcal{E}v) = \{(\mu, \widetilde{a}) \mid (\mu, \widetilde{a}) : e(\dots) \in \mu\mathcal{E}v\}$$
$$\mathit{Events}(E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v) = \mathit{Events}(\mathrm{Im}(\sigma), M) \uplus \mathit{Events}(\mu\mathcal{E}v)$$
$$\mathit{Events}(E, \mathcal{Q}, \mathcal{C}h) = \biguplus_{(\sigma', Q') \in \mathcal{Q}} \mathit{Events}(\mathrm{Im}(\sigma'), Q')$$
$$\mathit{Events}(E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v) = \mathit{Events}(\mathrm{Im}(\sigma), P) \uplus \biguplus_{(\sigma', Q') \in \mathcal{Q}} \mathit{Events}(\mathrm{Im}(\sigma'), Q') \uplus \mathit{Events}(\mu\mathcal{E}v)$$

The latter multiset contains all pairs $(\mu, \widetilde{a})$ (program point, value of replication indices) for events $e(\dots)$ that may be executed in a trace that contains the configuration $E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$.

The multiset $\mathit{Events}(\sigma_0, C[Q_0])$ contains no duplicates. Indeed, we show by induction on M that $\mathit{Events}(\widetilde{a}, M)$ is included in the multiset $\{(\mu, \widetilde{a}')\}$ where μ is a program point inside M and $\widetilde{a}$ is a prefix of $\widetilde{a}'$, and similarly for P and Q. That allows to show that all multiset unions in the computation of *Events* are disjoint unions, since all recursive calls in the computation of *Events* are either with disjoint processes or terms, or with different extensions of $\widetilde{a}$.

Moreover, by induction on the derivations, if $E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v \xrightarrow{p}_t E', \sigma', M', \mathcal{T}', \mu\mathcal{E}v'$, then $\mathit{Events}(E, \sigma, M, \mathcal{T}, \mu\mathcal{E}v) \supseteq \mathit{Events}(E', \sigma', M', \mathcal{T}', \mu\mathcal{E}v')$; if $E, \mathcal{Q}, \mathcal{C}h \rightsquigarrow E', \mathcal{Q}', \mathcal{C}h'$, then $\mathit{Events}(E, \mathcal{Q}, \mathcal{C}h) \supseteq \mathit{Events}(E', \mathcal{Q}', \mathcal{C}h')$; and if $\mathit{Conf} \xrightarrow{p}_t \mathit{Conf}'$, then $\mathit{Events}(\mathit{Conf}) \supseteq \mathit{Events}(\mathit{Conf}')$.

Therefore, the multiset $\mathit{Events}(\emptyset, \{(\sigma_0, C[Q_0])\}, \mathrm{fc}(C[Q_0]))$ contains no duplicates, and neither do the multisets $\mathit{Events}(\mathrm{reduce}(\emptyset, \{(\sigma_0, C[Q_0])\}, \mathrm{fc}(C[Q_0])))$, $\mathit{Events}(\mathrm{initConfig}(C[Q_0]))$, and

$\mathit{Events}(\mathit{Conf})$, where $\mathit{Conf} = E, (\sigma, P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ is the final configuration of the considered trace. Hence, $\mathit{Events}(\mu\mathcal{E}v)$ contains no duplicates, which implies the desired result. □

Lemma 41 *Let $\varphi = [\![\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T}'; \phi]\!]$ be a correspondence that does not use non-unique events, with $\psi = F_1 \wedge \ldots \wedge F_m$, $\widetilde{x} = \mathrm{var}(\psi)$, and $\widetilde{y} = \mathrm{var}(\phi) \setminus \mathrm{var}(\psi)$. Let Q_0 be a process that satisfies Properties 4 and 5. Suppose that, in Q_0, the arguments of the events that occur in ψ are always simple terms.*

Let $\mathcal{S} = \{(\mu_1, c_1, \ldots, \mu_m, c_m) \mid \forall j \le m, \mu_j$ executes F_j and c_j is a case for $\mathcal{F}_{\mu_j, c_j}\}$. Let C be an evaluation context acceptable for Q_0 with public variables V that does not contain events used by φ. Let Tr be a full trace of $C[Q_0]$ that does not execute any non-unique event of Q_0. If $\mathit{Tr} \vdash \neg\varphi$, then for any family of substitutions θ equal to the identity on $\widetilde{x}$, for any pseudo-formula $\mathcal{C}$, $\mathit{Tr} \vdash \neg\{\![\mathrm{prove}^{\varphi}(\mathcal{C}, \theta, \mathcal{S})]\!\}$.

Proof By contraposition, we suppose that, $\mathit{Tr} \vdash \{\![\mathrm{prove}^{\varphi}(\mathcal{C}, \theta, \mathcal{S})]\!\}$, so for every μ_1 that executes F_1, ..., for every μ_m that executes F_m, for every c_1, ..., c_m, $\mathit{Tr} \vdash \{\![\mathrm{prove}^{\varphi}(\mathcal{C}, \theta, \mu_1, c_1, \ldots, \mu_m, c_m)]\!\}$ and $\mathit{Tr} \vdash \{\![\vdash \mathcal{C}]\!\}$, and we show that $\mathit{Tr} \vdash \varphi$.

Let $\mu\mathcal{E}v$ be the sequence of events in the last configuration of Tr.

We use the notations of Definition 12. We construct the functions $f_1, \ldots, f_k$ as follows. Let ρ be a mapping of τ_1, ..., τ_m to elements of $\mathbb{N}$ and of $\widetilde{x}$ to elements of $\widetilde{T}$. Suppose that $\mathit{Tr}, \rho \vdash \psi^{\tau}$. Then, for all $j \le m$, $\mathit{Tr}, \rho \vdash F_j^{\tau}$. By Lemma 37, there exists a program point μ_j (in Q_0) that executes F_j and a case c_j such that, for any θ_j renaming of I_{μ_j} to fresh replication indices, there exists a mapping σ_j with domain $\theta_j I_{\mu_j}$ such that $\mu\mathcal{E}v(\rho(\tau_j)) = (\mu_j, \sigma_j(\theta_j I_{pp_j})) : \ldots$ and $\mathit{Tr}, \sigma_j \cup \rho \vdash \theta_j \mathcal{F}_{F_j, \mu_j, c_j}$. Let $\rho_1 = \sigma_1 \cup \cdots \cup \sigma_m \cup \rho$. We have $\mathit{Tr}, \rho_1 \vdash \theta_1 \mathcal{F}_{F_1, \mu_1, c_1} \cup \cdots \cup \theta_m \mathcal{F}_{F_m, \mu_m, c_m}$.

Let $\mathcal{F}(\mathcal{C}, \theta, \mu_1, c_1, \ldots, \mu_m, c_m) = \theta_1 \mathcal{F}_{F_1, \mu_1, c_1} \cup \cdots \cup \theta_m \mathcal{F}_{F_m, \mu_m, c_m} \cup \{\neg\mathrm{formula}(\mathrm{prove}^{\varphi}(\mathcal{C}, \theta, \mu_1, c_1, \ldots, \mu_m, c_m))\}$. By Lemma 39, $\{\![\mathrm{prove}^{\varphi}(\mathcal{C}, \theta, \mu_1, c_1, \ldots, \mu_m, c_m)]\!\} \Rightarrow \forall \theta_1 I_{\mu_1}, \ldots, \forall \theta_m I_{\mu_m}, \forall \widetilde{x} \in \widetilde{T}, \neg \bigwedge \mathcal{F}(\mathcal{C}, \theta, \mu_1, c_1, \ldots, \mu_m, c_m)$. So $\mathit{Tr} \vdash \forall \theta_1 I_{\mu_1}, \ldots, \forall \theta_m I_{\mu_m}, \forall \widetilde{x} \in \widetilde{T}, \neg \bigwedge \mathcal{F}(\mathcal{C}, \theta, \mu_1, c_1, \ldots, \mu_m, c_m)$.

Then $\mathit{Tr}, \rho_1 \vdash \neg \bigwedge \mathcal{F}(\mathcal{C}, \theta, \mu_1, c_1, \ldots, \mu_m, c_m)$, so $\mathit{Tr}, \rho_1 \vdash \mathrm{formula}(\mathrm{prove}^{\varphi}(\mathcal{C}, \theta, \mu_1, c_1, \ldots, \mu_m, c_m))$, that is, $\mathit{Tr}, \rho_1 \vdash \mathrm{formula}(\mathcal{F} \models_{\theta}^{\mathcal{I}, \mathcal{V}, \mathcal{C}} \phi)$, with $\mathcal{F} = \theta_1 \mathcal{F}_{F_1, \mu_1, c_1} \cup \cdots \cup \theta_m \mathcal{F}_{F_m, \mu_m, c_m}$, $\mathcal{I} = \{j \mapsto (\mu_j, \theta_j I_{\mu_j}) \mid F_j$ is an injective event$\}$, $\mathcal{V} = \mathrm{var}(\theta_1 I_{\mu_1}) \cup \cdots \cup \mathrm{var}(\theta_m I_{\mu_m}) \cup \{\widetilde{x}\}$, and $\theta = \theta(\mu_1, c_1, \ldots, \mu_m, c_m)$.

Consider an injective event in ϕ, associated to function f_l.

- If that injective event corresponds to

$$\bigvee_{(\mathcal{F}, N_0, \mathcal{I}, \mathcal{V}) \in \mathcal{S}} \exists \tau \in \mathbb{N}, N_0 : \theta\mathsf{event}(e(M_1, \ldots, M_m))@\tau$$

  in $\mathrm{formula}(\mathcal{F} \models_{\theta}^{\mathcal{I}, \mathcal{V}, \mathcal{C}} \phi)$, we have

$$\mathit{Tr}, \rho_1 \vdash \bigvee_{(\mathcal{F}, N_0, \mathcal{I}, \mathcal{V}) \in \mathcal{S}} \exists \tau \in \mathbb{N}, N_0 : \theta\mathsf{event}(e(M_1, \ldots, M_m))@\tau$$

  since $\mathrm{formula}(\mathcal{F} \models_{\theta}^{\mathcal{I}, \mathcal{V}, \mathcal{C}} \phi)$ is a conjunction. So $\mathit{Tr}, \rho_1[\tau \mapsto a] \vdash N_0 : \theta\mathsf{event}(e(M_1, \ldots, M_m))@\tau$ for some $a \in \mathbb{N}$ and some N_0 such that $(\mathcal{F}, N_0, \mathcal{I}, \mathcal{V}) \in \mathcal{S}$. We define $f_l(\rho(\tau_1), \ldots, \rho(\tau_m), \rho(\widetilde{x})) = a$, so that $\mathit{Tr}, \rho_1 \vdash N_0 : \theta\mathsf{event}(e(M_1, \ldots, M_m))@f_l(\tau_1, \ldots, \tau_m, \widetilde{x})$.

  Moreover, if $j \in I$, then F_j is an injective event, $F_j = \mathsf{inj\text{-}event}(e_j(M_{j,1}, \ldots, M_{j,m}))$. Moreover, $\mathit{Tr}, \rho \vdash \psi^{\tau}$, so $\mathit{Tr}, \rho \vdash F_j^{\tau}$, so $\mathit{Tr}, \rho \vdash \mathsf{event}(e_j(M_{j,1}, \ldots, M_{j,m}))@\tau_j$. Since $\mu\mathcal{E}v(\rho(\tau_j)) = (pp_j, \sigma_j(\theta_j I_{\mu_j}) : \cdots = (\mu_j, \rho_1(\theta_j I_{\mu_j})) : \ldots$ and $\mathcal{I}(j) = (\mu_j, \theta_j I_{\mu_j})$, we have $\mathit{Tr}, \rho_1 \vdash \mathcal{I}(j) : \mathsf{event}(e_j(M_{j,1}, \ldots, M_{j,m}))@\tau_j$.

- If that injective event is in a removed disjunct in formula($\mathcal{F} \models\!\!\Rightarrow_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi$), then we define $f_l(\rho(\tau_1), \dots, \rho(\tau_m), \rho(\widetilde{x})) = \bot$.

Then we have $Tr, \rho_1 \vdash \theta\phi^\tau$, so $Tr, \rho_1 \vdash \exists \widetilde{y} \in \widetilde{T}', \phi^\tau$.

Hence, applying this construction for all ρ, we obtain $Tr \vdash \forall \tau_1, \dots, \tau_m \in \mathbb{N}, \forall \widetilde{x} \in \widetilde{T}, (\psi^\tau \Rightarrow \exists \widetilde{y} \in \widetilde{T}', \phi^\tau)$. It remains to show $\mathrm{Inj}(I, f_l)$ for each $l \in \{1, \dots, k\}$.

Suppose $f_l(a_1, \dots, a_m, \widetilde{a}) = f_l(a'_1, \dots, a'_m, \widetilde{a}') \neq \bot$. Let $\rho = \{\tau_1 \mapsto a_1, \dots, \tau_m \mapsto a_m, \widetilde{x} \mapsto \widetilde{a}\}$. Let $\mathcal{S}$ be the leaf of $\mathcal{C}$ corresponding to the event associated to f_l. By the construction above, we have θ, $(\mathcal{F}, N_0, \mathcal{I}, \mathcal{V}) \in \mathcal{S}$, and an extension ρ_1 of ρ such that

$$Tr, \rho_1 \vdash N_0 : \theta\mathsf{event}(e(M_1, \dots, M_m))@f_l(\tau_1, \dots, \tau_m, \widetilde{x}) \tag{17}$$

$$Tr, \rho_1 \vdash \mathcal{F} \tag{18}$$

$$\text{for } j \in I,\ Tr, \rho_1 \vdash \mathcal{I}(j) : \mathsf{event}(e_j(M_{j,1}, \dots, M_{j,m}))@\tau_j \tag{19}$$

Let $\rho' = \{\tau_1 \mapsto a'_1, \dots, \tau_m \mapsto a'_m, \widetilde{x} \mapsto \widetilde{a'}\}$. In the same way, we have θ', $(\mathcal{F}', N'_0, \mathcal{I}', \mathcal{V}') \in \mathcal{S}$, and an extension ρ'_1 of ρ' such that

$$\begin{aligned} &Tr, \rho'_1 \vdash N'_0 : \theta'\mathsf{event}(e(M_1, \dots, M_m))@f_l(\tau_1, \dots, \tau_m, \widetilde{x}) \\ &Tr, \rho'_1 \vdash \mathcal{F}' \\ &\text{for } j \in I,\ Tr, \rho'_1 \vdash \mathcal{I}'(j) : \mathsf{event}(e_j(M_{j,1}, \dots, M_{j,m}))@\tau_j\,. \end{aligned}$$

Let θ'' be a renaming of the domain of ρ'_1 to fresh variables. We have

$$Tr, \rho'_1\theta''^{-1} \vdash \theta'' N'_0 : \theta''\theta'\mathsf{event}(e(M_1, \dots, M_m))@f_l(\tau_1, \dots, \tau_m, \widetilde{x}) \tag{20}$$

$$Tr, \rho'_1\theta''^{-1} \vdash \theta''\mathcal{F}' \tag{21}$$

$$\text{for } j \in I,\ Tr, \rho'_1\theta''^{-1} \vdash \theta''\mathsf{event}(e_j(\mathcal{I}'(j), M_{j,1}, \dots, M_{j,m}))@\tau_j\,. \tag{22}$$

Therefore, by (17) and (20),

$$\begin{aligned} &Tr, \rho_1 \cup \rho'_1\theta''^{-1} \vdash \theta N_0 : \mathsf{event}(e(M_1, \dots, M_m))@f_l(a_1, \dots, a_m, \widetilde{a}) \\ &Tr, \rho_1 \cup \rho'_1\theta''^{-1} \vdash \theta'' N'_0 : \theta''\theta'\mathsf{event}(e(M_1, \dots, M_m))@f_l(a'_1, \dots, a'_m, \widetilde{a'})\,. \end{aligned}$$

Since $f_l(a_1, \dots, a_m, \widetilde{a}) = f_l(a'_1, \dots, a'_m, \widetilde{a'})$, the events are the same, so

$$Tr, \rho_1 \cup \rho'_1\theta''^{-1} \vdash N_0 = \theta'' N'_0\,. \tag{23}$$

We also have by (18) and (21),

$$Tr, \rho_1 \cup \rho'_1\theta''^{-1} \vdash \mathcal{F} \cup \theta''\mathcal{F}'\,. \tag{24}$$

Since $Tr \vdash \{\!\![\vdash \mathcal{C}]\!\!\}$, we have

$$Tr, \rho_1 \cup \rho'_1\theta''^{-1} \vdash \neg\left(\bigwedge \mathcal{F} \wedge \bigwedge \theta''\mathcal{F}' \wedge \left(\bigvee_{j \in \mathrm{Dom}(\mathcal{I})} \mathcal{I}(j) \neq \theta''\mathcal{I}'(j)\right) \wedge N_0 = \theta'' N'_0\right)$$

so using (24) and (23), we conclude that

$$Tr, \rho_1 \cup \rho'_1\theta''^{-1} \vdash \bigwedge_{j \in I} \mathcal{I}(j) = \theta''\mathcal{I}'(j)$$

since $\mathrm{Dom}(\mathcal{I}) = I$. Let $\mu\mathcal{E}v$ be the sequence of events at the end of Tr. For $j \in I$, let $b_j = \rho_1(\mathcal{I}(j)) = \rho'_1(\mathcal{I}'(j))$. By (19) and (22), we have $\mu\mathcal{E}v(a_j) = b_j : e_j(\dots)$ and $\mu\mathcal{E}v(a'_j) = b_j : e_j(\dots)$. By Lemma 40, we have $a_j = a'_j$. That proves $\mathrm{Inj}(I, f_l)$, and concludes the proof that $Tr \vdash \varphi$. $\square$

Proof of Proposition 3 Let C be an evaluation context acceptable for Q_0 with public variables V that does not contain events used by φ. We have

$$\begin{aligned}\mathsf{Adv}_{Q_0}(C, \varphi, D_{\mathrm{false}}) &= \Pr[C[Q_0] : \neg\varphi \wedge \neg\mathsf{NonUnique}_{Q_0}] \\ &\leq \Pr[C[Q_0] : \neg\{\![\mathrm{prove}^{\varphi}(\mathcal{C}, \theta, \mathcal{S})]\!\}] && \text{by Lemma 41} \\ &\leq \Pr[C[Q_0] \preceq \neg\{\![\mathrm{prove}^{\varphi}(\mathcal{C}, \theta, \mathcal{S})]\!\}] && \text{by Lemma 1} \\ &\leq p(C)\end{aligned}$$

So $\mathsf{Bound}_{Q_0}(V, \varphi, D_{\mathrm{false}}, p)$. □

Example 7 Let us prove that the example G_1 satisfies (6). We prove the correspondence $(\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T}'; \phi) = (\forall x : \mathit{pkey}, y : \mathit{host}, z : \mathit{nonce}; \mathsf{inj\text{-}event}(e_B(x, y, z)) \Rightarrow \mathsf{inj\text{-}event}(e_A(x, y, z)))$. The program point μ_B executes $F = \mathsf{event}(e_B(x, y, z))$ and $\mathcal{F} = \theta'\mathcal{F}_{F,\mu_B} \supseteq \theta'\mathcal{F}_{\mu_B}\{i'_B/i_B\} \cup \{x_{pk_A}[i'_B] = x, B = y, N[i'_B] = z\}$ with $\theta' = \{i'_B/i_B\}$.

As in the proof of $\mathcal{F} \models\!\!\Rightarrow_{\theta(\mu_B)} \mathsf{event}(e_A(x, y, z))$ in Example 6, we show $\mathcal{F} \models\!\!\Rightarrow^{\mathcal{I},\mathcal{V},\mathcal{C}}_{\theta(\mu_B)} \mathsf{event}(e_A(x, y, z))$ where $\mathcal{I} = \{1 \mapsto (\mu_B, i'_B)\}$ encodes the program points and replication indices of the events of ψ, $\mathcal{V} = \{i'_B, x, y, z\}$ contains the replication indices of $\mathcal{F}$ and the variables of ψ, $\mathcal{C} = \mathcal{S} = \{(\mathcal{F}, (\mu_A, u[i'_B]), \mathcal{I}, \mathcal{V})\}$. ($\mathcal{C} = \mathcal{S}$ because the formula ψ is reduced to a single event; $M'_0 = (\mu_A, u[i'_B])$ contains the program point and replication indices of the event e_A contained in $\mathcal{F}$: $(\mu_A, u[i'_B]) : \mathsf{event}(e_A(pk_A, x_B[u[i'_B]], x_N[u[i'_B]])) \in \mathcal{F}$.)

In order to prove injectivity, it remains to show that $\vdash \mathcal{C}$. Let $\theta'' = \{i''_B/i'_B, x''/x, y''/y, z''/z\}$. We need to show that $\mathcal{F} \cup \theta''\mathcal{F} \cup \{(\mu_B, i'_B) \neq (\mu_B, i''_B), (\mu_A, u[i'_B]) = (\mu_A, u[i''_B])\}$ yields a contradiction, that is, if the pairs (program point, replication indices) of the event e_B in ψ are distinct ($(\mu_B, i'_B) \neq (\mu_B, i''_B)$), then the pairs (program point, replication indices) of the event e_A in ϕ are also distinct ($(\mu_A, u[i'_B]) \neq (\mu_A, u[i''_B])$).

$\mathcal{F}$ contains $N[i'_B] = x_N[u[i'_B]]$, so $\theta''\mathcal{F}$ contains $N[i''_B] = x_N[u[i''_B]]$. These two equalities combined with $u[i'_B] = u[i''_B]$ imply that $N[i'_B] = x_N[u[i'_B]] = x_N[u[i''_B]] = N[i''_B]$. Since N is defined by random choices of the large type *nonce*, $N[i'_B] = N[i''_B]$ implies $i'_B = i''_B$ up to probability $n^2/2|\mathit{nonce}|$, by eliminating collisions. This equality contradicts $(\mu_B, i'_B) \neq (\mu_B, i''_B)$, so we obtain the desired injectivity. Therefore, the game G_1 satisfies (6) with any public variables V up to probability $n^2/2|\mathit{nonce}|$.

5 Game Transformations

5.1 Syntactic Game Transformations

5.1.1 auto_SArename

The transformation **auto_SArename** renames all variables defined in conditions of find, so that they have distinct names. This transformation is a particular case of **SArename** (Section 5.1.7) that is particularly simple because these variables do not have array accesses by Invariant 3.

Lemma 42 *The transformation **auto_SArename** requires and preserves Properties 1, 2, 3, and 4. It preserves Property 6. If transformation **auto_SArename** transforms G into G′, then* $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : G, D, \mathit{EvUsed} \xrightarrow{V}_0 G', D, \mathit{EvUsed}$ *and* G' *satisfies Property 5.*

5.1.2 expand_tables [29]

The transformation **expand_tables** transforms insert and get into find, since the other transformations do not support tables. It proceeds by storing the inserted list elements in fresh array variables, and looking up in these arrays instead of performing get. More precisely, when $\mathsf{insert}\ \mathit{Tbl}(M_1, \dots, M_k); P$ is under the replications $!^{i_1 \leq n_1} \dots !^{i_l \leq n_l}$, it is transformed into

$$\mathsf{let}\ y_1[i_1, \dots, i_l] = M_1\ \mathsf{in}\ \dots \mathsf{let}\ y_k[i_1, \dots, i_l] = M_k\ \mathsf{in}\ P$$

where $y_1, \dots, y_k$ are fresh array variables, and we add $(y_1, \dots, y_k; i_1 \leq n_1, \dots, i_l \leq n_l)$ in a set S', to remember them. The construct $\mathsf{get}[\mathit{unique}?]\ \mathit{Tbl}(x_1 : T_1, \dots, x_k : T_k)\ \mathsf{suchthat}\ M\ \mathsf{in}\ P\ \mathsf{else}\ P'$ is then transformed into

$$\mathsf{find}[\mathit{unique}?] \left(\bigoplus_{(y_1,\dots,y_k; i_1 \leq n_1, \dots, i_l \leq n_l) \in S'} \begin{array}{l} u_1 = i'_1 \leq n_1, \dots, u_l = i'_l \leq n_l\ \mathsf{suchthat} \\ \mathsf{defined}(y_1[\widetilde{i'}], \dots, y_k[\widetilde{i'}]) \wedge M\{y_1[\widetilde{i'}]/x_1, \dots, y_k[\widetilde{i'}]/x_k\} \\ \mathsf{then}\ \mathsf{let}\ x_1 = y_1[\widetilde{u}]\ \mathsf{in}\ \dots\ \mathsf{let}\ x_k = y_k[\widetilde{u}]\ \mathsf{in}\ P \end{array} \right)$$
$$\mathsf{else}\ P'$$

where $[\mathit{unique}?]$ is either $[\mathsf{unique}_e]$ or empty and has the same value at both occurrences, $\widetilde{u}$ stands for $u_1, \dots, u_l$, and $\widetilde{i'}$ stands for $i'_1, \dots, i'_l$. This construct looks in all arrays used for translating insertion in table Tbl, for indices $\widetilde{i'}$ such that $y_1[\widetilde{i'}], \dots, y_k[\widetilde{i'}]$ are defined, that is, an element has been inserted at indices $\widetilde{i'}$, and $M\{y_1[\widetilde{i'}]/x_1, \dots, y_k[\widetilde{i'}]/x_k\}$ is true, that is, that element satisfies M. When it finds such an element, it stores it in $x_1, \dots x_k$, and runs P. (When it finds several elements, one of them is chosen randomly with uniform probability when $[\mathit{unique}?]$ is empty and the non-unique event e is raised when $[\mathit{unique}?]$ is $[\mathsf{unique}_e]$.) When it finds no element, it executes P'. These transformations are described for processes, but similar transformations are performed for insert and get terms.

After this transformation, **expand_tables** calls **auto_SArename** to guarantee Property 5.

Lemma 43 *The transformation **expand_tables** requires and preserves Properties 1, 2, and 3. It preserves Property 6. If transformation **expand_tables** transforms G into G', then $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : G, D, \mathit{EvUsed} \xrightarrow{V}_p G', D, \mathit{EvUsed}$, where p is ϵ_{find} times the maximal number of executions of* get *in G that is not obviously unique (that is, such that the* insert *in that table may be executed several times), and G' satisfies Properties 4 and 5.*

5.1.3 expand

The transformation **expand** transforms terms new, let, if, find, event, and $\mathsf{event\_abort}$ into processes, so that Property 6 is guaranteed. It simplifies the generated game on the fly, using many of the rules of **simplify** (Section 5.1.21), to avoid generating branches that can actually not be executed.

After this transformation, **expand** calls **auto_SArename** to guarantee Property 5.

Lemma 44 *The transformation **expand** requires and preserves Properties 1, 2, 3, 4, and 5. If transformation **expand** transforms G into G', then $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : G, D, \mathit{EvUsed} \xrightarrow{V}_p G', D, \mathit{EvUsed}$, where p is an upper bound of the probability that the simplification steps modify the execution, and G' satisfies Property 6.*

5.1.4 prove_unique

The transformation **prove_unique** tries to prove that each $\mathsf{find}[\mathsf{unique}_e]$ really has a unique possibility at runtime (up to a small probability), so that event e is executed with at most that probability. More precisely, $\mathsf{find}[\mathsf{unique}]$ are already proved; $\mathsf{find}[\mathsf{unique}_e]$ for which no query $\mathsf{event}(e) \Rightarrow \text{false}$ is active are also considered as already proved (they will be proved elsewhere; with the notations of Definition 6, e does not occur in $D_1 \vee D$, so e occurs in $\mathsf{NonUnique}_{Q,D_1 \vee D}$), and they are replaced with $\mathsf{find}[\mathsf{unique}]$. That corresponds to renaming event e to a special non-unique event that is always in $\mathsf{NonUnique}_{Q,D_1 \vee D}$. It remains to prove $\mathsf{find}[\mathsf{unique}_e]$ for which a query $\mathsf{event}(e) \Rightarrow \text{false}$ is active.

Suppose that $P_0 = \mathsf{find}[\mathsf{unique}_e]\ (\bigoplus_{j=1}^{m} \widetilde{u_j}[\widetilde{i}] = \widetilde{i_j} \leq \widetilde{n_j}$ $\mathsf{suchthat}$ $\mathsf{defined}(M_{j1}, \ldots, M_{jl_j}) \wedge M_j$ then $P_j)$ else P is such a $\mathsf{find}[\mathsf{unique}_e]$. (The same transformation is performed for $\mathsf{find}[\mathsf{unique}_e]$ terms.) CryptoVerif proves uniqueness by proving

- that we obtain a contradiction if the condition of a certain branch holds for two different values of the indices $\widetilde{i_j}$, that is, for all $j \in \{1, \ldots, m\}$, $\mathcal{F}_{P_0} \cup \{\mathsf{defined}(M_{j1}), \ldots, \mathsf{defined}(M_{jl_j}), M_j, \mathsf{defined}(\theta M_{j1}), \ldots, \mathsf{defined}(\theta M_{jl_j}), \theta M_j, \widetilde{i_j} \neq \theta \widetilde{i_j}\}$ yields a contradiction, where the substitution θ maps $\widetilde{i_j}$ to fresh replication indices;
- and that we obtain a contradiction if the conditions of two different branches hold simultaneously, that is, for all $j, j' \in \{1, \ldots, m\}$ with $j < j'$, $\mathcal{F}_{P_0} \cup \{\mathsf{defined}(M_{j1}), \ldots, \mathsf{defined}(M_{jl_j}), M_j, \mathsf{defined}(\theta M_{j'1}), \ldots, \mathsf{defined}(\theta M_{j'l_{j'}}), \theta M_{j'}\}$ yields a contradiction, where the substitution θ maps $\widetilde{i_{j'}}$ to fresh replication indices. (The substitution θ is useful in case the same replication indices are used in both branches j and j'.)

When uniqueness is proved, $\mathsf{find}[\mathsf{unique}_e]$ is replaced with $\mathsf{find}[\mathsf{unique}]$. A subsequent call to **success** (Section 4) will remove the query $\mathsf{event}(e) \Rightarrow \text{false}$ when event e no longer occurs in the game.

Lemma 45 *The transformation **prove_unique** requires and preserves Properties 1, 2, 3, 4, and 5. It preserves Property 6. If transformation **prove_unique** transforms G into G', then $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : G, D, EvUsed \xrightarrow{V}_p G', D, EvUsed$, where p is an upper bound of the probability that some $\mathsf{find}[\mathsf{unique}_e]$ proved in the transformation actually executes event e.*

5.1.5 remove_assign [24]

The transformation **remove_assign** applied to an assignment $\mathsf{let}\ x[i_1, \ldots, i_l] : T = M$ in P replaces x with its value M. (The same transformation is performed for assignment terms.) Precisely, the transformation is performed only when x does not occur in M (non-cyclic assignment) and M contains only variables, function applications, and tests (otherwise, copying the definition of x may break the invariant that each variable is assigned at most once). When x has several distinct definitions, we simply replace $x[i_1, \ldots, i_l]$ 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 or several identical definitions, we replace everywhere in the game $x[M_1, \ldots, M_l]$ with $M\{M_1/i_1, \ldots, M_l/i_l\}$. We additionally update the $\mathsf{defined}$ conditions of find to preserve Invariant 2 and to make sure that, if a condition of find guarantees that $x[M_1, \ldots, M_l]$ is defined in the initial game, then so does the corresponding condition of find in the transformed game. (Essentially, when $y[M'_1, \ldots, M'_{l'}]$ occurs in M, the transformation typically creates new occurrences of $y[M''_1, \ldots, M''_{l'}]$ for some $M''_1, \ldots, M''_{l'}$, so the condition that $y[M''_1, \ldots, M''_{l'}]$ is defined must sometimes be explicitly added to conditions of find in order to preserve Invariant 2.) Moreover, we replace as often as possible defined conditions $x[M_1, \ldots, M_l]$ with defined conditions

$y[M_1, \dots, M_l]$ where y is defined at the same time as x. 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.)

This removal of assignments is applied to all variables whose value is not used (those are used only at the root of defined conditions, or not at all). Depending on the argument of the transformation, it is also applied to other assignments:

- **findcond**: all assignments in conditions of find;
- **useless**: assignments that store a variable or a replication index, when the setting **expandAssignXY** is true; otherwise, no other assignment;
- **binder** $x_1 \dots x_n$: the assignments to variables $x_1, \dots, x_n$.

After this transformation, **remove_assign** calls **auto_SArename** to guarantee Property 5.

With the arguments **findcond** and **useless**, this is repeated as many times as specified by the setting **maxIterRemoveUselessAssign**. Repetition stops if a fixpoint is reached.

Lemma 46 *The transformation **remove_assign** requires and preserves Properties 1, 2, 3, 4, and 5. It preserves Property 6. If transformation **remove_assign** transforms G into G', then $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : G, D, EvUsed \xrightarrow{V}_0 G', D, EvUsed$.*

Example 8 In the game G_0 of Section 4.2.1, the transformation **remove_assign binder** sk_A substitutes $\mathrm{skgen}(rk_A)$ for sk_A in the whole process and removes the assignment let $sk_A = \mathrm{skgen}(rk_A)$. After this substitution, $\mathrm{sign}(\mathrm{concat}(pk_A, x_B, x_N), sk_A, r)$ becomes $\mathrm{sign}(\mathrm{concat}(pk_A, x_B, x_N), \mathrm{skgen}(rk_A), r)$ thus exhibiting a term required to apply the security assumption on signatures in the cryptographic transformation of Section 5.2.

5.1.6 use_variable

The transformation **use_variable** $x_1 \dots x_m$ tries to use variables $x_1, \dots, x_n$ instead of recomputing their value. More precisely, at each program point that corresponds to a simple term M and where $x_j[\widetilde{M}]$ is guaranteed to be defined (because x_j is defined above that program point or directly or indirectly because of defined conditions above that program point), if all definitions of x_j that can be executed before reaching that program point are let $x_j[\widetilde{i}] = M_0$ in, then we test whether M is equal to $M_0\{\widetilde{M}/\widetilde{i}\}$ modulo the built-in equations, and if yes, we replace M with $x_j[\widetilde{M}]$.

The defined conditions of find above the modified program points are updated to make sure that Invariant 2 is preserved. This is needed in particular to make sure that $x_j[\widetilde{M}]$ syntactically occurs in the defined conditions when it is used.

Lemma 47 *The transformation **use_variable** requires and preserves Properties 1, 2, 3, 4, and 5. It preserves Property 6. If transformation **use_variable** transforms G into G', then $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : G, D, EvUsed \xrightarrow{V}_0 G', D, EvUsed$.*

The transformation **use_variable** is a convenient way to perform common subexpression elimination, possibly by first inserting the definition of the desired variable(s) by **insert** (Section 5.1.13). This transformation could be done by several applications of the transformation **replace** (Section 5.1.14). However, **use_variable** is easier to use when it performs the desired replacement. For performance reasons, the equality tests performed by **use_variable** are considerably less powerful than those performed by **replace**, so if **use_variable** does not replace a term with $x_j[\widetilde{M}]$ at some occurrence, it is worth trying **replace**.

5.1.7 SArename [24]

The transformation **SArename** x (single assignment rename) aims at renaming x so that distinct definitions of x have different names; 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 $FB = \widetilde{u}[\widetilde{i}] = \widetilde{i'} \leq \widetilde{n}$ suchthat defined$(M'_1, \dots, M'_{l'}) \wedge M$ then $\dots$ 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 $1 \leq j \leq m$.

Moreover, the implementation takes into account that some variables cannot be simultaneously defined, to reduce the number of branches of find to generate.

After this transformation, **SArename** calls **auto_SArename** to guarantee Property 5.

As a particular case, **SArename random** performs the following transformation: when y is defined by new $y : T$ and has $m > 1$ definitions and all variable accesses to y are of the form $y[i_1, \dots, i_l]$ under a definition of $y[i_1, \dots, i_l]$, where $i_1, \dots, i_l$ are the current replication indices at this definition of y (that is, y has no array access using find), it renames y to $y_1, \dots, y_m$ with a different name for each definition of y by new $y : T$.

Lemma 48 *The transformation **SArename** requires and preserves Properties 1, 2, 3, 4, and 5. It preserves Property 6. If transformation **SArename** transforms G into G', then $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : G, D, EvUsed \xrightarrow{V}_p G', D, EvUsed$, where p is ϵ_{find} times the maximal number of executions of a modified* find *that is not proved unique.*

Example 9 Consider the following process

$$start(); \mathsf{new}\ k_A : T_k; \mathsf{new}\ k_B : T_k; \overline{yield}\langle\rangle; (Q_K \mid Q_S)$$

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

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 $k' = k$, by renaming the three definitions of k' to k'_1, k'_2, and k'_3 respectively. After transformation, we obtain the following processes:

$$Q'_K = !^{i \leq n} c[i](h : T_h, k : T_k)$$
$$\quad \mathsf{if}\ h = A\ \mathsf{then\ let}\ k'_1 : T_k = k_A\ \mathsf{in}\ \overline{yield}\langle\rangle\ \mathsf{else}$$
$$\quad \mathsf{if}\ h = B\ \mathsf{then\ let}\ k'_2 : T_k = k_B\ \mathsf{in}\ \overline{yield}\langle\rangle\ \mathsf{else}$$
$$\quad \mathsf{let}\ k'_3 : T_k = k\ \mathsf{in}\ \overline{yield}\langle\rangle$$

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

The find in Q_S, which looks for elements in array k', is transformed in Q'_S into a find with three branches, one for each new name of k' (k'_1, k'_2, and k'_3 respectively). After the simplification (Section 5.1.21), Q'_S becomes:

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

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

5.1.8 move [24]

The transformation **move** moves random choices and assignments downwards in the code as much as possible. A random choice 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 $c[M_1, \dots, M_l](x_1[\widetilde{i}] : T_1, \dots, x_k[\widetilde{i}] : T_k)$, output $\overline{c[M_1, \dots, M_l]}\langle N_1, \dots, N_k\rangle$ when x occurs in $M, M_1, \dots, M_l, N_1, \dots, N_k$, 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 random choice 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 random choice 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 considered beneficial when it satisfies the following conditions:

- for random choices, when the random choice 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, 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.

The performed moves are determined by the argument of the transformation:

- **all**: moves all random choices and assignments, provided the move is beneficial.
- **noarrayref**: moves all random choices and assignments that do not have array references, provided the move is beneficial.
- **random**: moves all random choices, provided the move is beneficial.

- **random_noarrayref**: moves all random choices that do not have array references, provided the move is beneficial.
- **assign**: moves all assignments, provided the move is beneficial.
- **binder** $x_1 \dots x_n$: move the variables $x_1, \dots, x_n$ (even when the move is not beneficial).

In all cases, only random choices and assignments at the process level (not inside terms) are moved.

Lemma 49 *The transformation **move** requires and preserves Properties 1, 2, 3, 4, and 5. It preserves Property 6. If transformation **move** transforms G into G', then $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : G, D, EvUsed \xrightarrow{V}_0 G', D, EvUsed$.*

5.1.9 move array [25]

The transformation **move array** X delays the generation of a random value X until the point at which it is first used (lazy sampling). This transformation is implemented as a particular case of a cryptographic transformation by the following equivalence:

```
$!^{i \leq n}$new $X : T$;
  $(!^{iX \leq nX}\mathrm{OX}() :=$ return$(X)$ |
   $!^{ieq \leq neq}\mathrm{Oeq}(X' : T) :=$ return$(X' = X))$
$\approx_{\#\mathrm{Oeq}/|T|}$  $[manual]$
$!^{i \leq n}$
  $(!^{iX \leq nX}\mathrm{OX}() :=$
    find[unique] $j \leq nX$ suchthat defined$(Y[j])$
    then return$(Y[j])$ else new $Y : T$; return$(Y)$ |
   $!^{ieq \leq neq}\mathrm{Oeq}(X' : T) :=$
    find[unique] $j \leq nX$ suchthat defined$(Y[j])$
    then return$(X' = Y[j])$ else return(false))
```

where T is the type of X. Two oracles are defined, OX and Oeq. In the left-hand side, OX returns the random X itself. In the right-hand side, OX uses a lookup to test if the random value was already generated; if yes, it returns the previously generated random value $Y[j]$; if no, it generates a fresh random value Y. Transforming the left-hand side into the right-hand side therefore moves the generation of the random number X to the first call to OX, that is, the first usage of X. The oracle Oeq provides an optimized treatment of equality tests $X' = X$: when the random value X was not already generated, we return false instead of generating a fresh X, so we exclude the case that X' is equal to a fresh X. This case has probability $1/|T|$ for each call to Oeq, so the probability of distinguishing the two games is $\#\mathrm{Oeq}/|T|$. (Notice that there never exist several choices of j that satisfy the conditions of the finds in the right-hand side of this equivalence, so these finds can be marked [unique] without modifying their behavior.)

5.1.10 move up

The transformation **move up** $x_1 \dots x_n$ **to** μ moves the random number generations or assignments of $x_1, \dots, x_n$ upwards in the syntax tree, to the program point μ. This program point must correspond to an output process.

The program point μ is an integer, which can be determined using the command **show_game occ**: this command displays the current game with the corresponding label $\{\mu\}$ at each program point. The command **show_game occ** also allows one to inspect the game, for instance to know the names of fresh variables created by CryptoVerif during previous transformations. Program points and variable names may depend on the version of CryptoVerif. Since CryptoVerif version 2.01, program points can also be designated by expressions like **before** *regexp*, which designates the program point at the beginning of the line that matches the regular expression *regexp*; **after** *regexp*, which designates the program point just after the line that matches *regexp*; **before_nth** *n regexp*, which designates the program point at the beginning of the n-th line that matches the regular expression *regexp*; **after_nth** *n regexp*, which designates the program point at the beginning of the first line that has an occurrence number after the n-th line that matches the regular expression *regexp*; **at** n' *regexp*, which designates the program point at the n'-th occurrence number that occurs inside the string that matches the regular expression *regexp* in the displayed game; **at_nth** n n' *regexp*, which designates the program point at the n'-th occurrence number that occurs inside the string corresponding to the n-th match of the regular expression *regexp* in the displayed game. This way of designating program points is more stable across versions of CryptoVerif.

After the game transformation, a variable x is defined at program point μ, and all other variables x_k are defined by $\mathsf{let}\ x_k = x\ \mathsf{in}$. The variable x is a variable x_k itself when x_k has no array accesses and the current replication indices at the definition of x_k are the same as at μ. Otherwise, the variable x is a fresh variable.

All variables $x_1, \dots, x_n$ must have the same type. They must not be defined syntactically above the program point μ. The definitions of the variables $x_1, \dots, x_n$ must be in distinct branches of if, find, let, so that they cannot be simultaneously defined. Either all variables $x_1, \dots, x_n$ must be defined by random number generations or all of them must be defined by assignments.

- If $x_1, \dots, x_n$ are defined by random number generations, this transformation performs eager sampling of x_i. The random number generation of $x_1, \dots x_n$ must be executed at most once for each execution of program point μ. This is proved by combining that the definitions of the variables $x_1, \dots, x_n$ are in distinct branches of if, find, let with the fact that each of these definitions (at μ_j) is executed at most once for each value of the current replication indices at μ. To show the latter fact, we notice that, since μ_j is syntactically under μ, the current replication indices at μ are a prefix of the replication indices at μ_j. If the replication indices at μ_j are the same as at μ, then the fact is proved. Otherwise, the replication indices at μ_j are $\widetilde{i}, \widetilde{i}_j$ while the replication indices at μ are $\widetilde{i}$ and we show that $\mathcal{F}_{\mu_j} \cup \mathcal{F}_{\mu_j}\{\widetilde{i}'_j/\widetilde{i}_j\} \cup \{\widetilde{i}_j \neq \widetilde{i}'_j\}$ yields a contradiction, where $\widetilde{i}'_j$ are fresh replication indices.

- If $x_1, \dots, x_n$ are defined by assignments of terms M_j, then all M_j must consist of variables, function applications, and tests; there must be one M_j defined at program point μ, using all defined variables collected in $\mathcal{F}_\mu$ (let M_{j_0} be that M_j, which will be used as the definition of x: $\mathsf{let}\ x = M_{j_0}\ \mathsf{in}$); and all terms M_j must be equal: for all $j \neq j_0$, $M_j = M_{j_0}$ knowing the facts $\mathcal{F}_{\mu_j}$ that hold at the program point μ_j of M_j.

  The $\mathsf{defined}$ conditions of find above μ are updated to syntactically guarantee the definition of M_{j_0}, as required by Invariant 2.

Lemma 50 *The transformation **move up** requires and preserves Properties 1, 2, 3, 4, and 5. It preserves Property 6. If transformation **move up** transforms G into G', then $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : G, D, EvUsed \xrightarrow{V}_p G', D, EvUsed$, where p is an upper bound of the probability of collisions eliminated in the proof that each μ_j is executed at most once for each execution of μ or that for all $j \neq j_0$, $M_j = M_{j_0}$.*

5.1.11 move_if_fun

The transformation **move_if_fun** moves the predefined function if_fun or transforms it into a term if ... then ... else... It supports the following variants:

- **move_if_fun** $loc_1 \dots loc_n$, where each loc_j is either a program point or a function symbol. When loc_j is a program point, it moves occurrences of if_fun from inside the term at that program point to the root of that term. (The program point μ is designated as explained in Section 5.1.10.) When loc_j is a function symbol, it moves occurrences of if_fun from under that function symbol to just above it. The move corresponds to rewriting $C[\text{if\_fun}(M_1, M_2, M_3)]$ into $\text{if\_fun}(M_1, C[M_2], C[M_3])$, where C is a term context built from the following grammar:

$C ::=$	simple term context
$[\,]$	hole
$x[M_1, \dots, M_{k-1}, C, M_{k+1}, \dots, M_m]$	variable
$f(M_1, \dots, M_{k-1}, C, M_{k+1}, \dots, M_m)$	function application

  and the root of C corresponds to a loc_j. (It is at program point loc_j when loc_j is a program point; its root symbol is loc_j when loc_j is a function symbol.) These moves are possible only when C is a simple term context, since otherwise they might lead to defining several times the same variable or repeating events since the context C is duplicated in the second and third arguments of if_fun and if_fun evaluates all its arguments. For simplicity, we allow them only when $C[\text{if\_fun}(M_1, M_2, M_3)]$ is a simple term.

- **move_if_fun level** n, where n is a positive integer, moves occurrences of if_fun n function symbols up in the syntax tree (provided those if_fun occur under at least n function symbols). As above, these moves are allowed only when they occur inside a simple term.

- **move_if_fun to_term** $\mu_1 \dots \mu_n$ transforms terms $\text{if\_fun}(M_1, M_2, M_3)$ that occur at program points $\mu_1, \dots, \mu_n$ into terms if M_1 then M_2 else M_3. When no program point is given, it performs that transformation everywhere in the game.

  When M_2 and M_3 have a visible effect, that is, they define some variable with array accesses (including by their usage in various kinds of secrecy queries) or they execute events, the transformation above would not be correct, because $\text{if\_fun}(M_1, M_2, M_3)$ evaluates both M_2 and M_3 while if M_1 then M_2 else M_3 evaluates either M_2 or M_3. In this case, we transform $\text{if\_fun}(M_1, M_2, M_3)$ into let $xcond = M_1$ in let $xthen = M_2$ in let $xelse = M_3$ in if $xcond$ then $xthen$ else $xelse$ to make sure that M_1, M_2, and M_3 are always evaluated, and in that order.

  When **autoExpand = true** (the default), a call to **expand** is automatically performed after **move_if_fun**, which transforms the terms let ... = ... in ... and if ... then ... else ... into processes.

Lemma 51 *The transformation **move_if_fun** requires and preserves Properties 1, 2, 3, 4, and 5. The variants **move_if_fun** $loc_1 \dots loc_n$ and **move_if_fun level** n preserve Property 6. If transformation **move_if_fun** transforms G into G', then $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : G, D, EvUsed \xrightarrow{V}_0 G', D, EvUsed$.*

5.1.12 insert_event [25]

The transformation **insert_event** e μ inserts $\mathsf{event\_abort}\ e$ at program point μ. (The program point μ is designated as explained in Section 5.1.10.)

The transformation **insert_event** e μ also adds to query $\mathsf{event}(e) \Rightarrow$ false in order to bound the probability of event e.

Lemma 52 *The transformation **insert_event** requires and preserves Properties 1, 2, 3, 4, and 5. It preserves Property 6 if the event is inserted at the process level. If transformation **insert_event** e μ transforms G into G', then $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : G, D, \mathit{EvUsed} \xrightarrow{V}_0 G', D \vee e, \mathit{EvUsed} \cup \{e\}$.*

5.1.13 insert [25]

The transformation **insert** μ *ins* adds instruction *ins* at the program point μ. The program point μ is designated as explained in Section 5.1.10. The instruction *ins* can for instance be a test, in which case all branches of the test will be copies of the code that follows program point μ (so that the semantics of the game is unchanged). It can also be an assignment or a random generation of a fresh variable or an $\mathsf{event\_abort}$ instruction. In all cases, CryptoVerif checks that this instruction preserves the semantics of the game except when we execute an inserted Shoup event, and rejects it with an error message if it does not.

After this transformation, **insert** calls **auto_SArename** to guarantee Property 5.

When the user inserts $\mathsf{event\_abort}\ e$, the transformation **insert** adds a query $\mathsf{event}(e) \Rightarrow$ false in order to bound the probability of event e.

When the user inserts a $\mathsf{find}[\mathsf{unique}_e]$ (the user actually types $\mathsf{find}[\mathsf{unique}]$ but CryptoVerif automatically generates a fresh event e and inserts $\mathsf{find}[\mathsf{unique}_e]$ instead, since uniqueness is not proved yet), the transformation **insert** adds a query $\mathsf{event}(e) \Rightarrow$ false and calls **prove_unique** (Section 5.1.4) in order to try proving uniqueness.

Lemma 53 *The transformation **insert** requires and preserves Properties 1, 2, 3, 4, 5, and 6. If transformation **insert** transforms G into G', then $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : G, D, \mathit{EvUsed} \xrightarrow{V}_p G', D \vee e_1 \vee \dots \vee e_m, \mathit{EvUsed} \cup \{e_1, \dots, e_m\}$ where $e_1, \dots, e_m$ are the events in the inserted instruction ($\mathsf{event\_abort}\ e_j$, $\mathsf{find}[\mathsf{unique}_{e_j}]$) and the probability p comes from **prove_unique**.*

5.1.14 replace

The transformation **replace** μ M replaces the term M_0 at program point μ with the term M. (M_0 and M must be simple. The program point μ is designated as explained in Section 5.1.10.) Before performing the replacement, it checks that M is equal to M_0 at that program point (up to a small probability): first, it collects all facts $\mathcal{F}_\mu$ that hold at program point μ; second, it tests equality between M_0 and M using $\mathcal{F}_\mu$ and built-in equations (it uses equalities inferred from $\mathcal{F}_\mu$ to replace variables with their values, trying to make the terms equal); third, it simplifies M_0 and M using user-defined rewrite rules of Section 3.1, and tests equality between the results using $\mathcal{F}_\mu$ and built-in equations; fourth, it rewrites M_0 and M at most **maxReplaceDepth** times using equalities inferred from $\mathcal{F}_\mu$ and user-defined rewrite rules of Section 3.1, until it finds a common term modulo the built-in equations. The transformation is performed as soon as the equality between M_0 and M is proved.

The $\mathsf{defined}$ conditions of find above the program point μ are updated to make sure that Invariant 2 is preserved. This is needed in particular when M makes array accesses that M_0 does not make.

Lemma 54 *The transformation **replace** requires and preserves Properties 1, 2, 3, 4, and 5. It preserves Property 6. If transformation **replace** μ M transforms G into G', then $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : G, D, EvUsed \xrightarrow{V}_p G', D, EvUsed$, where p is an upper bound of the probability that M is different from M_0 at μ.*

The variant **assume replace** μ M performs the same replacement, without checking the equality between M_0 and M. This transformation is obviously not sound, but can be used to experiment with modifications in the games. As soon as this transformation is used, CryptoVerif does not claim that any property is proved.

5.1.15 merge_branches [25]

The transformation **merge_branches** performs the following transformations:

1. If some then branches of a find[unique] execute the same code as the else branch (up to renaming of variables defined in these branches and that do not have array accesses, and up to equality of terms proved using facts $\mathcal{F}_\mu$ that hold at the program point μ of the considered find[unique]), the index variables bound in these then branches have no array accesses, and the conditions of these then branches do not contain event_abort nor unproved find[unique$_e$], then we remove these then branches.

   Indeed, these then branches have the same effect as the else branch. The hypotheses are needed for the following reasons:

   - The renamed variables must not have array accesses because renaming variables that have array accesses requires transforming these array accesses. The transformation **merge_arrays** presented in Section 5.1.16 can rename variables with array accesses.
   - The index variables bound in the removed branches must not have array accesses, because removing the definitions of these variables would modify the behavior of the array accesses.
   - The conditions must not contain event_abort nor unproved find[unique$_e$], because if they do, the find may abort while code after transformation would not abort.

2. If all branches of if, let with pattern matching, or find execute the same code (up to renaming of variables defined in these branches and that do not have array accesses, and up to equality of terms proved using facts $\mathcal{F}_\mu$ that hold at the program point μ of the considered if, let, or find), and in case of find, it is not marked [unique$_e$], the index variables bound in the then branches have no array accesses, and the conditions of the then branches do not contain event_abort nor unproved find[unique$_e$], then we replace that if or find with its else branch.

   In this transformation, we ignore the array accesses that occur in the conditions of the find under consideration, since these conditions will disappear after the transformation.

Furthermore, **merge_branches** applies these transformations globally to all finds of the game for which the simplification is possible. As a consequence, one can ignore array accesses to all variables in conditions of find that will be removed, so more transformations are enabled.

Lemma 55 *The transformation **merge_branches** requires and preserves Properties 1, 2, 3, 4, 5, and 6. If transformation **merge_branches** transforms G into G', then $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : G, D, EvUsed \xrightarrow{V}_p G', D, EvUsed$, where p is an upper bound on the probability that equalities between terms of merged branches do not hold.*

5.1.16 merge_arrays [25]

The transformation **merge_arrays** $x_{11}\ \dots\ x_{1n}, \dots, x_{m1}\ \dots\ x_{mn}$ merges the variables $x_{j1}, \dots, x_{jn}$ into a single variable x_{j1} for each $j \leq m$. Each variable x_{jk} must have a single definition. For each $j \leq n$, the variables $x_{j1}, \dots, x_{jn}$ must have the same type and indices of the same type. They must not be defined for the same value of their indices (that is, x_{jk} and $x_{jk'}$ must be defined in different branches of if or find when $k \neq k'$). The arrays $x_{j1}, \dots, x_{jn}$ are merged into a single array x_{j1} for each $j \leq m$. The transformation proceeds as follows:

- If, for each $k \leq n$, x_{1k} is defined above x_{jk} for all $1 < j < m$, we introduce a fresh variable b_k defined by $b_k \leftarrow$ mark just after the definition of x_{1k}. We call b_k a *branch variable*; it is used to detect that x_{jk} has been defined: $x_{jk}[\widetilde{M}]$ is defined before the transformation if and only if $x_{j1}[\widetilde{M}]$ and $b_k[\widetilde{M}]$ are defined after the transformation, and $x_{j1}[\widetilde{M}]$ after the transformation is equal to $x_{jk}[\widetilde{M}]$ before the transformation.

- For each find that requires that some variables x_{jk} are defined, we leave the branches that do not require the definition of x_{jk} unchanged and we try to transform the other branches $FB_l = (\tilde{u}_l = \tilde{i}_l \leq \tilde{n}_l$ suchthat defined$(\widetilde{M}_l) \wedge M_l$ then $P_l)$ as follows.

  1. We require that, for each l, there exists a distinct k such that the defined condition of FB_l refers to x_{jk} for some j but not to $x_{jk'}$ for any other k'. (Otherwise, the transformation fails.) We denote by $l(k)$ the value of l that corresponds to k.
  2. We choose a "target" branch $FB_{\mathrm{T}} = (\tilde{u} = \tilde{i} \leq \tilde{n}$ suchthat defined$(\widetilde{M}) \wedge M$ then $P)$: if the defined condition of some branch FB_l refers to x_{j1} for some j, we choose that branch FB_l. Otherwise, we choose any branch FB_l and rename its variables x_{jk} to x_{j1}. We require that the references $x_{j1}[\widetilde{M}]$ to the variables x_{j1} in the defined condition of the target branch all have the same indices $\widetilde{M}$. If the transformation succeeds, we will replace all branches FB_l with the target branch.
  3. The branch FB_{T} after transformation is equivalent to branches $\bigoplus_{k=1}^{n} FB_{\mathrm{T}}\{x_{jk}/x_{j1}, j = 1, \dots, m\}$ before transformation. We show that these branches are equivalent to the branches FB_l.

     For each $k \leq n$,

     – if $l(k)$ exists, then we show that $FB_{\mathrm{T}}\{x_{jk}/x_{j1}, j = 1, \dots, m\}$ is equivalent to $FB_{l(k)}$. Let $l = l(k)$. We first rename the variables $\tilde{u}_l$ of FB_l to the variables $\tilde{u}$ of the target branch. For simplicity, we still denote by $FB_l = (\tilde{u}_l = \tilde{i}_l \leq \tilde{n}_l$ suchthat defined$(\widetilde{M}_l) \wedge M_l$ then $P_l)$ the obtained branch. Then we show that, if the variables of $\widetilde{M}_l$ are defined, then the variables of $\widetilde{M}\{x_{jk}/x_{j1}, j = 1, \dots, m\}$ are defined, and conversely; $M_l = M\{x_{jk}/x_{j1}, j = 1, \dots, m\}$ (knowing the equalities that hold at that program point), and P_l and $P\{x_{jk}/x_{j1}, j = 1, \dots, m\}$ execute the same code up to renaming of variables defined in P_l or $P\{x_{jk}/x_{j1}, j = 1, \dots, m\}$ and that do not have array accesses, and up to equality of terms proved using facts $\mathcal{F}_\mu$ that hold at the program point μ of the considered find.

     – if $l(k)$ does not exist, then we show that $FB_{\mathrm{T}}\{x_{jk}/x_{j1}, j = 1, \dots, m\}$ can in fact not be executed, because its condition cannot hold: the variables of $\widetilde{M}\{x_{jk}/x_{j1}, j = 1, \dots, m\}$ cannot be simultaneously defined or $M\{x_{jk}/x_{j1}, j = 1, \dots, m\}$ cannot hold.

  If the transformation above fails and we have introduced branch variables, we replace each condition defined$(x_{jk}[\widetilde{M}])$ with defined$(x_{j1}[\widetilde{M}], b_k[\widetilde{M}])$.

If the transformation above fails and we have not introduced branch variables, the whole **merge_arrays** transformation fails.

- The definition of x_{jk} is renamed to x_{j1} and each reference to $x_{jk}[\tilde{M}]$ is renamed to $x_{j1}[\tilde{M}]$.

Lemma 56 *The transformation* ***merge_arrays*** *requires and preserves Properties 1, 2, 3, 4, 5, and 6. If transformation* ***merge_arrays*** *transforms G into G', then $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : G, D, EvUsed \xrightarrow{V}_p G', D, EvUsed$, where p is an upper bound on the probability that required equalities do not hold.*

5.1.17 guess i

When **guessRemoveUnique = true** and some (one-session) secrecy queries are present, the transformation **guess** i first transforms the game G into G_{RU}, by replacing all proved $\mathsf{find}[\mathsf{unique}]$ with find. Lemma 57 shows the soundness of this preliminary transformation. It may be advantageous for (one-session) secrecy proofs because the removed $\mathsf{find}[\mathsf{unique}]$ do not need to be proved, while the remaining ones must be reproved after the transformation, as we show below.

Lemma 57 *Let G_{RU} be the game obtained from G by replacing all proved* $\mathsf{find}[\mathsf{unique}]$ *with* find*. Let sp be a security property (sp is $\mathsf{1\text{-}ses.secr.}(x)$, $\mathsf{Secrecy}(x)$, $\mathsf{bit\ secr.}(x)$, or a correspondence φ, which does not use S, $\overline{\mathsf{S}}$, nor non-unique events). Let D be a disjunction of Shoup events and a subset of non-unique events e_i corresponding to unproved* $\mathsf{find}[\mathsf{unique}_{e_i}]$ *in G (D does not contain S nor $\overline{\mathsf{S}}$). If $\mathsf{Bound}_{G_{\mathrm{RU}}}(V, sp, D, p)$, then $\mathsf{Bound}_G(V, sp, D, p)$.*

Proof Let C' be an evaluation context acceptable for $C_{sp}[G]$ with public variables $V \setminus V_{sp}$ that does not contain events used by sp or D nor non-unique events in G. Let $C = C'[C_{sp}[\,]]$. The context C' is also acceptable for $C_{sp}[G_{\mathrm{RU}}]$ with public variables $V \setminus V_{sp}$, and a fortiori does not contain non-unique events in G_{RU}. Since $\mathsf{Bound}_{G_{\mathrm{RU}}}(V, sp, D, p)$, we have $\mathsf{Adv}_{G_{\mathrm{RU}}}(C, sp, D) \leq p(C)$.

First case: sp is a correspondence φ. We have

$$\begin{aligned}\mathsf{Adv}_G(C, sp, D) &= \Pr[C[G] : (\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G,D}]\\ &\leq \Pr[C[G_{\mathrm{RU}}] : (\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G_{\mathrm{RU}},D}] = \mathsf{Adv}_{G_{\mathrm{RU}}}(C, sp, D)\end{aligned}$$

because traces of G that satisfy $(\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G,D}$ correspond to similar traces of G_{RU} that satisfy $(\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G_{\mathrm{RU}},D}$. Only traces that satisfy e for some proved $\mathsf{find}[\mathsf{unique}_e]$ in G are mapped to different traces in G_{RU}. These traces satisfy $\mathsf{NonUnique}_{G,D}$.

Second case: sp is $\mathsf{1\text{-}ses.secr.}(x)$, $\mathsf{Secrecy}(x)$, or $\mathsf{bit\ secr.}(x)$. We have

$$\Pr[C[G] : \mathsf{S} \vee D] \leq \Pr[C[G_{\mathrm{RU}}] : \mathsf{S} \vee D]$$

since the traces of $C[G]$ that execute an event in $\mathsf{S} \vee D$ correspond to similar traces of $C[G_{\mathrm{RU}}]$ that also execute an event in $\mathsf{S} \vee D$ ($\mathsf{S} \vee D$ does not contain any proved non-unique event of G), and

$$\Pr[C[G_{\mathrm{RU}}] : \overline{\mathsf{S}} \vee \mathsf{NonUnique}_{G_{\mathrm{RU}},D}] \leq \Pr[C[G] : \overline{\mathsf{S}} \vee \mathsf{NonUnique}_{G,D}]$$

since the traces of $C[G_{\mathrm{RU}}]$ that execute $\overline{\mathsf{S}}$ or an event in $\mathsf{NonUnique}_{G_{\mathrm{RU}},D}$ correspond to either to similar traces of $C[G]$ that execute the same event or to traces that execute a proved non-unique event in $C[G]$, so an event in $\mathsf{NonUnique}_{G,D}$. Therefore,

$$\begin{aligned}\mathsf{Adv}_G(C, sp, D) &= \Pr[C[G] : \mathsf{S} \vee D] - \Pr[C[G] : \overline{\mathsf{S}} \vee \mathsf{NonUnique}_{G,D}]\\ &\leq \Pr[C[G_{\mathrm{RU}}] : \mathsf{S} \vee D] - \Pr[C[G_{\mathrm{RU}}] : \overline{\mathsf{S}} \vee \mathsf{NonUnique}_{G_{\mathrm{RU}},D}] = \mathsf{Adv}_{G_{\mathrm{RU}}}(C, sp, D).\end{aligned}$$

In both cases, we obtain $\mathsf{Adv}_G(C, sp, D) \leq p(C)$ and $\mathsf{Bound}_G(V, sp, D, p)$. □

Next, the main guessing transformation is performed. The transformation **guess** i consists in guessing the tested session of a principal in a protocol, which is a step frequently done in cryptographic proofs. In CryptoVerif, we consider a game G and define a transformed game G' by guessing a replication index i: we replace $!^{i\leq n}Q$ with $!^{i\leq n}Q'$ where Q' is obtained from Q by replacing the processes P under the first inputs with $\mathsf{if}\ i = i_{\text{tested}}\ \mathsf{then}\ P'\ \mathsf{else}\ P''$ and i_{tested} is a constant. The constant i_{tested} is the index of the tested session. We distinguish the process executed in the tested session, P', on which we are going to prove security properties, from the process P'' for other sessions which are executed, but for which we do not prove security properties. (In case **diff_constants = true**, the constant i_{tested} must not be considered different from other constants of the same type.) The process P' is obtained from P by

- duplicating all events: $\mathsf{event}\ e(\widetilde{M})$ is replaced with $\mathsf{event}\ e(\widetilde{M}); \mathsf{event}\ e'(\widetilde{M})$ and similarly $\mathsf{event\_abort}\ e$ is replaced with $\mathsf{event}\ e; \mathsf{event\_abort}\ e'$. We require that in the game G, the same event e cannot occur both under the modified replication $!^{i\leq n}Q$ and elsewhere in the game. (Otherwise, queries that use e are left unchanged.)

- duplicating definitions of every variable x used in queries for secrecy and one-session secrecy: $\mathsf{let}\ x' = x\ \mathsf{in}$ is added after each definition of x. We require that in the game G, the same variable x used in queries for secrecy or one-session secrecy cannot be defined both under the modified replication $!^{i\leq n}Q$ and elsewhere in the game. (Otherwise, the considered query is left unchanged.)

The process P'' is obtained from P by duplicating definitions of every variable x used in queries for secrecy (not one-session secrecy): $\mathsf{let}\ x'' = x\ \mathsf{in}$ is added after each definition of x.

We replace variables x in secrecy and one-session secrecy queries with their duplicated version x'. For secrecy queries, the duplicated version x'' is added to public variables. In both cases, we prove (one-session) secrecy for the variable x' defined in the tested session. In case of one-session secrecy, that is enough: it shows that x' is indistinguishable from a random value, and that proves one-session secrecy of x for all sessions by symmetry. However, for secrecy, we additionally want to show that the values of x in the various sessions are independent of each other; this is achieved by considering the value of x in sessions other than the tested session (that is, x'') as public: if x' is indistinguishable from random even when x'' is public, then x' is independent of x''.

In non-injective correspondence queries, we replace *one* non-injective event e before the arrow $\Rightarrow$ with its duplicated version e'. Hence, we prove the query for the tested session, which uses event e'. The proof is valid for all sessions by symmetry.

The probability of attack must basically be multiplied by n for all modified queries. The proof depends on the considered query and is detailed below.

For queries that are left unchanged, i.e. secrecy and one-session secrecy queries for variables not defined under the modified replication, non-injective correspondence queries with no event before the arrow $\Rightarrow$ under the modified replication (the previous queries prove properties about other roles than the one for which we guess the tested session), bit secrecy queries (because the secret is defined under no replication, so it is not under the guessed replication), as well as injective correspondence queries (see details below), the probability is unchanged. It is clear that these queries are not affected by the transformation.

After this transformation, **guess** calls **auto_SArename** to guarantee Property 5.

Lemma 58 *The transformation **guess** i requires and preserves Properties 1, 2, 3, 4, and 5. It preserves Property 6.*

Suppose the game G is transformed into G' by the transformation **guess** *i, where i is a replication index bounded by n. Below, we consider only the modified queries.*

Let φ and φ' be respectively the semantics of a non-injective correspondence and its transformed correspondence. If $\mathsf{Bound}_{G'}(V, \varphi', D_{\mathrm{false}}, p)$ and p is independent of the value of i_{tested}, then $\mathsf{Bound}_G(V, \varphi, D_{\mathrm{false}}, np)$.

If G' satisfies the one-session secrecy of x' with public variables V ($x, x', x'' \notin V$) up to probability p and p is independent of the value of i_{tested}, then G satisfies the one-session secrecy of x with public variables V up to probability np.

If G' satisfies the secrecy of x' with public variables $V \cup \{x''\}$ ($x, x', x'' \notin V$) up to probability p and p satisfies Property 7, then G satisfies the secrecy of x with public variables V up to probability $n \times p$ (neglecting a small additional runtime of the context).

If $\mathsf{Bound}_{G'}(V \cup \{x'\}, \mathsf{1\text{-}ses.secr.}(x'), \mathsf{NonUnique}_{G'}, p)$ and p satisfies Property 7, then we have $\mathsf{Bound}_G(V \cup \{x\}, \mathsf{1\text{-}ses.secr.}(x), D_{\mathrm{false}}, np)$.

If $\mathsf{Bound}_{G'}(V \cup \{x', x''\}, \mathsf{Secrecy}(x'), \mathsf{NonUnique}_{G'}, p)$ and p satisfies Property 7, then we have $\mathsf{Bound}_G(V \cup \{x\}, \mathsf{Secrecy}(x), D_{\mathrm{false}}, np)$ (neglecting a small additional runtime of the context).

Property 7 guarantees that p is independent of the value of i_{tested}, as well as other independence conditions needed for secrecy and for the properties on Bound because we modify the context in the proof. Since the third argument of Bound is always D_{false} in the conclusion of Lemma 58, we cannot use the optimization of considering the disjunction of several properties simultaneously, as outlined in Section 2.7.4: we must consider each property and event separately. Indeed, if we applied guessing to several properties at once, we might need to guess the tested session for each property, which would introduce several factors n. Since the third argument of Bound is $\mathsf{NonUnique}_{G'}$ in the hypothesis of Lemma 58 for (one-session) secrecy properties, uniqueness of $\mathsf{find}[\mathsf{unique}_e]$ must be reproved in game G' after the guess transformation (that is, the probability that these $\mathsf{find}[\mathsf{unique}_e]$ have several successful choices must be bounded again in G').

Proof

Non-injective correspondences We suppose that the events under the transformed replication contain as argument the replication index i of that replication. (CryptoVerif implicitly adds the current program point and replication indices to each event, and uses fresh distinct variables for the added replication indices in the queries. That does not change the meaning of the query.)

Let $\forall i_0 : [1, n], \widetilde{x} : \widetilde{T}; \mathsf{event}(e(\widetilde{M})) \wedge \psi \Rightarrow \exists \widetilde{y} : \widetilde{T}'; \phi$ be the initial query and $\forall i_0 : [1, n], \widetilde{x} : \widetilde{T}; \mathsf{event}(e'(\widetilde{M})) \wedge \psi \Rightarrow \exists \widetilde{y} : \widetilde{T}'; \phi$ be the transformed query, where $\{i_0, \widetilde{x}\} = \mathrm{var}(\mathsf{event}(e'(\widetilde{M})) \wedge \psi)$, $\widetilde{y} = \mathrm{var}(\phi) \setminus \mathrm{var}(\mathsf{event}(e'(\widetilde{M})) \wedge \psi)$, and i_0 be the variable for the index of the transformed replication in $\widetilde{M}$.

Let C be an evaluation context acceptable for G with public variables V that does not contain events used by φ.

$$\begin{aligned}
\mathsf{Adv}_G(C, \varphi, D_{\mathrm{false}}) &= \Pr[C[G] : \neg\varphi \wedge \neg\mathsf{NonUnique}_G] \\
&= \Pr\left[\begin{array}{l} C[G] : (\exists i_0 \in [1, n], \exists \widetilde{x} \in \widetilde{T}, \mathsf{event}(e(\widetilde{M})) \\ \quad \wedge \psi \wedge \neg\exists \widetilde{y} \in \widetilde{T}', \phi) \wedge \neg\mathsf{NonUnique}_G \end{array}\right] \\
&= \sum_{i_{\mathrm{val}}=1}^{n} \Pr\left[\begin{array}{l} C[G] : (\exists i_0 \in [1, n], \exists \widetilde{x} \in \widetilde{T}, i_0 = i_{\mathrm{val}} \wedge \mathsf{event}(e(\widetilde{M})) \\ \quad \wedge \psi \wedge \neg\exists \widetilde{y} \in \widetilde{T}', \phi) \wedge \neg\mathsf{NonUnique}_G \end{array}\right] \\
&= \sum_{i_{\mathrm{val}}=1}^{n} \Pr\left[\begin{array}{l} C[G'] : (\exists i_0 \in [1, n], \exists \widetilde{x} \in \widetilde{T}, \mathsf{event}(e'(\widetilde{M})) \\ \quad \wedge \psi \wedge \neg\exists \widetilde{y} \in \widetilde{T}', \phi) \wedge \neg\mathsf{NonUnique}_{G'} \end{array}\right] \text{ for } i_{\mathrm{tested}} = i_{\mathrm{val}}
\end{aligned}$$

$$= \sum_{i_{\mathrm{val}}=1}^{n} \mathsf{Adv}_{G'}(C, \varphi', D_{\mathrm{false}}) \text{ for } i_{\mathrm{tested}} = i_{\mathrm{val}}$$

Since $\mathsf{Bound}_{G'}(V, \varphi', D_{\mathrm{false}}, p)$, we have $\mathsf{Adv}_{G'}(C, \varphi', D_{\mathrm{false}}) \leq p(C)$ and the probability p is independent of the value of i_{tested}, so we obtain $\mathsf{Adv}_{G}(C, \varphi, D_{\mathrm{false}}) \leq n \times p(C)$. Therefore $\mathsf{Bound}_{G}(V, \varphi, D_{\mathrm{false}}, np)$.

One-session secrecy Let C be an evaluation context acceptable for $C_{\text{1-ses.secr.}(x)}[G]$ with public variables V that does not contain S nor $\overline{\mathsf{S}}$. Suppose that the modified replication corresponds to the j-th index of variable x. We have

$$\begin{aligned}
\mathsf{Adv}_G^{\text{1-ses.secr.}(x)}(C) &= \Pr[C[C_{\text{1-ses.secr.}(x)}[G]] : \mathsf{S}] - \Pr[C[C_{\text{1-ses.secr.}(x)}[G]] : \overline{\mathsf{S}}] \\
&= \sum_{v=1}^{n} \begin{array}{l} \Pr[C[C_{\text{1-ses.secr.}(x)}[G]] : \mathsf{S} \wedge u_j = v] \\ - \Pr[C[C_{\text{1-ses.secr.}(x)}[G]] : \overline{\mathsf{S}} \wedge u_j = v] \end{array} \\
&= \sum_{v=1}^{n} \begin{array}{l} \Pr[C[C_{\text{1-ses.secr.}(x')}[G']] : \mathsf{S} \wedge u_j = v] \\ - \Pr[C[C_{\text{1-ses.secr.}(x')}[G']] : \overline{\mathsf{S}} \wedge u_j = v] \end{array} \text{ for } i_{\mathrm{tested}} = v
\end{aligned}$$

because in G' with $i_{\mathrm{tested}} = v$, $x'[u_1, \dots, u_m] = x[u_1, \dots, u_m]$ when $u_j = v$, so $C_{\text{1-ses.secr.}(x)}[G]$ behaves like $C_{\text{1-ses.secr.}(x')}[G']$ (the events added in G' are not used).

Moreover, $\Pr[C[C_{\text{1-ses.secr.}(x')}[G']] : \mathsf{S} \wedge (u_j \text{ not defined} \vee u_j \neq v)] = \Pr[C[C_{\text{1-ses.secr.}(x')}[G']] : \overline{\mathsf{S}} \wedge (u_j \text{ not defined} \vee u_j \neq v)]$ when $i_{\mathrm{tested}} = v$. Indeed, when u_j is not defined or $u_j \neq v = i_{\mathrm{tested}}$, the query on c_s either is not executed or always yields, independently of the value of b. Hence, changing the value of b just swaps the events S and $\overline{\mathsf{S}}$. So

$$\begin{aligned}
&\Pr[C[C_{\text{1-ses.secr.}(x')}[G']] : \mathsf{S} \wedge (u_j \text{ not defined} \vee u_j \neq v) \wedge b = \text{true}] = \\
&\qquad \Pr[C[C_{\text{1-ses.secr.}(x')}[G']] : \overline{\mathsf{S}} \wedge (u_j \text{ not defined} \vee u_j \neq v) \wedge b = \text{false}] \\
&\Pr[C[C_{\text{1-ses.secr.}(x')}[G']] : \overline{\mathsf{S}} \wedge (u_j \text{ not defined} \vee u_j \neq v) \wedge b = \text{true}] = \\
&\qquad \Pr[C[C_{\text{1-ses.secr.}(x')}[G']] : \mathsf{S} \wedge (u_j \text{ not defined} \vee u_j \neq v) \wedge b = \text{false}].
\end{aligned}$$

We obtain the announced result by swapping the two sides of the second equality and adding the first equality to it. (The variable b is always defined when S or $\overline{\mathsf{S}}$ is executed.)

Therefore,

$$\begin{aligned}
\mathsf{Adv}_G^{\text{1-ses.secr.}(x)}(C) &= \sum_{v=1}^{n} \Pr[C[C_{\text{1-ses.secr.}(x')}[G']] : \mathsf{S}] - \Pr[C[C_{\text{1-ses.secr.}(x')}[G']] : \overline{\mathsf{S}}] \text{ for } i_{\mathrm{tested}} = v \\
&= \sum_{v=1}^{n} \mathsf{Adv}_{G'}^{\text{1-ses.secr.}(x')}(C) \text{ for } i_{\mathrm{tested}} = v
\end{aligned}$$

Since G' satisfies the one-session secrecy of x' with public variables V up to probability p, we have $\mathsf{Adv}_{G'}^{\text{1-ses.secr.}(x')}(C) \leq p(C)$ and the probability p is independent of the value of i_{tested}, so we obtain $\mathsf{Adv}_{G}^{\text{1-ses.secr.}(x)}(C) \leq n \times p(C)$. Therefore, G satisfies the one-session secrecy of x with public variables V up to probability np.

Secrecy Let C be an evaluation context acceptable for $C_{\mathsf{Secrecy}(x)}[G]$ with public variables V that does not contain S nor $\overline{\mathsf{S}}$. Suppose that the modified replication corresponds to the j-th index of variable x. We have

$$\begin{aligned}&\mathsf{Adv}_G^{\mathsf{Secrecy}(x)}(C)\\ &= \frac{1}{2}\begin{pmatrix} \Pr[C[G \mid Q_{\mathsf{Secrecy}(x)}] : \mathsf{S} \mid b = \text{true}] + \Pr[C[G \mid Q_{\mathsf{Secrecy}(x)}] : \mathsf{S} \mid b = \text{false}] \\ - \Pr[C[G \mid Q_{\mathsf{Secrecy}(x)}] : \overline{\mathsf{S}} \mid b = \text{true}] - \Pr[C[G \mid Q_{\mathsf{Secrecy}(x)}] : \overline{\mathsf{S}} \mid b = \text{false}]\end{pmatrix}\end{aligned}$$

Let

$$\begin{aligned}Q_{\mathsf{Secrecy}(x),\text{real}} = {}& c_{s0}(); \overline{c_{s0}}\langle\rangle;\\ & (!^{i_s \le n_s} c_s[i_s](u_1 : [1, n_1], \dots, u_m : [1, n_m]); \mathsf{if\ defined}(x[u_1, \dots, u_m])\ \mathsf{then}\\ & \quad \overline{c_s[i_s]}\langle x[u_1, \dots, u_m]\rangle\\ & \mid c'_s(b' : \mathit{bool}); \mathsf{if}\ b'\ \mathsf{then\ event\_abort\ S\ else\ event\_abort}\ \overline{\mathsf{S}})\\ Q_{\mathsf{Secrecy}(x),\text{random}} = {}& c_{s0}(); \overline{c_{s0}}\langle\rangle;\\ & (!^{i_s \le n_s} c_s[i_s](u_1 : [1, n_1], \dots, u_m : [1, n_m]); \mathsf{if\ defined}(x[u_1, \dots, u_m])\ \mathsf{then}\\ & \quad \mathsf{find}\ u'_s = i'_s \le n_s\ \mathsf{suchthat\ defined}(y[i'_s], u_1[i'_s], \dots, u_m[i'_s]) \wedge\\ & \qquad u_1[i'_s] = u_1 \wedge \dots \wedge u_m[i'_s] = u_m\\ & \quad \mathsf{then}\ \overline{c_s[i_s]}\langle y[u'_s]\rangle\\ & \quad \mathsf{else\ new}\ y : T; \overline{c_s[i_s]}\langle y\rangle\\ & \mid c'_s(b' : \mathit{bool}); \mathsf{if}\ b'\ \mathsf{then\ event\_abort\ S\ else\ event\_abort}\ \overline{\mathsf{S}})\end{aligned}$$

Then

$$\mathsf{Adv}_G^{\mathsf{Secrecy}(x)}(C) = \frac{1}{2}\begin{pmatrix} \Pr[C[G \mid Q_{\mathsf{Secrecy}(x),\text{real}}] : \mathsf{S}] + \Pr[C[G \mid Q_{\mathsf{Secrecy}(x),\text{random}}] : \overline{\mathsf{S}}] \\ - \Pr[C[G \mid Q_{\mathsf{Secrecy}(x),\text{real}}] : \overline{\mathsf{S}}] - \Pr[C[G \mid Q_{\mathsf{Secrecy}(x),\text{random}}] : \mathsf{S}]\end{pmatrix}$$

Let

$$\begin{aligned}Q_{\mathsf{Secrecy}(x),v} = {}& c_{s0}(); \overline{c_{s0}}\langle\rangle;\\ & (!^{i_s \le n_s} c_s[i_s](u_1 : [1, n_1], \dots, u_m : [1, n_m]); \mathsf{if\ defined}(x[u_1, \dots, u_m])\ \mathsf{then}\\ & \quad \mathsf{if}\ u_j \le v\ \mathsf{then}\ \overline{c_s[i_s]}\langle x[u_1, \dots, u_m]\rangle\ \mathsf{else}\\ & \quad \mathsf{find}\ u'_s = i'_s \le n_s\ \mathsf{suchthat\ defined}(y[i'_s], u_1[i'_s], \dots, u_m[i'_s]) \wedge\\ & \qquad u_1[i'_s] = u_1 \wedge \dots \wedge u_m[i'_s] = u_m\\ & \quad \mathsf{then}\ \overline{c_s[i_s]}\langle y[u'_s]\rangle\\ & \quad \mathsf{else\ new}\ y : T; \overline{c_s[i_s]}\langle y\rangle\\ & \mid c'_s(b' : \mathit{bool}); \mathsf{if}\ b'\ \mathsf{then\ event\_abort\ S\ else\ event\_abort}\ \overline{\mathsf{S}})\end{aligned}$$

The process $Q_{\mathsf{Secrecy}(x),0}$ behaves like $Q_{\mathsf{Secrecy}(x),\text{random}}$ and $Q_{\mathsf{Secrecy}(x),n}$ behaves like $Q_{\mathsf{Secrecy}(x),\text{real}}$ ($n_j = n$). Therefore,

$$\begin{aligned}\mathsf{Adv}_G^{\mathsf{Secrecy}(x)}(C) &= \frac{1}{2}\begin{pmatrix} (\Pr[C[G \mid Q_{\mathsf{Secrecy}(x),n}] : \mathsf{S}] - \Pr[C[G \mid Q_{\mathsf{Secrecy}(x),0}] : \mathsf{S}]) \\ - (\Pr[C[G \mid Q_{\mathsf{Secrecy}(x),n}] : \overline{\mathsf{S}}] - \Pr[C[G \mid Q_{\mathsf{Secrecy}(x),0}] : \overline{\mathsf{S}}])\end{pmatrix}\\ &= \frac{1}{2}\begin{pmatrix} \sum_{v=1}^{n}(\Pr[C[G \mid Q_{\mathsf{Secrecy}(x),v}] : \mathsf{S}] - \Pr[C[G \mid Q_{\mathsf{Secrecy}(x),v-1}] : \mathsf{S}]) \\ - \sum_{v=1}^{n}(\Pr[C[G \mid Q_{\mathsf{Secrecy}(x),v}] : \overline{\mathsf{S}}] - \Pr[C[G \mid Q_{\mathsf{Secrecy}(x),v-1}] : \overline{\mathsf{S}}])\end{pmatrix}\end{aligned}$$

We define a context C'_v that returns a random value for $u_j > v$, the real value of x obtained from the public variable x'' in G' for $u_j < v$, and calls $Q_{\mathsf{Secrecy}(x'),\mathrm{real}}$ or $Q_{\mathsf{Secrecy}(x'),\mathrm{random}}$ for $u_j = v$.

```
$C'_v = \mathsf{newChannel}\ c_{s1}; ([\,] \mid !^{i_s \leq n_s}\ c_s[i_s](u_1 : [1, n_1], \dots, u_m : [1, n_m]);$
            $\mathsf{if}\ u_j = v\ \mathsf{then}\ \overline{c_{s1}[i_s]}\langle u_1, \dots, u_m\rangle; c_{s1}[i_s](z : T); \overline{c_s[i_s]}\langle z\rangle\ \mathsf{else}$
            $\mathsf{if}\ \mathsf{defined}(x''[u_1, \dots, u_m])\ \mathsf{then}$
            $\mathsf{if}\ u_j < v\ \mathsf{then}\ \overline{c_s[i_s]}\langle x''[u_1, \dots, u_m]\rangle\ \mathsf{else}$
            $\mathsf{find}\ u'_s = i'_s \leq n_s\ \mathsf{suchthat}\ \mathsf{defined}(y[i'_s], u_1[i'_s], \dots, u_m[i'_s]) \wedge$
                $u_1[i'_s] = u_1 \wedge \dots \wedge u_m[i'_s] = u_m$
            $\mathsf{then}\ \overline{c_s[i_s]}\langle y[u'_s]\rangle$
            $\mathsf{else}\ \mathsf{new}\ y : T; \overline{c_s[i_s]}\langle y\rangle)$
```

where the processes $Q_{\mathsf{Secrecy}(x'),\mathrm{real}}$ and $Q_{\mathsf{Secrecy}(x'),\mathrm{random}}$ use channel c_{s1} instead of c_s. When $u_j = v$, the query on $c_s[i_s]$ is forwarded to $Q_{\mathsf{Secrecy}(x'),\mathrm{real}}$ (resp. $Q_{\mathsf{Secrecy}(x'),\mathrm{random}}$) on channel $c_{s1}[i_s]$. The values of x for sessions other than v are collected in x'' by G'; these are the values returned by the query on $c_s[i_s]$ when $u_j < v$. Finally, when $u_j > v$, the query on $c_s[i_s]$ is answered with a random value y.

Then

$$\begin{aligned}
&\Pr[C[G \mid Q_{\mathsf{Secrecy}(x),v}] : \mathsf{S}] = \Pr[C[C'_v[G' \mid Q_{\mathsf{Secrecy}(x'),\mathrm{real}}]] : \mathsf{S}],\\
&\Pr[C[G \mid Q_{\mathsf{Secrecy}(x),v}] : \overline{\mathsf{S}}] = \Pr[C[C'_v[G' \mid Q_{\mathsf{Secrecy}(x'),\mathrm{real}}]] : \overline{\mathsf{S}}],\\
&\Pr[C[G \mid Q_{\mathsf{Secrecy}(x),v-1}] : \mathsf{S}] = \Pr[C[C'_v[G' \mid Q_{\mathsf{Secrecy}(x'),\mathrm{random}}]] : \mathsf{S}],\ \text{and}\\
&\Pr[C[G \mid Q_{\mathsf{Secrecy}(x),v-1}] : \overline{\mathsf{S}}] = \Pr[C[C'_v[G' \mid Q_{\mathsf{Secrecy}(x'),\mathrm{random}}]] : \overline{\mathsf{S}}],
\end{aligned}$$

for $i_{\mathrm{tested}} = v$.

Then

$$\begin{aligned}
&\mathsf{Adv}_G^{\mathsf{Secrecy}(x)}(C)\\
&\quad = \sum_{v=1}^{n} \frac{1}{2} \begin{pmatrix} \Pr[C[C'_v[G' \mid Q_{\mathsf{Secrecy}(x'),\mathrm{real}}]] : \mathsf{S}] - \Pr[C[C'_v[G' \mid Q_{\mathsf{Secrecy}(x'),\mathrm{random}}]] : \mathsf{S}]\\ - \Pr[C[C'_v[G' \mid Q_{\mathsf{Secrecy}(x'),\mathrm{real}}]] : \overline{\mathsf{S}}] + \Pr[C[C'_v[G' \mid Q_{\mathsf{Secrecy}(x'),\mathrm{random}}]] : \overline{\mathsf{S}}] \end{pmatrix}\\
&\qquad \text{for } i_{\mathrm{tested}} = v\\
&\quad = \sum_{v=1}^{n} \mathsf{Adv}_{G'}^{\mathsf{Secrecy}(x')}(C[C'_v]) \text{ for } i_{\mathrm{tested}} = v
\end{aligned}$$

by the link between the advantage for secrecy and $Q_{\mathsf{Secrecy}(x'),\mathrm{real}}$, $Q_{\mathsf{Secrecy}(x'),\mathrm{random}}$ shown above. Since G' satisfies the secrecy of x' with public variables $V \cup \{x''\}$ up to probability p and $C[C'_v]$ is an evaluation context acceptable for $G' \mid Q_{\mathsf{Secrecy}(x')}$ with public variables $V \cup \{x''\}$ that does not contain S nor $\overline{\mathsf{S}}$, we have $\mathsf{Adv}_{G'}^{\mathsf{Secrecy}(x')}(C[C'_v]) \leq p(C[C'_v])$. Moreover, by Property 7, the probability $p(C[C'_v])$ is independent of $i_{\mathrm{tested}} = v$ (because the type of i_{tested} is bounded) and depends only on the runtime of C, the number of outputs C makes on the various channels (which determine replication bounds), and the length of bitstrings, so we have $p(C[C'_v]) = p(C)$ (the runtime of C'_v can be neglected). Hence, we obtain $\mathsf{Adv}_G^{\mathsf{Secrecy}(x)}(C) \leq n \times p(C)$. Therefore, G satisfies the secrecy of x with public variables V up to probability $n \times p$.

One-session secrecy, secrecy, and bit secrecy In this lemma, bit secrecy queries do not occur. However, we reuse this proof in Lemmas 59 and 60 where bit secrecy queries occur, so we also handle them here. Let sp be $\mathsf{1\text{-}ses.secr.}(x)$, $\mathsf{Secrecy}(x)$, or $\mathsf{bit\ secr.}(x)$, and sp' be the same property with x' instead of x. Let $V' = V$ when sp is $\mathsf{1\text{-}ses.secr.}(x)$ or $\mathsf{bit\ secr.}(x)$, and $V' = V \cup \{x''\}$ when sp is $\mathsf{Secrecy}(x)$. Since $\mathsf{Bound}_{G'}(V' \cup \{x'\}, sp', \mathsf{NonUnique}_{G'}, p)$, then by Lemma 27, Property 1, G' satisfies sp' with public variables V' up to probability p' such that $p'(C) = p(C[C_{sp'}])$. So by the previous result for (one-session or bit) secrecy, G satisfies sp with public variables V up to probability np'. Let C be an evaluation context acceptable for $C_{sp}[G]$ with public variables V that does not contain S nor $\overline{\mathsf{S}}$. Hence

$$\begin{aligned}&\mathsf{Adv}_G(C[C_{sp}[\,]], sp, D_{\text{false}})\\ &\quad = \Pr[C[C_{sp}[G]] : \mathsf{S}] - \Pr[C[C_{sp}[G]] : \overline{\mathsf{S}} \vee \mathsf{NonUnique}_G]\\ &\quad \le \Pr[C[C_{sp}[G]] : \mathsf{S}] - \Pr[C[C_{sp}[G]] : \overline{\mathsf{S}}]\\ &\quad = \mathsf{Adv}_G^{sp}(C) \le n \times p'(C) = n \times p(C[C_{sp}[\,]])\end{aligned}$$

Indeed, replacing $C_{sp'}$ with C_{sp} in the argument of p does not change its result, by Property 7. Therefore, we have $\mathsf{Bound}_G(V \cup \{x\}, sp, D_{\text{false}}, np)$. □

In the **guess** transformation, we cannot modify injective correspondence queries, because two executions of some injective event e with different indices i could be mapped to the same events in the conclusion of the query. In general, it even does not work for non-injective events inside injective queries. As a counter-example, consider the query: $\forall i : [1, n], x : T'; \mathsf{event}(e_1(i, x)) \wedge \mathsf{inj\text{-}event}(e_2(x)) \Rightarrow \mathsf{inj\text{-}event}(e_3())$ with events $e_1(i_1, x_1)$, $e_1(i_2, x_2)$, $e_2(x_1)$, $e_2(x_2)$ and e_3 each executed once. This query is false: we have two executions of e_2 (with matching executions of e_1) for a single execution of e_3. That contradicts injectivity. However, it is true if we restrict ourselves to one value of i (the index of the tested session), because we consider $e_1(i_1, x_1)$, $e_2(x_1)$ and e_3 for $i = i_1$ and $e_1(i_2, x_2)$, $e_2(x_2)$ and e_3 for $i = i_2$. Requiring the same value of x in e_1 and e_2 restricts the events e_2 that we consider when we guess the session for e_1. Therefore, proving the query for the tested session does not allow us to prove it in the initial game.

Hence, for injective correspondence queries $\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi$, we proceed as follows: we define a non-injective query $\text{noninj}(\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi)$ simply obtained by replacing injective events with non-injective events, and we try to prove that $\text{noninj}(\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi)$ implies $\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi$ in the current game. This proof is a modified version of the proof of injective queries (Section 4.2.4): we define the pseudo-formula $\mathcal{C}(\psi, \phi)$ by

$\mathcal{C}(\psi, M) = \bot$

$\mathcal{C}(\psi, \mathsf{event}(e(\widetilde{M}))) = \bot$

$\mathcal{C}(\psi, \mathsf{inj\text{-}event}(e(\widetilde{M})) = \mathcal{S}$ such that assuming $\psi = F_1 \wedge \cdots \wedge F_m$,
  for every μ_1 that executes F_1, ..., for every μ_m that executes F_m, letting
    $\mathcal{F} = \theta_1 \mathcal{F}_{F_1, \mu_1} \cup \cdots \cup \theta_m \mathcal{F}_{F_m, \mu_m}$,
    $\mathcal{I} = \{j \mapsto (\mu_j, \theta_j I_{\mu_j}) \mid F_j \text{ is an injective event}\}$,
    $\mathcal{V} = \text{var}(\theta_1 I_{\mu_1}) \cup \cdots \cup \text{var}(\theta_m I_{\mu_m}) \cup \{\widetilde{x}, \widetilde{y}\}$,
  where for $j \le m$, θ_j is a renaming of I_{μ_j} to fresh replication indices,
  we have $(\mathcal{F}, (\widetilde{M}), \mathcal{I}, \mathcal{V}) \in \mathcal{S}$.

$\mathcal{C}(\psi, \phi_1 \wedge \phi_2) = \mathcal{C}(\psi, \phi_1) \wedge \mathcal{C}(\psi, \phi_2)$

$\mathcal{C}(\psi, \phi_1 \vee \phi_2) = \mathcal{C}(\psi, \phi_1) \vee \mathcal{C}(\psi, \phi_2)$

Given a pseudo-formula $\mathcal{C}$, we define $\vdash \mathcal{C}$ as in Section 4.2.4.

Proposition 4 *Let $\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi$ be a correspondence, with $\widetilde{x} = \mathrm{var}(\psi)$ and $\widetilde{y} = \mathrm{var}(\phi) \setminus \mathrm{var}(\psi)$. Let $\varphi = [\![\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi]\!]$ be the semantics the correspondence $\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi$ (Definition 12), and $\varphi_{\mathrm{ni}} = [\![\mathrm{noninj}(\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi)]\!]$ be the semantics of the correspondence $\mathrm{noninj}(\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi)$. Let Q_0 be a process that satisfies Properties 4 and 5. Suppose that, in Q_0, the arguments of the events that occur in ψ are always simple terms.*

Assume that $\vdash \mathcal{C}(\psi, \phi)$ and for all evaluation contexts C acceptable for Q_0, $\Pr[C[Q_0] \preceq \neg\{[\![\vdash \mathcal{C}(\psi, \phi)]\!]\}] \leq p(C)$. If $\mathsf{Bound}_{Q_0}(V, \varphi_{\mathrm{ni}}, D_{\mathrm{false}}, p')$, then $\mathsf{Bound}_{Q_0}(V, \varphi, D_{\mathrm{false}}, p' + p)$.

Proposition 4 proves injectivity much like in Section 4.2.4, but using the arguments $\widetilde{M}$ of the events in ϕ instead of the program points and replication indices at their execution (which we do not have since in Q_0 we are not able to prove that these events have been executed; otherwise we would simply prove the correspondence in Q_0). Intuitively, $\vdash \mathcal{C}(\psi, \phi)$ shows that, if we have different executions of injective events in ψ, that is, executions of such events with different pairs (program point, replication indices), then the arguments $\widetilde{M}$ of each injective event in ϕ must be different, which implies different executions of this event.

Proof Let C be an evaluation context acceptable for Q_0 with public variables V that does not contain events used by φ. Let $\mathcal{C} = \mathcal{C}(\psi, \phi)$. Consider a trace Tr of $C[Q_0]$ such that $Tr \vdash \varphi_{\mathrm{ni}}$, $Tr \vdash \{[\![\vdash \mathcal{C}]\!]\}$, Tr does not execute a non-unique event of Q_0, and the last configuration of Tr cannot be reduced. Let $\mu\mathcal{E}v$ be the sequence of events in the last configuration of Tr. Since $\mu\mathcal{E}v \vdash \varphi_{\mathrm{ni}}$,

$$\mu\mathcal{E}v \vdash \forall \tau_1, \dots, \tau_m \in \mathbb{N}, \forall \widetilde{x} \in \widetilde{T}, (\psi^\tau \Rightarrow \exists \widetilde{y} \in \widetilde{T'}, \phi)$$

with the notations of Definition 12. By defining functions $f_j \in \mathbb{N}^m \times \prod \widetilde{T} \to \mathbb{N} \cup \{\bot\}$ that map $\tau_1, \dots, \tau_m, \widetilde{x}$ to the execution steps of injective events in the proof of ϕ, and to $\bot$ when the event is not used in the proof of ϕ, we have

$$\mu\mathcal{E}v \vdash \exists f_1, \dots, f_k \in \mathbb{N}^m \times \prod \widetilde{T} \to \mathbb{N} \cup \{\bot\}, \forall \tau_1, \dots, \tau_m \in \mathbb{N}, \forall \widetilde{x} \in \widetilde{T}, (\psi^\tau \Rightarrow \exists \widetilde{y} \in \widetilde{T'}, \phi^\tau) \quad (25)$$

It remains to show $\mathrm{Inj}(I, f)$ for all $f \in \{f_1, \dots, f_k\}$.

Let $f \in \{f_1, \dots, f_k\}$ and $\mathsf{event}(e(\widetilde{M}))@f(\tau_1, \dots, \tau_m, \widetilde{x})$ be the event labeled with f in ϕ^τ. Suppose that $f(\tau'_1, \dots, \tau'_m, \widetilde{a}') = f(\tau''_1, \dots, \tau''_m, \widetilde{a}'') \neq \bot$ and there exists $j \in I$ such that $\tau'_j \neq \tau''_j$, and let us prove a contradiction.

Since $f(\tau'_1, \dots, \tau'_m, \widetilde{a}') \neq \bot$, $\mathsf{event}(e(\widetilde{M}))@f(\tau'_1, \dots, \tau'_m, \widetilde{a}')$ is used in the proof of (25), so letting $\rho_1 = \{\tau_1 \mapsto \tau'_1, \dots, \tau_m \mapsto \tau'_m, \widetilde{x} \mapsto \widetilde{a}'\}$, $\rho_1, \mu\mathcal{E}v \vdash \psi^\tau$ and there exists an extension ρ'_1 of ρ_1 to $\widetilde{y}$ such that $\rho'_1, \mu\mathcal{E}v \vdash \phi^\tau$ and $\rho'_1, \mu\mathcal{E}v \vdash \mathsf{event}(e(\widetilde{M}))@f(\tau_1, \dots, \tau_m, \widetilde{x})$.

Since $\rho_1, \mu\mathcal{E}v \vdash \psi^\tau$, for all events $F_\ell = \mathsf{event}(e_\ell(\widetilde{M_\ell}))@\tau_\ell$ in ψ^τ, $Tr, \rho_1 \vdash F_\ell$ and $\mu\mathcal{E}v(\tau'_\ell) = (\dots) : e_\ell(\widetilde{a}_{\ell,1})$ for $\widetilde{a}_{\ell,1}$ such that $\rho_1, \widetilde{M_\ell} \Downarrow \widetilde{a}_{\ell,1}$. By Lemma 37, there exists a program point $\mu_{\ell,1}$ that executes F_ℓ (in Q_0) and a case $c_{\ell,1}$ such that, for any $\theta_{\ell,1}$ renaming of $I_{\mu_{\ell,1}}$ to fresh replication indices, there exists a mapping $\sigma_{\ell,1}$ with domain $\theta_{\ell,1} I_{\mu_{\ell,1}}$ such that $\mu\mathcal{E}v(\rho_1(\tau_\ell)) = (\mu_{\ell,1}, \sigma_{\ell,1}(\theta_{\ell,1} I_{\mu_{\ell,1}})) : \dots$ and $Tr, \sigma_{\ell,1} \cup \rho_1 \vdash \theta_{\ell,1} \mathcal{F}_{F_\ell, \mu_{\ell,1}, c_{\ell,1}}$. So $\mu\mathcal{E}v(\tau'_\ell) = (\mu_{\ell,1}, \sigma_{\ell,1}(\theta_{\ell,1} I_{\mu_{\ell,1}})) : e_\ell(\widetilde{a}_{\ell,1})$.

Let $\mathcal{S}$ be the label of $\mathcal{C}$ at the occurrence corresponding to f, and $\mathsf{inj\text{-}event}(e(\widetilde{M}))$ be the injective event at that occurrence in ϕ. Let $\mathcal{F}_1 = \bigcup_\ell \theta_{\ell,1} \mathcal{F}_{F_\ell, \mu_{\ell,1}, c_{\ell,1}}$, $\mathcal{I}_1 = \{\ell \mapsto (\mu_{\ell,1}, \theta_{\ell,1} I_{\mu_{\ell,1}}) \mid F_\ell \text{ is an injective event}\}$, and $\mathcal{V}_1 = \mathrm{var}(\theta_{1,1} I_{\mu_{1,1}}) \cup \dots \cup \mathrm{var}(\theta_{m,1} I_{\mu_{m,1}}) \cup \{\widetilde{x}, \widetilde{y}\}$. By construction of $\mathcal{C}$, we have $(\mathcal{F}_1, (\widetilde{M}), \mathcal{I}_1, \mathcal{V}_1) \in \mathcal{S}$.

So we have $Tr, \bigcup_\ell \sigma_{\ell,1} \cup \rho_1 \vdash \bigcup_\ell \theta_{\ell,1} \mathcal{F}_{F_\ell, \mu_{\ell,1}, c_{\ell,1}}$. Letting $\sigma_1 = \bigcup_\ell \sigma_{\ell,1}$, we have $Tr, \sigma_1 \cup \rho'_1 \vdash \mathcal{F}_1$; for ℓ such that F_ℓ is an injective event, $\mu\mathcal{E}v(\tau'_\ell) = \sigma_1 \mathcal{I}_1(\ell) : e_\ell(\dots)$; $\mathcal{V}_1 = \mathrm{Dom}(\sigma_1) \cup \{\widetilde{x}, \widetilde{y}\}$. Since

$\rho'_1, \mu\mathcal{E}v \vdash \mathsf{event}(e(\widetilde{M}))@f(\tau_1, \dots, \tau_m, \widetilde{x})$, we have $\rho'_1, \widetilde{M} \Downarrow \widetilde{a}_1$ and $\mu\mathcal{E}v(f(\tau'_1, \dots, \tau'_m, \widetilde{a}')) = (\dots) : e(\widetilde{a}_1)$ for some $\widetilde{a}_1$.

Since $f(\tau''_1, \dots, \tau''_m, \widetilde{a}'') \neq \bot$, we have similarly $(\mathcal{F}_2, (\widetilde{M}), \mathcal{I}_2, \mathcal{V}_2) \in \mathcal{S}$, ρ'_2, and σ_2 such that $Tr, \sigma_2 \cup \rho'_2 \vdash \mathcal{F}_2$; for ℓ such that F_ℓ is an injective event, $\mu\mathcal{E}v(\tau''_\ell) = \sigma_2\mathcal{I}_2(\ell) : e_\ell(\dots)$; $\mathcal{V}_2 = \mathrm{Dom}(\sigma_2) \cup \{\widetilde{x}, \widetilde{y}\}$; $\rho'_2, \widetilde{M} \Downarrow \widetilde{a}_2$ and $\mu\mathcal{E}v(f(\tau''_1, \dots, \tau''_m, \widetilde{a}'')) = (\dots) : e(\widetilde{a}_1)$ for some $\widetilde{a}_2$.

Let θ'' be a renaming of variables in $\mathcal{V}_2$. Then $Tr, \sigma_2\theta''^{-1} \cup \rho'_2\theta''^{-1} \vdash \theta''\mathcal{F}_2$; for ℓ such that F_ℓ is an injective event, $\mu\mathcal{E}v(\tau''_\ell) = \sigma_2\theta''^{-1}\theta''\mathcal{I}_2(\ell) : e_\ell(\dots)$; $\rho'_2\theta''^{-1}, \theta''\widetilde{M} \Downarrow \widetilde{a}_2$ and $\mu\mathcal{E}v(f(\tau''_1, \dots, \tau''_m, \widetilde{a}'')) = (\dots) : e(\widetilde{a}_2)$ for some $\widetilde{a}_2$.

Then $Tr, \sigma_1 \cup \sigma_2\theta''^{-1} \cup \rho'_1 \cup \rho'_2\theta''^{-1} \vdash \mathcal{F}_1 \cup \theta''\mathcal{F}_2$.

There exists $j \in I$ such that $\tau'_j \neq \tau''_j$, so $\sigma_1\mathcal{I}_1(j) \neq \sigma_2\theta''^{-1}\theta''\mathcal{I}_2(j)$ (distinct events have distinct pairs (program point, replication indices) by Lemma 40), so there exists $j \in \mathrm{Dom}(\mathcal{I}_1) = I$ such that $Tr, \sigma_1 \cup \sigma_2\theta''^{-1} \cup \rho'_1 \cup \rho'_2\theta''^{-1} \vdash \mathcal{I}_1(j) \neq \theta''\mathcal{I}_2(j)$.

Since $f(\tau'_1, \dots, \tau'_m, \widetilde{a}') = f(\tau''_1, \dots, \tau''_m, \widetilde{a}'')$, $\mu\mathcal{E}v(f(\tau'_1, \dots, \tau'_m, \widetilde{a}')) = \mu\mathcal{E}v(f(\tau''_1, \dots, \tau''_m, \widetilde{a}''))$, so $\widetilde{a}_1 = \widetilde{a}_2$, so $Tr, \sigma_1 \cup \sigma_2\theta''^{-1} \cup \rho'_1 \cup \rho'_2\theta''^{-1} \vdash \widetilde{M} = \theta''\widetilde{M}$.

So $Tr, \sigma_1 \cup \sigma_2\theta''^{-1} \cup \rho'_1 \cup \rho'_2\theta''^{-1} \vdash \mathcal{F}_1 \cup \theta''\mathcal{F}_2 \cup \{\bigvee_{j \in \mathrm{Dom}(\mathcal{I}_1)} \mathcal{I}_1(j) \neq \theta''\mathcal{I}_2(j), \widetilde{M} = \theta''\widetilde{M}\}$.

Since the trace satisfies $\{\![\vdash \mathcal{C}]\!\}$, this is a contradiction. Therefore, we conclude that the considered trace satisfies φ. Hence, every full trace of $C[Q_0]$ that satisfies φ_{ni}, $\{\![\vdash \mathcal{C}]\!\}$, and does not execute a non-unique event of Q_0 also satisfies φ. Therefore, every full trace of $C[Q_0]$ that satisfies $\neg\varphi$ satisfies $\neg(\varphi_{\mathrm{ni}} \wedge \{\![\vdash \mathcal{C}]\!\} \wedge \neg\mathsf{NonUnique}_{Q_0})$, so every full trace of $C[Q_0]$ that satisfies $\neg\varphi \wedge \neg\mathsf{NonUnique}_{Q_0}$ satisfies $\neg(\varphi_{\mathrm{ni}} \wedge \{\![\vdash \mathcal{C}]\!\} \wedge \neg\mathsf{NonUnique}_{Q_0}) \wedge \neg\mathsf{NonUnique}_{Q_0} = (\neg\varphi_{\mathrm{ni}} \vee \neg\{\![\vdash \mathcal{C}]\!\}) \wedge \neg\mathsf{NonUnique}_{Q_0} = (\neg\varphi_{\mathrm{ni}} \wedge \neg\mathsf{NonUnique}_{Q_0}) \vee (\neg\{\![\vdash \mathcal{C}]\!\} \wedge \neg\mathsf{NonUnique}_{Q_0,D_{\mathrm{false}}})$, so it satisfies $(\neg\varphi_{\mathrm{ni}} \wedge \neg\mathsf{NonUnique}_{Q_0}) \vee \neg\{\![\vdash \mathcal{C}]\!\}$. So

$$\begin{aligned}
\mathsf{Adv}_{Q_0}(C, \varphi, D_{\mathrm{false}}) &= \Pr[C[Q_0] : \neg\varphi \wedge \neg\mathsf{NonUnique}_{Q_0}] \\
&\leq \Pr[C[Q_0] : (\neg\varphi_{\mathrm{ni}} \wedge \neg\mathsf{NonUnique}_{Q_0}) \vee \neg\{\![\vdash \mathcal{C}]\!\}] \\
&\leq \Pr[C[Q_0] : \neg\{\![\vdash \mathcal{C}]\!\}] + \Pr[C[Q_0] : \neg\varphi_{\mathrm{ni}} \wedge \neg\mathsf{NonUnique}_{Q_0})] \\
&\leq \Pr[C[Q_0] : \neg\{\![\vdash \mathcal{C}]\!\}] + \mathsf{Adv}_{Q_0}(C, \varphi_{\mathrm{ni}}, D_{\mathrm{false}}) \\
&\leq \Pr[C[Q_0] : \neg\{\![\vdash \mathcal{C}]\!\}] + p(C) \qquad \text{since } \mathsf{Bound}_{Q_0}(V, \varphi_{\mathrm{ni}}, D_{\mathrm{false}}, p) \\
&\leq p'(C)\,.
\end{aligned}$$

Therefore $\mathsf{Bound}_{Q_0}(V, \varphi, D_{\mathrm{false}}, p')$. □

If the proof that $\mathrm{noninj}(\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi)$ implies $\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi$ works, we just have to prove $\mathrm{noninj}(\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi)$ and we can apply the **guess** transformation for non-injective correspondences. Otherwise, we simply leave the query $\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi$ unchanged.

A transformation **guess** i **&& above**, similar to **guess** i, can be used to guess the whole sequence $\widetilde{i}$ of replication indices above and including the modified replication, by testing the equality $\widetilde{i} = \widetilde{i}_{\mathrm{tested}}$ instead of $i = i_{\mathrm{tested}}$.

5.1.18 guess $x[c_1, \dots, c_m]$

Like the transformation **guess** i, when **guessRemoveUnique = true** and some (one-session or bit) secrecy queries are present, the transformation **guess** $x[c_1, \dots, c_m]$ first transforms the game G into G_{RU}, by replacing all proved $\mathsf{find[unique]}$ with find. Lemma 57 shows the soundness of this preliminary transformation.

Next, the transformation **guess** $x[c_1, \dots, c_m]$ transforms a game G into a game G' by guessing the value of a variable $x[c_1, \dots, c_m]$: it replaces the processes P under the definition of $x[c_1, \dots, c_m]$ with

$$\mathsf{if}\ x[c_1, \dots, c_m] = v_{\text{tested}}\ \mathsf{then}\ P\ \mathsf{else\ event\_abort}\ \text{bad\_guess}$$

and v_{tested} is a constant, which is the guessed value of $x[c_1, \dots, c_m]$. (At each definition of x, CryptoVerif must be able to determine whether it is a definition of $x[c_1, \dots, c_m]$ or not. The variable x must not be defined inside a term. In case **diff_constants = true**, the constant v_{tested} must not be considered different from other constants of the same type.)

In case there is a (one-session or bit) secrecy query, it uses instead

$$\mathsf{let}\ \mathit{guess\_x\_defined} = \text{true}\ \mathsf{in\ if}\ x[c_1, \dots, c_m] = v_{\text{tested}}\ \mathsf{then}\ P\ \mathsf{else\ event\_abort}\ \text{bad\_guess}$$

where *guess_x_defined* is a fresh variable, and we add *guess_x_defined* to the public variables of (one-session or bit) secrecy queries. That gives the adversary knowledge of whether the guessed variable is defined or not. This is useful because the adversary may need to swap its answer differently depending on whether the guessed variable is defined or not, so that the cases in which this variable is not defined always increase the probability of breaking (one-session or bit) secrecy.

When there are only correspondence queries, we can actually execute any code when $x[c_1, \dots, c_m]$ is different from the guessed value v_{tested}. In particular, we can execute P with $x[c_1, \dots, c_m]$ set to v_{tested}, which has the effect of replacing the definition of $x[c_1, \dots, c_m]$ with $\mathsf{let}\ x = v_{\text{tested}}$ and removing the test $x[c_1, \dots, c_m] = v_{\text{tested}}$.

To sum up, we also define a transformation **guess** $x[c_1, \dots, c_m]$ **no_test** that can be applied when there are only correspondence queries and when $x[c_1, \dots, c_m]$ is defined only by definitions of the form $\mathsf{let}\ x = M$. (This is the most useful case, since the definition of x can then be simplified.) This transformation replaces these definitions $\mathsf{let}\ x = M$ with $\mathsf{let}\ x = v_{\text{tested}}$ when M is a simple term and with $\mathsf{let}\ \mathit{ignore} = M\ \mathsf{in\ let}\ x = v_{\text{tested}}$ otherwise, where *ignore* is a fresh variable whose value is not used.

The transformation **guess** $x[c_1, \dots, c_m]$ **no_test** would not be valid in the presence of secrecy queries (at least not with the same probability), because before transformation the value of $x[c_1, \dots, c_m]$ may contain part of the secret variable and this value may leak, while after transformation, that leaks disappears and the variable may be perfectly secret for all values $x[c_1, \dots, c_m] = v_{\text{tested}}$.

Lemma 59 *The transformations* ***guess*** $x[c_1, \dots, c_m]$ *and* ***guess*** $x[c_1, \dots, c_m]$ ***no_test*** *require and preserve Properties 1, 2, 3, 4, and 5. They preserve Property 6.*

Suppose the game G is transformed into G' by the transformation ***guess*** $x[c_1, \dots, c_m]$ *or* ***guess*** $x[c_1, \dots, c_m]$ ***no_test****, where x is of type T and $T \neq \emptyset$.*

Let φ be the semantics of a correspondence. Let D be a disjunction of Shoup and non-unique events that does not contain S nor $\overline{\mathsf{S}}$. If $\mathsf{Bound}_{G'}(V, \varphi, D, p)$ and p is independent of the value of v_{tested}, then $\mathsf{Bound}_G(V, \varphi, D, |T|p)$.

Let sp be $\mathsf{1\text{-}ses.secr.}(y)$, $\mathsf{Secrecy}(y)$, or $\mathsf{bit\ secr.}(x)$. Then G is transformed into G' by the transformation ***guess*** $x[c_1, \dots, c_m]$*. If G' satisfies sp with public variables $V \cup \{\mathit{guess\_x\_defined}\}$ ($y \notin V$) up to probability p and p satisfies Property 7, then G satisfies sp with public variables V up to probability $|T| \times p$ (neglecting a small additional runtime of the context). If $\mathsf{Bound}_{G'}(V' \cup \{\mathit{guess\_x\_defined}\}, sp, \mathsf{NonUnique}_{G'}, p)$ and p satisfies Property 7, then $\mathsf{Bound}_G(V', sp, D_{\text{false}}, |T|p)$ (neglecting a small additional runtime of the context).*

Proof

Correspondences Let C be an evaluation context acceptable for G with any public variables that does not contain events used by φ or D.

$$\begin{aligned}
&\mathsf{Adv}_G(C,\varphi,D)\\
&\quad = \Pr[C[G] : (\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G,D}]\\
&\quad = \Pr[C[G] : (\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G,D} \wedge x[c_1,\dots,c_m] \text{ not defined}] +\\
&\qquad \sum_{v\in T} \Pr[C[G] : (\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G,D} \wedge x[c_1,\dots,c_m] = v]\\
&\quad \leq \sum_{v\in T} \Pr[C[G] : (\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G,D} \wedge (x[c_1,\dots,c_m] = v \vee x[c_1,\dots,c_m] \text{ not defined})]
\end{aligned}$$

Moreover,

$$\begin{aligned}
&\Pr[C[G'] : (\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G',D}]\\
&\quad \geq \Pr[C[G] : (\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G,D} \wedge (x[c_1,\dots,c_m] = v_{\text{tested}} \vee x[c_1,\dots,c_m] \text{ not defined})]
\end{aligned}$$

This property holds because G' behaves like G when $x[c_1,\dots,c_m]$ is not defined or $x[c_1,\dots,c_m] = v_{\text{tested}}$, in both transformations **guess** $x[c_1,\dots,c_m]$ and **guess** $x[c_1,\dots,c_m]$ **no_test**. So

$$\begin{aligned}
\mathsf{Adv}_G(C,\varphi,D) &\leq \sum_{v\in T} \Pr[C[G'] : (\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G',D}] \text{ for } v_{\text{tested}} = v\\
&\leq \sum_{v\in T} \mathsf{Adv}_{G'}(C,\varphi,D) \text{ for } v_{\text{tested}} = v
\end{aligned}$$

Since $\mathsf{Bound}_{G'}(V,\varphi,D,p)$, we have $\mathsf{Adv}_{G'}(C,\varphi,D) \leq p(C)$ and p is independent of the value of v_{tested}, so we obtain $\mathsf{Adv}_G(C,\varphi,D) \leq |T| \times p(C)$. Therefore $\mathsf{Bound}_G(V,\varphi,D,|T|p)$.

(One-session or bit) secrecy Let C be an evaluation context acceptable for $C_{sp}[G]$ with public variables V that does not contain S nor $\overline{\mathsf{S}}$. We have

$$\begin{aligned}
&\mathsf{Adv}^{sp}_G(C)\\
&\quad = \Pr[C[C_{sp}[G]] : \mathsf{S}] - \Pr[C[C_{sp}[G]] : \overline{\mathsf{S}}]\\
&\quad = \Pr[C[C_{sp}[G]] : \mathsf{S} \wedge x[c_1,\dots,c_m] \text{ not defined}] - \Pr[C[C_{sp}[G]] : \overline{\mathsf{S}} \wedge x[c_1,\dots,c_m] \text{ not defined}]\\
&\quad + \sum_{v=1}^{|T|} \Pr[C[C_{sp}[G]] : \mathsf{S} \wedge x[c_1,\dots,c_m] = v] - \Pr[C[C_{sp}[G]] : \overline{\mathsf{S}} \wedge x[c_1,\dots,c_m] = v]
\end{aligned}$$

From the adversary C, we define four adversaries C' that output b'' instead of b' on channel c'_s (c''_s for bit secrecy), where $b'' = \mathsf{if}\ \mathsf{defined}(\mathit{guess\_x\_defined})\ \mathsf{then}\ f_1(b')\ \mathsf{else}\ f_2(b')$ where $f_1(b')$ is either b' or $\neg b'$, and similarly for f_2, and consider the adversary $C'_{\max,v_{\text{tested}}}$ among those four that yields the maximum $\mathsf{Adv}^{sp}_{G'}(C')$. Changing b' into $\neg b'$ swaps the events S and $\overline{\mathsf{S}}$, and therefore swaps their probabilities. Hence, for this adversary $C'_{\max,v_{\text{tested}}}$,

$$\begin{aligned}
&\mathsf{Adv}^{sp}_{G'}(C'_{\max,v_{\text{tested}}})\\
&\quad = |\Pr[C[C_{sp}[G]] : \mathsf{S} \wedge x[c_1,\dots,c_m] = v_{\text{tested}}] - \Pr[C[C_{sp}[G]] : \overline{\mathsf{S}} \wedge x[c_1,\dots,c_m] = v_{\text{tested}}]|\\
&\quad + |\Pr[C[C_{sp}[G]] : \mathsf{S} \wedge x[c_1,\dots,c_m] \text{ not defined}] - \Pr[C[C_{sp}[G]] : \overline{\mathsf{S}} \wedge x[c_1,\dots,c_m] \text{ not defined}]|
\end{aligned}$$

Since G' satisfies sp with public variables $V \cup \{\mathit{guess\_x\_defined}\}$ ($y \notin V$) up to probability p and $C'_{\max,v_{\text{tested}}}$ is an evaluation context acceptable for $C_{sp}[G']$ with public variables $V \cup$

$\{guess\_x\_defined\}$ that does not contain S nor $\overline{\mathsf{S}}$, we have $\mathsf{Adv}^{sp}_{G'}(C'_{\max,v_{\text{tested}}}) \leq p(C'_{\max,v_{\text{tested}}})$. So we have

$$\begin{aligned}
\sum_{v_{\text{tested}}=1}^{|T|} p(C'_{\max,v_{\text{tested}}}) &\geq \sum_{v_{\text{tested}}=1}^{|T|} \mathsf{Adv}^{sp}_{G'}(C'_{\max,v_{\text{tested}}}) \\
&\geq \sum_{v_{\text{tested}}=1}^{|T|} \left| \begin{array}{l} \Pr[C[C_{sp}[G]] : \mathsf{S} \wedge x[c_1,\dots,c_m] = v_{\text{tested}}] \\ \ - \Pr[C[C_{sp}[G]] : \overline{\mathsf{S}} \wedge x[c_1,\dots,c_m] = v_{\text{tested}}] \end{array} \right| \\
&\quad + |T| \times \left| \begin{array}{l} \Pr[C[C_{sp}[G]] : \mathsf{S} \wedge x[c_1,\dots,c_m] \text{ not defined}] \\ \ - \Pr[C[C_{sp}[G]] : \overline{\mathsf{S}} \wedge x[c_1,\dots,c_m] \text{ not defined}] \end{array} \right| \\
&\geq \mathsf{Adv}^{sp}_{G}(C)
\end{aligned}$$

Moreover, by Property 7, p is independent of the value of v_{tested} (since the type T is bounded) and $p(C)$ depends only on the runtime of C, the number of outputs C makes on the various channels (which determine replication bounds), and the length of bitstrings, so we have $p(C'_{\max,v_{\text{tested}}}) = p(C)$. (The additional runtime of the context can be neglected.) So $\mathsf{Adv}^{sp}_{G}(C) \leq |T|p(C)$. Therefore, G satisfies *sp* with public variables V up to probability $|T| \times p$. The proof of the second property for (one-session or bit) secrecy proceeds as for **guess** i in Lemma 58. □

5.1.19 guess_branch

Like the transformation **guess** i, when **guessRemoveUnique = true** and some (one-session or bit) secrecy queries are present, the transformation **guess_branch** μ first transforms the game G into G_{RU}, by replacing all proved find[unique] with find. Lemma 57 shows the soundness of this preliminary transformation.

Next, the transformation **guess_branch** μ guesses the branch taken by a branching instruction (if, let, find) at program point μ. The program point μ is designated as explained in Section 5.1.10. The instruction at μ must be executed at most once (either because it is not under replication or because this is proved by CryptoVerif, showing that two executions with distinct replication indices lead to a contradiction: $\mathcal{F}_\mu \cup \mathcal{F}_\mu\{\widetilde{i'}/\widetilde{i}\} \cup \{\widetilde{i'} \neq \widetilde{i}\}$ yields a contradiction, where $\widetilde{i}$ are the current replication indices at μ and $\widetilde{i'}$ are fresh replication indices, using a mode of the equational prover of Section 3.3 that does not allow elimination of collisions, so that this property is proved without probability loss). Suppose this instruction has k branches.

We consider a game G and define transformed games G'_j ($0 \leq j < k$) in which branch j of the instruction at μ is kept and all other branches are replaced with event_abort bad_guess.

In case there is a (one-session or bit) secrecy query, let *guess_br_defined* = true in is added before the instruction at μ where *guess_br_defined* is a fresh variable, and we add *guess_br_defined* to the public variables of (one-session or bit) secrecy queries. That gives the adversary knowledge of whether the instruction at μ is executed or not. This is useful because the adversary may need to swap its answer differently depending on whether that instruction is executed or not, so that the cases in which that instruction is not executed always increase the probability of breaking (one-session or bit) secrecy.

When there are only correspondence queries, we can actually execute any code when the taken branch is different from the guessed one. In particular, we can execute the same code as in the tested branch, which has the effect of removing the test at μ when that test is if. (The tests find and let with pattern-matching have additional effects: guaranteeing the definition of variables for find; defining variables for let. In general, that prevents their removal.)

To sum up, we also define a transformation **guess_branch** μ **no_test** that can be applied when there are only correspondence queries and the instruction at μ is $\mathsf{if}\ M\ \mathsf{then}\ P_1\ \mathsf{else}\ P_0$. This transformation defines two transformed games G'_j ($j \in \{0,1\}$) in which the instruction at μ is replaced with P_j when M is a simple term and with $\mathsf{let}\ \mathit{ignore} = M\ \mathsf{in}\ P_j$ otherwise, where *ignore* is a fresh variable whose value is not used.

The transformation **guess_branch** μ **no_test** would not be valid in the presence of secrecy queries (at least not with the same probability), because before transformation the test at μ may make the value of M leak, which can reveal for instance one bit of the secret variable, while after transformation, that leaks disappears and the variable may be perfectly secret both when P_0 and when P_1 are executed.

Lemma 60 *The transformations **guess_branch** μ and **guess_branch** μ **no_test** require and preserve Properties 1, 2, 3, 4, and 5. They preserve Property 6.*

*Suppose the game G is transformed into games G'_j ($0 \leq j < k$) by the transformation **guess_branch** μ or **guess_branch** μ **no_test**.*

Let φ be the semantics of a correspondence. Let D be a disjunction of Shoup and non-unique events that does not contain S nor $\overline{\mathsf{S}}$. If for all $0 \leq j < k$, $\mathsf{Bound}_{G'_j}(V, \varphi, D, p_j)$, then $\mathsf{Bound}_G(V, \varphi, D, \sum_{j=0}^{k-1} p_j)$.

*Let sp be $\mathsf{1\text{-}ses.secr.}(y)$, $\mathsf{Secrecy}(y)$, or $\mathsf{bit\ secr.}(x)$. Then G is transformed into G'_j by the transformation **guess_branch** μ. If G'_j satisfies sp with public variables $V \cup \{\mathit{guess\_br\_defined}\}$ ($y \notin V$) up to probability p_j for $0 \leq j < k$ and the probabilities p_j satisfy Property 7, then G satisfies sp with public variables V up to probability $\sum_{j=0}^{k-1} p_j$ (neglecting a small additional runtime of the context). If for all $0 \leq j < k$, $\mathsf{Bound}_{G'_j}(V' \cup \{\mathit{guess\_br\_defined}\}, sp, \mathsf{NonUnique}_{G'_j}, p_j)$ and the probabilities p_j satisfy Property 7, then $\mathsf{Bound}_G(V', sp, D_{\mathrm{false}}, \sum_{j=0}^{k-1} p_j)$ (neglecting a small additional runtime of the context).*

Proof

Correspondences Let C be an evaluation context acceptable for G with any public variables that does not contain events used by φ or D.

$$\begin{aligned}
\mathsf{Adv}_G(C, \varphi, D) &= \Pr[C[G] : (\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G,D}] \\
&= \Pr[C[G] : (\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G,D} \wedge \mu \text{ not executed}] + \\
&\quad \sum_{j=0}^{k-1} \Pr[C[G] : (\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G,D} \wedge \text{branch } j \text{ is taken at } \mu] \\
&\leq \sum_{j=0}^{k-1} \Pr\left[\begin{array}{l} C[G] : (\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G,D} \wedge \\ \qquad (\mu \text{ not executed} \vee \text{branch } j \text{ is taken at } \mu)\end{array}\right] \\
&\leq \sum_{j=0}^{k-1} \Pr[C[G'_j] : (\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G'_j,D}] \qquad (26) \\
&\leq \sum_{j=0}^{k-1} \mathsf{Adv}_{G'_j}(C, \varphi, D) \\
&\leq \sum_{j=0}^{k-1} p_j(C) \qquad \text{since } \mathsf{Bound}_{G'_j}(V, \varphi, D, p_j)
\end{aligned}$$

The step (26) is valid because G'_j behaves like G when μ is not executed or branch j is taken at μ, in both transformations **guess_branch** μ and **guess_branch** μ **no_test**. Therefore, we obtain $\mathsf{Bound}_G(V, \varphi, D, \sum_{j=0}^{k-1} p_j)$.

(One-session or bit) secrecy Let C be an evaluation context acceptable for $C_{sp}[G]$ with public variables V that does not contain S nor $\overline{\mathsf{S}}$. We have

$$\begin{aligned}
&\mathsf{Adv}_G^{sp}(C)\\
&= \Pr[C[C_{sp}[G]] : \mathsf{S}] - \Pr[C[C_{sp}[G]] : \overline{\mathsf{S}}]\\
&= \Pr[C[C_{sp}[G]] : \mathsf{S} \wedge \mu \text{ not executed}] - \Pr[C[C_{sp}[G]] : \overline{\mathsf{S}} \wedge \mu \text{ not executed}]\\
&+ \sum_{j=0}^{k-1} \Pr[C[C_{sp}[G]] : \mathsf{S} \wedge \text{branch } j \text{ is taken at } \mu] - \Pr[C[C_{sp}[G]] : \overline{\mathsf{S}} \wedge \text{branch } j \text{ is taken at } \mu]
\end{aligned}$$

From the adversary C, we define four adversaries C' that output b'' instead of b' on channel c'_s (c''_s for bit secrecy), where $b'' = \mathsf{if}\ \mathsf{defined}(\mathit{guess\_br\_defined})\ \mathsf{then}\ f_1(b')\ \mathsf{else}\ f_2(b')$ where $f_1(b')$ is either b' or $\neg b'$, and similarly for f_2, and consider the adversary $C'_{\max,j}$ among those four that yields the maximum $\mathsf{Adv}_{G'_j}^{sp}(C')$. Changing b' into $\neg b'$ swaps the events S and $\overline{\mathsf{S}}$, and therefore swaps their probabilities. Hence, for this adversary $C'_{\max,j}$,

$$\begin{aligned}
&\mathsf{Adv}_{G'_j}^{sp}(C'_{\max,j})\\
&= |\Pr[C[C_{sp}[G]] : \mathsf{S} \wedge \text{branch } j \text{ is taken at } \mu] - \Pr[C[C_{sp}[G]] : \overline{\mathsf{S}} \wedge \text{branch } j \text{ is taken at } \mu]|\\
&+ |\Pr[C[C_{sp}[G]] : \mathsf{S} \wedge \mu \text{ not executed}] - \Pr[C[C_{sp}[G]] : \overline{\mathsf{S}} \wedge \mu \text{ not executed}]|
\end{aligned}$$

So

$$\begin{aligned}
&\sum_{j=0}^{k-1} \mathsf{Adv}_{G'_j}^{sp}(C'_{\max,j})\\
&\geq \sum_{j=0}^{k-1} \left| \begin{array}{l} \Pr[C[C_{sp}[G]] : \mathsf{S} \wedge \text{branch } j \text{ is taken at } \mu] \\ \ - \Pr[C[C_{sp}[G]] : \overline{\mathsf{S}} \wedge \text{branch } j \text{ is taken at } \mu] \end{array} \right|\\
&\quad + \sum_{j=0}^{k-1} |\Pr[C[C_{sp}[G]] : \mathsf{S} \wedge \mu \text{ not executed}] - \Pr[C[C_{sp}[G]] : \overline{\mathsf{S}} \wedge \mu \text{ not executed}]|\\
&\geq \mathsf{Adv}_G^{sp}(C)
\end{aligned}$$

Since G'_j satisfies sp with public variables $V \cup \{\mathit{guess\_br\_defined}\}$ up to probability p_j and $C'_{\max,j}$ is an evaluation context acceptable for $C_{sp}[G'_j]$ with public variables $V \cup \{\mathit{guess\_br\_defined}\}$ that does not contain S nor $\overline{\mathsf{S}}$, we have $\mathsf{Adv}_{G'_j}^{sp}(C'_{\max,j}) \leq p_j(C'_{\max,j})$. Moreover, by Property 7, $p_j(C)$ depends only on the runtime of C, the number of outputs C makes on the various channels (which determine replication bounds), and the length of bitstrings, so we have $p_j(C'_{\max,j}) = p_j(C)$. (The additional runtime of the context can be neglected.) So $\mathsf{Adv}_G^{sp}(C) \leq \sum_{j=0}^{k-1} p_j(C)$. Therefore, G satisfies sp with public variables V up to probability $\sum_{j=0}^{k-1} p_j$. The proof of the second property for (one-session or bit) secrecy proceeds as for **guess** i in Lemma 58. □

5.1.20 global_dep_anal [24]

The global dependency analysis **global_dep_anal** x 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 random choices $\mathsf{new}\ x : T$ 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 T' has at most $|T|/|T'|$ antecedents by f. In particular, this is true in the following two cases:

- f is such that $f(x)$ is uniformly distributed in T' if x is uniformly distributed in 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 T.) Precisely, when $x_k \in S_{\text{def}}$ is defined by a pattern-matching $\mathsf{let}\ f'(x_1, \dots, x_n) = M\ \mathsf{in}\ P\ \mathsf{else}\ P'$, 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 $f = {f'}_k^{-1}{}_{|\text{im}\ f'}$.

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 3.1 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 $S_{\text{def}}, S_{\text{dep}}$.

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 random choices. (We know that variables different from x and defined by random choices do not depend on x, so in the absence of more precise information, we can set $S_{\text{dep}} = S'$.)

We say that only_dep$(x) = S$ when intuitively, only variables in S depend on x, and the adversary cannot see the value of x. Formally, 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 $\mathsf{let}\ z : T = M\ \mathsf{in}\ P$, then $z \in S$.

- Variables in S may occur in $\mathsf{defined}$ conditions of find but only at the root of them.

- All terms M_j in processes $\mathsf{find}\ (\bigoplus_{j=1}^{m} \widetilde{u_j}[\widetilde{i}] \leq \widetilde{n_j}\ \mathsf{suchthat}\ \mathsf{defined}(M_{j1}, \ldots, M_{jl_j}) \wedge M_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P'$ 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 $f_1(\ldots f_k(x[\widetilde{M}]))$ and M_2 does not depend on $f_1(\ldots f_k(x[\widetilde{M}]))$, so the equality $M_1 = M_2$ happens for a single value of $f_1(\ldots f_k(x[\widetilde{M}]))$, which yields a negligible probability because $f_1, \ldots, f_k$ 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.

Lemma 61 *The transformation **global_dep_anal** requires and preserves Properties 1, 2, 3, 4, 5, and 6. If transformation **global_dep_anal** transforms G into G′, then* $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : G, D, EvUsed \xrightarrow{V}_p G', D, EvUsed$*, where p is an upper bound on the probability that required equalities do not hold.*

5.1.21 simplify [24, 25]

We use the following transformations in order to simplify games. These transformations exploit the information collected as explained in Section 3.

1. 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 3.1 and 3.3) 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.

2. When setting **inferUnique** is true, CryptoVerif tries to prove uniqueness of $\mathsf{find}[\mathsf{unique}_e]$, as in transformation **prove_unique** (Section 5.1.4).

3. If $P = \mathsf{find}[\mathit{unique?}]\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}$ $\mathsf{suchthat}$ $\mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j$ then $P_j)$ else P', M_j does not contain $\mathsf{event\_abort}$ nor unproved $\mathsf{find}[\mathsf{unique}_e]$, $u_{jk}[\widetilde{i}]$ reduces into M' by user-defined rewrite rules knowing $\mathcal{F}_{P_j}$ (see Sections 3.1 and 3.3) and the rewrite rules obtained from $\mathcal{F}_{P_j}$, and u_{jk} does not occur in M', then u_{jk} is removed from the j-th branch of this find, i_{jk} is replaced with $M'\{i_{j'k'}/u_{j'k'}, j' \leq m, k' \leq m_j\}$ in $M_{j1}, \dots, M_{jl_j}, M_j$ and P_j is replaced with let $u_{jk}[\widetilde{i}] : [1, n_{jk}] = M'$ 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'.)

4. Suppose that $P = \mathsf{find}[\mathit{unique?}]\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}$ $\mathsf{suchthat}$ $\mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j$ then $P_j)$ 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 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 $N_1, \dots, N_l$ and the current replication indices at the definitions of x. Then the j-th branch of the find is removed. (In this case, $x[N_1, \dots, N_l]$ cannot be defined at P, so the j-th branch of the find cannot be taken.)

5. Suppose that $P = \mathsf{find}[\mathit{unique?}]\ (\bigoplus_{j=1}^{m} u_{j1}[\widetilde{i}] = i_{j1} \leq n_{j1}, \dots, u_{jm_j}[\widetilde{i}] = i_{jm_j} \leq n_{jm_j}$ $\mathsf{suchthat}$ $\mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j$ then $P_j)$ 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 $k \leq \min(l, l')$, $x \neq x'$, 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.

6. If $P = \mathsf{find}[\mathit{unique?}]\ (\bigoplus_{j=1}^{m} \widetilde{u_j}[\widetilde{i}] = \widetilde{i_j} \leq \widetilde{n_j}$ $\mathsf{suchthat}$ $\mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j$ then $P_j)$ else P' and $\mathcal{F}_{P_j}$ yields a contradiction, then the j-th branch of the find is removed if M_j does not contain $\mathsf{event\_abort}$ nor unproved $\mathsf{find}[\mathsf{unique}_e]$, and P_j is replaced with $\overline{\mathit{yield}}\langle\rangle$ if M_j contains $\mathsf{event\_abort}$ or some unproved $\mathsf{find}[\mathsf{unique}_e]$.

7. If $P = \mathsf{find}[\mathit{unique?}]$ else P', then P is replaced with P'.

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

9. If $P = \mathsf{find}[\mathit{unique?}]\ \widetilde{u}[\widetilde{i}] = \widetilde{i'} \leq \widetilde{n}$ $\mathsf{suchthat}$ $\mathsf{defined}(M_1, \dots, M_l) \wedge M$ then P_1 else P', $[\mathit{unique?}]$ is not $[\mathsf{unique}_e]$ for some non-unique event e that is not proved yet to have negligible probability, $\mathcal{F}_{P'}$ yields a contradiction, M is simple (so M never aborts), 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.)

10. If $P = \mathsf{find}[\mathit{unique?}]\ (\bigoplus_{j=1}^{m} \widetilde{u_j}[\widetilde{i}] \leq \widetilde{n_j}$ $\mathsf{suchthat}$ $\mathsf{defined}(M_{j1}, \dots, M_{jl_j}) \wedge M_j$ then $\overline{\mathit{yield}}\langle\rangle)$ else $\overline{\mathit{yield}}\langle\rangle$, $[\mathit{unique?}]$ is not $[\mathsf{unique}_e]$ for some non-unique event e that is not proved yet to have negligible probability, the terms M_j do not contain $\mathsf{event\_abort}$ nor unproved $\mathsf{find}[\mathsf{unique}_e]$, and the variables in $\widetilde{u_j}$ are not used outside P and are not in V, then P is replaced with $\overline{\mathit{yield}}\langle\rangle$.

11. The defined conditions of find are updated so that Invariant 2 is satisfied. (When such a defined condition guarantees that M is defined, $\mathsf{defined}(M)$ implies $\mathsf{defined}(M')$, 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.)

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

13. If one of the then branches of a find[unique] always succeeds and the conditions of this find[unique] do not contain event_abort nor unproved find[unique$_e$], then we keep only that branch.

    Indeed, the other branches are never taken: the conditions of this find[unique] never abort in traces counted in the probability, and the find itself aborts when there are several successful choices.

14. We reorganize a find[unique] that occurs in a then branch of a find[unique]: we transform

$$\mathsf{find[unique]}\ (\bigoplus_{j=1}^{k} \tilde{u}_j = \tilde{i}_j \leq \tilde{n}_j\ \mathsf{suchthat}\ c_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P$$

where $P_{j_0} = \mathsf{find[unique]}\ (\bigoplus_{j'=1}^{k'} FB_{j'})\ \mathsf{else}\ P''_{j_0}$ into

$$\begin{aligned}
&\mathsf{find[unique]}\ (\bigoplus_{j=1,\dots,k; j \neq j_0} \tilde{u}_j = \tilde{i}_j \leq \tilde{n}_j\ \mathsf{suchthat}\ c_j\ \mathsf{then}\ P_j)\\
&\oplus (\bigoplus_{j'=1}^{k'} \bigoplus_{(\tilde{u}'=\tilde{i}'\leq\tilde{n}'\ \mathsf{suchthat}\ c'\ \mathsf{then}\ P')\in \mathsf{b}(FB_{j'}, L_{j'})}\\
&\qquad \tilde{u}_{j_0} = \tilde{i}_{j_0} \leq \tilde{n}_{j_0}, \tilde{u}' = \tilde{i}' \leq \tilde{n}'\ \mathsf{suchthat}\ c_{j_0} \wedge c'\ \mathsf{then}\ P')\\
&\mathsf{else}\ \mathsf{find[unique]}\ \tilde{u}_{j_0} = \tilde{i}_{j_0} \leq \tilde{n}_{j_0}\ \mathsf{suchthat}\ c_{j_0}\ \mathsf{then}\ P''_{j_0}\ \mathsf{else}\ P
\end{aligned}$$

where
– either for all $j \leq k$, c_j does not contain event_abort nor unproved find[unique$_e$] or c_{j_0} never aborts (this is true in particular when c_{j_0} does not contain event_abort nor proved or unproved find[unique$_e$])
– $\tilde{i}$ are the current replication indices at the transformation point
– for all j', $FB_{j'} = (\tilde{u}'_{j'} = \tilde{i}'_{j'} \leq \tilde{n}'_{j'}\ \mathsf{suchthat}\ c'_{j'}\ \mathsf{then}\ P'_{j'})$, $c'_{j'} = \mathsf{defined}(\tilde{M}_{j'}) \wedge \dots$, $c'_{j'}$ never aborts, and $L_{j'}$ is the list of $x[\tilde{N}]$ subterm of $\tilde{M}_{j'}$ with $x \in \tilde{u}_{j_0}$ and $\tilde{N} \neq \tilde{i}$, ordered by increasing size
– $\mathsf{b}(\tilde{u}' = \tilde{i}' \leq \tilde{n}'\ \mathsf{suchthat}\ c'\ \mathsf{then}\ P', []) = \{\tilde{u}' = \tilde{i}' \leq \tilde{n}'\ \mathsf{suchthat}\ c'\{\tilde{i}_{j_0}/\tilde{u}_{j_0}\}\ \mathsf{then}\ P'\}$
– $\mathsf{b}(FB, x[\tilde{N}] :: L) = \mathsf{b}(FB, L) \cup \{\tilde{u}' = \tilde{i}' \leq \tilde{n}'\ \mathsf{suchthat}\ c'\{i/x[\tilde{N}]\} \wedge \tilde{N} = \tilde{i}\ \mathsf{then}\ P' \mid (\tilde{u}' = \tilde{i}' \leq \tilde{n}'\ \mathsf{suchthat}\ c'\ \mathsf{then}\ P') \in \mathsf{b}(FB, L)\}$ where $i = x\{\tilde{i}_{j_0}/\tilde{u}_{j_0}\}$.

    The function b takes into account that, before the transformation, $\tilde{u}_{j_0}[\tilde{i}]$ is defined when we test $\mathsf{defined}(\tilde{M}_{j'})$ in $FB_{j'}$, while after the transformation, $\tilde{u}_{j_0}[\tilde{i}]$ is not defined yet when we perform this test. Furthermore, the value of $\tilde{u}_{j_0}[\tilde{i}]$ will be $\tilde{i}_{j_0}$. Therefore, 1) when we access $x[\tilde{i}]$ in $c'_{j'} = \mathsf{defined}(\tilde{M}_{j'}) \wedge \dots$, we replace this access with i, where $i = x\{\tilde{i}_{j_0}/\tilde{u}_{j_0}\}$; this is done in $\mathsf{b}(FB, [])$ by the substitution $\{\tilde{i}_{j_0}/\tilde{u}_{j_0}\}$; and 2) when we access $x[\tilde{N}]$ in $c'_{j'} = \mathsf{defined}(\tilde{M}_{j'}) \wedge \dots$ for $\tilde{N}$ not syntactically equal to $\tilde{i}$, we need to distinguish two cases: either at runtime $\tilde{N} = \tilde{i}$ and we replace this access with i (second part of the union in $\mathsf{b}(FB, x[\tilde{N}] :: L)$), or at runtime $\tilde{N} \neq \tilde{i}$ and we continue using $x[\tilde{N}]$ (first part of the union in $\mathsf{b}(FB, x[\tilde{N}] :: L)$). The array accesses $x[\tilde{N}]$ in $L_{j'}$ are ordered by increasing size because, in case of nested array accesses, we need to handle the bigger array access first (so it must occur last in the list), because after substitution of the smaller one with i, we would not recognize the bigger one.

This transformation cannot be performed when the outer find is not unique because it might change the probability of taking each branch. Moreover, we tried performing such a transformation when the inner find is not unique (in this case, after transformation, the outer find is not unique), but it had a negative impact in some examples. Furthermore, in the latter case, the transformation can be performed manually by inserting the desired outer find and simplifying the game: CryptoVerif will remove the useless branches of find.

The conditions c_{j_0} and $c'_{j'}$ are conjunctions of defined conditions and a term. In the current implementation, the transformation is not performed when the term in c_{j_0} is false (the branch of find will be removed by another transformation), and the branches such that the terms in $c'_{j'}$ are false are first removed. Furthermore, the transformation is performed only when one of the following conditions holds: the terms in c_{j_0} and all $c'_{j'}$ are simple, or the term in c_{j_0} is true and all $c'_{j'}$ never abort, or the terms in $c'_{j'}$ are all true and c_{j_0} never aborts. With the usual simplification of $\wedge$ true, this guarantees that the transformed game satisfies Property 6. Moreover, this implies the abortion conditions (c_{j_0} and all $c'_{j'}$ never abort).

After this transformation, we advise renaming the variables $\tilde{u}_{j_0}$ to distinct names, since they now have multiple definitions.

15. We reorganize a find[unique] that occurs in a condition of a find: we transform

$$\mathsf{find}[\mathit{unique}?]\ (\bigoplus_{j=1}^{k} \tilde{u}_j = \tilde{i}_j \leq \tilde{n}_j \ \mathsf{suchthat}\ c_j\ \mathsf{then}\ P_j)\ \mathsf{else}\ P$$

where

$$c_{j_0} = \mathsf{defined}(\tilde{M}') \wedge \mathsf{find}[\mathsf{unique}](\bigoplus_{j'=1}^{k'} \tilde{u}'_{j'} = \tilde{i}'_{j'} \leq \tilde{n}'_{j'}\ \mathsf{suchthat}\ c'_{j'}\ \mathsf{then}\ M'_{j'})\ \mathsf{else}\ \mathrm{false}$$

for all $j' \leq k'$, $M'_{j'}$ never aborts and either for all $j \leq k$, c_j does not contain event_abort nor unproved find[unique$_e$] or for all $j' \leq k'$, $c'_{j'}$ never aborts, into

$$\begin{aligned}
&\mathsf{find}[\mathit{unique}?]\ (\bigoplus_{j=1,\dots,k;j\neq j_0} \tilde{u}_j = \tilde{i}_j \leq \tilde{n}_j\ \mathsf{suchthat}\ c_j\ \mathsf{then}\ P_j)\\
&\oplus (\bigoplus_{j'=1}^{k'} \tilde{u}_{j_0} = \tilde{i}_{j_0} \leq \tilde{n}_{j_0}, \tilde{u}'_{j'} = \tilde{i}'_{j'} \leq \tilde{n}'_{j'}\ \mathsf{suchthat}\\
&\qquad\qquad \mathsf{defined}(\tilde{M}') \wedge c'_{j'} \wedge M'_{j'}\{\tilde{i}'_{j'}/\tilde{u}'_{j'}\}\ \mathsf{then}\ P_{j_0})\\
&\mathsf{else}\ P
\end{aligned}$$

The indication [*unique*?] corresponds to either [unique] or empty. The find is marked [unique] after transformation if the outer find was [unique] before transformation.

The variables $\tilde{u}'_{j'}$ are defined inside the condition of a find so by Invariant 3, they have no array accesses. The transformation performed by function b above is therefore not needed here.

The conditions $c'_{j'}$ are conjunctions of defined conditions and a term. The current implementation first removes the branches such that one of the following two conditions holds: the terms in $c'_{j'}$ are false or $M'_{j'}$ is false and $c'_{j'}$ never aborts. Furthermore, the transformation is performed only when for all j', one of the following conditions holds: the term in $c'_{j'}$ and $M_{j'}$ are simple, or the term in $c'_{j'}$ is true and $M'_{j'}$ never aborts, or $M'_{j'}$ is true or false and the term in $c'_{j'}$ never aborts. With the usual simplification of $\wedge$ true and $\wedge$ false, this guarantees that the transformed game satisfies Property 6. Moreover, this implies the abortion conditions ($c'_{j'}$ and $M'_{j'}$ never abort).

The simplification is iterated at most **maxIterSimplif** times. The iteration stops earlier in case a fixpoint is reached.

Lemma 62 *The transformation **simplify** requires and preserves Properties 1, 2, 3, 4, and 5. It preserves 6. If transformation **simplify** transforms G into G', then $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : G, D, EvUsed \xrightarrow{V}_p G', D, EvUsed$, where p is an upper bound on the probability that required equalities do not hold.*

5.1.22 all_simplify

all_simplify perform several simplifications on the game, as if

- **simplify**,
- **move all** if **autoMove = true**,
- **remove_assign useless** if **autoRemoveAssignFindCond = false**, **remove_assign findcond** if **autoRemoveAssignFindCond = true**,
- **SArename random** if **autoSARename = true**,
- and **merge_branches** if **autoMergeBranches = true**

had been called.

5.1.23 success simplify

The transformation **success simplify** is a combination of **success** (Section 4) and **simplify** (Section 5.1.21), with the following addition. First, in the **success** step, the command **success simplify** collects information that is known to be true when the adversary manages to break at least one of the desired properties. Then, the first iteration of the **simplify** step removes parts of the game that contradict this information and replaces them with $\mathsf{event\_abort}$ adv_loses.

In more detail, **success simplify** collects a set $\mathcal{L}^-$ of $(\mathcal{V}, \mathcal{F})$ and a set of formulas $\mathcal{L}^+$. If the adversary breaks a desired security property, then either there exists a set of facts $\mathcal{F}$ in $\mathcal{L}^-$ that holds or there exists a formula in $\mathcal{L}^+$ that holds. The sets of facts $\mathcal{L}^-$ correspond to cases in which the proof of the security property failed; their probability may be high. The associated variables $\mathcal{V}$ are replication indices and non-process variables that occur in $\mathcal{F}$. The formulas in $\mathcal{L}^+$ correspond to cases in which the security property was proved (up to a certain probability); these formulas are negations of the formulas that prove the security property in the considered case; the probability that they hold is bounded by the equational prover of CryptoVerif. The contents of $\mathcal{L}^+$ does not influence the game obtained after the transformation. It is useful to compute the probability difference coming from the transformation. The sets $\mathcal{L}^-$ and $\mathcal{L}^+$ are computed as follows:

- In case there is an indistinguishability query, or a (one-session or bit) secrecy query on a variable x not defined only by new or by assignments of variables defined by new, or a correspondence query $\forall \widetilde{x} : \widetilde{T}; \psi \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi$ with some event e in ψ such that the game contains $\mathsf{event}\ e(\widetilde{M})$ and some term in $\widetilde{M}$ is not simple, no information is collected at all and **simplify** is not performed.

- For each correspondence query $\mathsf{event}(e) \Rightarrow \text{false}$ where e is a non-unique event, for every μ that executes $\mathsf{event}(e)$, for every c, $(\theta' I_\mu, \theta' \mathcal{F}^0_{\mathsf{event}(e),\mu,c})$ is added to $\mathcal{L}^-$, for some θ' renaming of I_μ to fresh replication indices. (Indeed, in order to break $\mathsf{event}(e) \Rightarrow \text{false}$, event e must be executed, so the facts $\mathcal{F}^0_{\mathsf{event}(e),\mu,c}$ at some execution of event e hold.)

- For each other correspondence query $\forall \widetilde{x} : \widetilde{T}; F_1 \wedge \dots \wedge F_m \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi$, let $\varphi = [\![\forall \widetilde{x} : \widetilde{T}; F_1 \wedge \dots \wedge F_m \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi]\!]$ and $\mathcal{S}_0 = \{(\mu_1, c_1, \dots, \mu_m, c_m) \mid \forall j \leq m, \mu_j$ executes F_j and c_j is a case for $\mathcal{F}_{\mu_j,c_j}\}$; by trying to prove the correspondence, we build a subset $\mathcal{S}_1$ of $\mathcal{S}_0$, a family of substitutions θ, and a pseudo-formula $\mathcal{C}$ such that $\text{prove}^\varphi(\mathcal{C}, \theta, \mathcal{S}_1)$. ($\theta$ and $\mathcal{C}$ are computed incrementally on the successful cases in the proof of the correspondence.)

  For all $(\mu_1, c_1, \dots, \mu_m, c_m) \in \mathcal{S}_0 \setminus \mathcal{S}_1$ (**success** fails to prove the correspondence for those cases):

  - If the query is $\forall \widetilde{x} : \widetilde{T}; \mathsf{event}(e(\widetilde{N})) \Rightarrow \text{false}$ and the process or term at μ_1 is $\mathsf{event}\ e(\widetilde{M}); \dots$, then $(\widetilde{x} \cup \theta' I_{\mu_1}, \theta' \mathcal{F}_{\mu_1,c_1} \cup \{\mathit{lastdefprogrampoint}(\mu_1, \theta' I_{\mu_1}), \widetilde{N} = \theta' \widetilde{M}\})$ is added to $\mathcal{L}^-$, where θ' is a renaming of I_{μ_1} to fresh replication indices. (We can stop the trace just after event e without changing the truth of the query, and that is more precise because we can use the *elsefind* facts at e.)
  - Otherwise, $(\widetilde{x} \cup \bigcup_{j=1}^m \theta_j I_{\mu_j}, \bigcup_{j=1}^m \theta_j \mathcal{F}_{F_j,\mu_j,c_j})$ is added to $\mathcal{L}^-$, where for $j \leq m$, θ_j is a renaming of I_{μ_j} to fresh replication indices. (Indeed, in order to break the correspondence $\forall \widetilde{x} : \widetilde{T}; F_1 \wedge \dots \wedge F_m \Rightarrow \exists \widetilde{y} : \widetilde{T'}; \phi$, the events $F_1, \dots, F_m$ must be executed, so the facts $\mathcal{F}_{F_j,\mu_j,c_j}$ for $j \leq m$ that hold when $F_1, \dots, F_m$ are executed certainly hold when the correspondence is broken. In principle, we could add $\neg \exists \widetilde{y} : \widetilde{T'}; \phi$ to the facts $\bigcup_{j=1}^m \theta_j \mathcal{F}_{F_j,\mu_j,c_j}$ added to $\mathcal{L}^-$. However, we have no way to express universal quantification in general in known facts, so when $\widetilde{y}$ is not empty, we could not add $\forall \widetilde{y} : \widetilde{T'}; \neg\phi$ but would end up adding $\neg\phi$ which in fact means $\exists \widetilde{y} : \widetilde{T'}; \neg\phi$. That would remain sound assuming the types in $\widetilde{T'}$ are not empty, but would be weaker. Moreover, in practice, we end up having to distinguish precisely the case in which $\exists \widetilde{y} : \widetilde{T'}; \phi$ can be proved from the case in which it cannot, which can typically be done by inserting an appropriate find. We generally insert a Shoup event e in the else branch of that find, triggered when the correspondence cannot be proved, a case for which we want to bound the probability. After that, it remains to prove $\mathsf{event}(e) \Rightarrow \text{false}$: we apply **success simplify** to that correspondence. Adding $\neg\phi = \text{true}$ would not change anything for that correspondence, and we exploit that the condition of the inserted find is false at event e, which gives us more precise information than having added $\neg\phi$ for the initial correspondence.)

  Moreover, $\neg\{\![\text{prove}^\varphi(\mathcal{C}, \theta, \mathcal{S}_1)]\!\}$ is added to $\mathcal{L}^+$. (**success** proves the correspondence for the cases in $\mathcal{S}_1$.)

- For each secrecy, one-session secrecy, or bit secrecy query on a variable x defined only by new or by assignments of variables defined by new, let $\mathcal{S}_0 = \{\mu \mid \mu$ follows a definition of $x\}$ and $\mathcal{S}_1 = \{\mu \in \mathcal{S}_0 \mid \text{prove}^{\mathsf{1\text{-}ses.secr.}(x)}(\mu)\}$.

  For each $\mu \in \mathcal{S}_0 \setminus \mathcal{S}_1$ (**success** fails to prove one-session secrecy for those cases), $(\theta I_\mu, \theta \mathcal{F}_\mu)$ is added to $\mathcal{L}^-$, where θ is a renaming of I_μ to fresh replication indices. (Indeed, if secrecy, one-session secrecy, or bit secrecy of x is broken, a definition of x must have been executed, so the facts $\mathcal{F}_\mu$ at that definition hold.)

Moreover, $\neg\{\![\mathrm{prove}^{\text{1-ses.secr.}(x)}(\mathcal{S}_1)]\!\}$ is added to $\mathcal{L}^+$. (**success** proves one-session secrecy for the cases in $\mathcal{S}_1$.)

Additionally, if the considered query is a secrecy query, then for each $\mu_1, \mu_2 \in \mathcal{S}_0$, let $z_1[\widetilde{M_1}] = \mathrm{defRand}_{\mu_1}(x)$ and $z_2[\widetilde{M_2}] = \mathrm{defRand}_{\mu_2}(x)$. If $z_1 \neq z_2$, then the definitions at μ_1 and μ_2 are proved to be independent, and nothing is added to $\mathcal{L}^-$ nor $\mathcal{L}^+$. If $z_1 = z_2$, let $\widetilde{i}$ be the current replication indices at the definition of x, let θ_1 and θ_2 be two distinct renamings of $\widetilde{i}$ to fresh replication indices, let $\widetilde{i_1} = \theta_1\widetilde{i}$ and $\widetilde{i_2} = \theta_2\widetilde{i}$, let $\mathcal{F} = \theta_1\mathcal{F}_{\mu_1} \cup \theta_2\mathcal{F}_{\mu_2} \cup \{\theta_1\widetilde{M_1} = \theta_2\widetilde{M_2}, \widetilde{i_1} \neq \widetilde{i_2}\}$. If $\mathcal{F}$ yields a contradiction, then the definitions at μ_1 and μ_2 are proved to be independent up to a small probability, $\exists\widetilde{i_1}, \exists\widetilde{i_2}, \bigwedge\mathcal{F}$ is added to $\mathcal{L}^+$. Otherwise, $(\widetilde{i_1} \cup \widetilde{i_2}, \mathcal{F})$ is added to $\mathcal{L}^-$.

In the **simplify** step, the set $\mathcal{L}^-$ is used as follows: for each program point μ not in a condition of find, if for all $(\mathcal{V}, \mathcal{F}) \in \mathcal{L}^-$, $\mathcal{F}_\mu \cup \mathcal{F}$ yields a contradiction, then the code at μ is replaced with $\mathsf{event\_abort}$ adv_loses. (The reason why $\mathcal{V}$ is needed in the implementation is for the optimization of probabilities of collisions: we determine using which indices in $\mathcal{V}$ we get the smaller bound for the number of collisions.)

The probability that a security property is broken before the transformation and not after is then bounded by the probability that a modified program point μ is reached and the adversary breaks the property. If that breach corresponds to a case in $\mathcal{L}^+$, the probability of the breach itself is bounded by construction of $\mathcal{L}^+$. If that breach corresponds to a case in $\mathcal{L}^-$, the probability of the breach and reaching μ is bounded because for all $(\mathcal{V}, \mathcal{F}) \in \mathcal{L}^-$, $\mathcal{F}_\mu \cup \mathcal{F}$ yields a contradiction, which bounds the probability that the facts $\mathcal{F}_\mu \cup \mathcal{F}$ hold for some $(\mathcal{V}, \mathcal{F}) \in \mathcal{L}$, and $\mathcal{F}_\mu$ holds when μ is reached while some $\mathcal{F}$ in $\mathcal{L}$ holds when the adversary breaks the property. This is formalized by the following lemma.

Lemma 63 *The transformation **success simplify** requires and preserves Properties 1, 2, 3, 4, 5, and 6.*

*If transformation **success simplify** transforms G into G', the distinguisher D is a disjunction of Shoup and non-unique events, the property sp and the disjuncts in D correspond to active queries, $\mathcal{L}^- = \{(\mathcal{V}_j, \mathcal{F}_j) \mid j \in J\}$, the modified program points are μ_k for $k \in K$, $\mathcal{F}_k^{\mathrm{mod}} = \mathcal{F}_{\mu_k}$, for all evaluation contexts C acceptable for G, $\Pr[C[G] \preceq \left(\bigvee_{j\in J, k\in K} \exists I_{\mu_k}, \exists\mathcal{V}_j, \bigwedge\mathcal{F}_j \wedge \bigwedge\mathcal{F}_k^{\mathrm{mod}}\right) \vee (\bigvee\mathcal{L}^+)] \leq p(C)$, and $\mathsf{Bound}_{G'}(V, sp, D, p')$, then $\mathsf{Bound}_G(V, sp, D, p + p')$.*

Unfortunately, we cannot prove $\mathcal{D}, \mathcal{D}_{\mathsf{SNU}} : G, D, \mathit{EvUsed} \xrightarrow{V}_p G', D, \mathit{EvUsed} \cup \{\text{adv\_loses}\}$ for the transformation **success simplify**, because, in case of (one-session or bit) secrecy of a variable x, the inequality needed for this property may not hold: we need to take into account that, when x is not defined, traces that execute S and those that execute $\overline{\mathsf{S}}$ compensate in the computation of $\mathsf{Adv}_G(C, sp, D)$ in order to prove the soundness of this transformation. We cannot prove this soundness independently for $\Pr[C[G] : \mathsf{S}]$ and for $\Pr[C[G] : \neg\overline{\mathsf{S}}]$.

Moreover, in the implementation, for (one-session or bit) secrecy properties, a probability $2p(C)$ is added instead of just $p(C)$ as shown by the lemma above. The factor 2 is difficult to avoid because other simplifications are performed at the same time as described in the transformation **simplify** (Section 5.1.21), and the factor 2 is needed for these transformations.

Proof

Fact 1. Let φ be a correspondence not of the form $\forall\widetilde{x} \in \widetilde{T}, \mathsf{event}(e(\widetilde{N})) \Rightarrow$ false, or the correspondence $\mathsf{event}(e) \Rightarrow$ false for some Shoup event e. Let C be any evaluation context acceptable for G with public variables V that does not contain events used by φ. Let Tr be any full trace of $C[G]$ that does not execute any non-unique event of G and such that $\mathit{Tr} \vdash \neg\varphi$. Then $\mathit{Tr} \vdash \left(\bigvee_{j\in J} \exists \mathcal{V}_j, \bigwedge \mathcal{F}_j\right) \vee (\bigvee \mathcal{L}^+)$.

Proof of Fact 1. By Lemma 41, for any substitutions $\theta(\mu_1, c_1, \dots, \mu_m, c_m)$ equal to the identity on $\widetilde{x}$, for any pseudo-formula $\mathcal{C}$,

$$\mathit{Tr} \vdash \neg\{\![\mathrm{prove}^{\varphi}(\mathcal{C}, \theta, \mathcal{S}_0)]\!\}$$

So

$$\mathit{Tr} \vdash (\neg\{\![\mathrm{prove}^{\varphi}(\mathcal{C}, \theta, \mathcal{S}_1)]\!\}) \vee \left(\bigvee_{(\mu_1,c_1,\dots,\mu_m,c_m)\in \mathcal{S}_0\setminus\mathcal{S}_1} \neg\{\![\mathrm{prove}^{\varphi}(\mathcal{C}, \theta, \mu_1, c_1, \dots, \mu_m, c_m)]\!\}\right) .$$

Moreover, $\neg\{\![\mathcal{F} \models_{\theta}^{\mathcal{I},\mathcal{V},\mathcal{C}} \phi]\!\} \Rightarrow \exists \widetilde{z} \in \widetilde{T}'', \bigwedge \mathcal{F}$ where $\widetilde{z} = \mathcal{V}$ and $\widetilde{T}''$ are the types of these variables, by an easy induction on ϕ, so $\neg\{\![\mathrm{prove}^{\varphi}(\mathcal{C}, \theta, \mu_1, c_1, \dots, \mu_m, c_m)]\!\} \Rightarrow \exists \theta_1 I_{\mu_1}, \dots, \exists \theta_m I_{\mu_m}, \exists \widetilde{x} \in \widetilde{T}$, $\bigwedge \theta_1 \mathcal{F}_{F_1,\mu_1,c_1} \cup \dots \cup \theta_m \mathcal{F}_{F_m,\mu_m,c_m}$ where, for $j \leq m$, θ_j is a renaming of I_{μ_j} to fresh replication indices. So

$$\begin{aligned} \mathit{Tr} \vdash &(\neg\{\![\mathrm{prove}^{\varphi}(\mathcal{C}, \theta, \mathcal{S}_1)]\!\}) \vee \\ &\left(\bigvee_{(\mu_1,c_1,\dots,\mu_m,c_m)\in \mathcal{S}_0\setminus\mathcal{S}_1} \exists \theta_1 I_{\mu_1}, \dots, \exists \theta_m I_{\mu_m}, \exists \widetilde{x} \in \widetilde{T}, \bigwedge \theta_1 \mathcal{F}_{F_1,\mu_1,c_1} \cup \dots \cup \theta_m \mathcal{F}_{F_m,\mu_m,c_m}\right) . \end{aligned}$$

The formula $\neg\{\![\mathrm{prove}^{\varphi}(\mathcal{C}, \theta, \mathcal{S}_1)]\!\}$ is added to $\mathcal{L}^+$ and $(\widetilde{x} \cup \bigcup_{j=1}^m \theta_j I_{\mu_j}, \bigcup_{j=1}^m \theta_j \mathcal{F}_{F_j,\mu_j,c_j})$ is added to $\mathcal{L}^-$ when $(\mu_1, c_1, \dots, \mu_m, c_m) \in \mathcal{S}_0 \setminus \mathcal{S}_1$ (recall that, when e is Shoup event, e is always executed by $\mathsf{event\_abort}\ e$) so $\mathit{Tr} \vdash \left(\bigvee_{j\in J} \exists \mathcal{V}_j, \bigwedge \mathcal{F}_j\right) \vee (\bigvee \mathcal{L}^+)$.

Fact 2. Let e be a non-unique event. Let C be any evaluation context acceptable for G with public variables V that does not contain e. Let Tr be any full trace of $C[G]$ such that $\mathit{Tr} \vdash e$. Then $\mathit{Tr} \vdash \left(\bigvee_{j\in J} \exists \mathcal{V}_j, \bigwedge \mathcal{F}_j\right) \vee (\bigvee \mathcal{L}^+)$.

Proof of Fact 2. By Lemma 37, there exist a program point μ (in G) and a case c such that, for any θ' renaming of I_μ to fresh replication indices, there exists a mapping σ with domain $\theta' I_\mu$ such that $\mathit{Tr}, \sigma \vdash \theta' \mathcal{F}^0_{\mathsf{event}(e),\mu,c}$. Let $(\mathcal{V}, \mathcal{F}) = (\theta' I_\mu, \theta' \mathcal{F}^0_{\mathsf{event}(e),\mu,c})$ be the element of $\mathcal{L}^-$ for μ and c, in the treatment of correspondence $\mathsf{event}(e) \Rightarrow \mathrm{false}$. We have $\mathit{Tr} \vdash \exists \mathcal{V}, \bigwedge \mathcal{F}$. Therefore, $\mathit{Tr} \vdash \bigvee_{j\in J} \exists \mathcal{V}_j, \bigwedge \mathcal{F}_j$.

Fact 3. Let C be any evaluation context acceptable for G with public variables V. Let Tr be any trace of $C[G]$. If $\mathit{Tr} \vdash \neg \bigvee_{k\in K} \exists I_{\mu_k}, \bigwedge \mathcal{F}^{\mathrm{mod}}_k$, then there is no configuration in Tr at a modified program point μ_k.

Proof of Fact 3. By contraposition, if there is a configuration $\mathit{Conf} = E, \sigma, {}^{\mu_k}M, \mathcal{T}, \mu\mathcal{E}v$ or $\mathit{Conf} = E, (\sigma, {}^{\mu_k}P), \mathcal{Q}, \mathcal{C}h, \mathcal{T}, \mu\mathcal{E}v$ at a modified program point μ_k in trace Tr, then let θ be a renaming of I_{μ_k} to fresh indices and $\rho = \{\theta I_{\mu_k} \mapsto \sigma I_{\mu_k}\}$; by Corollary 30, $\mathit{Tr}, \rho \vdash \theta \mathcal{F}_{\mu_k}$. So $\mathit{Tr}, \rho \vdash \theta \mathcal{F}^{\mathrm{mod}}_k$, so $\mathit{Tr} \vdash \exists I_{\mu_k}, \bigwedge \mathcal{F}^{\mathrm{mod}}_k$.

We perform the proof for each query separately.

Case 1: sp **is some correspondence** φ **different from** $\forall \widetilde{x} \in \widetilde{T}, \mathsf{event}(e(\widetilde{N})) \Rightarrow \text{false}$ **(including** sp **is** true**).** Let C be any evaluation context acceptable for G with public variables V that does not contain events used by φ, D, nor non-unique events of G.

Consider any full trace Tr of $C[G]$ such that $Tr \vdash (\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G,D}$. Let us show that $Tr \vdash \left(\bigvee_{j\in J} \exists \mathcal{V}_j, \bigwedge \mathcal{F}_j\right) \vee (\bigvee \mathcal{L}^+)$.

Case 1.1: Tr does not execute any non-unique event of G. Then $Tr \vdash \neg\varphi \vee D_s$ where D_s is the disjunction of Shoup events in D.

Case 1.1.1: $Tr \vdash \neg\varphi$ We conclude by Fact 1.

Case 1.1.2: $Tr \vdash e$ for some Shoup event e in D_s. We conclude by Fact 1 for the correspondence $\mathsf{event}(e) \Rightarrow \text{false}$.

Case 1.2: Tr executes a non-unique event of G. Then $Tr \vdash e$ for some non-unique event e in G and in D. We conclude by Fact 2.

If $Tr \vdash \bigvee_{k\in K} \exists I_{\mu_k}, \bigwedge \mathcal{F}_k^{\text{mod}}$, then $Tr \vdash \left(\bigvee_{j\in J,k\in K} \exists I_{\mu_k}, \exists \mathcal{V}_j, \bigwedge \mathcal{F}_j \wedge \bigwedge \mathcal{F}_k^{\text{mod}}\right) \vee (\bigvee \mathcal{L}^+)$.

Otherwise, by Fact 3, there is no configuration in Tr at a modified program point μ_k, so Tr has a matching trace in $C[G']$ that also satisfies $(\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G',D}$.

We conclude that

$$\begin{aligned}
\mathsf{Adv}_G(C,\varphi,D) &\leq \Pr[C[G] : (\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G,D}] \\
&\leq \Pr[C[G] : \left(\bigvee_{j\in J,k\in K} \exists I_{\mu_k}, \exists \mathcal{V}_j, \bigwedge \mathcal{F}_j \wedge \bigwedge \mathcal{F}_k^{\text{mod}}\right) \vee \left(\bigvee \mathcal{L}^+\right)] \\
&\quad + \Pr[C[G'] : (\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G',D}] \\
&\leq \Pr[C[G] \preceq \left(\bigvee_{j\in J,k\in K} \exists I_{\mu_k}, \exists \mathcal{V}_j, \bigwedge \mathcal{F}_j \wedge \bigwedge \mathcal{F}_k^{\text{mod}}\right) \vee \left(\bigvee \mathcal{L}^+\right)] && \text{by Lemma 1} \\
&\quad + \Pr[C[G'] : (\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G',D}] \\
&\leq p(C) + \mathsf{Adv}_{G'}(C,\varphi,D) \\
&\leq p(C) + p'(C) && \text{since } \mathsf{Bound}_{G'}(V, sp, D, p')
\end{aligned}$$

Hence, we have $\mathsf{Bound}_G(V, sp, D, p+p')$.

Case 2: sp **is some correspondence** $\varphi = \forall \widetilde{x} \in \widetilde{T}, \mathsf{event}(e(\widetilde{N})) \Rightarrow \text{false}$**.** Let C be any evaluation context acceptable for G with public variables V that does not contain events used by φ, D, nor non-unique events of G.

Consider a full trace Tr of $C[G]$ such that $Tr \vdash (\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G,D}$. Let us show that either some prefix of Tr satisfies $\left(\bigvee_{j\in J,k\in K} \exists I_{\mu_k}, \exists \mathcal{V}_j, \bigwedge \mathcal{F}_j \wedge \bigwedge \mathcal{F}_k^{\text{mod}}\right) \vee (\bigvee \mathcal{L}^+)$ or there is a matching trace in $C[G']$ that satisfies $(\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G',D}$.

We have $Tr \vdash \exists \widetilde{x} \in \widetilde{T}, \mathsf{event}(e(\widetilde{N})) \vee D$.

Case 2.1: Tr executes a Shoup event e' of D. Then $Tr \vdash e'$ and Tr is actually a full trace that does not execute any non-unique event of G. By Fact 1, $Tr \vdash \left(\bigvee_{j\in J} \exists \mathcal{V}_j, \bigwedge_{F\in\mathcal{F}_j} F\right) \vee (\bigvee \mathcal{L}^+)$.

Case 2.2: Tr executes a non-unique event of G. Then $Tr \vdash e'$ for some non-unique event e' in G and in D. Then Tr is actually a full trace. By Fact 2, $Tr \vdash \left(\bigvee_{j\in J} \exists \mathcal{V}_j, \bigwedge_{F\in\mathcal{F}_j} F\right) \vee (\bigvee \mathcal{L}^+)$.

In cases 2.1 and 2.2,

- If $Tr \vdash \bigvee_{k\in K} \exists I_{\mu_k}, \bigwedge \mathcal{F}_k^{\text{mod}}$, then $Tr \vdash \left(\bigvee_{j\in J,k\in K} \exists I_{\mu_k}, \exists \mathcal{V}_j, \bigwedge \mathcal{F}_j \wedge \bigwedge \mathcal{F}_k^{\text{mod}}\right) \vee (\bigvee \mathcal{L}^+)$.

- Otherwise, by Fact 3, there is no configuration in Tr at a modified program point μ_k, so Tr has a matching trace in $C[G']$ that also satisfies $(\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G',D}$.

Case 2.3: Tr does not execute any non-unique event of G nor any Shoup event of D. Then $Tr \vdash \exists\widetilde{x} \in \widetilde{T}, \mathsf{event}(e(\widetilde{N}))$.

Let $\mu\mathcal{E}v$ be the sequence of events and E be the environment in the last configuration of Tr. There is a mapping ρ of the variables $\widetilde{x}$ to their values such that $Tr, \rho \vdash e(\widetilde{N})$. As in the proof of Lemma 37, there exist $\widetilde{a}$ and $\tau \in \mathbb{N}$ such that $\rho, \widetilde{N} \Downarrow \widetilde{a}$ and $\mu\mathcal{E}v(\tau) = (\mu, \widetilde{a}_0) : e(\widetilde{a})$ for some μ and $\widetilde{a}_0$. The rule of the semantics that may have added this element to $\mu\mathcal{E}v$ is (Event), (EventAbort), (CtxEvent), (FindE) or (EventT). (It cannot be (Find3) nor (Get3) because e is not a non-unique event. It cannot be (GetE) because G does not contain get by Property 4.)

- Case 2.3.1: In case (Event), we have reductions

$$\begin{aligned}
Conf &= E_0, (\sigma_0, {}^{\mu}\mathsf{event}\ e(\widetilde{M}); P), \mathcal{Q}_0, \mathcal{C}h_0, \mathcal{T}_0, \mu\mathcal{E}v_0\\
&\xrightarrow{p_0}_{t_0} \dots \xrightarrow{p_1}_{t_1} E_1, (\sigma_1, {}^{\mu}\mathsf{event}\ e(\widetilde{a}); P), \mathcal{Q}_0, \mathcal{C}h_0, \mathcal{T}_1, \mu\mathcal{E}v_1\\
&\xrightarrow{1} Conf' = E_1, (\sigma_1, P), \mathcal{Q}_0, \mathcal{C}h_0, \mathcal{T}_1, (\mu\mathcal{E}v_1, (\mu, \mathrm{Im}(\sigma_1)) : e(\widetilde{a}))
\end{aligned}$$

where ${}^{\mu}\mathsf{event}\ e(\widetilde{M}); P$ is a subprocess of $C[G]$ up to renaming of channels, by any number of applications of (Ctx) and a final application of (Event). The terms $\widetilde{M}$ are simple terms (when some term in $\widetilde{M}$ is not simple, no information is collected at all and **simplify** is not performed), so in fact

$$\begin{aligned}
Conf &= E_0, (\sigma_0, {}^{\mu}\mathsf{event}\ e(\widetilde{M}); P), \mathcal{Q}_0, \mathcal{C}h_0, \mathcal{T}_0, \mu\mathcal{E}v_0\\
&\xrightarrow{1}{}^{*}\ E_0, (\sigma_0, {}^{\mu}\mathsf{event}\ e(\widetilde{a}); P), \mathcal{Q}_0, \mathcal{C}h_0, \mathcal{T}_0, \mu\mathcal{E}v_0\\
&\xrightarrow{1} Conf' = E_0, (\sigma_0, P), \mathcal{Q}_0, \mathcal{C}h_0, \mathcal{T}_0, (\mu\mathcal{E}v_0, (\mu, \mathrm{Im}(\sigma_0)) : e(\widetilde{a}))
\end{aligned}$$

- Case 2.3.2: In case (EventAbort), we have a reduction

$$\begin{aligned}
Conf &= E, (\sigma_0, {}^{\mu}P), \mathcal{Q}_0, \mathcal{C}h_0, \mathcal{T}_0, \mu\mathcal{E}v_0\\
&\xrightarrow{p}_{t} Conf' = E, (\sigma_0, \mathsf{abort}), \mathcal{Q}_0, \mathcal{C}h_0, \mathcal{T}_0, (\mu\mathcal{E}v_0, (\mu, \mathrm{Im}(\sigma_0)) : e)
\end{aligned}$$

where ${}^{\mu}P$ is a subprocess of $C[G]$ up to renaming of channels, by (EventAbort).

- Case 2.3.3: In case (EventT), we have reductions

$$\begin{aligned}
Conf &= E_0, \sigma_0, {}^{\mu}\mathsf{event}\ e(\widetilde{M}); N, \mathcal{T}_0, \mu\mathcal{E}v_0\\
&\xrightarrow{p_0}_{t_0} \dots \xrightarrow{p_1}_{t_1} E_1, \sigma_1, {}^{\mu}\mathsf{event}\ e(\widetilde{a}); N, \mathcal{T}_1, \mu\mathcal{E}v_1\\
&\xrightarrow{1} E_1, \sigma_1, N, \mathcal{T}_1, (\mu\mathcal{E}v_1, (\mu, \mathrm{Im}(\sigma_1)) : e(\widetilde{a}))
\end{aligned}$$

where ${}^{\mu}\mathsf{event}\ e(\widetilde{M}); N$ is a subterm of $C[G]$, by any number of applications of (CtxT) and a final application of (EventT). The terms $\widetilde{M}$ are simple terms, so in fact

$$\begin{aligned}
Conf &= E_0, \sigma_0, {}^{\mu}\mathsf{event}\ e(\widetilde{M}); N, \mathcal{T}_0, \mu\mathcal{E}v_0\\
&\xrightarrow{1}{}^{*}\ E_0, \sigma_0, {}^{\mu}\mathsf{event}\ e(\widetilde{a}); N, \mathcal{T}_0, \mu\mathcal{E}v_0\\
&\xrightarrow{1} E_0, \sigma_0, N, \mathcal{T}_0, (\mu\mathcal{E}v_0, (\mu, \mathrm{Im}(\sigma_0)) : e(\widetilde{a}))
\end{aligned}$$

By Invariant 4, μ is not inside a condition of find or get, so by Lemma 5, *Conf* is not in the derivation of an hypothesis of a rule for find or get. The only rule for processes other than those for find or get that evaluates a non-simple term is (Ctx) and similarly, the only rule for terms other than those for find or get that evaluates a term is (CtxT), so we have

$$\begin{aligned}&E_0, (\sigma_0, C_0[C_1[\dots C_k[{}^{\mu}\mathsf{event}\ e(\widetilde{M}); N]\dots]]), \mathcal{Q}_0, \mathcal{C}h_0, \mathcal{T}_0, \mu\mathcal{E}v_0\\ &\quad\xrightarrow{1}{}^{*}\ \mathit{Conf}' = E_0, (\sigma_0, C_0[C_1[\dots C_k[N]\dots]]), \mathcal{Q}_0, \mathcal{C}h_0, \mathcal{T}_0, (\mu\mathcal{E}v_0, (\mu, \mathrm{Im}(\sigma_0)) : e(\widetilde{a}))\end{aligned}$$

for some C_0 context defined in Figure 10, $k \in \mathbb{N}$, and $C_1, \dots, C_k$ contexts defined in Figure 6, by k applications of (CtxT) and one application of (Ctx).

- Case 2.3.4: In case (CtxEvent), we have

$$\mathit{Conf} = E_0, \sigma_0, {}^{\mu}\mathsf{event\_abort}\ e, \mathcal{T}_0, \mu\mathcal{E}v_0 \xrightarrow{p}_t E_0, \sigma_0, \mathsf{event\_abort}\ (\mu, \mathrm{Im}(\sigma_0)) : e, \mathcal{T}_0, \mu\mathcal{E}v_0$$

by (EventAbortT) where ${}^{\mu}\mathsf{event\_abort}\ e$ is a subterm of $C[G]$, followed by applications (FindTE), (CtxT), (CtxEventT), (FindE), (Ctx), and (CtxEvent). We let *Conf*′ be the last configuration of *Tr*.

Let θ' be a renaming of I_μ to fresh replication indices. We have

$$\begin{aligned}&\mathrm{prove}^{\varphi}(\mathcal{C}, \theta_0, \mu, c) = (\theta'\mathcal{F}_{\mathsf{event}(e(\widetilde{N})),\mu,c} \text{ yields a contradiction}),\\ &\{\![\mathrm{prove}^{\varphi}(\mathcal{C}, \theta_0, \mu, c)]\!\} = \forall \theta' I_\mu, \forall \widetilde{x} \in \widetilde{T}, \neg \bigwedge \theta'\mathcal{F}_{\mathsf{event}(e(\widetilde{N})),\mu,c},\\ &\mathcal{C} \text{ is a pseudo-formula with all leaves } \bot, \text{ so } \{\![\vdash \mathcal{C}]\!\} = \text{true}.\end{aligned}$$

If $(\mu, c) \in \mathcal{S}_0 \setminus \mathcal{S}_1$, then in cases 2.3.1 and 2.3.3, $(\widetilde{x} \cup \theta' I_\mu, \theta'\mathcal{F}_{\mu,c} \cup \{\mathit{lastdefprogrampoint}(\mu, \theta' I_\mu), \widetilde{N} = \theta'\widetilde{M}\})$ is added to $\mathcal{L}^-$ and in cases 2.3.2 and 2.3.4, $(\theta' I_\mu, \theta'\mathcal{F}_{\mathsf{event}(e),\mu,c})$ is added to $\mathcal{L}^-$. Moreover, $\neg\{\![\mathrm{prove}^{\varphi}(\mathcal{C}, \theta_0, \mathcal{S}_1)]\!\} = \bigvee_{(\mu,c)\in\mathcal{S}_1} \exists\theta' I_\mu, \exists \widetilde{x} \in \widetilde{T}, \bigwedge \theta'\mathcal{F}_{\mathsf{event}(e(\widetilde{N})),\mu,c}$ is added to $\mathcal{L}^+$.

As in the proof of Lemma 37, we have $\sigma_0 = [I_\mu \mapsto \widetilde{a}_0]$. Let $\sigma = \{\theta' I_\mu \mapsto \widetilde{a}_0\}$. As in the proof of Lemma 37, we have $\mathit{Tr}, \sigma \cup \rho \vdash \theta'\mathcal{F}_{\mathsf{event}(e(\widetilde{N})),\mu,c}$, since *Tr* does not execute any non-unique event of G.

- In cases 2.3.1 and 2.3.3 when $(\mu, c) \in \mathcal{S}_1$ and in cases 2.3.2 and 2.3.4, we have $\mathit{Tr} \vdash \left(\bigvee_{j\in J} \exists \mathcal{V}_j, \bigwedge \mathcal{F}_j\right) \vee (\bigvee \mathcal{L}^+)$.
  - If $\mathit{Tr} \vdash \bigvee_{k\in K} \exists I_{\mu_k}, \bigwedge \mathcal{F}_k^{\mathrm{mod}}$, then $\mathit{Tr} \vdash \left(\bigvee_{j\in J,k\in K} \exists I_{\mu_k}, \exists \mathcal{V}_j, \bigwedge \mathcal{F}_j \wedge \bigwedge \mathcal{F}_k^{\mathrm{mod}}\right) \vee (\bigvee \mathcal{L}^+)$.
  - Otherwise, by Fact 3, there is no configuration in *Tr* at a modified program point μ_k, so *Tr* has a matching trace in $C[G']$ that also satisfies $(\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G',D}$.
- In cases 2.3.1 and 2.3.3 when $(\mu, c) \in \mathcal{S}_0 \setminus \mathcal{S}_1$, let *Tr*′ be the prefix of *Tr* that stops at *Conf*′. We have $\mathit{Conf} \preceq_{\mathit{Tr}} \mathit{Conf}'$, so by Corollary 3, $\mathit{Tr} \preceq \mathit{Conf}', \sigma \vdash \theta'\mathcal{F}_{\mu,c}$, that is, $\mathit{Tr}', \sigma \vdash \theta'\mathcal{F}_{\mu,c}$. We have $E_0, \sigma_0, \widetilde{M} \Downarrow \widetilde{a}$, so $E_0, \sigma, \theta'\widetilde{M} \Downarrow \widetilde{a}$. The environment $E_{\mathit{Tr}'}$ extends E_0, so $E_{\mathit{Tr}'}, \sigma, \theta'\widetilde{M} \Downarrow \widetilde{a}$, so $\mathit{Tr}', \sigma \cup \rho \vdash \theta'\widetilde{M} = \widetilde{N}$. Since $E_{\mathit{Tr}\preceq\mathit{Conf}'} = E_{\mathit{Tr}\preceq\mathit{Conf}}$ and $\sigma_{\mathit{Conf}'} = \sigma_{\mathit{Conf}}$, $\mathit{Tr}', \sigma \vdash \mathit{lastdefprogrampoint}(\mu, \theta'\widetilde{i})$. Hence $\mathit{Tr}' \vdash \bigvee_{j\in J} \exists \mathcal{V}_j, \bigwedge \mathcal{F}_j$.
  - If $\mathit{Tr}' \vdash \bigvee_{k\in K} \exists I_{\mu_k}, \bigwedge \mathcal{F}_k^{\mathrm{mod}}$, then $\mathit{Tr}' \vdash \left(\bigvee_{j\in J,k\in K} \exists I_{\mu_k}, \exists \mathcal{V}_j, \bigwedge \mathcal{F}_j \wedge \bigwedge \mathcal{F}_k^{\mathrm{mod}}\right) \vee (\bigvee \mathcal{L}^+)$ and *Tr*′ is a prefix of *Tr*.

– Otherwise, by Fact 3, there is no configuration in Tr' at a modified program point μ_k, so Tr has a matching trace in $C[G']$ that also satisfies $(\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G',D}$. (Note that Tr may still be modified by **success simplify**: the matching trace in $C[G']$ may execute a modified program point μ_k after $Conf'$, but still the matching trace executes $e(\widetilde{M})$, so satisfies $\neg\varphi$, and it does not execute any non-unique event of G'.)

We conclude that

$$\begin{aligned}\mathsf{Adv}_G(C,\varphi,D) &= \Pr[C[G] : (\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G,D}]\\ &\leq \Pr[C[G] \preceq \left(\bigvee_{j\in J, k\in K} \exists I_{\mu_k}, \exists \mathcal{V}_j, \bigwedge \mathcal{F}_j \wedge \bigwedge \mathcal{F}_k^{\mathrm{mod}}\right) \vee \left(\bigvee \mathcal{L}^+\right)]\\ &\quad + \Pr[C[G'] : (\neg\varphi \vee D) \wedge \neg\mathsf{NonUnique}_{G',D}]\\ &\leq p(C) + \mathsf{Adv}_{G'}(C,\varphi,D)\\ &\leq p(C) + p'(C) \qquad \text{since } \mathsf{Bound}_{G'}(V, sp, D, p')\end{aligned}$$

Hence, we have $\mathsf{Bound}_G(V, sp, D, p + p')$.

Case 3: sp **is** $\mathsf{1\text{-}ses.secr.}(x)$**,** $\mathsf{Secrecy}(x)$**, or** $\mathsf{bit\ secr.}(x)$ **with** $C = C'[C_{sp}[\,]]$ **and** x **is defined only by** new **or by assignments of variables defined by** new**.** Let C' be any evaluation context acceptable for $C_{sp}[G]$ with public variables $V \setminus V_{sp}$ that does not contain S, $\overline{\mathsf{S}}$, any event in D, nor any non-unique event of G.

$$\begin{aligned}\mathsf{Adv}_G(C, sp, D) &= \Pr[C[G] : \mathsf{S} \vee D] - \Pr[C[G] : \overline{\mathsf{S}} \vee \mathsf{NonUnique}_{G,D}]\\ &= \Pr[C[G] : \mathsf{S}] + \Pr[C[G] : D] - \Pr[C[G] : \overline{\mathsf{S}}] - \Pr[C[G] : \mathsf{NonUnique}_{G,D}]\\ &\qquad \text{since these events are mutually exclusive}\\ &= \Pr[C[G] : \mathsf{S} \wedge \neg sp] + \Pr[C[G] : D]\\ &\quad - \Pr[C[G] : \overline{\mathsf{S}} \wedge \neg sp] - \Pr[C[G] : \mathsf{NonUnique}_{G,D}] \qquad \text{by Lemma 35}\\ &= \Pr[C[G] : (\mathsf{S} \wedge \neg sp) \vee D] - \Pr[C[G] : \overline{\mathsf{S}} \wedge \neg sp] - \Pr[C[G] : \mathsf{NonUnique}_{G,D}]\end{aligned}$$

We have

- $\Pr[C[G'] : \mathsf{NonUnique}_{G',D}] \leq \Pr[C[G] : \mathsf{NonUnique}_{G,D}]$,
- $\Pr[C[G'] : \overline{\mathsf{S}} \wedge \neg sp] \leq \Pr[C[G] : \overline{\mathsf{S}} \wedge \neg sp]$

since a trace that executes a non-unique event or $\overline{\mathsf{S}}$ in G' cannot execute event adv_loses, so it executed without change in G. Let us show that $\Pr[C[G] : (\mathsf{S} \wedge \neg sp) \vee D] \leq p(C) + \Pr[C[G'] : (\mathsf{S} \wedge \neg sp) \vee D]$. Let Tr be a full trace of $C[G]$ such that $Tr \vdash (\mathsf{S} \wedge \neg sp) \vee D$. Let us show that $Tr \vdash \left(\bigvee_{j\in J} \exists \mathcal{V}_j, \bigwedge \mathcal{F}_j\right) \vee (\bigvee \mathcal{L}^+)$.

Case 3.1: Tr executes a non-unique event of G. Then $Tr \vdash e$ for some non-unique event e in G and in D. We conclude by Fact 2.

Case 3.2: $Tr \vdash e$ for some Shoup event e in D. Then Tr does not execute any non-unique event of G. We conclude by Fact 1 for the correspondence $\mathsf{event}(e) \Rightarrow \mathrm{false}$.

Case 3.3: $Tr \vdash \mathsf{S} \wedge \neg sp$. Since $Tr \vdash \neg sp$, we have $Tr \vdash \neg[\![\mathrm{prove}^{sp}(\mathrm{Tpp}(Tr))]\!]$, so we are in one of the following two cases:

- There exists $\mu \in \mathrm{Tpp}(Tr)$ such that $Tr \vdash \neg\{\![\mathrm{prove}^{\textsf{1-ses.secr.}(x)}(\mu)]\!\}$.
  - If $\mu \in \mathcal{S}_1$, then $Tr \vdash \bigvee \mathcal{L}^+$ since $\neg\{\![\mathrm{prove}^{\textsf{1-ses.secr.}(x)}(\mathcal{S}_1)]\!\}$ is added to $\mathcal{L}^+$.
  - If $\mu \in \mathcal{S}_0 \setminus \mathcal{S}_1$, then $(\theta I_\mu, \theta\mathcal{F}_\mu)$ is added to $\mathcal{L}^-$, where θ is a renaming of I_μ to fresh replication indices. Moreover, we have $\neg\{\![\mathrm{noleak}(z[\widetilde{M'}], \mathcal{I}, \mathcal{F})]\!\} \Rightarrow \exists\mathcal{I}, \bigwedge\mathcal{F}$, so $\neg\{\![\mathrm{prove}^{\textsf{1-ses.secr.}(x)}(\mu)]\!\} \Rightarrow \exists\theta I_\mu, \bigwedge\theta\mathcal{F}_\mu$ since $\mathrm{defRand}_\mu(x)$ is defined. Therefore, $Tr \vdash \bigvee_{j\in J} \exists\mathcal{V}_j, \bigwedge\mathcal{F}_j$.
- In case $sp = \mathsf{Secrecy}(x)$, there exist $\mu_1, \mu_2 \in \mathrm{Tpp}(Tr)$ such that $Tr \vdash \neg\{\![\mathrm{prove}^{\textsf{distinct}(x)}(\mu_1, \mu_2)]\!\}$. Let $z_1[\widetilde{M_1}] = \mathrm{defRand}_{\mu_1}(x)$, $z_2[\widetilde{M_2}] = \mathrm{defRand}_{\mu_2}(x)$, $\widetilde{i}$ be the current replication indices at the definition of x, θ_1 and θ_2 be two distinct renamings of $\widetilde{i}$ to fresh replication indices, $\widetilde{i_1} = \theta_1\widetilde{i}$, $\widetilde{i_2} = \theta_2\widetilde{i}$, and $\mathcal{F} = \theta_1\mathcal{F}_{\mu_1} \cup \theta_2\mathcal{F}_{\mu_2} \cup \{\theta_1\widetilde{M_1} = \theta_2\widetilde{M_2}, \widetilde{i_1} \neq \widetilde{i_2}\}$ Then $z_1 = z_2$ and $Tr \vdash \exists\widetilde{i_1}, \exists\widetilde{i_2}, \bigwedge\mathcal{F}$.
  - If $\mathcal{F}$ yields a contradiction, then $\exists\widetilde{i_1}, \exists\widetilde{i_2}, \bigwedge\mathcal{F}$ is added to $\mathcal{L}^+$, so $Tr \vdash \bigvee\mathcal{L}^+$.
  - Otherwise, $(\widetilde{i_1} \cup \widetilde{i_2}, \mathcal{F})$ is added to $\mathcal{L}^-$, so $Tr \vdash \bigvee_{j\in J} \exists\mathcal{V}_j, \bigwedge\mathcal{F}_j$.

If $Tr \vdash \bigvee_{k\in K} \exists I_{\mu_k}, \bigwedge\mathcal{F}_k^{\mathrm{mod}}$, then $Tr \vdash \left(\bigvee_{j\in J, k\in K} \exists I_{\mu_k}, \exists\mathcal{V}_j, \bigwedge\mathcal{F}_j \wedge \bigwedge\mathcal{F}_k^{\mathrm{mod}}\right) \vee (\bigvee\mathcal{L}^+)$.

Otherwise, by Fact 3, there is no configuration in Tr at a modified program point μ_k, so Tr has a matching trace in $C[G']$ that also satisfies $(\mathsf{S} \wedge \neg sp) \vee D$.

Therefore, $\Pr[C[G] : (\mathsf{S} \wedge \neg sp) \vee D] \leq p(C) + \Pr[C[G'] : (\mathsf{S} \wedge \neg sp) \vee D]$, so

$$\mathsf{Adv}_G(C, sp, D) \leq p(C) + \Pr[C[G'] : (\mathsf{S}\wedge\neg sp)\vee D] - \Pr[C[G'] : \overline{\mathsf{S}}\wedge\neg sp] - \Pr[C[G'] : \mathsf{NonUnique}_{G',D}]$$

Moreover, by applying on G' the same steps as on G at the beginning of this proof, we have

$$\mathsf{Adv}_{G'}(C, sp, D) = \Pr[C[G'] : (\mathsf{S} \wedge \neg sp) \vee D] - \Pr[C[G'] : \overline{\mathsf{S}} \wedge \neg sp] - \Pr[C[G'] : \mathsf{NonUnique}_{G',D}]$$

so

$$\mathsf{Adv}_G(C, sp, D) \leq p(C) + \mathsf{Adv}_{G'}(C, sp, D) \leq p(C) + p'(C)$$

since $\mathsf{Bound}_{G'}(V, sp, D, p')$. Therefore, we have $\mathsf{Bound}_G(V, sp, D, p + p')$. □

5.2 crypto: Applying the Security Assumptions on Primitives

The **crypto** transformation applies security assumptions on primitives. The first version of this transformation was presented in [24, Section 3.2 and Appendix D].

6 Proof Strategy

The first version of the automatic proof strategy was presented in [24, Section 5].

7 Conclusion

The tool CryptoVerif produces proofs by sequences of games like those manually written by cryptographers. It generates the games, using an automatic proof strategy or guidance from the user, who specifies the transformations to perform. It supports a wide variety of cryptographic primitives specified by indistinguishability axioms. Many of these primitives are included in a library so that the user does not have to redefine them. It can prove secrecy, correspondence, and indistinguishability properties. It has been applied to substantial case studies, including Signal [47], TLS 1.3 [21], and WireGuard [53].

CryptoVerif still has limitations. In particular, the size of games tends to grow too fast, which limits its ability to deal with large examples, especially because some game transformations require the game to be expanded first by the **expand** transformation, which duplicates the code from each test until the end of protocol. Planed improvements include allowing more game transformations to work without previous application of **expand**; allowing internal oracle calls in games, in order to share code between different parts of the game; using composition results in order to make proofs more modular. Moreover, some game transformations could be generalized. For instance, the transformation **merge_branches** merges branches of a test when they execute the same code; the detection that several branches execute equivalent code could be made more flexible, by allowing reorderings of instructions for instance. CryptoVerif only considers blackbox adversaries: it does not support proofs that manipulate the code of the adversary, such as the forking lemma [60].

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 and Charlie Jacomme for helpful discussions on this document. The design and implementation of CryptoVerif was partly done while I was at CNRS and at Ecole Normale Supérieure. This work was partly supported by the French National Research Agency (ANR) under the projects FormaCrypt (ARA SSIA 2005), ProSe (VERSO 2010, decision number 2010-VERS-004), TECAP (decision number ANR-17-CE39-0004-03) and received funding from the France 2030 program managed by the ANR under the reference ANR-22-PECY-0006 (PEPR Cybersecurity SVP).

References

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

[2] 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.

[3] R. Affeldt, D. Nowak, and K. Yamada. Certifying assembly with formal cryptographic proofs: the case of BBS. In *9th International Workshop on Automated Verification of Critical Systems (AVoCS'09)*, volume 23 of *Electronic Communications of the EASST*. EASST, Sept. 2009.

[4] J. B. Almeida, M. Barbosa, G. Barthe, M. Campagna, E. Cohen, B. Grégoire, V. Pereira, B. Portela, P.-Y. Strub, and S. Tasiran. A machine-checked proof of security for AWS key management service. In *ACM SIGSAC Conference on Computer and Communications Security (CCS'19)*, pages 63–78, New York, NY, Nov. 2019. ACM Press.

[5] M. Backes, D. Hofheinz, and D. Unruh. CoSP: A general framework for computational soundness proofs. In *ACM Conference on Computer and Communications Security (CCS'09)*, pages 66–78, New York, NY, Nov. 2009. ACM Press.

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

[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, Los Alamitos, CA, June 2004. IEEE Computer Society Press.

[8] 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, New York, NY, Oct. 2003. ACM Press.

[9] D. Baelde, S. Delaune, A. Koutsos, C. Jacomme, and S. Moreau. An interactive prover for protocol verification in the computational model. In *42nd IEEE Symposium on Security and Privacy (S&P'21),*, pages 537–554, Los Alamitos, CA, May 2021. IEEE Computer Society Press.

[10] G. Barthe, J. M. Crespo, Y. Lakhnech, and B. Schmidt. Mind the gap: Modular machine-checked proofs of one-round key exchange protocols. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, volume 9057 of *Lecture Notes in Computer Science*, pages 689–718, Berlin, Heidelberg, Apr. 2015. Springer.

[11] G. Barthe, M. Daubignard, B. Kapron, and Y. Lakhnech. Computational indistinguishability logic. In *17th ACM Conference on Computer and Communications Security (CCS'10)*, pages 375–386, New York, NY, Oct. 2010. ACM Press.

[12] G. Barthe, B. Grégoire, S. Z. Béguelin, and Y. Lakhnech. Beyond provable security. Verifiable IND-CCA security of OAEP. In A. Kiayias, editor, *Topics in Cryptology - CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 180–196, Berlin, Heidelberg, Feb. 2011. Springer.

[13] G. Barthe, B. Grégoire, S. Heraud, and S. Z. Béguelin. Formal certification of ElGamal encryption. A gentle introduction to CertiCrypt. In P. Degano, J. Guttman, and F. Martinelli, editors, *5th International Workshop on Formal Aspects in Security and Trust, FAST 2008*, volume 5491 of *Lecture Notes in Computer Science*, pages 1–19, Berlin, Heidelberg, 2009. Springer.

[14] G. Barthe, B. Grégoire, S. Heraud, and S. Z. Béguelin. Computer-aided security proofs for the working cryptographer. In P. Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90, Berlin, Heidelberg, Aug. 2011. Springer.

[15] G. Barthe, B. Grégoire, and S. Zanella. Formal certification of code-based cryptographic proofs. In *36th ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages (POPL'09)*, pages 90–101, New York, NY, Jan. 2009. ACM Press.

[16] D. Basin, A. Lochbihler, and S. R. Sefidgar. CryptHOL: Game-based proofs in higher-order logic. *Journal of Cryptology*, 33:494–566, 2020.

[17] S. Z. Béguelin, G. Barthe, S. Heraud, B. Grégoire, and D. Hedin. A machine-checked formalization of sigma-protocols. In *23rd Computer Security Foundations Symposium (CSF'10)*, pages 246–260, Los Alamitos, CA, July 2010. IEEE Computer Society Press.

[18] S. Z. Béguelin, B. Grégoire, G. Barthe, and F. Olmedo. Formally certifying the security of digital signature schemes. In *30th IEEE Symposium on Security and Privacy, S&P 2009*, pages 237–250, Los Alamitos, CA, May 2009. IEEE Computer Society Press.

[19] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In T. Okamoto, editor, *Advances in Cryptology – ASIACRYPT'00*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545, Berlin, Heidelberg, Dec. 2000. Springer.

[20] 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 in Computer Science*, pages 409–426, Berlin, Heidelberg, May 2006. Springer. Extended version available at `http://eprint.iacr.org/2004/331`.

[21] K. Bhargavan, B. Blanchet, and N. Kobeissi. Verified models and reference implementations for the TLS 1.3 standard candidate. In *IEEE Symposium on Security and Privacy (S&P'17)*, pages 483–503, Los Alamitos, CA, May 2017. IEEE Computer Society Press.

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

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

[24] B. Blanchet. A computationally sound mechanized prover for security protocols. *IEEE Transactions on Dependable and Secure Computing*, 5(4):193–207, Oct.–Dec. 2008.

[25] B. Blanchet. Automatically verified mechanized proof of one-encryption key exchange. Cryptology ePrint Archive, Report 2012/173, Apr. 2012. Available at `http://eprint.iacr.org/2012/173`.

[26] B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 75(1):3–51, Feb.–Mar. 2008.

[27] I. Boureanu, C. C. Drăgan, F. Dupressoir, D. Gérault, and P. Lafourcade. Mechanised models and proofs for distance-bounding. In *34th IEEE Computer Security Foundations Symposium (CSF'21)*, Los Alamitos, CA, 2021. IEEE Computer Society Press.

[28] D. Cadé and B. Blanchet. From computationally-proved protocol specifications to implementations. In *7th International Conference on Availability, Reliability and Security (AReS 2012)*, pages 65–74, Los Alamitos, CA, Aug. 2012. IEEE Computer Society Press.

[29] D. Cadé and B. Blanchet. From computationally-proved protocol specifications to implementations and application to SSH. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, 4(1):4–31, Mar. 2013.

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

[31] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Linch, O. Pereira, and R. Segala. Time-bounded task-PIOAs: A framework for analyzing security protocols. In S. Dolev, editor, *20th Symposium on Distributed Computing (DISC)*, volume 4167 of *Lecture Notes in Computer Science*, pages 238–253, Berlin, Heidelberg, Sept. 2006. Springer.

[32] R. Canetti and J. Herzog. Universally composable symbolic analysis of cryptographic protocols (the case of encryption-based mutual authentication and key exchange). Cryptology ePrint Archive, Report 2004/334, 2004. Available at `http://eprint.iacr.org/2004/334`.

[33] H. Comon-Lundh and V. Cortier. Computational soundness of observational equivalence. In *15th ACM conference on Computer and communications security (CCS'08)*, pages 109–118, New York, NY, Oct. 2008. ACM Press.

[34] V. Cortier, C. C. Drăgan, F. Dupressoir, B. Schmidt, P.-Y. Strub, and B. Warinschi. Machine-checked proofs of privacy for electronic voting protocols. In *IEEE Symposium on Security and Privacy (SP'17)*, pages 993–1008, Los Alamitos, CA, 2017. IEEE Computer Society Press.

[35] V. Cortier, H. Hördegen, and B. Warinschi. Explicit randomness is not necessary when modeling probabilistic encryption. In C. Dima, M. Minea, and F. Tiplea, editors, *Workshop on Information and Computer Security (ICS 2006)*, volume 186 of *Electronic Notes in Theoretical Computer Science*, pages 49–65. Elsevier, Sept. 2006.

[36] V. Cortier, S. Kremer, and B. Warinschi. A survey of symbolic methods in computational analysis of cryptographic systems. *Journal of Automated Reasoning*, 46(3-4):225–259, Apr. 2011.

[37] 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 in Computer Science*, pages 157–171, Berlin, Heidelberg, Apr. 2005. Springer.

[38] V. Cortier and B. Warinschi. A composable computational soundness notion. In *18th ACM Conference on Computer and Communications Security (CCS'11)*, pages 63–74, New York, NY, Oct. 2011. ACM Press.

[39] J. Courant, M. Daubignard, C. Ene, P. Lafourcade, and Y. Lakhnech. Towards automated proofs for asymmetric encryption schemes in the random oracle model. In *15th ACM conference on Computer and communications security (CCS'08)*, pages 371–380, New York, NY, Oct. 2008. ACM Press.

[40] J. Courant, M. Daubignard, C. Ene, P. Lafourcade, and Y. Lakhnech. Automated proofs for asymmetric encryption. In D. Dams, U. Hannemann, and M. Steffen, editors, *Concurrency, Compositionality, and Correctness*, volume 5930 of *Lecture Notes in Computer Science*, pages 300–321, Berlin, Heidelberg, 2010. Springer.

[41] J. Courant, C. Ene, and Y. Lakhnech. Computationally sound typing for non-interference: The case of deterministic encryption. In V. Arvind and S. Prasad, editors, *27th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07)*, volume 4855 of *Lecture Notes in Computer Science*, pages 364–375, Berlin, Heidelberg, Dec. 2007. Springer.

[42] 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 in Computer Science*, pages 16–29, Berlin, Heidelberg, July 2005. Springer.

[43] A. Datta, A. Derek, J. C. Mitchell, and B. Warinschi. Computationally sound compositional logic for key exchange protocols. In *19th IEEE Computer Security Foundations Workshop (CSFW'06)*, pages 321–334, Los Alamitos, CA, July 2006. IEEE Computer Society Press.

[44] J. Gancher, S. Gibson, P. Singh, S. Dharanikota, and B. Parno. OWL: Compositional verification of security protocols via an information-flow type system. In *2023 IEEE Symposium on Security and Privacy (S&P)*, pages 1114–1131, Los Alamitos, CA, May 2023. IEEE Computer Society Press.

[45] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptative chosen-message attacks. *SIAM Journal of Computing*, 17(2):281–308, Apr. 1988.

[46] 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 in Computer Science*, pages 172–185, Berlin, Heidelberg, Apr. 2005. Springer.

[47] N. Kobeissi, K. Bhargavan, and B. Blanchet. Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In *2nd IEEE European Symposium on Security and Privacy (EuroS&P'17)*, pages 435–450, Los Alamitos, CA, Apr. 2017. IEEE Computer Society Press.

[48] 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 in Computer Science*, pages 159–173, Berlin, Heidelberg, Apr. 2003. Springer.

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

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

[51] P. Laud and I. Tšahhirov. A user interface for a game-based protocol verification tool. In P. Degano and J. Guttman, editors, *6th International Workshop on Formal Aspects in Security and Trust (FAST2009)*, volume 5983 of *Lecture Notes in Computer Science*, pages 263–278, Berlin, Heidelberg, Nov. 2009. Springer.

[52] P. Laud and V. Vene. A type system for computationally secure information flow. In M. Liśkiewicz and R. Reischuk, editors, *15th International Symposium on Fundamentals of Computation Theory (FCT'05)*, volume 3623 of *Lecture Notes in Computer Science*, pages 365–377, Berlin, Heidelberg, Aug. 2005. Springer.

[53] B. Lipp, B. Blanchet, and K. Bhargavan. A mechanised cryptographic proof of the WireGuard virtual private network protocol. In *IEEE European Symposium on Security and Privacy (EuroS&P'19)*, pages 231–246, Stockholm, Sweden, June 2019. IEEE Computer Society.

[54] 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.

[55] D. Nowak. A framework for game-based security proofs. In S. Qing, H. Imai, and G. Wang, editors, *Information and Communications Security, 9th International Conference, ICICS 2007*, volume 4861 of *Lecture Notes in Computer Science*, pages 319–333, Berlin, Heidelberg, Dec. 2007. Springer.

[56] D. Nowak. On formal verification of arithmetic-based cryptographic primitives. In P. J. Lee and J. H. Cheon, editors, *Information Security and Cryptology - ICISC 2008, 11th International Conference*, volume 5461 of *Lecture Notes in Computer Science*, pages 368–382, Berlin, Heidelberg, Dec. 2008. Springer.

[57] D. Nowak and Y. Zhang. A calculus for game-based security proofs. In *Provable Security, Fourth International Conference, ProvSec 2010*, volume 6402 of *Lecture Notes in Computer Science*, pages 35–52, Berlin, Heidelberg, Oct. 2010. Springer.

[58] T. Okamoto and D. Pointcheval. The gap-problems: a new class of problems for the security of cryptographic schemes. In K. Kim, editor, *International Workshop on Practice and Theory in Public Key Cryptography (PKC'2001)*, volume 1992 of *Lecture Notes in Computer Science*, pages 104–118, Berlin, Heidelberg, Feb. 2001. Springer.

[59] A. Petcher and G. Morrisett. The foundational cryptography framework. In R. Focardi and A. C. Myers, editors, *4th International Conference on Principles of Security and Trust (POST'15)*, volume 9036 of *Lecture Notes in Computer Science*, pages 53–72, Berlin, Heidelberg, Apr. 2015. Springer.

[60] D. Pointcheval and J. Stern. Security proofs for signature schemes. In U. Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398, Berlin, Heidelberg, May 1996. Springer.

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

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

[63] V. Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, Nov. 2004. Available at `http://eprint.iacr.org/2004/332`.

[64] G. Smith and R. Alpízar. Secure information flow with random assignment and encryption. In *4th ACM Workshop on Formal Methods in Security Engineering (FMSE'06)*, pages 33–43, Nov. 2006.

[65] 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, Los Alamitos, CA, July 2006. IEEE Computer Society Press.

[66] C. Sprenger and D. Basin. Cryptographically-sound protocol-model abstractions. In *23rd Annual IEEE Symposium on Logic in Computer Science*, pages 3–17, Los Alamitos, CA, June 2008. IEEE Computer Society Press.

[67] I. Tšahhirov and P. Laud. Application of dependency graphs to security protocol analysis. In G. Barthe and C. Fournet, editors, *3rd Symposium on Trustworthy Global Computing (TGC'07)*, volume 4912 of *Lecture Notes in Computer Science*, pages 294–311, Berlin, Heidelberg, Nov. 2007. Springer.

[68] T. Y. C. Woo and S. S. Lam. Authentication for distributed systems. In D. Denning and P. Denning, editors, *Internet Besieged: Countering Cyberspace Scofflaws*, pages 319–355. ACM Press and Addison-Wesley, Oct. 1997.

RESEARCH CENTRE
PARIS

2 rue Simone Iff - CS 42112
75589 Paris Cedex 12

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399