Automated Verification of Selected Equivalences for Security Protocols*

Bruno Blanchet
CNRS, École Normale Supérieure, Paris

Martín Abadi
University of California, Santa Cruz
and Microsoft Research, Silicon valley

Cédric Fournet
Microsoft Research, Cambridge

July 11, 2007

Abstract

In the analysis of security protocols, methods and tools for reasoning about protocol behaviors have been quite effective. We aim to expand the scope of those methods and tools. We focus on proving equivalences $P \approx Q$ in which P and Q are two processes that differ only in the choice of some terms. These equivalences arise often in applications. We show how to treat them as predicates on the behaviors of a process that represents P and Q at the same time. We develop our techniques in the context of the applied pi calculus and implement them in the tool ProVerif.

1 Introduction

Many security properties can be expressed as predicates on system behaviors. These properties include some kinds of secrecy properties (for instance, "the system never broadcasts the key k"). They also include correspondence properties (for instance, "if the system deletes file f, then the administrator must have requested it"). Such predicates on system behaviors are the focus of many successful methods for security analysis. In recent years, several tools have made it possible to prove many such predicates automatically or semi-automatically, even for infinite-state systems (e.g., [15, 40, 43]).

Our goal in this work is to expand the scope of those methods and tools. We aim to apply them to important security properties that have been hard to prove and that cannot be easily phrased as predicates on system behaviors. Many such properties can

*A preliminary version of this work was presented at the 20th IEEE Symposium on Logic in Computer Science (LICS 2005) [20].

be written as equivalences. For instance, the secrecy of a boolean parameter x of a protocol $P(x)$ may be written as the equivalence $P(\mathrm{true}) \approx P(\mathrm{false})$. Similarly, as is common in theoretical cryptography, we may wish to express the correctness of a construction P by comparing it to an ideal functionality Q, writing $P \approx Q$. Here the relation $\approx$ represents observational equivalence: $P \approx Q$ means that no context (that is, no attacker) can distinguish P and Q. A priori, $P \approx Q$ is not a simple predicate on the behaviors of P or Q.

We focus on proving equivalences $P \approx Q$ in which P and Q are two variants of the same process obtained by selecting different terms on the left and on the right. In particular, $P(\mathrm{true}) \approx P(\mathrm{false})$ is such an equivalence, since $P(\mathrm{true})$ and $P(\mathrm{false})$ differ only in the choice of value for the parameter x. Both $P(\mathrm{true})$ and $P(\mathrm{false})$ are variants of a process that we may write $P(\mathsf{diff}[\mathrm{true}, \mathrm{false}])$; the two variants are obtained by giving different interpretations to $\mathsf{diff}[\mathrm{true}, \mathrm{false}]$, making it select either true or false.

Although the notation diff can be viewed as a simple informal abbreviation, we find that there is some value in giving it a formal status. We define a calculus that supports diff. With a careful definition of the operational semantics of this calculus, we can establish the equivalence $P(\mathrm{true}) \approx P(\mathrm{false})$ by reasoning about behaviors of $P(\mathsf{diff}[\mathrm{true}, \mathrm{false}])$.

In this operational semantics, $P(\mathsf{diff}[\mathrm{true}, \mathrm{false}])$ behaves like both $P(\mathrm{true})$ and $P(\mathrm{false})$ from the point of view of the attacker, as long as the attacker cannot distinguish $P(\mathrm{true})$ and $P(\mathrm{false})$. The semantics requires that the results of reducing $P(\mathrm{true})$ and $P(\mathrm{false})$ can be written as a process with subexpressions of the form $\mathsf{diff}[M_1, M_2]$. On the other hand, when $P(\mathrm{true})$ and $P(\mathrm{false})$ would do something that may differentiate them, the semantics specifies that the execution of $P(\mathsf{diff}[\mathrm{true}, \mathrm{false}])$ gets stuck. Hence, if no behavior of $P(\mathsf{diff}[\mathrm{true}, \mathrm{false}])$ ever gets stuck, then $P(\mathrm{true}) \approx P(\mathrm{false})$. Thus, we can prove equivalences by reasoning about behaviors, though not the behaviors of the original processes in isolation.

This technique applies not only to an equivalence $P(\mathrm{true}) \approx P(\mathrm{false})$ that represents the concealment of a boolean parameter, but to a much broader class of equivalences that arise in security analysis and that go beyond secrecy properties. In principle, every equivalence could be rewritten as an equivalence in our class: we might try to prove $P \approx Q$ by examining the behaviors of

$$\mathit{if}\ \mathsf{diff}[\mathrm{true}, \mathrm{false}] = \mathrm{true}\ \mathit{then}\ P\ \mathit{else}\ Q$$

This observation suggests that we should not expect completeness for an automatic technique. Indeed, the class of equivalences that we can establish automatically does not include some traditional bisimilarities. Accordingly, we aim to complement, not to replace, other proof methods. Moreover, we are primarily concerned with soundness and usefulness, and (in contrast with some related work [7, 23–25, 29, 38]) we emphasize simplicity and automation over generality. We believe, however, that the use of diff is not "just a hack", because diff is amenable to a rigorous treatment and because operators much like diff have already proved useful in other contexts—in particular, in elegant soundness proofs of information-flow type systems [44, 45]. Baudet's recent thesis includes a further study of diff and obtains a decidability result for processes without replication [12].

We implement our technique in the tool ProVerif [15]. This tool is a protocol analyzer for protocols written in the applied pi calculus [6], an extension of the pi calculus with function symbols that may represent cryptographic operations. Internally, ProVerif translates protocols to Horn clauses in classical logic, and uses resolution on these clauses. The mapping to classical logic (rather than linear logic) embodies a safe abstraction which ignores the number of repetitions of each action, and which is key to the treatment of infinite-state systems [19]. We extend the translation into Horn clauses and also the manipulation of these Horn clauses.

While the implementation in ProVerif requires a non-trivial development of theory and code, it is rather fruitful. It enables us to treat, automatically, interesting proofs of equivalences. In particular, as in previous ProVerif proofs, it does not require that all systems under consideration be finite-state. We demonstrate these points through small examples and larger applications.

Specifically, we apply our technique to an infinite-state analysis of the important Encrypted Key Exchange (EKE) protocol [13, 14]. (Password-based protocols such as EKE have attracted much attention in recent years, partly because of the difficulty of reasoning about them.) We also use our technique for checking certain equivalences that express authenticity properties in an example from the literature [8]. In other applications, automated proofs of equivalences serve as lemmas for manual proofs of other results. We illustrate this combination by revisiting proofs for the JFK protocol [9].

One of the main features of the approach presented in this paper is that it is compatible with the inclusion of equational theories on function symbols. We devote considerable attention to their proper, sound integration. Those equational theories serve in modelling properties of the underlying cryptographic operations; they are virtually necessary in many applications. For instance, an equational theory may describe a decryption function that returns "junk" when its input is not a ciphertext under the expected key. Without equational theories, we may be able to model decryption only as a destructor that fails when there is a mismatch between ciphertext and key. Because the failure of decryption would be observable, it can result in false indications of attacks. Our approach overcomes this problem.

In contrast, a previous method for proving equivalences with ProVerif [17] does not address equivalences that depend on equational theories. Moreover, that method applies only to pairs of processes in which the terms that differ are global constants, not arbitrary terms. In these respects, the approach presented in this paper constitutes a clear advance. It enables significant proofs that were previously beyond the reach of automated techniques.

ProVerif belongs in a substantial body of work on sound, useful, but incomplete methods for protocol analysis. These methods rely on a variety of techniques from the programming-language literature, such as type systems, control-flow analyses, and abstract interpretation (e.g., [1, 22, 37, 42]). The methods are of similar power for proving predicates on behaviors [3, 21]. On the other hand, they typically do not target proofs of equivalences, or treat only specific classes of equivalences for particular equational theories.

The next section describes the process calculus that serves as setting for this work. Section 3 defines and studies observational equivalence. Section 4 contains some examples. Section 5 deals with equational theories. Section 6 explains how ProVerif

$M, N ::=$	terms
x, y, z	variable
a, b, c, k, s	name
$f(M_1, \ldots, M_n)$	constructor application
$D ::=$	term evaluations
M	term
eval $h(D_1, \ldots, D_n)$	function evaluation
$P, Q, R ::=$	processes
$M(x).P$	input
$\overline{M}\langle N \rangle.P$	output
$\mathbf{0}$	nil
$P \mid Q$	parallel composition
$!P$	replication
$(\nu a)P$	restriction
$let\ x = D\ in\ P\ else\ Q$	term evaluation

Figure 1: Syntax for terms and processes

maps protocols with diff to Horn clauses. Section 7 is concerned with proof techniques for those Horn clauses. Section 8 introduces a simple construct for breaking protocols into stages, as a convenience for applications. Section 9 describes applications. Section 10 mentions other related work and concludes. The Appendix contains proofs. The proof scripts for all examples and applications of this paper, as well as the tool ProVerif, are available at `http://www.di.ens.fr/~blanchet/obsequi/`.

2 The process calculus

This section introduces our process calculus, by giving its syntax and its operational semantics. This calculus is a combination of the original applied pi calculus [6] with one of its dialects [17]. This choice of calculus gives us the richness of the original applied pi calculus (in particular with regard to equational theories) while enabling us to leverage ProVerif.

2.1 Syntax and informal semantics

Figure 1 summarizes the syntax of our calculus. It defines a category of terms (data) and processes (programs). It assumes an infinite set of names and an infinite set of variables; a, b, c, k, s, and similar identifiers range over names, and x, y, and z range over variables. It also assumes a signature Σ (a set of function symbols, with arities and with associated definitions as explained below). We distinguish two categories of function symbols: constructors and destructors. We often write f for a constructor, g for a destructor, and h for a constructor or a destructor. Constructors are used for building terms. Thus, the terms $M, N, \ldots$ are variables, names, and constructor applications of the form $f(M_1, \ldots, M_n)$.

As in the applied pi calculus [6], terms are subject to an equational theory. Identifying an equational theory with its signature Σ, we write $\Sigma \vdash M = N$ for an equality modulo the equational theory, and $\Sigma \vdash M \neq N$ an inequality modulo the equational theory. (We write $M = N$ and $M \neq N$ for syntactic equality and inequality, respectively.) The equational theory is defined by a finite set of equations $\Sigma \vdash M_i = N_i$, where M_i and N_i are terms that contain only constructors and variables. The equational theory is then obtained from this set of equations by reflexive, symmetric, and transitive closure, closure by substitution (for any substitution σ, if $\Sigma \vdash M = N$ then $\Sigma \vdash \sigma M = \sigma N$), and closure by context application (if $\Sigma \vdash M = N$ then $\Sigma \vdash M'\{M/x\} = M'\{N/x\}$, where $\{M/x\}$ is the substitution that replaces x with M). We assume that there exist M and N such that $\Sigma \vdash M \neq N$.

As previously implemented in ProVerif, destructors are partial, non-deterministic operations on terms that processes can apply. More precisely, the semantics of a destructor g of arity n is given by a finite set $\mathrm{def}_\Sigma(g)$ of rewrite rules $g(M'_1, \ldots, M'_n) \to M'$, where $M'_1, \ldots, M'_n, M'$ are terms that contain only constructors and variables, the variables of M' are bound in $M'_1, \ldots, M'_n$, and variables are subject to renaming. Then $g(M_1, \ldots, M_n)$ is defined if and only if there exists a substitution σ and a rewrite rule $g(M'_1, \ldots, M'_n) \to M'$ in $\mathrm{def}_\Sigma(g)$ such that $M_i = \sigma M'_i$ for all $i \in \{1, \ldots, n\}$, and in this case $g(M_1, \ldots, M_n) \to \sigma M'$. In order to avoid distinguishing constructors and destructors in the definition of term evaluation, we let $\mathrm{def}_\Sigma(f)$ be $\{f(x_1, \ldots, x_n) \to f(x_1, \ldots, x_n)\}$ when f is a constructor of arity n.

The process $\mathit{let}\ x = D\ \mathit{in}\ P\ \mathit{else}\ Q$ tries to evaluate D; if this succeeds, then x is bound to the result and P is executed, else Q is executed. Here the reader may ignore the prefix eval which may occur in D, since $\mathrm{eval}\ f$ and f have the same semantics when f is a constructor, and destructors are used only with eval. In Section 5, we distinguish $\mathrm{eval}\ f$ and f in order to indicate when terms are evaluated.

Using constructors, destructors, and equations, we can model various data structures (tuples, lists, ...) and cryptographic primitives (shared-key encryption, public-key encryption, signatures, ...). Typically, destructors represent primitives that can visibly succeed or fail, while equations apply to primitives that always succeed but may sometimes return "junk". For instance, suppose that one can detect whether shared-key decryption succeeds or fails; then we would use a constructor enc, a destructor dec, and the rewrite rule $dec(enc(x, y), y) \to x$. Otherwise, we would use two constructors enc and dec, and the equations $dec(enc(x, y), y) = x$ and $enc(dec(x, y), y) = x$. The second equation prevents that the equality test $enc(dec(M, N), N) = M$ reveal that M must be a ciphertext under N. (The first equation is standard; the second is not, but it holds for block ciphers.) We refer to previous work [6, 17] for additional explanations and examples.

The rest of the syntax of Figure 1 is fairly standard pi calculus. The input process $M(x).P$ inputs a message on channel M, and executes P with x bound to the input message. The output process $\overline{M}\langle N\rangle.P$ outputs the message N on channel M and then executes P. (We allow M to be an arbitrary term; we could require that M be a name, as is frequently done, and adapt other definitions accordingly.) The nil process $\mathbf{0}$ does nothing and is sometimes omitted in examples. The process $P \mid Q$ is the parallel composition of P and Q. The replication $!P$ represents an unbounded number of copies of P in parallel. The restriction $(\nu a)P$ creates a new name a, and then executes P.

$M \Downarrow M$

eval $h(D_1, \ldots, D_n) \Downarrow \sigma N$
  if $h(N_1, \ldots, N_n) \to N \in \mathrm{def}_\Sigma(h)$,
  and σ is such that for all i, $D_i \Downarrow M_i$ and $\Sigma \vdash M_i = \sigma N_i$

$P \mid 0 \equiv P$	$P \equiv P$
$P \mid Q \equiv Q \mid P$	$Q \equiv P \Rightarrow P \equiv Q$
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$
$(\nu a)(\nu b)P \equiv (\nu b)(\nu a)P$	$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$
$(\nu a)(P \mid Q) \equiv P \mid (\nu a)Q$	$P \equiv Q \Rightarrow (\nu a)P \equiv (\nu a)Q$
if $a \notin fn(P)$	

$\overline{N}\langle M \rangle.Q \mid N'(x).P \to Q \mid P\{M/x\}$
  if $\Sigma \vdash N = N'$ (Red I/O)

$let\ x = D\ in\ P\ else\ Q \to P\{M/x\}$
  if $D \Downarrow M$ (Red Fun 1)

$let\ x = D\ in\ P\ else\ Q \to Q$
  if there is no M such that $D \Downarrow M$ (Red Fun 2)

$!P \to P \mid !P$ (Red Repl)

$P \to Q \Rightarrow P \mid R \to Q \mid R$ (Red Par)

$P \to Q \Rightarrow (\nu a)P \to (\nu a)Q$ (Red Res)

$P' \equiv P, P \to Q, Q \equiv Q' \Rightarrow P' \to Q'$ (Red $\equiv$)

Figure 2: Semantics for terms and processes

The syntax does not include the conditional $if\ M = N\ then\ P\ else\ Q$, which can be defined as $let\ x = \mathrm{equals}(M, N)\ in\ P\ else\ Q$ where x is a fresh variable and equals is a binary destructor with the rewrite rule $\mathrm{equals}(x, x) \to x$. We always include this destructor in Σ.

We write $fn(P)$ and $fv(P)$ for the sets of names and variables free in P, respectively, which are defined as usual. A process is closed if it has no free variables; it may have free names. We identify processes up to renaming of bound names and variables. An evaluation context C is a closed context built from $[\,]$, $C \mid P$, $P \mid C$, and $(\nu a)C$.

2.2 Formal semantics

The rules of Figure 2 axiomatize the reduction relation for processes ($\to_\Sigma$), thus defining the operational semantics of our calculus. Auxiliary rules define term evaluation ($\Downarrow_\Sigma$) and the structural congruence relation ($\equiv$); this relation is useful for transforming processes so that the reduction rules can be applied. Both $\equiv$ and $\to_\Sigma$ are defined only on closed processes. We write $\to_\Sigma^*$ for the reflexive and transitive closure of $\to_\Sigma$, and $\to_\Sigma^* \equiv$ for its union with $\equiv$. When Σ is clear from the context, we abbreviate $\to_\Sigma$ and $\Downarrow_\Sigma$ to $\to$ and $\Downarrow$, respectively.

This semantics differs in minor ways from the semantics of the applied pi calculus [6]. In particular, we do not substitute equals for equals in structural congruence, but only in a controlled way in certain rules. Thus, the rule for I/O does not require

a priori that the input and output channels be equal: it explicitly uses the equational theory to compare them. We also use a reduction rule (Red Repl) for modelling replication, instead of the more standard, but essentially equivalent, structural congruence rule. This weakening of structural congruence in favor of the reduction relation is designed to simplify our proofs.

3 Observational equivalence

In this section we introduce diff formally and establish a sufficient condition for observational equivalence. We first recall the standard definition of observational equivalence from the pi calculus:

Definition 1 The process P *emits on* M ($P \downarrow_M$) if and only if $P \equiv C[\overline{M'}\langle N\rangle.R]$ for some evaluation context C that does not bind $fn(M)$ and $\Sigma \vdash M = M'$.

(Strong) observational equivalence $\sim$ is the largest symmetric relation $\mathcal{R}$ on closed processes such that $P\ \mathcal{R}\ Q$ implies

1. if $P \downarrow_M$ then $Q \downarrow_M$;
2. if $P \to P'$ then $Q \to Q'$ and $P'\ \mathcal{R}\ Q'$ for some Q';
3. $C[P]\ \mathcal{R}\ C[Q]$ for all evaluation contexts C.

Weak observational equivalence $\approx$ is defined similarly, with $\to^* \downarrow_M$ instead of $\downarrow_M$ and $\to^*$ instead of $\to$.

Intuitively, a context may represent an adversary, and two processes are observationally equivalent when no adversary can distinguish them.

Next we introduce a new calculus that can represent pairs of processes that have the same structure and differ only by the terms and term evaluations that they contain. We call such a pair of processes a *biprocess*. The grammar for the calculus is a simple extension of the grammar of Figure 1, with additional cases so that $\mathsf{diff}[M, M']$ is a term and $\mathsf{diff}[D, D']$ is a term evaluation. We also extend the definition of contexts to permit the use of diff, and sometimes refer to contexts without diff as plain contexts.

Given a biprocess P, we define two processes $\mathsf{fst}(P)$ and $\mathsf{snd}(P)$, as follows: $\mathsf{fst}(P)$ is obtained by replacing all occurrences of $\mathsf{diff}[M, M']$ with M and $\mathsf{diff}[D, D']$ with D in P, and similarly, $\mathsf{snd}(P)$ is obtained by replacing $\mathsf{diff}[M, M']$ with M' and $\mathsf{diff}[D, D']$ with D' in P. We define $\mathsf{fst}(D)$, $\mathsf{fst}(M)$, $\mathsf{snd}(D)$, and $\mathsf{snd}(M)$ similarly. Our goal is to show that the processes $\mathsf{fst}(P)$ and $\mathsf{snd}(P)$ are observationally equivalent:

Definition 2 Let P be a closed biprocess. We say that P *satisfies observational equivalence* when $\mathsf{fst}(P) \sim \mathsf{snd}(P)$.

The semantics for biprocesses is defined as in Figure 2 with generalized rules (Red I/O), (Red Fun 1), and (Red Fun 2) given in Figure 3. Reductions for biprocesses bundle those for processes: if $P \to Q$ then $\mathsf{fst}(P) \to \mathsf{fst}(Q)$ and $\mathsf{snd}(P) \to \mathsf{snd}(Q)$.

$\overline{N}\langle M\rangle.Q \mid N'(x).P \;\rightarrow\; Q \mid P\{M/x\}$ (Red I/O)
if $\Sigma \vdash \mathsf{fst}(N) = \mathsf{fst}(N')$ and $\Sigma \vdash \mathsf{snd}(N) = \mathsf{snd}(N')$

$\mathit{let}\ x = D\ \mathit{in}\ P\ \mathit{else}\ Q \rightarrow P\{\mathsf{diff}[M_1, M_2]/x\}$ (Red Fun 1)
if $\mathsf{fst}(D) \Downarrow M_1$ and $\mathsf{snd}(D) \Downarrow M_2$

$\mathit{let}\ x = D\ \mathit{in}\ P\ \mathit{else}\ Q \rightarrow Q$ (Red Fun 2)
if there is no M_1 such that $\mathsf{fst}(D) \Downarrow M_1$ and
there is no M_2 such that $\mathsf{snd}(D) \Downarrow M_2$

Figure 3: Generalized rules for biprocesses

Conversely, however, reductions in $\mathsf{fst}(P)$ and $\mathsf{snd}(P)$ need not correspond to any biprocess reduction, in particular when they do not match up. Our first theorem shows that the processes are equivalent when this does not happen.

Definition 3 We say that the biprocess P is *uniform* when $\mathsf{fst}(P) \rightarrow Q_1$ implies that $P \rightarrow Q$ for some biprocess Q with $\mathsf{fst}(Q) \equiv Q_1$, and symmetrically for $\mathsf{snd}(P) \rightarrow Q_2$.

Theorem 1 *Let P_0 be a closed biprocess. If, for all plain evaluation contexts C and reductions $C[P_0] \rightarrow^* P$, the biprocess P is uniform, then P_0 satisfies observational equivalence.*

Proof Let P be a closed biprocess such that $C[P] \rightarrow^* \equiv Q$ always yields a uniform biprocess Q, and consider the relation

$$\mathcal{R} = \{(\mathsf{fst}(Q), \mathsf{snd}(Q)) \mid C[P] \rightarrow^* \equiv Q\}$$

In particular, we have $\mathsf{fst}(P)\ \mathcal{R}\ \mathsf{snd}(P)$, so we can show that P satisfies observational equivalence by establishing that the relation $\mathcal{R}' = \mathcal{R} \cup \mathcal{R}^{-1}$ meets the three conditions of Definition 1. By symmetry, we focus on $\mathcal{R}$. Assume $\mathsf{fst}(Q)\ \mathcal{R}\ \mathsf{snd}(Q)$.

1. Assume $\mathsf{fst}(Q) \downarrow_M$, and let $T_M = M(x).\overline{c}\langle c\rangle$ for some fresh name c. As usual in the pi calculus, the predicate $_\downarrow_M$ tests the ability to send any message on M, hence for any plain process Q_i, we have $Q_i \downarrow_M$ if and only if $Q_i \mid T_M \rightarrow R_i \mid \overline{c}\langle c\rangle$ for some R_i.

   Here, we have $\mathsf{fst}(Q) \mid T_M \rightarrow R_1 \mid \overline{c}\langle c\rangle$ for some R_1. The reductions $C[P] \rightarrow^* \equiv Q$ imply $C[P] \mid T_M \rightarrow^* \equiv Q \mid T_M$. By hypothesis (with the context $C[_] \mid T_M$), $Q \mid T_M$ is uniform, hence $Q \mid T_M \rightarrow Q'$ for some Q' with $\mathsf{fst}(Q') \equiv R_1 \mid \overline{c}\langle c\rangle$. Since c does not occur anywhere in Q, by case analysis on this reduction step with our semantics for biprocesses we obtain $Q' \equiv R \mid \overline{c}\langle c\rangle$ for some biprocess R. Thus, we obtain $\mathsf{snd}(Q) \mid T_M \rightarrow \mathsf{snd}(R) \mid \overline{c}\langle c\rangle$, and finally $\mathsf{snd}(Q) \downarrow_M$.

2. If $\mathsf{fst}(Q) \rightarrow Q'_1$ then, by uniformity, we have $Q \rightarrow Q'$ with $\mathsf{fst}(Q') = Q'_1$. Thus, $C[P] \rightarrow^* \equiv \rightarrow Q'$ and, by definition of $\mathcal{R}$, we obtain $\mathsf{fst}(Q')\ \mathcal{R}\ \mathsf{snd}(Q')$. Finally, by definition of the semantics of biprocesses, $Q \rightarrow Q'$ implies $\mathsf{snd}(Q) \rightarrow \mathsf{snd}(Q')$.

3. Let C' be a plain evaluation context. By definition of the semantics of biprocesses, $C[P] \rightarrow^* \equiv Q$ always implies $C'[C[P]] \rightarrow^* \equiv C'[Q]$, hence $C'[\mathsf{fst}(Q)] = \mathsf{fst}(C'[Q])\ \mathcal{R}\ \mathsf{snd}(C'[Q]) = C'[\mathsf{snd}(Q)]$. □

Our plan is to establish the hypothesis of Theorem 1 by automatically verifying that all the biprocesses P in question meet conditions that imply uniformity. The next corollary details those conditions, which guarantee that a communication and an evaluation, respectively, succeed in $\mathsf{fst}(P)$ if and only if they succeed in $\mathsf{snd}(P)$:

Corollary 1 *Let P_0 be a closed biprocess. Suppose that, for all plain evaluation contexts C, all evaluation contexts C', and all reductions $C[P_0] \rightarrow^* P$,*

1. *if $P \equiv C'[\overline{N}\langle M\rangle.Q \mid N'(x).R]$, then $\Sigma \vdash \mathsf{fst}(N) = \mathsf{fst}(N')$ if and only if $\Sigma \vdash \mathsf{snd}(N) = \mathsf{snd}(N')$,*
2. *if $P \equiv C'[let\ x = D\ in\ Q\ else\ R]$, then there exists M_1 such that $\mathsf{fst}(D) \Downarrow M_1$ if and only if there exists M_2 such that $\mathsf{snd}(D) \Downarrow M_2$.*

Then P_0 satisfies observational equivalence.

Proof We show that P is uniform, then we conclude by Theorem 1. Let us show that, if $\mathsf{fst}(P) \rightarrow P'_1$ then there exists a biprocess P' such that $P \rightarrow P'$ and $\mathsf{fst}(P') \equiv P'_1$. The case for $\mathsf{snd}(P) \rightarrow P'_2$ is symmetric.

By induction on the derivation of $\mathsf{fst}(P) \rightarrow P'_1$, we first show that there exist C, Q, and Q'_1 such that $P \equiv C[Q]$, $P'_1 \equiv \mathsf{fst}(C)[Q'_1]$, and $\mathsf{fst}(Q) \rightarrow Q'_1$ using one of the four process rules (Red I/O), (Red Fun 1), (Red Fun 2), or (Red Repl): every step in this derivation trivially commutes with fst, except for structural steps that involve a parallel composition and a restriction, in case $a \in fn(P)$ but $a \notin fn(\mathsf{fst}(P))$. In that case, we use a preliminary renaming from a to some fresh $a' \notin fn(P)$.

For each of these four rules, relying on a hypothesis of Corollary 1, we find Q' such that $\mathsf{fst}(Q') = Q'_1$ and $Q \rightarrow Q'$ using the corresponding biprocess rule:

(Red I/O): We have $Q = \overline{N}\langle M\rangle.R \mid N'(x).R'$ with $\Sigma \vdash \mathsf{fst}(N) = \mathsf{fst}(N')$ and $Q'_1 = \mathsf{fst}(R) \mid \mathsf{fst}(R')\{\mathsf{fst}(M)/x\}$. For $Q' = R \mid R'\{M/x\}$, we have $\mathsf{fst}(Q') = Q'_1$ and, by hypothesis 1, $\Sigma \vdash \mathsf{snd}(N) = \mathsf{snd}(N')$, hence $Q \rightarrow Q'$.

(Red Fun 1): We have $Q = let\ x = D\ in\ R\ else\ R'$ with $\mathsf{fst}(D) \Downarrow M_1$ and $Q'_1 = \mathsf{fst}(R)\{M_1/x\}$. By hypothesis 2, $\mathsf{snd}(D) \Downarrow M_2$ for some M_2. We take $Q' = R\{\mathsf{diff}[M_1, M_2]\}$, so that $\mathsf{fst}(Q') = Q'_1$ and $Q \rightarrow Q'$.

(Red Fun 2): We have $Q = let\ x = D\ in\ R\ else\ R'$ with no M_1 such that $\mathsf{fst}(D) \Downarrow M_1$ and $Q'_1 = \mathsf{fst}(R')$. By hypothesis 2, there is no M_2 such that $\mathsf{snd}(D) \Downarrow M_2$. We obtain $Q \rightarrow Q'$ for $Q' = R'$.

(Red Repl): We have $Q = !R$ and $Q'_1 = \mathsf{fst}(R) \mid !\mathsf{fst}(R)$. We take $Q' = R \mid !R$, so that $\mathsf{fst}(Q') = Q'_1$ and $Q \rightarrow Q'$.

To conclude, we take the biprocess $P' = C[Q']$ and the reduction $P \rightarrow P'$. □

Thus, we have a sufficient condition for observational equivalence of biprocesses. This condition is essentially a reachability condition on biprocesses. Starting in Section 5, we adapt existing techniques for reasoning about processes in order to prove this condition. The condition is however not necessary: as suggested in the introduction, if $P \sim Q$, then $\mathit{if}\ \mathsf{diff}[\mathrm{true}, \mathrm{false}] = \mathrm{true}\ \mathit{then}\ P\ \mathit{else}\ Q$ satisfies observational equivalence, but Theorem 1 and Corollary 1 will not enable us to prove this fact.

4 Examples in the applied pi calculus

This section illustrates our approach by revisiting examples of observational equivalences presented with the applied pi calculus [6]. Interestingly, all those equivalences can be formulated using biprocesses, proved via Theorem 1 and, it turns out, verified automatically by ProVerif. Section 9 presents more complex examples.

We begin with equivalences that can be expressed with biprocesses that perform a single output, of the form $(\nu a_1, \ldots, a_k)\overline{c}\langle M\rangle$ where c is a name that does not occur in $a_1, \ldots, a_k$ or in M. Intuitively, such equivalences state that no environment can differentiate $\mathsf{fst}(M)$ from $\mathsf{snd}(M)$ without knowing some name in a_1, ..., a_k. Such equivalences on terms under restrictions are called static equivalences [6]. They arise when one considers attackers that first intercept a series of messages, then attempt to differentiate two configurations of the protocol by computing on those messages without interacting with the protocol further. Here, the term M may be a tuple $\mathsf{diff}[M_1, M_1']$, ..., $\mathsf{diff}[M_n, M_n']$ that collects all pairs of intercepted messages, and a_1 ,..., a_k may be names that represent all local secrets and fresh values used by the protocol.

Static equivalences play a central role in the extension of proof techniques from the pure pi calculus to the applied pi calculus. In particular, observational equivalence in the applied pi calculus can be reduced to standard pi calculus requirements plus static equivalences [6]. In other words, proofs of observational equivalences can be decomposed into lemmas that deal with terms and general arguments that relate processes with different structures; the former depend on the signature, while the latter come from the pure pi calculus. In our experience, a large fraction of the proof effort is typically devoted to those lemmas on terms, and Theorem 1 is a good tool for establishing them.

Example 1 Consider a cryptographic hash function, modelled as a constructor h with neither rewrite rule nor equation. The environment should not be able to distinguish a freshly generated random value, modelled as a fresh name a, from its hash $h(a)$ [6, Section 4.2]. Formally, using the automated technique presented in this paper, we verify that the biprocess $(\nu a)\overline{c}\langle\mathsf{diff}[a, h(a)]\rangle$ satisfies equivalence. On the other hand, $P = (\nu a, a')\overline{c}\langle(a, \mathsf{diff}[a', h(a)])\rangle$ does not satisfy equivalence: although both processes emit a pair of fresh terms, the environment can distinguish one process from the other by computing a hash of the first element of the pair and comparing it to the second element of the pair, using the context

$$C[_] = c(x, y).\mathit{if}\ y = h(x)\ \mathit{then}\ \overline{d}\langle c\rangle\ \mathit{else}\ 0 \mid [_]$$

With our biprocess semantics, $C[P]$ performs a (Red I/0) step then gets stuck on the test $(\nu a, a')$*if* $\mathsf{diff}[a', h(a)] = h(a)$ *then* $\overline{d}\langle c\rangle$ *else* 0. □

Example 2 Diffie-Hellman computations used in key agreement protocols can be expressed in terms of a constant b, a binary constructor $\hat{}$, and the equation $(\mathsf{b}\hat{}x)\hat{}y = (\mathsf{b}\hat{}y)\hat{}x$ [4, 6]. With this signature, we verify that

$$(\nu a_1, a_2, a_3)\overline{c}\langle(\mathsf{b}\hat{}a_1, \mathsf{b}\hat{}a_2, \mathsf{diff}[(\mathsf{b}\hat{}a_1)\hat{}a_2, \mathsf{b}\hat{}a_3])\rangle$$

satisfies equivalence. This equivalence closely corresponds to the Decisional Diffie-Hellman assumption often made by cryptographers; it is also the main lemma in the proof of [6, Theorem 3]. Intuitively, even if the environment is given access to the exponentials $\mathsf{b}\hat{}a_1$ and $\mathsf{b}\hat{}a_2$, those values are (apparently) unrelated to the Diffie-Hellman secret $(\mathsf{b}\hat{}a_1)\hat{}a_2$, since the environment cannot distinguish this secret from the exponential of any fresh unrelated value a_3. □

The remaining two examples concern applications beyond proofs of static equivalences.

Example 3 Non-deterministic encryption is a variant of public-key encryption that further protects the secrecy of the plaintext by embedding some additional, fresh value in each encryption. It can be modelled using three functions for public-key decryption, public-key encryption, and public-key derivation, linked by the equation

$$pdec(penc(x, pk(y), z), y) = x$$

where z is the additional parameter for the encryption. A key property of non-deterministic encryption is that, without knowledge of the decryption key, ciphertexts appear to be unrelated to the plaintexts, even if the attacker knows the plaintexts and the encryption key. A strong version of this property is that the ciphertexts cannot be distinguished from freshly generated random values. Formally, we state that

$$(\nu s)(\overline{c}\langle pk(s)\rangle \mid !c'(x).(\nu a)\overline{c}\langle\mathsf{diff}[penc(x, pk(s), a), a]\rangle)$$

satisfies equivalence. This biprocess is more complex than those presented above; instead of a single output, it performs a first output to reveal the public key $pk(s)$ (but not s!), then repeatedly inputs a term x from the environment and either outputs its encryption under $pk(s)$ or outputs a fresh, unrelated name. Thus, a single biprocess represents the family of static equivalences that relate a series of non-deterministic encryptions for any series of plaintext to a series of fresh, independent names. (Formally, each such equivalence can be obtained as a corollary of this biprocess equivalence, by applying the congruence property of equivalence for the particular context that sends the plaintexts of values on channel c' and reads the encryption key and encryptions on channel c.) □

Example 4 Biprocesses can also be used for relating an abstract specification of a cryptographic primitive with its implementation in terms of lower-level functions. As

an example, we consider the construction of message authentication codes (MACs) for messages of arbitrary length, as modelled in the applied pi calculus [6, Section 6]. MAC functions are essentially keyed hash functions; MACs should not be subject to tampering or forgery. More formally, the usage of MACs can be captured via a little protocol that generates MACs on demand and checks them:

$$\begin{aligned} P_0 = (\nu k)(!c'(x).\overline{c}\langle x, mac(k,x)\rangle \\ \quad | \, c(x,y).\mathit{if}\ y = mac(k,x)\ \mathit{then}\ \overline{c''}\langle x\rangle) \end{aligned}$$

The unforgeability of MACs means that the MAC checker succeeds and forwards a message x on c'' only if a MAC has been generated for x by sending it to the MAC generator on c'.

Let P be P_0 with the term $\mathsf{diff}[mac(k,x), impl(k,x)]$ instead of the two occurrences of $mac(k,x)$. For a given signature with no equation for mac, a function $impl$ may be said to implement mac correctly when P satisfies equivalence. With this formulation, we can verify the correctness of the second construction considered in [6], $impl(k,x) = f(k, f(k,x))$, with equation $f(k,(x,y)) = h(f(k,x),y)$, where f is a keyed hash function that iterates a compression function h on the message blocks. We can also confirm that the first construction considered in [6], $impl(k,x) = f(k,x)$ with the same equation $f(k,(x,y)) = h(f(k,x),y)$, is subject to a standard extension attack: anyone that obtains the MAC $impl(k,N_1)$ can produce the MAC $impl(k,(N_1,N_2))$ as $h(impl(k,N_1),N_2)$ for any message extension N_2 without knowing k. □

5 Modelling equations with rewrite rules

We handle equations by translating from a signature with equations to a signature without equations. This translation is designed to ease implementation: with it, resolution can continue to rely on ordinary syntactic unification, and remains very efficient. Although our technique is general and automatic, it does have limitations: it does not apply to some equational theories, in particular theories with associative symbols such as XOR. (It may be possible to handle some of those theories by shifting from syntactic unification to unification modulo the theory in question, at the cost of increased complexity.)

5.1 Definitions

We consider an auxiliary rewriting system on terms, $\mathcal{S}$, that defines partial normal forms. The terms manipulated by $\mathcal{S}$ do not contain diff, but they may contain variables. The rules of $\mathcal{S}$ do not contain names and do not have a single variable on the left-hand side. We say that a term is irreducible by $\mathcal{S}$ when none of the rewrite rules of $\mathcal{S}$ applies to it; we say that the set of terms $\mathcal{M}$ is in normal form relatively to $\mathcal{S}$ and Σ, and write $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{M})$, if and only if all terms of $\mathcal{M}$ are irreducible by $\mathcal{S}$ and, for all subterms N_1 and N_2 of terms of $\mathcal{M}$, if $\Sigma \vdash N_1 = N_2$ then $N_1 = N_2$. Intuitively, we allow for the possibility that terms may have several irreducible forms (see Example 6 below), requiring that $\mathcal{M}$ use irreducible forms consistently. This requirement implies, for

instance, that if the rewrite rule $f(x,x) \to x$ applies modulo the equational theory to a term $f(N_1, N_2)$ then N_1 and N_2 are identical and the rule $f(x,x) \to x$ also applies without invoking the equational theory. We extend the definition of $\mathrm{nf}_{\mathcal{S},\Sigma}(\cdot)$ to sets of processes: $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{P})$ if and only if the set of terms that appear in processes in $\mathcal{P}$ is in normal form.

For a signature Σ' (without equations), we define evaluation on open terms as a relation $D \Downarrow' (M, \sigma)$, where σ collects instantiations of D obtained by unification:

$$M \Downarrow' (M, \emptyset)$$

$$\begin{array}{l}\mathrm{eval}\ h(D_1, \ldots, D_n) \Downarrow' (\sigma_u N, \sigma_u \sigma') \\ \quad \text{if } (D_1, \ldots, D_n) \Downarrow' ((M_1, \ldots, M_n), \sigma'), \\ \quad h(N_1, \ldots, N_n) \to N \text{ is in } \mathrm{def}_{\Sigma'}(h) \text{ and} \\ \quad \sigma_u \text{ is a most general unifier of } (M_1, N_1), \ldots, (M_n, N_n)\end{array}$$

$$\begin{array}{l}(D_1, \ldots, D_n) \Downarrow' ((\sigma_n M_1, \ldots, \sigma_n M_{n-1}, M_n), \sigma_n \sigma) \\ \quad \text{if } (D_1, \ldots, D_{n-1}) \Downarrow' ((M_1, \ldots, M_{n-1}), \sigma) \text{ and } \sigma D_n \Downarrow' (M_n, \sigma_n)\end{array}$$

As suggested in Section 2, we rely on eval for indicating term evaluations: while $f(M_1, \ldots, M_n) \Downarrow' (f(M_1, \ldots, M_n), \emptyset)$, deriving $\mathrm{eval}\ f(M_1, \ldots, M_n) \Downarrow' (M, \sigma)$ requires an application of a rewrite rule for the constructor f.

We let $\mathrm{addeval}(M_1, \ldots, M_n)$ be the tuple of term evaluations obtained by adding eval before each function symbol of $M_1, \ldots, M_n$. Using these definitions, we describe when a signature Σ' with rewrite rules models another signature Σ with equations:

Definition 4 Let Σ and Σ' be signatures on the same function symbols. We say that Σ' models Σ if and only if

1. The equational theory of Σ' is syntactic equality: $\Sigma' \vdash M = N$ if and only if $M = N$.

2. The constructors of Σ' are the constructors of Σ; their definition $\mathrm{def}_{\Sigma'}(f)$ contains the rule $f(x_1, \ldots, x_n) \to f(x_1, \ldots, x_n)$, plus perhaps other rules such that there exists a rewriting system $\mathcal{S}$ on terms that satisfies the following properties:

   S1. If $M \to N$ is in $\mathcal{S}$, then $\Sigma \vdash M = N$.

   S2. If $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{M})$, then for any term M there exists M' such that $\Sigma \vdash M' = M$ and $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{M} \cup \{M'\})$.

   S3. If $f(N_1, \ldots, N_n) \to N$ is in $\mathrm{def}_{\Sigma'}(f)$, then $\Sigma \vdash f(N_1, \ldots, N_n) = N$.

   S4. If $\Sigma \vdash f(M_1, \ldots, M_n) = M$ and $\mathrm{nf}_{\mathcal{S},\Sigma}(\{M_1, \ldots, M_n, M\})$, then there exist σ and $f(N_1, \ldots, N_n) \to N$ in $\mathrm{def}_{\Sigma'}(f)$ such that $M = \sigma N$ and $M_i = \sigma N_i$ for all $i \in \{1, \ldots, n\}$.

3. The destructors of Σ' are the destructors of Σ, with a rule $g(M'_1, \ldots, M'_n) \to M'$ in $\mathrm{def}_{\Sigma'}(g)$ for each $g(M_1, \ldots, M_n) \to M$ in $\mathrm{def}_{\Sigma}(g)$ and each $\mathrm{addeval}(M_1, \ldots, M_n, M) \Downarrow' ((M'_1, \ldots, M'_n, M'), \sigma)$.

Condition 1 says that the equational theory of Σ' is trivial. In Condition 2, Properties S1 and S2 concern the relation of $\mathcal{S}$ and Σ. Property S1 guarantees that all rewrite rules of $\mathcal{S}$ are sound according to the equational theory of Σ. Property S2 requires there are "enough" normal forms: that for every term M there is an $\mathcal{S}$-irreducible Σ-equal term M', and that M' can be chosen consistently with a set $\mathcal{M}$ in normal form. Properties S3 and S4 concern the definition of constructors in Σ'. Property S3 guarantees that the rewrite rules that define the constructors are sound according to the equational theory of Σ. Property S4 requires that there are "enough" rewrite rules: basically, that when $M_1, \ldots, M_n$ are in normal form, every normal form of $f(M_1, \ldots, M_n)$ can be generated by applying a rewrite rule for f in Σ' to $f(M_1, \ldots, M_n)$. Finally, according to Condition 3, the definition of destructors in Σ' can be computed by applying the rewrite rules of constructors in Σ' to the definition of destructors in Σ.

According to this definition, we deal with any equations on f in Σ by evaluating f once in Σ'. (We use eval markers in expressions accordingly: $\mathrm{eval}\ f$ and f represent f before and after this evaluation, respectively.) This characteristic entails a limitation of our approach. For instance, suppose that we have $f(f'(x)) = f'(f(x))$ in the equational theory of Σ, and we want a Σ' that models Σ. In Σ', we should equate $f'(f(\ldots f(a)))$ and $f(\ldots f(f'(a)))$ by one reduction step, so we need one rewrite rule for each length of sequence of applications of f, so $\mathrm{def}_{\Sigma'}(f')$ cannot be finite. Associative symbols like XOR pose a similar problem.

5.2 Examples

The following two examples illustrate the definitions of Section 5.1. ProVerif handles these examples automatically, using the approach of Section 5.3.

Example 5 Suppose that Σ has the constructors enc and dec with the equations

$$dec(enc(x, y), y) = x \qquad enc(dec(x, y), y) = x$$

In Σ', we adopt the rewrite rules:

$$\begin{array}{ll} dec(x, y) \to dec(x, y) & enc(x, y) \to enc(x, y) \\ dec(enc(x, y), y) \to x & enc(dec(x, y), y) \to x \end{array}$$

We have that Σ' models Σ for the rewriting system $\mathcal{S}$ with rules $dec(enc(x, y), y) \to x$ and $enc(dec(x, y), y) \to x$, and a single normal form for every term. □

Example 6 In order to model the Diffie-Hellman equation of Example 2, we define Σ' with three rewrite rules:

$$\mathtt{b} \to \mathtt{b} \qquad x\mathtt{\hat{}}y \to x\mathtt{\hat{}}y \qquad (\mathtt{b}\mathtt{\hat{}}x)\mathtt{\hat{}}y \to (\mathtt{b}\mathtt{\hat{}}y)\mathtt{\hat{}}x$$

and use an empty $\mathcal{S}$. Intuitively, applying $\mathtt{\hat{}}$ to $(\mathtt{b}\mathtt{\hat{}}x)$ and y yields both possible forms of $(\mathtt{b}\mathtt{\hat{}}x)\mathtt{\hat{}}y$ modulo the equational theory, $(\mathtt{b}\mathtt{\hat{}}x)\mathtt{\hat{}}y$ and $(\mathtt{b}\mathtt{\hat{}}y)\mathtt{\hat{}}x$. Hence, a term M may have several irreducible forms M' that satisfy $\mathrm{nf}_{\mathcal{S},\Sigma}(\{M'\})$ and $\Sigma \vdash M' = M$: one can choose $(\mathtt{b}\mathtt{\hat{}}N)\mathtt{\hat{}}N'$ or $(\mathtt{b}\mathtt{\hat{}}N')\mathtt{\hat{}}N$. □

5.3 Algorithms

Next we explain a method for finding, for a given signature Σ, a signature Σ' that models Σ and a corresponding rewriting system $\mathcal{S}$. This method is embodied in algorithms that, when they terminate, yield the definition of $\mathrm{def}_{\Sigma'}(f)$ for each constructor of Σ. The definition of $\mathrm{def}_{\Sigma'}(g)$ for each destructor of Σ follows from Condition 3 of Definition 4. These algorithms do not always terminate because, for some equational theories, they generate an unbounded number of rewrite rules. However, they often terminate in practice, as our examples illustrate; moreover, Lemma 7 in Appendix A.2 establishes a termination result for a significant class of theories, the convergent subterm theories [5].

Our first algorithm handles convergent (terminating and confluent) theories. It applies, for instance, to Example 5. Here and elsewhere, we write T for a term context (a term with a hole).

Algorithm 1 (Convergent theories) Let $M_i = N_i$ (for $i \in \{1, \ldots, m\}$) be the equations that define the equational theory of Σ. Let $\mathcal{S}$ be defined by the rewrite rules $M_i \to N_i$. Assume that $\mathcal{S}$ is convergent, and let $M{\downarrow}$ be the normal form of M relatively to $\mathcal{S}$.

When E is a set of rewrite rules, we define $normalize(E)$ by
– replacing each rule $f(M_1, \ldots, M_n) \to N$ of E with $f(M_1{\downarrow}, \ldots, M_n{\downarrow}) \to N{\downarrow}$;
– removing rules of the form $M \to M$ from E;
– if $M \to N$ is in E, removing all other rules of the form $T[\sigma M] \to T[\sigma N]$ from E.

Let $E = normalize(\mathcal{S})$.
Repeat until E reaches a fixpoint
  For each pair of rules $M \to M'$ and $N \to N'$ in E and each T
    such that $M' = T[M'']$, M'' is not a variable,
    and σ_u is the most general unifier of M'' and N,
      set $E = normalize(E \cup \{\sigma_u M \to \sigma_u T[\sigma_u N']\})$.
For each constructor f,
  $\mathrm{def}_{\Sigma'}(f) = \{f(M_1, \ldots, M_n) \to N \in E\} \cup \{f(x_1, \ldots, x_n) \to f(x_1, \ldots, x_n)\}$.

In this algorithm, we add to E new rewrite rules obtained by composing two rewrite rules of E, until a fixpoint is reached. Lemma 7 in Appendix A.2 shows that a fixpoint is reached immediately for convergent subterm theories.

Before running the algorithm, we can check that $\mathcal{S}$ is convergent as follows.

- The termination of $\mathcal{S}$ can be established via a reduction ordering $>$, by showing that if $M \to M'$ is in $\mathcal{S}$, then $M > M'$. In the implementation, we use a lexicographic path ordering.
- The confluence of $\mathcal{S}$ can be established via the critical-pair theorem, by showing that all critical pairs are joinable [31].

Alternatively, one could use the Knuth-Bendix completion algorithm in order to transform a rewriting system into a convergent one.

Our next algorithm handles linear theories, such as that of Example 6.

Algorithm 2 (Linear theories) Let Σ be a signature such that all equations of Σ are linear: each variable occurs at most once in the left-hand side and at most once in the right-hand side. Let $\mathcal{S}$ be empty.

When E is a set of rewrite rules, we define $normalize(E)$ by:
– removing rules of the form $M \to M$ from E;
– if $M \to N$ is in E, removing all other rules of the form $T[\sigma M] \to T[\sigma N]$ from E.

Let $E = normalize(\{M \to N, N \to M \mid M = N \text{ is an equation of } \Sigma\})$.
Repeat until E reaches a fixpoint
  For each pair of rules $M \to M'$ and $N \to N'$ in E and each T
    such that $M' = T[M'']$, M'' and N are not variables,
    and σ_u is the most general unifier of M'' and N,
      set $E = normalize(E \cup \{\sigma_u M \to \sigma_u T[\sigma_u N']\})$.
  For each pair of rules $M \to M'$ and $N \to N'$ in E and each T
    such that $N = T[N'']$, M' and N'' are not variables,
    and σ_u is the most general unifier of M' and N'',
      set $E = normalize(E \cup \{\sigma_u T[\sigma_u M] \to \sigma_u N'\})$.
For each constructor f,
  $\mathrm{def}_{\Sigma'}(f) = \{f(M_1, \ldots, M_n) \to N \in E\} \cup \{f(x_1, \ldots, x_n) \to f(x_1, \ldots, x_n)\}$.

In this algorithm, when two rewrite rules of E have a critical pair with one another, we compose them and add the result to E.

Algorithms 1 and 2 are similar. The main difference is that Algorithm 1 performs additional optimizations that are sound for convergent rewriting systems but not for linear equational theories. In particular, in the initial definition of E, Algorithm 1 considers rewrite rules oriented only in the direction of $\mathcal{S}$, while Algorithm 2 considers both directions. Furthermore, in Algorithm 1, $normalize$ reduces the right-hand sides and the strict subterms of the left-hand sides of rewrite rules by $\mathcal{S}$, while Algorithm 2 does not include this reduction. As a consequence, the second way of combining rules of E in Algorithm 2 is not necessary in Algorithm 1, since the rules thus created would be reduced by $normalize$ into an instance of an already existing rule.

Our final algorithm combines the two previous ones:

Algorithm 3 (Union) Let Σ be a signature.
Split the set of equations of Σ into subsets E_i that use disjoint sets of constructors.
Let E_{conv} be the union of those subsets E_i that we can prove convergent.
Let E_{lin} be the union of those subsets E_i that are linear and are not in E_{conv}.
If some subsets E_i are neither in E_{conv} nor in E_{lin}, then fail.
Apply Algorithm 1 to E_{conv}, obtaining the rewriting system $\mathcal{S}_{\mathrm{conv}}$ and the definition $\mathrm{def}_{\Sigma'}(f)$ of the constructors of E_{conv}.
Apply Algorithm 2 to E_{lin}, obtaining the definition $\mathrm{def}_{\Sigma'}(f)$ of the constructors of E_{lin}.
Let $\mathcal{S} = \mathcal{S}_{\mathrm{conv}}$.

The following theorem says that these three algorithms are correct. It is proved in Appendix A.

Theorem 2 *If Algorithm 1, 2, or 3 produces a signature Σ' from a signature Σ, then Σ' models Σ.*

5.4 Reductions with equations and rewrite rules

From this point on, we assume that Σ' models Σ. We extend equality modulo Σ from terms to biprocesses and term evaluations: $\Sigma \vdash P = P'$ if and only if P' can be obtained from P by replacing some of its subterms M (not containing diff or eval) with subterms equal modulo Σ. We define $\Sigma \vdash D = D'$ similarly. Finally, we define $P \rightarrow_{\Sigma',\Sigma} P'$ as $P \rightarrow_{\Sigma} P'$ except that signature Σ' is used for reduction rules (Red I/O) and (Red Fun 1)—signature Σ is still used for (Red Fun 2).

We say that a biprocess P_0 is unevaluated when every term in P_0 is either a variable or $\mathsf{diff}[a, a]$ for some name a. Hence, every function symbol in P_0 must be in a term evaluation and prefixed by eval. For any biprocess P, we can build an unevaluated biprocess $\mathrm{unevaluated}(P)$ by introducing a term evaluation for every non-trivial term and a diff for every name (with $P \approx \mathrm{unevaluated}(P)$). For instance, the unevaluated biprocess built from the process of Example 3 is:

$$\begin{array}{l}(\nu s)(let\ y = \mathrm{eval}\ pk(\mathsf{diff}[s, s])\ in\ \overline{\mathsf{diff}[c, c]}\langle y\rangle \mid \\ \quad !\mathsf{diff}[c', c'](x).(\nu a) \\ \quad\ let\ z = \mathsf{diff}[\mathrm{eval}\ enc(x, \mathrm{eval}\ pk(\mathsf{diff}[s, s]), \mathsf{diff}[a, a]), \mathsf{diff}[a, a]]\ in\ \overline{\mathsf{diff}[c, c]}\langle z\rangle)\end{array}$$

Lemma 1 *Let P_0 be a closed, unevaluated biprocess. If $P_0 \rightarrow^*_{\Sigma} \equiv P'_0$, $\Sigma \vdash P'_0 = P'$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(\{P'\})$, then $P_0 \rightarrow^*_{\Sigma',\Sigma} \equiv P'$. Conversely, if $P_0 \rightarrow^*_{\Sigma',\Sigma} \equiv P'$ then there exists P'_0 such that $\Sigma \vdash P'_0 = P'$ and $P_0 \rightarrow^*_{\Sigma} \equiv P'_0$.*

This lemma gives an operational correspondence between $\rightarrow_{\Sigma}$ and $\rightarrow_{\Sigma',\Sigma}$. A similar lemma holds for processes instead of biprocesses, and can be used for extending previous proof techniques for secrecy [3] and correspondence [16] properties, so that they apply under equational theories. These extensions are implemented in ProVerif. We do not detail them further since we focus on equivalences in this paper. Using Lemma 1, we obtain:

Lemma 2 *A closed biprocess P_0 satisfies the conditions of Corollary 1 if and only if, for all plain evaluation contexts C, all evaluation contexts C', and all reductions $\mathrm{unevaluated}(C[P_0]) \rightarrow^*_{\Sigma',\Sigma} P$, we have*

1. *if $P \equiv C'[\overline{N}\langle M\rangle.Q \mid N'(x).R]$ and $\mathsf{fst}(N) = \mathsf{fst}(N')$, then $\Sigma \vdash \mathsf{snd}(N) = \mathsf{snd}(N')$,*
2. *if $P \equiv C'[let\ x = D\ in\ Q\ else\ R]$ and $\mathsf{fst}(D) \Downarrow_{\Sigma'} M_1$ for some M_1, then $\mathsf{snd}(D) \Downarrow_{\Sigma} M_2$ for some M_2,*

as well as the symmetric properties where we swap fst *and* snd.

The lemmas above are proved in Appendix B.

6 Clause generation

Given a closed biprocess P_0, our protocol verifier builds a set of Horn clauses. This section explains the generation of the clauses, substantially extending to biprocesses previous work at the level of processes.

6.1 Patterns and facts

In the clauses, the terms of processes are represented by patterns, with the following grammar:

$p ::=$	patterns
x, y, z, i	variable
$f(p_1, \ldots, p_n)$	constructor application
$a[p_1, \ldots, p_n]$	name
g	element of $GVar$

We assign a distinct, fresh session identifier variable i to each replication of P_0. (We will use a distinct value for i for each copy of the replicated process.) We assign a pattern $a[p_1, \ldots, p_n]$ to each name a of P_0. We treat a as a function symbol, and write $a[p_1, \ldots, p_n]$ rather than $a(p_1, \ldots, p_n)$ only for clarity. We sometimes write a for $a[\,]$. If a is a free name, then its pattern is $a[\,]$. If a is bound by a restriction (νa) in P_0, then its pattern takes as arguments the terms received as inputs, the results of term evaluations, and the session identifiers of replications in the context that encloses the restriction. For example, in the process $!c'(x).(\nu a)P$, each name created by (νa) is represented by $a[i, x]$ where i is the session identifier for the replication and x is the message received as input in $c'(x)$. We assume that each restriction (νa) in P_0 has a different name a, distinct from any free name of P_0. Moreover, session identifiers enable us to distinguish names created in different copies of processes. Hence, each name created in the process calculus is represented by a different pattern in the verifier.

Patterns include an infinite set of constants $GVar$. These constants are basically universally quantified variables, and occur only in arguments of the predicate nounif, defined in Definition 5 below. We write $GVar(M)$ for the term obtained from M by replacing the variables of M with new constants in the set $GVar$.

Clauses are built from the following predicates:

$F ::=$	facts
$\mathrm{att}'(p, p')$	attacker knowledge
$\mathrm{msg}'(p_1, p_2, p_1', p_2')$	output message p_2 on p_1 (resp. p_2' on p_1')
$\mathrm{input}'(p, p')$	input on p (resp. p')
$\mathrm{nounif}(p, p')$	impossible unification
bad	bad

Informally, $\mathrm{att}'(p, p')$ means that the attacker may obtain p in $\mathsf{fst}(P)$ and p' in $\mathsf{snd}(P)$ by the same operations; $\mathrm{msg}'(p_1, p_2, p_1', p_2')$ means that message p_2 may appear on channel p_1 in $\mathsf{fst}(P)$ and that message p_2' may appear on channel p_1' in $\mathsf{snd}(P)$ after the same reductions; $\mathrm{input}'(p, p')$ means that an input may be executed on channel p

in $\mathsf{fst}(P)$ and on channel p' in $\mathsf{snd}(P)$, thus enabling the attacker to infer whether p (resp. p') is equal to another channel used for output; $\mathrm{nounif}(p, p')$ means that p and p' cannot be unified modulo Σ by substituting elements of $GVar$ with patterns; finally, bad serves in detecting violations of observational equivalence: when bad is not derivable, we have observational equivalence.

An evident difference with respect to previous translations from processes to clauses is that predicates have twice as many arguments: we use the binary predicate att' instead of the unary one att and the 4-ary predicate msg' instead of the binary one msg. This extension allows us to represent information for both variants of a biprocess.

The predicate nounif is not defined by clauses, but by special simplification steps in the solver, defined in Section 7.

Definition 5 Let p and p' be closed patterns. The fact $\mathrm{nounif}(p, p')$ holds if and only if there is no closed substitution σ with domain $GVar$ such that $\Sigma \vdash \sigma p = \sigma p'$.

6.2 Clauses for the attacker

The following clauses represent the capabilities of the attacker:

$$\text{For each } a \in fn(P_0),\ \mathrm{att}'(a[\,], a[\,]) \tag{Rinit}$$

$$\text{For some } b \text{ that does not occur in } P_0,\ \mathrm{att}'(b[x], b[x]) \tag{Rn}$$

$$\begin{array}{l}\text{For each function } h, \text{ for each pair of rewrite rules}\\ \quad h(M_1, \ldots, M_n) \to M \text{ and } h(M'_1, \ldots, M'_n) \to M'\\ \quad \text{in } \mathrm{def}_{\Sigma'}(h) \text{ (after renaming of variables),}\\ \quad \mathrm{att}'(M_1, M'_1) \wedge \ldots \wedge \mathrm{att}'(M_n, M'_n) \to \mathrm{att}'(M, M')\end{array} \tag{Rf}$$

$$\mathrm{msg}'(x, y, x', y') \wedge \mathrm{att}'(x, x') \to \mathrm{att}'(y, y') \tag{Rl}$$

$$\mathrm{att}'(x, x') \wedge \mathrm{att}'(y, y') \to \mathrm{msg}'(x, y, x', y') \tag{Rs}$$

$$\mathrm{att}'(x, x') \to \mathrm{input}'(x, x') \tag{Ri}$$

$$\mathrm{input}'(x, x') \wedge \mathrm{msg}'(x, z, y', z') \wedge \mathrm{nounif}(x', y') \to \mathrm{bad} \tag{Rcom}$$

$$\begin{array}{l}\text{For each destructor } g,\\ \text{for each rewrite rule } g(M_1, \ldots, M_n) \to M \text{ in } \mathrm{def}_{\Sigma'}(g),\\ \quad \bigwedge_{g(M'_1, \ldots, M'_n) \to M' \text{ in } \mathrm{def}_{\Sigma'}(g)} \mathrm{nounif}((x'_1, \ldots, x'_n), GVar((M'_1, \ldots, M'_n)))\\ \quad \wedge\, \mathrm{att}'(M_1, x'_1) \wedge \ldots \wedge \mathrm{att}'(M_n, x'_n) \to \mathrm{bad}\end{array} \tag{Rt}$$

plus symmetric clauses (Rcom′) and (Rt′) obtained from (Rcom) and (Rt) by swapping the first and second arguments of input' and att' and the first and third arguments of msg'.

Clause (Ri) means that, if the attacker has x (resp. x'), then it can attempt an input on x (resp. x'), thereby testing whether it is equal to some other channel used for output. Clauses (Rcom) and (Rcom′) detect when a communication can occur in one variant of the biprocess and not in the other: the input and output channels are equal in one

variant and different in the other. These clauses check that condition 1 of Lemma 2 and its symmetric are true.

Clause (Rt) checks that for all applications of a destructor g, if this application succeeds in $\mathsf{fst}(P)$, then it succeeds in $\mathsf{snd}(P)$, possibly using another rule. Clause (Rt′) checks the converse. These two clauses are essential for obtaining condition 2 of Lemma 2. Consider, for instance, the destructor equals of Section 2.2. After a minor simplification, Clauses (Rt) and (Rt′) become

$$\mathrm{att}'(x, y) \wedge \mathrm{att}'(x, y') \wedge \mathrm{nounif}(y, y') \rightarrow \mathrm{bad} \quad (1)$$

$$\mathrm{att}'(y, x) \wedge \mathrm{att}'(y', x) \wedge \mathrm{nounif}(y, y') \rightarrow \mathrm{bad} \quad (2)$$

The other clauses are adapted from previous work [3, 16] by replacing unary (resp. binary) predicates with binary (resp. 4-ary) ones. Clause (Rinit) indicates that the attacker initially has all free names of P_0. Clause (Rn) means that the attacker can generate fresh names $b[x]$. Clause (Rf) mean that the attacker can apply all functions to all terms it has. In this clause, the rewrite rules $h(M_1, \ldots, M_n) \rightarrow M$ and $h(M'_1, \ldots, M'_n) \rightarrow M'$ may be different elements of $\mathrm{def}_{\Sigma'}(h)$; their variables are renamed so that $M_1, \ldots, M_n, M$ on the one hand and $M'_1, \ldots, M'_n, M'$ on the other hand do not share variables. Clause (Rl) means that the attacker can listen on all the channels it has, and (Rs) that it can send all the messages it has on all the channels it has.

6.3 Clauses for the protocol

The translation $[\![P]\!]\rho ss'H$ of a biprocess P is a set of clauses, where ρ is an environment that associates a pair of patterns with each name and variable, s and s' are sequences of patterns, and H is a sequence of facts. The empty sequence is written $\emptyset$; the concatenation of a pattern p to the sequence s is written s, p; the concatenation of a fact F to the sequence H is written $H \wedge F$. When ρ associates a pair of patterns with each name and variable, and f is a constructor, we extend ρ as a substitution by $\rho(f(M_1, \ldots, M_n)) = (f(p_1, \ldots, p_n), f(p'_1, \ldots, p'_n))$ where $\rho(M_i) = (p_i, p'_i)$ for all $i \in \{1, \ldots, n\}$. We denote by $\rho(M)_1$ and $\rho(M)_2$ the components of the pair $\rho(M)$. We let $\rho(\mathsf{diff}[M, M']) = (\rho(M)_1, \rho(M')_2)$. We define $[\![P]\!]\rho ss'H$ as follows:

$[\![0]\!]\rho ss'H = \emptyset$

$[\![!P]\!]\rho ss'H = [\![P]\!]\rho(s, i)(s', i)H$

where i is a fresh variable

$[\![P \mid Q]\!]\rho ss'H = [\![P]\!]\rho ss'H \cup [\![Q]\!]\rho ss'H$

$[\![(\nu a)P]\!]\rho ss'H = [\![P]\!](\rho[a \mapsto (a[s], a[s'])])ss'H$

$[\![M(x).P]\!]\rho ss'H =$

$\quad [\![P]\!](\rho[x \mapsto (x', x'')])(s, x')(s', x'')(H \wedge \mathrm{msg}'(\rho(M)_1, x', \rho(M)_2, x''))$

$\cup \{H \rightarrow \mathrm{input}'(\rho(M)_1, \rho(M)_2)\}$

where x' and x'' are fresh variables

$[\![\overline{M}\langle N\rangle.P]\!]\rho ss'H = [\![P]\!]\rho ss'H \cup \{H \rightarrow \mathrm{msg}'(\rho(M)_1, \rho(N)_1, \rho(M)_2, \rho(N)_2)\}$

$$
\begin{aligned}
&[\![\mathit{let}\ x = D\ \mathit{in}\ P\ \mathit{else}\ Q]\!]\rho s s' H = \\
&\quad \bigcup \{ [\![P]\!]((\sigma\rho)[x \mapsto (p, p')])(\sigma s, p)(\sigma s', p')(\sigma H) \mid (\rho(D)_1, \rho(D)_2) \Downarrow' ((p, p'), \sigma) \} \\
&\quad \cup\ [\![Q]\!]\rho s s'(H \wedge \rho(\mathrm{fails}(\mathsf{fst}(D)))_1 \wedge \rho(\mathrm{fails}(\mathsf{snd}(D)))_2) \\
&\quad \cup \{ \sigma H \wedge \sigma\rho(\mathrm{fails}(\mathsf{snd}(D)))_2 \rightarrow \mathrm{bad} \mid \rho(D)_1 \Downarrow' (p, \sigma) \} \\
&\quad \cup \{ \sigma H \wedge \sigma\rho(\mathrm{fails}(\mathsf{fst}(D)))_1 \rightarrow \mathrm{bad} \mid \rho(D)_2 \Downarrow' (p', \sigma) \} \\
&\quad \text{where } \mathrm{fails}(D) = \textstyle\bigwedge_{\sigma \mid D \Downarrow' (p,\sigma)} \mathrm{nounif}(D, GVar(\sigma D))
\end{aligned}
$$

In the translation, the environment ρ maps names and variables to their corresponding pair of patterns—one pattern for each variant of the biprocess. The sequences s and s' contain all input messages, session identifiers, and results of term evaluations in the enclosing context—one sequence for each variant of the biprocess. They are used in the restriction case $(\nu a)P$, to build patterns $a[s]$ and $a[s']$ that correspond to the name a. The sequence H contains all facts that must be true to run the current process.

The clauses generated are similar to those of [16], but clauses are added to indicate which tests the adversary can perform, and predicates have twice as many arguments.

- Replication creates a new session identifier i, added to s and s'. Replication is otherwise ignored, since Horn clauses can be applied any number of times anyway.
- In the translation of an input, the sequence H is extended with the input in question and the environment ρ with a binding of x to a new variable x' in variant 1, x'' in variant 2. Moreover, a new clause $H \rightarrow \mathrm{input}'(\rho(M)_1, \rho(M)_2)$ is added, indicating that when all conditions in H are true, an input on channel M may be executed. This input may enable the adversary to infer that M is equal to some channel used for output; Clauses (Rcom) or (Rcom′) derive bad when this information may break equivalence.
- The output case adds a clause stating that message N may be sent on channel M.
- Finally, the clauses for a term evaluation are the union of clauses for the cases where the term evaluation succeeds on both sides (then we execute P), where the term evaluation fails on both sides (then we execute Q), and where the term evaluation fails on one side and succeeds on the other (then we derive bad). Indeed, in the last case, the adversary may get to know whether the term evaluation succeeds or fails (when the code executed in the success case is visibly different from the code executed in the failure case).

Example 7 The biprocess of Example 3 yields the clauses:

$$
\begin{aligned}
&\mathrm{msg}'(c, pk(s), c, pk(s)) \\
&\mathrm{msg}'(c', x, c', x') \rightarrow \mathrm{msg}'(c, penc(x, pk(s), a[i, x]), c, a[i, x'])
\end{aligned}
$$

The first clause corresponds to the output of the public key $pk(s)$. The second clause corresponds to the other output: if a message x (resp. x') is received on channel c', then the message $penc(x, pk(s), a[i, x])$ in the first variant (resp. $a[i, x']$ in the second

variant) is sent on channel c. The encoding of the fresh name a as a pattern $a[i, x]$ is explained in Section 6.1. □

Example 8 The process $c(x).let\ y = \text{eval}\ dec(x, a)\ in\ \overline{c}\langle y\rangle$, where dec is a destructor defined by $dec(enc(x, y), y) \to x$, yields the clauses:

$$\begin{aligned}
&\text{msg}'(c, enc(y, a), c, x') \wedge \text{nounif}(x', enc(\mathsf{g}, a)) \to \text{bad}\\
&\text{msg}'(c, x, c, enc(y', a)) \wedge \text{nounif}(x, enc(\mathsf{g}, a)) \to \text{bad}\\
&\text{msg}'(c, enc(y, a), c, enc(y', a)) \to \text{msg}'(c, y, c, y')
\end{aligned}$$

In the first clause, a message received on c is of the form $enc(y, a)$ in the first variant but not in the second variant; decryption succeeds only in the first variant, so the process is not uniform and we derive bad. The second clause is the symmetric case. In the third clause, decryption succeeds in both variants, and yields an output on channel c. □

6.4 Proving equivalences

Let $\rho_0 = \{a \mapsto (a[\,], a[\,]) \mid a \in fn(P_0)\}$. We define the set of clauses that corresponds to biprocess P_0 as:

$$\mathcal{R}_{P_0} = [\![\text{unevaluated}(P_0)]\!]\rho_0\emptyset\emptyset\emptyset \cup \{(\text{Rinit}), (\text{Rn}), \ldots, (\text{Rt}), (\text{Rt}')\}$$

The following result is proved in Appendix D. It shows the soundness of the translation.

Theorem 3 *If* bad *is not a logical consequence of $\mathcal{R}_{P_0}$, then P_0 satisfies observational equivalence.*

When bad is a logical consequence of $\mathcal{R}_{P_0}$, the derivation of bad from $\mathcal{R}_{P_0}$ can serve for reconstructing a violation of the hypothesis of Corollary 1, via an extension of recent techniques for secrecy analysis [10]. However, the translation of protocols to Horn clauses performs safe abstractions that sometimes result in false counterexamples: the Horn clauses can be applied any number of times, so the translation ignores the number of repetitions of actions. For instance, $(\nu c)(\overline{c}\langle M\rangle \mid c(x).c(x).P)$ satisfies equivalence for any P because P is never executed, and $(\nu c)(\overline{c}\langle \mathsf{diff}[M_1, M_2]\rangle \mid c(x).\overline{d}\langle c\rangle)$ satisfies equivalence for any M_1 and M_2 because its diff disappears before the attacker obtains channel c. Our technique cannot prove these equivalences in general. The latter example illustrates that our technique typically fails for biprocesses that first keep some value secret and later reveal it. The reason for the failures on $(\nu c)(\overline{c}\langle M\rangle \mid c(x).c(x).P)$ and $(\nu c)(\overline{c}\langle \mathsf{diff}[M_1, M_2]\rangle \mid c(x).\overline{d}\langle c\rangle)$ is that the translation to classical Horn clauses basically treats these two biprocesses like variants with additional replications, namely $(\nu c)(!\overline{c}\langle M\rangle \mid c(x).c(x).P)$ and $(\nu c)(!\overline{c}\langle \mathsf{diff}[M_1, M_2]\rangle \mid\ !c(x).\overline{d}\langle c\rangle)$ respectively, and these variants do not necessarily satisfy equivalence. On the other hand, the safe abstractions that the translation performs are crucial for the applicability of our technique to infinite-state systems, which is illustrated by many of the examples in this paper.

We also have the following lemma, which is important for proving the soundness of some simplification steps in the solving algorithm below, enabling us to work with terms in normal form only. It is proved in Appendix C.

Lemma 3 *If* bad *is derivable from* $\mathcal{R}_{P_0}$ *then* bad *is derivable from* $\mathcal{R}_{P_0}$ *by a derivation such that* $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{F})$ *where* $\mathcal{F}$ *is the set of intermediately derived facts in this derivation, excluding* nounif *facts.*

7 Solving algorithm

In order to determine whether bad is a logical consequence of $\mathcal{R}_{P_0}$, we use an algorithm based on resolution with free selection, adapting a previous algorithm [17].

7.1 The basic resolution algorithm

The algorithm infers new clauses by resolution, as follows. From two clauses $R = H \to C$ and $R' = F \wedge H' \to C'$ (where F is any hypothesis of R') such that C and F are unifiable, with most general unifier σ, it infers $R \circ_F R' = \sigma H \wedge \sigma H' \to \sigma C'$:

$$\frac{H \to C \qquad F \wedge H' \to C'}{\sigma H \wedge \sigma H' \to \sigma C'}$$

The clause $R \circ_F R'$ is the combination of R and R', in which R proves the hypothesis F of R'. Resolution is guided by a selection function: $\mathsf{sel}(R)$ returns a subset of the hypotheses of R, and the resolution step above applies only when $\mathsf{sel}(R) = \emptyset$ and $F \in \mathsf{sel}(R')$. When $\mathsf{sel}(R) = \emptyset$, we say that the conclusion of R is selected. In this paper, we use the following selection rules:

- $\mathrm{nounif}(p_1, p_2)$ is never selected. (It is handled by special simplification steps.)
- bad is never selected, except in the clause bad, and in clauses whose hypotheses are all of the form $\mathrm{nounif}(p_1, p_2)$. (If we select bad in a clause $H \to \mathrm{bad}$, then the algorithm will fail to prove that bad is not derivable. That is why we avoid selecting bad when possible.)
- $\mathrm{att}'(x, x')$ with any variables x, x' is selected only when no other fact can be selected. (Our intent is to obtain termination, whereas facts $\mathrm{att}'(x, x')$ can be unified with all facts $\mathrm{att}'(p, p')$ to generate many additional clauses.) In this case, $\mathrm{att}'(x, x')$ is selected preferably when x (or x') occurs in a fact $\mathrm{nounif}(x, p')$ where p' is not a variable. (When we select $\mathrm{att}'(x, x')$, this fact will be unified with some other fact, thus hopefully instantiating x, so that we make progress determining whether $\mathrm{nounif}(x, p')$ is true or not.)

7.2 General simplifications

As part of the algorithm, we apply a series of simplification functions on clauses. Some of them are standard, such as the elimination of tautologies (performed by $elimtaut$) and duplicate hypotheses (performed by $elimdup$). We omit their definitions. Others are specific to our purpose:

- Elimination of $\mathrm{att}'(x, y)$: *elimattx* removes hypotheses $\mathrm{att}'(x, y)$ when x and y do not appear elsewhere in the clause, except possibly in nounif facts. The variables x and y may be the same variable.

- Elimination of useless variables: *elimvar* transforms clauses of the form

$$R = \mathrm{att}'(x, y) \wedge \mathrm{att}'(x, y') \wedge H \rightarrow C$$

  into $R\{y/y'\}$, when R is not Clause (1).

  The soundness of *elimvar* can be established by cases. If we can derive facts $\mathrm{att}'(p_x, p_y)$ and $\mathrm{att}'(p_x, p_{y'})$ such that $\Sigma \vdash p_y \neq p_{y'}$ from the other clauses, then we can derive bad by applying Clause (1), included in the clause base as Clause (Rt) for $g = \mathrm{equals}$. Otherwise, in any derivation of bad obtained by Lemma 3, any application of R uses the same fact to match both $\mathrm{att}'(x, y)$ and $\mathrm{att}'(x, y')$, and the transformed clause also applies. (The clause R uses $\mathrm{att}'(p_x, p_y)$ and $\mathrm{att}'(p_x, p_{y'})$ with $\Sigma \vdash p_y = p_{y'}$, hence $p_y = p_{y'}$ by the conclusion of Lemma 3.)

  The function *elimvar* also performs the symmetric simplification, relying on the presence of Clause (2).

- Elimination of useless forms modulo equality: *simpeq* removes clauses that contain a fact F that is not a nounif fact and is not in normal form relatively to $\mathcal{S}$. The soundness of this simplification follows from Lemma 3. A typical example concerns decryption, when it is defined by an equation (as in Example 5): we can remove any clause that contains $dec(enc(y, x), x)$.

  This simplification could be extended to clauses that contain several syntactically different forms of the same term modulo the equational theory, although that would be more difficult to implement.

7.3 Simplifications for nounif

These simplifications are adapted from those for $\mathrm{testunif}$ (from [17]).

- Unification: *unify* transforms clauses of the form $H \wedge \mathrm{nounif}(p_1, p_2) \rightarrow C$ as follows. For every $\mathrm{nounif}(p_1, p_2)$ hypothesis in turn, it tries to unify p_1 and p_2 modulo the equational theory, considering elements of $GVar$ as variables. If this unification fails, then the clause becomes $H \rightarrow C$, because $\mathrm{nounif}(p_1, p_2)$ holds when $\Sigma \vdash \sigma p_1 \neq \sigma p_2$ for all σ. Otherwise, *unify* replaces the clause with

$$H \wedge \bigwedge_{j=1}^{n} \mathrm{nounif}((x_1^j, \ldots, x_{k_j}^j), (\sigma_j x_1^j, \ldots, \sigma_j x_{k_j}^j)) \rightarrow C$$

  where $\sigma_1, \ldots, \sigma_n$ are the most general unifiers of p_1 and p_2 modulo the equational theory and $x_1^j, \ldots, x_{k_j}^j$ are all variables affected by σ_j. (These may include elements of $GVar$.) In this unification, σ_j is built so that all variables in its domain and its image are variables of p_1 and p_2, and the variables in its domain

do not occur in its image. Note that an instance of $\bigwedge_{j=1}^{n} \mathrm{nounif}((x_1^j, \ldots, x_{k_j}^j), (\sigma_j x_1^j, \ldots, \sigma_j x_{k_j}^j))$ is true if and only if the same instance of $\mathrm{nounif}(p_1, p_2)$ is, because $\sigma p_1 = \sigma p_2$ if and only if there exists $j \in \{1, \ldots, n\}$ such that $\sigma(x_1^j, \ldots, x_{k_j}^j) = \sigma\sigma_j(x_1^j, \ldots, x_{k_j}^j)$, for all σ with domain $GVar \cup Var$ where Var is the set of variables.

In order to compute unification modulo the equational theory of p_1 and p_2, we rewrite both terms according to the rewrite rules for the function symbols that they contain (generating some bindings for variables), then syntactically unify the results. Formally, the most general unifiers of p_1 and p_2 modulo Σ are the substitutions $\sigma_u\sigma$ such that $\mathrm{addeval}(p_1, p_2) \Downarrow' ((p_1', p_2'), \sigma)$ and σ_u is the most general unifier of p_1' and p_2'.

For instance, with an empty equational theory, *unify* transforms the clause

$$H \wedge \mathrm{nounif}((enc(x', y'), z'), (enc(\mathsf{g}, y), \mathsf{g})) \rightarrow C$$

into

$$H \wedge \mathrm{nounif}((x', y', z'), (\mathsf{g}, y, \mathsf{g})) \rightarrow C \tag{3}$$

Assuming the equational theory of Example 6, *unify* transforms the clause

$$H \wedge \mathrm{nounif}(x \mathsf{\hat{}} y, x' \mathsf{\hat{}} y') \rightarrow C$$

into

$$H \wedge \mathrm{nounif}((x, y), (x', y')) \wedge \mathrm{nounif}((x, x'), (\mathsf{b} \mathsf{\hat{}} y', \mathsf{b} \mathsf{\hat{}} y)) \rightarrow C$$

- Swap: *swap* transforms facts $\mathrm{nounif}((p_1, \ldots, p_n), (p_1', \ldots, p_n'))$ in clauses obtained after *unify*. When p_i is a variable and $p_i' \in GVar$, it swaps p_i and p_i' everywhere in the nounif fact. Note that an instance of the new nounif fact is true if and only if the same instance of the old one is, since the unification constraints remain the same.

  For instance, *swap* transforms Clause (3) into

  $$H \wedge \mathrm{nounif}((\mathsf{g}, y', z'), (x', y, x')) \rightarrow C \tag{4}$$

- Elimination of elements of $GVar$: $elimGVar$ transforms facts $\mathrm{nounif}((p_1, \ldots, p_n), (p_1', \ldots, p_n'))$ in clauses obtained after *unify* and *swap*: when $p_i = \mathsf{g} \in GVar$, it eliminates the pair p_i, p_i' from the nounif fact.

  An instance of the new nounif fact is true if and only if the same instance of the old one is, because $\mathsf{g} \in GVar$ cannot occur elsewhere in the nounif fact. (This property comes from the result of *unify* and is preserved by *swap*.)

  For instance, $elimGVar$ transforms Clause (4) into

  $$H \wedge \mathrm{nounif}((y', z'), (y, x')) \rightarrow C$$

- Detection of failed nounif: *elimnouniffalse* removes clauses that contain the hypothesis $\mathrm{nounif}((), ())$.

7.4 Combining the simplifications

We group all simplifications, as follows:

- We define the simplification function $simplify = elimtaut \circ elimattx \circ elimdup \circ$ $\circ elimnouniffalse \circ repeat(elimGVar \circ swap \circ unify \circ elimvar \circ simpeq)$. The expression $repeat(f)$ means that the application of function f is repeated until a fixpoint is obtained, that is, $f(\mathcal{R}) = \mathcal{R}$. It is enough to repeat the simplification only when $elimvar$ has modified the set of clauses. Indeed, no new simplification would be done in the other cases. The repetition never leads to an infinite loop, because the number of variables decreases at each iteration.

- We let $condense(\mathcal{R})$ apply $simplify$ to $\mathcal{R}$ and then eliminate subsumed clauses. We say that $H_1 \to C_1$ subsumes $H_2 \to C_2$ (and we write $(H_1 \to C_1) \sqsupseteq (H_2 \to C_2)$) if and only if there exists a substitution σ such that $\sigma C_1 = C_2$ and $\sigma H_1 \subseteq H_2$ (as a multiset inclusion). If $\mathcal{R}$ contains clauses R and R' such that R subsumes R', then R' is removed. (In that case, R can do all derivations that R' can do.)

Finally, we define the algorithm $\mathsf{saturate}(\mathcal{R}_0)$. Starting from $condense(\mathcal{R}_0)$, the algorithm adds clauses inferred by resolution with the selection function sel and condenses the resulting clause set until a fixpoint is reached. When a fixpoint is reached, $\mathsf{saturate}(\mathcal{R}_0)$ is the set of clauses R in the clause set such that $\mathsf{sel}(R) = \emptyset$.

We have the following soundness result:

Theorem 4 *If* $\mathsf{saturate}(\mathcal{R}_{P_0})$ *terminates and its result contains no clause of the form* $H \to \mathrm{bad}$*, then* bad *is not derivable from* $\mathcal{R}_{P_0}$.

This result is proved in Appendix E.

8 Extension to scenarios with several stages

Many protocols can be broken into stages, and their security properties can be formulated in terms of these stages. Typically, for instance, if a protocol discloses a session key after the conclusion of a session, then the secrecy of the data exchanged during the session may be compromised but not its authenticity. In some cases, the disclosure of keys and other keying material is harmless and even useful at certain points in protocol executions (e.g., [2]). In this section we extend our technique to protocols with several successive stages. This extension consists in the following changes:

- The syntax of processes is supplemented with a stage prefix, $t : P$, where t is a nonnegative integer. Intuitively, t represents a global clock, and the process $t : P$ is active only during stage t.

- The semantics of processes (and biprocesses) is extended by adding the rules of Figure 4 to those of Figures 2 and 3. This new semantics is a refinement, since $P \to Q$ in the simple semantics if and only if $t : P \to_t t : Q$ in the refined semantics. Conversely, if $P' \to_t Q'$ for staged processes, then $P \to Q$ in the

$$(\nu a)t : P \equiv t : (\nu a)P$$
$$t : (P \mid Q) \equiv t : P \mid t : Q$$
$$t : t' : P \equiv t' : P \quad \text{if } t < t'$$

$$P \rightarrow Q \Rightarrow t : P \rightarrow_t t : Q \qquad \text{(Red Stage)}$$
$$P \rightarrow_t Q \Rightarrow P \mid R \rightarrow_t Q \mid R \qquad \text{(Red Par)}$$
$$P \rightarrow_t Q \Rightarrow (\nu a)P \rightarrow_t (\nu a)Q \qquad \text{(Red Res)}$$
$$P' \equiv P,\ P \rightarrow_t Q,\ Q \equiv Q' \Rightarrow P' \rightarrow_t Q' \qquad \text{(Red } \equiv\text{)}$$

Figure 4: Semantics for stages

simple semantics, where P and Q are obtained from P' and Q' by erasing all stage prefixes.

- Instead of att', msg', and input', the clause generation uses distinct predicates att'_t, msg'_t, and input'_t for each stage t used in the protocol. The clauses for the protocol use the predicates indexed by t when translating the process P in $t : P$. The clauses for the attacker are replicated for each att'_t. In addition, new clauses carry over the attacker's knowledge from one stage to the next:

  $$\mathrm{att}'_t(x, x') \rightarrow \mathrm{att}'_{t+1}(x, x') \qquad \text{(Rp)}$$

  As an optimization, when the protocol uses only plain processes for the initial stages $t \leq i$ (that is, diff occurs only at later stages), we translate these processes using the more efficient clause generation of [3], with predicates that keep track of a single process, rather than the two variants of a biprocess.

- Our main theorems hold for staged biprocesses, with minor adaptations and extra optimizations in algorithms. In particular, all definitions and theorems now use $\rightarrow_{_} = \cup_{t \geq 0} \rightarrow_t$ instead of $\rightarrow$.

9 Applications

This section surveys some of the applications of our proof method. The total runtime of all proof scripts for the experiments described below is 45 s on a Pentium M 1.8 GHz. None of these applications could be handled by ProVerif without the extensions presented in this paper.

9.1 Weak secrets

A *weak secret* represents a secret value with low entropy, such as a human-memorizable password. Protocols that rely on weak secrets are often subject to guessing attacks, whereby an attacker guesses a weak secret, perhaps using a dictionary, and verifies its guess. The guess verification may rely on interaction with protocol participants or on computations on intercepted messages (e.g., [13, 35, 36]). With some care in protocol design, however, those attacks can be prevented:

- On-line guessing attacks can be mitigated by limiting the number of retries that participants allow. An attacker that repeatedly attempts to guess the weak secret should be eventually detected and stopped if it tries to verify its guesses by interacting with other participants.

- Off-line guessing attacks can be prevented by making sure that, even if the attacker (systematically) guesses the weak secret, it cannot verify whether its guess is correct by computing on intercepted traffic.

Off-line guessing attacks can be explained and modelled in terms of a 2-stage scenario. In stage 0, on-line attacks are possible, but the weak secret is otherwise unguessable. In stage 1, the attacker obtains a possible value for the weak secret (intuitively, by guessing it). The absence of off-line attacks is characterized by an equivalence: the attacker cannot distinguish the weak secret used in stage 0 from an unrelated fresh value.

In our calculus, we arrive at the following definition:

Definition 6 (Weak secrecy) Let P be a closed process with no stage prefix. We say that P prevents off-line attacks against w when $(\nu w)(0 : P \mid 1 : (\nu w')\overline{c}\langle \mathsf{diff}[w, w']\rangle)$ satisfies observational equivalence.

This definition is in line with the work of Cohen, Corin et al., Delaune and Jacquemard, Drielsma et al., and Lowe [26–28, 30, 32, 41]. Lowe uses the model-checker FDR to handle a bounded number of sessions, while Delaune and Jacquemard give a decision procedure in this case. Corin et al. give a definition based on equivalence like ours, but do not consider the first, active stage; they analyze only one session.

As a first example, assume that a principal attempts to prove knowledge of a shared password w to a trusted server by sending a hash of this password encrypted under the server's public key. (For simplicity, the protocol does not aim to provide freshness guarantees, so anyone may replay this proof.) Omitting the code for the server, a first protocol may be written:

$$P = (\nu s)\overline{c}\langle pk(s)\rangle.\overline{c}\langle penc(h(w), pk(s))\rangle$$

The first output reveals the public key of the server; the second output communicates the proof of knowledge of w. This protocol does not prevent off-line attacks against w. ProVerif finds an attack that corresponds to the following adversary:

$$\begin{aligned} A = 0 &: c(pk).c(e). \\ 1 &: c(w).\mathit{if}\ e = penc(h(w), pk)\ \mathit{then}\ \overline{\mathrm{Guessed}}\langle\rangle \end{aligned}$$

A corrected protocol uses non-deterministic encryption (see Example 3):

$$P = (\nu s, a)\overline{c}\langle pk(s)\rangle.\overline{c}\langle penc(h(w), pk(s), a)\rangle$$

ProVerif automatically produces a proof for this corrected protocol.

As a second example, we consider a simplified version of EKE [13]:

$$\begin{array}{rcl} P_A & = & (\nu d_A)\overline{c}\langle enc(\mathtt{b}\hat{}d_A, w)\rangle \\ P_B & = & c(x).(\nu d_B) let\ k = dec(x, w)\hat{}d_B\ in\ \overline{c}\langle enc(\mathtt{b}\hat{}d_B, w), k\rangle \\ P & = & !P_A \mid !P_B \end{array}$$

Here, two parties obtain a shared session key $k = (\mathtt{b}\hat{}d_A)\hat{}d_B$ via a Diffie-Hellman exchange, in which $\mathtt{b}\hat{}d_A$ and $\mathtt{b}\hat{}d_B$ are exchanged protected by a weak secret w. The EKE protocol has several rounds of key confirmation; here, instead, we immediately give the session key k to the attacker. Still, relying on the contextual property of equivalence, we can define a context that performs these key confirmations. Since that context does not use the weak secret, the resulting protocol prevents off-line attacks against w as long as the original protocol does.

We have proved security properties of several versions of EKE: the public-key and the Diffie-Hellman versions for EKE [13], and the version with hashed passwords and the one with signatures for Augmented EKE [14]. Unlike the protocol displayed above, our models include an unbounded number of possibly dishonest principals that run parallel sessions.

For the analysis of such protocols, we define encryption under a weak secret by the equational theory of Example 5. The use of this equational theory is important, as it entails that the adversary cannot check whether a decryption is successful and thereby check a guess. In contrast, a straightforward presentation with constructors and destructors but without the equational theory (see Section 2.1) would not be adequate in this respect: with that presentation, an attacker could verify a guess w' of w by testing whether the decryption of the first message of the protocol with w' succeeds.

9.2 Authenticity

Abadi and Gordon [8] use equivalences for expressing authenticity properties, and treat a variant of the Wide-Mouth-Frog protocol as an example. In this protocol, two participants A and B share secret keys k_{AS} and k_{SB} with a server S, respectively. Participant A generates a key k_{AB}, sends it encrypted to S, which forwards it reencrypted to B. Then A sends the payload x to B encrypted under k_{AB}. Finally, B forwards the payload that it receives, possibly for further processing. Essentially, authenticity is defined as an equivalence between the protocol and a specification. The specification is an idealized variant of the protocol, obtained by modifying B so that, independently of what it receives, it forwards A's payload x.

For the one-session version [8, Section 3.2.2], the protocol and the specification can be combined into the following biprocess P_0:

$$\begin{array}{l} P_A = (\nu k_{AB})\overline{c}\langle enc(k_{AB}, k_{AS})\rangle.\overline{c}\langle enc(x, k_{AB})\rangle \\ P_S = c(y_1).let\ y_2 = dec(y_1, k_{AS})\ in\ \overline{c}\langle enc(y_2, k_{SB})\rangle \\ P_B = c(y_3).let\ y_4 = dec(y_3, k_{SB})\ in\ c(y_5).let\ x' = dec(y_5, y_4)\ in\ \overline{e}\langle \mathsf{diff}[x, x']\rangle \\ P_0 = c(x).(\nu k_{AS})(\nu k_{SB})(P_A \mid P_S \mid P_B) \end{array}$$

with the rewrite rule $dec(enc(x, y), y) \to x$ for the destructor dec.

The technique presented in this paper automatically proves that P_0 satisfies observational equivalence, and hence establishes the desired authenticity property. Thus, it eliminates the need for a laborious manual proof. The technique can also be used for simplifying the proof of authenticity for the multi-session version.

Authenticity properties are sometimes formulated as correspondence assertions on behaviors, rather than as equivalences. Previous work shows how to check those assertions with ProVerif [16]. However, that previous work does not apply to equivalences.

9.3 Complete sessions in JFK

Finally, we show other ways in which automated proofs of equivalences can contribute to protocol analyses, specifically studying JFK, a modern session-establishment protocol for IP security [9].

In recent work [4], we modelled JFK in the applied pi calculus. We used processes for representing the reachable states of JFK, for any number of principals and sessions, and stated security properties as equivalences. Although we relied on ProVerif for reasoning about behaviors, our main proofs of equivalences were manual. Applying the techniques of this paper, we can revise and largely automate those proofs. The resulting proofs rely on equivalences on biprocesses, verified by ProVerif, composed with standard pi calculus equivalences that do not depend on the signature for terms.

In particular, a core property of JFK is that, once a session completes, its session key is (apparently) unrelated to the cryptographic material exchanged during the session, and all those values can be replaced by distinct fresh names [4, Theorem 2]. This property can be stated and proved in terms of a biprocess S that outputs either the actual results of JFK computations (in $\mathsf{fst}(S)$) or distinct fresh names (in $\mathsf{snd}(S)$), in parallel with the rest of the JFK system to account for any other sessions. The proof of this property goes as follows. The JFK system is split into $S \approx C[S']$, where S' is similar to S but omits unimportant parts of JFK, collected in the evaluation context $C[_]$. The proof that $S \approx C[S']$ is straightforward; it relies on pi calculus equivalences that eliminate communications on private channels introduced in the split. ProVerif shows that S' satisfies equivalence. Using the contextual property of equivalence, $C[S']$ satisfies equivalence, hence $\mathsf{fst}(S) \approx \mathsf{snd}(S)$.

10 Conclusion

In the last decade, there has been substantial research on proof methods for security protocols. While many of those proof methods have focused on predicates on behaviors, others have addressed equivalences between systems (e.g., [1, 6–8, 23–25, 29, 33, 34, 38, 39, 46]). Much of this research is concerned with obtaining sound and complete proof systems, often via sophisticated bisimulations, and eventually decision algorithms for restricted cases. In our opinion, these are important goals, and the results to date are significant.

In the present paper, we aim to contribute to this body of research with a different approach. We do not emphasize the development of bisimulation techniques. Rather,

we leverage behavior-oriented techniques and tools (ProVerif, in particular) for equivalence proofs. We show how to derive equivalences by reasoning about behaviors—specifically, by reasoning about behaviors of applied pi calculus biprocesses. We also show how to translate those biprocesses to Horn clauses and how to reason about their behaviors by resolution. The resulting proof method is sound, although that is not simple to establish. We demonstrate the usefulness of the method through automated analyses of interesting, infinite-state systems.

Acknowledgments Bruno Blanchet's work was partly done at Max-Planck-Institut für Informatik, Saarbrücken. Martín Abadi's work was partly supported by the National Science Foundation under Grants CCR-0204162, CCR-0208800, and CCF-0524078. We are grateful to Harald Ganzinger for helpful discussions on the treatment of equational theories.

References

[1] M. Abadi. Secrecy by typing in security protocols. *Journal of the ACM*, 46(5):749–786, Sept. 1999.

[2] M. Abadi, A. Birrell, M. Burrows, F. Dabek, and T. Wobber. Bankable postage for network services. In V. Saraswat, editor, *Advances in Computing Science – ASIAN 2003, Programming Languages and Distributed Computation, 8th Asian Computing Science Conference*, volume 2896 of *Lecture Notes on Computer Science*, pages 72–90, Mumbai, India, Dec. 2003. Springer.

[3] M. Abadi and B. Blanchet. Analyzing security protocols with secrecy types and logic programs. *Journal of the ACM*, 52(1):102–146, Jan. 2005.

[4] M. Abadi, B. Blanchet, and C. Fournet. Just fast keying in the pi calculus. *ACM Transactions on Information and System Security (TISSEC)*. To appear. An extended abstract appears in Programming Languages and Systems, 13th European Symposium on Programming (ESOP 2004).

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

[6] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *28th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115, London, United Kingdom, Jan. 2001. ACM Press.

[7] M. Abadi and A. D. Gordon. A bisimulation method for cryptographic protocols. *Nordic Journal of Computing*, 5(4):267–303, Winter 1998.

[8] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, Jan. 1999. An extended version appeared as Digital Equipment Corporation Systems Research Center report No. 149, January 1998.

[9] W. Aiello, S. M. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, K. Keromytis, and O. Reingold. Just Fast Keying: Key agreement in a hostile Internet. *ACM Transactions on Information and System Security*, 7(2):242–273, May 2004.

[10] X. Allamigeon and B. Blanchet. Reconstruction of attacks against cryptographic protocols. In *18th IEEE Computer Security Foundations Workshop (CSFW-18)*, pages 140–154, Aix-en-Provence, France, June 2005. IEEE.

[11] F. Baader and C. Tinelli. Deciding the word problem in the union of equational theories. Technical Report UIUCDCS-R-98-2073, UILU-ENG-98-1724, University of Illinois at Urbana-Champaign, Oct. 1998.

[12] M. Baudet. *Sécurité des protocoles cryptographiques: aspects logiques et calculatoires*. PhD thesis, Ecole Normale Supérieure de Cachan, 2007.

[13] S. M. Bellovin and M. Merritt. Encrypted Key Exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 72–84, May 1992.

[14] S. M. Bellovin and M. Merritt. Augmented Encrypted Key Exchange: a password-based protocol secure against dictionary attacks and password file compromise. In *Proceedings of the First ACM Conference on Computer and Communications Security*, pages 244–250, Nov. 1993.

[15] B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 82–96, Cape Breton, Nova Scotia, Canada, June 2001. IEEE Computer Society.

[16] B. Blanchet. From secrecy to authenticity in security protocols. In M. Hermenegildo and G. Puebla, editors, *9th International Static Analysis Symposium (SAS'02)*, volume 2477 of *Lecture Notes on Computer Science*, pages 342–359, Madrid, Spain, Sept. 2002. Springer.

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

[18] B. Blanchet. Automatic proof of strong secrecy for security protocols. Technical Report MPI-I-2004-NWG1-001, Max-Planck-Institut für Informatik, Saarbrücken, Germany, July 2004.

[19] B. Blanchet. Security protocols: From linear to classical logic by abstract interpretation. *Information Processing Letters*, 95(5):473–479, Sept. 2005.

[20] B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005)*, pages 331–340, Chicago, IL, June 2005. IEEE Computer Society.

[21] C. Bodei. *Security Issues in Process Calculi*. PhD thesis, Università di Pisa, Jan. 2000.

[22] C. Bodei, P. Degano, F. Nielson, and H. R. Nielson. Control flow analysis for the π-calculus. In *International Conference on Concurrency Theory (Concur'98)*, volume 1466 of *Lecture Notes on Computer Science*, pages 84–98. Springer, Sept. 1998.

[23] M. Boreale, R. De Nicola, and R. Pugliese. Proof techniques for cryptographic processes. *SIAM Journal on Computing*, 31(3):947–986, 2002.

[24] J. Borgström, S. Briais, and U. Nestmann. Symbolic bisimulation in the spi calculus. In P. Gardner and N. Yoshida, editors, *CONCUR 2004: Concurrency Theory*, volume 3170 of *Lecture Notes on Computer Science*, pages 161–176. Springer, Aug. 2004.

[25] J. Borgström and U. Nestmann. On bisimulations for the spi calculus. In H. Kirchner and C. Ringeissen, editors, *Algebraic Methodology and Software Technology: 9th International Conference, AMAST 2002*, volume 2422 of *Lecture Notes on Computer Science*, pages 287–303, Saint-Gilles-les- Bains, Reunion Island, France, Sept. 2002. Springer.

[26] E. Cohen. Proving protocols safe from guessing. In *Foundations of Computer Security*, Copenhagen, Denmark, July 2002.

[27] R. Corin, J. M. Doumen, and S. Etalle. Analysing password protocol security against off-line dictionary attacks. In *2nd Int. Workshop on Security Issues with Petri Nets and other Computational Models (WISP)*, Electronic Notes in Theoretical Computer Science, June 2004.

[28] R. Corin, S. Malladi, J. Alves-Foss, and S. Etalle. Guess what? here is a new tool that finds some new guessing attacks. In R. Gorrieri, editor, *Workshop on Issues in the Theory of Security (WITS'03)*, Warsaw, Poland, Apr. 2003.

[29] V. Cortier. *Vérification automatique des protocoles cryptographiques*. PhD thesis, ENS de Cachan, Mar. 2003.

[30] S. Delaune and F. Jacquemard. A theory of dictionary attacks and its complexity. In *17th IEEE Computer Security Foundations Workshop*, pages 2–15, Pacific Grove, CA, June 2004. IEEE.

[31] N. Dershowitz and D. A. Plaisted. Rewriting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 9, pages 535–610. Elsevier Science, 2001.

[32] P. H. Drielsma, S. Mödersheim, and L. Viganò. A formalization of off-line guessing for security protocol analysis. In F. Baader and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning: 11th International Conference, LPAR 2004*, volume 3452 of *Lecture Notes on Computer Science*, pages 363–379, Montevideo, Uruguay, Mar. 2005. Springer.

[33] L. Durante, R. Sisto, and A. Valenzano. Automatic testing equivalence verification of spi calculus specifications. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(2):222–284, Apr. 2003.

[34] R. Focardi and R. Gorrieri. The compositional security checker: A tool for the verification of information flow security properties. *IEEE Transactions on Software Engineering*, 23(9):550–571, Sept. 1997.

[35] L. Gong. Verifiable-text attacks in cryptographic protocols. In *INFOCOM '90, The Conference on Computer Communications*, pages 686–693, San Francisco, CA, June 1990. IEEE.

[36] L. Gong, T. M. A. Lomas, R. M. Needham, and J. H. Saltzer. Protecting poorly chosen secrets from guessing attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–656, June 1993.

[37] A. Gordon and A. Jeffrey. Authenticity by typing for security protocols. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 145–159, Cape Breton, Nova Scotia, Canada, June 2001. IEEE Computer Society.

[38] H. Hüttel. Deciding framed bisimilarity. In *4th International Workshop on Verification of Infinite-State Systems (INFINITY'02)*, pages 1–20, Brno, Czech Republic, Aug. 2002.

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

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

[41] G. Lowe. Analyzing protocols subject to guessing attacks. In *Workshop on Issues in the Theory of Security (WITS'02)*, Portland, Oregon, Jan. 2002.

[42] D. Monniaux. Abstracting cryptographic protocols with tree automata. *Science of Computer Programming*, 47(2–3):177–202, 2003.

[43] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1–2):85–128, 1998.

[44] F. Pottier. A simple view of type-secure information flow in the π-calculus. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pages 320–330, Cape Breton, Nova Scotia, June 2002.

[45] F. Pottier and V. Simonet. Information flow inference for ML. In *Proceedings of the 29th ACM Symposium on Principles of Programming Languages (POPL'02)*, pages 319–330, Portland, Oregon, Jan. 2002.

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

Appendix

The Appendix contains proofs of the main results of this paper. Proof scripts for all examples and applications, as well as the tool ProVerif, are available at `http://www.di.ens.fr/~blanchet/obsequi/`.

A Proof of Theorem 2

In this section, we prove the correctness of Algorithms 1, 2, and 3 given in Section 5.3. We begin with preliminary lemmas on modelling equational theories.

A.1 Preliminary lemmas

Lemma 4 *Let N be either a name or a variable. If $\Sigma \vdash M = N$ and $\mathrm{nf}_{\mathcal{S},\Sigma}(\{M\})$, then $M = N$. For any set of terms $\mathcal{M}$, if $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{M})$, then $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{M} \cup \{N\})$.*

Proof We detail the proof for $N = a$.

If we had $M \neq a$, then either M contains a, so M does not satisfy $\mathrm{nf}_{\mathcal{S},\Sigma}(\{M\})$, or M does not contain a. In this latter case, since Σ is invariant by substitution of terms for names, for all M', we have $\Sigma \vdash a\{M'/a\} = M\{M'/a\}$ so $\Sigma \vdash M' = M$, then all terms are equated by Σ, which contradicts the hypothesis.

Let M'' be any subterm of an element of $\mathcal{M}$. We have $\mathrm{nf}_{\mathcal{S},\Sigma}(\{M''\})$, so if $\Sigma \vdash M'' = a$, then $M'' = a$, by the previous property. Moreover, a is irreducible by $\mathcal{S}$. So we have $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{M} \cup \{a\})$. □

Lemma 5 *Assume $\mathcal{S} = \emptyset$ and Σ is any equational theory. Then Property S2 is true.*

Proof We show the following property: If $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{M})$, then for any term M there exists M' such that $\Sigma \vdash M' = M$ and $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{M} \cup \{M'\})$.

The proof is by induction on M.

- Cases $M = a$ and $M = x$: Let $M' = M$; by Lemma 4, $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{M} \cup \{M'\})$.
- Case $M = f(N_1, \ldots, N_n)$: By induction hypothesis, there exist $N'_1, \ldots, N'_n$ such that $\Sigma \vdash N_i = N'_i$ and $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{M} \cup \{N'_1, \ldots, N'_n\})$. (For N_i, we apply the induction hypothesis with $\mathcal{M} \cup \{N'_1, \ldots, N'_{i-1}\}$ instead of $\mathcal{M}$.)

  If there exists a subterm M' of $\mathcal{M} \cup \{N'_1, \ldots, N'_n\}$ such that $\Sigma \vdash f(N_1, \ldots, N_n) = M'$, then we have $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{M} \cup \{M'\})$.

Otherwise, let $M' = f(N'_1, \ldots, N'_n)$. We have $\Sigma \vdash M' = f(N_1, \ldots, N_n)$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{M} \cup \{M'\})$ since the subterms of $\mathcal{M} \cup \{M'\}$ are the subterms of $\mathcal{M} \cup \{N'_1, \ldots, N'_n\}$ and the term M', $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{M} \cup \{N'_1, \ldots, N'_n\})$ and the new subterm M' is different from any subterm of $\mathcal{M} \cup \{N'_1, \ldots, N'_n\}$ modulo the equational theory of Σ. □

A.2 Convergent theories

Lemma 6 *The signature Σ' built by Algorithm 1 models Σ.*

Proof Properties S1 and S3 are obvious.

Let us prove Property S2. Assume that $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{M})$. Let $M' = M{\downarrow}$. Then $\Sigma \vdash M = M'$ and M' is irreducible by $\mathcal{S}$. Let N_1 and N_2 be two subterms of $\mathcal{M} \cup \{M'\}$ such that $\Sigma \vdash N_1 = N_2$, that is, $N_1{\downarrow} = N_2{\downarrow}$. Moreover, N_1 and N_2 are in normal form relatively to $\mathcal{S}$, so $N_1 = N_2$. Hence $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{M} \cup \{M'\})$.

Finally, we prove Property S4. When $M = f(M_1, \ldots, M_n)$, we let $M{\downarrow^s} = f(M_1{\downarrow}, \ldots, M_n{\downarrow})$ be the term obtained by reducing to normal form the strict subterms of M. We first note a few elementary properties of the algorithm:

P1. If $N \to N'$ is in $\mathcal{S}$, then there is $N_1 \to N'_1$ in E such that $T[\sigma N_1] = N{\downarrow^s}$ and $T[\sigma N'_1] = N'{\downarrow}$ for some σ and T. (This is true at the beginning of an execution of the algorithm, and remains true during the execution, since a rule $N_1 \to N'_1$ is removed from E only when there is another rule $N_2 \to N'_2$ such that $N_1 = T[\sigma N_2]$ and $N'_1 = T[\sigma N'_2]$ for some σ and T.)

P2. If N is reducible by a rule in E, then it is also reducible by a rule in $\mathcal{S}$. (This is true at the beginning of an execution of the algorithm and remains true during the execution.)

P3. If $N \to N'$ is in E, then N is not a variable, and all variables of N' occur in N. (This is true at the beginning of an execution of the algorithm and remains true during the execution.)

P4. At the end of the algorithm, if $N_1 \to N'_1$ and $N_2 \to N'_2$ in E are such that $N'_1 = T[N''_1]$, N''_1 is not a variable, and σ_u is the most general unifier of N''_1 and N_2, then there exist $N_3 \to N'_3$ in E, T', and σ such that $T'[\sigma N_3] = (\sigma_u N_1){\downarrow^s}$ and $T'[\sigma N'_3] = \sigma_u T[\sigma_u N'_2]{\downarrow}$. (This simply expresses that the fixpoint is reached: the rule $(\sigma_u N_1){\downarrow^s} \to \sigma_u T[\sigma_u N'_2]{\downarrow}$ has been added to E.)

We show the following two properties, P5(n) for $n > 0$ and P6(n) for $n \geq 0$:

P5(n). If the longest reduction of M by $\mathcal{S}$ is of length n and $M = M{\downarrow^s}$, then there exist $N \to N'$ in E and σ such that $M = \sigma N$ and $M{\downarrow} = \sigma N'$.

P6(n). If the longest reduction of $\sigma N'_1$ by $\mathcal{S}$ is of length n, $M = \sigma N_1 \to^*_{\mathcal{S}} \sigma N'_1$, $M = M{\downarrow^s}$, and $N_1 \to N'_1$ is in E, then there exist $N_2 \to N'_2$ in E and σ' such that $M = \sigma' N_2 \to^*_{\mathcal{S}} \sigma' N'_2 = M{\downarrow}$.

The proof is by induction on n.

- Proof of P5(n) ($n > 0$).

  Since $M = M{\downarrow^s}$, the strict subterms of M are irreducible, so the first application of a rewrite rule in any reduction of M much touch the root function symbol of M. Let $N \to N'$ be this rewrite rule. There exists σ_1 such that $M = \sigma_1 N$. Since $N \to N'$ is in $\mathcal{S}$, by P1, there is $N_1 \to N_1'$ in E such that $T[\sigma_2 N_1] = N{\downarrow^s}$ and $T[\sigma_2 N_1'] = N'{\downarrow}$ for some σ_2 and T. Since the strict subterms of $\sigma_1 N$ are irreducible by $\mathcal{S}$, the strict subterms of N are also irreducible by $\mathcal{S}$, hence $N{\downarrow^s} = N$. Furthermore, $T = [\,]$, since otherwise a strict subterm of N would be reducible by $N_1 \to N_1'$ in E, so using P2, it would also be reducible by $\mathcal{S}$. Hence $\sigma_1\sigma_2 N_1 = \sigma_1 N = M$ and $\sigma_1\sigma_2 N_1' = \sigma_1(N'{\downarrow})$. Let $\sigma = \sigma_1\sigma_2$. Then $M = \sigma N_1 \to_{\mathcal{S}}^+ \sigma N_1'$.

  By P6(n') where n' is the length of the longest reduction of $\sigma N_1'$ ($n' < n$), there exist $N_2 \to N_2'$ in E and σ' such that $M = \sigma' N_2 \to_{\mathcal{S}}^* \sigma' N_2' = M{\downarrow}$, which is P5($n$).

- Proof of P6(n), $n = 0$, $\sigma N_1'$ is irreducible by $\mathcal{S}$. Then $\sigma N_1' = M{\downarrow}$ and we have P6(0) by taking $\sigma' = \sigma$, $N_2 = N_1$, and $N_2' = N_1'$.

- Proof of P6(n), $n > 0$, $\sigma N_1'$ is reducible by $\mathcal{S}$.

  Let us consider a minimal subterm of $\sigma N_1'$ which is reducible by $\mathcal{S}$, that is, a subterm of $\sigma N_1'$ reducible by $\mathcal{S}$ but such that all its strict subterms are irreducible by $\mathcal{S}$. Such a term is of the form $\sigma N_3'$, where N_3' is a non-variable subterm of N_1'. (Indeed, all terms σx and their subterms are irreducible by $\mathcal{S}$, since they are strict subterms of M and $M = M{\downarrow^s}$.)

  The longest reduction of $\sigma N_3'$ is at most as long as the one of $\sigma N_1'$, so by P5, there exist $N_4 \to N_4'$ in E and σ'' such that $\sigma N_3' = \sigma'' N_4$ and $(\sigma N_3'){\downarrow} = \sigma'' N_4'$. Thus we have $M = \sigma N_1 \to_{\mathcal{S}}^* \sigma N_1' = \sigma T[\sigma N_3'] = \sigma T[\sigma'' N_4] \to_{\mathcal{S}}^+ \sigma T[\sigma'' N_4'] = \sigma T[(\sigma N_3'){\downarrow}]$.

  The rewrite rules $N_1 \to N_1'$ and $N_4 \to N_4'$ have a critical pair, that is, $N_1' = T[N_3']$, N_3' is not a variable, and N_3' and N_4 unify, with most general unifier σ_u. By P4, there is $N_5 \to N_5'$ in E such that $T'[\sigma_5 N_5] = (\sigma_u N_1){\downarrow^s}$ and $T'[\sigma_5 N_5'] = \sigma_u T[\sigma_u N_4']{\downarrow}$ for some T' and σ_5. Moreover, $\sigma_u N_1$ is more general than σN_1, so the strict subterms of $\sigma_u N_1$ are irreducible, since the strict subterms of σN_1 are. So $(\sigma_u N_1){\downarrow^s} = \sigma_u N_1$. Furthermore $T' = [\,]$, since otherwise a strict subterm of $\sigma_u N_1$ would be reducible by E, so using P2, it would also be reducible by $\mathcal{S}$. Hence $\sigma_5 N_5 = \sigma_u N_1$ and $\sigma_5 N_5' = \sigma_u T[\sigma_u N_4']{\downarrow}$.

  Then $M = \sigma_1 N_5 \to_{\mathcal{S}}^* \sigma_1 N_5'$ for some σ_1. Moreover, $\sigma N_1' = \sigma T[\sigma'' N_4] \to_{\mathcal{S}}^+ \sigma T[\sigma'' N_4'] \to_{\mathcal{S}}^* \sigma_1 N_5'$, so the longest reduction of $\sigma_1 N_5'$ is strictly shorter than the longest reduction of $\sigma N_1'$, hence by P6 applied to $\sigma_1 N_5'$, there exist σ' and $N_2 \to N_2'$ in E such that $M = \sigma' N_2 \to_{\mathcal{S}}^* \sigma' N_2' = M{\downarrow}$. This yields P6 for $\sigma N_1'$.

We now turn to the proof of Property S4 itself. Assume $\Sigma \vdash f(M_1, \ldots, M_n) = M$ and $\mathrm{nf}_{\mathcal{S},\Sigma}(\{M, M_1, \ldots, M_n\})$. We show that there exist $f(N_1, \ldots, N_n) \to N$ in $\mathrm{def}_{\Sigma'}(f)$ and σ such that $M = \sigma N$ and $M_i = \sigma N_i$. Since M is irreducible, we have $M = f(M_1, \ldots, M_n){\downarrow}$.

- If $f(M_1, \ldots, M_n)$ is irreducible by $\mathcal{S}$, then $f(M_1, \ldots, M_n) = M$ and we have the result using the rule $f(x_1, \ldots, x_n) \to f(x_1, \ldots, x_n)$ always in $\mathrm{def}_{\Sigma'}(f)$.
- Otherwise, we have $f(M_1, \ldots, M_n)\downarrow^s = f(M_1, \ldots, M_n)$ since M_i is irreducible for all $i \in \{1, \ldots, n\}$. Property P5 for the term $f(M_1, \ldots, M_n)$ yields the desired result, since every rule $f(N_1, \ldots, N_n) \to N'$ of E is in $\mathrm{def}_{\Sigma'}(f)$. □

We say that $\mathcal{S}$ is a convergent subterm system when $\mathcal{S}$ is convergent and all its rewrite rules are of the form $M \to N$ where N is either a strict subterm of M or a closed term in normal form with respect to $\mathcal{S}$ [5, 12].

Lemma 7 *When $\mathcal{S}$ is a convergent subterm rewriting system, Algorithm 1 terminates and the final value of E is $normalize(\mathcal{S})$.*

Proof Let $\mathcal{S}$ be a convergent subterm system, with Σ the associated equational theory. Let E_1 be obtained by replacing each rule $f(M_1, \ldots, M_n) \to N$ of $\mathcal{S}$ with $f(M_1\downarrow, \ldots, M_n\downarrow) \to N\downarrow$ and removing rules of the form $M \to M$. Let $E_2 = normalize(\mathcal{S})$. We first show:

P1. if $M \neq N$, $\Sigma \vdash M = N$, and N and the strict subterms of M are in normal form, then there exist $M_1 \to N_1$ in E_2 and σ such that $M = \sigma M_1$ and $N = \sigma N_1$.

Since $\Sigma \vdash M = N$, we have $M\downarrow = N\downarrow$. The term N is in normal form, so $M\downarrow = N$, so $M \to_{\mathcal{S}}^* N$. Since $M \neq N$, $M \to_{\mathcal{S}} M' \to_{\mathcal{S}}^* N$. Since the strict subterms of M are in normal form, there are a rewrite rule $M_1 \to M_1'$ of $\mathcal{S}$ and a substitution σ such that $M = \sigma M_1$ and $M' = \sigma M_1'$. If M_1' is a strict subterm of M_1, M' is a strict subterm of M, so M' is in normal form, hence $M' = N$. If M_1' is a closed term in normal form, $M' = M_1'$ is in normal form, so we also have $M' = N$.

Moreover, M_1' and the strict subterms of M_1 are in normal form since M' and the strict subterms of M are. So the rewrite rule $M_1 \to N_1$ is preserved by the transformation of $\mathcal{S}$ into E_1, so $M_1 \to N_1$ is in E_1. Finally, if $M_1 \to N_1$ is removed when transforming E_1 into E_2, there are another rule $M_1' \to N_1'$ in E_2 and a substitution σ' such that $M_1 = \sigma' M_1'$ and $N_1 = \sigma' N_1'$, so Property P1 holds in any case.

Let $M_0 \to N_0$ be a rewrite rule added by Algorithm 1. We show that $E_2 = normalize(E_2 \cup \{M_0 \to N_0\})$. Let E_3 be obtained by replacing each rule $f(M_1, \ldots, M_n) \to N$ of $E_2 \cup \{M_0 \to N_0\}$ with $f(M_1\downarrow, \ldots, M_n\downarrow) \to N\downarrow$ and removing rules of the form $M \to M$. Since E_2 has already been normalized, when we transform $E_2 \cup \{M_0 \to N_0\}$ into E_3, only $M_0 \to N_0$ is transformed, into a rule $M \to N$. If $M = N$, the rule $M \to N$ is removed, so we immediately have $E_2 = normalize(E_2 \cup \{M_0 \to N_0\})$. Otherwise, by Property P1, there exist $M_1 \to N_1$ in E_2 (so in E_3) and σ such that $M = \sigma M_1$ and $N = \sigma N_1$. Hence $M_0 \to N_0$ is removed by the last step of $normalize$, so $E_2 = normalize(E_2 \cup \{M_0 \to N_0\})$. We conclude that the fixpoint is reached before iterating, and it is E_2. □

A.3 Linear theories

Lemma 8 *The signature Σ' built by Algorithm 2 models Σ.*

Proof Property S1 is obvious. Property S2 follows from Lemma 5. Property S3 follows from the invariant that, if $M \to M'$ is in E, then $\Sigma \vdash M = M'$. Next, we prove Property S4. We first note a few elementary properties of the algorithm:

P1. If $N = N'$ or $N' = N$ is an equation of Σ, then there is $N_1 \to N_1'$ in E such that $T[\sigma N_1] = N$ and $T[\sigma N_1'] = N'$ for some σ and T. (This is true at the beginning of an execution of the algorithm, and remains true during the execution, since a rule $N_1 \to N_1'$ is removed from E only when there is another rule $N_2 \to N_2'$ in E such that $N_1 = T[\sigma N_2]$ and $N_1' = T[\sigma N_2']$ for some σ and T.)

P2. At the end of the algorithm, if $N_1 \to N_1'$ and $N_2 \to N_2'$ in E are such that $N_1' = T[N_1'']$, N_1'' and N_2 are not variables, and σ_u is the most general unifier of N_1'' and N_2, then there exist $N_3 \to N_3'$ in E, T', and σ such that $T'[\sigma N_3] = \sigma_u N_1$ and $T'[\sigma N_3'] = \sigma_u T[\sigma_u N_2']$. (This simply expresses that the fixpoint is reached: the rule $(\sigma_u N_1) \to \sigma_u T[\sigma_u N_2']$ has been added to E.)

Similarly, if $N_1 \to N_1'$ and $N_2 \to N_2'$ in E are such that $N_2 = T[N_2'']$, N_1' and N_2'' are not variables, and σ_u is the most general unifier of N_1' and N_2'', then there exist $N_3 \to N_3'$ in E, T', and σ such that $T'[\sigma N_3] = \sigma_u T[\sigma_u N_1]$ and $T'[\sigma N_3'] = \sigma_u N_2'$.

Let us now prove a few more properties:

P3. For all M, M', if $\Sigma \vdash M = M'$ then $M \to_E^* M'$.

Assume that $\Sigma \vdash M = M'$ comes from one equation of Σ. Then there are $N = N'$ in Σ, T, and σ such that $M = T[\sigma N]$ and $M' = T[\sigma N']$. Hence, by P1, there are $N_1 \to N_1'$ in E, T', and σ' such that $N = T'[\sigma' N_1]$ and $N' = T'[\sigma' N_1']$. So $M = T[(\sigma T')[\sigma\sigma' N_1]] \to_E T[(\sigma T')[\sigma\sigma' N_1']] = M'$. The property stated above follows immediately.

P4. If $M_1 \to_E M_2 \to_E M_3$ using two rules $M \to N$ and $M' \to N'$ of E such that neither N nor M' are variables, $M_1 = T_1[\sigma_1 M]$, $M_2 = T_1[\sigma_1 N] = T_2[\sigma_2 M']$, and $M_3 = T_2[\sigma_2 N']$ for some contexts T_1 and T_2 and substitutions σ_1 and σ_2, then

- either $M_1 \to_E M_3$ in a single step;
- or the rules commute: $M_1 \to_E M_2' \to_E M_3$ where $M_1 \to_E M_2'$ comes from $M' \to N'$ and $M_2' \to_E M_3$ comes from $M \to N$.

We prove the property by case analysis on T_1 and T_2:

(1) The occurrences of the holes of T_1 and T_2 are not nested: there exists T'' such that $T_1 = T''[[\,], \sigma_2 M']$ and $T_2 = T''[\sigma_1 N, [\,]]$. So $M_1 = T''[\sigma_1 M, \sigma_2 M']$, $M_2 = T''[\sigma_1 N, \sigma_2 M']$, and $M_3 = T''[\sigma_1 N, \sigma_2 N']$. Then the rules commute: $M_1 = T''[\sigma_1 M, \sigma_2 M'] \to_E M_2' = T''[\sigma_1 M, \sigma_2 N'] \to_E M_3 = T''[\sigma_1 N, \sigma_2 N']$.

(2) The occurrence of the hole of T_1 is inside the one of T_2: $T_1 = T_2[T']$. We distinguish two subcases:

(2a) T' is an instance of M': $T' = \sigma_3 M'$. So we have $M_1 = T_2[(\sigma_3 M')[\sigma_1 M]]$, $M_2 = T_2[(\sigma_3 M')[\sigma_1 N]]$, and $M_3 = T_2[(\sigma_3 N')[\sigma_1 N]]$. The linearity of N' guarantees that $\sigma_3 N'$ contains at most one hole, since $\sigma_3 M'$ contains one hole.

If $\sigma_3 N'$ contains no hole (that is, the variable x of M' such that $\sigma_3 x$ contains a hole does not occur in N'), then $M_1 = T_2[(\sigma_3 M')[\sigma_1 M]] \to_E M_3 = T_2[(\sigma_3 N')]$ by $M' \to_E N'$.

If $\sigma_3 N'$ contains exactly one hole, the rules commute: $M_1 = T_2[(\sigma_3 M')[\sigma_1 M]] \to_E M_2' = T_2[(\sigma_3 N')[\sigma_1 M]] \to_E M_3 = T_2[(\sigma_3 N')[\sigma_1 N]]$.

(2b) T' is not an instance of M'. Since $T'[\sigma_1 N] = \sigma_2 M'$ and M' is linear, the hole of T' occurs at a non-variable position in M', so N and M' form a critical pair and, by Property P2, E contains a rule that corresponds to the application of both rewrite rules $M \to N$ and $M' \to N'$.

(3) The occurrence of the hole of T_2 is inside the one of T_1: $T_2 = T_1[T']$. The proof is similar to the one for case (2).

Let $\Sigma \vdash f(M_1, \ldots, M_n) = M$ with $\mathrm{nf}_{\mathcal{S},\Sigma}(\{M, M_1, \ldots, M_n\})$. We show that there exist $f(N_1, \ldots, N_n) \to N$ in $\mathrm{def}_{\Sigma'}(f)$ and σ such that $M = \sigma N$ and $M_i = \sigma N_i$ for all $i \in \{1, \ldots, n\}$.

Since $\Sigma \vdash f(M_1, \ldots, M_n) = M$, we have $f(M_1, \ldots, M_n) \to_E^* M$ by P3. Consider a shortest sequence such that $f(M_1, \ldots, M_n) \to_E^* M$.

- In this sequence, consecutive rewrite rules always commute, because otherwise we would obtain a shorter sequence by P4.

- If this sequence uses a rule $x \to M'$ in E, consider the last such rule. It commutes with the rule that immediately follows. So we obtain a sequence in which $x \to M'$ is applied last. This is impossible since $\mathrm{nf}_{\mathcal{S},\Sigma}(\{M\})$. From now on, we consider a sequence that does not use any rewrite rule of the form $x \to M'$.

- If this sequence uses no rewrite rule applied with empty context, then $M = f(M_1', \ldots, M_n')$ and $M_i \to_E^* M_i'$, so $\Sigma \vdash M_i = M_i'$. Since $\mathrm{nf}_{\mathcal{S},\Sigma}(\{M_1, \ldots, M_n, M\})$, $M_i = M_i'$, so $M = f(M_1, \ldots, M_n)$. Then $f(x_1, \ldots, x_n) \to f(x_1, \ldots, x_n)$ in $\mathrm{def}_{\Sigma'}(f)$ and $\sigma x_i = M_i$ yields the desired result.

- If this sequence uses at least one rewrite rule applied with empty context, let $f(N_1, \ldots, N_n) \to N$ be the first such rule.

  If the sequence uses a rule $M' \to x$ in E before $f(N_1, \ldots, N_n) \to N$, then this rule is applied with non-empty context (because otherwise $f(N_1, \ldots, N_n) \to N$ would not be the first rule with empty context). Consider the first such rule. This rule commutes with the rule just before it. Moreover, after commutation, $M' \to x$ is still applied with non-empty context. (The only case that would make the context disappear is when the rewrite rule before was $y \to M''$, but this case cannot occur as shown above.) So we obtain a sequence in which $M' \to x$ is applied first, and with non-empty context. This is impossible since $\mathrm{nf}_{\mathcal{S},\Sigma}(\{M_1, \ldots, M_n\})$. So we consider a sequence not using rules of the form $M' \to x$ before $f(N_1, \ldots, N_n) \to N$.

The rule $f(N_1, \ldots, N_n) \to N$ commutes with the rule just before it. The rule $f(N_1, \ldots, N_n) \to N$ is still applied with an empty context after commutation. So we can obtain a sequence in which $f(N_1, \ldots, N_n) \to N$ is applied first. All rewrite rules after the first one are applied with a context that is an instance of N (because otherwise N is not a variable and the first rule applied with a context that is not an instance of N can be commuted with other rewrite rules so that it occurs just after $f(N_1, \ldots, N_n) \to N$, so it has a critical pair with $f(N_1, \ldots, N_n) \to N$, so we could obtain a shorter sequence by P2). So $M = \sigma' N$ for some σ'. Furthermore $f(M_1, \ldots, M_n) = f(\sigma N_1, \ldots, \sigma N_n) \to_E \sigma N \to_E^* M = \sigma' N$ for some σ. Then for all $x \in fv(N)$, $\sigma x \to_E^* \sigma' x$, so for all $x \in fv(N)$, $\Sigma \vdash \sigma x = \sigma' x$. Moreover, $\mathrm{nf}_{\mathcal{S},\Sigma}(M_1, \ldots, M_n, M)$, so $\sigma x = \sigma' x$, and $M = \sigma N$, which yields the result. □

A.4 Union of disjoint equational theories

Let Σ be a signature such that its set of function symbols can be partitioned into $F_1 \cup F_2$ and its set of equations can be partitioned into $E_1' \cup E_2'$, where E_1' contains only function symbols in F_1 and E_2' in F_2. Let Σ_1 be the signature obtained by considering only the equations E_1', and Σ_2 only the equations E_2'.

Lemma 9 *If* $\Sigma \vdash f(M_1, \ldots, M_n) = M$, $\mathrm{nf}_{\emptyset,\Sigma}(\{M, M_1, \ldots, M_n\})$, *and* $f \in F_i$ *(*$i = 1$ *or* 2*) then* $\Sigma_i \vdash f(M_1, \ldots, M_n) = M$.

Proof To prove this result, we use the decision algorithm for the word problem in a union of disjoint equational theories, by Baader and Tinelli [11, Section 4]. We use the notations of [11], and we refer the reader to that paper for details.

Assume $i = 1$. (The case $i = 2$ is symmetric.) Let us start with $S_0 = \{x_0 \not\equiv y_0, x_0 \equiv f(M_1, \ldots, M_n), y_0 \equiv M\}$. Since $\Sigma \vdash f(M_1, \ldots, M_n) = M$, by completeness of their algorithm, their algorithm terminates with $S = \{v \not\equiv v\} \cup T$.

Let S be a set of equations and disequations, such that all equations of S are of the form $v \equiv M$, if $v \equiv M$ and $v \equiv N$ are in S then $M = N$, and $\prec$ is acyclic on S. Let us define a substitution σ by $\sigma v = M$ when $(v \equiv M) \in S$. Since $\prec$ is acyclic on S, we can define σ^* as the substitution obtained by composing σ with itself as many times as needed so that terms do not change any more. Let $\mathrm{rec}_S(v) = \sigma^* v$.

If we apply the rules of the algorithm according to a suitable strategy (made explicit below), we can show that the algorithm preserves the following invariant:

P1. There is no equation $v \equiv M'$ in S_j such that $(v \equiv M') \prec (x_0 \equiv M'')$.

P2. If $j > 0$, then for all $v \neq x_0$ such that v occurs in S_{j-1} and S_j, we have $\mathrm{rec}_{S_{j-1}}(v) = \mathrm{rec}_{S_j}(v)$.

P3. For all $v \neq x_0$ such that v occurs in S_j, $\mathrm{rec}_{S_j}(v)$ is a subterm of $M_1, \ldots, M_n, M$ (so if $v, v' \neq x_0$ occur in S_j and $\Sigma \vdash \mathrm{rec}_{S_j}(v) = \mathrm{rec}_{S_j}(v')$, then $\mathrm{rec}_{S_j}(v) = \mathrm{rec}_{S_j}(v')$, since $\mathrm{nf}_{\emptyset,\Sigma}(\{M, M_1, \ldots, M_n\})$).

P4. If $j > 0$ and $x_0 \equiv M'' \in S_j$, then $\Sigma_1 \vdash \mathrm{rec}_{S_{j-1}}(x_0) = \mathrm{rec}_{S_j}(x_0)$.

P5. When $x_0 \equiv M'' \in S_j$, M'' is a non-variable 1-term.

P6. If $j > 0$, $u \not\equiv u' \in S_{j-1}$, and $v \not\equiv v' \in S_j$, then $\Sigma_1 \vdash \mathrm{rec}_{S_{j-1}}(u) = \mathrm{rec}_{S_j}(v)$ and $\Sigma_1 \vdash \mathrm{rec}_{S_{j-1}}(u') = \mathrm{rec}_{S_j}(v')$.

During the first stage (construction of the abstraction system), these properties are obvious. We even have $\mathrm{rec}_{S_{j-1}}(v) = \mathrm{rec}_{S_j}(v)$ for all v that occur in S_{j-1}, and the disequation $x_0 \not\equiv y_0$ is not changed.

During the second stage (application of Coll1, Coll2, Ident, Simpl), we do not apply Simpl since the authors remark that it is not necessary. We show that if S_{j-1} is transformed into S_j by Coll1, Coll2, or Ident, and S_{j-1} satisfies the invariant, then so does S_j.

- For Coll1 and Coll2 with $x \neq x_0$, $\Sigma_i \vdash y = t$, so $\Sigma \vdash \mathrm{rec}_{S_{j-1}}(x) = \mathrm{rec}_{S_{j-1}}(y)$, so by P3, $\mathrm{rec}_{S_{j-1}}(x) = \mathrm{rec}_{S_{j-1}}(y)$, so $y = t$. Then for all v that occur in S_j, $\mathrm{rec}_{S_{j-1}}(v) = \mathrm{rec}_{S_j}(v)$, so we have P2 and P4 for S_j. P3 holds for S_j since P2 holds for S_j and P3 holds for S_{j-1}. We have $\mathrm{rec}_{S_{j-1}}(x) = \mathrm{rec}_{S_{j-1}}(y) = \mathrm{rec}_{S_j}(y)$, so P6 follows. P1 and P5 are easy to show.

- For Coll1 with $x = x_0$, $\Sigma_1 \vdash y = t$, and $T\{r/x_0\} = T$ since x_0 does not occur in the right-hand side of equalities by P1. So for all v that occur in S_j (that is, all v that occur in S_{j-1} except x_0), $\mathrm{rec}_{S_{j-1}}(v) = \mathrm{rec}_{S_j}(v)$, so we have P2 for S_j; P3 follows. P4 and P5 hold since S_j contains no equation of the form $x_0 \equiv M''$. Since $\Sigma \vdash y = t$, we have $\Sigma_1 \vdash \mathrm{rec}_{S_{j-1}}(x_0) = \mathrm{rec}_{S_{j-1}}(y) = \mathrm{rec}_{S_j}(y)$, so P6 follows. P1 is easy to show.

- For Coll2 with $x = x_0$, $\Sigma_1 \vdash y = t$, and T is replaced with $T\{y/x_0\}$, which modifies only the disequation, since x_0 does not occur in the right-hand side of equalities by P1. We conclude as in the previous case.

- For Ident, we never apply Ident with $y = x_0$; when Ident would be applicable with $y = x_0$, we apply instead Ident with $x = x_0$ (which is possible by P1).

  If we apply Ident with $x, y \neq x_0$, then $\Sigma_i \vdash s = t$, so $\Sigma \vdash \mathrm{rec}_{S_{j-1}}(x) = \mathrm{rec}_{S_{j-1}}(y)$, so by P3, $\mathrm{rec}_{S_{j-1}}(x) = \mathrm{rec}_{S_{j-1}}(y)$, so $s = t$. Then, for all v that occur in S_j, $\mathrm{rec}_{S_{j-1}}(v) = \mathrm{rec}_{S_j}(v)$, so we have P2 and P4; P3 follows. We have $\mathrm{rec}_{S_{j-1}}(x) = \mathrm{rec}_{S_{j-1}}(y) = \mathrm{rec}_{S_j}(y)$, so P6 follows. P1 and P5 are easy to show.

  If we apply Ident with $x = x_0, y \neq x_0$, then $\Sigma_1 \vdash s = t$. x_0 does not occur in the right-hand side of equalities by P1. So replacing x_0 with y in T changes only the disequation. Then for all v that occur in S_j, $\mathrm{rec}_{S_{j-1}}(v) = \mathrm{rec}_{S_j}(v)$, so we have P2 and P4; P3 follows. Since $\Sigma_1 \vdash s = t$, we have $\Sigma \vdash \mathrm{rec}_{S_{j-1}}(x_0) = \mathrm{rec}_{S_{j-1}}(y) = \mathrm{rec}_{S_j}(y)$, so P6 follows. P1 and P5 are easy to show.

Since, in the end, S contains $v \not\equiv v$, by P6, we have $\Sigma_1 \vdash \mathrm{rec}_{S_0}(x_0) = \mathrm{rec}_S(v)$ and $\Sigma_1 \vdash \mathrm{rec}_{S_0}(y_0) = \mathrm{rec}_S(v)$ which implies $\Sigma_1 \vdash f(M_1, \ldots, M_n) = M$. □

This result can be used to prove the correctness of Algorithm 3.

Lemma 10 *The signature Σ' built by Algorithm 3 models Σ.*

Proof The set of function symbols of Σ can be partitioned into $F_1 \cup F_2$, where E_{conv} contains only function symbols in F_1 and E_{lin} in F_2. Let Σ_1 be the signature obtained by considering only equations E_{conv}, and Σ_2 only E_{lin}.

Because of the particular way in which we prove that subsets E_i are convergent, we have that their union E_{conv} is also convergent, so we can apply Algorithm 1 to E_{conv}. (When we prove termination of each E_i via a lexicographic path ordering, we order the function symbols of E_i. We order the function symbols of E_{conv} by the union of these orderings. Then the corresponding lexicographic path ordering shows the termination of E_{conv}. The confluence of E_{conv} follows from the confluence of every E_i by the critical-pair theorem.)

Properties S1 and S3 are obvious. We prove Property S2 by induction on M:

- Cases $M = a$ and $M = x$: Let $M' = M$; by Lemma 4, $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{M} \cup \{M\})$.
- Case $M = f(M_1, \ldots, M_n)$: By induction hypothesis, there exist $M'_1, \ldots, M'_n$ such that $\Sigma \vdash M_i = M'_i$ and $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{M} \cup \{M'_1, \ldots, M'_n\})$. (For M_i, we apply the induction hypothesis with $\mathcal{M} \cup \{M'_1, \ldots, M'_{i-1}\}$ instead of $\mathcal{M}$.)

  Case 1: there exists a subterm M' of $\mathcal{M} \cup \{M'_1, \ldots, M'_n\}$ such that $\Sigma \vdash f(M_1, \ldots, M_n) = M'$. Then M' is irreducible by $\mathcal{S}$ and $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{M} \cup \{M'\})$, so we have the result.

  Case 2: there exists no subterm M'' of $\mathcal{M} \cup \{M'_1, \ldots, M'_n\}$ such that $\Sigma \vdash f(M_1, \ldots, M_n) = M''$.

  Case 2.1: Assume $f \in F_2$. Let $M' = f(M'_1, \ldots, M'_n)$. We have $\Sigma \vdash f(M_1, \ldots, M_n) = M'$. Moreover, M' is irreducible by $\mathcal{S}$ since $M'_1, \ldots, M'_n$ are, $f \in F_2$, and no rewrite rule of $\mathcal{S}$ contains a function symbol in F_2 or a variable in the left-hand side. Then $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{M} \cup \{M'\})$ since the subterms of $\mathcal{M} \cup \{M'\}$ are the subterms of $\mathcal{M} \cup \{M'_1, \ldots, M'_n\}$ and the term M', $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{M} \cup \{M'_1, \ldots, M'_n\})$, and the new subterm M' is different from any subterms of $\mathcal{M} \cup \{M'_1, \ldots, M'_n\}$ modulo the equational theory of Σ.

  Case 2.2: Assume $f \in F_1$. Let $M' = f(M'_1, \ldots, M'_n)\downarrow$. We have $\Sigma \vdash f(M_1, \ldots, M_n) = M'$. Moreover, M' is irreducible by $\mathcal{S}$ by definition. If $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{M} \cup \{M'\})$ was wrong, there would exist N and N' subterms of $\mathcal{M} \cup \{M'\}$ such that $\Sigma \vdash N = N'$ and $N \neq N'$. Let us choose such terms N and N' such that the pair $(\max(\mathit{size}(N), \mathit{size}(N')), \min(\mathit{size}(N), \mathit{size}(N')))$ ordered lexicographically is minimal. When $\mathit{size}(N) < \mathit{size}(N')$, we swap N and N', so that we always have $\mathit{size}(N) \geq \mathit{size}(N')$. Let $N = f'(N_1, \ldots, N_{n'})$. We have $\mathrm{nf}_{\mathcal{S},\Sigma}(N_1, \ldots, N_{n'}, N')$. (If $\mathrm{nf}_{\mathcal{S},\Sigma}(N_1, \ldots, N_{n'}, N')$ was not true, considering subterms of $N_1, \ldots, N_{n'}, N'$ that falsify $\mathrm{nf}_{\mathcal{S},\Sigma}(N_1, \ldots, N_{n'}, N')$ would yield a smaller counterexample.)

  Notice that $\mathrm{nf}_{\mathcal{S},\Sigma}(N_1, \ldots, N_{n'}, N')$ implies $\mathrm{nf}_{\emptyset,\Sigma}(N_1, \ldots, N_{n'}, N')$, so we can apply Lemma 9.

  If $f' \in F_1$, then $\Sigma_1 \vdash f'(N_1, \ldots, N_{n'}) = N'$ by Lemma 9. Hence $f'(N_1, \ldots, N_{n'})\downarrow = N'\downarrow$. The terms N' and $f'(N_1, \ldots, N_{n'})$ are subterms of $\mathcal{M} \cup \{M'\}$,

so they are irreducible by $\mathcal{S}$, so $f'(N_1, \ldots, N_{n'}) = N'$. Hence, we have a contradiction.

If $f' \in F_2$, then $\Sigma_2 \vdash f'(N_1, \ldots, N_{n'}) = N'$ by Lemma 9. Since the reduction of $f(M'_1, \ldots, M'_n)$ into M' modifies only the top-level context of M' within F_1, all subterms of $\mathcal{M} \cup \{M'\}$ with root symbol in F_2 are also subterms of $\mathcal{M} \cup \{M'_1, \ldots, M'_n\}$, so they satisfy $\mathrm{nf}_{\mathcal{S},\Sigma}$. So the root symbol of N' is in F_1. Let $N' = f''(N'_1, \ldots, N'_{n''})$, $f'' \in F_1$. If $\mathrm{nf}_{\mathcal{S},\Sigma}(N'_1, \ldots, N'_{n''}, N)$, we can apply the case $f' \in F_1$ above to $\Sigma \vdash f''(N'_1, \ldots, N'_{n''}) = N$. Otherwise, the counterexample to $\mathrm{nf}_{\mathcal{S},\Sigma}(N'_1, \ldots, N'_{n''}, N)$ is not smaller than $\Sigma \vdash N = N'$ since it is minimal, and $size(N) \geq size(N')$, so the counterexample to $\mathrm{nf}_{\mathcal{S},\Sigma}(N'_1, \ldots, N'_{n''}, N)$ consists of two subterms of N; this situation is impossible since $N = f'(N_1, \ldots, N_{n'})$ is a subterm of $\mathcal{M} \cup \{M'_1, \ldots, M'_n\}$, so all its subterms satisfy $\mathrm{nf}_{\mathcal{S},\Sigma}$.

Hence we have $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{M} \cup \{M'\})$.

Finally, we prove Property S4. Let $\Sigma \vdash f(M_1, \ldots, M_n) = M$ with $\mathrm{nf}_{\mathcal{S},\Sigma}(\{M, M_1, \ldots, M_n\})$. If $f \in F_i$ $(i = 1, 2)$, then $\Sigma_i \vdash f(M_1, \ldots, M_n) = M$ by Lemma 9 (since $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{M}')$ implies $\mathrm{nf}_{\emptyset,\Sigma}(\mathcal{M}')$). If $i = 1$, we conclude by Property S4 for Algorithm 1. If $i = 2$, we conclude by Property S4 for Algorithm 2. □

B Proofs of Lemmas 1 and 2

From this point on, we assume that Σ' models Σ. We say that a term or a term evaluation is plain when it does not contain diff.

B.1 Preliminary lemmas

The following lemma shows the soundness of $D' \Downarrow' (M', \sigma')$ with respect to $D \Downarrow_{\Sigma'} M$.

Lemma 11 *Let σ be a closed substitution.*

Let D be a plain term evaluation. If $\sigma D \Downarrow_{\Sigma'} M$, then there exist M', σ_1, and σ'_1 such that $D \Downarrow' (M', \sigma_1)$, $M = \sigma'_1 M'$, and $\sigma = \sigma'_1 \sigma_1$ except on fresh variables introduced in the computation of $D \Downarrow' (M', \sigma_1)$.

Let $D_1, \ldots, D_n$ be plain term evaluations. If for all $i \in \{1, \ldots, n\}$, $\sigma D_i \Downarrow_{\Sigma'} M_i$, then there exist $M'_1, \ldots, M'_n$, σ_1, and σ'_1 such that $(D_1, \ldots, D_n) \Downarrow' ((M'_1, \ldots, M'_n), \sigma_1)$, $M_i = \sigma'_1 M'_i$ for all $i \in \{1, \ldots, n\}$, and $\sigma = \sigma'_1 \sigma_1$ except on fresh variables introduced in the computation of $(D_1, \ldots, D_n) \Downarrow' ((M'_1, \ldots, M'_n), \sigma_1)$.

Proof The proof is by mutual induction following the definition of $\Downarrow'$.

- Case $D = M'$: Take $\sigma_1 = \emptyset$, $\sigma'_1 = \sigma$. Since $M = \sigma M'$, we have the result.
- Case $D = \mathrm{eval}\ h(D_1, \ldots, D_n)$: Since $\mathrm{eval}\ h(\sigma D_1, \ldots, \sigma D_n) \Downarrow_{\Sigma'} M$, there exist $h(N_1, \ldots, N_n) \to N$ in $\mathrm{def}_{\Sigma'}(h)$ and σ_m such that $\sigma D_i \Downarrow_{\Sigma'} \sigma_m N_i$ and $M = \sigma_m N$.

By induction hypothesis, there exist M'_i, σ_1, and σ'_1 such that $(D_1, \ldots, D_n) \Downarrow' ((M'_1, \ldots, M'_n), \sigma_1)$, $\sigma_m N_i = \sigma'_1 M'_i$ for all $i \in \{1, \ldots, n\}$, and $\sigma = \sigma'_1 \sigma_1$ except on fresh variables introduced in the computation of $(D_1, \ldots, D_n) \Downarrow' ((M'_1, \ldots, M'_n), \sigma_1)$.

Let σ_u be the most general unifier of M'_i and N_i for $i \in \{1, \ldots, n\}$. (The substitution σ_u exists since $\sigma_m N_i = \sigma'_1 M'_i$.) Then $\text{eval } h(D_1, \ldots, D_n) \Downarrow' (\sigma_u N, \sigma_u \sigma_1)$. The substitution that maps variables of N_i, N as σ_m and other variables as σ'_1 is a unifier of M'_i and N_i, so there exists σ''_1 such that $\sigma_m = \sigma''_1 \sigma_u$ on variables of N_i, N, and $\sigma'_1 = \sigma''_1 \sigma_u$ on other variables.

Then $\sigma''_1 \sigma_u N = \sigma_m N = M$ and $\sigma''_1 \sigma_u \sigma_1 = \sigma'_1 \sigma_1 = \sigma$ except on fresh variables introduced in the computation of $(D_1, \ldots, D_n) \Downarrow' ((M'_1, \ldots, M'_n), \sigma_1)$ and variables of $N_1, \ldots, N_n, N$, that is, fresh variables introduced in the computation of $D \Downarrow' (\sigma_u N, \sigma_u \sigma_1)$.

- Case $(D_1, \ldots, D_n)$: We have, for all i in $\{1, \ldots, n\}$, $\sigma D_i \Downarrow_{\Sigma'} M_i$.

  By induction hypothesis, there exist M'_i, σ_1, and σ'_1 such that $(D_1, \ldots, D_{n-1}) \Downarrow' ((M'_1, \ldots, M'_{n-1}), \sigma_1)$, $M_i = \sigma'_1 M'_i$ for all $i \in \{1, \ldots, n-1\}$, and $\sigma = \sigma'_1 \sigma_1$ except on fresh variables introduced in the computation of $(D_1, \ldots, D_{n-1}) \Downarrow' ((M'_1, \ldots, M'_{n-1}), \sigma_1)$.

  Then $\sigma D_n = \sigma'_1 \sigma_1 D_n$, so $\sigma'_1(\sigma_1 D_n) \Downarrow_{\Sigma'} M_n$. So by induction hypothesis, there exist M'_n, σ_2, and σ'_2 such that $\sigma_1 D_n \Downarrow' (M'_n, \sigma_2)$, $M_n = \sigma'_2 M'_n$, and $\sigma'_1 = \sigma'_2 \sigma_2$ except on fresh variables introduced in the computation of $\sigma_1 D_n \Downarrow' (M'_n, \sigma_2)$.

  Hence $(D_1, \ldots, D_n) \Downarrow' ((\sigma_2 M'_1, \ldots, \sigma_2 M'_{n-1}, M'_n), \sigma_2 \sigma_1)$, $M_i = \sigma'_1 M'_i = \sigma'_2(\sigma_2 M'_i)$ for all $i \in \{1, \ldots, n-1\}$, $M_n = \sigma'_2 M'_n$, and $\sigma = \sigma'_1 \sigma_1 = \sigma'_2 \sigma_2 \sigma_1$ except on fresh variables introduced in the computation of $(D_1, \ldots, D_n) \Downarrow' ((\sigma_2 M'_1, \ldots, \sigma_2 M'_{n-1}, M'_n), \sigma_2 \sigma_1)$. □

Lemma 12 *Let σ be a closed substitution and M a plain term. If $\Sigma \vdash M' = \sigma M$ and $\text{nf}_{\mathcal{S},\Sigma}(\{M'\} \cup \{\sigma x \mid x \in fv(M)\})$ then $\sigma\text{addeval}(M) \Downarrow_{\Sigma'} M'$.*

Proof The proof is by induction on M.

- Case $M = x$: We have $\Sigma \vdash \sigma x = \sigma M = M'$. Since $\text{nf}_{\mathcal{S},\Sigma}(\{\sigma x, M'\})$, $\sigma x = M'$. Moreover, $\sigma\text{addeval}(M) = \sigma x \Downarrow_{\Sigma'} \sigma x = M'$.

- Case $M = a$: Since $\Sigma \vdash M' = \sigma M$ and $\text{nf}_{\mathcal{S},\Sigma}(\{M'\})$, we have $M' = a$ by Lemma 4, so $\sigma\text{addeval}(M) = a \Downarrow_{\Sigma'} a = M'$.

- Case $M = f(M_1, \ldots, M_n)$: We have $\Sigma \vdash M' = \sigma M = f(\sigma M_1, \ldots, \sigma M_n)$ and $\text{nf}_{\mathcal{S},\Sigma}(\{M'\} \cup \{\sigma x \mid x \in fv(M)\})$. By Property S2, there exist $M'_1, \ldots, M'_n$ such that $\Sigma \vdash \sigma M_i = M'_i$ and $\text{nf}_{\mathcal{S},\Sigma}(\{M', M'_1, \ldots, M'_n\} \cup \{\sigma x \mid x \in fv(M)\})$. By Property S4, there exist $f(N_1, \ldots, N_n) \to N$ in $\text{def}_{\Sigma'}(f)$ and σ' such that $M' = \sigma' N$ and $\sigma' N_i = M'_i$ for all $i \in \{1, \ldots, n\}$. By induction hypothesis, $\sigma\text{addeval}(M_i) \Downarrow_{\Sigma'} M'_i = \sigma' N_i$ for all $i \in \{1, \ldots, n\}$. By definition of $\Downarrow_{\Sigma'}$, $\sigma\text{addeval}(M) = \text{eval } f(\sigma\text{addeval}(M_1), \ldots, \sigma\text{addeval}(M_n)) \Downarrow_{\Sigma'} \sigma' N = M'$. □

The following lemma shows the soundness of the rewrite rules of h in Σ' with respect to these rewrite rules in Σ. When h is a destructor, this is proved using the previous two lemmas, and when h is a constructor, this follows from the definition of "Σ' models Σ". Lemma 14 extends this result to a term evaluation D by induction on D.

Lemma 13 *If $h(N_1, \ldots, N_n) \to N$ is in $\mathrm{def}_\Sigma(h)$, $\Sigma \vdash M_i = \sigma N_i$ for all $i \in \{1, \ldots, n\}$, $\Sigma \vdash M = \sigma N$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(\{M_1, \ldots, M_n, M\})$, then there exist $h(N'_1, \ldots, N'_n) \to N'$ in $\mathrm{def}_{\Sigma'}(h)$ and σ' such that $M_i = \sigma' N'_i$ for all $i \in \{1, \ldots, n\}$ and $M = \sigma' N'$.*

Proof Case 1: h is a constructor in Σ. We have $\Sigma \vdash M = h(M_1, \ldots, M_n)$. The result follows from Property S4.

Case 2: h is a destructor in Σ. By Property S2, there exists σ_0 such that $\Sigma \vdash \sigma_0 x = \sigma x$ for all $x \in fv(N_1, \ldots, N_n, N)$ and $\mathrm{nf}_{\mathcal{S},\Sigma}(\{M_1, \ldots, M_n, M\} \cup \{\sigma_0 x \mid x \in fv(N_1, \ldots, N_n, N)\})$. So $\Sigma \vdash M = \sigma_0 N$ and $\Sigma \vdash M_i = \sigma_0 N_i$ for all $i \in \{1, \ldots, n\}$. By Lemma 12, $\sigma_0 \mathrm{addeval}(N) \Downarrow_{\Sigma'} M$ and $\sigma_0 \mathrm{addeval}(N_i) \Downarrow_{\Sigma'} M_i$ for all $i \in \{1, \ldots, n\}$. By Lemma 11, there exist $N'_1, \ldots, N'_n$, N', σ_1, and σ' such that $\mathrm{addeval}(N_1, \ldots, N_n, N) \Downarrow' ((N'_1, \ldots, N'_n, N'), \sigma_1)$, $\sigma' N'_i = M_i$ for all $i \in \{1, \ldots, n\}$, and $\sigma' N' = M$. Then $h(N'_1, \ldots, N'_n) \to N'$ is in $\mathrm{def}_{\Sigma'}(h)$, $\sigma' N'_i = M_i$ for all $i \in \{1, \ldots, n\}$, and $\sigma' N' = M$. □

Lemma 14 *Let D be a plain term evaluation. If $D \Downarrow_\Sigma M$, $\Sigma \vdash M' = M$, $\Sigma \vdash D' = D$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(\{M', D'\})$, then $D' \Downarrow_{\Sigma'} M'$.*

Proof The proof is by induction on D.

- Case $D = M$: We have $M \Downarrow_\Sigma M$, so $\Sigma \vdash D' = D = M = M'$ and $\mathrm{nf}_{\mathcal{S},\Sigma}(\{M', D'\})$ so $D' = M'$, and $D' \Downarrow_{\Sigma'} M'$.
- Case $D = \mathrm{eval}\, h(D_1, \ldots, D_n)$: Since $D \Downarrow_\Sigma M$, we have that $h(N_1, \ldots, N_n) \to N$ is in $\mathrm{def}_\Sigma(h)$, $D_i \Downarrow_\Sigma M_i$ and $\Sigma \vdash \sigma N_i = M_i$ for all $i \in \{1, \ldots, n\}$, and $\sigma N = M$. So $\Sigma \vdash \sigma N = M'$. Since $\Sigma \vdash D' = D$, we have $D' = \mathrm{eval}\, h(D'_1, \ldots, D'_n)$, with $\Sigma \vdash D'_i = D_i$ for all $i \in \{1, \ldots, n\}$. By Property S2, there exist $M'_1, \ldots, M'_n$ such that $\Sigma \vdash M_i = M'_i$ for all $i \in \{1, \ldots, n\}$ and $\mathrm{nf}_{\mathcal{S},\Sigma}(\{M', D', M'_1, \ldots, M'_n\})$. By induction hypothesis, $D'_i \Downarrow_{\Sigma'} M'_i$ for all $i \in \{1, \ldots, n\}$. By Lemma 13, there exist $h(N'_1, \ldots, N'_n) \to N'$ in $\mathrm{def}_{\Sigma'}(h)$ and σ' such that $M' = \sigma' N'$ and $\sigma' N'_i = M'_i$ for all $i \in \{1, \ldots, n\}$. Then $D' \Downarrow_{\Sigma'} \sigma' N' = M'$. □

We define the function removeeval such that $\mathrm{removeeval}(D) = M$ where D is a term evaluation that contains no destructor, and M is the term obtained by removing any eval before the function symbols of D.

Lemma 15 *Assume that D is a plain term evaluation that contains no destructor. If $D \Downarrow' (M, \sigma)$ then $\Sigma \vdash \sigma \mathrm{removeeval}(D) = M$.*

Assume that $D_1, \ldots, D_n$ are plain term evaluations that contain no destructor. If $(D_1, \ldots, D_n) \Downarrow' ((M_1, \ldots, M_n), \sigma)$ then $\Sigma \vdash \sigma\mathrm{removeeval}(D_i) = M_i$ for all $i \in \{1, \ldots, n\}$.

Proof The proof is by mutual induction following the definition of $\Downarrow'$.

- Case $D = M$: We have $\sigma = \emptyset$, so $\Sigma \vdash \sigma M = M$.
- Case $D = \mathrm{eval}\ f(D_1, \ldots, D_n)$: We have $\mathrm{eval}\ f(D_1, \ldots, D_n) \Downarrow' (\sigma_u N, \sigma_u\sigma)$ where $(D_1, \ldots, D_n) \Downarrow' ((M_1, \ldots, M_n), \sigma)$, f is a constructor in Σ, $f(N_1, \ldots, N_n) \to N$ is in $\mathrm{def}_{\Sigma'}(f)$ (with new variables), and σ_u is the most general unifier of $(M_1, N_1), \ldots, (M_n, N_n)$. Then by Property S3, $\Sigma \vdash f(N_1, \ldots, N_n) = N$. By induction hypothesis, $\Sigma \vdash \sigma\mathrm{removeeval}(D_i) = M_i$. Moreover we have $\sigma_u M_i = \sigma_u N_i$. Hence we obtain $\Sigma \vdash \sigma_u\sigma\mathrm{removeeval}(\mathrm{eval}\ f(D_1, \ldots, D_n)) = f(\sigma_u\sigma\mathrm{removeeval}(D_1), \ldots, \sigma_u\sigma\mathrm{removeeval}(D_n)) = f(\sigma_u M_1, \ldots, \sigma_u M_n) = f(\sigma_u N_1, \ldots, \sigma_u N_n) = \sigma_u N$.
- Case $(D_1, \ldots, D_n)$: We have $(D_1, \ldots, D_n) \Downarrow' ((\sigma' M_1, \ldots, \sigma' M_{n-1}, M_n), \sigma'\sigma)$ where $(D_1, \ldots, D_{n-1}) \Downarrow' ((M_1, \ldots, M_{n-1}), \sigma)$ and $\sigma D_n \Downarrow' (M_n, \sigma')$. Then by induction hypothesis, $\Sigma \vdash \sigma\mathrm{removeeval}(D_i) = M_i$ for $i \in \{1, \ldots, n-1\}$ and $\Sigma \vdash \sigma'\mathrm{removeeval}(\sigma D_n) = M_n$. Hence, $\Sigma \vdash \sigma'\sigma\mathrm{removeeval}(D_i) = \sigma' M_i$ for $i \in \{1, \ldots, n-1\}$ and $\Sigma \vdash \sigma'\sigma\mathrm{removeeval}(D_n) = M_n$. □

The following two lemmas show a completeness property: we do not lose precision by translating computation in Σ into computations in Σ'. The proof of Lemma 16 relies on Lemma 15 for destructor applications.

Lemma 16 *If $h(N_1, \ldots, N_n) \to N$ is in $\mathrm{def}_{\Sigma'}(h)$ then there exists $h(N'_1, \ldots, N'_n) \to N'$ in $\mathrm{def}_{\Sigma}(h)$ and σ such that $\Sigma \vdash N_i = \sigma N'_i$ for all $i \in \{1, \ldots, n\}$ and $\Sigma \vdash N = \sigma N'$.*

Proof Case 1: h is a constructor in Σ. By Property S3, $\Sigma \vdash h(N_1, \ldots, N_n) = N$. Let σ be defined by $\sigma x_i = N_i$ for all $i \in \{1, \ldots, n\}$, $N'_i = x_i$ for all $i \in \{1, \ldots, n\}$, and $N' = h(x_1, \ldots, x_n)$. We have $h(N'_1, \ldots, N'_n) \to N'$ in $\mathrm{def}_{\Sigma}(h)$ because $h(x_1, \ldots, x_n) \to h(x_1, \ldots, x_n)$ is in $\mathrm{def}_{\Sigma}(h)$. We also have $\Sigma \vdash N_i = \sigma N'_i$ for all $i \in \{1, \ldots, n\}$ and $\Sigma \vdash N = h(N_1, \ldots, N_n) = \sigma N'$.

Case 2: h is a destructor in Σ. Then there exists $h(N'_1, \ldots, N'_n) \to N'$ in $\mathrm{def}_{\Sigma}(h)$, such that $\mathrm{addeval}(N'_1, \ldots, N'_n, N') \Downarrow' ((N_1, \ldots, N_n, N), \sigma)$. By Lemma 15, $\Sigma \vdash N = \sigma N'$ and for all $i \in \{1, \ldots, n\}$, $\Sigma \vdash N_i = \sigma N'_i$. □

Lemma 17 *Let D be a plain term evaluation. If $\Sigma \vdash D' = D$ and $D' \Downarrow_{\Sigma'} M'$ then $D \Downarrow_{\Sigma} M$ for some M such that $\Sigma \vdash M = M'$.*

Proof The proof is by induction on D.

- Case $D = M$: We have $D \Downarrow_{\Sigma} M$. Moreover $\Sigma \vdash D' = D$, so D' is also a term, and $M' = D'$. Finally, $D = M$, $D' = M'$, and $\Sigma \vdash D' = D$, so $\Sigma \vdash M = M'$.

- Case $D = \mathrm{eval}\, h(D_1, \ldots, D_n)$: Since $\Sigma \vdash D' = D$, we have $D' = \mathrm{eval}\, h(D'_1, \ldots, D'_n)$ with $\Sigma \vdash D'_i = D_i$. Since $D' \Downarrow_{\Sigma'} M'$, there exist $h(N_1, \ldots, N_n) \to N$ in $\mathrm{def}_{\Sigma'}(h)$ and σ such that $M' = \sigma N$, and for all $i \in \{1, \ldots, n\}$, $D'_i \Downarrow_{\Sigma'} \sigma N_i$. By induction hypothesis, $D_i \Downarrow_{\Sigma} M_i$ with $\Sigma \vdash M_i = \sigma N_i$.

  By Lemma 16, there exist $h(N'_1, \ldots, N'_n) \to N'$ in $\mathrm{def}_{\Sigma}(h)$ and σ', such that $\Sigma \vdash N = \sigma' N'$ and for all $i \in \{1, \ldots, n\}$, $\Sigma \vdash N_i = \sigma' N'_i$. Then $D_i \Downarrow_{\Sigma} M_i$, $\Sigma \vdash M_i = \sigma N_i = \sigma\sigma' N'_i$, and $h(N'_1, \ldots, N'_n) \to N'$ is in $\mathrm{def}_{\Sigma}(h)$, so $D \Downarrow_{\Sigma} \sigma\sigma' N'$. Moreover, $\Sigma \vdash M' = \sigma N = \sigma\sigma' N'$. □

The following lemma is useful to deal with rule (Red Fun 2): when D fails to evaluate, the lemma ensures that D' also fails to evaluate, even with the equational theory of Σ. To this end, Lemma 18 requires $D' \Downarrow_{\Sigma} M'$, whereas Lemma 17 requires $D' \Downarrow_{\Sigma'} M'$.

Lemma 18 *Let D be a plain term evaluation. If $\Sigma \vdash D' = D$ and $D' \Downarrow_{\Sigma} M'$ then $D \Downarrow_{\Sigma} M$ for some M such that $\Sigma \vdash M = M'$.*

Proof The proof is by induction on D.

- Case $D = M$: We have $D \Downarrow_{\Sigma} M$. Moreover $\Sigma \vdash D' = D$, so D' is also a term, and $M' = D'$. Finally, $D = M$, $D' = M'$, and $\Sigma \vdash D' = D$, so $\Sigma \vdash M = M'$.

- Case $D = \mathrm{eval}\, h(D_1, \ldots, D_n)$: Since $\Sigma \vdash D' = D$, we have $D' = \mathrm{eval}\, h(D'_1, \ldots, D'_n)$ with $\Sigma \vdash D'_i = D_i$. Since $D' \Downarrow_{\Sigma} M'$, there exist $h(N_1, \ldots, N_n) \to N$ in $\mathrm{def}_{\Sigma}(h)$ and σ such that $M' = \sigma N$, and for all $i \in \{1, \ldots, n\}$, $D'_i \Downarrow_{\Sigma} M'_i$ with $\Sigma \vdash M'_i = \sigma N_i$. By induction hypothesis, $D_i \Downarrow_{\Sigma} M_i$ with $\Sigma \vdash M_i = \sigma N_i$. Then $D = \mathrm{eval}\, h(D_1, \ldots, D_n) \Downarrow_{\Sigma} \sigma N$ and $\Sigma \vdash \sigma N = M'$. □

B.2 Proof of Lemma 1

Lemma 1 is an obvious consequence of the following lemma.

Lemma 19 *Let P_0 be a closed, unevaluated biprocess.*

If $P_0 \to^*_{\Sigma} \equiv P'_0$, $\Sigma \vdash Q'_0 = P'_0$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(\{Q'_0\})$, then $P_0 \to^*_{\Sigma',\Sigma} \equiv Q'_0$ by a reduction whose intermediate biprocesses Q all satisfy $\mathrm{nf}_{\mathcal{S},\Sigma}(\{Q\})$.

Conversely, if $P_0 \to^*_{\Sigma',\Sigma} \equiv Q'_0$ then there exists P'_0 such that $\Sigma \vdash Q'_0 = P'_0$ and $P_0 \to^*_{\Sigma} \equiv P'_0$.

Proof We write $VC(P)$ when P is a closed process whose terms M are either variables or terms of the form $\mathsf{diff}[M_1, M_2]$ where M_1 and M_2 are closed terms that do not contain diff. (Function symbols prefixed by eval are not constrained.) We have the following properties:

P1. If $VC(P)$ and $P \equiv P'$ then $VC(P')$. The proof is by induction on the derivation of $P \equiv P'$. All cases are easy, since $\equiv$ cannot change terms.

P2. If $VC(P)$ and $P \rightarrow_\Sigma P'$ then $VC(P')$. The proof is by induction on the derivation of $P \rightarrow_\Sigma P'$. The only change of terms is done by the substitution $\{M/x\}$ in the rules (Red I/O) and (Red Fun 1). This substitution replaces a variable with a closed term $M = \mathsf{diff}[M_1, M_2]$, hence the result. (For (Red I/O), M is of the form $\mathsf{diff}[M_1, M_2]$ because of $VC(P)$.)

P3. If $VC(P\{\mathsf{diff}[M_1, M_2]/x\})$, $\Sigma \vdash P\{\mathsf{diff}[M_1, M_2]/x\} = P''$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{P} \cup \{P''\})$, then there exist P', M_1', and M_2' such that $\Sigma \vdash P = P'$, $\Sigma \vdash M_1 = M_1'$, $\Sigma \vdash M_2 = M_2'$, $P'' = P'\{\mathsf{diff}[M_1', M_2']/x\}$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{P} \cup \{P', M_1', M_2'\})$.

Since P_0 is closed and unevaluated, $VC(P_0)$. Therefore, by P1 and P2, if $P_0 \rightarrow_\Sigma^* \equiv P$, then $VC(P)$. Moreover, the only process P such that $\Sigma \vdash P_0 = P$ and $\mathrm{nf}_{\mathcal{S},\Sigma}(\{P\})$ is P_0 by Lemma 4.

Let us show that, if $P \equiv P'$, $\Sigma \vdash Q' = P'$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{P} \cup \{Q'\})$, then there exists Q such that $\Sigma \vdash Q = P$, $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{P} \cup \{Q\})$, and $Q \equiv Q'$. The proof is by induction on the derivation of $P \equiv P'$. All cases are easy, since $\equiv$ does not depend on terms.

Let us show that, if $VC(P)$, $P \rightarrow_\Sigma P'$, $\Sigma \vdash Q' = P'$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{P} \cup \{Q'\})$, then there exists Q such that $\Sigma \vdash Q = P$, $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{P} \cup \{Q\})$, and $Q \rightarrow_{\Sigma',\Sigma} Q'$. The proof is by induction on the derivation of $P \rightarrow_\Sigma P'$.

- Case (Red I/O): Since $VC(P)$, we have $P = \overline{\mathsf{diff}[M_1, M_2]}\langle N\rangle.R \mid \mathsf{diff}[M_1', M_2'](x).R' \rightarrow_\Sigma R \mid R'\{N/x\} = P'$ with $\Sigma \vdash M_1 = M_1'$ and $\Sigma \vdash M_2 = M_2'$. Since $\Sigma \vdash Q' = P'$ and $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{P} \cup \{Q'\})$, we have $Q' = R_1 \mid R_1'\{N_1/x\}$ for some R_1, R_1', N_1 such that $\Sigma \vdash R_1 = R$, $\Sigma \vdash R_1' = R'$, $\Sigma \vdash N_1 = N$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{P} \cup \{R_1, R_1', N_1\})$ by P3.

  By Property S2, there exist M_1'' and M_2'' such that $\Sigma \vdash M_1'' = M_1 = M_1'$, $\Sigma \vdash M_2'' = M_2 = M_2'$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{P} \cup \{R_1, R_1', N_1, M_1'', M_2''\})$.

  We let $Q = \overline{\mathsf{diff}[M_1'', M_2'']}\langle N_1\rangle.R_1 \mid \mathsf{diff}[M_1'', M_2''](x).R_1'$. Then $\Sigma \vdash Q = P$. Moreover $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{P} \cup \{Q\})$ since $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{P} \cup \{R_1, R_1', N_1, M_1'', M_2''\})$, and $Q \rightarrow_{\Sigma',\Sigma} Q'$, hence the result.

- Case (Red Fun 1): We have $P = \mathit{let}\ x = D\ \mathit{in}\ R\ \mathit{else}\ R' \rightarrow_\Sigma R\{\mathsf{diff}[M, M']/x\} = P'$ with $\mathsf{fst}(D) \Downarrow_\Sigma M$ and $\mathsf{snd}(D) \Downarrow_\Sigma M'$. Since $\Sigma \vdash Q' = P'$ and $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{P} \cup \{Q'\})$, we have $Q' = R_1\{\mathsf{diff}[M_1, M_1']/x\}$ for some R_1, M_1, M_1' such that $\Sigma \vdash R_1 = R$, $\Sigma \vdash M_1 = M$, $\Sigma \vdash M_1' = M'$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{P} \cup \{R_1, M_1, M_1'\})$ by P3.

  By Property S2, there exist D_1 and R_1' such that $\Sigma \vdash D_1 = D$, $\Sigma \vdash R_1' = R'$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{P} \cup \{D_1, R_1', R_1, M_1, M_1'\})$. By Lemma 14, $\mathsf{fst}(D_1) \Downarrow_{\Sigma'} M_1$ and $\mathsf{snd}(D_1) \Downarrow_{\Sigma'} M_1'$. Let $Q = \mathit{let}\ x = D_1\ \mathit{in}\ R_1\ \mathit{else}\ R_1'$. Then $\Sigma \vdash Q = P$, $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{P} \cup \{Q\})$, and $Q \rightarrow_{\Sigma',\Sigma} Q'$.

- Case (Red Fun 2): We have $P = \mathit{let}\ x = D\ \mathit{in}\ R\ \mathit{else}\ P' \rightarrow_\Sigma P'$, there exists no M such that $\mathsf{fst}(D) \Downarrow_\Sigma M$, and there exists no M' such that $\mathsf{snd}(D) \Downarrow_\Sigma M'$. We have $\Sigma \vdash Q' = P'$ and $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{P} \cup \{Q'\})$.

  By Property S2, there exist D_1 and R_1 such that $\Sigma \vdash D_1 = D$, $\Sigma \vdash R_1 = R$ and $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{P} \cup \{R_1, D_1, Q'\})$. Then, there exists no M such that $\mathsf{fst}(D_1) \Downarrow_\Sigma M$,

and there exists no M' such that $\mathsf{snd}(D_1) \Downarrow_\Sigma M'$. (Otherwise, by Lemma 18, there would exist M such that $\mathsf{fst}(D) \Downarrow_\Sigma M$, and M' such that $\mathsf{snd}(D) \Downarrow_\Sigma M'$.) Let $Q = \mathit{let}\ x = D_1\ \mathit{in}\ R_1\ \mathit{else}\ Q'$. Then $\Sigma \vdash Q = P$, $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{P} \cup \{Q\})$, and $Q \rightarrow_{\Sigma',\Sigma} Q'$.

- Case (Red Repl): We have $P = !R \rightarrow_\Sigma R \mid !R = P'$. Since $\Sigma \vdash Q' = P'$ and $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{P} \cup \{Q'\})$, we have $Q' = R_1 \mid !R_1$ for some R_1 such that $\Sigma \vdash R_1 = R$. Let $Q = !R_1$. Then we have $\Sigma \vdash Q = P$, $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{P} \cup \{Q\})$, and $Q \rightarrow_{\Sigma',\Sigma} Q'$.

- Cases (Red Par) and (Red Res): Easy by induction hypothesis.

- Case (Red $\equiv$): Easy using the corresponding property for $\equiv$ and the induction hypothesis.

Therefore, if $P_0 \rightarrow^*_\Sigma\equiv\ P'_0$, $\Sigma \vdash Q'_0 = P'_0$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(\{Q'_0\})$, then there exists Q_0 such that $\mathrm{nf}_{\mathcal{S},\Sigma}(\{Q_0\})$, $\Sigma \vdash Q_0 = P_0$, and $Q_0 \rightarrow^*_{\Sigma',\Sigma}\equiv\ Q'_0$ by a reduction whose intermediate biprocesses Q all satisfy $\mathrm{nf}_{\mathcal{S},\Sigma}(\{Q\})$, simply by applying several times the results shown above. Since the only process P such that $\Sigma \vdash P_0 = P$ and $\mathrm{nf}_{\mathcal{S},\Sigma}(\{P\})$ is P_0, we have $Q_0 = P_0$, so we conclude that if $P_0 \rightarrow^*_\Sigma\equiv\ P'_0$, $\Sigma \vdash Q'_0 = P'_0$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(\{Q'_0\})$, then $P_0 \rightarrow^*_{\Sigma',\Sigma}\equiv\ Q'_0$ by a reduction whose intermediate biprocesses Q all satisfy $\mathrm{nf}_{\mathcal{S},\Sigma}(\{Q\})$.

For the converse, we show that, if $P \equiv P'$ and $\Sigma \vdash Q = P$, then there exists Q' such that $\Sigma \vdash Q' = P'$ and $Q \equiv Q'$. The proof is by induction on the derivation of $P \equiv P'$. All cases are easy, since $\equiv$ does not depend on terms.

We also show that, if $VC(P)$, $P \rightarrow_{\Sigma',\Sigma} P'$ and $\Sigma \vdash Q = P$, then there exists Q' such that $\Sigma \vdash Q' = P'$, and $Q \rightarrow_\Sigma Q'$. The proof is by induction on the derivation of $P \rightarrow_{\Sigma',\Sigma} P'$.

- Case (Red I/O): Since $VC(P)$, we have $P = \overline{\mathsf{diff}[M_1, M_2]}\langle N\rangle.R \mid \mathsf{diff}[M_1, M_2](x).R' \rightarrow_{\Sigma',\Sigma} R \mid R'\{N/x\} = P'$. Since $\Sigma \vdash Q = P$, we have $Q = \overline{\mathsf{diff}[M'_1, M'_2]}\langle N'\rangle.R_1 \mid \mathsf{diff}[M''_1, M''_2](x).R'_1$ with $\Sigma \vdash M_1 = M'_1 = M''_1$, $\Sigma \vdash M_2 = M'_2 = M''_2$, $\Sigma \vdash N' = N$, $\Sigma \vdash R = R_1$, and $\Sigma \vdash R' = R'_1$. Then $Q \rightarrow_\Sigma Q' = R_1 \mid R'_1\{N'/x\}$ with $\Sigma \vdash Q' = P'$.

- Case (Red Fun 1): We have $P = \mathit{let}\ x = D\ \mathit{in}\ R\ \mathit{else}\ R' \rightarrow_{\Sigma',\Sigma} R\{\mathsf{diff}[M_1, M_2]/x\} = P'$ with $\mathsf{fst}(D) \Downarrow_{\Sigma'} M_1$ and $\mathsf{snd}(D) \Downarrow_{\Sigma'} M_2$. Since $\Sigma \vdash Q = P$, we have $Q = \mathit{let}\ x = D'\ \mathit{in}\ R_1\ \mathit{else}\ R'_1$ with $\Sigma \vdash D' = D$, $\Sigma \vdash R_1 = R$, and $\Sigma \vdash R'_1 = R'$. By Lemma 17, $\mathsf{fst}(D') \Downarrow_\Sigma M'_1$ with $\Sigma \vdash M_1 = M'_1$ and $\mathsf{snd}(D') \Downarrow_\Sigma M'_2$ with $\Sigma \vdash M_2 = M'_2$. Hence $Q \rightarrow_\Sigma Q' = C'[R_1\{\mathsf{diff}[M'_1, M'_2]/x\}]$ with $\Sigma \vdash Q' = P'$.

- Case (Red Fun 2): We have $P = \mathit{let}\ x = D\ \mathit{in}\ R\ \mathit{else}\ P' \rightarrow_{\Sigma',\Sigma} P'$, there exists no M_1 such that $\mathsf{fst}(D) \Downarrow_\Sigma M_1$, and there exists no M_2 such that $\mathsf{snd}(D) \Downarrow_\Sigma M_2$. Since $\Sigma \vdash Q = P$, we have $Q = \mathit{let}\ x = D'\ \mathit{in}\ R_1\ \mathit{else}\ Q'$ with $\Sigma \vdash D' = D$, $\Sigma \vdash R_1 = R$, and $\Sigma \vdash Q' = P'$. Then, there exists no M'_1 such that $\mathsf{fst}(D') \Downarrow_\Sigma M'_1$, and there exists no M'_2 such that $\mathsf{snd}(D') \Downarrow_\Sigma M'_2$. (Otherwise, by Lemma 18, there would exist M_1 such that $\mathsf{fst}(D) \Downarrow_\Sigma M_1$ and M_2 such that $\mathsf{snd}(D) \Downarrow_\Sigma M_2$.) Hence $Q \rightarrow_\Sigma Q'$ and $\Sigma \vdash Q' = P'$.

- Case (Red Repl): We have $P = !R \rightarrow_{\Sigma',\Sigma} R \mid !R = P'$. Since $\Sigma \vdash Q = P$, we have $Q = !R_1$ with $\Sigma \vdash R_1 = R$. Let $Q' = R_1 \mid !R_1$. So $\Sigma \vdash Q' = P'$ and $Q \rightarrow_\Sigma Q'$.
- Cases (Red Par) and (Red Res): Easy by induction hypothesis.
- Case (Red $\equiv$): Easy using the corresponding property for $\equiv$ and the induction hypothesis.

We conclude that, if $P_0 \rightarrow^*_{\Sigma',\Sigma} \equiv Q'_0$ then there exists P'_0 such that $\Sigma \vdash Q'_0 = P'_0$ and $P_0 \rightarrow^*_\Sigma \equiv P'_0$, simply by applying several times the results shown above, with $Q = P$ in the first application. □

B.3 Proof of Lemma 2

We first show that it is enough to consider unevaluated processes as initial configurations (Lemma 22), then prove Lemma 2 itself.

Let $P \mathcal{R} P'$ if and only if P' is obtained from P by adding some lets on terms with constructors that occur in inputs or outputs (for instance transforming $\overline{M}\langle N\rangle.P$ into $let\ x = M\ in\ let\ y = N\ in\ \overline{x}\langle y\rangle.P$ where x and y are fresh variables), prefixing some constructors in lets with eval, and replacing some terms M with $\mathsf{diff}[\mathsf{fst}(M), \mathsf{snd}(M)]$.

For the next two proofs, we consider an alternative, equivalent definition of $\equiv$, in which a symmetric rule $Q \equiv P$ is added for each rule $P \equiv Q$ in the definition of $\equiv$ and the implication $P \equiv Q \Rightarrow Q \equiv P$ is removed from the definition of $\equiv$.

Lemma 20 *If $P \mathcal{R} Q$ and $P \equiv P'$ then there exists Q' such that $P' \mathcal{R} Q'$ and $Q \equiv Q'$.*
If $P \mathcal{R} Q$ and $P \rightarrow_\Sigma P'$ then there exists Q' such that $P' \mathcal{R} Q'$ and $Q \rightarrow^+_\Sigma Q'$.

Proof Obvious, by induction on the derivation of $P \equiv P'$ and $P \rightarrow_\Sigma P'$ respectively. □

Lemma 21 *If $\Sigma \vdash P = Q$, $Q \mathcal{R} R$, and $R \equiv R'$ then there exists P' and Q' such that $\Sigma \vdash P' = Q'$, $Q' \mathcal{R} R'$, and $P \equiv P'$.*
If $\Sigma \vdash P = Q$, $Q \mathcal{R} R$, and $R \rightarrow_\Sigma R'$ then there exists P' and Q' such that $\Sigma \vdash P' = Q'$, $Q' \mathcal{R} R'$, and $P \rightarrow_\Sigma P'$ or $P = P'$.

Proof Obvious, by induction on the derivation of $R \equiv R'$ and $R \rightarrow_\Sigma R'$ respectively. □

Lemma 22 *Let P_0 be a closed biprocess. The hypotheses of Corollary 1 are true if and only if they are true with* $\mathrm{unevaluated}(C[P_0])$ *instead of $C[P_0]$.*

Proof We have $C[P_0]\ \mathcal{R}\ \mathrm{unevaluated}(C[P_0])$. We first show that if the hypotheses of Corollary 1 are true for $\mathrm{unevaluated}(C[P_0])$, then they are true for $C[P_0]$.

- If $C[P_0] \rightarrow_\Sigma^* \equiv C_1'[\overline{N_1}\langle M_1\rangle.Q_1 \mid N_1'(x).R_1]$, then by Lemma 20, we have $\mathrm{unevaluated}(C[P_0]) \rightarrow_\Sigma^* \equiv P'$ with $C_1'[\overline{N_1}\langle M_1\rangle.Q_1 \mid N_1'(x).R_1] \mathcal{R} P'$. Then we have $P' \rightarrow_\Sigma^* C'[\overline{N}\langle M\rangle.Q \mid N'(x).R]$ with $C_1' \mathcal{R} C'$, $\mathsf{fst}(N) = \mathsf{fst}(N_1)$, $\mathsf{snd}(N) = \mathsf{snd}(N_1)$, $\mathsf{fst}(N') = \mathsf{fst}(N_1')$, $\mathsf{snd}(N') = \mathsf{snd}(N_1')$, $\mathsf{fst}(M) = \mathsf{fst}(M_1)$, $\mathsf{snd}(M) = \mathsf{snd}(M_1)$, $Q_1 \mathcal{R} Q$, and $R_1 \mathcal{R} R$, by reducing the term evaluations of constructors that may occur above inputs and outputs in P'. So $\mathrm{unevaluated}(C[P_0]) \rightarrow_\Sigma^* \equiv C'[\overline{N}\langle M\rangle.Q \mid N'(x).R]$, with $\mathsf{fst}(N) = \mathsf{fst}(N_1)$, $\mathsf{snd}(N) = \mathsf{snd}(N_1)$, $\mathsf{fst}(N') = \mathsf{fst}(N_1')$, and $\mathsf{snd}(N') = \mathsf{snd}(N_1')$. Hence, if the first hypothesis of Corollary 1 is true with $\mathrm{unevaluated}(C[P_0])$, then it is true with $C[P_0]$.

- If $C[P_0] \rightarrow_\Sigma^* \equiv C_1'[\mathit{let}\ y_1 = D_1\ \mathit{in}\ Q_1\ \mathit{else}\ R_1]$, then by the same reasoning as above, $\mathrm{unevaluated}(C[P_0]) \rightarrow_\Sigma^* \equiv C'[P]$ where $\mathit{let}\ y_1 = D_1\ \mathit{in}\ Q_1\ \mathit{else}\ R_1\ \mathcal{R}\ P$. Hence, we have $P = \mathit{let}\ y_1 = D_1'\ \mathit{in}\ Q_1'\ \mathit{else}\ R_1'$ where D_1' is obtained by prefixing some constructors of D_1 with eval and reorganizing diffs. We have $\mathsf{fst}(D_1) \Downarrow_\Sigma M_1$ if and only if $\mathsf{fst}(D_1') \Downarrow_\Sigma M_1$, if and only if $\mathsf{snd}(D_1') \Downarrow_\Sigma M_2$ (by the second hypothesis of Corollary 1 for $\mathrm{unevaluated}(C[P_0])$), if and only if $\mathsf{snd}(D_1) \Downarrow_\Sigma M_2$. This yields the second hypothesis of Corollary 1 for $C[P_0]$.

We now show the converse: if the hypotheses of Corollary 1 are true for $C[P_0]$, then they are true for $\mathrm{unevaluated}(C[P_0])$.

- Assume that $\mathrm{unevaluated}(C[P_0]) \rightarrow_\Sigma^* \equiv C_1'[\overline{N_1}\langle M_1\rangle.Q_1 \mid N_1'(x).R_1]$. By Lemma 21, $C[P_0] \rightarrow_\Sigma^* \equiv P$ with $\Sigma \vdash P = P'$ and $P' \mathcal{R} C_1'[\overline{N_1}\langle M_1\rangle.Q_1 \mid N_1'(x).R_1]$. Then $P = C'[\overline{N}\langle M\rangle.Q \mid N'(x).R]$ with $\Sigma \vdash \mathsf{fst}(N) = \mathsf{fst}(N_1)$, $\Sigma \vdash \mathsf{snd}(N) = \mathsf{snd}(N_1)$, $\Sigma \vdash \mathsf{fst}(N') = \mathsf{fst}(N_1')$, $\Sigma \vdash \mathsf{snd}(N') = \mathsf{snd}(N_1')$, $\Sigma \vdash \mathsf{fst}(M) = \mathsf{fst}(M_1)$, and $\Sigma \vdash \mathsf{snd}(M) = \mathsf{snd}(M_1)$. So, if the first hypothesis of Corollary 1 is true with $C[P_0]$, then it is true with $\mathrm{unevaluated}(C[P_0])$.

- Assume that $\mathrm{unevaluated}(C[P_0]) \rightarrow_\Sigma^* \equiv C_1'[\mathit{let}\ y_1 = D_1\ \mathit{in}\ Q_1\ \mathit{else}\ R_1]$. By Lemma 21, $C[P_0] \rightarrow_\Sigma^* \equiv P$ with $\Sigma \vdash P = P'$ and $P' \mathcal{R} C_1'[\mathit{let}\ y_1 = D_1\ \mathit{in}\ Q_1\ \mathit{else}\ R_1]$. We have two cases:

  - Case 1: $\mathit{let}\ y_1$ is introduced by $\mathcal{R}$. Then $R_1 = 0$ and D_1 does not contain destructors. Hence there exists M_1 such that $\mathsf{fst}(D_1) \Downarrow_\Sigma M_1$ and there exists M_2 such that $\mathsf{snd}(D_1) \Downarrow_\Sigma M_2$.
  - Case 2: $\mathit{let}\ y_1$ comes from P'. Hence $P = C'[\mathit{let}\ y_1 = D_1'\ \mathit{in}\ Q_1'\ \mathit{else}\ R_1']$ where D_1' is obtained by removing some eval prefixes of D_1, reorganizing diffs, and replacing terms with equal terms modulo Σ. We have $\mathsf{fst}(D_1) \Downarrow_\Sigma M_1$ for some M_1 if and only if $\mathsf{fst}(D_1') \Downarrow_\Sigma M_1$ for some M_1, if and only if $\mathsf{snd}(D_1') \Downarrow_\Sigma M_2$ for some M_2 (by the second hypothesis of Corollary 1 for $C[P_0]$), if and only if $\mathsf{snd}(D_1) \Downarrow_\Sigma M_2$ for some M_2.

  This yields the second hypothesis of Corollary 1 for $\mathrm{unevaluated}(C[P_0])$. □

Lemma 2 is an obvious consequence of the following lemma.

Lemma 23 *Let P_0 be a closed biprocess. Suppose that, for all plain evaluation contexts C, all evaluation contexts C', and all reductions* $\mathrm{unevaluated}(C[P_0]) \rightarrow^*_{\Sigma',\Sigma} P$ *whose intermediate biprocesses P' all satisfy* $\mathrm{nf}_{\mathcal{S},\Sigma}(\{P'\})$*,*

1. *if $P \equiv C'[\overline{N}\langle M\rangle.Q \mid N'(x).R]$ and $\mathsf{fst}(N) = \mathsf{fst}(N')$, then $\Sigma \vdash \mathsf{snd}(N) = \mathsf{snd}(N')$,*
2. *if $P \equiv C'[let\ x = D\ in\ Q\ else\ R]$ and $\mathsf{fst}(D) \Downarrow_{\Sigma'} M_1$ for some M_1, then $\mathsf{snd}(D) \Downarrow_{\Sigma} M_2$ for some M_2,*

as well as the symmetric properties where we swap fst *and* snd*. Then P_0 satisfies the hypotheses of Corollary 1.*

Conversely, if P_0 satisfies the hypotheses of Corollary 1, then for all plain evaluation contexts C, evaluation contexts C', and reductions $\mathrm{unevaluated}(C[P_0]) \rightarrow^*_{\Sigma'} P$*, we have properties 1 and 2 above, as well as the symmetric properties where we swap* fst *and* snd*.*

Proof By Lemma 22, we can work with $\mathrm{unevaluated}(C[P_0])$ instead of $C[P_0]$. We show the two hypotheses of Corollary 1.

- Assume that $\mathrm{unevaluated}(C[P_0]) \rightarrow^*_{\Sigma} \equiv C'[\overline{N}\langle M\rangle.Q \mid N'(x).R]$ and $\Sigma \vdash \mathsf{fst}(N) = \mathsf{fst}(N')$. By Property S2, there exists P' such that $\Sigma \vdash P' = C'[\overline{N}\langle M\rangle.Q \mid N'(x).R]$ and $\mathrm{nf}_{\mathcal{S},\Sigma}(\{P'\})$. By Lemma 19, $\mathrm{unevaluated}(C[P_0]) \rightarrow^*_{\Sigma',\Sigma} \equiv P'$. Moreover, $P' = C''[\overline{\mathsf{diff}[N_1, N_2]}\langle M'\rangle.Q_1 \mid \mathsf{diff}[N_1', N_2'](x).R_1]$, where $\Sigma \vdash N_1 = \mathsf{fst}(N)$, $\Sigma \vdash N_2 = \mathsf{snd}(N)$, $\Sigma \vdash N_1' = \mathsf{fst}(N')$, and $\Sigma \vdash N_2' = \mathsf{snd}(N')$. Since $\mathrm{nf}_{\mathcal{S},\Sigma}(\{P'\})$, $N_1 = N_1'$. Hence, by hypothesis 1, $\Sigma \vdash N_2 = N_2'$. So $\Sigma \vdash \mathsf{snd}(N) = \mathsf{snd}(N')$.

  We obtain the case $\mathrm{unevaluated}(C[P_0]) \rightarrow^* \equiv C'[\overline{N}\langle M\rangle.Q \mid N'(x).R]$ and $\Sigma \vdash \mathsf{snd}(N) = \mathsf{snd}(N')$ by symmetry.

- Assume that $\mathrm{unevaluated}(C[P_0]) \rightarrow^*_{\Sigma} \equiv C'[let\ y = D\ in\ Q\ else\ R]$ and there exists M_1 such that $\mathsf{fst}(D) \Downarrow_{\Sigma} M_1$. By Property S2, there exist P', M_1', and D' such that $\Sigma \vdash P' = C'[let\ y = D\ in\ Q\ else\ R]$, $\Sigma \vdash M_1' = M_1$, $\Sigma \vdash D' = D$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(\{P', M_1', D'\})$. Then $P' = C''[let\ y = D'\ in\ Q'\ else\ R']$. By Lemma 19, $\mathrm{unevaluated}(C[P_0]) \rightarrow^*_{\Sigma',\Sigma} \equiv P'$. By Lemma 14, $\mathsf{fst}(D') \Downarrow_{\Sigma'} M_1'$. By hypothesis 2, $\mathsf{snd}(D') \Downarrow_{\Sigma} M_2'$. By Lemma 18, since $\Sigma \vdash \mathsf{snd}(D') = \mathsf{snd}(D)$ and $\mathsf{snd}(D') \Downarrow_{\Sigma} M_2'$, we have $\mathsf{snd}(D) \Downarrow_{\Sigma} M_2$.

  We obtain the case $\mathrm{unevaluated}(C[P_0]) \rightarrow^* \equiv C'[let\ y = D\ in\ Q\ else\ R]$ and there exists M_2 such that $\mathsf{snd}(D) \Downarrow_{\Sigma} M_2$ by symmetry.

Next, we prove the converse property.

- Assume that $\mathrm{unevaluated}(C[P_0]) \rightarrow^*_{\Sigma',\Sigma} \equiv C'[\overline{N}\langle M\rangle.Q \mid N'(x).R]$ and $\mathsf{fst}(N) = \mathsf{fst}(N')$. By Lemma 19, we have $\mathrm{unevaluated}(C[P_0]) \rightarrow^*_{\Sigma} \equiv C_1[\overline{N_1}\langle M_1\rangle.Q_1 \mid N_1(x).R_1]$ with $\Sigma \vdash C'[\overline{N}\langle M\rangle.Q \mid N'(x).R] = C_1[\overline{N_1}\langle M_1\rangle.Q_1 \mid N_1(x).R_1]$ so $\Sigma \vdash N = N_1$ and $\Sigma \vdash N' = N_1'$. Using the first hypothesis of Corollary 1, since $\Sigma \vdash \mathsf{fst}(N_1) = \mathsf{fst}(N_1')$, we have $\Sigma \vdash \mathsf{snd}(N_1) = \mathsf{snd}(N_1')$, hence $\Sigma \vdash \mathsf{snd}(N) = \mathsf{snd}(N')$.

We obtain the case $\mathrm{unevaluated}(C[P_0]) \rightarrow^*_{\Sigma',\Sigma} \equiv C'[\overline{N}\langle M\rangle.Q \mid N'(x).R]$ and $\mathsf{snd}(N) = \mathsf{snd}(N')$ by symmetry.

- Assume that $\mathrm{unevaluated}(C[P_0]) \rightarrow^*_{\Sigma',\Sigma} \equiv C'[let\ y = D\ in\ Q\ else\ R]$ and there exists M_1 such that $\mathsf{fst}(D) \Downarrow_{\Sigma'} M_1$. As above, $\mathrm{unevaluated}(C[P_0]) \rightarrow^*_{\Sigma} \equiv C_1[let\ y = D_1\ in\ Q_1\ else\ R_1]$ with $\Sigma \vdash D_1 = D$. By Lemma 17, $\mathsf{fst}(D_1) \Downarrow_{\Sigma} M_1'$ for some M_1'. Using the second hypothesis of Corollary 1, $\mathsf{snd}(D_1) \Downarrow_{\Sigma} M_2'$, hence by Lemma 18, $\mathsf{snd}(D) \Downarrow_{\Sigma} M_2$.

  We obtain the case $\mathrm{unevaluated}(C[P_0]) \rightarrow^*_{\Sigma',\Sigma} \equiv C'[let\ y = D\ in\ Q\ else\ R]$ and there exists M_2 such that $\mathsf{snd}(D) \Downarrow_{\Sigma'} M_2$ by symmetry. □

C Proof of Lemma 3

When $\mathcal{F}$ is a set that contains patterns, facts, sequences of patterns or facts, clauses, environments that map variables and names to pairs of patterns, ..., we say that $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{F})$ if and only if all patterns that appear in $\mathcal{F}$ are irreducible by $\mathcal{S}$ and for all p_1, p_2 subpatterns of elements of $\mathcal{F}$, if $\Sigma \vdash p_1 = p_2$ then $p_1 = p_2$.

We say that $\mathrm{nf}'_{\mathcal{S},\Sigma}(\mathcal{F})$ if and only if $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{F}')$ where $\mathcal{F}'$ is obtained from $\mathcal{F}$ by removing nounif facts. When $\mathcal{D}$ is a derivation, we say that $\mathrm{nf}'_{\mathcal{S},\Sigma}(\mathcal{D})$ when $\mathrm{nf}'_{\mathcal{S},\Sigma}(\mathcal{F})$ where $\mathcal{F}$ is the set of intermediately derived facts of $\mathcal{D}$.

We say that $F_1 \wedge \cdots \wedge F_n \sim F_1' \wedge \cdots \wedge F_n'$ when, for all $i \in \{1, \ldots, n\}$, either $F_i = F_i'$ or F_i and F_i' are nounif facts and $\Sigma \vdash F_i = F_i'$. We say that $\Sigma \vdash F_1 \wedge \cdots \wedge F_n \sim F_1' \wedge \cdots \wedge F_n'$ when for all $i \in \{1, \ldots, n\}$, $\Sigma \vdash F_i = F_i'$. This definition is naturally extended to clauses.

The special treatment of nounif facts in the definition of $\sim$ and in Lemma 3 is necessary so that the following results hold. In particular, Lemma 28 would be wrong for Clauses (Rt) and (Rt′), which contain nounif facts.

Lemma 24 *If $h(N_1, \ldots, N_n) \rightarrow N$ is in $\mathrm{def}_{\Sigma'}(h)$, $\Sigma \vdash N'' = \sigma N$, $\Sigma \vdash N_i'' = \sigma N_i$ for all $i \in \{1, \ldots, n\}$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(\{N_1'', \ldots, N_n'', N''\})$, then there exist a closed substitution σ' and $h(N_1', \ldots, N_n') \rightarrow N'$ in $\mathrm{def}_{\Sigma'}(h)$ such that $N'' = \sigma' N'$ and $N_i'' = \sigma' N_i'$ for all $i \in \{1, \ldots, n\}$.*

Proof The result follows from Lemmas 16 and 13. □

The following lemma generalizes Lemma 15 to the case in which D may contain destructors. It is used in the proof of Lemma 26 below.

Lemma 25 *Let D be a plain term evaluation. If $D \Downarrow' (p, \sigma)$ and σ' is a closed substitution, then there exists p' such that $\sigma'\sigma D \Downarrow_{\Sigma} p'$ and $\Sigma \vdash p' = \sigma' p$.*

Let $D_1, \ldots, D_n$ be plain term evaluations. If $(D_1, \ldots, D_n) \Downarrow' ((p_1, \ldots, p_n), \sigma)$ and σ' is a closed substitution then there exist $p_1', \ldots, p_n'$ such that for all $i \in \{1, \ldots, n\}$, $\sigma'\sigma D_i \Downarrow_{\Sigma} p_i'$ and $\Sigma \vdash p_i' = \sigma' p_i$.

Proof The proof is by mutual induction following the definition of $\Downarrow'$.

- Case $D = p$: We have $p \Downarrow' (p, \emptyset)$, $\sigma = \emptyset$, so $\sigma'\sigma D = \sigma' p \Downarrow_\Sigma \sigma' p$, so we have the result with $p' = \sigma' p$.

- Case $D = \mathrm{eval}\, h(D_1, \ldots, D_n)$: Since $\mathrm{eval}\, h(D_1, \ldots, D_n) \Downarrow' (p, \sigma)$, there exist $h(N_1, \ldots, N_n) \to N$ in $\mathrm{def}_{\Sigma'}(h)$, $p_1, \ldots, p_n$, σ'', and σ_u such that $(D_1, \ldots, D_n) \Downarrow' ((p_1, \ldots, p_n), \sigma'')$, σ_u is a most general unifier of $(p_1, \ldots, p_n)$ and $(N_1, \ldots, N_n)$, $p = \sigma_u N$, and $\sigma = \sigma_u \sigma''$. By induction hypothesis, there exist $p'_1, \ldots, p'_n$ such that for all $i \in \{1, \ldots, n\}$, $\sigma'\sigma_u\sigma'' D_i \Downarrow_\Sigma p'_i$ and $\Sigma \vdash p'_i = \sigma'\sigma_u p_i$, so $\sigma'\sigma D_i \Downarrow_\Sigma p'_i$. By Lemma 16, there exist $h(N'_1, \ldots, N'_n) \to N'$ in $\mathrm{def}_\Sigma(h)$ and σ_1 such that $\Sigma \vdash N_i = \sigma_1 N'_i$ for all $i \in \{1, \ldots, n\}$ and $\Sigma \vdash N = \sigma_1 N'$. So $\Sigma \vdash p'_i = \sigma'\sigma_u p_i = \sigma'\sigma_u N_i = \sigma'\sigma_u\sigma_1 N'_i$ and $\Sigma \vdash \sigma' p = \sigma'\sigma_u N = \sigma'\sigma_u\sigma_1 N'$. Let $p' = \sigma'\sigma_u\sigma_1 N'$. We have $\sigma'\sigma D \Downarrow_\Sigma p'$ and $\Sigma \vdash p' = \sigma' p$.

- Case $(D_1, \ldots, D_n)$: Since $(D_1, \ldots, D_n) \Downarrow' ((p_1, \ldots, p_n), \sigma)$, we have $(D_1, \ldots, D_{n-1}) \Downarrow' ((p''_1, \ldots, p''_{n-1}), \sigma_1)$, $\sigma_1 D_n \Downarrow' (p_n, \sigma_2)$, $p_i = \sigma_2 p''_i$ for all $i \in \{1, \ldots, n-1\}$, and $\sigma = \sigma_2\sigma_1$. By induction hypothesis, there exist $p'_1, \ldots, p'_{n-1}$ such that for all $i \in \{1, \ldots, n-1\}$, $\sigma'\sigma_2\sigma_1 D_i \Downarrow_{\Sigma'} p'_i$ and $\Sigma \vdash p'_i = \sigma'\sigma_2 p''_i$, so $\sigma'\sigma D_i \Downarrow_{\Sigma'} p'_i$ and $\Sigma \vdash p'_i = \sigma' p_i$. Also by induction hypothesis, there exists p'_n such that $\sigma'\sigma_2\sigma_1 D_n \Downarrow_{\Sigma'} p'_n$ and $\Sigma \vdash p'_n = \sigma' p_n$, so $\sigma'\sigma D_n \Downarrow_{\Sigma'} p'_n$ and $\Sigma \vdash p'_n = \sigma' p_n$. □

Lemma 26 *Let D be a plain term evaluation such that the subterms M of D are variables or names. If $\rho(D) \Downarrow' (p', \sigma')$, σ is a closed substitution, $\Sigma \vdash p = \sigma p'$, $\Sigma \vdash \sigma'_0\rho' = \sigma\sigma'\rho$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(\{p, \sigma'_0\rho'\})$, then there exist σ'', p'', σ''_0 such that $\rho'(D) \Downarrow' (p'', \sigma'')$, $\sigma'_0 = \sigma''_0\sigma''$ except on fresh variables introduced in the computation of $\rho'(D) \Downarrow' (p'', \sigma'')$, and $p = \sigma''_0 p''$.*

Let D_i ($i \in \{1, \ldots, n\}$) be plain term evaluations such that the subterms M of D_i are variables or names. If $(\rho(D_1), \ldots, \rho(D_n)) \Downarrow' ((p'_1, \ldots, p'_n), \sigma')$, σ is a closed substitution, $\Sigma \vdash p_i = \sigma p'_i$ for all $i \in \{1, \ldots, n\}$, $\Sigma \vdash \sigma'_0\rho' = \sigma\sigma'\rho$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(\{p_1, \ldots, p_n, \sigma'_0\rho'\})$, then there exist σ'', $p''_1, \ldots, p''_n$, σ''_0 such that $(\rho'(D_1), \ldots, \rho'(D_n)) \Downarrow' ((p''_1, \ldots, p''_n), \sigma'')$, $\sigma'_0 = \sigma''_0\sigma''$ except on fresh variables introduced in the computation of $(\rho'(D_1), \ldots, \rho'(D_n)) \Downarrow' ((p''_1, \ldots, p''_n), \sigma'')$, and $p_i = \sigma''_0 p''_i$ for all $i \in \{1, \ldots, n\}$.

Proof We prove the first property. (The second one follows in a similar way.) By Lemma 25, there exists p_1 such that $\sigma\sigma'\rho(D) \Downarrow_\Sigma p_1$ and $\Sigma \vdash p_1 = \sigma p'$. Then $\Sigma \vdash p = p_1$, $\Sigma \vdash \sigma'_0\rho'(D) = \sigma\sigma'\rho(D)$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(\{p, \sigma'_0\rho'(D)\})$. So by a variant of Lemma 14 for patterns instead of terms, $\sigma'_0\rho'(D) \Downarrow_{\Sigma'} p$. By a variant of Lemma 11 for patterns instead of terms, we obtain the desired result. □

Lemma 27 *Let P_0 be a closed, unevaluated process. If $[\![P]\!]\rho s s' H$ is called during the generation of $[\![P_0]\!]\rho_0\emptyset\emptyset\emptyset$, σ is a closed substitution, $\Sigma \vdash \rho_2 = \sigma\rho$, $\Sigma \vdash s_2 = \sigma s$, $\Sigma \vdash s'_2 = \sigma s'$, $\Sigma \vdash H_2 \sim \sigma H$, and $\mathrm{nf}'_{\mathcal{S},\Sigma}(\{\rho_2, s_2, s'_2, H_2\})$, then there exist σ_1, ρ_1, H_1, s_1, s'_1 such that $\rho_2 = \sigma_1\rho_1$, $s_2 = \sigma_1 s_1$, $s'_2 = \sigma_1 s'_1$, $H_2 \sim \sigma_1 H_1$, and $[\![P]\!]\rho_1 s_1 s'_1 H_1$ is called during the generation of $[\![P_0]\!]\rho_0\emptyset\emptyset\emptyset$.*

Proof The process P is a subprocess of P_0. We proceed by induction on P: we show the result for P_0 itself, and we show that if the result is true for some occurrence of P, then it is also true for the occurrences of the direct subprocesses of P.

- Case P_0: We have $\rho_2 = \rho_0$, $s_2 = s'_2 = \emptyset$, and $H_2 = \emptyset$. Then we obtain the result by letting σ_1 be any substitution, $\rho_1 = \rho_0$, $s_1 = s'_1 = \emptyset$, and $H_1 = \emptyset$.

- Case 0: Void, since it has no subprocesses.

- Case $P \mid Q$: Obvious by induction hypothesis.

- Case $!P$: Assume $[\![P]\!]\rho s s' H$ is called. Then $\rho = \rho_3$, $s = (s_3, i)$, $s' = (s'_3, i)$, $H = H_3$, and $[\![!P]\!]\rho_3 s_3 s'_3 H_3$ has been called. Let ρ_2, s_2, s'_2, H_2 such that $\Sigma \vdash \rho_2 = \sigma\rho$, $\Sigma \vdash s_2 = \sigma s$, $\Sigma \vdash s'_2 = \sigma s'$, $\Sigma \vdash H_2 \sim \sigma H$, and $\mathrm{nf}'_{\mathcal{S},\Sigma}(\{\rho_2, s_2, s'_2, H_2\})$.

  Then $\rho_2 = \rho_4$, $s_2 = (s_4, p)$, $s'_2 = (s'_4, p)$, $H_2 = H_4$ where $\Sigma \vdash \rho_4 = \sigma\rho_3$, $\Sigma \vdash s_4 = \sigma s_3$, $\Sigma \vdash s'_4 = \sigma s'_3$, $\Sigma \vdash H_4 \sim \sigma H_3$, and $\Sigma \vdash p = \sigma i$.

  By induction hypothesis, there exist σ_1, ρ_5, s_5, s'_5, H_5 such that $\rho_4 = \sigma_1\rho_5$, $s_4 = \sigma_1 s_5$, $s'_4 = \sigma_1 s'_5$, $H_4 \sim \sigma_1 H_5$, and $[\![!P]\!]\rho_5 s_5 s'_5 H_5$ has been called. Since i is a fresh variable, we can define $\sigma_1 i = p$.

  Then $[\![P]\!]\rho_5(s_5, i)(s'_5, i)H_5$ has been called, $\rho_2 = \sigma_1\rho_5$, $s_2 = \sigma_1(s_5, i)$, $s'_2 = \sigma_1(s'_5, i)$, and $H_2 \sim \sigma_1 H_5$.

- Case $(\nu a)P$: Assume $[\![P]\!]\rho s s' H$ is called. Then $\rho = \rho_3[a \mapsto (a[s], a[s'])]$ and $[\![(\nu a)P]\!]\rho_3 s s' H$ has been called. Let ρ_2, s_2, s'_2, H_2 such that $\Sigma \vdash \rho_2 = \sigma\rho$, $\Sigma \vdash s_2 = \sigma s$, $\Sigma \vdash s'_2 = \sigma s'$, $\Sigma \vdash H_2 \sim \sigma H$, and $\mathrm{nf}'_{\mathcal{S},\Sigma}(\{\rho_2, s_2, s'_2, H_2\})$.

  Then $\rho_2 = \rho_4[a \mapsto (a[s_2], a[s'_2])]$ where $\Sigma \vdash \rho_4 = \sigma\rho_3$.

  By induction hypothesis, there exist σ_1, ρ_5, s_1, s'_1, H_1 such that $\rho_4 = \sigma_1\rho_5$, $s_2 = \sigma_1 s_1$, $s'_2 = \sigma_1 s'_1$, $H_2 \sim \sigma_1 H_1$, and $[\![(\nu a)P]\!]\rho_5 s_1 s'_1 H_1$ has been called.

  Then $[\![P]\!](\rho_5[a \mapsto (a[s_1], a[s'_1])])s_1 s'_1 H_1$ has been called, $\rho_2 = \sigma_1(\rho_5[a \mapsto (a[s_1], a[s'_1])])$, $s_2 = \sigma_1 s_1$, $s'_2 = \sigma_1 s'_1$, and $H_2 \sim \sigma_1 H_1$.

- Case $\overline{M}\langle N\rangle.P$: Obvious by induction hypothesis.

- Case $M(x).P$: Assume $[\![P]\!]\rho s s' H$ is called. Then $\rho = \rho_3[x \mapsto (x', x'')]$, $s = (s_3, x')$, $s' = (s'_3, x'')$, $H = H_3 \wedge \mathrm{msg}'(\rho_3(M)_1, x', \rho_3(M)_2, x'')$, and $[\![M(x).P]\!]\rho_3 s_3 s'_3 H_3$ has been called. Let ρ_2, s_2, s'_2, H_2 such that $\Sigma \vdash \rho_2 = \sigma\rho$, $\Sigma \vdash s_2 = \sigma s$, $\Sigma \vdash s'_2 = \sigma s'$, $\Sigma \vdash H_2 \sim \sigma H$, and $\mathrm{nf}'_{\mathcal{S},\Sigma}(\{\rho_2, s_2, s'_2, H_2\})$.

  Then $\rho_2 = \rho_4[x \mapsto (p', p'')]$, $s_2 = (s_4, p')$, $s'_2 = (s'_4, p'')$, $H_2 = H_4 \wedge \mathrm{msg}'(\rho_4(M)_1, p', \rho_4(M)_2, p'')$ where $\Sigma \vdash \rho_4 = \sigma\rho_3$, $\Sigma \vdash s_4 = \sigma s_3$, $\Sigma \vdash s'_4 = \sigma s'_3$, $\Sigma \vdash H_4 \sim \sigma H_3$, $\Sigma \vdash p' = \sigma x'$, and $\Sigma \vdash p'' = \sigma x''$. (Since P_0 is unevaluated, M is a variable y or $\mathsf{diff}[a, a]$ for some name a. Let $u = y$ in the first case and $u = a$ in the second case. We have $u \in dom(\rho_3) = dom(\rho_4)$. We have $\mathrm{nf}'_{\mathcal{S},\Sigma}(\{\rho_2, s_2, s'_2, H_2\})$ so a fortiori $\mathrm{nf}'_{\mathcal{S},\Sigma}(\{\rho_4, H_2\})$, and the first and third arguments of msg' are equal to $\rho_4(M)_1 = \rho_4(u)_1$ and $\rho_4(M)_2 = \rho_4(u)_2$ modulo Σ respectively, so they are exactly $\rho_4(M)_1$ and $\rho_4(M)_2$.)

By induction hypothesis, there exist σ_1, ρ_5, s_5, s'_5, H_5 such that $\rho_4 = \sigma_1\rho_5$, $s_4 = \sigma_1 s_5$, $s'_4 = \sigma_1 s'_5$, $H_4 \sim \sigma_1 H_5$, and $[\![M(x).P]\!]\rho_5 s_5 s'_5 H_5$ has been called. Since x' and x'' are fresh variables, we can define $\sigma_1 x' = p'$ and $\sigma_1 x'' = p''$.

Then $[\![P]\!](\rho_5[x \mapsto (x', x'')])(s_5, x')(s'_5, x'')(H_5 \wedge \mathrm{msg}'(\rho_5(M)_1, x', \rho_5(M)_2, x''))$ has been called, and $\rho_2 = \sigma_1(\rho_5[x \mapsto (x', x'')])$, $s_2 = \sigma_1(s_5, x')$, $s'_2 = \sigma_1(s'_5, x'')$, and $H_2 \sim \sigma_1(H_5 \wedge \mathrm{msg}'(\rho_5(M)_1, x', \rho_5(M)_2, x''))$.

- Case *let* $x = D$ *in* P *else* Q:

  Subprocess P: Assume $[\![P]\!]\rho s s' H$ is called. Then we have $\rho = (\sigma_1\rho_3)[x \mapsto (p_1, p'_1)]$, $s = (\sigma_1 s_3, p_1)$, $s' = (\sigma_1 s'_3, p'_1)$, and $H = \sigma_1 H_3$ where $[\![let\ x = D\ in\ P\ else\ Q]\!]\rho_3 s_3 s'_3 H_3$ has been called and $(\rho(D)_1, \rho(D)_2) \Downarrow' ((p_1, p'_1), \sigma_1)$. Let ρ_2, s_2, s'_2, H_2 such that $\Sigma \vdash \rho_2 = \sigma\rho$, $\Sigma \vdash s_2 = \sigma s$, $\Sigma \vdash s'_2 = \sigma s'$, $\Sigma \vdash H_2 \sim \sigma H$, and $\mathrm{nf}'_{\mathcal{S},\Sigma}(\{\rho_2, s_2, s'_2, H_2\})$.

  Then $\rho_2 = \rho_4[x \mapsto (p_4, p'_4)]$, $s_2 = (s_4, p_4)$, $s'_2 = (s'_4, p'_4)$, $H_2 = H_4$ with $\Sigma \vdash \rho_4 = \sigma\sigma_1\rho_3$, $\Sigma \vdash s_4 = \sigma\sigma_1 s_3$, $\Sigma \vdash s_4 = \sigma\sigma_1 s'_3$, $\Sigma \vdash H_4 \sim \sigma\sigma_1 H_3$, $\Sigma \vdash p_4 = \sigma p_1$, $\Sigma \vdash p'_4 = \sigma p'_1$, and $\mathrm{nf}'_{\mathcal{S},\Sigma}(\{\rho_4, s_4, s'_4, H_4, p_4, p'_4\})$.

  By induction hypothesis, there exist $\sigma'_0, \rho_5, s_5, s'_5, H_5$ such that $\rho_4 = \sigma'_0\rho_5$, $s_4 = \sigma'_0 s_5$, $s'_4 = \sigma'_0 s'_4$, $H_4 \sim \sigma'_0 H_5$, and $[\![let\ x = D\ in\ P\ else\ Q]\!]\rho_5 s_5 s'_5 H_5$ has been called.

  By Lemma 26, there exist σ_2, p_2, p'_2, and σ_3 such that $(\rho_5(D)_1, \rho_5(D)_2) \Downarrow' ((p_2, p'_2), \sigma_2)$, $\sigma'_0 = \sigma_3\sigma_2$ except on fresh variables introduced in the computation of $(\rho_5(D)_1, \rho_5(D)_2) \Downarrow' ((p_2, p'_2), \sigma_2)$, $p_4 = \sigma_3 p_2$, and $p'_4 = \sigma_3 p'_2$.

  Moreover, by definition of $[\![let\ x = D\ in\ P\ else\ Q]\!]$, $[\![P]\!]((\sigma_2\rho_5)[x \mapsto (p_2, p'_2)])(\sigma_2 s_5, p_2)(\sigma_2 s'_5, p'_2)(\sigma_2 H_5)$ has been called, so we obtain the result by letting $\rho_1 = (\sigma_2\rho_5)[x \mapsto (p_2, p'_2)]$, $s_1 = (\sigma_2 s_5, p_2)$, $s'_1 = (\sigma_2 s'_5, p'_2)$, $H_1 = \sigma_2 H_5$: we have $\rho_2 = \rho_4[x \mapsto (p_4, p'_4)] = (\sigma'_0\rho_5)[x \mapsto (\sigma_3 p_2, \sigma_3 p'_2)] = \sigma_3((\sigma_2\rho_5)[x \mapsto (p_2, p'_2)]) = \sigma_3\rho_1$, and similarly for s_2, s'_2, and H_2.

  Subprocess Q: Assume $[\![Q]\!]\rho s s' H$ is called. Then $H = H_3 \wedge \rho(\mathrm{fails}(\mathsf{fst}(D)))_1 \wedge \rho(\mathrm{fails}(\mathsf{snd}(D)))_2$ and $[\![let\ x = D\ in\ P\ else\ Q]\!]\rho s s' H_3$ has been called. Let ρ_2, s_2, s'_2, H_2 such that $\Sigma \vdash \rho_2 = \sigma\rho$, $\Sigma \vdash s_2 = \sigma s$, $\Sigma \vdash s'_2 = \sigma s'$, $\Sigma \vdash H_2 \sim \sigma H$, and $\mathrm{nf}'_{\mathcal{S},\Sigma}(\{\rho_2, s_2, s'_2, H_2\})$.

  Then $H_2 = H_4 \wedge H_{4\mathrm{nounif}}$ where $H_{4\mathrm{nounif}}$ consists of nounif facts, $\Sigma \vdash H_{4\mathrm{nounif}} \sim \sigma\rho(\mathrm{fails}(\mathsf{fst}(D)))_1 \wedge \sigma\rho(\mathrm{fails}(\mathsf{snd}(D)))_2$, and $\Sigma \vdash H_4 \sim \sigma H_3$.

  By induction hypothesis, there exist $\sigma_1, \rho_1, s_1, s'_1, H_5$ such that $\rho_2 = \sigma_1\rho_1$, $s_2 = \sigma_1 s_1$, $s'_2 = \sigma_1 s'_1$, $H_4 \sim \sigma_1 H_5$, and $[\![let\ x = D\ in\ P\ else\ Q]\!]\rho_1 s_1 s'_1 H_5$ has been called.

  Then $[\![Q]\!]\rho_1 s_1 s'_1 (H_5 \wedge \rho_1(\mathrm{fails}(\mathsf{fst}(D)))_1 \wedge \rho_1(\mathrm{fails}(\mathsf{snd}(D)))_2)$ has been called, which yields the desired result, knowing that $H_2 = H_4 \wedge H_{4\mathrm{nounif}} \sim \sigma_1 H_5 \wedge \sigma_1\rho_1(\mathrm{fails}(\mathsf{fst}(D)))_1 \wedge \sigma_1\rho_1(\mathrm{fails}(\mathsf{snd}(D)))_2$, since $\Sigma \vdash \sigma_1\rho_1 = \rho_2 = \sigma\rho$. □

Lemma 28 *Let P_0 be a closed, unevaluated process. For all clauses $H \to C \in \mathcal{R}_{P_0}$, for all closed substitutions σ, for all $H_2 \to C_2$ such that $\Sigma \vdash H_2 \to C_2 \sim \sigma(H \to C)$ and $\mathrm{nf}'_{\mathcal{S},\Sigma}(\{H_2, C_2\})$, there exist a closed substitution σ_1 and a clause $H_1 \to C_1 \in \mathcal{R}_{P_0}$ such that $H_2 \sim \sigma_1 H_1$ and $C_2 = \sigma_1 C_1$.*

Proof The clauses of $[\![P_0]\!]\rho_0\emptyset\emptyset\emptyset$ are generated from the following cases:

- $H \to C = H \to \mathrm{input}'(\rho(M)_1, \rho(M)_2)$ where $[\![M(x).P]\!]\rho ss'H$ has been called during the generation of $[\![P_0]\!]\rho_0\emptyset\emptyset\emptyset$. Since $\Sigma \vdash H_2 \to C_2 \sim \sigma(H \to C)$ and $\mathrm{nf}'_{\mathcal{S},\Sigma}(\{H_2, C_2\})$, we have $\Sigma \vdash H_2 \sim \sigma H$, $C_2 = \mathrm{input}'(p_2, p'_2)$, $\Sigma \vdash p_2 = \sigma\rho(M)_1$, and $\Sigma \vdash p'_2 = \sigma\rho(M)_2$.

  Since P_0 is unevaluated, M is a variable y or $\mathsf{diff}[a, a]$ for some name a. Let $u = y$ in the first case and $u = a$ in the second case. We have $u \in dom(\rho)$. We define ρ_2 by $\rho_2(u) = \mathsf{diff}[p_2, p'_2]$ and extend ρ_2 to $dom(\rho)$ in such a way that $\Sigma \vdash \rho_2 = \sigma\rho$ and $\mathrm{nf}'_{\mathcal{S},\Sigma}(\{H_2, \rho_2\})$ by Property S2. We also define s_2 and s'_2 so that $\Sigma \vdash s_2 = \sigma s$, $\Sigma \vdash s'_2 = \sigma s'$, and $\mathrm{nf}'_{\mathcal{S},\Sigma}(\{H_2, \rho_2, s_2, s'_2\})$ by Property S2. By Lemma 27, there exist σ_1, ρ_1, s_1, s'_1, H_1 such that $\rho_2 = \sigma_1\rho_1$, $s_2 = \sigma_1 s_1$, $s'_2 = \sigma_1 s'_1$, $H_2 \sim \sigma_1 H_1$, and $[\![M(x).P]\!]\rho_1 s_1 s'_1 H_1$ has been called.

  Then $H_1 \to \mathrm{input}'(\rho_1(M)_1, \rho_1(M)_2)$ is in $[\![P_0]\!]\rho_0\emptyset\emptyset\emptyset$, $H_2 \sim \sigma_1 H_1$, $C_2 = \mathrm{input}'(p_2, p'_2) = \mathrm{input}'(\rho_2(M)_1, \rho_2(M)_2) = \sigma_1\mathrm{input}'(\rho_1(M)_1, \rho_1(M)_2)$.

- $H \to C = H \to \mathrm{msg}'(\rho(M)_1, \rho(N)_1, \rho(M)_2, \rho(N)_2)$ where $[\![\overline{M}\langle N\rangle.P]\!]\rho ss'H$ has been called. This case is similar to the previous one. (The terms M and N are variables or $\mathsf{diff}[a, a]$ for some name a.)

- $H \to C = \sigma'H' \wedge \sigma'\rho(\mathrm{fails}(\mathsf{snd}(D)))_2 \to \mathrm{bad}$ where $[\![let\ x = D\ in\ P\ else\ Q]\!]\rho ss'H'$ has been called and $\rho(D)_1 \Downarrow' (p', \sigma')$. Since $\Sigma \vdash H_2 \to C_2 \sim \sigma(H \to C)$ and $\mathrm{nf}'_{\mathcal{S},\Sigma}(\{H_2, C_2\})$, we have $H_2 = H_3 \wedge H_{3\mathrm{nounif}}$ where $\Sigma \vdash H_3 \sim \sigma\sigma'H'$ and $H_{3\mathrm{nounif}}$ consists of nounif facts such that $\Sigma \vdash H_{3\mathrm{nounif}} \sim \sigma\sigma'\rho(\mathrm{fails}(\mathsf{snd}(D)))_2$. By Property S2, there exist ρ_3, s_3, s'_3 such that $\Sigma \vdash \rho_3 = \sigma\sigma'\rho$, $\Sigma \vdash s_3 = \sigma\sigma's$, $\Sigma \vdash s'_3 = \sigma\sigma's'$, and $\mathrm{nf}'_{\mathcal{S},\Sigma}(\{\rho_3, s_3, s'_3, H_3\})$.

  By Lemma 27, there exist σ_1, ρ_1, s_1, s'_1, H_1 such that $\rho_3 = \sigma_1\rho_1$, $s_3 = \sigma_1 s_1$, $s'_3 = \sigma_1 s'_1$, $H_3 \sim \sigma_1 H_1$, and $[\![let\ x = D\ in\ P\ else\ Q]\!]\rho_1 s_1 s'_1 H_1$ has been called.

  By Property S2, we can choose p such that $\Sigma \vdash p = \sigma p'$ and $\mathrm{nf}_{\mathcal{S},\Sigma}(\{p, \sigma_1\rho_1\})$. By Lemma 26, there exist σ'_1, p'_1, and σ''_1 such that $\rho_1(D)_1 \Downarrow' (p'_1, \sigma'_1)$ and $\sigma_1 = \sigma''_1\sigma'_1$ except on fresh variables introduced in the computation of $\rho_1(D)_1 \Downarrow' (p'_1, \sigma'_1)$. Then $\sigma'_1 H_1 \wedge \sigma'_1\rho_1(\mathrm{fails}(\mathsf{snd}(D)))_2 \to \mathrm{bad}$ is in $[\![P_0]\!]\rho_0\emptyset\emptyset\emptyset$. Moreover $\sigma''_1(\sigma'_1 H_1 \wedge \sigma'_1\rho_1(\mathrm{fails}(\mathsf{snd}(D))))_2) = \sigma_1 H_1 \wedge \sigma_1\rho_1(\mathrm{fails}(\mathsf{snd}(D)))_2 \sim H_3 \wedge H_{3\mathrm{nounif}} \sim H_2$, since $\Sigma \vdash \sigma_1\rho_1 = \rho_3 = \sigma\sigma'\rho$, and $\sigma''_1\mathrm{bad} = \mathrm{bad} = C$, so we have the desired result.

- $H \to C = \sigma'H' \wedge \sigma'\rho(\mathrm{fails}(\mathsf{fst}(D)))_1 \to \mathrm{bad}$ where $[\![let\ x = D\ in\ P\ else\ Q]\!]\rho ss'H'$ has been called and $\rho(D)_2 \Downarrow' (p', \sigma')$. This case is symmetric from the previous one.

For the other clauses:

- For Clause (Rinit), $C_2 = C$, $H_2 = \emptyset$, so we have the result by taking $H_1 \to C_1 = H \to C$.

- For Clauses (Rn), (Rl), (Rs), (Ri), (Rcom), and (Rcom′), $H_2 = \sigma' H$ and $C_2 = \sigma' C$ where for all $x \in fv(H \to C)$, $\Sigma \vdash \sigma' x = \sigma x$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(\{\sigma' x \mid x \in fv(H \to C)\})$. (Indeed, the function symbols in H, C do not appear in equations of Σ.) So we obtain the result by taking $H_1 \to C_1 = H \to C$ and $\sigma_1 = \sigma'$.

- For Clause (Rf), $H = \mathrm{att}'(M_1, N_1) \wedge \ldots \wedge \mathrm{att}'(M_n, N_n)$, $C = \mathrm{att}'(M, N)$, $h(M_1, \ldots, M_n) \to M$ in $\mathrm{def}_{\Sigma'}(h)$, $h(N_1, \ldots, N_n) \to N$ in $\mathrm{def}_{\Sigma'}(h)$, $H_2 = \mathrm{att}'(M_1'', N_1'') \wedge \ldots \wedge \mathrm{att}'(M_n'', N_n'')$, $C_2 = \mathrm{att}'(M'', N'')$ with $\Sigma \vdash M'' = \sigma M$, $\Sigma \vdash N'' = \sigma N$, $\Sigma \vdash M_i'' = \sigma M_i$ and $\Sigma \vdash N_i'' = \sigma N_i$ for all $i \in \{1, \ldots, n\}$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(\{M'', N'', M_1'', \ldots, M_n'', N_1'', \ldots, N_n''\})$. By Lemma 24, there exist σ_1 and $h(M_1', \ldots, M_n') \to M'$ in $\mathrm{def}_{\Sigma'}(h)$ such that $M'' = \sigma_1 M'$ and for all $i \in \{1, \ldots, n\}$, $M_i'' = \sigma_1 M_i'$. By Lemma 24 again, there exist σ_1 and $h(N_1', \ldots, N_n') \to N'$ in $\mathrm{def}_{\Sigma'}(h)$ such that $N'' = \sigma_1 N'$ and for all $i \in \{1, \ldots, n\}$, $N_i'' = \sigma_1 N_i'$. (We can use the same substitution σ_1 since the first and second arguments of the predicate att' do not share variables.) Hence $\sigma_1 \mathrm{att}'(M_i', N_i') = \mathrm{att}'(M_i'', N_i'')$ for all $i \in \{1, \ldots, n\}$ and $\sigma_1 \mathrm{att}'(M', N') = \mathrm{att}'(M'', N'')$. We take $H_1 \to C_1 = \mathrm{att}'(M_1', N_1') \wedge \ldots \wedge \mathrm{att}'(M_n', N_n') \to \mathrm{att}'(M', N')$, which yields the desired result.

- For Clause (Rt), we have $C_2 = C = \mathrm{bad}$, $H = H_{\mathrm{nounif}} \wedge \mathrm{att}'(M_1, x_1) \wedge \ldots \wedge \mathrm{att}'(M_n, x_n)$, $H_2 = H_{2\mathrm{nounif}} \wedge \mathrm{att}'(M_1'', N_1'') \wedge \ldots \wedge \mathrm{att}'(M_n'', N_n'')$ where H_{nounif} and $H_{2\mathrm{nounif}}$ consist of nounif facts, $\Sigma \vdash H_{2\mathrm{nounif}} = \sigma H_{\mathrm{nounif}}$, $g(M_1, \ldots, M_n) \to M$ in $\mathrm{def}_{\Sigma'}(g)$, $\Sigma \vdash M_i'' = \sigma M_i$ and $\Sigma \vdash N_i'' = \sigma x_i$ for all $i \in \{1, \ldots, n\}$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(\{M_1'', \ldots, M_n'', N_1'', \ldots, N_n''\})$. By Lemma 24, there exist σ_1 and $g(M_1', \ldots, M_n') \to M'$ in $\mathrm{def}_{\Sigma'}(g)$ such that $M'' = \sigma_1 M'$ and for all $i \in \{1, \ldots, n\}$, $M_i'' = \sigma_1 M_i'$. We extend σ_1 by defining for all $i \in \{1, \ldots, n\}$, $\sigma_1 x_i = N_i''$. Hence $\sigma_1 \mathrm{att}'(M_i', x_i) = \mathrm{att}'(M_i'', N_i'')$ for all $i \in \{1, \ldots, n\}$ and $\Sigma \vdash H_{2\mathrm{nounif}} = \sigma H_{\mathrm{nounif}} = \sigma_1 H_{\mathrm{nounif}}$ since for all $i \in \{1, \ldots, n\}$, $\Sigma \vdash \sigma_1 x_i = N_i'' = \sigma x_i$ and $fv(H_{\mathrm{nounif}}) = \{x_1, \ldots, x_n\}$. We take $H_1 \to C_1 = H_{\mathrm{nounif}} \wedge \mathrm{att}'(M_1', x_1) \wedge \ldots \wedge \mathrm{att}'(M_n', x_n) \to \mathrm{bad}$ which yields the result.

  The case of Clause (Rt′) is symmetric. □

Lemma 29 *Assume P_0 is a closed, unevaluated process. If F is derivable from $\mathcal{R}_{P_0}$, $\Sigma \vdash F'' \sim F$, and $\mathrm{nf}'_{\mathcal{S},\Sigma}(\mathcal{F} \cup \{F''\})$, then F'' is derivable from $\mathcal{R}_{P_0}$ by a derivation $\mathcal{D}$ such that $\mathrm{nf}'_{\mathcal{S},\Sigma}(\mathcal{F} \cup \{\mathcal{D}\})$.*

Proof The proof is by induction on the derivation of F. Assume that F is derived from $F_1, \ldots, F_n$, using a clause $R \in \mathcal{R}_{P_0}$: there exists a closed substitution σ such that $\sigma R = F_1 \wedge \ldots \wedge F_n \to F$. Let F'' such that $\Sigma \vdash F'' \sim F$ and $\mathrm{nf}'_{\mathcal{S},\Sigma}(\mathcal{F} \cup \{F''\})$. By Property S2, there exist $F_1'', \ldots, F_n''$ such that $\Sigma \vdash F_i'' \sim F_i$ for all $i \in \{1, \ldots, n\}$ and $\mathrm{nf}'_{\mathcal{S},\Sigma}(\mathcal{F} \cup \{F'', F_1'', \ldots, F_n''\})$. By Lemma 28, there exist a closed substitution σ_1 and a clause $R' = F_1' \wedge \ldots \wedge F_n' \to F' \in \mathcal{R}_{P_0}$ such that $F'' = \sigma_1 F'$ and $F_i'' \sim \sigma_1 F_i'$ for all $i \in \{1, \ldots, n\}$. So F'' is derivable from $F_1'', \ldots, F_n''$ by R'. Furthermore, for all $i \in \{1, \ldots, n\}$, $\Sigma \vdash F_i'' \sim F_i$, $\mathrm{nf}'_{\mathcal{S},\Sigma}(\mathcal{F} \cup \{F_1'', \ldots, F_n'', F''\})$, and F_i is derivable from $\mathcal{R}_{P_0}$. So by induction hypothesis, F_i'' is derivable from $\mathcal{R}_{P_0}$, by a derivation

$\mathcal{D}_i$ such that $\mathrm{nf}_{\mathcal{S},\Sigma}(\mathcal{F} \cup \{\mathcal{D}_1, \ldots, \mathcal{D}_i, F''_{i+1}, \ldots, F''_n, F''\})$. (We apply the induction hypothesis with $\mathcal{F} \cup \{\mathcal{D}_1, \ldots, \mathcal{D}_{i-1}, F''_{i+1}, \ldots, F''_n, F''\}$ instead of $\mathcal{F} \cup \{F''\}$.) Then F'' is derivable from $\mathcal{R}_{P_0}$ by a derivation $\mathcal{D}$ built from $\mathcal{D}_1, \ldots, \mathcal{D}_n$ and R', such that $\mathrm{nf}'_{\mathcal{S},\Sigma}(\mathcal{F} \cup \{\mathcal{D}\})$. □

Lemma 3 is a particular case of Lemma 29, taking $F = F'' = \mathrm{bad}$.

D Proof of Theorem 3

The following lemma is useful for establishing the soundness of the translation of "there exists no p such that $\sigma D \Downarrow_\Sigma p$" into $\sigma\mathrm{fails}(D)$. This translation appears when we show the soundness of clauses for term evaluations.

Lemma 30 *If $\sigma\mathrm{fails}(D)$ is false then there exists a pattern p such that $\sigma D \Downarrow_\Sigma p$.*

Proof By definition of fails, there exist a pattern p and σ' such that $D \Downarrow' (p, \sigma')$ and $\sigma\mathrm{nounif}(D, GVar(\sigma' D))$ is false. By definition of nounif, there exists a closed σ'' such that $\Sigma \vdash \sigma D = \sigma''\sigma' D$. By Lemma 25, since $D \Downarrow' (p, \sigma')$, there exists p' such that $\sigma''\sigma' D \Downarrow_\Sigma p'$ and $\Sigma \vdash p' = \sigma'' p$. By a variant of Lemma 18 for patterns instead of terms, $\Sigma \vdash \sigma D = \sigma''\sigma' D$ implies $\sigma D \Downarrow_\Sigma p''$ for some p'' such that $\Sigma \vdash p'' = p'$. □

Proof (of Theorem 3) We exploit the theory developed in [3, 16] to prove the hypotheses of Lemma 2. This theory uses a type system to express the invariant that corresponds to the soundness of the clauses, and a subject reduction theorem to show that the invariant is indeed preserved. Here, types range over *pairs of closed patterns*, after adding constant session identifiers λ to the grammar of patterns.

We first define instrumented biprocesses in which a pattern is associated with each name. The syntax of instrumented biprocesses is the same as the syntax of biprocesses except that the replication is replaced with $!^i P$ where i is a variable session identifier and the restriction is replaced with $(\nu a : a_0[M_1, \ldots, M_n])$ where a_0 is a function symbol and $M_1, \ldots, M_n$ are terms or (constant or variable) session identifiers. In Section 6.3 and below, we reuse the name a as function symbol a_0. In contrast with a and any names occurring in $M_1, \ldots, M_n$, however, the function symbol a_0 is not subject to renaming, so we may have $a \neq a_0$ after an α-conversion on a.

To every closed biprocess P with pairwise distinct bound variables, we associate the instrumented biprocess $\mathrm{instr}(P)$ obtained by adding a distinct session identifier i to each replication in P and by labelling each restriction (νa) of P with $(\nu a : a[x_1, \ldots, x_n])$ where $x_1, \ldots, x_n$ are the variables and session identifiers bound above (νa) in $\mathrm{instr}(P)$. Conversely, we let $\mathrm{delete}(P)$ be the biprocess obtained by erasing instrumentation from any instrumented biprocess P.

We define the semantics of instrumented biprocesses using configurations $\Lambda; P$ where Λ is a countable set of constant session identifiers and P is an instrumented biprocess. Intuitively, Λ is the set of session identifiers not yet used in the reduction of P. The rule (Red Repl) is defined as follows for instrumented biprocesses:

$$\Lambda; !^i P \to \Lambda - \{\lambda\}; !^i P \mid P\{\lambda/i\} \text{ if } \lambda \in \Lambda$$

This rule chooses a fresh session identifier λ in Λ, removes it from Λ, and uses it for the new copy of P. The other rules of Figures 2 and 3 that define reduction and structural congruence are lifted from $P \to Q$ to $\Lambda; P \to \Lambda; Q$ and from $P \equiv Q$ to $\Lambda; P \equiv \Lambda; Q$.

By construction, instrumented biprocesses include the variables that were collected by s and s' in the definition of $[\![_]\!]\rho s s' H$ of Section 6.3. Hence, the clauses $[\![P]\!]\rho_0\emptyset\emptyset\emptyset$ can be computed from $\mathrm{instr}(P)$ as follows: $[\![P]\!]\rho_0\emptyset\emptyset\emptyset = [\![\mathrm{instr}(P)]\!]\rho_0\emptyset$ where

$$
\begin{aligned}
&[\![0]\!]\rho H = \emptyset\\
&[\![!^i P]\!]\rho H = [\![P]\!](\rho[i \mapsto (i,i)])H\\
&[\![P \mid Q]\!]\rho H = [\![P]\!]\rho H \cup [\![Q]\!]\rho H\\
&[\![(\nu a : a[x_1,\ldots,x_n])P]\!]\rho H =\\
&\quad [\![P]\!](\rho[a \mapsto (a[\rho(x_1)_1,\ldots,\rho(x_n)_1], a[\rho(x_1)_2,\ldots,\rho(x_n)_2])])H\\
&[\![M(x).P]\!]\rho H = [\![P]\!](\rho[x \mapsto (x', x'')])(H \wedge \mathrm{msg}'(\rho(M)_1, x', \rho(M)_2, x''))\\
&\quad \cup \{H \to \mathrm{input}'(\rho(M)_1, \rho(M)_2)\}\\
&\quad \text{where } x' \text{ and } x'' \text{ are fresh variables}\\
&[\![\overline{M}\langle N\rangle.P]\!]\rho H = [\![P]\!]\rho H \cup \{H \to \mathrm{msg}'(\rho(M)_1, \rho(N)_1, \rho(M)_2, \rho(N)_2)\}\\
&[\![\mathit{let}\ x = D\ \mathit{in}\ P\ \mathit{else}\ Q]\!]\rho H =\\
&\quad \bigcup\{[\![P]\!]((\sigma\rho)[x \mapsto (p,p')])(\sigma H) \mid (\rho(D)_1, \rho(D)_2) \Downarrow' ((p,p'),\sigma)\}\\
&\quad \cup [\![Q]\!]\rho(H \wedge \rho(\mathrm{fails}(\mathsf{fst}(D)))_1 \wedge \rho(\mathrm{fails}(\mathsf{snd}(D)))_2)\\
&\quad \cup \{\sigma H \wedge \sigma\rho(\mathrm{fails}(\mathsf{snd}(D)))_2 \to \mathrm{bad} \mid \rho(D)_1 \Downarrow' (p,\sigma)\}\\
&\quad \cup \{\sigma H \wedge \sigma\rho(\mathrm{fails}(\mathsf{fst}(D)))_1 \to \mathrm{bad} \mid \rho(D)_2 \Downarrow' (p',\sigma)\}
\end{aligned}
$$

Let C be a plain evaluation context. For each reduction $\mathrm{unevaluated}(C[P_0]) \to^*_{\Sigma',\Sigma}\equiv P$, there is a reduction $\Lambda_0; \mathrm{instr}(\mathrm{unevaluated}(C[P_0])) \to^*_{\Sigma',\Sigma}\equiv \Lambda; P'$ such that $\mathrm{delete}(P') = P$ (and conversely). Let $P'_0 = \mathrm{instr}(\mathrm{unevaluated}(P_0))$. There exists an unevaluated evaluation context C' such that diff occurs only in terms $\mathsf{diff}[a,a]$ for some names a in C' and $\mathrm{instr}(\mathrm{unevaluated}(C[P_0])) = C'[P'_0]$.

Let $\mathcal{R}_{C',P_0}$ be the set of clauses obtained by adding to $\mathcal{R}_{P_0}$ the clauses

$$\mathrm{att}'(a[x_1,\ldots,x_n], a[x_1,\ldots,x_n]) \qquad \text{(Rn}')$$

such that either $(\nu a : a[x'_1,\ldots,x'_n])$ occurs in C' or $n = 0$, $a \in \mathit{fn}(C')$, and $a \notin \mathit{fn}(P'_0)$. The fact bad is derivable from $\mathcal{R}_{C',P_0}$ if and only if bad is derivable from $\mathcal{R}_{P_0}$, since we can replace all patterns $a[\ldots]$ of names created by the context C' with patterns $b[i]$, as long as different names have different images, so we can replace the Clauses (Rn′) with Clause (Rn). Hence, the definition of $\mathcal{R}_{P_0}$ is sufficient.

Next, we define a type system, similar to that of [3, Section 7]. Here, the types are pairs of closed patterns. The type environment E is a function from variables and names to types. It is extended to terms as a substitution, so that a term M has type $E(M)$. The typing judgment $E \vdash P$ says that the instrumented biprocess P is well-typed in environment E. This judgment is formally defined in Figure 5, where $\mathcal{F}_{C',P_0}$ is the set of closed facts derivable from $\mathcal{R}_{C',P_0}$.

$$\frac{\begin{array}{c}\mathrm{input}'(E(M)_1, E(M)_2) \in \mathcal{F}_{C',P_0} \\ \forall p_1, p_2 \text{ such that } \mathrm{msg}'(E(M)_1, p_1, E(M)_2, p_2) \in \mathcal{F}_{C',P_0}, E[x \mapsto (p_1, p_2)] \vdash P\end{array}}{E \vdash M(x).P} \quad \text{(Input)}$$

$$\frac{\mathrm{msg}'(E(M)_1, E(N)_1, E(M)_2, E(N)_2) \in \mathcal{F}_{C',P_0} \qquad E \vdash P}{E \vdash \overline{M}\langle N\rangle.P} \quad \text{(Output)}$$

$$\frac{}{E \vdash 0} \quad \text{(Nil)}$$

$$\frac{E \vdash P \qquad E \vdash Q}{E \vdash P \mid Q} \quad \text{(Parallel)}$$

$$\frac{\forall \lambda, E[i \mapsto (\lambda, \lambda)] \vdash P}{E \vdash !^i P} \quad \text{(Replication)}$$

$$\frac{E[a \mapsto (a_0[E(M_1)_1, \ldots, E(M_n)_1], a_0[E(M_1)_2, \ldots, E(M_n)_2])\,] \vdash P}{E \vdash (\nu a : a_0[M_1, \ldots, M_n])P} \quad \text{(Restriction)}$$

$$\frac{\begin{array}{c}\forall p_1, p_2 \text{ such that } E(D)_1 \Downarrow_{\Sigma'} p_1 \text{ and } E(D)_2 \Downarrow_{\Sigma'} p_2, E[x \mapsto (p_1, p_2)] \vdash P \\ \text{if } \nexists p_1, E(D)_1 \Downarrow_{\Sigma} p_1 \text{ and } \nexists p_2, E(D)_2 \Downarrow_{\Sigma} p_2, \text{ then } E \vdash Q \\ \text{if } \exists p_1, E(D)_1 \Downarrow_{\Sigma'} p_1 \text{ and } \nexists p_2, E(D)_2 \Downarrow_{\Sigma} p_2, \text{ then } \mathrm{bad} \in \mathcal{F}_{C',P_0} \\ \text{if } \nexists p_1, E(D)_1 \Downarrow_{\Sigma} p_1 \text{ and } \exists p_2, E(D)_2 \Downarrow_{\Sigma'} p_2, \text{ then } \mathrm{bad} \in \mathcal{F}_{C',P_0}\end{array}}{E \vdash \mathit{let}\ x = D\ \mathit{in}\ P\ \mathit{else}\ Q} \quad \text{(Term evaluation)}$$

Figure 5: Type rules

When $M_1, \ldots, M_n$ is a sequence of terms and (variable or constant) session identifiers, as in labels of restrictions, we define $\mathrm{last}(M_1, \ldots, M_n)$ as the last M_i that is a session identifier, or $\emptyset$ when no M_i is a session identifier. Let us define the multiset $\mathit{Label}(P)$ as follows: $\mathit{Label}((\nu a : a_0[M_1, \ldots, M_n])P) = \{(a_0, \mathrm{last}(M_1, \ldots, M_n))\} \cup \mathit{Label}(P)$, $\mathit{Label}(!^i P) = \emptyset$, and in all other cases, $\mathit{Label}(P)$ is the union of the $\mathit{Label}(P')$ for all immediate subprocesses P' of P. When E maps names to closed patterns, let $\mathit{Label}(E) = \{(a_0, \mathrm{last}(M_1, \ldots, M_n)) \mid (a \mapsto a_0[M_1, \ldots, M_n] \in E\}$. Let $\mathit{Label}(\Lambda) = \{(a, \lambda) \mid \lambda \in \Lambda\}$. We say that $E \vdash \Lambda; P$ is well-labelled when the multisets $\mathit{Label}(E_1) \cup \mathit{Label}(\Lambda) \cup \mathit{Label}(P)$ and $\mathit{Label}(E_2) \cup \mathit{Label}(\Lambda) \cup \mathit{Label}(P)$ contain no duplicates, where E_1 and E_2 are the first and second components of E. We say that $E \vdash \Lambda; P$ when $E \vdash \Lambda; P$ is well-labelled and $E \vdash P$. Showing that $\mathit{Label}(E_1)$ and $\mathit{Label}(E_2)$ contain no duplicates guarantees that different terms have different types. More precisely, if E maps names to closed patterns $a[\ldots]$, E is extended to terms as a substitution, and $\mathit{Label}(E)$ contains no duplicates, then we have the following properties:

E1. E is an injection (if $E(M) = E(N)$ then $M = N$) and also an injection modulo Σ (if $\Sigma \vdash E(M) = E(N)$ then $\Sigma \vdash M = N$).

E2. Let N be a term not containing names; if $E(M')$ is an instance of N, then M' is an instance of N; if $E(M')$ is an instance of N modulo Σ, then M' is an instance of N modulo Σ.

E3. If $D \Downarrow_{\Sigma'} M$, then $E(D) \Downarrow_{\Sigma'} E(M)$. (This is proved by induction on D.)

E4. If $\Sigma \vdash D' = E(D)$ and $D' \Downarrow_{\Sigma} p'$ then there exists M such that $\Sigma \vdash p' = E(M)$ and $D \Downarrow_{\Sigma} M$. (This is proved by induction on D, using E2.)

Let $E_0 = \{a \mapsto (a[\,], a[\,]) \mid a \in fn(C'[P'_0])\}$.

1. *Typability of the adversary:* Let P' be a subprocess of C'. Let E be an environment such that for all $a \in fn(P')$, $\mathrm{att}'(E(a), E(a)) \in \mathcal{F}_{C',P_0}$ and for all $x \in fv(P')$, $\mathrm{att}'(E(x), E(x)) \in \mathcal{F}_{C',P_0}$. We show that $E \vdash P'$ by induction on P', similarly to [3, Lemma 5.1.4].

   We detail the case of term evaluations, since it significantly differs from that in [3]. In order to show the desired property in this case, it suffices to show that if for all $a \in fn(D)$, $\mathrm{att}'(E(a), E(a)) \in \mathcal{F}_{C',P_0}$ and for all $x \in fv(D)$, $\mathrm{att}'(E(x), E(x)) \in \mathcal{F}_{C',P_0}$, then we have the two properties:

   (a) if $E(D)_1 \Downarrow_{\Sigma'} p_1$ and $E(D)_2 \Downarrow_{\Sigma'} p_2$, then $\mathrm{att}'(p_1, p_2) \in \mathcal{F}_{C',P_0}$;

   (b) if $E(D)_1 \Downarrow_{\Sigma'} p_1$ and $\nexists p_2, E(D)_2 \Downarrow_{\Sigma} p_2$, then $\mathrm{bad} \in \mathcal{F}_{C',P_0}$; symmetrically, if $E(D)_2 \Downarrow_{\Sigma'} p_2$ and $\nexists p_1, E(D)_1 \Downarrow_{\Sigma} p_1$, then $\mathrm{bad} \in \mathcal{F}_{C',P_0}$.

   The proof is by induction on D.

   - Case $D = \mathsf{diff}[a, a]$: We have $E(D)_1 = E(a)_1 \Downarrow_{\Sigma'} E(a)_1$ and $E(D)_2 = E(a)_2 \Downarrow_{\Sigma'} E(a)_2$, and by hypothesis $\mathrm{att}'(E(a)_1, E(a)_2) \in \mathcal{F}_{C',P_0}$, so Property (a) holds. We also have $E(D)_1 = E(a)_1 \Downarrow_{\Sigma} E(a)_1$ and $E(D)_2 = E(a)_2 \Downarrow_{\Sigma} E(a)_2$, so Property (b) holds.
   - Case $D = x$: This case is similar to that for $D = \mathsf{diff}[a, a]$.
   - Case $D = \mathrm{eval}\, h(D_1, \ldots, D_n)$: Property (a) follows from the induction hypothesis and Clause (Rf). Next, we prove the first part of Property (b). The second part of Property (b) follows by symmetry.

     Since $E(D)_1 \Downarrow_{\Sigma'} p_1$, there exist $h(N_1, \ldots, N_n) \to N$ in $\mathrm{def}_{\Sigma'}(h)$, p_1, $p_{1,1}$, …, $p_{1,n}$, and σ such that $E(D_i)_1 \Downarrow_{\Sigma'} p_{1,i}$ for all $i \in \{1, \ldots, n\}$, $p_1 = \sigma N$, and $p_{1,i} = \sigma N_i$ for all $i \in \{1, \ldots, n\}$. Since there exists no p_2 such that $E(D)_2 \Downarrow_{\Sigma} p_2$, either for some $i \in \{1, \ldots, n\}$ there exists no $p_{2,i}$ such that $E(D_i)_2 \Downarrow_{\Sigma} p_{2,i}$ (and $\mathrm{bad} \in \mathcal{F}_{C',P_0}$ by induction hypothesis), or for all $i \in \{1, \ldots, n\}$ there exists $p_{2,i}$ such that $E(D_i)_2 \Downarrow_{\Sigma} p_{2,i}$, and there exist no $h(N'_1, \ldots, N'_n) \to N'$ in $\mathrm{def}_{\Sigma}(h)$ and σ such that for all $i \in \{1, \ldots, n\}$, $\Sigma \vdash p_{2,i} = \sigma N'_i$. Hence, h must be a destructor.

     By Property S2, there exists an environment E' such that $\Sigma \vdash E'(a) = E(a)$ for all $a \in fn(D)$, $\Sigma \vdash E'(x) = E(x)$ for all $x \in fv(D)$, and

$\mathrm{nf}_{\mathcal{S},\Sigma}(E')$. By Lemma 29, $\mathrm{att}'(E'(a)_1, E'(a)_2) \in \mathcal{F}_{C',P_0}$ for all $a \in fn(D)$ and $\mathrm{att}'(E'(x)_1, E'(x)_2) \in \mathcal{F}_{C',P_0}$ for all $x \in fv(D)$. We have $\mathrm{nf}_{\mathcal{S},\Sigma}(E'(D_i))$ and $\Sigma \vdash E'(D_i)_2 = E(D_i)_2$. By Property S2, there exist $p'_{2,1}, \ldots, p'_{2,n}$ such that $\Sigma \vdash p'_{2,i} = p_{2,i}$ for all $i \in \{1, \ldots, n\}$ and $\mathrm{nf}_{\mathcal{S},\Sigma}(E', p'_{2,1}, \ldots, p'_{2,n})$. By a variant of Lemma 14 for patterns instead of terms, $E'(D_i)_2 \Downarrow_{\Sigma'} p'_{2,i}$ for all $i \in \{1, \ldots, n\}$.

By a variant of Lemma 18 for patterns instead of terms, $E'(D_i)_1 \Downarrow_{\Sigma} p'_{1,i}$ for some $p'_{1,i}$ such that $\Sigma \vdash p'_{1,i} = p_{1,i}$. By Property S2, there exist $p''_{1,1}, \ldots, p''_{1,n}, p''_1$ such that $\Sigma \vdash p''_{1,i} = p'_{1,i}$ for all $i \in \{1, \ldots, n\}$, $\Sigma \vdash p''_1 = p_1$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(E', p'_{2,1}, \ldots, p'_{2,n}, p''_{1,1}, \ldots, p''_{1,n}, p''_1)$. By a variant of Lemma 14 for patterns instead of terms, $E'(D_i)_1 \Downarrow_{\Sigma'} p''_{1,i}$ for all $i \in \{1, \ldots, n\}$. By induction hypothesis, Property (a), we obtain $\mathrm{att}'(p''_{1,i}, p'_{2,i}) \in \mathcal{F}_{C',P_0}$ for all $i \in \{1, \ldots, n\}$.

Since $\Sigma \vdash p'_{2,i} = p_{2,i}$, there exist no σ and $h(N'_1, \ldots, N'_n) \to N'$ in $\mathrm{def}_{\Sigma}(h)$ such that for all $i \in \{1, \ldots, n\}$, $\Sigma \vdash p'_{2,i} = \sigma N'_i$. By Lemma 16, there exist no σ and $h(N'_1, \ldots, N'_n) \to N'$ in $\mathrm{def}_{\Sigma'}(h)$ such that for all $i \in \{1, \ldots, n\}$, $\Sigma \vdash p'_{2,i} = \sigma N'_i$, that is, we have

$$\bigwedge_{h(N'_1,\ldots,N'_n)\to N' \text{ in } \mathrm{def}_{\Sigma'}(h)} \mathrm{nounif}((p'_{2,1}, \ldots, p'_{2,n}), GVar(N'_1, \ldots, N'_n))$$

Since $\Sigma \vdash p''_{1,i} = p_{1,i}$, $\Sigma \vdash p''_1 = p_1$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(p''_{1,1}, \ldots, p''_{1,n}, p''_1)$, by Lemma 16 and a variant of Lemma 13 for patterns instead of terms, there exist $h(N_1, \ldots, N_n) \to N$ is $\mathrm{def}_{\Sigma'}(h)$ and Σ such that $p''_{1,i} = \sigma N_i$ for all $i \in \{1, \ldots, n\}$ and $p''_1 = \sigma N$. Hence, by Clause (Rt), $\mathrm{bad} \in \mathcal{F}_{C',P_0}$.

2. *Typability of P'_0:* We prove by induction on the process P, subprocess of P'_0, that, if (a) ρ binds all free names and variables of P, (b) σ is a closed substitution, (c) $\mathcal{R}_{C',P_0} \supseteq [\![P]\!]\rho H$, and (d) σH can be derived from $\mathcal{R}_{C',P_0}$, then $\sigma\rho \vdash P$.

   Again, we detail the case of term evaluations. We suppose that ρ binds all free names and variables of $let\ x = D\ in\ P\ else\ Q$, σ is a closed substitution, $\mathcal{R}_{C',P_0} \supseteq [\![let\ x = D\ in\ P\ else\ Q]\!]\rho H$, and σH is derivable from $\mathcal{R}_{C',P_0}$. We show that $\sigma\rho \vdash let\ x = D\ in\ P\ else\ Q$. To apply the type rule (Term evaluation), it suffices to show that:

   - For all p_1, p_2 such that $\sigma\rho(D)_1 \Downarrow_{\Sigma'} p_1$ and $\sigma\rho(D)_2 \Downarrow_{\Sigma'} p_2$, we have $\sigma\rho[x \mapsto (p_1, p_2)] \vdash P$.

     By a variant of Lemma 11 for patterns instead of terms, there exist p'_1, p'_2, σ', and σ'' such that $(\rho(D)_1, \rho(D)_2) \Downarrow' ((p'_1, p'_2), \sigma')$, $p_1 = \sigma'' p'_1$, $p_2 = \sigma'' p'_2$, and $\sigma = \sigma''\sigma'$ except on the fresh variables introduced in the computation of $(\rho(D)_1, \rho(D)_2) \Downarrow' ((p'_1, p'_2), \sigma')$.

     Hence $\sigma''\sigma' H = \sigma H$ can be derived from $\mathcal{R}_{C',P_0}$, and $[\![P]\!]((\sigma'\rho)[x \mapsto (p'_1, p'_2)])(\sigma' H) \subseteq [\![let\ x = D\ in\ P\ else\ Q]\!]\rho H \subseteq \mathcal{R}_{C',P_0}$ so, by induction hypothesis, $\sigma''(\sigma'\rho[x \mapsto (p'_1, p'_2)]) \vdash P$, that is, $\sigma\rho[x \mapsto (p_1, p_2)] \vdash P$.
   - If there exists no p_1 such that $\sigma\rho(D)_1 \Downarrow_{\Sigma} p_1$ and there exists no p_2 such that $\sigma\rho(D)_2 \Downarrow_{\Sigma} p_2$, then $\sigma\rho \vdash Q$.

By Lemma 30, $\sigma\rho(\text{fails}(\mathsf{fst}(D)))_1$ and $\sigma\rho(\text{fails}(\mathsf{snd}(D)))_2$ are true, so $\sigma(H \wedge \rho(\text{fails}(\mathsf{fst}(D)))_1 \wedge \rho(\text{fails}(\mathsf{snd}(D)))_2)$ can be derived from $\mathcal{R}_{C',P_0}$. Moreover $[\![Q]\!]\rho(H \wedge \rho(\text{fails}(\mathsf{fst}(D)))_1 \wedge \rho(\text{fails}(\mathsf{snd}(D)))_2) \subseteq [\![let\ x = D\ in\ P\ else\ Q]\!]\rho H \subseteq \mathcal{R}_{C',P_0}$ so, by induction hypothesis, $\sigma\rho \vdash Q$.

- If there exists p_1 such that $\sigma\rho(D)_1 \Downarrow_{\Sigma'} p_1$ and there exists no p_2 such that $\sigma\rho(D)_2 \Downarrow_{\Sigma} p_2$, then $\text{bad} \in \mathcal{F}_{C',P_0}$.

  By a variant of Lemma 11 for patterns instead of terms, there exist p_1', σ', and σ'' such that $\rho(D)_1 \Downarrow' (p_1', \sigma')$, $p_1 = \sigma'' p_1'$, and $\sigma = \sigma''\sigma'$ except on the fresh variables introduced in the computation of $\rho(D)_1 \Downarrow' (p_1', \sigma')$. There exists no p_2 such that $\sigma''\sigma'\rho(D)_2 \Downarrow_{\Sigma} p_2$, so by Lemma 30, $\sigma''\sigma'\rho(\text{fails}(\mathsf{snd}(D)))_2$ holds, hence $\sigma''(\sigma' H \wedge \sigma'\rho(\text{fails}(\mathsf{snd}(D)))_2)$ can be derived from $\mathcal{R}_{C',P_0}$. Since $\sigma' H \wedge \sigma'\rho(\text{fails}(\mathsf{snd}(D)))_2 \rightarrow \text{bad} \in [\![let\ x = D\ in\ P\ else\ Q]\!]\rho H \subseteq \mathcal{R}_{C',P_0}$, $\text{bad} \in \mathcal{F}_{C',P_0}$.

- If there exists no p_1 such that $\sigma\rho(D)_1 \Downarrow_{\Sigma} p_1$ and there exists p_2 such that $\sigma\rho(D)_2 \Downarrow_{\Sigma'} p_2$, then $\text{bad} \in \mathcal{F}_{C',P_0}$. This property follows from the one above by symmetry.

By definition, $\mathcal{R}_{C',P_0} \supseteq [\![P_0']\!]\rho_0\emptyset$, where $\rho_0 = \{a \mapsto (a[\,], a[\,]) \mid a \in fn(P_0')\}$. Taking $P = P_0'$, we obtain $E \vdash P_0'$ with $E = \sigma\rho_0 = \{a \mapsto (a[\,], a[\,]) \mid a \in fn(P_0')\}$. (This result is similar to [3, Lemma 7.2.2].)

3. *Properties of $C'[P_0']$:* We show that $E_0 \vdash \Lambda_0; C'[P_0']$. In order to prove this result, we show that $E_0 \vdash C'[P_0']$ by induction on C'.

   When $C' = [\,]$, the result follows from Property 2. When $C' = (\nu a : a[\,])C''$, the result follows by induction hypothesis and the type rule (Restriction). When $C' = C'' \mid Q$, the result follows from Property 1 and the type rule (Parallel).

4. *Substitution lemma:* Let $E' = E[x \mapsto (E(M)_1, E(M)_2)]$. We show by induction on M' that $E(M'\{M/x\}) = E'(M')$. We show by induction on P that, if $E' \vdash P$, then $E \vdash P\{M/x\}$. This is similar to [3, Lemma 5.1.1].

5. *Subject congruence:* If $E \vdash \Lambda; P$ and $P \equiv P'$, then $E \vdash \Lambda; P'$. We prove by induction on the derivation of $P \equiv P'$ that if $E \vdash P$ and $P \equiv P'$, then $E \vdash P'$ and $Label(P') = Label(P)$, similarly to [3, Lemma 5.1.2].

6. *Subject reduction:* If $E \vdash \Lambda; P$ and $\Lambda; P \rightarrow \Lambda'; P'$, then $E \vdash \Lambda', P'$. We prove by induction on the derivation of $\Lambda; P \rightarrow \Lambda'; P'$ that if $E \vdash \Lambda; P$ and $\Lambda; P \rightarrow \Lambda'; P'$, then $E \vdash \Lambda', P'$ and $Label(\Lambda') \cup Label(P') \subseteq Label(\Lambda) \cup Label(P)$, similarly to [3, Lemma 5.1.3].

7. *Proof of the second hypothesis of Lemma 2:* Assume that

$$\text{unevaluated}(C[P_0]) \rightarrow^*_{\Sigma',\Sigma} \equiv C_1[let\ y = D\ in\ Q\ else\ Q']$$

   and $\mathsf{fst}(D) \Downarrow_{\Sigma'} M_1$ for some M_1. Then $\Lambda_0; C'[P_0'] \rightarrow^*_{\Sigma',\Sigma} \equiv \Lambda; C_1'[let\ y = D\ in\ Q_1\ else\ Q_1']$ where $\text{delete}(C_1'[let\ y = D\ in\ Q_1\ else\ Q_1']) = C_1[let\ y = D\ in\ Q\ else\ Q']$. We have $E_0 \vdash \Lambda_0; C'[P_0']$, so by subject reduction and subject

congruence, $E_0 \vdash \Lambda; C'_1[\mathit{let}\ y = D\ \mathit{in}\ Q_1\ \mathit{else}\ Q'_1]$. Since $E_0 \vdash C'_1[\mathit{let}\ y = D\ \mathit{in}\ Q_1\ \mathit{else}\ Q'_1]$ has been derived by type rules (Restriction) and (Parallel), there exists an environment E such that $E \vdash \mathit{let}\ y = D\ \mathit{in}\ Q_1\ \mathit{else}\ Q'_1$ and since $\mathit{Label}((E_0)_1) \cup \mathit{Label}(\Lambda) \cup \mathit{Label}(C'_1[\mathit{let}\ y = D\ \mathit{in}\ Q_1\ \mathit{else}\ Q'_1])$ and $\mathit{Label}((E_0)_2) \cup \mathit{Label}(\Lambda) \cup \mathit{Label}(C'_1[\mathit{let}\ y = D\ \mathit{in}\ Q_1\ \mathit{else}\ Q'_1])$ contain no duplicates, $\mathit{Label}(E_1)$ and $\mathit{Label}(E_2)$ contain no duplicates.

Since $\mathsf{fst}(D) \Downarrow_{\Sigma'} M_1$, by Property E3, $E(D)_1 \Downarrow_{\Sigma'} E(M_1)_1$. Since $E \vdash \mathit{let}\ y = D\ \mathit{in}\ Q_1\ \mathit{else}\ Q'_1$ has been derived by type rule (Term evaluation) and $\mathrm{bad} \notin \mathcal{F}_{C',P_0}$, there exists p_2 such that $E(D)_2 \Downarrow_\Sigma p_2$. So by Property E4, there exists M_2 such that $\mathsf{snd}(D) \Downarrow_\Sigma M_2$, which establishes the second hypothesis of Lemma 2.

8. *Proof of the first hypothesis of Lemma 2:* Assume that

$$\mathrm{unevaluated}(C[P_0]) \rightarrow^*_{\Sigma',\Sigma\equiv} C_1[\overline{N}\langle M\rangle.Q \mid N'(x).R]$$

and $\mathsf{fst}(N) = \mathsf{fst}(N')$. As above, there exists an environment E such that $E \vdash \overline{N}\langle M\rangle.Q' \mid N'(x).R'$ and E_1 and E_2 satisfy Properties E1, E2, E3, and E4.

Since $E \vdash \overline{N}\langle M\rangle.Q' \mid N'(x).R'$ has been derived by type rules (Parallel), (Output), and (Input), we have $\mathrm{msg}'(E(N)_1, E(M)_1, E(N)_2, E(M)_2) \in \mathcal{F}_{C',P_0}$ and $\mathrm{input}'(E(N')_1, E(N')_2) \in \mathcal{F}_{C',P_0}$. Since $\mathsf{fst}(N) = \mathsf{fst}(N')$, $E(N)_1 = E(N')_1$. Since bad is not derivable from $\mathcal{R}_{C',P_0}$, $\mathrm{nounif}(E(N)_2, E(N')_2)$ is false—otherwise bad would be derivable by (Rcom)—so, by definition of nounif, $\Sigma \vdash E(N)_2 = E(N')_2$. By Property E1, E_2 is injective modulo Σ and we obtain $\Sigma \vdash \mathsf{snd}(N') = \mathsf{snd}(N)$.

The symmetric hypotheses of Lemma 2 follow by symmetry.

To conclude our proof of Theorem 3, we apply Lemma 2 and Corollary 1. □

E Proof of Theorem 4

E.1 Unification modulo the equational theory

We use the standard convention that, when computing a most general unifier σ_u of M_i, N_i for $i \in \{1, \ldots, n\}$, we always arrange that $dom(\sigma_u) \cap fv(im(\sigma_u)) = \emptyset$ and $dom(\sigma_u) \cup fv(\sigma_u M_1, \sigma_u N_1, \ldots, \sigma_u M_n, \sigma_u N_n) \subseteq \cup_i (fv(M_i) \cup fv(N_i))$. (We recall that $dom(\sigma) = \{x \mid x \neq \sigma x\}$.) Since $dom(\sigma_u) \cap fv(im(\sigma_u)) = \emptyset$, σ_u is idempotent.

If σ is a most general unifier of M_i, N_i for $i \in \{1, \ldots, n\}$ and σ' is a most general unifier of $\sigma M'_i, \sigma N'_i$ for $i \in \{1, \ldots, n'\}$ then $\sigma'\sigma$ is a most general unifier of M_i, N_i for $i \in \{1, \ldots, n\}$ and M'_i, N'_i for $i \in \{1, \ldots, n'\}$.

Lemma 31 *If $\sigma D \Downarrow' (M', \sigma')$ and σ is a most general unifier, then $\sigma'\sigma$ is also a most general unifier, and there exists M'' such that $M' = \sigma'\sigma M''$.*

Proof The proof is by mutual induction following the definition of $\Downarrow'$. All cases are easy. □

Lemma 32 *We have $\Sigma \vdash \sigma M = \sigma M'$ if and only if there exist N, N', σ', and σ_u such that* $\mathrm{addeval}(M, M') \Downarrow' ((N, N'), \sigma')$*, σ_u is the most general unifier of N and N', and for all $x \in fv(M, M')$, $\Sigma \vdash \sigma x = \sigma\sigma_u\sigma' x$.*

Proof Assume $\Sigma \vdash \sigma M = \sigma M'$. By Property S2, there exist M'' and σ' such that $\Sigma \vdash M'' = \sigma M = \sigma M'$, $\Sigma \vdash \sigma x = \sigma' x$ for all $x \in fv(M, M')$, and $\mathrm{nf}_{\mathcal{S},\Sigma}(\{M''\} \cup \{\sigma' x \mid x \in fv(M, M')\})$. Since $\Sigma \vdash \sigma' M = \sigma' M' = M''$, by Lemma 12 we have $\sigma'\mathrm{addeval}(M) \Downarrow_{\Sigma'} M''$ and $\sigma'\mathrm{addeval}(M') \Downarrow_{\Sigma'} M''$. By Lemma 11, there exist $N, N', \sigma_1, \sigma_1'$ such that $\mathrm{addeval}(M, M') \Downarrow' ((N, N'), \sigma_1)$, $M'' = \sigma_1' N$, $M'' = \sigma_1' N'$, and $\sigma' = \sigma_1'\sigma_1$ except on fresh variables introduced in the computation of $\mathrm{addeval}(M, M') \Downarrow' ((N, N'), \sigma_1)$.

So N and N' unify. Let σ_u be their most general unifier. Let σ_1'' such that $\sigma_1' = \sigma_1''\sigma_u$. Let $x \in fv(M, M')$. We have $\Sigma \vdash \sigma x = \sigma' x = \sigma_1''\sigma_u\sigma_1 x = \sigma_1''\sigma_u\sigma_1\sigma_u\sigma_1 x = \sigma'\sigma_u\sigma_1 x = \sigma\sigma_u\sigma_1 x$. (Indeed, $\sigma_u\sigma_1$ is a most general unifier by Lemma 31 and the composition of most general unifiers, so it is idempotent.)

Conversely, assume that there exist $N, N', \sigma'\sigma_u$ such that $\mathrm{addeval}(M, M') \Downarrow' ((N, N'), \sigma')$, σ_u is the most general unifier of N and N' and for all $x \in fv(M, M')$, $\Sigma \vdash \sigma x = \sigma\sigma_u\sigma' x$. Then

$$
\begin{aligned}
\Sigma \vdash \sigma M &= \sigma\sigma_u\sigma' M \\
&= \sigma\sigma_u N && \text{by Lemma 15} \\
&= \sigma\sigma_u N' && \text{since } \sigma_u \text{ is the most general unifier of } N \text{ and } N' \\
&= \sigma\sigma_u\sigma' M' && \text{by Lemma 15 again} \\
&= \sigma M' && \square
\end{aligned}
$$

E.2 Soundness of the solving algorithm

The following proofs are partly adaptations of previous proofs [15, 18]. In addition, they establish the soundness of all simplifications for nounif.

Let $\mathcal{R}_0 = \mathcal{R}_{P_0}$ be the initial set of clauses, $\mathcal{R}_1 = \mathsf{saturate}(\mathcal{R}_0)$ be the final set of clauses, and $\mathcal{R}$ be the set of clauses during the saturation. At the end of the saturation algorithm, we have $\mathcal{R}_1 = \{R \in \mathcal{R} \mid \mathsf{sel}(R) = \emptyset\}$.

Lemma 33 *At the end of the saturation, $\mathcal{R}$ satisfies the following properties:*

1. *For all $R \in \mathit{simplify}(\mathcal{R}_0)$, there exists $R' \in \mathcal{R}$ such that $R' \sqsupseteq R$.*
2. *Let $R \in \mathcal{R}$ and $R' \in \mathcal{R}$. Assume that $\mathsf{sel}(R) = \emptyset$ and there exists $F_0 \in \mathsf{sel}(R')$ such that $R \circ_{F_0} R'$ is defined. In this case, for all $R'' \in \mathit{simplify}(R \circ_{F_0} R')$, there exists $R''' \in \mathcal{R}$ such that $R''' \sqsupseteq R''$.*

Proof To prove the first property, let $R \in \mathit{simplify}(\mathcal{R}_0)$. We show that during the whole execution of the saturation, there exists $R' \in \mathcal{R}$ such that $R' \sqsupseteq R$.

The algorithm first builds $\mathit{simplify}(\mathcal{R}_0)$ (which obviously satisfies the required property), then removes subsumed clauses by $\mathit{condense}$. The property is preserved by

elimination of subsumed clauses. So $\mathcal{R} = condense(\mathcal{R}_0)$ satisfies the property. Further additions of clauses and eliminations of subsumed clauses preserve the property, so we have the result.

The second property states that the fixpoint is reached at the end of saturation. □

We now give a precise definition of derivations.

Definition 7 (Derivation) Let $\mathcal{T}_{\text{facts}}$ be the set of true nounif facts. Let $\mathcal{R}$ be a set of clauses and F be a closed fact. A derivation of F from $\mathcal{R}$ is a finite tree defined as follows:

1. Nodes (except the root) are labelled by clauses $R \in \mathcal{R}$ or nounif facts in $\mathcal{T}_{\text{facts}}$.
2. Edges are labelled by closed facts. (Edges go from a node to each of its sons.)
3. The root has one outgoing edge, labelled by F.
4. If the tree contains a node labelled by R with one incoming edge labelled by F_0 and n outgoing edges labelled by $F_1, \ldots, F_n$, then $R \sqsupseteq \{F_1, \ldots, F_n\} \rightarrow F_0$. If the tree contains a node labelled by a fact in $\mathcal{T}_{\text{facts}}$, then this node has one incoming edge labelled by the same fact and no outgoing edge.

In a derivation, if there is a node labelled by R with one incoming edge labelled by F_0 and n outgoing edges labelled by $F_1, \ldots, F_n$, then the clause R can be used to infer F_0 from $F_1, \ldots, F_n$. Therefore, there exists a derivation of F from $\mathcal{R}$ if and only if F can be inferred from clauses in $\mathcal{R}$.

The key idea of the proof of the algorithm is the following. Assume that bad is derivable from $\mathcal{R}_0$ and consider a derivation of bad from $\mathcal{R}_0$. Assume that the clauses R and R' are applied one after the other in the derivation of bad. Also assume that these clauses have been combined by $R \circ_{F_0} R'$, yielding clause R''. In this case, we replace R' by R'' in the derivation of bad. When no more replacement can be made, we show that all remaining clauses have no selected hypothesis. Then all these clauses are in $\mathcal{R}_1 = \mathsf{saturate}(\mathcal{R}_0)$, and we have built a derivation of bad from $\mathcal{R}_1$. Moreover, this replacement process terminates because the number of nodes of the derivation strictly decreases.

Lemma 34 *Consider a derivation that contains a node η', labelled R'. Let F_0 be a hypothesis of R'. Then there exists a son η of η', labelled R, such that the edge from η' to η is labelled by an instance of F_0, $R \circ_{F_0} R'$ is defined, and we still have a derivation of the same fact if we replace the nodes η and η' by a node η'' labelled $R'' = R \circ_{F_0} R'$.*

Proof This proof is already given in [18], with a figure. Let $R' = H' \rightarrow C'$, H'_1 be the multiset of the labels of the outgoing edges of η', and C'_1 the label of its incoming edge. We have $R' \sqsupseteq (H'_1 \rightarrow C'_1)$, then there exists σ such that $\sigma H' \subseteq H'_1$ and $\sigma C' = C'_1$. Then there is an outgoing edge of η' labelled σF_0, since $\sigma F_0 \in H'_1$. Let η be the node at the end of this edge, let $R = H \rightarrow C$ be the label of η. We rename the variables of R so that they are distinct from the variables of R'. Let H_1 be the multiset of the labels

of the outgoing edges of η. Then $R \sqsupseteq (H_1 \to \sigma F_0)$. By the above choice of distinct variables, we can then extend σ in such a way that $\sigma H \subseteq H_1$ and $\sigma C = \sigma F_0$.

The edge from η' to η is labelled σF_0, which is an instance of F_0. We have $\sigma C = \sigma F_0$, then C and F_0 are unifiable, then $R \circ_{F_0} R'$ is defined. Let σ' be the most general unifier of C and F_0, and σ'' such that $\sigma = \sigma''\sigma'$. We have $R \circ_{F_0} R' = \sigma'(H \cup (H' - F_0)) \to \sigma' C'$. Moreover, $\sigma''\sigma'(H \cup (H' - F_0)) \subseteq H_1 \cup (H'_1 - \sigma F_0)$ and $\sigma''\sigma' C' = \sigma C' = C'_1$. Then $R'' = R \circ_{F_0} R' \sqsupseteq (H_1 \cup (H'_1 - \sigma F_0)) \to C'_1$. The multiset of labels of outgoing edges of η'' is precisely $H_1 \cup (H'_1 - \sigma F_0)$ and the label of its incoming edge is C'_1, so we have obtained a correct derivation by replacing η and η' with η''. □

Lemma 35 *If $\mathcal{D}$ is a derivation whose node η is labelled R, then we obtain a derivation $\mathcal{D}'$ of the same fact by relabelling η with a clause R' such that $R' \sqsupseteq R$.*

Proof Let H be the multiset of labels of outgoing edges of the considered node η, and C be the label of its incoming edge. We have $R \sqsupseteq H \to C$. By transitivity of $\sqsupseteq$, $R' \sqsupseteq H \to C$. So we can relabel η with R'. □

We now prove the soundness of each simplification function described in Section 7, and of their composition $simplify$.

Lemma 36 *Let f range over the simplification functions $simpeq$, $elimvar$, $elimGVar \circ swap \circ unify$, $elimnouniffalse$, $elimdup$, $elimattx$, $elimtaut$, and $simplify$.*

Let $\mathcal{R}_t = \{(1), (2)\}$ when $f \in \{elimvar, simplify\}$ and $\mathcal{R}_t = \emptyset$ otherwise.

Let $\mathcal{D}$ be a derivation of bad *such that* $\mathrm{nf}'_{\mathcal{S},\Sigma}(\mathcal{D})$ *with a node η labelled R.*

We obtain a derivation $\mathcal{D}'$ of bad *by relabelling the node η with some clause $R' \in f(\{R\}) \cup \mathcal{R}_t$, deleting nodes, and modifying nodes labelled by a fact in $\mathcal{T}_{\mathrm{facts}}$.*

The set of clauses $\mathcal{R}_t$ collects clauses that must be included in the clause set for the transformation to be correct. The proofs closely follow the intuitions for soundness given in Section 7.

Proof (for *simpeq***)** Since $\mathrm{nf}'_{\mathcal{S},\Sigma}(\mathcal{D})$, the facts of σR (except nounif facts) are irreducible by $\mathcal{S}$, so a fortiori the facts of R (except nounif facts) are irreducible by $\mathcal{S}$, hence $simpeq(\{R\}) = \{R\}$, which obviously implies the desired result. □

Proof (for *elimvar***)** Let $R = H \to C$, where $H = \mathrm{att}'(x, y) \wedge \mathrm{att}'(x, y') \wedge \ldots$ and $R' = R\{y/y'\}$. (The case $H = \mathrm{att}'(y, x) \wedge \mathrm{att}'(y', x) \wedge \ldots$ is symmetric.) Let H' be the multiset of labels of outgoing edges of η and C' the label of its incoming edge. Since $\mathcal{D}$ is a derivation, there exists σ such that $\sigma H \subseteq H'$, and $\sigma C = C'$.

- Assume $\Sigma \vdash \sigma y = \sigma y'$. Since we have $\mathrm{nf}'_{\mathcal{S},\Sigma}(\mathcal{D})$, $\sigma y = \sigma y'$. Then $\sigma R' = \sigma R$, so $\mathcal{D}'$ obtained from $\mathcal{D}$ by relabelling η with R' is a derivation.
- Otherwise, $\Sigma \vdash \sigma y \neq \sigma y'$ and thus $\sigma\mathrm{nounif}(y, y') \in \mathcal{T}_{\mathrm{facts}}$. Let $\mathcal{D}'$ be obtained by relabelling the node η with the clause $\mathrm{att}'(x, y) \wedge \mathrm{att}'(x, y') \wedge \mathrm{nounif}(y, y') \to$ bad (1), adding the son $\sigma\mathrm{nounif}(y, y')$, and returning the subtree with root η.

Since $\mathrm{att}'(x, y) \in H$, we have $\sigma\mathrm{att}'(x, y) \in \sigma H \subseteq H'$, and similarly for $\mathrm{att}'(x, y')$. Thus, $\mathcal{D}'$ is a derivation of bad. □

Proof (for *elimGVar* ∘ *swap* ∘ *unify***)** Let $R = H \wedge F \to C$ be the clause modified by $elimGVar \circ swap \circ unify$. We show that if $\sigma F \in \mathcal{T}_{\mathrm{facts}}$, then $elimGVar \circ swap \circ unify$ replaces R with $R' = H \wedge F'_1 \wedge \ldots \wedge F'_n \to C$, and $\sigma F'_1, \ldots, \sigma F'_n \in \mathcal{T}_{\mathrm{facts}}$.

It is easy to infer the lemma from this property. Indeed, let H' be the multiset of labels of outgoing edges of η and C' the label of its incoming edge. Since $\mathcal{D}$ is a derivation, there exists σ such that $\sigma H \wedge \sigma F \subseteq H'$, and $\sigma C = C'$. Then σF is derived by a son of η, so $\sigma F \in \mathcal{T}_{\mathrm{facts}}$. Then by the above property $\sigma F'_1, \ldots, \sigma F'_n \in \mathcal{T}_{\mathrm{facts}}$, and $\mathcal{D}'$ obtained from $\mathcal{D}$ by relabelling η with $R' = H \wedge F'_1 \wedge \ldots \wedge F'_n \to C$ and replacing σF with $\sigma F'_1, \ldots, \sigma F'_n$ as sons of η is also a derivation.

- We now prove that *unify* replaces $F = \mathrm{nounif}(p, p')$ with $F'_1 \wedge \ldots \wedge F'_n$ such that, if $\sigma F \in \mathcal{T}_{\mathrm{facts}}$, then $\sigma F'_1, \ldots, \sigma F'_n \in \mathcal{T}_{\mathrm{facts}}$.

  By definition of nounif, $\sigma F \in \mathcal{T}_{\mathrm{facts}}$ if and only if there exists no closed substitution σ' with domain $GVar$ such that $\Sigma \vdash \sigma'\sigma p = \sigma'\sigma p'$. By Lemma 32, $\Sigma \vdash \sigma'\sigma p = \sigma'\sigma p'$ if and only if there exist $N, N', \sigma'', \sigma_u$ such that $\mathrm{addeval}(p, p') \Downarrow' ((N, N'), \sigma'')$, σ_u is the most general unifier of N and N', and for all $x \in fv(M, M')$, $\Sigma \vdash \sigma\sigma' x = \sigma\sigma'\sigma_u\sigma'' x$. The fact F is replaced with $F'_1, \ldots, F'_n$, where $F'_j = \mathrm{nounif}(p_j, p'_j) = \mathrm{nounif}((x^j_1, \ldots, x^j_{k_j}), \sigma_u\sigma''(x^j_1, \ldots, x^j_{k_j}))$ for each $\sigma_u\sigma''$ obtained as above. So $\Sigma \vdash \sigma'\sigma p = \sigma'\sigma p'$ if and only if there exists $j \in \{1, \ldots, n\}$ such that $\Sigma \vdash \sigma'\sigma p_j = \sigma'\sigma p'_j$. So $\sigma F \in \mathcal{T}_{\mathrm{facts}}$ if and only if $\sigma F'_1, \ldots, \sigma F'_n \in \mathcal{T}_{\mathrm{facts}}$. This equivalence implies the result.

- Next, we show that *swap* replaces $F = \mathrm{nounif}(p_1, p_2)$ with $F' = \mathrm{nounif}(p'_1, p'_2)$ such that, if $\sigma F \in \mathcal{T}_{\mathrm{facts}}$, then $\sigma F' \in \mathcal{T}_{\mathrm{facts}}$.

  We can easily show that for all σ' with domain $GVar \cup Var$, $\Sigma \vdash \sigma' p_1 = \sigma' p_2$ if and only if $\Sigma \vdash \sigma' p'_1 = \sigma' p'_2$. This equivalence yields the result.

- Finally, we show that $elimGVar$ replaces $F = \mathrm{nounif}((\mathsf{g}, p_1, \ldots, p_n), (p'_0, \ldots, p'_n))$ (where $\mathsf{g} \in GVar$) with $F' = \mathrm{nounif}((p_1, \ldots, p_n), (p'_1, \ldots, p'_n))$ such that, if $\sigma F \in \mathcal{T}_{\mathrm{facts}}$, then $\sigma F' \in \mathcal{T}_{\mathrm{facts}}$.

  Assume $\sigma F \in \mathcal{T}_{\mathrm{facts}}$. Then there exists no σ' with domain $GVar$ such that $\Sigma \vdash \sigma'\sigma(\mathsf{g}, p_1, \ldots, p_n) = \sigma'\sigma(p'_0, \ldots, p'_n)$. So there exists no σ'_1 such that $\Sigma \vdash \sigma'_1\sigma(p_1, \ldots, p_n) = \sigma'_1\sigma(p'_1, \ldots, p'_n)$. Indeed, if σ'_1 existed, $\sigma' = \sigma'_1\{\sigma p'_0/\mathsf{g}\}$ would contradict the non-existence of σ'. (Note that g does not occur elsewhere in F, because F is obtained after applying *unify* and *swap*.) Then $\sigma F' \in \mathcal{T}_{\mathrm{facts}}$. □

Proof (for *elimnouniffalse***)** Let $F = \mathrm{nounif}((), ())$. For all σ, $\sigma F \notin \mathcal{T}_{\mathrm{facts}}$. So $R = H \wedge F \to C$ cannot be the label of a node in a derivation $\mathcal{D}$. (Hence *elimnouniffalse* may harmlessly remove R.) □

Proof (for *elimdup***)** The result is obvious: the hypotheses of R' are included in the hypotheses of R, so $R' \sqsupseteq R$. □

Proof (for $elimattx$**)** The result is obvious: the hypotheses of R' are included in the hypotheses of R, so $R' \sqsupseteq R$. □

Proof (for $elimtaut$**)** The result is obvious: we remove η and replace it with one of its subtrees. □

Proof (for $simplify$**)** We apply Lemma 36 for every simplification function that defines $simplify$. □

Theorem 5 *If* $\mathsf{saturate}(\mathcal{R}_0)$ *terminates and there is a derivation* $\mathcal{D}$ *of* bad *from* $\mathcal{R}_0$ *with* $\mathrm{nf}'_{\mathcal{S},\Sigma}(\mathcal{D})$*, then there is a derivation* $\mathcal{D}'$ *of* bad *from* $\mathsf{saturate}(\mathcal{R}_0)$ *with* $\mathrm{nf}'_{\mathcal{S},\Sigma}(\mathcal{D}')$.

The key idea of the proof is to replace clauses as allowed by the previous lemmas. When the replacement terminates, we can show that all clauses are in $\mathsf{saturate}(\mathcal{R}_0)$. We show the termination using the decrease of the number of nodes of the derivation not in $\mathcal{T}_{\mathrm{facts}}$.

Proof Let us consider a derivation $\mathcal{D}$ of bad from $\mathcal{R}_0$ such that $\mathrm{nf}'_{\mathcal{S},\Sigma}(\mathcal{D})$. (The property $\mathrm{nf}'_{\mathcal{S},\Sigma}(\mathcal{D})$ is preserved by the transformations of the derivation described below: these transformations do not introduce new non-nounif intermediately derived facts.)

For each clause R in $\mathcal{R}_0$, for each $R'' \in \mathit{simplify}(R)$, there exists a clause R' in $\mathcal{R}$ such that $R' \sqsupseteq R''$ (Lemma 33, Property 1). Assume that there exists a node labelled R in this derivation. By Lemma 36, we can replace R with some $R'' \in \mathit{simplify}(R) \cup \{(1), (2)\}$. Clauses (1) and (2) are subsumed by some clause in $\mathcal{R}$, since they are obtained by simplification from (Rt), resp. (Rt′) for $g = \mathrm{equals}$. So, in all cases, there exists $R' \in \mathcal{R}$ such that $R' \sqsupseteq R''$. By Lemma 35, we can replace R'' with R'. Therefore, we can replace nodes labelled by R with nodes labelled by R'. This way, we obtain a derivation of bad from $\mathcal{R}$.

Next, we build a derivation of bad from $\mathcal{R}_1$, where $\mathcal{R}_1 = \mathsf{saturate}(\mathcal{R}_0)$.

Consider a derivation $\mathcal{D}$ of bad from $\mathcal{R}$ such that $\mathrm{nf}'_{\mathcal{S},\Sigma}(\mathcal{D})$. If $\mathcal{D}$ contains a node labelled by a clause not in $\mathcal{R}_1 \cup \mathcal{T}_{\mathrm{facts}}$, we can transform $\mathcal{D}$ as follows. Let η' be a lowest node of $\mathcal{D}$ labelled by a clause not in $\mathcal{R}_1 \cup \mathcal{T}_{\mathrm{facts}}$. Then all sons of η' are labelled by clauses in $\mathcal{R}_1 \cup \mathcal{T}_{\mathrm{facts}}$. Let R' be the clause labelling η'. Since $R' \notin \mathcal{R}_1 \cup \mathcal{T}_{\mathrm{facts}}$, $\mathsf{sel}(R') \neq \emptyset$. Take $F_0 \in \mathsf{sel}(R')$. By Lemma 34, there exists a son η of η' labelled R, such that $R \circ_{F_0} R'$ is defined. Since all sons of η' are labelled by clauses in $\mathcal{R}_1 \cup \mathcal{T}_{\mathrm{facts}}$, $R \in \mathcal{R}_1 \cup \mathcal{T}_{\mathrm{facts}}$. Moreover, by definition of the selection function, F_0 is not a nounif fact, so $R \notin \mathcal{T}_{\mathrm{facts}}$, so $R \in \mathcal{R}_1$, hence $\mathsf{sel}(R) = \emptyset$ and $R \in \mathcal{R}$. By Lemma 34, we can replace η and η' with η'' labelled by $R \circ_{F_0} R'$. By Lemma 36, we can replace $R \circ_{F_0} R'$ with some $R''' \in \mathit{simplify}(R \circ_{F_0} R') \cup \{(1), (2)\}$. By Lemma 33, Property 2, for each $R''' \in \mathit{simplify}(R \circ_{F_0} R')$, there exists $R'' \in \mathcal{R}$ such that $R'' \sqsupseteq R'''$; as noted above, this is also true for (1) and (2) so for all $R''' \in \mathit{simplify}(R \circ_{F_0} R') \cup \{(1), (2)\}$, there exists $R'' \in \mathcal{R}$ such that $R'' \sqsupseteq R'''$. By Lemma 35, we can replace R''' with R'', and we obtain a derivation $\mathcal{D}'$ of bad from $\mathcal{R}$, such that $\mathrm{nf}'_{\mathcal{S},\Sigma}(\mathcal{D}')$ and $\mathcal{D}'$ contains fewer nodes not in $\mathcal{T}_{\mathrm{facts}}$ as $\mathcal{D}$ (since the resolution of two clauses removes one node, and simplifications do not add nodes not in $\mathcal{T}_{\mathrm{facts}}$).

Since the number of nodes not in $\mathcal{T}_{\mathrm{facts}}$ strictly decreases, this transformation process terminates.

When we cannot perform this transformation any more, all nodes of the derivation are labelled by clauses in $\mathcal{R}_1 \cup \mathcal{T}_{\mathrm{facts}}$, hence we have obtained a derivation $\mathcal{D}'$ of bad from $\mathcal{R}_1$ such that $\mathrm{nf}'_{\mathcal{S},\Sigma}(\mathcal{D}')$. □

Proof (of Theorem 4) If bad is derivable from $\mathcal{R}_{P_0}$ then it is derivable from $\mathcal{R}_{P_0}$ by a derivation that satisfies $\mathrm{nf}'_{\mathcal{S},\Sigma}$ (by Lemma 3), then it is derivable from $\mathsf{saturate}(\mathcal{R}_{P_0})$ by a derivation that satisfies $\mathrm{nf}'_{\mathcal{S},\Sigma}$ (by Theorem 5), then $\mathsf{saturate}(\mathcal{R}_{P_0})$ contains a clause of the form $H \to \mathrm{bad}$. □