On the Symbolic Reduction of Processes with Cryptographic Functions

Roberto M. Amadio, Denis Lugiez, Vincent Vanackère

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

On the symbolic reduction of processes with cryptographic functions

Roberto M. Amadio   Denis Lugiez   Vincent Vanackère

N° 4147

Mars 2001

THÈME 1

Rapport de recherche

ISSN 0249-6399   ISRN INRIA/RR--4147--FR

On the symbolic reduction of processes with cryptographic functions

Roberto M. Amadio   Denis Lugiez   Vincent Vanackère

Thème 1 — Réseaux et systèmes
Projet MIMOSA

Rapport de recherche n° 4147 — Mars 2001 — 57 pages

Abstract: We study the reachability problem for cryptographic protocols represented as processes relying on *perfect* cryptographic functions. We introduce a *symbolic* reduction system that can handle hashing functions, symmetric keys, and public keys. Desirable properties such as secrecy or authenticity are *specified* by inserting *logical assertions* in the processes.

We show that the symbolic reduction system provides a flexible decision procedure for *finite* processes and a reference for sound implementations. The symbolic reduction system can be regarded as a variant of syntactic unification which is compatible with certain set-membership constraints. For a significant fragment of our formalism, we argue that a *dag* implementation of the symbolic reduction system leads to an algorithm running in NPTIME thus matching the lower bound of the problem.

In the case of *iterated or finite control* processes, we show that the problem is undecidable in general and in PTIME for a subclass of iterated processes that do not rely on pairing. Our technique is based on rational transductions of regular languages and it applies to a class of processes containing the *ping-pong protocols* presented in [DEK82].

Key-words: cryptographic protocols, verification, symbolic computation

The authors work at *Centre de Mathémathiques et d'Informatique* (LIM-CNRS), 39 rue Joliot-Curie, F-13453, Marseille, France. e-mail: {amadio,lugiez,vvanacke}@cmi.univ-mrs.fr and are partially supported by Action VERNAM. The first author and third authors are members of *Projet MIMOSA*. Instances of the symbolic reduction method studied here were presented in [AL00] and [Van00].

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, B.P. 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : 04 92 38 77 77 - International : +33 4 92 38 77 77 — Fax : 04 92 38 77 65 - International : +33 4 92 38 77 65

Sur la réduction symbolique de processus avec fonctions cryptographiques

Résumé : Nous étudions le problème d'accessibilité pour les protocoles cryptographiques représentés comme des processus utilisant des fonctions cryptographiques *parfaites.* Nous introduisons un système de réduction *symbolique* qui peut traiter les fonctions de hachage, les clefs symétriques et les clefs publiques. Les propriétés voulues comme le secret et l'authentification peuvent être spécifiées en insérant des *assertions logiques* dans les processus.

Nous montrons que le système de réduction symbolique fournit une procédure de décision souple pour les processus *finis* et une spécification pour des implémentations correctes. Le système de réduction symbolique peut être considéré comme une variante de l'unification syntaxique qui est compatible avec certaines contraintes ensemblistes. Pour un fragment significatif de notre formalisme, nous présentons une implémentation du système de réduction symbolique basée sur les graphes dirigés acycliques. Ceci donne un algorithme dont la complexité est en temps polynomial non-déterministe, ce qui correspond à la borne inférieure du problème.

Dans le cas de processus *itérés ou à contrôle fini*, nous montrons que le problème est undecidable en général et qu'il est résoluble en temps polynomial déterministe pour une sous-classe de processus itérés qui n'utilisent pas les couples. Notre technique est basée sur les transductions rationnelles de langages rationnels et elle s'applique à une classe de processus qui contient les protocoles *ping-pong* présentés dans [DEK82].

Mots-clés : protocoles cryptographiques, vérification, calcul symbolique.

Contents

1 Introduction

Protocols based on cryptographic operations are used to protect the access to computer systems and the transmission of data on networks. Other application domains such as *smart cards* are emerging.

The experience has shown that the design of these protocols is error-prone. Intuitively, the reason is that the protocols are asynchronous and communicate in the presence of an adversary that can intercept, analyse, and modify the messages exchanged. Thus the execution of a protocol is highly non-deterministic and even for simple, finite protocols it requires the analysis of a potentially infinite number of behaviours of the adversary.

Following a tradition that originates with the work of Dolev and Yao [DY83], we take a *formal approach* that is we consider idealised/perfect versions of the cryptographic operations (see [AR00] for an interesting comparison with the *complexity-based* approaches) and we focus in particular on *fully automatic* decision procedures which return *exact* answers with respect to the abstract model. This excludes methods such as theorem proving [Bol96, Pau97], model-checking based on finite models [Low96, CJM98, MMS97], and abstract interpretation [Mon99, Gou00]. Of course, there are good reasons to study these methods but we will not comment on them here.

A variety of formalisms for the specification of cryptographic protocols have been proposed based on, *e.g.*, first-order logic [Wei99], linear logic [DLMS99], process calculi [Hui99], and equational logic [JRV00]. Here we follow [AP99, Hui99, AL00] and we model the protocol and the specification as a parallel *process*[1], decorated with logical assertions, and interacting with an adversary. The verification problem then amounts to checking that no invalid assertion is reachable. In previous work [AL00], we have proposed a symbolic reduction system that allows to decide the reachability problem for finite processes (*i.e.* processes without iteration or recursion) relying on *symmetric* keys. In this paper, we further develop our method in two main directions.

(1) We show that our symbolic reduction system can be generalised to handle *hashing* functions, *public* keys, and certain *logical assertions*. The logical assertions we consider are arbitrary boolean combinations of atomic predicates stating the equality of two messages or the secrecy of a message. We claim that our method provides a flexible decision procedure for *finite* processes and a good reference for optimised implementations that match the complexity lower bound of the problem. We note that this result does not follow trivially from those presented in [DLMS99] and [Wei99] where cryptographic protocols are described as 'Horn theories'. In particular:

(*i*) Not all processes considered in this paper can be readily compiled into Horn theories. As pointed out in [DLMS99], at least the conditional presents some difficulty.

(*ii*) The translation to Horn theories does not seem to offer an interesting upper bound to the reachability problem for finite processes.

[1]Henceforth, in the context of our model, we will speak of *processes* rather than of *protocols*.

(*iii*) The DEXPTIME decision procedure described in [DLMS99] (based on complexity results for DATALOG) concerns protocols which run an arbitrary number of sessions, without nonce creation, and with a bound on the size of the messages (this implies that only a finite number of distinct messages can be generated). Our decision result is orthogonal in that we consider a finite number of sessions but we put no bound on the size of the messages. The best lower bound we can offer for this problem is NPTIME which is actually optimal for a non-trivial class of processes (cf. section 8).

(2) In the case of *iterated or finite control* processes, we show that the problem is undecidable in general and PTIME decidable for a subclass of iterated processes that do not rely on the *pairing* constructor. This result is based on the classical theory of rational transduction of regular languages. We will show that it applies to a class of processes strictly containing the 'ping-pong' protocols studied in [DEK82].

The paper is organised as follows. In section 2 we introduce our model and state the reachability problem. In section 3 we show that the problem is NP-hard for finite processes and undecidable for iterated or finite control processes. In section 4, we prove some basic properties on the symbolic representation of *configurations*. In section 5, we apply these properties to build a *basic* symbolic reduction system. This system is further extended in section 6 to fully cover our process model, and in section 7 to handle a language of assertions with which we annotate the processes. In section 7, we also present the specification of a cryptographic protocol and comment on a prototype implementation of the symbolic reduction system. In section 8, we show that (part of) our symbolic reduction system can be implemented to run in NPTIME. Finally, in section 9 we exhibit a class of iterated processes without pairing for which the reachability problem is decidable in PTIME.

Given the considerable length of this work, we will often hint to the main ideas and shift the technical proofs to the appendix.

2 Model

We consider terms over an infinite signature: $\Sigma = \{C^0_n\}_{n\in\omega} \cup \{E^2, \langle\_\,,\_\rangle^2\}$. Thus we have an infinite set of constants and two binary constructors E for encoding and $\langle\_\,,\_\rangle$ for pairing. We use the following standard notation: $x, y, \ldots$ for (term) variables; V for the set of variables; $T_\Sigma(V)$ for the collection of finite terms over $\Sigma \cup V$; $t, t', \ldots$ for terms in $T_\Sigma(V)$; $\vec{t}$ for vectors of terms; $[t/x]$ for the substitution of t for x. We denote with $Var(t)$ the variables occurring in the term t, and by extension, if T is a set of terms then $Var(T) = \bigcup_{t\in T} Var(t)$.

Names We distinguish between basic names (agent's names, nonces, keys,...) and composed messages. The set of names $\mathcal{N}$ is defined as $\{C^0_n\}_{n\in\omega}$. We assume a (computable) relation $\mathcal{D} \subseteq \mathcal{N} \times \mathcal{N}$ with the interpretation:

$$(C, C') \in \mathcal{D} \text{ iff } \text{ messages encrypted with } C \text{ can be decrypted with } C'.$$

We define $Inv(C) = \{C' \mid (C, C') \in \mathcal{D}\}$. Further hypotheses, on the properties of $\mathcal{D}$ allow to model hashing, symmetric, and public keys. In particular: (i) for a *hashing* key C, $Inv(C) = \emptyset$, (ii) for a *symmetric* key C, $Inv(C) = \{C\}$, and (iii) for a *public* key C there is another key C' such that $Inv(C) = \{C'\}$ and $Inv(C') = \{C\}$.

Messages The set of closed messages $\mathcal{M}$ is defined as the least set that contains $\mathcal{N}$ and such that:

$$\begin{array}{lll} t \in \mathcal{M} \text{ and } t' \in \mathcal{N} & \Rightarrow & E(t,t') \in \mathcal{M} \\ t, t' \in \mathcal{M} & \Rightarrow & \langle t, t' \rangle \in \mathcal{M} \ . \end{array}$$

We also define the set of *open* messages $\mathcal{M}_V$ as the least set that contains $\mathcal{N} \cup V$ and such that:

$$\begin{array}{lll} t \in \mathcal{M}_V \text{ and } t' \in \mathcal{N} & \Rightarrow & E(t,t') \in \mathcal{M}_V \\ t, t' \in \mathcal{M}_V & \Rightarrow & \langle t, t' \rangle \in \mathcal{M}_V \ . \end{array}$$

We note that: (i) if $E(t, t')$ is a subterm of a term in $\mathcal{M}_V$ then $t' \in \mathcal{N}$, and that (ii) if $t \in \mathcal{M}_V$ and σ is a substitution mapping variables to elements of $\mathcal{M}_V$ then $\sigma(t) \in \mathcal{M}_V$.

The *formal* model of messages we study in this paper is fairly standard, cf., *e.g.*, [AR00, BDNP99, Low99]. We note that specific crypto-systems such as *RSA* may satisfy additional algebraic properties. The axioms above are not meant to describe a specific crypto-system instead they should be read as a specification to be *implemented* on top of some standard crypto-system. We also remark that in our model keys can only be simple names and not 'complex' keys such as pairs or encrypted messages. Extending the symbolic reduction to a model with complex keys appears to be a non-trivial task.

Synthesis and Analysis The functions S (synthesis) and A (analysis) are closure operators over the power set of terms $T_\Sigma(V)$ defined as follows:

- $S(T)$ is the least set that contains T and such that:

$$\begin{array}{lll} t_1, t_2 \in S(T) & \Rightarrow & \langle t_1, t_2 \rangle \in S(T) \\ t_1 \in S(T), t_2 \in T \cap \mathcal{N} & \Rightarrow & E(t_1, t_2) \in S(T) \ . \end{array}$$

- $A(T)$ is the least set that contains T and such that:

$$\begin{array}{lll} \langle t_1, t_2 \rangle \in A(T) & \Rightarrow & t_i \in A(T), \ i = 1, 2 \\ E(t_1, t_2) \in A(T), A(T) \cap Inv(t_2) \neq \emptyset & \Rightarrow & t_1 \in A(T) \ . \end{array}$$

We note the following properties.

Proposition 2.1 *Suppose $T \subseteq \mathcal{M}_V$. Then:*

(1) *$S(T), A(T) \subseteq \mathcal{M}_V$. Moreover, if $T \subseteq \mathcal{M}$ then $S(T), A(T) \subseteq \mathcal{M}$.*

(2) *S preserves set-theoretic containment and $S(S(T)) = S(T) \supseteq T$.*

(3) *A preserves set-theoretic containment and $A(A(T)) = A(T) \supseteq T$.*

(4) *$A(S(T)) = S(T) \cup A(T)$.*

(5) *$A(S(A(T))) = S(A(T)) = S(A(S(T)))$.*

PROOF. We observe that the operators A and S admit the following iterative definitions:

$$\begin{array}{l} A(T) = \bigcup_{n\in\omega} A(T)_n \quad A(T)_0 = T \\ A(T)_{n+1} = A(T)_n \cup \{t_1, t_2 \mid \langle t_1, t_2\rangle \in A(T)_n\} \\ \qquad\qquad\qquad \cup \{t \mid E(t,C) \in A(T)_n, \mathit{Inv}(C) \cap A(T)_n \neq \emptyset\} \end{array}$$

$$\begin{array}{l} S(T) = \bigcup_{n\in\omega} S(T)_n \quad S(T)_0 = T \\ S(T)_{n+1} = S(T)_n \cup \{\langle t_1, t_2\rangle \mid t_i \in S(T)_n, i = 1,2\} \cup \{E(t,C) \mid t \in S(T)_n, C \in T\} \end{array}$$

From this remark the proof of (1-6) is rather direct and delayed to appendix A.1. $\diamond$

Processes Processes are defined as follows:

$$\begin{array}{rl} P ::= & 0 \mid !t.P \mid ?x.P \mid x \leftarrow \mathsf{dec}(t,t').P \mid x \leftarrow \mathsf{prjl}(t).P \mid x \leftarrow \mathsf{prjr}(t).P \mid \\ & \mathsf{asrt}(\varphi).P \mid P \mid P' \mid [t = t']P, P' \mid \mathsf{it}\, P \end{array}$$

A process performs internal computation, interacts with the adversary via input and output, and, from time to time, checks certain logical assertions φ. The informal description of the execution of a process is as follows: 0 is the process which is terminated; $!t.P$ evaluates t and if t is a message, sends it to the adversary and becomes P (otherwise it terminates); $?x.P$ receives a message t from the adversary and becomes $[t/x]P$; $x \leftarrow \mathsf{dec}(t,t').P$ (dec for decryption) evaluates t, t' and if t is a message $E(t'',C)$ and $t' \in \mathit{Inv}(C)$ then it becomes $[t''/x]P$ (otherwise it terminates); similarly, $x \leftarrow \mathsf{prjl}(t).P$ (prjl for left projection) evaluates t and if t is a message $\langle t', t''\rangle$ then it becomes $[t'/x]P$ ($[t''/x]P$) (otherwise it terminates); a symmetric interpretation applies to the right projection prjr; $\mathsf{asrt}(\varphi).P$ evaluates the boolean predicate φ with respect to T and becomes P if φ holds (otherwise we terminate in the erroneous configuration); $P \mid P'$ is the asynchronous parallel composition of P and P'; $[t = t']P, P'$ evaluates t and t' and if both are messages then it becomes P if $t \equiv t'$ and P' if $t \not\equiv t'$ (otherwise it terminates); $\mathsf{it}\, P$ is an unbounded iteration of P so that $\mathsf{it}\, P$ behaves as $P \mid \mathsf{it}\, P$. We denote with $FV(P)$ the set of variables occurring free in P.

Definition 2.2 *A* well-formed configuration k *is either a special* erroneous *configuration* err *or a pair* (P,T) *where (i)* P *is a closed process and (ii)* T *is a non-empty finite subset of* $\mathcal{M}$ *which represents the current knowledge of the adversary.*

In figure 1, we define a reduction relation on well-formed configurations. In these rules, we always reduce the leftmost process with the proviso that parallel composition is associative and commutative. Moreover, we take the freedom of writing P as $P \mid 0$ whenever needed to apply a rewriting rule and we omit the symmetric rule for the right projection.

We note that a thread is stuck whenever it tries (i) to evaluate a term which is not a message, or (ii) to decrypt with a wrong key, or (iii) to project a message which is not a pair. Concerning the language of assertions φ, for the time being, we just assume that the condition $\models_T \varphi$ is decidable and that the formula *false* is an (invalid) assertion. We turn next to the definition of the reachability problem.

$$
\begin{array}{lll}
(!) & (!t.P \mid P', T) & \rightarrow (P \mid P', T \cup \{t\}) \text{ if } t \in \mathcal{M} \\
(?) & (?x.P \mid P', T) & \rightarrow ([t/x]P \mid P', T) \;\text{ if } t \in S(A(T)) \\
(\mathsf{d}) & (x \leftarrow \mathsf{dec}(E(t, C), C').P \mid P', T) & \rightarrow ([t/x]P \mid P', T) \text{ if } C' \in \mathit{Inv}(C), t \in \mathcal{M} \\
(\mathsf{pl}) & (x \leftarrow \mathsf{prjl}(\langle t, t' \rangle).P \mid P', T) & \rightarrow ([t/x]P \mid P', T) \;\text{ if } t, t' \in \mathcal{M} \\
(\mathsf{a}) & (\mathsf{asrt}(\varphi).P \mid P', T) & \rightarrow \begin{cases} (P \mid P', T) & \text{if } \models_T \varphi \\ \mathsf{err} & \text{if } \not\models_T \varphi \end{cases} \\
(\mathsf{m}_1) & ([t = t]P_1, P_2 \mid P', T) & \rightarrow (P_1 \mid P', T) \text{ if } t \in \mathcal{M} \\
(\mathsf{m}_2) & ([t = t']P_1, P_2 \mid P', T) & \rightarrow (P_2 \mid P', T) \text{ if } t \neq t', t, t' \in \mathcal{M} \\
(\mathsf{it}) & (\mathsf{it}\, P \mid P', T) & \rightarrow (P \mid \mathsf{it}\, P \mid P', T)
\end{array}
$$

Figure 1: Reduction on configurations

Definition 2.3 *Let k be a configuration. We say that k can reach error if $k \xrightarrow{*} \mathsf{err}$.*

Thus, the *reachability problem* is the problem of determining whether a configuration can reach the erroneous one.

Remark 2.4 *In [AL00], we also consider a* name generator *operator $\nu x\, P$ replacing x with a* fresh *name. This operator can be specified by adding a* counter *to a configuration. We note that for finite processes the name generator can be statically eliminated, and that for iterated/finite state processes the name generator adds another source of undecidability as shown in [DLMS99].*

3 Lower bounds

We show that reachability is NP-hard for finite processes (without iteration) and undecidable for processes allowing iteration. While these results appear to be folklore, it is worth to spell them out trying to rely on as little machinery as possible. In the following, we will just rely on symmetric keys, input, output, decryption, projection, parallel composition, and the assertion *false*.

First, we introduce a notation to *filter* inputs. Let $t \in \mathcal{M}_V$ be a linear term (all variables are distinct) generated by the grammar: $t ::= y \mid E(t, C) \mid \langle t, t \rangle$. We define $\mathsf{case}\ t = x\ \mathsf{in}\ P$ by induction on t (x', x'' are fresh variables):

$$
\begin{array}{ll}
\mathsf{case}\ y = x\ \mathsf{in}\ P & = [x/y]P \\
\mathsf{case}\ E(t, C) = x\ \mathsf{in}\ P & = x' \leftarrow \mathsf{dec}(x, C).\mathsf{case}\ t = x'\ \mathsf{in}\ P \\
\mathsf{case}\ \langle t', t'' \rangle = x\ \mathsf{in}\ P & = x' \leftarrow \mathsf{prjl}(x).x'' \leftarrow \mathsf{prjr}(x). \\
 & \quad \mathsf{case}\ t' = x'\ \mathsf{in}\ \mathsf{case}\ t'' = x''\ \mathsf{in}\ P\ .
\end{array}
$$

Finally, we abbreviate $?x = t.P \equiv ?x.\mathsf{case}\ t = x\ \mathsf{in}\ P$.

3.1 NP-hardness for finite processes

We code satisfaction of a boolean formula in conjunctive normal form (CNF). Let ϕ be a formula in CNF depending on the boolean variables $x_1, \ldots, x_n$ and composed of the disjunctions $\psi_1, \ldots, \psi_m$. We assume the following distinct constants: $C, C_0, C_1, V_1, \ldots, V_n, D_1, \ldots, D_m$. Initially, we set the knowledge of the adversary to:

$$T_0 = \{E(E(C, C_0), C), E(E(C, C_1), C)\}$$

where we interpret $E(C, C_0)$ as the boolean value 0 and $E(C, C_1)$ as the boolean value 1. We define a process P_ϕ which is the parallel composition of the following processes:

(1)	$?x = E(y, C).!E(y, V_i).0, \; i = 1, \ldots, n$	('write once' boolean value in V_i)
(2)	$?x = E(E(y, C_1), V_i).!D_k.0, \; x_i$ literal in ψ_k	(disjunction ψ_k evaluates to 1)
(2′)	$?x = E(E(y, C_0), V_i).!D_k.0, \; \overline{x_i}$ literal in ψ_k	(disjunction ψ_k evaluates to 1)
(3)	$?x = E(\cdots E(y, D_m), \ldots, D_1).\mathsf{asrt}(\mathit{false}).0$	(all disjunctions evaluate to 1)

We use a double level of encryption to be able to test in (2) and (2′) the contents of a variable without relying on the conditional. The proof of the following proposition is rather direct and delayed to appendix A.2.

Proposition 3.1 *Let ϕ be a boolean formula in CNF. Then ϕ is satisfiable iff $(P_\phi, T_0) \xrightarrow{*} \mathsf{err}$.*

We point out that processes with several alternations of filtered inputs and outputs can be compiled into the parallel composition of processes of the simpler shape above, *i.e.*, a filtered input followed by an output. For instance,

$$?x_1 = t_1.!s_1.?x_2 = t_2.!s_2.?x_3 = t_3.!s_3.0, \;\; \mathit{Var}(s_i) \subseteq \mathit{Var}(t_i), i = 1,2,3 \;\; \text{becomes}$$
$$?x_1 = t_1.!s_1.0 \mid ?x' = \langle t_1, t_2 \rangle.!s_2.0 \mid ?x'' = \langle \langle t_1, t_2 \rangle, t_3 \rangle.!s_3.0$$

which is a transformation that can *square* the size of the program. It is interesting to formulate the reachability problem for these simple processes as the refutation of a 'logic program' (this idea is already found in [DLMS99, Wei99]). We introduce a monadic predicate T with the interpretation:

$$T(t) \;\; \text{iff} \;\; t \text{ is known by the adversary.}$$

Given a configuration $k \equiv (P, T_0)$ we define the 'logic' program L_k as follows:

	$\supset T(t)$	if $t \in T_0$
$T(t), T(t')$	$\supset T(\langle t, t' \rangle)$	(synthesis pair)
$T(t), T(t')$	$\supset T(E(t, t'))$	(synthesis encryption)
$T(\langle t, t' \rangle)$	$\supset T(t)$	(analysis pair left)
$T(\langle t, t' \rangle)$	$\supset T(t')$	(analysis pair right)
$T(E(t, t')), T(t')$	$\supset T(t)$	(analysis encryption)
$T(t)$	$\supset T(t')$	if $?x = t.!t'.0$ in P
		(clauses executed at most once)
$T(t)$	$\supset \bot$	if $?x = t.\mathsf{asrt}(\mathit{false}).0$ in P

The logic program L_k is actually a bit more liberal than our model as it allows to conclude $T(t)$ even when $t \notin \mathcal{M}$. However, it is easy to get an exact correspondence by introducing a second predicate M such that $M(t)$ iff $t \in \mathcal{M}$. Then we refine, *e.g.*, the clause for synthesis of encryption as follows:

$$T(t), T(t'), M(E(t,t')) \supset T(E(t,t')) \ .$$

We also note that the logic program obtained is not quite standard as the clauses corresponding to the program can be executed at most once while the clauses corresponding to the adversary (initial knowledge, synthesis, and analysis) can be executed arbitrary many times. For this reason, the presentation as a logic program does not seem to offer an immediate upper bound on the complexity of the reachability problem and this (suggestive) point of view will not be further pursued in this paper.

3.2 Undecidability for iterated processes

We encode the halting problem for 2-counter machines into a reachability problem for iterated processes (this has been suggested to us by H. Comon). We recall that a two-counter non-deterministic machine is a tuple (Q, Q_F, q_0, Δ) with Q a set of *states*, $Q_F \subseteq Q$ a set of *final* states $q_0 \in Q$ the *initial* state and $\Delta \subseteq Q \times \{0,1\}^2 \times Q \times \{-1,0,1\}^2$ a *transition relation*. We assume that if $(q, x_1, x_2, q', y_1, y_2) \in \Delta$ then $y_i = -1$ implies $x_i = 1$, for $i = 1, 2$ (we can decrement a positive counter only). A *configuration* of the machine is a triple (q, n_1, n_2) with $q \in Q, n_i \in \omega$ (the n_i's are the counters). We have a *reduction* $(q, n_1, n_2) \to (q', n_1 + y_1, n_2 + y_2)$ if $(q, x_1, x_2, q', y_1, y_2) \in \Delta$ and $n_i = 0$ iff $x_i = 0$ $(i = 1, 2)$. A configuration (q, n_1, n_2) is *reachable* if $(q_0, 0, 0) \xrightarrow{*} (q, n_1, n_2)$. A configuration (q, n_1, n_2) is *final* if $q \in Q_F$. For two-counter machines it is undecidable whether a final configuration is reachable (see, *e.g.*, [HU79], page 172).

Next we describe an encoding of 2-counter machines into configurations. We assume distinct constants C, C_0, C_1, D and a distinct constant $\underline{q} \in \mathcal{N}$ for every state $q \in Q$. We associate a message to every natural number as follows:

$$\underline{0} = E(C, C_0) \quad \underline{n+1} = E(\underline{n}, C_1) \ .$$

We associate a message to every 2-counter machine's configuration as follows:

$$\underline{(q, n, n')} = E(\langle \underline{n}, \underline{n'} \rangle, \underline{q}) \ .$$

We associate to every rule of the 2-counter machine an iterated process. For instance, consider the rule $r \equiv (0, 1, q, q', 1, -1)$ which means that if the first counter is null, the second one is not null, and the machine is in state q then the machine can go to state q', add 1 to the first counter, and subtract 1 to the second. This is encoded by the process (by the way, here we could use tail recursive/finite control processes rather than iterated processes):

$$P_r \equiv \mathsf{it}\, ?x = E(\langle E(C, C_0), E(y', C_1) \rangle, \underline{q}).!E(\langle E(E(C, C_0), C_1), y' \rangle, \underline{q'}).0 \ .$$

For each final state $q \in Q_F$, we introduce the process:

$$P_q \equiv ?x = E(\langle x, y\rangle, \underline{q}).\mathsf{asrt}(\mathit{false}).0 \ .$$

The initial knowledge T_0 of the adversary is the encoding of the initial configuration: $T_0 = \{\underline{(q_0, 0, 0)}\}$. Finally, we associate to a 2-counter machine M a process P_M

$$P_M \equiv \Pi_{q \in Q_F} P_q \ \mid \ \Pi_{r \in \Delta} P_r$$

which eventually informs the adversary of (the translation of) every reachable configuration. Given this encoding, we note the following proposition whose proof is given in appendix A.3.

Proposition 3.2 (1) $(q_0, 0, 0) \xrightarrow{*} (q, n, n')$ *iff for some* P, T, $(P_M, T_0) \xrightarrow{*} (P, T)$ *and* $\underline{(q, n, n')} \in T$.

(2) *A final configuration is reachable iff* $(P_M, T_0) \xrightarrow{*} \mathsf{err}$.

The encoding of 2-counter machines makes an essential use of pairing. Indeed we will show in section 9 that without pairing the reachability problem for the simple iterated processes considered above is decidable in polynomial time. Since the encoding of satisfaction of CNF in section 3.1 does not rely on pairing, it turns out that in this particular case reachability is easier to check for iterated than for finite processes.

4 Symbolic analysis

In this section, we study the effect of certain *admissible* substitutions on the synthesis and analysis operators. Our analysis culminates in the *decomposition* theorem 4.10 that makes it possible to compute on *symbolic* configurations, *i.e.*, on configurations containing free variables ranging over certain infinite sets of messages. Let us point out that the generalisation from 'symmetric keys' to 'public keys' is not straightforward. In particular, the notion of *irreducible* set presented in [AL00] needs to be generalised to the notion of *(minimum) generator* and the fact that we can actually compute this set and with this set is not quite obvious as witnessed by proposition 4.6 and lemma 4.9.

Definition 4.1 *Let* $T \subseteq \mathcal{M}_V$ *and* $K \subseteq_{fin} \mathcal{N}$.

(1) *Suppose* $t \in \mathcal{M}_V$. *We say that* t' *is* K-accessible *in* t *iff either* $t \equiv t'$ *or* $t \equiv \langle t_1, t_2\rangle$ *and for some* $i \in \{1, 2\}$, t' *is* K*-accessible in* t_i *or* $t \equiv E(t_1, C)$, $\mathit{Inv}(C) \cap K \neq \emptyset$, *and* t' *is* K*-accessible in* t_1.

(2) *We define* $P_K(T)$, *the* K-accessible parts *of* T, *as the set of terms that are* K*-accessible in a term* $t \in T$.

(3) *We define* $S_K(T)$, *the* K-synthesis *of* T, *as the least set of terms that contains* $T \cup K$ *and is closed under pairing and encryption by a name in* K.

(4) *Finally, we define* $K(T)$ *as the least set of names* K *such that* $C \in P_K(T)$ *implies* $C \in K$.

Example 4.2 *Suppose* $T = \{C_1, \langle E(C_2, C_1), E(E(C_3, C_3), C_2)\rangle\}$ *where all keys are symmetric. Then* $P_\emptyset(T) \cap \mathcal{N} = \{C_1\}$ *and* $P_{\{C_1\}}(T) \cap \mathcal{N} = \{C_1, C_2\} = P_{\{C_1,C_2\}}(T) \cap \mathcal{N}$. *Hence,* $K(T) = \{C_1, C_2\}$.

Proposition 4.3 *Suppose* $T \subseteq \mathcal{M}_V$. *Then:*

(1) $K(T) = A(T) \cap \mathcal{N}$.

(2) $A(T) = P_{K(T)}(T)$.

(3) $S(A(T)) = S_{K(T)}(P_{K(T)}(T))$.

PROOF. We observe that the set $K(T)$ admits the following iterative definition:

$$\begin{array}{l} K(T) = \bigcup_{n\in\omega} K(T)_n \quad K(T)_0 = P_\emptyset(T) \cap \mathcal{N} \\ K(T)_{n+1} = K(T)_n \cup (P_{K(T)_n}(T) \cap \mathcal{N}) \ . \end{array}$$

We also note that $P_K(T) = \bigcup_{t\in T} acc_K(t)$ where $acc_K(t)$ is a set of terms defined inductively on the structure of t:

$$\begin{array}{lll} acc_K(C) & = C & \\ acc_K(x) & = x & \\ acc_K(\langle t_1, t_2\rangle) & = \{\langle t_1, t_2\rangle\} \cup acc_K(t_1) \cup acc_K(t_2) & \\ acc_K(E(t, C)) & = \{E(t, C)\} \cup acc_K(t) & \text{if } Inv(C) \cap K \neq \emptyset \\ acc_K(E(t, C)) & = \{E(t, C)\} & \text{if } Inv(C) \cap K = \emptyset \ . \end{array}$$

Given this inductive characterisation the proof of (1-3) is rather direct and delayed to appendix A.4. ◇

Definition 4.4 *Let* $T \subseteq \mathcal{M}_V$. *We say that the set* G *is a generator for* T *if* $S(A(T)) = S(G)$.

The proof of the following proposition relies on propositions 2.1 and 4.3. In particular, for (4) we prove that $t \in S(A(T))$ implies $t \in S(G \cap G')$ by induction on the structure of t (see appendix A.5).

Proposition 4.5 *Suppose* $T \subseteq \mathcal{M}_V$ *and* G, G' *are generators for* T. *Then:*

(1) $P_{K(T)}(T)$ *is a generator for* T.

(2) $K(T) = G \cap \mathcal{N}$.

(3) $Var(T) = Var(S(A(T)) = Var(S(G)) = Var(G)$.

(4) $G \cap G'$ *is a generator for* T.

From proposition 4.5(4), we know that there exists a minimum generator. The following proposition gives a method to compute it.

Proposition 4.6 (minimum generator) *Suppose $T \subseteq_{fin} \mathcal{M}_V$. Then the application of the following rules always terminates and computes the smallest generator for T that we denote with $G_\mu(T)$.*

$$\begin{array}{llll} (a) & \{t\} \cup T & \to T & \text{if } t \in S(T) \\ (b) & \{\langle t, t' \rangle\} \cup T & \to \{t, t'\} \cup T & \\ (c) & \{E(t, C)\} \cup T & \to \{t, E(t, C)\} \cup T & \text{if } T \cap Inv(C) \neq \emptyset \text{ and } t \notin S(T). \end{array}$$

PROOF HINT. Note that when we write $\{t\} \cup T$ on the left hand side of a rule it is intended that $t \notin T$. First we show by case analysis that if $T_1 \to T_2$ then (i) $S(A(T_1)) = S(A(T_2))$ and (ii) $T_1 \subseteq S(T_2)$. The properties presented in proposition 2.1 are useful here. Termination is easily established.

Suppose $T \xrightarrow{*} T'$ and T' is in normal form. To show that T' is the smallest generator we prove first by induction on n that $A(T')_n \subseteq S(T')$. Then we observe:

$$\begin{array}{lll} S(A(T)) & = S(A(T')) & \text{by (i)} \\ & = S(A(S(T'))) & \text{by proposition 2.1(5)} \\ & = S(S(T') \cup A(T')) & \text{by proposition 2.1(4)} \\ & = S(S(T')) & \text{by } A(T') \subseteq S(T') \\ & = S(T') & \text{by proposition 2.1(2)} . \end{array}$$

Suppose exists $G \subset T'$ such that $S(G) = S(T') = S(A(T))$. We choose $t \in T' \backslash G$. Then $T' = T'' \cup \{t\}$, $t \in S(T'') = S(G)$ and $T' \to T''$ by (a) which is a contradiction. Thus T' is a minimal generator and from proposition 4.5(4) it follows that it is minimum. $\diamond$

Example 4.7 *If we take $T = \{E(\langle C_1, x \rangle, C_2), E(C_1, C_3), C_3\}$ with $Inv(C_2) = \{C_3\}$ and $Inv(C_3) = \{C_2\}$ then $G_\mu(T) = \{E(\langle C_1, x \rangle, C_2), E(C_1, C_3), C_1, C_3, x\}$ which is strictly contained in $A(T)$.*

Definition 4.8 (1) *An* environment *E is a possibly empty list of the shape $x_1 : T_1; \ldots ; x_n : T_n$ where $T_1 \subseteq \cdots \subseteq T_n \subseteq_{fin} \mathcal{M}_V$ is a non decreasing sequence of finite sets of open messages and $Var(T_i) \subseteq \{x_1, \ldots, x_{i-1}\}$.*

(2) *A substitution σ is* admissible *for the environment $x_1 : T_1; \ldots ; x_n : T_n$ if*

$$\begin{array}{ll} \sigma(x_i) \in S(A(\sigma(T_i))) & \text{for } i = 1, \ldots, n, \\ \sigma(y) = y & \text{if } y \notin \{x_1, \ldots, x_n\}. \end{array}$$

Lemma 4.9 (symbolic representation) *Let $E \equiv x_1 : T_1; \ldots ; x_n : T_n$ be an environment, σ be an admissible substitution, and G_i be a generator for T_i. Then:*

(1) $K(\sigma T_i) = K(T_i)$.

(2) $S(A(\sigma T_i)) = \sigma S(A(T_i))$.

(3) $S(\sigma G_i) = \sigma S(G_i) = S(\sigma(G_i \backslash V))$.

PROOF HINT. The full proof is given in appendix A.7.

(1) It is easy to establish $K(T_i) \subseteq K(\sigma T_i)$. To show the other inclusion, we prove

$$(1') \quad C \in acc_{K(T_i)}(\sigma x_j) \Rightarrow C \in K(T_i) \quad \text{if } j < i \ .$$

by induction on the pair (i, j) lexicographically ordered.

(2) Relying on proposition 4.3(3) we prove $\sigma S(A(T_i)) \subseteq S(A(\sigma T_i))$. To show the other inclusion, we prove by induction on i that

$$P_{K(T_i)}(\sigma T_i) \subseteq S_{K(T_i)}(\sigma P_{K(T_i)}(T_i)) \ .$$

(3) To prove $S(\sigma G_i) = \sigma S(G_i)$ we recall proposition 4.5(2) and proceed by induction on the structure of the term. To establish the equation $S(\sigma G_i) = S(\sigma(G_i \backslash V))$, we prove by induction on j ($j \leq i$) that $\sigma V_j \subseteq S(\sigma(G_i \backslash V))$. From this we derive $\sigma(G_i \cap V) \subseteq S(\sigma(G_i \backslash V))$ that implies the desired property. $\diamond$

Theorem 4.10 (decomposition) *Let $E \equiv x_1 : T_1; \ldots; x_n : T_n$ be an environment, σ be an admissible substitution, and G_i be a generator for T_i. Then:*

(1) $C \in S(A(\sigma T_i))$ *iff* $C \in K(T_i)$.

(2) $\langle t, t' \rangle \in S(A(\sigma T_i))$ *iff* $t, t' \in S(A(\sigma T_i))$.

(3) $E(t, C) \in S(A(\sigma T_i))$ *iff* $\exists\, E(t', C) \in G_i \ E(t, C) = \sigma(E(t', C))$ *or* $(C \in K(T_i)$ *and* $t \in S(A(\sigma T_i)))$.

PROOF. (1) We observe:

$$\begin{array}{ll} C \in S(A(\sigma T_i)) & \text{iff } C \in A(\sigma T_i) \cap \mathcal{N} \\ \text{iff } C \in K(\sigma T_i) & \text{iff } C \in K(T_i) \ . \end{array}$$

(2) $\langle t, t' \rangle \in S(A(\sigma T_i)) = A(S(A(\sigma T_i)))$ implies $t, t' \in S(A(\sigma T_i))$ by definition of analysis. Vice versa, $t, t' \in S(A(\sigma T_i)) = S(S(A(\sigma T_i)))$ implies $\langle t, t' \rangle \in S(A(\sigma T_i))$ by definition of synthesis.

(3) ($\Leftarrow$) We consider the two cases.

- If $E(t', C) \in G_i$ then, by lemma 4.9(2-3):

$$\sigma E(t', C) \in \sigma G_i \subseteq S(\sigma G_i) = \sigma S(G_i) = \sigma S(A(T_i)) = S(A(\sigma T_i)) \ .$$

- If $C \in K(T_i)$ and $t \in S(A(\sigma T_i))$ then $E(t, C) \in S(S(A(\sigma T_i))) = S(A(\sigma T_i))$.

($\Rightarrow$) Suppose $E(t, C) \in S(A(\sigma T_i))$ and ($C \notin K(T_i)$ or $t \notin S(A(\sigma t_i))$). Then

$$\begin{array}{ll} S(A(\sigma T_i)) = \sigma S(A(T_i)) = \sigma S(G_i) & \text{by lemma 4.9(2)} \\ = S(\sigma G_i) = S(\sigma(G_i \backslash V)) & \text{by lemma 4.9(3)} \end{array}$$

We conclude that $E(t, C) \equiv \sigma E(t', C)$ for some $E(t', C) \in G_i$. $\diamond$

5 Basic symbolic reduction

In defining the symbolic reduction, our strategy is to maintain the constraints $x_1 : T_1; \ldots ; x_n : T_n$ in the form required to apply theorem 4.10. In this section, we will just consider simple processes P satisfying the following conditions:

(1) All terms occurring in P belong to $\mathcal{M}_V$ (cf. section 2).

(2) P does not contain the operators of parallel composition, conditional, and iteration.

(3) The only assertions allowed are of the form $\mathsf{asrt}(\mathit{false})$.

Note that these conditions are preserved by reduction. We will see in section 6 how to lift restrictions (1-2) and in section 7 how to handle more general assertions.

Definition 5.1 *A symbolic configuration is either* err *or a triple* (P, T, E) *where:*

(1) *P is a process and T is a non-empty finite set of terms in $\mathcal{M}_V$ such that for some set of variables $\{x_1, \ldots, x_n\}$, $FV(P) \cup Var(T) \subseteq \{x_1, \ldots, x_n\}$.*

(2) *$E \equiv x_1 : T_1; \ldots ; x_n : T_n$ is an environment (cf. definition 4.8) such that $T_1 \neq \emptyset$ if $E \neq \emptyset$.*

The basic rules for symbolic reduction are presented in figure 2 where the symmetric rules for the right projection are omitted. We note that the rules $(\mathsf{d}^s_{2,3})$ are not mutually exclusive in the case where $C \in K(T)$ and $Inv(C) \cap K(T) = \emptyset$. This case, which does not arise in the symmetric case, corresponds to a situation where the adversary knows C but not its inverses so that it can synthesise a message of the form $E(t, C)$ but it is unable to decrypt such a message.

Let us examine the soundness and completeness of the symbolic reduction system.

Definition 5.2 *Let $c \equiv (P, T, E)$ be a symbolic configuration and σ be a ground substitution. We write $\sigma \models E$ iff σ is compatible with the sequence $T_1 \subseteq \cdots \subseteq T_n$ in E.*

The following theorem 5.3 implies that a symbolic configuration $(P, T, \emptyset)$ reduces to err iff the configuration (P, T) does. Moreover, all symbolic reductions are terminating and finitely branching. This entails a simple decision procedure for the reachability problem: explore all symbolic reductions and check whether they lead to the configuration err.

Theorem 5.3 (1) *Symbolic reduction always terminates.*

(2) $(P, T, E) \xrightarrow{*} \mathsf{err}$ *iff* $\exists\, \sigma \models E \;\; \sigma(P, T) \xrightarrow{*} \mathsf{err}$.

PROOF HINT. This result is generalised in theorem 6.13, to which we refer for a complete proof.

(1) Symbolic reduction terminates because every rule decreases by one the number of prefixes in the process P.

$$
\begin{array}{lll}
(?^s) & (?x.P,T,E) \rightarrow (P,T,E,x:T) & \\
(!^s) & (!t.P,T,E) \rightarrow (P,T\cup\{t\},E) & \\
(\mathsf{pl}^s_1) & (x \leftarrow \mathsf{prjl}(\langle t,t'\rangle).P,T,E) \rightarrow ([t/x]P,T,E) & \\
(\mathsf{pl}^s_2) & (x \leftarrow \mathsf{prjl}(x_i).P,T,E) \rightarrow [\langle x,x'\rangle/x_i](P,T,E) & x' \text{ fresh} \\
(\mathsf{d}^s_1) & (x \leftarrow \mathsf{dec}(E(t,C),C').P,T,E) \rightarrow ([t/x]P,T,E) & \text{if } C' \in \mathit{Inv}(C) \\
(\mathsf{d}^s_2) & (x \leftarrow \mathsf{dec}(x_i,C').P,T,E) \rightarrow [E(x,C)/x_i](P,T,E) & \text{if } C \in K(T_i), \\
 & & C' \in \mathit{Inv}(C) \\
(\mathsf{d}^s_3) & (x \leftarrow \mathsf{dec}(x_i,C').P,T,E) \rightarrow [E(t,C)/x_i]([t/x]P,T,E) & \text{if } E(t,C) \in G_\mu(T_i), \\
 & & C' \in \mathit{Inv}(C) \\
(\mathsf{a}^s) & (\mathsf{asrt}(\mathit{false}).P,T,E) \rightarrow \mathsf{err} &
\end{array}
$$

$$
\begin{array}{ll}
[\langle x,x'\rangle/x_i]E \equiv & x_1:T_1;\ldots;x_{i-1}:T_{i-1};x:T_i;x':T_i; \\
 & x_{i+1}:[\langle x,x'\rangle/x_i]T_{i+1};\ldots;x_n:[\langle x,x'\rangle/x_i]T_n \\
 & \\
[E(x,C)/x_i]E \equiv & x_1:T_1;\ldots;x_{i-1}:T_{i-1};x:T_i; \\
 & x_{i+1}:[E(x,C)/x_i]T_{i+1};\ldots;x_n:[E(x,C)/x_i]T_n \\
 & \\
[E(t,C)/x_i]E \equiv & x_1:T_1;\ldots;x_{i-1}:T_{i-1}; \\
 & x_{i+1}:[E(t,C)/x_i]T_{i+1};\ldots;x_n:[E(t,C)/x_i]T_n
\end{array}
$$

Figure 2: Basic symbolic reduction

(2) In both directions we proceed by induction on the length of the reduction to err. To show *completeness* (implication $\Leftarrow$) we rely on the decomposition theorem 4.10. $\diamond$

We consider an example of application of the basic symbolic reduction procedure.

Example 5.4 *We assume symmetric keys. Consider the process:*

$$P_1 \equiv ?x_1.!E(x_1,C_1).?x_2.x_3 \leftarrow \mathsf{dec}(x_2,C_1).x_4 \leftarrow \mathsf{dec}(x_3,C_0).\mathsf{asrt}(\mathit{false}).0$$

where initially $T_1 = \{C_0\}$. *We show* $(P_1,T_1,\emptyset) \xrightarrow{*} \mathsf{err}$. *We have:*

$$
\begin{array}{l}
(P_1,T_1,\emptyset) \\
\rightarrow (!E(x_1,C_1)\ldots,T_1,x_1:T_1) \ \textit{by}\ (?^s) \\
\rightarrow (?x_2.\ldots,T_2,x_1:T_1) \ \textit{where}\ T_2 = T_1\cup\{E(x_1,C_1)\}, \ \textit{by}\ (!^s) \\
\rightarrow (x_3 \leftarrow \mathsf{dec}(x_2,C_1)\ldots,T_2,E_2) \ \textit{where}\ E_2 = x_1:T_1;x_2:T_2, \ \textit{by}\ (?^s) \\
\rightarrow [E(x_1,C_1)/x_2]([x_1/x_3]x_4 \leftarrow \mathsf{dec}(x_3,C_0).\mathsf{asrt}(\mathit{false}).0,T_2,E_2), \ \textit{by}\ (\mathsf{d}^s_3) \\
\equiv (x_4 \leftarrow \mathsf{dec}(x_1,C_0).\mathsf{asrt}(\mathit{false}).0,T_2,x_1:T_1), \ \textit{by substitution} \\
\rightarrow (\mathsf{asrt}(\mathit{false}).0,T_2,x_4:T_1), \ \textit{by}\ (\mathsf{d}^s_2) \\
\rightarrow \mathsf{err}\ \textit{by}\ (\mathsf{a}^s).
\end{array}
$$

Following the substitutions backwards, we can express the set of successful 'attacks' of the adversary as $x_1 = E(x_4,C_0), x_2 = E(E(x_4,C_0),C_1)$ *where* $x_4 \in S(\{C_0\})$.

6 Extensions of basic symbolic reduction

In this section, we show how to lift the restrictions imposed in section 5 thus defining a symbolic reduction for the full process model introduced in section 2.

6.1 Variables in key position

Let us denote with $Var_{key}(t)$ the collection of variables x such that $E(t', x)$ is a subterm of t and with $\mathcal{M}_V^o$ the collection of terms in $T_\Sigma(V)$ such that in every subterm $E(t', t'')$, t'' is either a name or a variable.

Then consider a term $t \in T_\Sigma(V)$, an environment E constraining the variables in $Var(t)$, and an admissible substitution $\sigma \models E$. We observe that $\sigma t \in \mathcal{M}$ iff $t \in \mathcal{M}_V^o$ and $\sigma(x_i) \in K(T_i)$ whenever $x_i \in Var_{key}(t)$. We write $\tau \downarrow (t, E)$ if:

(1) $\tau = id$ and $t \in \mathcal{M}_V$ or

(2) $t \in \mathcal{M}_V^o$, $Var_{key}(t) = \{x_1, \ldots, x_m\}$, $m \geq 1$, and $\tau(x_i) \in K(T_i)$, if $x_i \in Var_{key}(t)$ and $\tau(x_i) = x_i$, otherwise.

We note that for all t, E there are only a *finite* number of substitutions τ such that $\tau \downarrow (t, E)$.

With this notation, we can write the rules for output, decryption, projection,... by combining reduction and instantiation of variables in key position. For instance, the rule $(!^s)$ becomes:

$$(!^s) \quad (!t.P, T, E) \;\rightarrow\; \tau(P, T \cup \{t\}, E) \;\text{ if } \tau \downarrow (t, E)$$

where, as expected, $\tau(P, T, E) = (\tau P, \tau T, \tau E)$ and the definition of $[C/x_i]E$ amounts to remove the constraint $x_i : T_i$ and replace x_i with C in E (cf. bottom of figure 4).

6.2 Parallel composition and iteration

We can handle parallel composition and iteration along the lines of the non-symbolic reduction system in figure 1. Namely, we assume that parallel composition is associative and commutative and that $P \equiv P \mid 0$. Then, without loss of generality, we always reduce the leftmost thread of a process and we suppose that there is at least another thread running in parallel. Then, *e.g.*, the rule for output is rewritten, once more, as follows:

$$(!^s) \quad (!t.P \mid P', T, E) \;\rightarrow\; \tau(P \mid P', T \cup \{t\}, E) \;\text{ if } \tau \downarrow (t, E)\ .$$

The rule for iteration can be written along the pattern of rule (it) in figure 1. However, in view of the undecidability result in section 3.2 this is not very interesting. Summarising our discussion, we present in figure 3 the symbolic reduction system not including the rules for conditional and assertions. The system operates on quadruples (P, T, E, I) where I is a finite set of inequalities whose introduction will be motivated in the following section 6.3. We denote with u either a name or a variable occurring in the environment E.

$(?^s)$ $(?x.P \mid P', T, E, I) \rightarrow (P \mid P', T, E; x : T, I)$

$(!^s)$ $(!t.P \mid P', T, E, I) \rightarrow \tau(P \mid P', T \cup \{t\}, E, I)$
if $\tau \downarrow (t, E)$

(pl_1^s) $(x \leftarrow \mathsf{prjl}(\langle t, t' \rangle).P \mid P', T, E, I) \rightarrow \tau([t/x]P \mid P', T, E, I)$
if $\tau \downarrow (\langle t, t' \rangle, E)$

(pl_2^s) $(x \leftarrow \mathsf{prjl}(x_i).P \mid P', T, E, I) \rightarrow [\langle x, x' \rangle / x_i](P \mid P', T, E, I)$
x' fresh

(d_1^s) $(x \leftarrow \mathsf{dec}(E(t, u), u').P \mid P', T, E, I) \rightarrow \tau([t/x]P \mid P', T, E, I)$
if $\tau \downarrow (E(E(t, u), u'), E), \tau(u') \in Inv(\tau(u))$

(d_2^s) $(x \leftarrow \mathsf{dec}(x_i, u).P \mid P', T, E, I) \rightarrow ([E(x, C)/x_i] \circ \tau)(P \mid P', T, E, I)$
if $\tau \downarrow (E(u, u), E), C \in K(\tau T_i), u \neq x_i$ and $\tau(u) \in Inv(C)$

(d_3^s) $(x \leftarrow \mathsf{dec}(x_i, u).P \mid P', T, E, I) \rightarrow ([E(t, C)/x_i] \circ \tau)([t/x]P \mid P', T, E, I)$
if $\tau \downarrow (E(u, u), E), E(t, C) \in G_\mu(\tau T_i), u \neq x_i$ and $\tau(u) \in Inv(C)$.

Figure 3: Symbolic reduction (without conditional and assertions)

6.3 Conditional

The symbolic handling of the conditional is the most technical extension we consider. We start by enriching a symbolic configuration with a fourth component I which is a finite set of inequalities $s \neq t$ where $s, t \in \mathcal{M}_V$. We write $I \downarrow$ if for no t, $t \neq t \in I$ and in this case we say that I is consistent. Consider a configuration $c \equiv (\mathsf{asrt}(\mathit{false}).P \mid P', T, E, I)$. Then it is correct to reduce c to err iff $I \downarrow$. Indeed, in this case we can always build a substitution σ such that $\sigma \models E, I$ exploiting the fact that the variables range over *infinite* sets of messages (see appendix A.8 for a proof).

Proposition 6.1 *Let (P, T, E, I) be a symbolic configuration. If $I \downarrow$ then $\exists \sigma\ \sigma \models E, I$.*

Given the proposition 6.1 above, we can rewrite the rule (a^s) in figure 2 as follows:

$$(\mathsf{a}^s) \quad (\mathsf{asrt}(\mathit{false}).P \mid P', T, E, I) \rightarrow \mathsf{err} \text{ if } I \downarrow$$

In general, we note that it is *useless* to evaluate a configuration (P, T, E, I) such that I is *not* consistent.

We can now present in figure 4 the rules to handle the conditional. Rules (m_1^s) and (m_2^s) assign a name to each variable in key position (cf. section 6.1) and respectively handle equality and inequality (they are the symbolic counterpart of (m_1) and (m_2)). The rule

$$
\begin{array}{ll}
(\mathsf{m}_1^s) & ([s=t]P_1, P_2, T, E, I) \;\rightarrow\; (\rho\circ\tau)(P_1, T, E, I) \\
 & \text{if } \tau\downarrow(\langle s,t\rangle, E) \text{ and } ([\tau s = \tau t], \tau E, id) \xrightarrow{*} (\emptyset, E', \rho) \\[1em]
(\mathsf{m}_2^s) & ([s=t]P_1, P_2, T, E, I) \rightarrow \tau(P_2, T, E, I\cup\{s\neq t\}) \\
 & \text{if } \tau\downarrow(\langle s,t\rangle, E) \\[1em]
(\mathsf{e}_1^s) & ([f(\vec{t}) = f(\vec{t'})]Eq, E, \rho) \;\rightarrow\; ([t \vec{=} s]Eq, E, \rho) \; f \text{ constr.} \\[1em]
(\mathsf{e}_2^s) & ([x_i = x_i]Eq, E, \rho) \;\rightarrow\; (Eq, E, \rho) \\[1em]
(\mathsf{e}_3^s) & ([x_i = x_j]Eq, E, \rho) \;\rightarrow\; (\rho' Eq, \rho' E, \rho'\circ\rho) \;\; \text{if } i<j, \rho' = [x_i/x_j] \\[1em]
(\mathsf{e}_4^s) & ([x_i = C]Eq, E, \rho) \;\rightarrow\; (\rho' Eq, \rho' E, \rho'\circ\rho) \;\; \text{if } C\in K(T_i), \rho' = [C/x_i] \\[1em]
(\mathsf{e}_5^s) & ([x_i = \langle t_1, t_2\rangle]Eq, E, \rho) \;\rightarrow\; ([x' = t_1][x'' = t_2]\rho' Eq, \rho' E, \rho'\circ\rho) \\
 & \text{if } x_i \notin \mathit{Var}(\langle t_1, t_2\rangle), \rho' = [\langle x', x''\rangle/x_i] \\[1em]
(\mathsf{e}_6^s) & ([x_i = E(t, C)]Eq, E, \rho) \;\rightarrow\; ([x = t]\rho' Eq, \rho' E, \rho'\circ\rho) \\
 & \text{if } x_i \notin \mathit{Var}(t), C\in K(T_i), \rho' = [E(x, C)/x_i] \\[1em]
(\mathsf{e}_7^s) & ([x_i = E(t, C)]Eq, E, \rho) \;\rightarrow\; ([t = t']\rho' Eq, \rho' E, \rho'\circ\rho) \\
 & \text{if } x_i \notin \mathit{Var}(t), E(t', C)\in G_\mu(T_i), \rho' = [E(t', C)/x_i]
\end{array}
$$

$$
\begin{array}{rl}
[x_i/x_j]E \equiv & x_1 : T_1; \ldots; x_{j-1} : T_{j-1}; \\
 & x_{j+1} : [x_i/x_j]T_{j+1}; \ldots; x_n : [x_i/x_j]T_n \\[1em]
[C/x_i]E \equiv & x_1 : T_1; \ldots; x_{i-1} : T_{i-1}; \\
 & x_{i+1} : [C/x_i]T_{i+1}; \ldots; x_n : [C/x_i]T_n \;.
\end{array}
$$

Figure 4: Symbolic reduction for conditional

(m_1^s) relies on the rules (e_{1-7}^s) to check the satisfiability of the equality (strictly speaking, we should consider $[s = t]P$ as a special syntax for a conditional without else branch). The rules (e_{1-7}^s) operate on triples (Eq, E, ρ), where Eq is a possibly empty list of equations among terms in $\mathcal{M}_V$, E is an environment, and ρ keeps track of the substitutions performed. Basically, the rules (e_{1-7}^s) perform a variant of syntactic unification on the equations Eq which is compatible with the set-membership constraints E. Rules (e_{1-3}^s) should be self-explanatory and rules (e_{4-7}^s) are a direct application of the decomposition theorem 4.10.

We denote with $\rho(P, T, E, I)$ the configuration $(\rho P, \rho T, \rho E, \rho I)$ where ρ is the composition of 'basic' substitutions $\rho_1 \circ \cdots \circ \rho_n$ and ρE stands for $\rho_1(\cdots(\rho_n E)\cdots)$ (the application of 'basic' substitutions to an environment is defined at the bottom of figures 2 and 4).

Next, we adapt a method introduced in [AL00], to show that the rules (e_{1-7}^s) always terminate and give a sound and complete method to check the satisfiability of a list of equations subject to the set-membership constraints E.

Definition 6.2 *Given a triple (Eq, E, ρ), we associate a* rank *$rk(x)$ to every variable x occurring in an environment E' such that $(Eq, E, \rho) \xrightarrow{*} (Eq', E', \rho')$ as follows: if the variable is in E then its rank is its position in E, otherwise the variable inherits the rank of the variable it replaces.*

We note that the maximal rank of a variable occurring in a pair (Eq', E') reachable from (Eq, E) is bound by the size of the list E.

Assume every variable is assigned a rank ranging between 1 and n. Then we define the rank of a term t as 0 if the term is closed and i if i is the maximal rank of a variable occurring in t. We define the complexity of an equation $[s = t]$ with respect to the maximal rank n as

$$\mu([s = t]) = (r_n, \ldots, r_1, max(|s|, |t|)) \tag{1}$$

where r_i is the number of occurrences of variables of rank i in $[s = t]$, and $|s|$ is the number of symbols in s. We define a well-founded partial ordering on triples by setting

$$([s = t]Eq, E, \rho) \succ ([s' = t']Eq', E', \rho') \text{ iff } \mu([s = t]) > \mu([s' = t']) \tag{2}$$

where $>$ denotes the lexicographic ordering. Remark that this ordering is not a lexicographic extension of the ordering on equations to sequences of equations (which is not well-founded), for instance pairs with an empty sequence of equations are incomparable with any other pair.

The domain of a substitution σ is the set $Dom(\sigma) = \{x \mid \sigma x \not\equiv x\}$. We say that a substitution σ is *decreasing* if $\forall x \in Dom(\sigma)\ rk(x) > rk(\sigma x)$. For instance, the identity substitution is decreasing since its domain is empty. We note the following properties of decreasing substitutions.

Lemma 6.3 (1) *The composition of decreasing substitutions is a decreasing substitution.*
(2) *If σ is decreasing then either $\sigma[t = t'] \equiv [t = t']$ or $\mu([t = t']) > \mu(\sigma[t = t'])$.*
(3) *The composition $\sigma \circ [t/x]$ of a decreasing substitution σ and of the substitution $[t/x]$ is decreasing if for all $y \in Var(t)$, $rk(y) \not> rk(x)$ and $y \in Dom(\sigma)$.*

PROOF HINT. (1) $\forall y \in Var(\sigma_1(x))$, $rk(x) > rk(y)$ and either $\sigma_2(y) = y$ or $\forall z \in Var(\sigma_2(y)$ we have $rk(x) > rk(y) > rk(z)$ since σ_2 decreasing.

(2) For an equation $[t = t']$ either no variable is in $Dom(\sigma)$ or at least one is replaced by occurrences of smaller variables, thus decreasing the complexity measure of the equation.

(3) The composition yields $\sigma(y)$ for $y \neq x$, and $\sigma(t)$ for x. Since each variable of t is in $Dom(\sigma)$ it is instantiated by σ yielding variables of a smaller rank (hence smaller than the rank of x since $rk(y) \not> rk(x)$). $\diamond$.

The termination of rules (e_{1-7}) in figure 4 relies on the following technical lemma proven in section A.9 by induction on the order $\succ$.

Lemma 6.4 (termination) *Let $p \equiv ([s = t]Eq, E, \rho)$ be a triple which is reduced by rules (e^s_{1-7}). Then two cases can arise: either $p \xrightarrow{*} ([s' = t']Eq', E', \rho')$ and no rule applies, or we can reduce p to a configuration $(Eq', E', \rho') \equiv \sigma(Eq, E, \rho)$, where σ satisfies: (i) σ is decreasing, (ii) if $s \equiv x$ and $t \notin V$ then $x \in Dom(\sigma)$.*

By iterating lemma 6.4, we can conclude that equations can be eliminated.

Proposition 6.5 *The simplification of triples (Eq, E, ρ) always terminates.*

Concerning soundness and completeness we note the following proposition.

Proposition 6.6 (1) *If $([s = t], Eq, E, \rho) \to ([s \vec{=} t]\rho' Eq, \rho' E, \rho' \circ \rho)$ and $\sigma \models [s \vec{=} t]\rho' Eq$ and $\sigma \models \rho' E$ then $\sigma \circ \rho' \models [s = t], Eq, E$.*

(2) *If $\sigma \models [s = t], Eq, E$ then $([s = t], Eq, E, \rho) \to ([s \vec{=} t]\rho' Eq, \rho' E, \rho' \circ \rho)$ and there is a σ' such that $\sigma' \models [s \vec{=} t]\rho' Eq, \rho' E$ and $\sigma = \sigma' \circ \rho'$.*

PROOF HINT. (1) We proceed by case analysis on the rule applied. We consider a typical case.

(e^s_6) We know $\sigma \models [x = t]\rho' Eq, \rho' E$, $\rho' = [E(x, C)/x_i]$, $x_i \notin Var(t)$, and $C \in K(T_i)$. Obviously $\sigma \circ \rho' \models Eq$. Moreover, since $\sigma(x) = \sigma t$ and $x_i \notin Var(t)$ it follows that $\sigma \circ [E(x, C)/x_i] \models x_i = E(t, C)$.

It remains to prove that $\sigma \circ \rho' \models E$. Suppose $E \equiv x_1 : T_1; \ldots; x_i : T_i; \ldots; x_n : T_n$. We distinguish three cases:

$(j < i)$ Then $\sigma(\rho'(x_j)) = \sigma(x_j)$, $\sigma(\rho' T_j) = \sigma(T_j)$, and we know that $\sigma(x_j) \in S(A(\sigma T_j))$ since $\sigma \models \rho' E$.

$(j = i)$ Then $\sigma(\rho'(x_i)) = E(\sigma t, C)$ and $\sigma(\rho' T_i) = \sigma(T_i)$. Then $\sigma(x) = \sigma t \in S(A(\sigma T_i))$ since $\sigma \models \rho' E$, and $E(\sigma t, C) \in S(A(\sigma T_i))$ since $C \in K(T_i)$.

$(j > i)$ Then $\sigma(\rho'(x_j)) = \sigma(x_j)$ and $\sigma(x_j) \in S(A(\sigma(\rho'(T_j))))$ since $\sigma \models \rho' E$.

(2) We consider the structure of $s = t$ knowing that $\sigma \models s = t$. For all possible cases, we verify that a rule applies (up to symmetry). The substitution σ' is defined as follows:

$$\begin{array}{llll} (\mathsf{e}_1^s) & \sigma' = id & (\mathsf{e}_2^s) & \sigma' = id \\ (\mathsf{e}_3^s) & \sigma' = \sigma[x_j/x_j] & (\mathsf{e}_4^s) & \sigma' = \sigma[x_i/x_i] \\ (\mathsf{e}_5^s) & \sigma' = \sigma[t_1'/x', t_2'/x'', x_i/x_i] & (\mathsf{e}_6^s) & \sigma' = \sigma[t'/x, x_i/x_i] \\ (\mathsf{e}_7^s) & \sigma' = \sigma[x_i/x_i]) \ . & & \end{array} \quad \diamond$$

From the proposition 6.6 above we derive the following properties.

Corollary 6.7 (1) *If* $([s = t], E, id) \xrightarrow{*} (\emptyset, E', \rho)$ *and* $\sigma \models E'$ *then* $\sigma \circ \rho \models [s = t], E$.
(2) *If* $\sigma \models [s = t], E$ *then* $([s = t], E, id) \xrightarrow{*} (\emptyset, E', \rho')$ *and there is a* σ' *such that* $\sigma' \models E'$ *and* $\sigma = \sigma' \circ \rho'$.

PROOF HINT. For (1) iterate proposition 6.6(1) and for (2) iterate proposition 6.6(2). $\diamond$

6.4 Soundness and completeness

We have formulated the symbolic reduction rules so that they are in lockstep with the reduction rules. This leads to a modular and simple approach to the proof of soundness and completeness which is presented next.

Definition 6.8 *Given a set of transition rules* $\mathcal{R}$ *and a configuration* k*, we define*

$$\mathsf{Succ}^{\mathcal{R}}(k) = \{k' \mid k \xrightarrow{r} k' \text{ and } r \in \mathcal{R}\} \ .$$

We define in a similar way a successor function for symbolic configurations:

$$\mathsf{Succ}_s^{\mathcal{R}^s}(c) = \{c' \mid c \xrightarrow{r} c' \text{ and } r \in \mathcal{R}^s\} \ .$$

These functions are extended canonically on configuration sets.

Definition 6.9 *Given a symbolic configuration* $c \equiv (P, T, E, I)$*, the* image *of* c*, noted* $Im(c)$ *is the set* $\{(\sigma P, \sigma T) \mid \sigma \models E, I\}$*. Moreover, we define* $Im(\mathsf{err}) = \{\mathsf{err}\}$*. We extend* Im *on a set of symbolic configurations* $\mathcal{C}$ *by* $Im(\mathcal{C}) = \bigcup_{c \in \mathcal{C}} Im(c)$.

Definition 6.10 (symbolic equivalent) *Given two transition sets* $\mathcal{R}$ *and* $\mathcal{R}^s$ *respectively on configurations and symbolic configurations, we say that* $\mathcal{R}^s$ *is a* symbolic equivalent *of* $\mathcal{R}$ *if*

$$\forall c \equiv (P, T, E, I) \ Im(\mathsf{Succ}_s^{\mathcal{R}^s}(c)) = \mathsf{Succ}^{\mathcal{R}}(Im(c)) \ .$$

The following lemma presents the properties of symbolic equivalence with respect to union and composition (the proof is immediate).

Lemma 6.11 (1) *If ${\mathcal{R}_1}^s$ is a symbolic equivalent of $\mathcal{R}_1$ and ${\mathcal{R}_2}^s$ is a symbolic equivalent of $\mathcal{R}_2$ then ${\mathcal{R}_1}^s \cup {\mathcal{R}_2}^s$ is a symbolic equivalent of $\mathcal{R}_1 \cup \mathcal{R}_2$.*

(2) *If $\mathcal{R}^s$ is a symbolic equivalent of $\mathcal{R}$ then for all $n \geq 1$ and every symbolic configuration c we have :*

$$Im((\mathsf{Succ}_s^{\mathcal{R}^s})^n(c)) = (\mathsf{Succ}^{\mathcal{R}})^n(Im(c))$$

Then a rather long case analysis presented in appendix A.10 allows to derive the following result.

Proposition 6.12 *In the following cases, $\mathcal{R}^s$ is a symbolic equivalent of $\mathcal{R}$.*

$$\begin{array}{llll}
(1) & \mathcal{R} = \{?\}, \mathcal{R}^s = \{?^s\} & (2) & \mathcal{R} = \{!\}, \mathcal{R}^s = \{!^s\} \\
(3) & \mathcal{R} = \{\mathsf{d}\}, \mathcal{R}^s = \{\mathsf{d}_1^s, \mathsf{d}_2^s, \mathsf{d}_3^s\} & (4) & \mathcal{R} = \{\mathsf{pl}\}, \mathcal{R}^s = \{\mathsf{pl}_1^s, \mathsf{pl}_2^s\} \\
(5) & \mathcal{R} = \{\mathsf{m}_1\}, \mathcal{R}^s = \{\mathsf{m}_1^s\} & (6) & \mathcal{R} = \{\mathsf{m}_2\}, \mathcal{R}^s = \{\mathsf{m}_2^s\} \\
(7) & \mathcal{R} = \{\mathsf{a}\}, \mathcal{R}^s = \{\mathsf{a}^s\} \ . & &
\end{array}$$

From this, we derive the generalisation of theorem 5.3 on the soundness and completeness of the symbolic reduction.

Theorem 6.13 (1) *Symbolic reduction always terminates.*

(2) $(P, T, E, I) \xrightarrow{*} \mathsf{err}$ *iff* $\exists\, \sigma \models E, I \ \ \sigma(P, T) \xrightarrow{*} \mathsf{err}$.

PROOF. (1) By proposition 6.5.

(2) We consider the full set of rules $\mathcal{R} = \{?, !, \mathsf{d}, \mathsf{pl}, \mathsf{m}_1, \mathsf{m}_2, \mathsf{a}_1, \mathsf{a}_2\}$ and $\mathcal{R}^s = \{?^s, !^s, \mathsf{d}_1^s, \mathsf{d}_2^s, \mathsf{d}_3^s, \mathsf{pl}_1^s, \mathsf{pl}_2^s, {\mathsf{m}_1}^s, {\mathsf{m}_2}^s, \mathsf{a}^s\}$. We note that $Im((P, T, \emptyset, \emptyset)) = \{(P, T)\}$ and $Im(\mathsf{Succ}_s^n((P, T, \emptyset, \emptyset))) = \mathsf{Succ}^n(Im((P, T, \emptyset, \emptyset))) = \mathsf{Succ}^n((P, T))$ and therefore:

$$\begin{array}{lll}
(P, T) \xrightarrow{*} \mathsf{err} & \text{iff} & \exists\, n \ \ \mathsf{err} \in \mathsf{Succ}^n((P, T)) \\
& \text{iff} & \exists\, n \ \ \mathsf{err} \in Im(\mathsf{Succ}_s^n((P, T, \emptyset, \emptyset))) \\
& \text{iff} & \exists\, n \ \ \mathsf{err} \in \mathsf{Succ}_s^n((P, T, \emptyset, \emptyset)) \\
& \text{iff} & (P, T, \emptyset, \emptyset) \xrightarrow{*} \mathsf{err} \ . \ \diamond
\end{array}$$

7 Assertions

As shown in [AP99], the combination of the conditional and the assertion $\mathsf{asrt}(\mathit{false})$ already allows to express secrecy and authentication properties. Nevertheless expressing authentication in this way is rather cumbersome. In order to obtain simpler specifications, we introduce a predicate $\mathsf{known}(t)$ which is defined by

$$\models_T \mathsf{known}(t) \ \ \text{iff} \ \ t \in S(A(T)) \ .$$

For instance, to verify an authentication property we proceed as follows: (i) when emitting a message subject to authentication we store it, in a crypted form, in the knowledge of the adversary, and (ii) upon reception we authenticate a message by checking with the predicate known that it was stored, and hence emitted, in the knowledge of the adversary.

In the following, the full assertion language we will consider is the following:

$$\varphi ::= \quad \mathit{true} \parallel \mathit{false} \mid t = t' \mid t \neq t' \mid \mathsf{known}(t) \mid \mathsf{secret}(t) \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2$$

where:

$$\begin{array}{lllll}
\models_T \mathit{true} & & & \models_T t = t' & \text{iff } t \equiv t' \\
\models_T t \neq t' & \text{iff } t \not\equiv t' & & \models_T \mathsf{known}(t) & \text{iff } t \in S(A(T)) \\
\models_T \mathsf{secret}(t) & \text{iff } t \notin S(A(T)) & & \models_T \varphi_1 \wedge \varphi_2 & \text{iff } \models_T \varphi_1 \text{ and } \models_T \varphi_2 \\
\models_T \varphi_1 \vee \varphi_2 & \text{iff } \models_T \varphi_1 \text{ or } \models_T \varphi_2 \ . & & &
\end{array}$$

This is equivalent to saying that we consider arbitrary boolean combinations of atomic formulas checking the equality of two messages $t = t'$ and the secrecy of a message $\mathsf{secret}(t)$ with respect to the current knowledge of the adversary.

7.1 Symbolic reduction of assertions

To check symbolically the validity of an assertion we proceed as follows. Without loss of generality, we suppose that the formula φ is presented in *conjunctive normal formal*. Then we take the negation of the assertion and check its satisfiability by reducing it to a reachability problem for a program with a certain structure. To this end, it is convenient to introduce a test $[\mathsf{secret}(X)]P$ whose reduction and symbolic reduction are defined below (the test $[\mathsf{secret}(X)]$ is only used to compile the assertions and it is not part of the official syntax of processes).

Definition 7.1 *If X is a set of terms (X may be empty):*

$$\begin{array}{ll}
(\mathsf{sc}) & ([\mathsf{secret}(X)]P \mid P', T) \to (P \mid P', T) \\
& \mathit{if}\ \forall t \in X \quad t \notin S(A(T)) \\[2ex]
(\mathsf{sc}^s) & ([\mathsf{secret}(X)]P \mid P', T, E, I) \to (P \mid P', T, E, I) \\
& \mathit{if}\ I \downarrow, \forall t \in X \quad t \notin S(G_\mu(T) \cup \mathit{Var}(E)) \ .
\end{array}$$

We note that rule (sc^s) will always test the condition $I \downarrow$. We can express the relationship between rules (sc^s) and (sc) as follows (the proof is presented in appendix A.11).

Proposition 7.2 *Let c be a symbolic configuration. Then:*

$$\mathsf{Succ}_s^{\{\mathsf{sc}^s\}}(c) = \emptyset \quad \mathit{iff} \quad \mathsf{Succ}^{\{\mathsf{sc}\}}(\mathit{Im}(c)) = \emptyset \ .$$

This formula still holds for any set of symbolic configurations.

Next we define an auxiliary symbolic configuration corresponding to a formula φ with no conjunction.

Definition 7.3 *Let (P, T, E, I) be a symbolic configuration and φ be a disjunction of tests of the shape:*

$$\varphi = \bigvee_{i=1,\ldots,m} \mathsf{secret}(t_i) \vee \bigvee_{j=1,\ldots,n} (r_j \neq r'_j) \vee \bigvee_{k=1,\ldots,p} (s_k = s'_k) \vee \bigvee_{l=1,\ldots,q} \mathsf{known}(t'_l)$$

We define the symbolic configuration $c(\varphi)_{(T,E,I)}$ as:

$$\begin{array}{rcl} c(\varphi)_{(T,E,I)} & \equiv & (?x_1.\cdots.?x_m.[x_1 = t_1]\cdots[x_m = t_m] \\ & & [r_1 = r'_1]\cdots[r_n = r'_n] \\ & & [s_1 \neq s'_1]\cdots[s_p \neq s'_p] \\ & & [\mathsf{secret}(\{t'_1, \ldots, t'_q\})]0, T, E, I)\ . \end{array}$$

We point out that if φ does not contain any $\mathsf{known}(t'_l)$ predicate, $c(\varphi)_{(T,E,I)}$ must end with a $\mathsf{secret}(\emptyset)$ construct. With these notations, the length of $c(\varphi)$ is then defined by: $|c(\varphi)| = 2m + n + o + 1$.

The idea behind $c(\varphi)_{(T,E,I)}$ is that this (symbolic) configuration will test the negation of φ. Namely, it will be able to reduce to a configuration $(0, T', E', I')$ iff there exists $\sigma \models E, I$ such that the assertion $\mathsf{asrt}(\sigma\varphi)$ with the adversary knowledge σT is false. We write $\sigma \models_T \varphi$ as a shortcut for $\models_{(\sigma T)} (\sigma\varphi)$.

Proposition 7.4 *Let φ be a disjunction of equality, inequality, secret, and known predicates (as given in definition 7.3). Then :*

$$(\forall \sigma \models E, I \quad \sigma \models_T \varphi) \quad \textit{iff} \quad (\mathsf{Succ})^{|c(\varphi)|}(Im(c(\varphi))) = \emptyset$$

PROOF HINT. We just have to write the constraints associated with a ground reduction of an element of $Im(c(\varphi))$ and notice that there exists such a reduction iff there exists $\sigma \models E, I$ such that $\sigma \not\models_T \varphi$. $\diamond$

We can now derive a result that leads to a symbolic reduction rule for assertions.

Corollary 7.5 *If φ is a disjunction of equality, inequality, secret, and known predicates, then:*

$$(\forall \sigma \models E, I \quad \sigma \models_T \varphi) \quad \textit{iff} \quad \exists (T', E', I') c(\varphi)_{(T,E,I)} \xrightarrow{*} (0, T', E', I')\ .$$

PROOF. We note $k = |c(\varphi)| - 1$.

$$\begin{array}{llll} (\forall \sigma \models E, I \quad \sigma \models_T \varphi) & \text{iff} & (\mathsf{Succ})^{k+1}(Im(c(\varphi))) = \emptyset & \text{by proposition 7.4} \\ & \text{iff} & \mathsf{Succ}^{\{\mathsf{sc}\}}((\mathsf{Succ})^k(Im(c(\varphi)))) = \emptyset & \\ & \text{iff} & \mathsf{Succ}^{\{\mathsf{sc}\}}(Im(\mathsf{Succ}_s^k(c(\varphi)))) = \emptyset & \text{by lemma 6.11} \\ & \text{iff} & \mathsf{Succ}_s^{\{\mathsf{sc}\}}(\mathsf{Succ}_s^k(c(\varphi))) = \emptyset & \text{by proposition 7.2} \\ & \text{iff} & (\mathsf{Succ}_s)^{k+1}(c(\varphi)) = \emptyset\ . & \end{array}$$

We conclude by noticing that condition $(\mathsf{Succ}_s)^{k+1}(c(\varphi)) = \emptyset$ is equivalent to $c(\varphi) \xrightarrow{*} (0, T', E', I')$. $\diamond$

We now have all necessary tools to introduce a function $eval_{(T,E,I)}$ that, given the adversary knowledge T and the constraints E, I, symbolically evaluates an assertion.

Definition 7.6 *Let φ be an assertion. We denote by $red(\varphi)$ the formula φ in 'reduced' conjunctive normal form so that $red(\varphi)$ ::= true | false | $\varphi_1 \wedge \ldots \wedge \varphi_n$ where $\varphi_1, \ldots, \varphi_n$ are disjunctions of equality, inequality, secret, and known predicates. We define the symbolic evaluation of φ in (T, E, I) as follows:*

$$eval(\varphi)_{(T,E,I)} = \begin{cases} \mathit{false} & \text{if } red(\varphi) = \varphi_1 \wedge \ldots \wedge \varphi_n \text{ and } \exists i \; c(\varphi_i)_{(T,E,I)} \xrightarrow{*} (0, T', E', I') \\ & \quad \text{or } red(\varphi) = \mathit{false} \text{ and } I \downarrow \\ \mathit{true} & \text{otherwise .} \end{cases}$$

From corollary 7.5, it is easy to derive the following result.

Corollary 7.7 *Let φ be an assertion. Then:*

$$(\forall \sigma \models E, I \;\; \sigma \models_T \varphi) \;\; \textit{iff} \;\; eval(\varphi)_{(T,E,I)} = \mathit{true}$$

We can now present the new symbolic reduction rule for assertions.

Definition 7.8

$$(\mathsf{a}^s) \quad (\mathsf{asrt}(\varphi).P \mid P', T, E, I) \rightarrow \begin{cases} \mathsf{err} & \text{if } eval(\varphi)_{(T,E,I)} = \mathit{false} \\ (P \mid P', T, E, I) & \text{otherwise} \end{cases}$$

This rule obviously preserves the termination property. It just remains to show soundness and completeness for the *full* system including the new assertion rule.

Theorem 7.9 (soundness and completeness) *Let (P, T, E, I) be a symbolic configuration. Then:*

$$(P, T, E, I) \xrightarrow{*} \mathsf{err} \;\; \textit{iff} \;\; \exists \sigma \models E, I \;\; \sigma(P, T) \xrightarrow{*} \mathsf{err} \; .$$

PROOF. It follows from definition of (a^s) and corollary 7.5 that:

$$\begin{cases} \mathsf{err} \in \mathsf{Succ}_s^{\{\mathsf{a}^s\}}(c) \;\; \text{iff} \;\; \mathsf{err} \in \mathsf{Succ}^{\{\mathsf{a}\}}(Im(c)) \\ \mathsf{err} \notin \mathsf{Succ}_s^{\{\mathsf{a}^s\}}(c) \text{ implies } Im(\mathsf{Succ}_s^{\{\mathsf{a}^s\}}(c)) = \mathsf{Succ}^{\{\mathsf{a}\}}(Im(c)) \; . \end{cases}$$

We combine this fact with proposition 6.12 (symbolic equivalence for all rules but the pair a, a^s) to show by induction on n:

$$\mathsf{err} \notin \bigcup_{i=1}^{n} \mathsf{Succ}^i(Im(c)) \text{ implies } \mathsf{Succ}^n(Im(c)) = Im(\mathsf{Succ}_s^n(c)) \; .$$

From this, we deduce:

$$\mathsf{err} \in \bigcup_{i=1}^{n} \mathsf{Succ}^i(Im(c)) \;\; \text{iff} \;\; \mathsf{err} \in \bigcup_{i=1}^{n} Im(\mathsf{Succ}_s^i(c)) \; .$$

And this last property implies:

$$(P, T, E, I) \xrightarrow{*} \mathsf{err} \;\; \text{iff} \;\; \exists \sigma \models E, I \;\; \sigma(P, T) \xrightarrow{*} \mathsf{err} \; \diamond$$

7.2 A prototype implementation

The symbolic reduction system we have described forms the basis of a prototype verifier developed in CAML by one of the authors [Van00]. The most important optimisation performed by the verifier is the *pruning of equivalent schedulings* of parallel processes. Indeed, it was already observed in [AP99, AL00] that all reductions but input are strongly confluent up to equivalence and this remark can be lifted to some extent to symbolic reductions.

Our analyser also performs *lazy instantiation of variables in key position.* In practice, we found that variables in key position are quite often instantiated automatically. Intuitively, one can attribute this phenomenon to the fact that 'good' protocols do not encrypt messages with a name chosen by the adversary.

Of course, the time required to do a full search is heavily dependant on the modelling of the protocol. In first approximation, we can say that *one-session* runs of typical protocols from the literature are usually performed in less than $0.1s$. Figure 5 gives some figures for a flawed variant of the Otway-Rees protocol presented in [BAN89] (the full specification and the outcome of our analyser is given in appendix A.12). All measures were done on a Pentium II at 266MHz. The time for the 3 parallel sessions test can in fact be down-sized to $48s$ if we write the 3 servers in sequence rather than in parallel, which does not affect the reachability of an error.

# initiators	# responders	# servers	time
1	1	1	$< 0.01s$
1	1	2	$0.02s$
2	2	2	$0.28s$
3	3	3	$280s$

Figure 5: Times for the analysis of the Otway-Rees protocol

Obviously our tool suffers from the intrinsic complexity of the problem. Nevertheless our implementation is performant enough to handle around 10 parallel processes each composed of $2, 3$ alternations of input-output. This allows to analyse (at least) 2 parallel sessions of most typical protocols.

To achieve the current performance, much care has been taken in the choice of the algorithmic structures in order to maximise the speed and minimise the memory usage of the tool. The current version has a speed-up with respect to our first 'naive' prototype which is greater than 500. An interesting feature is that in practice the memory usage is almost constant and quite small (around 1 MByte), so finding an error in a protocol is mainly a matter of time.

8 NP-completeness

In this section, we argue that a significant fragment of the symbolic reduction system can be implemented to run in NPTIME thus matching the lower bound proved in section 3. This result was conjectured in [ALV00]. Independently, Rusinowitch and Turuani [RT01] have obtained an NP-completeness result for a related system (still unpublished). Their approach is quite different from ours, in particular it does not rely on symbolic reduction. An advantage of their approach is that it handles more general cases such as *complex* symmetric keys. A disadvantage is that it relies more than ours on non-deterministic choice and therefore it seems to lead to less efficient implementations.

If we restrict our attention to symmetric keys and the parallel composition of processes of the shape: $?x = E(\ldots E(y, C_1), \ldots, C_n).Q$ where Q can be $!E(\ldots E(y, D_1), \ldots, D_m).0$, $!E(\ldots E(D, D_1), \ldots, D_m).0$, or $\mathsf{asrt}(\mathit{false}).0$ then a rather straightforward implementation of symbolic reduction entails the following result.

Proposition 8.1 *The reachability problem for processes of the shape above is* NP*-complete.*

PROOF. We have shown in section 3.1 that the problem in NP-hard. Next we show that the symbolic reduction system can be implemented to solve the problem in NPTIME. Let K be the finite set of constants occurring in the initial configuration (P, T_0). We adopt the following abbreviations:

$$\begin{array}{ll} C_n \ldots C_1 y & \text{for } E(\ldots E(y, C_1), \ldots, C_n) \\ C_n \ldots C_1 C & \text{for } E(\ldots E(C, C_1), \ldots, C_n) \end{array}$$

and denote with α, β finite words over K.

Given a program P of the shape above, we guess in polynomial time an execution schedule. We then obtain a sequential thread of the shape:

$$P_k \equiv ?x_k = \alpha_k y_k.!\beta_k(y_k).\ldots.?x_{m-1} = \alpha_{m-1} y_{m-1}.!\beta_{m-1}(y_{m-1}).?x_m = \alpha_m y_m.\mathsf{asrt}(\mathit{false}).0 \ .$$

where $k \leq m$ and initially $k = 0$. The (y_j) in the output $!\beta_j(y_j)$ is optional.

Suppose $(P_0, T_0, \emptyset) \xrightarrow{*} (P_k, T_k, E_k)$ and suppose the head of P_k has the shape $?x_k = \alpha_k y_k.!\beta_k y_k$ (the case $?x_k = \alpha_k y_k.!\beta_k$ is similar). We distinguish the following cases:

$\alpha_k \in K(T_k)^*$. This means that the prefix α_k and y_k can be synthesised by the adversary. Two cases may arise:

$\beta_k \in K(T_k)^*$. Then we reduce to (P_{k+1}, T_k, E_k) since $\beta_k y_k$ can be synthesised from T_k.

$\beta_k = \beta' C \beta'', \beta' \in K(T_k)^*, C \notin K(T_k)$. Then we reduce to $(P_{k+1}, T_k \cup \{C\beta'' y_k\}, E_k, y_k : T_k)$.

$\alpha_k = \alpha' C \alpha'', \alpha' \in K(T_k)^*, C \notin K(T_k)$. We select non-deterministically $C\alpha'' t \in G_\mu(T_k)$ (if no such term exists we are done). As above, two cases may arise:

$\beta_k \in K(T_k)^*$. Then we reduce to $(P_{k+1}, T_k \cup \{t\}, E_k)$.

$\beta_k = \beta' C \beta'', \beta' \in K(T_k)^*, C \notin K(T_k)$. Then we reduce to $(P_{k+1}, T_k \cup \{C\beta'' t\}, E_k)$.

We note that the symbolic reduction maintains the invariant that the size of T_k represented as a directed acyclic graph (dag) is bound by the size of $T_0 \cup \{\beta_0(y_0), \ldots, \beta_{m-1}(y_{m-1})\}$ which is bound in turn by the size of the initial configuration. Since the computation of $K(T_k)$ and $G_\mu(T_k)$ can be implemented to run in deterministic polynomial time, we have shown that the reachability of the invalid assertion can be checked in NPTIME. $\diamond$

In the following, we present in a semi-formal style a generalization of this result. We consider again symmetric keys and programs of the shape:

$$!o_1.?x = i_1.!o_2.?x = i_2 \ldots !o_n.?x = i_n.\mathsf{asrt}(\mathit{false}).0 \tag{3}$$

As in section 3, the notation $?x = i$ stands for the input of a message which is filtered against a pattern i. In what follows, i does *not* need to be linear (non-linear patterns can be compiled into our basic formalism using the conditional). We assume that variables bound by the filters have been renamed so that:

$$\mathit{Var}(i_k) \cap (\mathit{Var}(i_1) \cup \cdots \cup \mathit{Var}(i_{k-1})) = \emptyset\ .$$

The variables in the output messages satisfy the condition:

$$\mathit{Var}(o_k) \subseteq \mathit{Var}(i_1) \cup \cdots \cup \mathit{Var}(i_{k-1})\ .$$

As in proposition 8.1, we can extend the decision procedure to the parallel composition of programs of the shape (3) without affecting the complexity class. To do this, we guess non-deterministically the interleaving of the output-input actions.

As pointed out in section 6, our symbolic reduction procedure is basically a variant of syntactic unification which is compatible with certain set-membership constraints. Our data structures and algorithms are directly inspired by those proposed for (polynomial time) syntactic unification in [CB83] (see also [BN98] for a more accessible reference). In this approach, terms are represented as dags with *shared* variables, *i.e.* a variable occurring in the terms is represented by exactly one node.

Data types We rely on three record data types.

```
term = record             term_list = record     term_list_tag = record
     {isvar  : bool,         {tr     : term,        {tp     : term,
      is     : term,          next   : term}         pred   : term,
      stamp  : nat,                                    tag    : nat}
      fn     : string,
      arg    : term_list,
      cstr   : term_list_tag}
```

To each node we associate a record of type `term`. The field `isvar` is a boolean specifying whether the term is a variable. The field `is` is a pointer to another term. It is initialised to `nil` and it allows to share terms during the unification procedure. The counter `stamp` is utilised in the occur check (to avoid visiting a term several times) and it is initialised to `0`. The string `fn` represents a function symbol. In our case either a pair `pair`, or an encryption function `E(_,C)`, or a constant `C`. The pointer `arg` points to a list of terms of record type `term_list` representing the arguments of the function. Here, `tr` is the pointer to the term and `next` is the pointer to the next term in the list. Finally, the pointer `cstr` points to

```
procedure C(t: term, T:term_list_tag)
var t1, t2, s: term; T': term_list_tag;
(1)  s:=Find(t);
(2)  (let T'=Min(s.cstr,T) in
(3)  if s.cstr/=T' then
(4)    s.cstr := T';
(5)    case s
(6)    s.isvar             : skip
(7)    s.fn=pair           : t1:=s.arg.tr; t2:=s.arg.next.tr;
                             C(t1,T); C(t2,T)
(8)    s.fn=E(_,C), K(C,T) : t1:=s.arg.tr; C(t1,T)
(9)    s.fn=E(_,C), not K(C,T)
(10)                       : let G=Gmu(C,T) in
(11)                             if G=nil then Stop
(12)                             else t1:=Guess(G);
(13)                                  if U(t1,s.arg.tr) then skip
(14)                                  else Stop
(15)   s.fn=C, K(C,T)      : skip
(16)   s.fn=C, not K(C,T)  : Stop
```

Figure 6: Membership set-constraints propagation procedure

a list of terms which represents the knowledge of the adversary. In our case, the current knowledge is given by the sets $T_i = \{o_1, \ldots, o_i\}$, $i = 1 \ldots, n$ which is a list of terms tagged by a natural number. We assume that the first element tagged i points to o_i and the last element tagged 1 points to o_1. The field `pred` is employed to scan the list.

Initialisation and main program In a program of the shape (3), an erroneous configuration is reachable if and only if the following set-membership constraints can be satisfied:

$$i_1 \in S(A(T_1)), \ldots, i_n \in S(A(T_n)) \ . \tag{4}$$

Initially, we build the dags corresponding to the terms $i_1, \ldots, i_n, o_1, \ldots, o_n$ and the lists $T_1, \ldots, T_n$. We denote with v be number of vertices and e the number of edges of the resulting dag. To check the satisfiability of the condition (4) the main program calls the procedure `C` (described next) on each pair term-constraint:

$$\mathtt{C(i1,T1)}; \ldots; \mathtt{C(in,Tn)}; \mathtt{true} \ . \tag{5}$$

Constraint-propagation We present the procedure `C` in figure 6. The basic idea is to propagate the constraint `T` towards the leaves of the term `t` according to the decomposition theorem 4.10. The procedure relies on a number of procedures that we describe in the following.

`Find` The procedure `Find(t)` returns the last element of the chain pointed by `t.is` (possibly `t` itself). This procedure is computed in $O(v)$.

```
function Find(t: term): term
  if t.is = nil then t else Find(t.is)
```

`Min(T,T')` The procedure `Min` returns the most restrictive set-membership constraint between `T` and `T'` where we take `nil` as the most liberal set-constraint. Since the rank of the set-membership constraint is explicitly presented in the `tag` field, this procedure is computed in $O(1)$.

```
function Min(T,T':term_list_tag): term_list_tag
  case (T,T')
  (nil,_) : T'
  (_,nil) : T
  (_,_)   : if T.tag < T'.tag then T else T'
```

`K(C,T)`, `Gmu(C,T)` The function `K(C,T)` checks whether the constant `C` is in $K(T)$ and the function `Gmu(C,T)` computes the list of terms t such $E(t,C) \in G_\mu(T)$. Both functions can be computed as follows. Let $\{C_1,\ldots,C_p\}$ be the set of constants occurring in the program. We allocate an array `A` with the following initialisation:

```
  A = array [1..p] of {known:bool, point: term_list}
  A[i].known := false;
  A[i].point := nil
```

We then visit the terms pointed by `T`. If we cross a term $E(t,C_i)$ such that `A[i].known = false`, we insert t in the corresponding list. When we cross a constant `C_i` we set `A[i].known := true` and visit the terms in the corresponding list. This procedure can be computed in $O(e)$. When we are done, the list corresponding to `C_i` contains the result for the function `Gmu(C_i,T)`.

`Guess(G)` The function `Guess(G)` non-deterministically selects a term in the list `G`. This can be done in $O(v)$. Note that thanks to sharing we can conclude that the size of the minimum generator set is linear in the size of the program.

`Stop` The function `Stop` halts the computation and returns `false`, having found that the constraints are not satisfiable. This can be done in $O(1)$.

`U(t1,t2)` At control point `(13)` we call the unification procedure `U` that tries to unify the current term `E(t,C)` with an element `E(s,C)` in the minimum generator set. The procedure is presented in appendix A.13. This is the unification procedure described in [BN98] with the exception of procedure `Union(t1,t2)` which let the field `t1.is` point to `t2`. In our case, this procedure must also propagate the set-membership constraint of `t1` to the term `t2`: if we identify the term `t1` with the term `t2` then `t2` must satisfy the set-membership constraint of `t1` too.

Complexity We argue that the main program (5) runs in NPTIME. A flow analysis reveals that:

- A call to `C(t,T)` either terminates or decreases the rank of the constraint in the field `cstr` and makes either one or two calls to `C` or one call to `U`. Since we have n ranks and v nodes, the situation where the rank is decreased may arise at most nv times.

- A call to `U(t1,t2)` either terminates or calls `Union` on `Find(t1), Find(t2)` and possibly their arguments. `Union` in turn makes `Find(t1)` unreachable and may call `C`. The number of calls to `Union` is then in $O(e)$.

9 Decidability of iteration without pairing

In this section we show that we can still decide the reachability problem for a certain class of iterated processes without pairing. This class allows to encode the *ping-pong protocols* studied by Dolev, Even, and Karp [DEK82]. They give an $O(n^3)$ algorithm (n being the size of the protocol) that checks 'correctness' by computing the intersection of a context-free language and a regular language. Our approach is quite different as it relies on recognisable relations (or equivalently rational transductions). The algorithm we derive still runs in PTIME and it allows to handle more general protocols (cf. section 9.3). Moreover, it opens the way to more general results such as: (i) the decidability of any first-order formula constructed on the reachability predicate, and (ii) the possibility of starting from an infinite (regular) initial knowledge (this can be used for parametrised verification).

We consider first the case of symmetric keys and abbreviate a term $E(\ldots(E(t, C_n), \ldots), C_2), C_1)$ with $C_1C_2\ldots C_nt$. We will also denote with $\alpha, \beta, \ldots$ finite words over the set of names $\mathcal{N}$.

We consider processes that are the parallel composition of iterated threads, where a thread is either a filtered input followed by an output or a filtered input followed by an invalid assertion (cf. hypotheses of proposition 8.1):

$$\begin{array}{lcl} P & ::= & P \mid P \mid \mathsf{it}\, Q \\ Q & ::= & ?x = \alpha y.!\beta y.0 \mid ?x = \alpha y.\mathsf{asrt}(\mathit{false}).0\ . \end{array}$$

Let us consider a configuration (P, T_0) where P is a process of the type specified above and $T_0 \subseteq_{fin} \mathcal{M}$ is a set of messages not containing the pairing constructor. We denote with K the set of names occurring in either P or T_0 plus a fresh name F to signal an invalid assertion.

We regard a process $\mathsf{it}\,?x = \alpha y.!\beta y.0$ as a rewrite rule $\alpha y \mapsto \beta y$ and a process $?x = \alpha y.\mathsf{asrt}(\mathit{false}).0$ as a rewrite rule $\alpha y \mapsto F y$. Let R_P be the set of rewrite rules associated to the program P. The reduction of a configuration (P, T) can then be described as

$$(P, T) \to (P, T \cup \{\beta t\}) \quad \text{if } \alpha t \in S(A(T)) \text{ and } \alpha y \mapsto \beta y \in R_P \tag{6}$$

and the reachability problem reduces to checking whether $(P, T_0) \xrightarrow{*} (P, T')$ and $Ft \in T'$.

9.1 Simplification of derivations

We note that, in principle, the message αt offered by the adversary in the reduction (6) may contain in t the pairing constructor. We show next that if pairing does not occur in the initial knowledge T_0 then we can restrict our attention to reductions where no pairing is used. The simple idea is to replace pairs synthesised by the adversary by some term t_0 in the initial knowledge. To eliminate pairs, we define $\overline{t}$ for a term $t \in S(A(T))$ as follows:

$$\begin{array}{ll} \overline{C} = C & \text{if } C \in \mathcal{N} \\ \overline{Ct} = C\overline{t} & \\ \overline{\langle t_1, t_2 \rangle} = t_0 & \text{where } t_0 \text{ is some fixed term in } T_0. \end{array}$$

From a derivation $(P_0, T_0) \xrightarrow{*} (P_n, T_n)$, we derive the sequence $(P_0, \overline{T_0}) \xrightarrow{*} (P_n, \overline{T_n})$ by stating that if s is emitted (respectively received) in the original sequence then $\overline{s}$ is emitted (respectively received) in the derived sequence. For general processes, the second derivation is usually not a correct derivation. The next proposition states that this is not the case for our restricted class and gives the result we are looking for. The related proof is given in appendix A.14.

Proposition 9.1 *Let* $(P_0, T_0) \xrightarrow{*} (P_n, T_n)$ *then* $(P_0, \overline{T_0}) \xrightarrow{*} (P_n, \overline{T_n})$ *is a correct derivation, moreover* $(P_0, T_0) \xrightarrow{*} (P_n, T_n) \rightarrow \mathsf{err}$ *iff* $(P_0, \overline{T_0}) \xrightarrow{*} (P_n, \overline{T_n}) \rightarrow \mathsf{err}$.

9.2 A polynomial time algorithm for deciding reachability

Given the previous result, we assume in the following that the operation of synthesis does *not* introduce pairs. Then it is possible to understand the reduction of configurations via the notion of *prefix word rewriting* which is defined next. Let R be a finite set of rewrite rules of the form $\alpha y \mapsto \beta y$ where $\alpha, \beta \in K^*$. The *prefix* rewrite relation induced by R is the least relation $\mapsto$ on K^* such that $\alpha\gamma \mapsto \beta\gamma$ whenever $\gamma \in K^+$ and $\alpha y \mapsto \beta y \in R$. The reflexive transitive closure of the relation is denoted by $\overset{*}{\mapsto}$. The requirement γ not empty is not standard but we can go back to usual word prefix rewriting by replacing each rule $\alpha y \mapsto \beta y$ by the $|K|$ rules $\alpha C y \mapsto \beta C y$ for all $C \in K$. This new system generates the same relation and the transformation is linear in $|K|$. This allows to use the classical result on regular languages and prefix rewrite systems that we state now.

Given a language L and a finite set R of rewrite rules defining a prefix rewrite relation $\mapsto$, the set of successors of L is defined as:

$$Post_R^*(L) = \{\delta \mid \exists \gamma \in L \ \gamma \overset{*}{\mapsto} \delta\} \ .$$

The relation $\overset{*}{\mapsto}$ is a recognisable relation as shown by [Buc64, Cau92]. In particular, we will rely on the following fact.

Fact 9.2 *If L is a regular language, then $Post_R^*(L)$ is a regular language. Given an automaton $\mathcal{A}$ accepting L, we can compute in time $O(|\mathcal{A}||R|^2)$ an automaton accepting $Post_R^*(L)$ ($|R|$ is the size of R, $|\mathcal{A}|$ is the size of $\mathcal{A}$).*

Consider a set $T \subseteq_{fin} K^+$. Then the operations of analysis and synthesis performed by the adversary can be represented by the rules:

$$\begin{array}{lll} Cy \mapsto y & \text{if } C \in K(T) & \text{(analysis)} \\ y \mapsto Cy & \text{if } C \in K(T) & \text{(synthesis).} \end{array}$$

Initially, we adjoin these rules to R_P for all the constants $C \in K(T_0)$. However during the reduction the sets T and $K(T)$ may evolve forcing the introduction of new rules. We may compute this information iteratively by defining a sequence (K_n, R_n) as follows:

$$\begin{array}{ll} K_0 = K(T_0) & R_0 = R_P \cup \{C\gamma \mapsto \gamma, \gamma \mapsto C\gamma \mid C \in K_0\} \\ K_{n+1} = Post^*_{R_n}(T_0) \cap K & R_{n+1} = R_n \cup \{C\gamma \mapsto \gamma, \gamma \mapsto C\gamma \mid C \in K_{n+1}\}. \end{array}$$

Since $K_n \subseteq K_{n+1}$ and K is finite the iteration converges to a pair (K_p, R_p) such that $K_p = K_{p+1}, R_p = R_{p+1}$ and we define the *closure of* R to be $Cl(R) = R_p$. Computing $Cl(R)$ requires at most $|K|$ steps. Each step involves computing an automaton recognising $Post^*_{R_n}(T_0)$ which is done in $O(|T_0|(|K| + |R_P|)^2)$ and testing if $C \in Post^*_{R_n}(T_0)$ for all C's which is also done in $O(|T_0|(|K| + |R|)^2)$. By construction, we have that the image of T_0 by $Cl(R)$ is stable under synthesis and analysis $S(A(Post^*_{Cl(R)}(T_0))) = Post^*_{Cl(R)}(T_0)$. The closure operator Cl depends on T_0 and we assume in the remaining of this section that this initial knowledge T_0 is some fixed set (hence we write Cl instead of Cl_{T_0}).

Theorem 9.3 *Under the hypotheses above,* $(P, T_0) \xrightarrow{*} \mathsf{err}$ *iff* $Ft \in Post^*_{Cl(R)}(T_0)$ *for some* t*, and this can be decided in time polynomial in the size of the processes and the initial knowledge.*

PROOF. We remark that the hypothesis that the initial knowledge is a finite set of messages is not required and that the result holds for any initial set which is a (possibly infinite) regular set.

(i) $S(A(T_n)) \subseteq Post^*_{Cl(R)}(T_0)$. The proof is by induction on the number n of reduction steps. The case $n = 0$ is obvious. Let us assume that $(P, T_0) \xrightarrow{*} (P_n, T_n) \rightarrow (P_{n+1}, T_{n+1})$ with $S(A(T_n)) \subseteq Post^*_{Cl(R)}(T_0)$. The only case to consider is $T_{n+1} = T_n \cup \{t\}$ where $t = \beta\gamma$ such that $\alpha\gamma \in S(A(T_n))$. By induction hypothesis $\alpha\gamma \in S(A(T_n)) \subseteq Post^*_{Cl(R)}(T_0)$, therefore $\beta\gamma \in Post^*_{Cl(R)}(T_0)$ since $\alpha y \mapsto \beta y \in R$. The closure of $Post^*_{Cl(R)}(T_0)$ by synthesis and analysis yields the result.

(ii) $Post^*_{Cl(R)}(T_0) \subseteq S(A(T_n))$. We consider the sequence $Post^*_{R_n}(T_0)$ occurring in the computation of $Cl(R)$ and we show that for all $s \in Post^*_{R_n}(T_0)$, there is some sequence $(P_0, T_0) \xrightarrow{*} (P_p, T_p)$ with $s \in S(A(T_p))$. To a rewrite sequence $s_0 \mapsto \ldots \mapsto s_m = s$, $s_0 \in T_0$, we associate the pair (n, m) where m is the length of the sequence and n is such that $s \in Post^*_{R_n}(T_0)$ and $s \notin Post^*_{R_l}(T_0)$ with $l < n$. This measure defines a well-founded ordering and the proof is by induction on this ordering.

Base case. This means that $s \in T_0$ and the property holds.

Induction step. $s_0 \mapsto \ldots \mapsto s_{m-1} \mapsto s_m = s$. The last rule is some $\alpha y \mapsto \beta y$ with $s_{m-1} = \alpha s'$ and $s = \beta s'$ or $Cy \mapsto y$ with $s_{m-1} = Cs'$ and $s = s'$ or $y \mapsto Cy$ with $s = Cs_{m-1}$. In these latter cases, we have that C is smaller than s in our ordering, therefore there is some $(P_0, T_0) \xrightarrow{*} (P_p, T_p)$ with $C \in S(A(T_p))$. Moreover there is some $(P_0, T_0) \xrightarrow{*} (P_k, T_k)$ such that $s_{m-1} \in S(A(T_k))$.

Since we consider iterated processes, we also have a reduction

$$(P_0, T_0) \xrightarrow{*} (P_p \mid P_k, T_p \cup T_k) \text{ with } C, s_m \in S(A(T_p \cup T_k)) \ .$$

Therefore either $s \in S(A(T_p \cup T_k))$ if s is obtained from s_{m-1} by a rule $Cy \mapsto y$ or $y \mapsto Cy$, or there is a reduction by a process $?s = \alpha y.!\beta y.0$ which yields $T_p \cup T_k \cup \{s\}$. $\diamond$

Remark 9.4 *Actually a stronger result holds since we have shown that the relation $\overset{*}{\mapsto}$ is a rational relation: the first-order logic based upon atoms $s \overset{*}{\mapsto} t$ with the usual boolean connectives and quantification on configurations (i.e. words) is decidable.*

9.3 Extensions and ping-pong protocols

An interesting extension concerns public keys which can be handled as follows: to a process

$$\mathsf{it}\ ?x = \alpha y.z \leftarrow \mathsf{dec}(y, C).!\beta z.0$$

we associate the rules $\alpha C' y \mapsto \beta y$ for all C' such that $C \in Inv(C')$ and we proceed as before. Slight variations on the process grammar -usually needed in the real encoding of processes-can be handled in the same way.

As a second extension, we consider the use of an arbitrary number of alternations of filtered inputs and outputs (without shared variables between two input-output steps). The idea of the construction remains the same, but it is a little bit more complex. The processes that we consider consist of an iteration of a parallel composition of elementary processes. Each elementary process can be represented as a sequence of rewrite rules $\alpha_1 y_1 \mapsto \beta_1 y_1, \ldots, \alpha_p y_p \mapsto \beta_p y_p$. It is not possible to take R as the set of all these rewrite rules, since there is an ordering on these rules depending on the structure of the process. For instance, consider the process:

$$\mathsf{it}\,(?x_1 = C_1 y_1.!C_2 y_1.?x_2 = C_0 y_2.!C_1 y_2.0)$$

and $T_0 = \{C_0\}$. The set of rules is $\{C_1 y_1 \mapsto C_2 y_1, C_0 y_2 \mapsto C_1 y_2\}$ which implies that, *e.g.*, $C_1 C_0, C_2 C_0$ can be known by the adversary. This does not respect the intended behaviour of the process that actually cannot perform any action.

The decision algorithm consists of two rules $\mathcal{R}_1, \mathcal{R}_2$ which transform a pair (rewrite rules, process expression) into a simpler pair using the closure operation defined in the previous section 9.2. Given some -fixed- initial knowledge T_0, $Cl(R)$ denotes the closure of the set of rewrite rules R related to T_0.

$$\begin{array}{lllll}
(\mathcal{R}_1) & (R, P) & \to & (Cl(R), P) \\
 & & & \textit{if } R \neq Cl(R) \\
(\mathcal{R}_2) & (R, \mathsf{it}\,(?x = \alpha y.!\beta y.P) \mid P') & \to & (R \cup \{\alpha y \mapsto \beta y\}, \mathsf{it}\, P \mid P') \\
 & & & \textit{if } Post^*_R(T_0) \cap \{\alpha\gamma \mid \gamma \in K^+\} \neq \emptyset
\end{array}$$

Proposition 9.5 *The rewrite system $\mathcal{R}_1, \mathcal{R}_2$ is terminating and confluent.*

Termination is straightforward, and the fact that $Cl(R \cup \{\alpha \to \beta\}) = Cl(Cl(R) \cup \{\alpha \to \beta\})$ yields local confluence, hence confluence. Assuming the same coding as for simpler processes, we can state the following proposition.

Proposition 9.6 *$(P, T_0) \xrightarrow{*} \mathsf{err}$ iff $Ft \in Post^*_{Cl(R)}(T_0)$ which can be decided in time polynomial in the size of the processes and the initial knowledge.*

This proposition can be used to check the correctness of ping-pong protocols which describe the exchange of messages between two participants. At each step each participant applies a sequence of decryptions and encryptions to the last message received and sends it back to the other participant.

One of the simplest examples of such protocols can be described as follows: We have a finite set of participants $I, J, \ldots$.

> If participant S wants to make sure a secret information SEC is received exclusively by participant R, it sends to the latter the secret message SEC encrypted with the public key of R. The receiver R decrypts the message, encrypts it with the public key of S and sends it back to S.

In presence of a dishonest participant A, this protocol is insecure. For instance, A can intercept the message from S, send it to R as if he was the original sender, let R decrypt the message and send it back to A encrypted with the public key of A. Here is a model that will detect this type of attack.

• To each channel between a participant I and J, we associate a symmetric key $C_{I,J}$ which belongs to the initial knowledge (hence any message can be intercepted). For a participant I, we denote with C_I^{Pub} its public key and with C_I^{Priv} its secret key.

• Assume SEC is some name used nowhere else. The (iterated) protocol describing the communication between a sender S and a receiver R is the (possibly iterated) parallel composition of the processes

$$?x = start.!C_{S,R}C_R^{Pub}SEC.?y.z \leftarrow \mathsf{dec}(y, C_S^{Priv}).[z = SEC]\ldots$$

for S and

$$\begin{array}{l} ?x.[x = C_{S,R}x'].y \leftarrow \mathsf{dec}(x', C_R^{Priv}).!C_{R,S}C_S^{Pub}y.0 \mid \\ ?x = C_{A,R}x'.y \leftarrow \mathsf{dec}(x', C_R^{Priv}).!C_{R,A}C_A^{Pub}y.0\ . \end{array}$$

for R. To specify that the name SEC must stay secret we add the following observer which runs in parallel with the other processes.

$$?x = SEC.\mathsf{asrt}(false).0\ .$$

• The initial knowledge includes the name $start$, the symmetric keys $C_{I,J}$ representing the public channels, the public key C_I^{Pub} of each participant, and the private key of a dishonest participant C_A^{Priv}.

Iterated versions of this type of protocols can be handled in our approach by introducing suitable (prefix) rewrite rules. The flexibility of our model allows to encode protocols which are not ping-pong protocols, for instance some participant may interact with several participants in the same round.

References

[AL00] R. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In *Proc. CONCUR00, Springer LNCS 1877*, 2000. Also RR-INRIA 3915.

[ALV00] R. Amadio and D. Lugiez and V. Vanackere. On the reachability problem in cryptographic protocols. *Workshop on issues in the theory of security*, Geneve, 2000. Short presentation of results in [AL00] and in Vanackere's DEA.

[AP99] R. Amadio and S. Prasad. The game of the name in cryptographic tables. In *Proc. ASIAN99, Springer LNCS 1742*, 1999.

[AR00] M. Abadi and P. Rogaway. Reconciling two views of cryptography. In *Proc. IFIP TCS, Sendai*, 2000.

[BAN89] M. Burrow and M. Abadi and R. Needham. A logic of authentication. Proceedings of the Royal Society of London, 426:233–271,1989.

[BN98] F. Baader and T. Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998.

[BDNP99] M. Boreale, R. De Nicola, and R. Pugliese. Proof techniques for cryptographic processes. In *Proc. IEEE Logic in Comp. Sci.*, 1999.

[Bol96] D. Bolignano. Formal verification of cryptographic protocols. In *Proc. ACM Conf. on Computer Communication and Security*, 1996.

[Buc64] R. Büchi. Regular canonical systems. *Arch. Math. Logik Grundlag.*, 6:91–111, 1964.

[Cau92] D. Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106:61–86, 1992.

[CB83] J. Corbin and M. Bidoit. A rehabilitation of Robinson's unification algorithm. In *IFIP Congress*, 1983.

[CJM98] E. Clarke, S. Jha, and W. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *Proc. IFIP PROCOMET*, 1998.

[DEK82] D. Dolev, S. Even, and R.M. Karp. On the security of ping-pong protocols. *Information and Computation*, (55):57–68, 1982.

[DLMS99] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proc. Formal methods and security protocols, FLOC Workshop, Trento*, 1999.

[DY83] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Trans. on Information Theory*, 29(2):198–208, 1983.

[Gou00] J. Goubault. A method for automatic cryptographic protocol verification. In *Proc. FMPPTA, Springer-Verlag*, 2000.

[HU79] J. Hopcroft and J. Ullman. *Introduction to automata theory, languages, and computation.* Addison-Wesley, 1979.

[Hui99] A. Huima. Efficient infinite-state analysis of security protocols. In *Proc. Formal methods and security protocols, FLOC Workshop, Trento*, 1999.

[JRV00] F. Jacquemard, M. Rusinowitch, and F. Vigneron. Compiling and verifying security protocols. In *Proc. Logic Programming and Automated Reasoning*, 2000. Also RR-INRIA 3938.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. TACAS, Springer LNCS*, 1996.

[Low99] G. Lowe. Towards a completeness result for model-checking of security protocols. Journal of Computer Security, 2-3(89–146), 1999.

[MMS97] J. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murϕ. In *Proc. IEEE Symposium on Security and Privacy*, 1997.

[Mon99] D. Monniaux. Abstracting cryptographic protocols with tree automata. In *Proc. Static Analysis Symposium, Springer LNCS*, 1999.

[Pau97] L. Paulson. Proving properties of security protocols by induction. In *Proc. IEEE Computer Security Foundations Workshop*, 1997.

[RT01] M. Rusinowitch and M. Turuani Protocol insecurity with finite number of sessions is NP-complete. RR INRIA 4134, March 2001.

[Van00] V. Vanackère. *Protocoles cryptographiques et calcul symbolique.* ENS-Lyon, July 2000. Mémoire de DEA.

[Wei99] C. Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In *Proc. CADE 99*. Springer LNCS 1632, 1999.

A Appendix

A.1 Proof of proposition 2.1

(1) We prove by induction on n that $A(T)_n, S(T)_n \subseteq \mathcal{M}_V$ if $T \subseteq \mathcal{M}_V$. The same method applies to $\mathcal{M}$.

$(2-3)$ It is immediate to check that S, A preserve set-theoretic containment and that $S(T) \supseteq T$ and $A(T) \supseteq T$. It follows that $S(S(T)) \supseteq S(T)$ and $A(A(T)) \supseteq A(T)$. Then we prove

$$S(S(T))_n \subseteq S(T) \text{ and } A(A(T))_n \subseteq A(T)$$

by induction on n.

(4) By (2-3), $A(S(T)) \supseteq S(T)$ and $A(S(T)) \supseteq A(T)$. Then it is enough to prove by induction on n that

$$A(S(T))_n \subseteq S(T) \cup A(T) \ .$$

We consider a typical case. Suppose $t \in A(S(T))_{n+1}$ because $E(t, C) \in A(S(T))_n$ and $Inv(C) \cap A(S(T))_n \neq \emptyset$. By inductive hypothesis, $E(t, C) \in S(T) \cup A(T)$ and $Inv(C) \cap (S(T) \cup A(T)) = Inv(C) \cap A(T) \neq \emptyset$. Two cases may arise:

$E(t, C) \in A(T)$. Then we conclude that $t \in A(T)$.

$E(t, C) \in S(T) \backslash A(T)$. Then it must be that $t, C \in S(T)$.

(5) For the leftmost equality we note:

$$A(S(A(T))) = S(A(T)) \cup A(A(T)) = S(A(T)) \cup A(T) = S(A(T)) \ .$$

For the rightmost equality, we have $S(A(T)) \subseteq S(A(S(T)))$ since $T \subseteq S(T)$. On the other hand:

$$S(A(S(T))) = S(S(T) \cup A(T)) \subseteq S(S(A(T))) = S(A(T)) \ . \ \diamond$$

A.2 Proof of proposition 3.1

Let $\rho : \{x_1, \ldots, x_n\} \to \{0, 1\}$ denote a boolean assignment and let $[\![\phi]\!]\rho$ denote the boolean evaluation of the formula ϕ in the assignment ρ. If $\phi \equiv \psi_1 \cdot \ldots \cdot \psi_m$ is in CNF then

$$[\![\phi]\!]\rho = 1 \text{ iff } \forall k \in \{1, \ldots, m\}\, \exists x_{i_k} \quad \begin{array}{l} \rho(x_{i_k}) = 1 \text{ and } (x_{i_k} \text{ literal in } \psi_k) \text{ or} \\ \rho(x_{i_k}) = 0 \text{ and } (\overline{x_{i_k}} \text{ literal in } \psi_k) \ . \end{array}$$

In the following, we say that a process in $(1), (2), (2'), (3)$ is *run* if the input message satisfies the filter condition and the continuation (output or assertion) is executed.

$(\Rightarrow)$ Suppose $[\![\phi]\!]\rho = 1$ we build a reduction to err. In (1) for $i = 1, \ldots, n$ we run the process $?x = E(y, C).!E(y, V_i).0$ with input $E(E(C, C_{\rho(x_i)}), C)$. Then the knowledge of the adversary becomes:

$$T_1 = T_0 \cup \{E(E(C, C_{\rho(x_i)}), V_i) \mid i = 1, \ldots n\} \ .$$

For $k = 1, \ldots, m$ we know that $[\![\psi_k]\!]\rho = 1$ and by the decomposition above we can find x_{i_k} such that either x_{i_k} is a literal in ψ_k or $\overline{x_{i_k}}$ is a literal in ψ_k. In the first case, we run the corresponding process $?x = E(E(y, C_1), V_{i_k}).!D_k.0$ in (2). In the second case, we run the corresponding process in $(2')$. At the end of this phase, the knowledge of the adversary is given by:

$$T_2 = T_1 \cup \{D_k \mid k = 1 \ldots, m\} \ .$$

Then the adversary builds a message $E(\cdots E(t, D_m), \cdots, D_1)$ for some t, which is provided as input to the process (3), finally reaching an invalid assertion.

$(\Leftarrow)$ Given a reduction to err of minimal length we build a satisfying assignment. The reduction must end with a run of the process (3) which is the only one that can lead to err. For this to happen, the adversary must be able to synthesise a message of the form $E(\cdots E(t, D_m), \cdots, D_1)$ for some t. Given T_0 and the shape of the messages output by the processes in $(1), (2), (2')$ it must be the case that for every $k \in \{1, \ldots, m\}$ and for some $i_k \in \{1, \ldots, n\}$ either the process $?x = E(E(y, C_1), V_{i_k}).!D_k.0$ in (2) or the process $?x = E(E(y, C_0), V_{i_k}).!D_k.0$ in $(2')$ is run.

Then we define:

$$\rho(x_i) = \begin{cases} 1 & \text{if } ?x = E(E(y, C_1), V_i).!D_k.0 \text{ is run for some } k \\ 0 & \text{if } ?x = E(E(y, C_0), V_i).!D_k.0 \text{ is run for some } k \\ \_ & \text{otherwise} \end{cases}$$

where _ can be chosen to be either 0 or 1.

We note that the assignment ρ is *well defined* because the runs of the processes $?x = E(E(y, C_1), V_i).!D_k.0$ and $?x = E(E(y, C_1), V_i).!D_{k'}.0$ are mutually exclusive. Indeed, at most one message of the shape $E(E(t, C_b), V_i)$, $b \in \{0, 1\}$, can be synthesised by the adversary following the run of the corresponding process in (1). We also remark that ρ *satisfies* ϕ since for every ψ_k at least one literal evaluates to 1. $\diamond$

A.3 Proof of proposition 3.2

There is no one to one correspondence between reductions on processes and reductions on the 2-counter machine. Actually, the processes can do much more reductions, many of them being redundant or useless. This slightly complicates the proof. We prove statement (1), since statement (2) is a direct consequence.

($\Rightarrow$) We show that $(q_0, 0, 0) \xrightarrow{*} (q, n, n')$ (a sequence of computations of the 2-counter machine) implies that there is a reduction $(P_M, T_0) \xrightarrow{*} (P, T)$ with $\underline{(q, n, n')} \in T$. The proof is by induction on the length of the derivation. The base case is obvious. Now let $(q_0, 0, 0) \xrightarrow{*} (q, n, n') \rightarrow (q, m, m')$ with r the last rule applied. By induction hypothesis there is a derivation $(P_M, T_0) \xrightarrow{*} (P, T)$ with $\underline{(q, n, n')} \in T$ (and $P = P_M \mid \ldots$), therefore the process P can perform the reduction $(P, T) \xrightarrow{*} (P', T \cup \{\underline{(q, m, m')}\})$ since $P_M = \ldots \mid P_r \mid \ldots$.

($\Leftarrow$) The proof is by induction on p taken as the length of the derivation $(P_M, T_0) \xrightarrow{*} (P, T)$. The base case is obvious. Let $(P_M, T_0) \xrightarrow{*} (P_p, T_p)$ with $\underline{(q, n, n')} \in T_p$. Without loss of generality, we can assume $\underline{(q, n, n')} \notin T_l$ for $l < p$. We remark that for all k, k', $\underline{(q, k, k')} \in S(A(T))$ implies that $\underline{(q, k, k')} \in T$. We assume that q is not a final state (the proof is similar for final states). Since $\underline{(q, n, n')} \in T_p$, there is some process P_r which we represent, in an informal notation, as it $?x = E(\langle s, s' \rangle, \underline{q'}).!E(\langle t, t' \rangle, \underline{q})$ such that $\underline{(q, n, n')}$ is an instance of $E(\langle t, t' \rangle, \underline{q})$. Let $\underline{(q, m, m')}$ be the corresponding instance of $E(\langle s, s' \rangle, \underline{q'})$. The derivation $(P_M, T_0) \xrightarrow{*} (P_p, T_p)$ must have the form:

$$
\begin{array}{ll}
(P_M, T_0) \xrightarrow{*} & (P, T_l \cup \{\underline{(q', m, m')}\}) \\
\xrightarrow{*} & (?x = E(\langle s, s' \rangle, \underline{q'}).!E(\langle t, t' \rangle, \underline{q}) \mid P_r \mid P, T_l \cup \{\underline{(q', m, m')} \cup \ldots) \\
\xrightarrow{*} & (!(\underline{q', n, n'}) \mid P_r \mid P, T_l \cup \{\underline{(q, m, m')}\} \cup \ldots) \\
\xrightarrow{*} & (P_p, T_l \cup \{\underline{(q, m, m')}\} \cup \{\underline{(q, n, n')}\} \cup \ldots) .
\end{array}
$$

By induction hypothesis, there is a derivation $(q, 0, 0) \xrightarrow{*} (q', m, m')$ of the 2-counter machine. Therefore, by definition of P_r there is a derivation:

$$(q, 0, 0) \xrightarrow{*} (q', m, m') \rightarrow (q, n, n') . \diamond$$

A.4 Proof of proposition 4.3

First we note that:

$$(A) \qquad K \subseteq A(T) \cap \mathcal{N} \;\Rightarrow\; P_K(T) \subseteq A(T) .$$

To prove (A) we show by induction on the structure of t, that if $t \in T$ then $acc_K(t) \subseteq A(T)$.

(1) To prove $K(T) \subseteq A(T) \cap \mathcal{N}$, we show by induction on n that:

$$K(T)_n \subseteq A(T) \cap \mathcal{N} .$$

We consider a typical case. Suppose $K(T)_n \subseteq A(T) \cap \mathcal{N}$. By definition, $K(T)_{n+1} = P_{K(T)_n}(T) \cap \mathcal{N}$. By (A), $P_{K(T)_n}(T) \subseteq A(T)$. Hence $K(T)_{n+1} \subseteq A(T) \cap \mathcal{N}$. We conclude observing:

$$K(T) = \bigcup_{n \in \omega} K(T)_n \subseteq A(T) \cap \mathcal{N} \ .$$

To prove $A(T) \cap \mathcal{N} \subseteq K(T)$, we show first by induction on n that:

$$(B) \quad A(T)_n \subseteq P_{K(T)}(T) \ .$$

We consider a typical case. Suppose $E(t, C) \in A(T)_n \subseteq P_{K(T)}(T)$ and $Inv(C) \cap A(T)_n \neq \emptyset$, so that $t \in A(T)_{n+1}$. Then $Inv(C) \cap P_{K(T)}(T) \neq \emptyset$ which implies $Inv(C) \cap K(T) \neq \emptyset$, since $P_{K(T)}(T) \cap \mathcal{N} = K(T)$ by definition of $K(T)$. Therefore $t \in P_{K(T)}(T)$. We conclude observing:

$$A(T) \cap \mathcal{N} = \bigcup_{n \in \omega} (A(T)_n \cap \mathcal{N}) \subseteq P_{K(T)}(T) \cap \mathcal{N} = K(T) \ .$$

(2) By (B) we have:

$$A(T) = \bigcup_{n \in \omega} A(T)_n \subseteq P_{K(T)}(T) \ .$$

To show the other inclusion, from $K(T) = A(T) \cap \mathcal{N}$ and (A) it follows:

$$P_{K(T)}(T) \subseteq A(T) \ .$$

(3) We have:

$$\begin{array}{lll} S(A(T)) & = S(P_{K(T)}(T)) & \text{(by (2))} \\ & = S_{K(T)}(P_{K(T)}(T)) & (\text{by } P_{K(T)}(T) \cap \mathcal{N} = K(T)) \end{array}$$

A.5 Proof of proposition 4.5

(1) By proposition 4.3(2), $A(T) = P_{K(T)}(T)$. Hence $S(A(T)) = S(P_{K(T)}(T))$.

(2) To prove $G \cap \mathcal{N} \subseteq K(T)$ we observe:

$$\begin{array}{lll} C \in G \cap \mathcal{N} & \Rightarrow C \in S(G) = S(A(T)) & \\ & \Rightarrow C \in A(T) & \text{by definition of } S, \\ & \Rightarrow C \in K(T) & \text{since } K(T) = A(T) \cap \mathcal{N} \text{ by 4.3(1)} \ . \end{array}$$

In the other direction, we note:

$$\begin{array}{lll} C \in K(T) & \Rightarrow C \in A(T) & \text{by 4.3(1)} \\ & \Rightarrow C \in S(A(T)) & \text{as } S(A(T)) \supseteq A(T) \\ & \Rightarrow C \in S(G) & \text{by hypothesis} \\ & \Rightarrow C \in G & \text{by definition of } S \ . \end{array}$$

(3) We observe that $Var(T) = Var(A(T))$ and $Var(S(T)) = Var(T)$.

(4) Obviously $S(G \cap G') \subseteq S(G) = S(A(T))$. To prove the other inclusion, we show that $t \in S(A(T))$ implies $t \in S(G \cap G')$ by induction on the structure of t.

$t \equiv C$. If $C \in S(A(T))$ then $C \in S(G)$ and $C \in S(G')$ which by definition of S implies $C \in G$ and $C \in G'$. Thus $C \in G \cap G' \subseteq S(G \cap G')$.

$t \equiv x$. The argument for constants applies also to variables.

$t \equiv \langle t_1, t_2 \rangle$. By proposition 2.1(4), we know that $t_1, t_2 \in S(A(T))$. By inductive hypothesis, $t_i \in S(G \cap G')$ for $i = 1, 2$, and by definition of S, $\langle t_1, t_2 \rangle \in S(G \cap G')$.

$t \equiv E(t_1, C)$. Four cases can arise.

$t_1 \in S(A(T)), C \in A(T)$. Then $t_1, C \in S(G \cap G')$ by inductive hypothesis and $E(t_1, C) \in S(G \cap G')$ by definition of S.

$t_1 \notin S(A(T)), C \in A(T)$. Then $t_1 \notin S(G) \cup S(G')$. Therefore $E(t_1, C) \in G \cap G' \subseteq S(G \cap G')$.

$t_1 \in S(A(T)), C \notin A(T)$. Then $C \notin G \cup G'$ and therefore $E(t_1, C) \in G \cap G' \subseteq S(G \cap G')$.

$t_1 \notin S(A(T)), C \notin A(T)$. Then $E(t_1, C) \in G \cap G' \subseteq S(G \cap G')$. $\diamond$

A.6 Proof of proposition 4.6

When we write $\{t\} \cup T$ on the left hand side of a rule it is intended that $t \notin T$. First we show by case analysis that if $T_1 \to T_2$ then (i) $S(A(T_1)) = S(A(T_2))$ and (ii) $T_1 \subseteq S(T_2)$. The properties presented in proposition 2.1 are useful here.

Termination. We define a function h measuring the *height* of a set of terms.

$$\begin{array}{ll} h(T) = \Sigma_{t \in T} h(t) & h(C) = h(x) = 1 \\ h(\langle t_1, t_2 \rangle) = 1 + h(t_1) + h(t_2) & h(E(t, C) = 1 + h(t) \end{array}$$

If T is a set of terms, we denote with $Sub(T)$ be the collection of its subterms. We show that every reduction sequence starting from a given set T_0 terminates. To this end, we define:

$$\begin{array}{ll} s(T) & = \sharp(Sub(T_0) \backslash S(T)) \\ \mu(T) & = (s(T), h(T)) \end{array}$$

We claim that if $T \to T'$ then $\mu(T) > \mu(T')$ with respect to the lexicographic order. We proceed by case analysis:

(a) $s(T \cup \{t\}) = s(T)$ because $t \in S(T)$ and $h(T \cup \{t\}) > h(T)$.

(b) $s(T \cup \{\langle t_1, t_2 \rangle\}) \geq s(T \cup \{t_1, t_2\})$ because $\langle t_1, t_2 \rangle \in S(T \cup \{t_1, t_2\})$ and $h(T \cup \langle t_1, t_2 \rangle) > h(T \cup \{t_1, t_2\})$.

(c) $s(T \cup \{E(t, C)\}) > s(T \cup \{t, E(t, C)\})$ because $t \in Sub(T_0) \backslash S(T \cup \{E(t, C)\})$.

Smallest generator. Suppose $T \xrightarrow{*} T'$ and T' is in normal form. We take the following steps.

$A(T') \subseteq S(T')$. We prove by induction on n that $A(T')_n \subseteq S(T')$.

• Obviously, $A(T')_0 = T' \subseteq S(T')$.

• Suppose $\langle t_1, t_2 \rangle \in A(T')_n \subseteq S(T')$. Then $\langle t_1, t_2 \rangle \notin T'$, for otherwise rule (b) would apply. Therefore $t_1, t_2 \in S(T')$.

• Suppose $E(t, C) \in A(T')_n \subseteq S(T')$ and $Inv(C) \cap A(T')_n \neq \emptyset$. Then $Inv(C) \cap S(T') \neq \emptyset$ implies $Inv(C) \cap T' \neq \emptyset$. Two cases may arise: either $C \in T'$ and $t \in S(T')$ or $E(t, C) \in T'$. In the first case we are done, in the second it must be that $t \in S(T')$ for otherwise rule (c) would apply.

T' is a generator.

$$\begin{array}{lll} S(A(T)) & = S(A(T')) & \text{by (i)} \\ & = S(A(S(T'))) & \text{by proposition 2.1(5)} \\ & = S(S(T') \cup A(T')) & \text{by proposition 2.1(4)} \\ & = S(S(T')) & \text{by } A(T') \subseteq S(T') \\ & = S(T') & \text{by proposition 2.1(2) .} \end{array}$$

T' is minimal. Suppose exists $G \subset T'$ such that $S(G) = S(T') = S(A(T))$. We choose $t \in T' \backslash G$. Then $T' = T'' \cup \{t\}$, $t \in S(T'') = S(G)$ and $T' \rightarrow T''$ by (a) which is a contradiction.

T' is minimum. By proposition 4.5(4) if there is another generator G for T then $T' \cap G$ is also a generator but since T' is minimal it must be that $T' \subseteq G$. $\diamond$

A.7 Proof of lemma 4.9

(1) First we observe that for any set of messages T and ground substitution σ:

$$(A) \qquad P_K(T) \cap \mathcal{N} \subseteq P_K(\sigma T) \cap \mathcal{N} \ .$$

Then we prove by induction on n that:

$$K(T)_n \subseteq K(\sigma T) \ .$$

It follows that

$$K(T_i) = \bigcup_{n \in \omega} K(T_i)_n \subseteq K(\sigma T_i) \ .$$

To show the other inclusion, we prove

$$(1') \qquad C \in acc_{K(T_i)}(\sigma x_j) \ \Rightarrow \ C \in K(T_i) \quad \text{if } j < i \ .$$

by induction on the pair (i, j) lexicographically ordered.

$(i = 1)$ The condition $j < i$ is not realized.

$(i > 1)$ Suppose $C \in acc_{K(T_i)}(\sigma x_j)$. Since σ is admissible we know that $\sigma x_j \in S(A(\sigma T_j))$.

We show that $K(\sigma T_j) = K(T_j)$. Since $K(T_j) \subseteq K(\sigma T_j)$, by definition of K, it is enough to prove:

$$P_{K(T_j)}(\sigma T_j) \cap \mathcal{N} = K(T_j) \ .$$

Suppose $C \in P_{K(T_j)}(\sigma T_j) \cap \mathcal{N}$ and $C \notin K(T_j)$ (otherwise we are done). Then

$$\exists\, k < j \;\; x_k \in P_{K(T_j)}(T_k) \text{ and } C \in acc_{K(T_j)}(\sigma x_k)$$

and by inductive hypothesis on (j,k) it follows $C \in K(T_j) \subseteq K(T_i)$.

Therefore, by proposition 4.3(3),

$$\sigma x_j \in S(A(\sigma T_j)) = S_{K(T_j)}(P_{K(T_j)}(\sigma T_j)) \ .$$

If $C \in K(T_j) \subseteq K(T_i)$ we are done. Otherwise:

$$\exists\, k < j \;\; x_k \in P_{K(T_i)}(T_j) \text{ and } C \in acc_{K(T_i)}(\sigma x_k) \ .$$

Then, by inductive hypothesis on (i,k), $C \in K(T_i)$.

(2) By (1) and proposition 4.3(3)

$$\begin{array}{lll} S(A(\sigma T_i)) & = S_{K(T_i)}(P_{K(T_i)}(\sigma T_i)) & \text{and} \\ \sigma S(A(T_i)) & = \sigma S_{K(T_i)}(P_{K(T_i)}(T_i)) & = S_{K(T_i)}(\sigma P_{K(T_i)}(T_i)) \ . \end{array}$$

We note that if $t \in \sigma P_{K(T_i)}(T_i)$ then $\exists\, t' \in P_{K(T_i)}(T_i) \;\; t = \sigma t'$. We conclude that $\sigma t' \in P_{K(T_i)}(\sigma T_i)$ and $\sigma S(A(T_i)) \subseteq S(A(\sigma T_i))$.

To show the other inclusion, we prove by induction on i that

$$P_{K(T_i)}(\sigma T_i) \subseteq S_{K(T_i)}(\sigma P_{K(T_i)}(T_i)) \ .$$

This holds obviously for $i = 1$ since T_1 contains no variables. So suppose $i > 1$ and $t \in P_{K(T_i)}(\sigma T_i)$. Two cases can arise:

$\exists\, t' \in P_{K(T_i)}(T_i) \;\; t = \sigma t'$. Then $t = \sigma t' \in \sigma P_{K(T_i)}(T_i)$.

$\exists\, x_j \in P_{K(T_i)}(T_i) \;\; t \in P_{K(T_i)}(\sigma x_j)$. Since σ is admissible:

$$\begin{array}{lll} \sigma(x_j) \in S(A(\sigma T_j)) & = S_{K(T_j)}(P_{K(T_j)}(\sigma T_j)) & \\ & = S_{K(T_j)}(\sigma(P_{K(T_j)}(T_j))) & \text{by inductive hypothesis} \\ & \subseteq S_{K(T_i)}(\sigma(P_{K(T_i)}(T_i))) & \text{since } T_j \subseteq T_i \ . \ \diamond \end{array}$$

(3) To prove $S(\sigma G_i) = \sigma S(G_i)$ we recall proposition 4.5(2) and proceed by induction on the structure of the term. To establish the equation $S(\sigma G_i) = S(\sigma(G_i \backslash V))$, it suffices to prove $\sigma(G_i \cap V) \subseteq S(\sigma(G_i \setminus V))$.

To this end, we define $V_j = \{x_1, \ldots, x_{j-1}\}$ and for $X \subseteq \mathcal{M}_V$ we set

$$v_j(X) = \{t \in X \mid Var(t) \subseteq V_j\} \ .$$

For $j \leq i$ we have $G_j \subseteq S(A(T_j)) \subseteq S(A(T_i)) = S(G_i)$, and thus $G_j = v_j(G_j) \subseteq v_j(S(G_i))$. We prove by induction on j, $j \leq i$, that:

$$\sigma V_j \subseteq S(\sigma(G_i \setminus V)) \ .$$

- The base case ($j = 1$) is obvious since $V_1 = \emptyset$.
- For the inductive step, assume the property holds for V_j. We show $\sigma x_j \subseteq S(\sigma(G_i \setminus V))$:

$$\begin{array}{lll} \sigma x_j & \in S(\sigma G_j) & \\ & \subseteq S(\sigma(v_j(S(G_i)))) & \\ & \subseteq S(\sigma(v_j(G_i))) & \text{since } S \text{ commutes with } \sigma \text{ and } v_j \\ & \subseteq S(\sigma((G_i \setminus V) \cup V_j)) & \\ & = S(\sigma(G_i \setminus V) \cup \sigma V_j) \ . & \end{array}$$

By inductive hypothesis $\sigma V_j \subseteq S(\sigma(G_i \setminus V))$, thus $S(\sigma(G_i \setminus V) \cup \sigma V_j) = S(\sigma(G_i \setminus V))$. Therefore we have established the property for V_{j+1}. We end the proof by noticing $\sigma(G_i \cap V) \subseteq \sigma(V_i)$. $\diamond$

A.8 Proof of proposition 6.1

Consider the following simplification rules on sets of inequalities:

$$\begin{array}{llll} (\mathsf{i}_1) & \{f(\vec{s}) \neq g(\vec{t})\} \cup I & \to I & \text{if } f \not\equiv g \\ (\mathsf{i}_2) & \{f(\vec{s}) \neq f(\vec{t})\} \cup I & \to \{s_i \neq t_i\} \cup I & \text{if } s_i \not\equiv t_i \\ (\mathsf{i}_3) & \{x \neq t\} \cup I & \to I & \text{if } x \in \mathit{Var}(t) \textit{ and } t \not\equiv x \end{array}$$

Since each rule decreases the number of symbols in I, it is clear that reduction always terminates. Moreover, if $I \to I'$ then (i) if $I \downarrow$ then $I' \downarrow$ and (ii) if $\sigma \models I'$ then $\sigma \models I$. Now given I such that $I \downarrow$ compute a I' in normal form such that $I \xrightarrow{*} I'$. We show that we can build σ such that $\sigma \models E, I'$ and therefore by (ii) $\sigma \models E, I$. If $I' = \emptyset$ then we are done. So suppose I' is not empty. Then it can be written as

$$\bigwedge_{i=1,\ldots,n} \bigwedge_{j=1,\ldots,m_i} x_i \neq t_{i,j} \text{ where } x_i \notin \mathit{Var}(t_{i,j}) \ .$$

We define the height $h(t)$ of a term t as $h(C) = h(x) = 1$ and $h(f(t_1, \ldots, t_n)) = 1 + \mathit{max}\{h(t_i) \mid i = 1, \ldots, n\}$. We note that if σ is a substitution then

$$h(\sigma t) \leq \mathit{max}(h(t), \mathit{max}\{h(t) - 1 + h(\sigma x) \mid x \in \mathit{Var}(t)\}) \ .$$

We define:

$$\begin{array}{ll} h_0 & = \mathit{min}\{h(t) \mid t \in T_i\} \\ H_m & = \mathit{max}\{h(t_{i,j}) \mid 1 \leq i \leq n, 1 \leq j \leq m_i\} \\ H & = \mathit{max}(H_m, h_0) \ . \end{array}$$

We claim that we build a substitution σ such that:

$$h(\sigma x_i) = (i+1)H \text{ and } \sigma x_i \in S(A(\sigma T_i))$$

To see this, choose $t_0 \in T_1$ such that $h(t_0) = h_0$ and then use pairing to build a term t of height $(i+1)H$. By definition of synthesis, this term is in $S(T_1) \subseteq S(A(\sigma T_i))$.

Thus $\sigma \models E$ and it remains to show that $\sigma \models I$. We distinguish two cases:

$x_i \neq t_{i,j},\ Var(t_{i,j}) \subseteq \{x_1, \ldots, x_{i-1}\}$. Then:

$$h(\sigma x_i) = (i+1)H > max_{j=1,\ldots,i-1}(H_m, H_m - 1 + (j+1)H) \geq h(\sigma t_{i,j}) \ .$$

$x_i \neq t_{i,j},\ \exists\, j > i\ \ x_j \in Var(t_{i,j})$. Then:

$$h(\sigma t_{i,j}) \geq h(\sigma x_j) = (j+1)H > (i+1)H = h(\sigma x_i) \ . \diamond$$

A.9 Proof of lemma 6.4

Let n be the size of E. All the variables we consider will have a rank bound by n. We proceed by induction on the order $\succ$ defined with respect to the maximal rank n.

(e_1) $([f(\vec{s}) = f(\vec{t})]Eq, E, \rho) \xrightarrow{\mathsf{e}_1} ([s_1 = t_1] \ldots [s_m = t_m]Eq, E, \rho)$.

By definition, we have $\mu([s = t]) > \mu([s_i = t_i])$ for $i = 1, \ldots, m$. Then either the reduction terminates or there exists a substitution σ_1 satisfying (i)(ii) such that $([s_1 = t_1] \ldots [s_m = t_m]Eq, E, \rho) \to \sigma_1([s_2 = t_2] \ldots [s_m = t_m]Eq, E, \rho)$.

Since σ_1 is decreasing $\mu([s = t]) > \mu(\sigma_1[s_i = t_i])$ for $i = 2, \ldots, m$. Therefore there is some decreasing substitution σ_2 such that $\sigma_1([s_2 = t_2] \ldots [s_m = t_m]Eq, E, \rho) \xrightarrow{*} (\sigma_2 \circ \sigma_1)([s_3 = t_3] \ldots [s_m = t_m]Eq, E, \rho)$. Iterating the process, we obtain a substitution $\sigma = \sigma_n \circ \cdots \circ \sigma_1$ and lemma 6.3(1) proves that σ is decreasing.

(e_{2-4}) $([x_i = t]Eq, E, \rho) \xrightarrow{e_{2-4}} \sigma(Eq, E, \rho)$ where σ is a decreasing substitution by construction. Moreover if $t \notin Var$ then $x_i \in Dom(\sigma)$.

(e_5) $([x_i = \langle t_1, t_2 \rangle]Eq, E, \rho) \xrightarrow{\mathsf{e}_5} ([x' = t_1][x'' = t_2]\rho' Eq, \rho' E, \rho' \circ \rho)$, where $\rho' = [\langle x', x'' \rangle / x_i]$.

By definition, we have (a) $\mu([x_i = \langle t_1, t_2 \rangle]) > \mu([x' = t_1])$.
(b) $\mu([x_i = \langle t_1, t_2 \rangle]) > \mu([x'' = t_2])$.

By (a), the induction hypothesis applies. Then either the reduction terminates or there exists a substitution σ_1 satisfying: (i) σ_1 is decreasing, (ii) $x' \in Dom(\sigma_1)$. By induction hypothesis, we have the following reduction:

$$([x_i = \langle t_1, t_2 \rangle]Eq, E, \rho) \xrightarrow{*} ([\sigma_1 x'' = \sigma_1 t_2]\sigma_1[\langle x', x'' \rangle / x_i]Eq, \sigma_1[\langle x', x'' \rangle / x_i]E, \sigma_1 \circ \rho) \ .$$

Since σ_1 is decreasing, by lemma 6.3(2) either $\mu([x'' = t_2]) > \mu([\sigma_1 x'' = \sigma_1 t_2])$ or $\sigma_1([x'' = t_2]) \equiv [x'' = t_2]$. By (b), the induction hypothesis applies. Then either the reduction terminates or there exists some substitution σ_2 satisfying (i) and (ii). We distinguish two cases depending on whether $\sigma_1 x'' = x''$ or not.

- $\sigma_1 x'' = x''$. The induction hypothesis yields: σ_2 is decreasing and $x'' \in Dom(\sigma_2)$.

  (i) Let $\sigma = \sigma_2 \circ \sigma_1 \circ [\langle x', x''\rangle / x_i]$. If $x' \in Dom(\sigma_1), x'' \in Dom(\sigma_2)$, $rk(x') = rk(x'') = rk(x_i)$, and σ_i decreasing for $i = 1, 2$, then $(\sigma_2 \circ \sigma_1)(x') \not\equiv x'$ and $(\sigma_2 \circ \sigma_1)(x'') \not\equiv x''$. Then $x' \in Dom(\sigma_2\sigma_1), x'' \in Dom(\sigma_2 \circ \sigma_1)$ and, by lemma 6.3(3) σ is decreasing.

  (ii) By definition, $x_i \in Dom(\sigma)$.

- $\sigma_1 x'' \not\equiv x''$. Let $\sigma = \sigma_2\sigma_1[\langle x', x''\rangle / x_i]$.

  (i) We have $x', x'' \in Dom(\sigma_1)$, and σ_1 decreasing implies that $rk(x') > rk(\sigma_1 x')$ and $rk(x'') > rk(\sigma_1 x'')$. Therefore $rk(x') > rk(\sigma_2\sigma_1 x')$ and $rk(x'') > rk(\sigma_2\sigma_1 x'')$ which proves that $x' \not\equiv \sigma_2\sigma_1 x'$ and $x'' \not\equiv \sigma_2\sigma_1 x''$. Then $x', x'' \in Dom(\sigma_2\sigma_1)$ and lemma 6.3(3) proves that σ is decreasing.

  (ii) By definition $x_i \in Dom(\sigma)$.

In each case, we get a substitution σ satisfying the requirements of the lemma.

(e_6) $([x_i = E(t, C)]Eq, E, \rho) \xrightarrow{\mathsf{e}_6} ([x = t]\rho' Eq, \rho' E, \rho' \circ \rho)$,
with x fresh, $\rho' = [E(x, C)/x_i]$, $rk(x) = rk(x_i)$.

By definition, $\mu([x_i = E(t, C)]) > \mu([x = t])$ since $|E(t, C)| > |t|$. The induction hypothesis applies: either the reduction terminates or there exists a substitution σ_1 satisfying (i) and (ii). Let $\sigma = \sigma_1 \circ [E(x, C)/x_i]$.

(i) We observe that $x \in Dom(\sigma_1)$, σ_1 is decreasing, and $rk(x) = rk(x_i)$. Then σ is decreasing by lemma 6.3(3).

(ii) By definition, $x_i \in Dom(\sigma)$.

Therefore $([x_i = E(t, C)]Eq, E, \rho) \xrightarrow{*} \sigma(Eq, E, \rho)$ and σ satisfies the requirements of the lemma.

(e_7) $([x_i = E(t, C)]Eq, E, \rho) \xrightarrow{\mathsf{e}_7} ([t' = t]$
$[E(t', C)/x_i](Eq, E, \rho)$
with $rk(x_i) > rk(t')$.

By definition $\mu([x_i = E(t, C)]) > \mu([t' = t])$ since $rk(x_i) > rk(t')$. The induction hypothesis applies and either the reduction terminates or there is some σ_1 satisfying (i), (ii).

(i) Let $\sigma = \sigma_1 \circ [E(t', C)/x_i]$. By lemma 6.3(1), σ is decreasing. (ii) We observe that $x_i \in Dom(\sigma)$.

Therefore $([x_i = E(t, C)]Eq, E, \rho) \xrightarrow{*} \sigma(Eq, E, \rho)$ and σ satisfies the requirement of the lemma. $\diamond$

A.10 Proof of proposition 6.12

We recall that we denote with $k, k', \ldots$ configurations and with $c, c', \ldots$ symbolic configurations. In this section the notation $\sigma[t/x]$ denotes the substitution σ' such that $\sigma'(y) = \sigma(y)$ if $y \neq x$ and $\sigma'(x) = t$. For the sake of simplicity, we will not show the processes in parallel composition of the leftmost thread when writing a symbolic configuration. In all the cases (1-7) we prove the following two properties:

(A) If $c \xrightarrow{\mathcal{R}^s} c'$ and $k' \in Im(c')$ then $\exists k \in Im(c)(k \xrightarrow{\mathcal{R}} k')$.

(B) If $k \in Im(c)$ and $k \xrightarrow{\mathcal{R}} k'$ then $\exists c'(c \xrightarrow{\mathcal{R}^s} c'$ and $k' \in Im(c'))$.

Property (A) implies

$$Im(\mathsf{Succ}_s^{\mathcal{R}^s}(c)) \subseteq \mathsf{Succ}^{\mathcal{R}}(Im(c)))$$

whereas property (B) implies

$$\mathsf{Succ}^{\mathcal{R}}(Im(c)) \subseteq Im(\mathsf{Succ}_s^{\mathcal{R}^s}(c)) \ .$$

Moreover, to handle easily instantiations of variables in key position, we define a subset of all symbolic configuration $\mathcal{S}^0$ as follows.

Definition A.1 *We denote by $\mathcal{S}$ the set of all symbolic configurations and by $\mathcal{S}^0$ the subset of $\mathcal{S}$ such that $c \in \mathcal{S}^0$ iff the process part of c is of the shape (we omit the case for $\_ \leftarrow \mathsf{prjr}(\_)$):*

$$\begin{array}{ll} & ?x.P \\ | & !t.P \ \textit{and}\ t \in \mathcal{M}_V \\ | & x \leftarrow \mathsf{prjl}(\langle t, t'\rangle).P \ \textit{and}\ \langle t, t'\rangle \in \mathcal{M}_V \\ | & x \leftarrow \mathsf{dec}(t, C).P \ \textit{and}\ t \in \mathcal{M}_V \\ | & [s = t]P_1, P_2 \ \textit{and}\ s, t \in \mathcal{M}_V \ . \end{array}$$

Then we will make use of the following lemma.

Lemma A.2 (1) *If (A) holds for any $c \in \mathcal{S}^0$ then (A) holds for any $c \in \mathcal{S}$.*
(2) *If (B) holds for any $c \in \mathcal{S}^0$ then (B) holds for any $c \in \mathcal{S}$.*

PROOF. In the following, τ denotes the substitution of variables in key positions that is used in symbolic rules $!^s$, pl_1^s, d_1^s, d_2^s, d_3^s, m_1^s, and m_2^s. We outline the proof in the cases ($\{!^s\}$ and $\{!\}$) (the other cases are similar).

(1) If $c \in \mathcal{S}$ and $c \xrightarrow{!^s} c'$ and $k' \in Im(c')$ with the substitution of variables in key position τ, then we must have $c \equiv (!t.P, T, E, I)$ and $\tau \downarrow (t, E)$. We notice that $\tau c \xrightarrow{!^s} c'$: as $\tau c \in \mathcal{S}^0$, we can conclude that $\exists k \in Im(\tau c)(k \xrightarrow{!} k')$. But then, $k \in Im(\tau c) \subseteq Im(c)$; hence (A) holds for c.

(2) If $k \xrightarrow{!} k'$ and $k \in Im(c)$, then it must be that $c \equiv (!t.P, T, E, I)$. We notice that there exists $\tau \downarrow (t, E)$ such that $k \in Im(\tau c)$ and thus, as $\tau c \in \mathcal{S}^0$, we can conclude that $\exists c'(\tau c \xrightarrow{!^s} c'$ and $k' \in Im(c'))$ But then, we just have to remark that $c \xrightarrow{!^s} c'$; hence (B) holds for c. $\diamond$

By lemma A.2, when proving (A) and (B) we just have to to consider the case where $c \in \mathcal{S}^0$. This will make the following proof much simpler. In the following we consider the three most interesting cases: input, decryption, and positive conditional.

Input

$(1-A)$ If $c \xrightarrow{?^s} c'$ and $k' \in Im(c')$ then it must be that:

$$\begin{cases} c & \equiv & (?x.P, T, E, I) \\ c' & \equiv & (P, T, (E; x : T), I) \\ k' & \equiv & (\sigma' P, \sigma' T),\ \sigma' \models (E; x : T), I \end{cases}$$

We define $t = \sigma'(x)$ and $\sigma = \sigma'[x/x]$ (we note that $[t/x] \circ \sigma = \sigma'$). With these definitions, it follows from $\sigma' \models (E; x : T), I$ (and the hypothesis $x \notin Var(T, E, I)$) that:

$$\begin{cases} \sigma \models E, I \\ t \in S(A(\sigma T)) \end{cases}$$

So if we define $k \equiv (?x.\sigma P, \sigma T)$, we have:

$$\begin{cases} k \in Im(c) \\ k \xrightarrow{?} ([t/x]\sigma P, \sigma T) = (\sigma' P, \sigma' T) = k' \end{cases}$$

$(1-B)$ If $k \in Im(c)$ and $k \xrightarrow{?} k'$ then it must be that:

$$\begin{cases} c & \equiv & (?x.P, T, E, I) \\ k & \equiv & (?x.\sigma P, \sigma T),\ \sigma \models E, I \\ k' & \equiv & ([t/x]\sigma P, \sigma T),\ t \in S(A(\sigma T)) \end{cases}$$

We define $\sigma' = [t/x] \circ \sigma$ and $c' = (P, T, (E; x : T), I)$. As $c \xrightarrow{?^s} c'$, it remains to show that $k' \in Im(c')$, which follows from $k' = (\sigma' P, \sigma T) = (\sigma' P, \sigma' T)$ ($\sigma' T = \sigma T$ as $x \notin Var(T)$) and $\sigma' \models (E; x : T), I$.

Decryption

$(3-A)$ This case will make use of the following lemma, whose proof is left to the reader.

Lemma A.3 *Suppose $x \notin Var(E, I)$, $x_i : T_i \in E$ and either (i) $\rho = [E(x, C)/x_i]$ and $C \in K(T_i)$ or (ii) $\rho = [E(t, C)/x_i]$, $E(t, C) \in G_\mu(T_i)$. Then:*

$$(\sigma \models \rho E, \rho I \Rightarrow \sigma \circ \rho \models E, I) \ .$$

- If $c \xrightarrow{\mathsf{d}_1^s} c'$ and $k' \in Im(c')$ then it must be that:

$$\begin{cases} c & \equiv & (x \leftarrow \mathsf{dec}(E(t, C), C').P, T, E, I) \text{ and } C' \in Inv(C) \\ c' & \equiv & ([t/x]P, T, E, I) \\ k' & \equiv & (\sigma[t/x]P, \sigma T),\ \sigma \models E, I \end{cases}$$

So if we define $k \equiv (x \leftarrow \mathsf{dec}(E(\sigma t, C), C').\sigma P, \sigma T)$, we have:

$$\begin{cases} k \in \mathit{Im}(c) \\ k \xrightarrow{\mathsf{d}} ([\sigma t/x]\sigma P, \sigma T) = k' \quad \text{as } [\sigma t/x] \circ \sigma = \sigma \circ [t/x] \end{cases}$$

- If $c \xrightarrow{\mathsf{d}_2^s} c'$ and $k' \in \mathit{Im}(c')$ then it must be that:

$$\begin{cases} c & \equiv & (x \leftarrow \mathsf{dec}(x_i, C').P, T, E, I), \;\; C \in K(T_i), \;\; C' \in \mathit{Inv}(C) \\ c' & \equiv & [E(x,C)/x_i](P, T, E, I) \\ k' & \equiv & (\sigma' \circ [E(x,C)/x_i])(P,T), \;\; \sigma' \models [E(x,C)/x_i]E, [E(x,C)/x_i]I \end{cases}$$

We define σ as follows:

$$\begin{cases} \sigma(x_j) = \sigma'(x_j) \;\; \text{if } x_j \notin \{x, x_i\} \\ \sigma(x_i) = E(\sigma' x, C) \\ \sigma(x) = x \end{cases}$$

We note that $[\sigma' x/x] \circ \sigma = \sigma' \circ [E(x,C)/x_i]$. With these definitions, it follows from $\sigma' \models [E(x,C)/x_i]E, [E(x,C)/x_i]I$ and $C \in K(T_i)$ that $\sigma \models E, I$ (justified by lemma A.3(1)).

So if we define $k \equiv (x \leftarrow \mathsf{dec}(\sigma x_i, C').\sigma P, \sigma T) \equiv (x \leftarrow \mathsf{dec}(E(\sigma' x, C), C').\sigma P, \sigma T)$, we have:

$$\begin{cases} k \in \mathit{Im}(c) & & \\ k \xrightarrow{\mathsf{d}} ([\sigma' x/x]\sigma P, \sigma T) & = & ([\sigma' x/x] \circ \sigma)(P,T) \;\; \text{by } x \notin \mathit{Var}(T) \\ & = & k' \;\; \text{by } [\sigma' x/x] \circ \sigma = \sigma' \circ [E(x,C)/x_i] \end{cases}$$

- If $c \xrightarrow{\mathsf{d}_3^s} c'$ and $k' \in \mathit{Im}(c')$ then it must be that:

$$\begin{cases} c & \equiv & (x \leftarrow \mathsf{dec}(x_i, C').P, T, E, I), \;\; C' \in \mathit{Inv}(C) \\ c' & \equiv & [E(t,C)/x_i]([t/x]P, T, E, I), \;\; E(t,C) \in G_\mu(T_i) \\ k' & \equiv & (\sigma' \circ [E(x,C)/x_i])([t/x]P, T), \;\; \sigma' \models [E(t,C)/x_i]E, [E(t,C)/x_i]I \; . \end{cases}$$

We define σ as follows:

$$\begin{cases} \sigma(x_j) = \sigma'(x_j) \;\; \text{if } x_j \notin \{x, x_i\} \\ \sigma(x_i) = E(\sigma' t, C) \\ \sigma(x) = x \; . \end{cases}$$

We note that $[\sigma' t/x] \circ \sigma = \sigma' \circ [E(t,C)/x_i]$. With these definitions, it follows from $\sigma' \models [E(t,C)/x_i]E, [E(t,C)/x_i]I$ and $E(t,C) \in G_\mu(T_i)$ that $\sigma \models E, I$ (justified by lemma A.3(2)).

So if we define $k \equiv (x \leftarrow \mathsf{dec}(\sigma x_i, C').\sigma P, \sigma T) \equiv (x \leftarrow \mathsf{dec}(E(\sigma' t, C), C').\sigma P, \sigma T)$, we have:

$$\begin{cases} k \in \mathit{Im}(c) & & \\ k \xrightarrow{\mathsf{d}} ([\sigma' t/x]\sigma P, \sigma T) & = & ([\sigma' t/x] \circ \sigma)(P,T) \;\; \text{by } x \notin \mathit{Var}(T) \\ & = & k' \;\; \text{by } [\sigma' t/x] \circ \sigma = \sigma' \circ [E(t,C)/x_i] \; . \end{cases}$$

$(3-B)$ If $k \in Im(c)$ and $k \xrightarrow{\mathsf{d}} k'$ then there are only two cases (recall that we only consider configurations c in $\mathcal{S}^0$):

$$
\begin{array}{ll}
(i) & \left\{ \begin{array}{lcl} c & \equiv & (x \leftarrow \mathsf{dec}(E(t,C'),C).P,T,E,I),\ C' \in Inv(C) \\ k & \equiv & (x \leftarrow \mathsf{dec}(E(\sigma t,C'),C).\sigma P, \sigma T),\ \sigma \models E,I \\ k' & \equiv & ([\sigma t/x]\sigma P, \sigma T)\ . \end{array} \right. \\
(ii) & \left\{ \begin{array}{lcl} c & \equiv & (x \leftarrow \mathsf{dec}(x_i,C').P,T,E,I),\ C' \in Inv(C) \\ k & \equiv & (x \leftarrow \mathsf{dec}(\sigma x_i,C').\sigma P, \sigma T),\ \sigma \models E,I \\ k' & \equiv & ([t'/x]\sigma P, \sigma T), \sigma x_i = E(t',C)\ . \end{array} \right.
\end{array}
$$

In the first case, it is easy to show that $c \xrightarrow{\mathsf{d}_1^s} c'$ with $c' \equiv ([t/x]P,T,E,I)$ and we have $k' \in Im(c')$. In the second case, as $E(t',C) = \sigma x_i \in S(A(\sigma T_i))$, the decomposition theorem 4.10 states that there are only two possibilities.

- If $t' \in S(A(\sigma T_i))$ and $C \in K(T_i)$ then rule d_2^s will apply and $c \xrightarrow{\mathsf{d}_2^s} c'$ where $c' \equiv [E(x,C)/x_i](P,T,E,I)$. We define $\sigma' = \sigma[x_i/x_i, t'/x]$ and note that $\sigma' \circ [E(x,C)/x_i] = [t'/x] \circ \sigma$. Then it is easy to show:

$$\sigma' \models [E(x,C)/x_i]E, [E(x,C)/x_i]I, \text{ and}$$

$$
\begin{array}{ll}
\sigma'([E(x,C)/x_i](P,T)) & = [t'/x] \circ \sigma(P,T) \\
 & = ([t'/x]\sigma P, \sigma T) \text{ by } x \notin Var(T) \\
 & = k'
\end{array}
$$

i.e. $k' \in Im(c')$.

- Else we must have $E(t',C) = \sigma(E(t,C))$ and $E(t,C) \in G_\mu(T_i)$, which implies $c \xrightarrow{\mathsf{d}_3^s} c'$ where $c' \equiv [E(t,C)/x_i]([t/x]P,T,E,I)$. We define $\sigma' = \sigma[x_i/x_i]$ and note that $\sigma' \circ [E(t,C)/x_i] = \sigma$. Then it is easy to show:

$$\sigma' \models [E(t,C)/x_i]E, [E(t,C)/x_i]I, \text{ and}$$

$$
\begin{array}{ll}
\sigma'([E(t,C)/x_i]([t/x]P,T)) & = \sigma([t/x]P,T) \\
 & = ([\sigma t/x]\sigma P, \sigma T) \\
 & = k'
\end{array}
$$

i.e. $k' \in Im(c')$.

Positive conditional

$(5-A)$ If $c \xrightarrow{\mathsf{m}_1^s} c'$ and $k' \in Im(c')$ then it must be that:

$$
\left\{ \begin{array}{lcl} c & \equiv & ([s=t]P_1, P_2, T, E, I) \\ c' & \equiv & \rho(P_1,T,E,I),\ ([s=t],E,id) \xrightarrow{*} (\emptyset, E', \rho) \\ k' & \equiv & (\sigma \circ \rho)(P_1,T),\ \sigma \models E', \rho I\ . \end{array} \right.
$$

We define $k \equiv (\sigma \circ \rho)([s = t]P_1, P_2, T)$. By corollary 6.7(1) we know that $(\sigma \circ \rho) \models s = t, E$, and since $\sigma \models \rho I$ we have, $(\sigma \circ \rho) \models I$. Therefore:

$$\begin{cases} k \in Im(c) \\ k \xrightarrow{\mathsf{m}_1} k' \ . \end{cases}$$

$(5 - B)$ If $k \in Im(c)$ and $k \xrightarrow{\mathsf{m}_2} k'$ then it must be that:

$$\begin{cases} c & \equiv & ([s = t]P_1, P_2, T, E, I) \\ k & \equiv & ([\sigma s = \sigma t]\sigma P_1, \sigma P_2, \sigma T), \ \sigma \models E, I \text{ and } \sigma s = \sigma t \\ k' & \equiv & (\sigma P_1, \sigma T) \ . \end{cases}$$

By corollary 6.7(2) we know that $([s = t], E, id) \xrightarrow{*} (\emptyset, E', \rho')$ and there is a σ' such that $\sigma' \models E'$ and $\sigma = \sigma' \circ \rho'$. We define $c' \equiv \rho'(P_1, T, E, I)$, and we have $c \xrightarrow{\mathsf{m}_1^s} c'$ and $k' = \sigma'(\rho' P_1, \rho' T) \in Im(c')$ (the property $\sigma' \models \rho' I$ follows directly from $\sigma' \circ \rho' \models I$). $\diamond$

A.11 Proof of proposition 7.2

The proof relies on the following lemma.

Lemma A.4 *Let $E \equiv x_1 : T_1; \ldots; x_n : T_n$ be an environment, $T \supseteq T_n$, G be a generator for T, $V = \{x_1, \ldots, x_{n-1}\}$, and $t \in \mathcal{M}_V$ such that $Var(t) \subseteq V$. Then:*

$$t \notin S(G \cup V) \Rightarrow \exists I \ I \downarrow \ \textit{and} \ \forall \sigma \models E, I \ \sigma t \notin S(\sigma G)$$

Proof. By structural induction on t:

- If $t \equiv C$ and $C \notin S(G \cup V)$ then $C \notin K(T)$ which implies $C \notin K(\sigma T) = S(\sigma G) \cap \mathcal{N}$.
- If $t \equiv x$, then we have $t \in S(G \cup V)$ and thus the implication holds trivially.
- If $t \equiv \langle t_1, t_2 \rangle$, $t \notin S(G \cup V)$ then $\exists i \in \{1, 2\} \mid t_i \notin S(G \cup V)$. We apply the induction hypothesis to t_i to deduce that $\forall \sigma \models E, I \ \sigma t_i \notin S(\sigma G)$ and notice that $\sigma t_i \notin S(\sigma G)$ implies $\sigma t = \langle \sigma t_1, \sigma t_2 \rangle \notin S(\sigma G)$ (remember that $S(\sigma G) = S(A(\sigma T))$).
- If $t \equiv E(t', C)$, we consider the (possibly empty) set:

$$X = \{t_i \mid E(t_i, C) \in G\} \ .$$

We suppose $t \notin S(G \cup V)$. In particular $\forall t_i \in X \ t_i \neq t'$, so we can define a set of inequalities $I_X = \{t_i \neq t' \mid t_i \in X\}$. Note that $I_X \downarrow$. There are two cases:

(1) $t' \in S(G \cup V)$. Then we must have $C \notin S(G \cup V)$ and thus $C \notin G$. In that case, if $\sigma \models E, I_X$ then $\sigma t = E(\sigma t', C) \in S(\sigma G)$ would imply (by theorem 4.10) $\sigma t' = \sigma t_i$ with $t_i \in X$, which is impossible because $\sigma \models I_X$.

(2) $t' \notin S(G \cup V)$. The induction hypothesis gives us a finite and consistent set of inequalities I. We denote $I' = I \cup I_X$. Now if we take $\sigma \models E, I'$, the condition $\sigma t \in S(\sigma G)$ would

imply (by theorem 4.10) that either $C \in K(T_i)$ and $\sigma t' \in S(\sigma G)$, or $\sigma t' = \sigma t_i$ with $t_i \in X$. The former cannot hold because $\sigma \models I$ and the latter because $\sigma \models I_X$. $\diamond$

From this we can deduce the following lemma.

Lemma A.5 *Let (P, T, E, I) be a symbolic configuration, G be a generator for T, $V = Var(E)$, and $\{t_1, \ldots, t_n\} \subseteq \mathcal{M}_V$. Then:*

$$(\forall \sigma \models E, I\ \sigma t_1 \in S(A(\sigma T)) \vee \cdots \vee \sigma t_n \in S(A(\sigma T))) \quad \textit{iff} \quad \exists i t_i \in S(G \cup V)\ .$$

PROOF. ($\Leftarrow$) Immediate. ($\Rightarrow$) We show:

$$(\forall i \in \{1, \ldots, n\}\ t_i \notin S(G \cup V)) \Rightarrow \exists \sigma \models E, I \forall i \in \{1, \ldots, n\}\ \sigma t_i \notin S(\sigma G)\ .$$

For each t_i it follows from lemma A.4 that:

$$t_i \notin S(G \cup V) \Rightarrow \exists I_i \downarrow \forall \sigma \models E, I_i\ \sigma t_i \notin S(\sigma G)\ .$$

If we define $I' = I \cup \left(\bigcup_{i=1,\ldots,n} I_i\right)$, proposition 6.1 gives us a σ such that $\sigma \models E, I$ and $\forall i\ \sigma t_i \notin S(\sigma G)$. $\diamond$

Now proposition 7.2 follows directly from lemma A.5, using the fact that rule (sc^s) will always test the condition $I \downarrow$.

A.12 Example: a variant of the Otway-Rees protocol

A flawed variant of the Otway-Rees protocol [BAN89] as taken by our tool is presented in figure 7. In this particular example, the assertion we want to check is that if principal A uses the key *kab* to encrypt a message after his run of the protocol, the content of the message will remain unknown to an intruder. The syntax used is close to that of our formal model, with some additional syntaxic sugar:

- All variable names begin with a lowercase character whereas constant names begin with an uppercase character.
- Projection is written as `<x1,x2> <- t`. Moreover, we allow the syntax

  ```
  <na2,kab> <- decrypt(e,Ka)
  ```

  to decrypt e with K_a and project the result in one step. The same convention holds in the case of `read <m,e>`, which alleviates the need to introduce temporary variables.
- The *map* construction `k <- map(A->Ka,B->Kb,C->Kc) x` is equivalent to:

  ```
  if x=A then k <- Ka
  else if x=B then k <- Kb
  else if x=C then k <- Kc
  ```

```
Principals:

A: fresh na
   write <na,A,B,E(<na,A,B>,Ka)>
   read <m,e> ; [m=na] ; <na2,kab> <- decrypt(e,Ka) ; [na2=na]
   write E(HIDDEN,kab)
   assert(secret(HIDDEN))
   nil

B: read <na,a,b,e> ; [b=B]
   fresh nb
   write <na,a,b,e,nb,E(<na,a,b>,Kb)>
   read <na2,e1,e2> ; [na2=na] ; <nb2,kab> <- decrypt(e2,Kb) ; [nb2=nb]
   write <na,e1>
   nil

S: read <na,a,b,e1,nb,e2>
   k1<-map(A->Ka,B->Kb,C->Kc) a
   k2<-map(A->Ka,B->Kb,C->Kc) b
   <na1,a1,b1> <- decrypt(e1,k1) ; [<na1,a1,b1>=<na,a,b>]
   <na2,a2,b2> <- decrypt(e2,k2) ; [<na2,a2,b2>=<na,a,b>]
   fresh kab
   write <na,E(<na,kab>,k1),E(<nb,kab>,k2)>
   nil

Environment:

A; B; C; Kc
```

Figure 7: Specification of a flawed Otway-Rees protocol

```
Principals:
Ai :    assert(secret(HIDDEN)) ;
        nil
S  :    nil
Ar :    read <na2,e1,e2> ;
        [na2=na] ;
        <nb2,kab> <- decrypt(e2,Ka) ;
        [nb2=N16] ;
        write <na,e1> ;
        nil

Current environment:
                HIDDEN N24 N16 N1 Kc C B A
                Crypt(<N1,N24>,Ka) Crypt(<na,C,A>,Ka) Crypt(<N1,A,B>,Ka)

Variables:
           na : rk(0) N1 Kc C B A
                Crypt(<N1,A,B>,Ka)

Protocol history:

    Ai sends <N1,A,B,Crypt(<N1,A,B>,Ka)>

    Ar gets <na,C,A,e>
    Ar sends <na,C,A,e,N16,Crypt(<na,C,A>,Ka)>

    S gets <na,C,A,Crypt(<na,C,A>,Kc),N1,Crypt(<na,C,A>,Ka)>
    S sends <na,Crypt(<na,N24>,Kc),Crypt(<N1,N24>,Ka)>

    Ai gets <N1,Crypt(<N1,N24>,Ka)>
    Ai sends Crypt(HIDDEN,N24)
```

Figure 8: An error as reported by the Symbolic Protocol Analyser

Figure 8 exhibits an error as reported by our analyser when running several instances of roles A, B and S in parallel. In the case considered A_i and A_r denote A playing the role of initiator and responder, respectively. Nonces generated during the protocol run are denoted by $N1, N2 \ldots$

A.13 Unification procedure

The unification procedure `U` relies on the the procedure `Occ(v,t)` checking whether the variable `v` occurs in the term `t`. It also relies on a global variable `time` of type `nat` which is initialised to `0`. When `Occ` is called, it increases the counter `time` and then calls the auxiliary procedure `Occ_aux`. Whenever a node is visited the procedure `Occ_aux` sets the `stamp` field to the current `time`. This avoids performing the occur check several times on the same node and entails a complexity $O(e)$ for the occur check.

```
function U(s1, s2: term): bool
 var t1, t2: term;
```

```
 let t1=Find(s1), t2=Find(s2) in
 case
 t1=t2              : true
 t1.isvar,t2.isvar : Union(t1,t2); true
 t1.isvar           : if Occ(t1,t2) then false else Union(t1,t2); true
 t2.isvar           : if Occ(t2,t1) then false else Union(t2,t1); true
 else               : if t1.fn \= t2.fn then false
                         else Union(t1,t2); U_list(t1.arg, t2.arg)

function U_list(t1, t2: term_list): bool
 case
 t1=t2=nil          : true
 --                 : if U(t1.tr,t2.tr) then U_list(t1.next,t2.next) else false

procedure Union(t1, t2: term)
 var T: term_list_tag;
 t1.is := t2
 T:=Min(t1.cstr,t2.cstr)
 if T\=t2.cstr then C(t2,T)

function Occ(v,t: term): bool
 time:=time+1;
 Occ_aux(v,t)

function Occ_aux(v,t: term): bool
 case
 t.isvar          : (v=t)
 t.stamp=time     : false
 else             : t.stamp:=time; Occ_list(v,t.arg)

function Occ_list(v:term,s:term_list): bool
 case
 s.tr=nil         : false
 Occ(Find(s.tr)) : true
 else : Occ_list(v,s.next)
```

A.14 Proof of proposition 9.1

We must prove that from a derivation we can construct another derivation not including pairs. Two technical results must be set first.

Proposition A.6 *Let $T_0 \subseteq K^+$, and let $(P_0, T_0) \xrightarrow{*} (P_n, T_n)$ be some reduction. Let $t \in T_n \setminus T_{n-1}$, then for all $s = \langle s_1, s_2 \rangle$ subterm of t, we have $s_1, s_2 \in S(A(T_p))$ for some $p < n$.*

PROOF. *Base case.* The property holds for $T_0 \subseteq K^+$.

Induction step. Let $T_{n+1} = T_n \cup \{\beta y\}$ with $\alpha y \in S(A(T_n))$, therefore $t = \beta y$. Let $s = \langle s_1, s_2 \rangle$ be a subterm of βy hence of y. Therefore s is a subterm of αy and two cases may occur:

(1) s is a subterm of some term in $G_\mu(T_n)$ and by induction hypothesis $s \in S(A(T_p))$ for $p < n$ (by construction a term in $G_\mu(T_n)$ is a subterm of some element of T_n).

(2) $s \in S(A(T_n))$ therefore $s_1, s_2 \in S(A(T_n))$ and the property holds since $n < n+1$. $\diamond$

The next step is to show that we can restrict ourselves to derivations where emitted messages do not contain pairing (assuming that the initial knowledge T_0 does not contain pairings). First, we show that pairs do not matter for computing minimal generators. We remark that we can safely add the rule $T \cup \{\langle t_1, t_2 \rangle\} \to T$ if $\langle t_1, t_2 \rangle \in S(A(T))$ to the set of rules used in the computation of a minimal generator.

Proposition A.7 *Let $T \supseteq T_0$ be a finite set of terms such that if $\langle t_1, t_2 \rangle$ is a subterm of $t \in T$ then $\langle t_1, t_2 \rangle \in S(A(T \setminus \{t\}))$ (elimination property). Then $\overline{G_\mu(T)} = G_\mu(\overline{T})$.*

PROOF. From a sequence $M_0 = T, \ldots, M_p = G_\mu(T)$ we derive a sequence $N_0 = \overline{T}, \ldots, N_p = G_\mu(\overline{T})$ such that $N_i = \overline{M_i}$ for all i.

Base case. $n = 0$ the property holds by construction.

Inductive case.

(1) $M_n = M \cup \{\langle t_1, t_2 \rangle\} \to M_{n+1} = M$ since $\langle t_1, t_2 \rangle \in S(A(M_n))$
$N_n = \overline{M} \cup \{\overline{\langle t_1, t_2 \rangle}\} = \overline{M} \cup \{t_0\} \to N_{n+1} = \overline{M} = \overline{M_{n+1}}$ since $N_0 = \overline{T} \supseteq T_0$

(2) $M_n = M \cup \{Ct\} \to M \cup \{t\}$ if $C \in M$ (hence $C \in \overline{M}$)
$N_n = \overline{M} \cup \{\overline{Ct}\} \to N_{n+1} = \overline{M} \cup \{\overline{t}\}$ since $\overline{C} = C \in \overline{M}$

(3) $M_n = M \cup \{C\} \to M_{n+1} = M$ if $C \in M$
$N_n = \overline{M} \cup \{\overline{C}\} \to N_{n+1} = M$ since $\overline{C} = C \in \overline{M}$

Moreover if no rule applies to M_n then no rule applies to $\overline{M_n}$. This proves the claim $\overline{G_\mu(T)} = G_\mu(\overline{T})$. $\diamond$

To get the proof of proposition 9.1, we remark that T_p satisfies the elimination property by proposition A.6 (the result remains true for public keys). This implies that the derivation $(P_0, \overline{T_0}) \xrightarrow{*} (P_n, \overline{T_n})$ is correct. By construction, it reaches err iff the initial derivation reaches err.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - B.P. 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 Villers lès Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot St Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, B.P. 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399