Basic Research in Computer Science

Monotonic Set-Extended Prefix Rewriting and Verification of Recursive Ping-Pong Protocols

Giorgio Delzanno
Javier Esparza
Jiří Srba

BRICS Report Series **RS-06-14**

ISSN 0909-0878 **July 2006**

**See back inner page for a list of recent BRICS Report Series publications.
Copies may be obtained by contacting:**

**BRICS
Department of Computer Science
University of Aarhus
IT-parken, Aabogade 34
DK–8200 Aarhus N
Denmark
Telephone: +45 8942 9300
Telefax: +45 8942 5601
Internet: BRICS@brics.dk**

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `RS/06/14/`

Monotonic Set-Extended Prefix Rewriting and Verification of Recursive Ping-Pong Protocols

Giorgio Delzanno[1], Javier Esparza[2⋆] and Jiří Srba[3⋆⋆]

[1] Dipartimento di Informatica e Scienze dell'Informazione
Università di Genova, Italy
[2] Institut für Formale Methoden der Informatik
Universität Stuttgart, Germany
[3] **BRICS**⋆⋆⋆, Department of Computer Science
Aalborg University, Denmark

Abstract. Ping-pong protocols with recursive definitions of agents, but without any active intruder, are a Turing powerful model. We show that under the environment sensitive semantics (i.e. by adding an active intruder capable of storing all exchanged messages including full analysis and synthesis of messages) some verification problems become decidable. In particular we give an algorithm to decide control state reachability, a problem related to security properties like secrecy and authenticity. The proof is via a reduction to a new prefix rewriting model called Monotonic Set-extended Prefix rewriting (MSP). We demonstrate further applicability of the introduced model by encoding a fragment of the ccp (concurrent constraint programming) language into MSP.

1 Introduction

Motivation and related work. In recent years there has been an increasing interest in formal analysis of cryptographic protocols. Even under the *perfect encryption hypothesis* (an intruder cannot exploit weaknesses of the encryption algorithm itself) a number of protocols presented in the literature were flawed, which escalated the need for automatic verification of protocol properties like secrecy and authenticity. Unfortunately, the general problem for fully featured languages like the spi-calculus [1] is undecidable and hence finding a decidable yet reasonably expressive subset of such Turing-powerful formalisms is desirable. We contribute to this area by investigating the decidability borderline for protocols with a restricted set of cryptographic primitives while still preserving complex control-flow structures and with no restriction on the length of messages.

Recently, in [5, 13, 14] this kind of study has been carried out for models of cryptographic protocols with the basic *ping-pong behaviour* as introduced

⋆ Partially supported by the DFG project "Algorithms for Software Model Checking".
⋆⋆ Partially supported by the research center ITI, project No. 1M0021620808, and by the grant MSM 0021622419 of Ministry of Education, Czech Republic.
⋆⋆⋆ **B**asic **R**esearch **I**n **C**omputer **S**cience, Danish National Research Foundation.

by Dolev and Yao [11]. In a ping-pong protocol a message is a single piece of data (plain text) possibly encrypted with a finite sequence of keys. Agents are memory-less. The ping-pong communication mechanism can be naturally modelled using prefix rewriting over finite words. The connection is based on the idea of representing a piece of data d encrypted, e.g., with k_1, k_2 and then k_3, as the word $k_3k_2k_1d$. On reception of a message, an agent can only apply a finite sequence of keys to decrypt the message, and then use another sequence of keys applied to the decrypted message to forge the reply. For example the prefix rewrite rule $k_3k_2 \to k_4$ transforms $k_3k_2k_1d$ into k_4k_1d (the suffix k_1d of the first word is copied into the reply).

In [10] Dolev, Even and Karp showed that secrecy properties are decidable in polynomial time for finite ping-pong protocols under an environment sensitive semantics (active attacker) used to model possibly malicious agents. (Where finite means that the length of all computations is syntactically bounded.) In the context of cryptographic protocols, the aim of the attacker is to augment his/her initial knowledge by listening on the communication channels, e.g., to learn some of the secrets exchanged by the honest agents. A general way of defining active attackers was introduced by Dolev and Yao in [11], now commonly known as the Dolev-Yao intruder model. In this model, the communication among the agents is asynchronous. The attacker can store and analyze all messages exchanged among the agents using the current set of *compromised keys*. The attacker can also synthesize new messages starting from the stored messages and compromised keys.

In [5] Amadio, Lugiez and Vanackère extended the result of [10] by showing that secrecy is decidable in polynomial time for ping-pong protocols with replication. The replication operator $!P$ is peculiar of process algebraic languages. The agent $!P$ can generate an arbitrary number of identical copies of P operating in parallel. This work was later extended to protocols with a limited use of pairing [4, 9].

A more powerful way of extending the class of finite ping-pong protocols is to allow for recursive process definitions, as in CCS. Loosely speaking, recursion allows to define processes with arbitrary flow-graphs; the finite case [11, 10] corresponds to acyclic graphs. Recursive definitions are more powerful than replicative ones, in particular recursive protocols are not memory-less any more as every agent can be seen as an automaton with finite memory. This enables to verify not only secrecy but also authenticity (see e.g. a protocol by Woo and Lam in Appendix A). The combination of ping-pong behaviour with recursive definitions and finite memory enables us to encode several protocols studied in the literature, including features like a limited notion of pairing, public key encryption and others (see Appendix).

A process algebra for recursive ping-pong protocols was introduced in [13, 14] where it was proved that the resulting model (without any notion of an attacker) is Turing powerful.

Novel contribution. The results from [13, 14] were obtained for protocols *in the absence of an attacker*. In this paper, we show that, maybe surprisingly, the

control state reachability problem for recursive ping-pong protocols in the presence of a Dolev-Yao intruder is decidable (in particular, this new model is no longer Turing powerful). Since secrecy/authenticity properties can be reduced to the control state reachability problem by adding new control points that can be reached if and only if secrecy/authenticity is violated, this also implies the decidability of these properties. A few examples demonstrating the modelling power of recursive ping-pong protocols are described in detail in Appendix.

Our main decidability result is consistent with the results on tail-recursive cryptographic protocols from [4]. Indeed the necessary (but not sufficient) conditions defined in [4] (locality, linearity and independency) for decidability of control state reachability are all satisfied by recursive ping-pong protocols.

Methodology: reduction to a new computational model. In order to achieve this result, we first introduce a new model called *Monotonic Set-extended Prefix rewriting system (MSP)*. Configurations in MSPs have the form (p, T) where p is a control state and T is a *set* of words (the current store or pool). MSP rules enrich prefix rewrite rules with the update of the control state. Control states are partially ordered, and a state update can only lead to states that are greater or equal than the current one, like for instance in weak Büchi automata [19, 15], or weak Process Rewrite Systems (wPRSs) [17].

Furthermore, when a rule is applied to a word w in the current store T with the result w', both w and w' are included in the new store. Thus, the store can only grow monotonically. In our application to ping-pong protocols, T represents the current knowledge of the attacker (modulo analysis and synthesis). More generally, it can be viewed as a monotonic store used for agent communication in languages like ccp [22].

Technical contribution. As a main technical contribution, we will show that known results on prefix rewrite systems, namely the efficient representation of predecessor sets of words in prefix rewriting by nondeterministic finite automata [6], can be used to decide the control state reachability problem for MSPs. Furthermore, we will demonstrate how to reduce the control state reachability problem for recursive ping-pong protocols with Dolev-Yao attacker model to the control state reachability problem for MSPs. This reduction gives us an EXPTIME algorithm to decide the control state reachability problem for recursive ping-pong protocols. We also show that the problem is NP-hard. Closing the gap between both results is left for future research. Finally, we also demonstrate that an (infinite) fragment of the concurrent constraint programming language [22] can be naturally encoded into our MSP formalism.

2 Facts about Prefix Rewriting on Words

Let us first state some standard facts about prefix rewriting.

Let Γ be a finite alphabet. A *prefix rewriting system* is a finite set R of *rules* such that $R \subseteq \Gamma^* \times \Gamma^*$. For an element $(v, w) \in R$ we usually write $v \longrightarrow w$.

The system R generates a transition system via the standard prefix rewriting.

$$\frac{(v \longrightarrow w) \in R, \quad t \in \Gamma^*}{vt \longrightarrow_R wt}$$

Proposition 1 (see, e.g., [7, 12]). *Let $T \subseteq \Gamma^*$ be a regular set of words. Then the sets $pre_R(T) \stackrel{\text{def}}{=} \{u' \in \Gamma^* \mid \exists u \in T.\ u' \longrightarrow_R u\}$ and $pre_R^*(T) \stackrel{\text{def}}{=} \{u' \in \Gamma^* \mid \exists u \in T.\ u' \longrightarrow_R^* u\}$ are also regular sets. Moreover, if T is given by a nondeterministic finite automaton A then we can in polynomial time construct the automata for $pre_R(T)$ and $pre_R^*(T)$ of polynomial size w.r.t. to A.*

3 Monotonic Set-Extended Prefix Rewriting

In this section we shall introduce a new computational model called *Monotonic Set-extended Prefix rewriting* (MSP). First, we provide its definition and then we argue for the decidability of control state reachability in MSP.

Let Γ be a finite alphabet and let Q be a finite set of control states together with a partial ordering relation $\leq \subseteq Q \times Q$. By $p < q$ we denote that $p \leq q$ and $p \neq q$. A *monotonic set-extended prefix rewriting system* (MSP) is a finite set R of rules of the form $pv \longrightarrow qw$ where $p, q \in Q$ such that $p \leq q$ and $v, w \in \Gamma^*$.

Assume a fixed MSP R. A *configuration* of R is a pair (p, T) where $p \in Q$ and $T \subseteq \Gamma^*$. The semantics is given by the following rule.

$$\frac{(pv \longrightarrow qw) \in R, \quad vt \in T}{(p, T) \longrightarrow_R (q, T \cup \{wt\})}$$

Let (p_0, T_0) be an *initial configuration* of MSP R such that $T_0 \neq \emptyset$ is a regular set and let $p_G \in Q$. The *control state reachability problem* is to decide whether $(p_0, T_0) \longrightarrow_R^* (p_G, T)$ for some T.

We will demonstrate the decidability of control state reachability for MSPs. From now on assume a fixed MSP R with an initial configuration (p_0, T_0) and a goal control state p_G. We proceed in three steps. First, we give some preliminaries on the relationship between MSPs and prefix rewriting systems. Then we introduce several notions: control path, π-scheme, and feasibility of a π-scheme. We show that the control state reachability problem reduces to the feasibility problem of π-schemes. Finally, we give an algorithm for feasibility of π-schemes, and give an upper bound on the complexity of the control state reachability problem.

Preliminaries. Given a rule $r = pv \to qw$ of R, we denote by $u_1 \longrightarrow_r u_2$ the fact that qu_2 can be obtained from pu_1 by applying r, i.e., that there is $t \in \Gamma^*$ such that $u_1 = vt$ and $u_2 = wt$. Furthermore, for every state $p \in Q$ we define the set R_p of rules from R that start from p and do not change the control state, i.e., $R_p \stackrel{\text{def}}{=} \{pv \longrightarrow pw \mid (pv \longrightarrow pw) \in R\}$, and write $v \longrightarrow_{R_p}^* w$ to denote that there is a sequence $v \longrightarrow_{r_1} v_1 \longrightarrow_{r_2} \ldots \longrightarrow_{r_n} w$ such that $r_i \in R_p$ for every $i \in \{1, \ldots, n\}$. We have the following obvious connection between $(p, T) \longrightarrow_{R_p}^* (p, T')$ and $v \longrightarrow_{R_p}^* w$.

Lemma 1. *If* $(p,T) \longrightarrow^*_{R_p} (p,T')$ *then for every* $w \in T'$ *there is* $v \in T$ *such that* $v \longrightarrow^*_{R_p} w$.

Control paths and π-schemes. Assume a given MSP R. A *control path* is a sequence $\pi = p_0 r_1 p_1 r_2 p_2 \dots p_{n-1} r_n p_n$, where $n \geq 0$, satisfying the following properties:

- $p_i \in Q$ for $i \in \{0, \dots, n\}$ and $r_j \in R$ for every $j \in \{1, \dots n\}$,
- $p_0 < p_1 < p_2 < \dots < p_n$, and
- for every $j \in \{1, \dots n\}$, r_j is a rule of the form $p_{j-1} v \longrightarrow p_j w$ for some v and w.

Note that the length of π is bounded by the length of the longest chain in $(Q, \leq)$. An execution of R starting at (p_0, T_0) *conforms to* π if the sequence of rules used in it belongs to the regular expression $\mathcal{E}(\pi) = R^*_{p_0} r_1 R^*_{p_1} \dots R^*_{p_{n-1}} r_n$ (for $n = 0$, to the regular expression ϵ). Obviously, p_G is reachable from (p_0, T_0) if and only if there is a control path $\pi = p_0 r_1 \dots r_{n-1} p_n$ such that $p_n = p_G$ and some execution of R ending in p_G conforms to π.

In the next lines, we will need to distinguish more precisely to which words the rules from a control path are applied in a particular computation of R. For this we introduce the notions of a π-scheme and feasibility of π-schemes.

A *π-scheme* is a labelled directed acyclic graph $S = (N, E, \lambda)$ where N is a finite set of nodes, $E \subseteq N \times N$ is a set of edges, and $\lambda \colon E \to X$ is a function that assigns to each edge e an element $\lambda(e)$ from the set $X = \{R^*_{p_0}, r_1, R^*_{p_1}, \dots, R^*_{p_{n-1}}, r_n\}$. Moreover, S satisfies the following properties (where $\mathbf{n} \xrightarrow{l} \mathbf{n}'$ denotes that S has an edge from $\mathbf{n}$ to $\mathbf{n}'$ labelled by l):

(a) every node has at most one predecessor (i.e., S is a forest) and there are no isolated nodes,
(b) for every $i \in \{1, \dots, n\}$, there is exactly one edge labelled by r_i, and
(c) for every path $\mathbf{n}_0 \xrightarrow{l_1} \mathbf{n}_1 \dots \mathbf{n}_{k-1} \xrightarrow{l_k} \mathbf{n}_k$ leading from a root to a leaf, the sequence $l_1 \dots l_k$ can be obtained from $\mathcal{E}(\pi)$ by deleting 0 or more, but not all, of $r_1, r_2, \dots, r_n$, and there are no two different paths with the same sequence of labels.

Figure 1 shows a π-scheme for the control path $\pi = p_0 r_1 \dots p_3 r_4 p_4$. Intuitively, a π-scheme describes what type of words were necessary to perform the changes of control states described by a given control path. In our example, the first upper chain means that in order to employ the rule r_4 which changes a control state p_3 into p_4, we need to take some word from the initial pool T_0, modify it possibly by the rules from $R^*_{p_0}, \dots, R^*_{p_3}$ (in this order) and finally use the resulting word to enable the application of the rule r_4. In general, the situation can be more complicated as demonstrated in the lower part of Figure 1 for the remaining rules r_1, r_2 and r_3. A word resulting from an initial word taken from the set T_0 and possibly modified by $R^*_{p_0}$ is used to enable the application of the rule r_1. The resulting word is later on necessary for both the application of the rule r_2 and r_3.

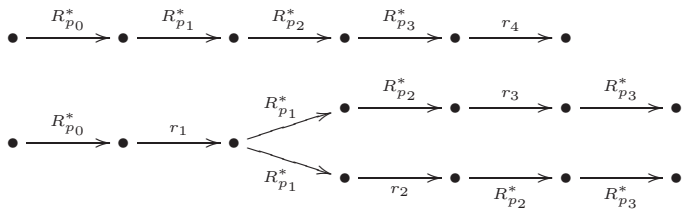

Fig. 1. A π-scheme for $\pi = p_0 r_1 \dots r_4 p_4$

Two π-schemes are *isomorphic* if they are equal up to renaming of the nodes. Note that every π-scheme is finite and there are only finitely many non-isomorphic π-schemes. We obtain a very rough upper bound on the number of π-schemes for a given control path π.

Lemma 2. *Let $\pi = p_0 r_1 p_1 r_2 p_2 \dots r_n p_n$ be a control path. There are at most $n^{O(n)}$ π-schemes up to isomorphism.*

Proof. Let $X = \{R^*_{p_0}, r_1, R^*_{p_1}, \dots, R^*_{p_{n-1}}, r_n\}$ be the alphabet of regular expressions associated with the control path π. Given a π-scheme S, denote by $P(S)$ the words $l_1 \dots l_n \in X^*$ such that $\mathbf{n}_0 \xrightarrow{l_1} \dots \xrightarrow{l_n} \mathbf{n}_n$ is a path of S leading from a root to a leaf. By condition (c), every element of $P(S)$ contains all the letters $R^*_{p_0}, \dots, R^*_{p_{n-1}}$ exactly once (and in that order), plus one or more of the letters $r_1, \dots, r_n$, also in that order. Therefore, there exists a bijection between the partitions of the set $\{r_1, \dots, r_n\}$ and the languages $L \subseteq X^*$ for which there is a π-scheme S satisfying $P(S) = L$. Since the number of partitions is bounded by n^n, this is also an upper bound of the number of languages L.

Now, given a fixed language L, we give a bound on the number of schemes S such that $P(S) = L$. For this, we observe that a π-scheme can be constructed by adding paths corresponding to the words of L one by one. For instance, the π-scheme of Figure 1 can be constructed by adding paths for $R^*_{p_0} R^*_{p_1} R^*_{p_2} R^*_{p_3} r_4$, $R^*_{p_0} r_1 R^*_{p_1} R^*_{p_2} r_3 R^*_{p_3}$ and $R^*_{p_0} r_1 R^*_{p_1} r_2 R^*_{p_2} R^*_{p_3}$. Each time a new path is added, we decide how to merge it with the previous ones. In the example, we decide to keep the second path disjoint with the first, and merge the third path with the second one up to $R^*_{p_0} r_1$. Since the paths have length at most $2n$ (condition (c)), for the second path we have at most $2n + 1$ different ways of merging it with the first one, for the third path at most $2(2n+1)$ ways, and for the i-th we have $i(2n + 1)$. Their product is bounded by $n^{O(n)}$ and so for each language L there are at most $n^{O(n)}$ π-schemes. Since the number of languages is bounded by n^n, we get a bound of $n^n \cdot n^{O(n)} \in n^{O(n)}$ on the total number of schemes. □

We shall now formally define feasibility of π-schemes. A π-scheme is *feasible from $T \subseteq \Gamma^*$* if there is a function $f\colon N \to \Gamma^*$ such that

(d) if $\mathbf{n}$ is a root, then $f(\mathbf{n}) \in T$, and
(e) if $\mathbf{n} \xrightarrow{R^*_{p_i}} \mathbf{n}'$, then $f(\mathbf{n}) \longrightarrow^*_{R_{p_i}} f(\mathbf{n}')$, and if $\mathbf{n} \xrightarrow{r_i} \mathbf{n}'$, then $f(\mathbf{n}) \longrightarrow_{r_i} f(\mathbf{n}')$.

Intuitively, the function f determines which particular words are used in order to realize a given π-scheme by some concrete execution in R.

Proposition 2. *Let π be a control path. There is an execution of R starting from (p_0, T_0) and conforming to π iff some π-scheme is feasible from T_0.*

Proof. ($\Rightarrow$): Let $\pi = p_0 r_1 p_1 r_2 p_2 \dots r_n p_n$. The proof is by induction on n. In fact, we are going to prove a stronger claim which moreover requires that the function f from the definition of feasibility is injective on the roots of the scheme. If $n = 0$, then $\pi = p_0$. The empty π-scheme is obviously feasible from T_0.

Assume $n > 0$. Let $(p_0, T_0) \longrightarrow^*_{R_{p_0}} (p_0, T_0') \longrightarrow_{r_1} (p_1, T_1)$ be the initial part of the execution conforming to π, which exists by assumption. Let v_0' be the word of T_0' to which the rule r_1 is applied, and let v_1 be the word obtained by the application of the rule. We have $T_1 = T_0' \cup \{v_1\}$. By Lemma 1, there is a word $v_0 \in T_0$ such that $v_0 \longrightarrow^*_{R_{p_0}} v_0' \longrightarrow_{r_1} v_1$.

Now, let $\pi_1 = p_1 r_2 p_2 \dots r_n p_n$. The rest of the execution whose initial part is given above conforms to π_1. By induction hypothesis there is a π_1-scheme S_1 feasible from T_1 by means of a function f_1. Let $\mathbf{n}_1$ be a root of S_1 such that $f_1(\mathbf{n}_1) = v_1$ (if there is no such root, then redefine S_1 as the result of adding a new isolated path with root node $\mathbf{n}_1$ and with edges labelled by $R^*_{p_1}, \dots, R^*_{p_{n-1}}$ and extend f_1 by $f_1(\mathbf{n}) = v_1$ for all nodes $\mathbf{n}$ on the added path). We construct a π-scheme S and a function f showing that S is feasible from (p_0, T_0). S is obtained by adding new nodes and edges to S_1, and by extending f_1 to a new function f.

In a first step, we add a new edge $\mathbf{n}' \xrightarrow{r_1} \mathbf{n}_1$ and set $f(\mathbf{n}') = v_0'$. We claim that every root $\mathbf{n}_1'$ of this new graph satisfies $f(\mathbf{n}_1') \in T_0'$. For the proof, consider two cases. If $\mathbf{n}_1' = \mathbf{n}'$, $f(\mathbf{n}_1') \in T_0'$ by definition. Otherwise, $\mathbf{n}_1'$ is a root of π_1, $\mathbf{n}_1' \neq \mathbf{n}_1$. By the definition of feasibility, we have $f(\mathbf{n}_1') \in T_1$. Since f_1 is injective on roots, we have $f_1(\mathbf{n}_1') \neq v_1$, and so, since $T_1 = T_0' \cup \{v_1\}$ by the semantics of MSPs, we have $f(\mathbf{n}_1') \in T_0'$, and the claim is proved.

For the second step in the construction, observe that, by Lemma 1, for every root $\mathbf{n}_1'$ of the graph obtained after the first step there is a word $v_0 \in T_0$ such that $v_0 \longrightarrow^*_{R_{p_0}} f(\mathbf{n}_1')$. For each such v_0 we add a new node $\mathbf{n}_{v_0}$ and a new edge $\mathbf{n}_{v_0} \xrightarrow{R^*_{p_0}} \mathbf{n}_1'$ to the graph, and set $f(\mathbf{n}_{v_0}) = v_0$. It is easy to see that the result is a graph satisfying conditions (a)-(e) by means of the function f, and so this π-scheme is feasible from T_0.

($\Leftarrow$): Let $\pi = p_0 r_1 p_1 r_2 p_2 \dots r_n p_n$. Let S be a π-scheme feasible from T_0 by means of a function f. The proof is by induction on n. If $n = 0$ then $\mathcal{E}(\pi) = \epsilon$ and, by condition (c) and (a), S has no edges and no nodes. It follows that the empty execution conforms to π.

If $n > 0$, let S_0' be the graph obtained from S be removing all edges labelled by $R^*_{p_0}$ and all nodes that became isolated, and let S_1 be the graph obtained from S_0' by removing the unique edge labelled by r_1 and possibly the source node of this edge should it became an isolated node. Furthermore, let T_0', T_1 be the sets of words v such that $f(\mathbf{n}) = v$ for some root of S_0', S_1, respectively. It is easy to see that S_1 is a π_1-scheme feasible from T_1 by means of the restriction of

f to the nodes of S_1, where $\pi_1 = p_1 r_2 p_2 \dots r_n p_n$. By induction hypothesis, there is an execution of R starting from (p_1, T_1) and conforming to π_1. We show that there is a sequence

$$(p_0, T_0) \longrightarrow^*_{R_{p_0}} (p_0, T_0' \cup T) \longrightarrow_{r_1} (p_1, T_1 \cup T)$$

conforming to π for an adequate set T.

Let $\mathbf{n}_0' \xrightarrow{r_1} \mathbf{n}_1$ be the unique edge of S_0' labelled by r_1. Then, the set of roots of S_0' is equal to the set of roots of S_1 minus $\mathbf{n}_1$ plus $\mathbf{n}_0'$, and so $T_0' = (T_1 \setminus f(\mathbf{n}_1)) \cup \{f(\mathbf{n}_0')\}$. Since π is feasible by means of f, we have $f(\mathbf{n}_0') \longrightarrow_{r_1} f(\mathbf{n}_1)$,and so $(p_0, T_0' \cup T) \longrightarrow_{r_1} (p_1, T_1 \cup T)$ for any set T. Let us now show that some set T satisfies $(p_0, T_0) \longrightarrow^*_{R_{p_0}} (p_0, T_0' \cup T)$. For this, it suffices to show that for every $v_0' \in T_0'$ there is a word $v_0 \in T_0$ such that $v_0 \longrightarrow^*_{R_{p_0}} v_0'$, because in this case we can take for T all the words reached during the executions of the sequences $v_0 \longrightarrow^*_{R_{p_0}} v_0'$. To prove this, choose an arbitrary $v_0' \in T_0'$. By definition, there is a root $\mathbf{n}_0'$ of S_0' such that $f(\mathbf{n}_0') = v_0'$. By the definition of π-scheme, S has an edge $\mathbf{n}_0 \xrightarrow{R^*_{p_0}} \mathbf{n}_0'$, where $\mathbf{n}_0$ is a root. Since S is feasible by means of f, we have $f(\mathbf{n}_0) \longrightarrow^*_{R_{p_0}} f(\mathbf{n}_0')$. So we can just take $v_0 = f(\mathbf{n}_0)$. □

Proposition 2 and Lemma 2 lead to the following algorithmic idea for deciding if there is a set T such that $(p_0, T_0) \longrightarrow^*_R (p_G, T)$:

- enumerate all control paths $\pi = p_0 r_1 \dots r_n p_n$ such that $p_n = p_G$ (their number is finite, because the length of a control path is bounded by the length of the longest $\leq$-chain in Q),
- for each control path π, enumerate all π-schemes (their number is finite by Lemma 2), and
- for each π-scheme S, decide if S is feasible.

Checking feasibility of π-schemes. To check feasibility of a π-scheme S, we first need to define the feasibility of a node $\mathbf{n}$ for a word $v \in \Gamma^*$. Let $\mathbf{n}$ be a node of S, and let $N_{\mathbf{n}}$ denote the set of all descendants of $\mathbf{n}$. We say that $\mathbf{n}$ is *feasible* for $v \in \Gamma^*$ if there is a function $f_n \colon N_{\mathbf{n}} \to \Gamma^*$ satisfying condition (e) of the definition of feasibility of a π-scheme, and such that $f_n(\mathbf{n}) = v$. Now, let $W(\mathbf{n})$ denote the set of all words v such that $\mathbf{n}$ is feasible for v. By Proposition 2, S is feasible from a set $T \subseteq \Gamma^*$ iff $T \cap W(\mathbf{n}) \neq \emptyset$ for every root $\mathbf{n}$ of S.

An apparent complication to compute the set $W(\mathbf{n})$ is the fact that it may be infinite, which prevents us from enumerating its elements in finite time. We solve this problem by showing that $W(\mathbf{n})$ is always a regular language, and that it is possible to effectively construct a nondeterministic automaton recognising it. The key is the following characterization of W.

Proposition 3. *Let $\mathbf{n}$ be a node of a π-scheme S, then*

$$W(\mathbf{n}) = \Gamma^* \cap \bigcap_{\mathbf{n} \xrightarrow{R_p{}^*} \mathbf{n}'} pre^*_{R_p}(W(\mathbf{n}')) \quad \cap \bigcap_{\mathbf{n} \xrightarrow{r} \mathbf{n}'} pre_r(W(\mathbf{n}'))$$

where $pre_r(T) \stackrel{\text{def}}{=} pre_{\{v \longrightarrow w\}}(T)$ such that r is of the form $pv \longrightarrow qw$.

Proof. By condition (e) in the definition of feasibility, $\mathbf{n}$ is feasible for v iff either $\mathbf{n}$ is a leaf, or for every edge $\mathbf{n} \xrightarrow{R_p{}^*} \mathbf{n}'$ there is a word $w \in W(\mathbf{n}')$ such that $v \longrightarrow_{R_p}^* w$ and for every edge $\mathbf{n} \xrightarrow{r} \mathbf{n}'$ there is a word $w \in W(\mathbf{n}')$ such that $v \longrightarrow_r w$. The claim follows then immediately from the definition of $pre^*_{R_p}$ and pre_r. □

Notice that if $\mathbf{n}$ is a leaf then $W(\mathbf{n}) = \Gamma^*$. Let $\mathbf{n}_0$ and $\mathbf{n}_1$ be the upper and lower root in the π-scheme of Figure 1. If we abbreviate the expression $pre^*_{R_{p_i}}(pre^*_{R_{p_{i+1}}}(\ldots(pre^*_{R_{p_j}}(T))\ldots)$ to $pre^*_{i\ldots j}(T)$ for $i \leq j$, we get

$$\begin{array}{l} W(\mathbf{n}_0) = pre^*_{0123}(pre_{r_4}(\Gamma^*)) \\ W(\mathbf{n}_1) = pre^*_0\big(pre_{r_1}\big(\ pre^*_{12}(pre_{r_3}(pre^*_3(\Gamma^*))) \cap pre^*_1(pre_{r_2}(pre^*_{23}(\Gamma^*)))\ \big)\big) \quad . \end{array}$$

Proposition 3 allows us to compute $W(\mathbf{n})$ bottom-up, starting at the leaves of S, and computing $W(\mathbf{n})$ after having computed $W(\mathbf{n}')$ for every immediate successor of $\mathbf{n}$. By Proposition 1, the pre^* and pre operations preserve regularity, and are effectively computable. Since regular languages are closed under intersection, $W(\mathbf{n})$ is effectively computable.

Hence control state reachability of monotonic set-extended prefix rewriting systems is decidable.

Theorem 1. *Control state reachability of monotonic set-extended prefix rewriting systems is decidable.*

Finally, we also establish a singly exponential upper bound of the running time of the algorithm.

Proposition 4. *Let R be an MSP over a finite alphabet Γ and a set of control states $(Q, \leq)$ and let c be the length of the longest $\leq$-chain. Let m be the maximum over all $p, q \in Q$, $p \neq q$, of the number of rules of the form $pv \longrightarrow qw$ in R. Let $T_0 \subseteq \Gamma^*$ be a regular set of words represented by a nondeterministic automaton of size a. We can decide if there is a set T such that $(p_0, T_0) \longrightarrow_R^* (p_G, T)$ for a given control state p_G in deterministic time $(|Q| + m + |\Gamma|)^{O(c)} \cdot a$.*

Proof. Since the number of states appearing in a control path is at most c, the number of control paths of R is bounded by $|Q|^c \cdot m^c$. Since the number of π-schemes for a given control path with n states is $n^{O(n)}$ (Lemma 2) the total number of schemes is at most $|Q|^c \cdot m^c \cdot c^{O(c)}$, which can be bounded by $(|Q| + m)^{O(c)}$ because $c \leq |Q|$.

Let us now compute the time required to check the feasibility of a scheme. We claim that, given a scheme S with k leaves, the cost of computing $W(\mathbf{n})$ for every root $\mathbf{n}$ of S is $|\Gamma|^{O(k)}$. To see this, observe first that the size of the automata accepting Γ^* (the set $W(\mathbf{n})$ for a leaf $\mathbf{n}$) is $O(|\Gamma|)$. Moreover, if the sizes of the automata for the children of a node have sizes $a_1, \ldots, a_k$, the size of the automaton for the node is bounded by the product $a_1 a_2 \ldots a_k$, since we have to intersect all the automata. Since the scheme has k leaves, we obtain $|\Gamma|^{O(k)}$ as the total cost, which proves the claim. Finally, we have to check whether

$T_0 \cap W(\mathbf{n}) \neq \emptyset$ holds for every root $\mathbf{n}$, which can be done in time $a \cdot |\Gamma|^{O(k)} \cdot k$ time. Since $k \leq c$ due to condition (c) in the definition of a π-scheme, the feasibility of a scheme can be thus checked in $a \cdot |\Gamma|^{O(c)} \cdot c$.

Since the number of schemes is $(|Q| + m)^{O(c)}$ and the feasibility of each one of them can be checked in $a \cdot |\Gamma|^{O(c)} \cdot c$ time, the running time of the algorithm is $(|Q| + m + |\Gamma|)^{O(c)} \cdot a$. □

4 Recursive Ping-Pong Protocols

In this section we define the class of recursive ping-pong protocols.

Let $\mathcal{K}$ be a set of *symmetric encryption keys*. A word $w \in \mathcal{K}^*$ naturally represents an encrypted message with the outer-most encryption on the left hand-side. For example k_1k_2k represents the plain text message (key) k encrypted first by the key k_2, followed by the key k_1. In the usual notation k_1k_2k hence stands for $\{\{k\}_{k_2}\}_{k_1}$. The *analysis* of a set of messages $T \subseteq \mathcal{K}^*$ is the least set $\mathcal{A}(T)$ satisfying

$$\mathcal{A}(T) = T \cup \{w \mid kw \in \mathcal{A}(T),\ k \in \mathcal{K} \cap \mathcal{A}(T)\}. \tag{1}$$

The *synthesis* of a set of messages $T \subseteq \mathcal{K}^*$ is the least set $\mathcal{S}(T)$ satisfying

$$\mathcal{S}(T) = T \cup \{kw \mid w \in \mathcal{S}(T),\ k \in \mathcal{K} \cap \mathcal{S}(T)\}. \tag{2}$$

Lemma 3. *Let n be a natural number, $T \subseteq \mathcal{K}^*$ and let $Q_i \in \{\mathcal{A}, \mathcal{S}\}$ for all i, $1 \leq i \leq n$. It holds that $Q_1(Q_2(\ldots(Q_n(T))\ldots)) \subseteq \mathcal{S}(\mathcal{A}(T))$.*

Proof. This standard fact (see also [5, Prop. 2.1]) follows directly from the following straightforward laws: $\mathcal{S}(\mathcal{S}(T)) = \mathcal{S}(T)$; $\mathcal{A}(\mathcal{A}(T)) = \mathcal{A}(T)$; $\mathcal{A}(\mathcal{S}(T)) \subseteq \mathcal{S}(\mathcal{A}(T))$; and $T_1 \subseteq T_2$ implies $\mathcal{S}(T_1) \subseteq \mathcal{S}(T_2)$. □

The set of *compromised keys* $C(T) \subseteq \mathcal{K}$ for a given set $T \subseteq \mathcal{K}^*$ of messages is defined by $C(T) \stackrel{\text{def}}{=} \mathcal{K} \cap \mathcal{A}(T)$. A *recursive ping-pong protocol* is a finite set Δ of *process definitions* over a finite set $\mathcal{C}onst$ of *process constants* such that for every $P \in \mathcal{C}onst$ the set Δ contains exactly one process definition of the form

$$P \stackrel{\text{def}}{=} \sum_{i \in I} [?v_i \triangleright .!w_i \triangleright].P_i$$

where I is a finite index set such that $P_i \in \mathcal{C}onst$ and $v_i, w_i \in \mathcal{K}^*$ for all $i \in I$. We shall denote the empty sum as *Nil*. The intuition is that for any $i \in I$ the process P can input a message of the form $v_it \in \mathcal{K}^*$, output w_it, and behave as P_i. The symbol $'?'$ represents the input prefix, $'!'$ the output prefix, and $'\triangleright'$ the rest (suffix) of the communicated message.

A *configuration* of a ping-pong protocol Δ is a pair (P, T) where $P \in \mathcal{C}onst$ and $T \subseteq \mathcal{K}^*$. The set T is also called a *pool*. The reduction semantics is defined by the following rule.

$$\frac{P \stackrel{\text{def}}{=} \sum_{i \in I} [?v_i \triangleright .!w_i \triangleright].P_i, \quad i \in I, \quad v_it \in \mathcal{S}(\mathcal{A}(T))}{(P, T) \longrightarrow_\Delta (P_i, T \cup \{w_it\})}$$

Definition 1. *Let* (P_0, T_0) *be a given* initial configuration *such that* $T_0 \neq \emptyset$ *is a regular set and let* $P_G \in \mathcal{C}onst$. *The* control state reachability problem *is to decide whether* $(P_0, T_0) \longrightarrow_\Delta^* (P_G, T)$ *for some* T.

Example 1. Let Δ be a protocol consisting of $P_0 \stackrel{\text{def}}{=} [?k_1k_2 \triangleright .!k_2k_1 \triangleright].P_1$, $P_1 \stackrel{\text{def}}{=} [?k_2k_1 \triangleright .!k_*k_2 \triangleright].P_2$, and $P_2 \stackrel{\text{def}}{=} \mathit{Nil}$. Let $T_0 = \{k_*, k_1k_2\}$ be the initial pool in which k_* is the only compromised key. Then, $(P_0, T_0) \longrightarrow_\Delta (P_1, T_1) \longrightarrow_\Delta (P_2, T_2)$ where $T_1 = T_0 \cup \{k_2k_1\}$, and $T_2 = T_1 \cup \{k_*k_2\}$. At control point P_2 (but not before) the attacker can learn the keys k_1 and k_2. Indeed, he can use the compromised key k_* to extract k_2 from the last message k_*k_2 exchanged in the protocol, and k_2 to extract k_1 from the message k_2k_1. Thus, we have that $C(T_2) = \{k_*, k_1, k_2\}$. Suppose that messages are always terminated by the symbol $\perp$. In order to test if the attacker has uncovered, e.g., the key k_1, we can add (using $+$) to each process definition the observer process defined as $[?k_1 \perp \triangleright .!k_1 \perp \triangleright].\mathit{Error}$. Reachability of the control state *Error* denotes a violation of secrecy for our protocol.

Remark 1. Since we allow nondeterminism in the definitions of process constants, the control state reachability problem for a parallel composition of recursive ping-pong processes can be reduced (using a standard product construction) to control state reachability for a single recursive process. For example assume that $\mathcal{C}onst = \{P_1, P_2, P_2'\}$ such that $P_1 \stackrel{\text{def}}{=} [?k_1 \triangleright .!k_2 \triangleright].P_1$, $P_2 \stackrel{\text{def}}{=} [?k_1 \triangleright .!\triangleright].P_2' + [?k_2 \triangleright .!\triangleright].P_2$, and $P_2' \stackrel{\text{def}}{=} [?k_1k_2 \triangleright .!k_2k_1 \triangleright].P_2$.

The parallel composition $P_1 \parallel P_2$ as defined e.g. in [4] can be modelled by the following protocol with $\mathcal{C}onst = \{(P_1, P_2), (P_1, P_2')\}$, where

$$(P_1, P_2) \stackrel{\text{def}}{=} [?k_1 \triangleright .!k_2 \triangleright].(P_1, P_2) + [?k_1 \triangleright .!\triangleright].(P_1, P_2') + [?k_2 \triangleright .!\triangleright].(P_1, P_2)$$
$$(P_1, P_2') \stackrel{\text{def}}{=} [?k_1 \triangleright .!k_2 \triangleright].(P_1, P_2') + [?k_1k_2 \triangleright .!k_2k_1 \triangleright].(P_1, P_2) \quad .$$

Note that by applying the reduction above, there is a possible exponential state-space explosion (however, it is exponential only in the number of parallel agents; in many protocols this number is fixed and small). In what follows we measure our complexity results in terms of the flat (single process) system.

Remark 2. In [14] the reachability problem for a replicative variant of the ping-pong calculus without any notion of an active intruder was reduced to reachability of weak Process Rewrite Systems (wPRSs) [17]. In a wPRS rewriting rules contain both parallel and sequential operators and are moreover enriched with a control state unit defined over a partially ordered set of states. Configurations consist of a control state together with a term built using sequential and parallel composition as in PRS [18]. These kinds of terms can be used, e.g., to represent a *multiset* of words (for this we in fact only need a subclass of wPRS called wPAD). The main restriction of wPRSs, which guarantees that reachability is still decidable, is that state updates can only lead to states that are greater or equal than the current one. MSPs borrow this idea from wPRSs although they

represent a different (incomparable) extension of prefix rewriting. To illustrate this, let us go back to Example 1. Suppose that we model the protocol using the wPRS rules

$$p_0\ k_1k_2 \to p_1\ k_2k_1$$
$$p_1\ k_2k_1 \to p_2\ k_*k_2$$

where p_0, p_1 and p_2 are control states such that $p_0 < p_1 < p_2$. In wPRS we can consider configurations like $(p, w_1 \parallel \ldots \parallel w_n)$ where p is a control state and $w_1, \ldots, w_n$ are words. The initial pool can be modelled then as $(p_0, k_1k_2 \parallel k_*)$. However, in order to give the attacker the possibility of extracting k_1, we in general need to duplicate an arbitrary number of times any message floating in the pool (otherwise they get consumed by a rewriting step). Thus, we would need an additional meta-rule $w \to w||w$ for any word w. This kind of rules are not expressible in wPRS (they generate term languages that are not regular) and replicating a bounded number of initial messages is not sufficient to solve the problem.

5 Translating Recursive Ping-Pong Protocols to MSP

In this section we provide a reduction from control state reachability for recursive ping-pong protocols to control state reachability for MSP.

There are two main problems: (i) How can the analysis and synthesis be captured by prefix rewriting rules? and (ii) How to ensure that the control state unit is monotonic even for arbitrary recursive ping-pong protocols?

We shall now provide answers to these problems. Intuitively, problem (i) can be solved by keeping track of the set of compromised keys. The set of compromised keys grows monotonically and can be stored as a part of the control state. The rules for analysis and synthesis can then use the knowledge of the currently compromised keys and once a new compromised key is discovered, the control state unit is updated accordingly. Problem (ii) is more challenging. We cannot simply store the current process constant in the control state as this would destroy monotonicity (we allow arbitrary recursive behaviour in the protocol). Instead, we observe that a recursive ping-pong protocol is essentially a directed graph where nodes are process constants and edges are labelled by actions of the form $\alpha = [?v \rhd .!w\rhd]$. Once a certain action was taken due to some message present in the pool then it is permanently enabled also any time in the future (messages added to the pool T are persistent). Assume that there is a cycle of length ℓ (counting the number of edges) in the graph such that all the actions $\alpha_1, \ldots, \alpha_\ell$ on this cycle were already taken in the past. Then it is irrelevant in exactly which process constant on the cycle we are as we can freely move along the cycle as many times as needed. This essentially means that we can replace such a cycle with $!(\alpha_1) \parallel \cdots \parallel !(\alpha_\ell)$ where ! is the operator of replication. This observation can be further generalized to strongly connected components in the graph.

Let Δ be a recursive ping-pong protocol with a set of process constants $\mathcal{C}onst$ and encryption keys $\mathcal{K}$. We shall formally demonstrate the reduction men-

tioned above. First, we introduce some notation. Let $\mathcal{T} \stackrel{\text{def}}{=} \{(P, \alpha_i, P_i) \mid P \in \mathcal{C}onst,\ P \stackrel{\text{def}}{=} \sum_{i \in I} \alpha_i.P_i,\ i \in I\}$ be a set of directed edges between process constants labelled by the corresponding actions. Let $E \subseteq \mathcal{T}$. We write $P \Longrightarrow_E P'$ whenever there is some α such that $(P, \alpha, P') \in E$. Assume that $P \in \mathcal{C}onst$ and $E \subseteq \mathcal{T}$. We define a strongly connected component in E represented by a process constant P as $\mathcal{S}cc(P, E) \stackrel{\text{def}}{=} \{P' \in \mathcal{C}onst \mid P \Longrightarrow_E^* P' \ \wedge \ P' \Longrightarrow_E^* P\}$.

Let us now define an MSP R. The alphabet is $\Gamma \stackrel{\text{def}}{=} \mathcal{K} \cup \{\bot\}$ where $\bot$ is a fresh symbol representing the end of messages. The control states of R are of the form $\langle S, E, C\rangle$ where

- $S \subseteq \mathcal{C}onst$ is the current strongly connected component,
- $E \subseteq \mathcal{T}$ is the set of already executed edges, and
- $C \subseteq \mathcal{K}$ is the set of compromised keys.

There are four types of rules in R called (analz), (synth), (learn) and (comm). The first three rules represent intruder's capabilities and the fourth rule models the communication with the environment.

(analz)	$\langle S, E, C\rangle k \longrightarrow \langle S, E, C\rangle \epsilon$	for all $k \in C$
(synth)	$\langle S, E, C\rangle \epsilon \longrightarrow \langle S, E, C\rangle k$	for all $k \in C$
(learn)	$\langle S, E, C\rangle k\bot \longrightarrow \langle S, E, C \cup \{k\}\rangle k\bot$	for all $k \in \mathcal{K}$
(comm)	$\langle S, E, C\rangle v \longrightarrow \langle \mathcal{S}cc(P', E'), E', C\rangle w$	where $E' = E \cup \{(P, \alpha, P')\}$ whenever there exists $P \in S$ and $(P, \alpha, P') \in \mathcal{T}$ such that $\alpha = [?v\triangleright .!w\triangleright]$

It is easy to define an ordering on states such that R is monotonic. The second and third component in the control states are non-decreasing w.r.t. $\subseteq$ and $\mathcal{T}$ and $\mathcal{K}$ are finite sets. For a fixed second coordinate E the strongly connected components (i.e. the values that the first coordinate S in the control state can take) form a directed acyclic graph. Let $T \subseteq \mathcal{K}^*$. By $T^\bot$ we denote the set $\{w\bot \mid w \in T\}$, i.e., the end symbol $\bot$ is appended to every message from T.

Lemma 4. *Let $P_0, P \in \mathcal{C}onst$ and $T_0 \subseteq \mathcal{K}^*$. If $(P_0, T_0) \longrightarrow_\Delta^* (P, T)$ for some T then $(\langle\{P_0\}, \emptyset, \emptyset\rangle, T_0^\bot) \longrightarrow_R^* (\langle S, E, C\rangle, T'^\bot)$ for some S, E, C and T' such that $P \in S$ and $T^\bot \subseteq T'^\bot$.*

Proof. By induction on the length of derivation in Δ we shall prove that for all natural numbers n it holds that if $(P_0, T_0) \longrightarrow_\Delta^n (P, T)$ for some T then $(\langle\{P_0\}, \emptyset, \emptyset\rangle, T_0^\bot) \longrightarrow_R^* (\langle S, E, C\rangle, T'^\bot)$ for some S, E, C and T' such that $P \in S$ and $T^\bot \subseteq T'^\bot$.

The case $n = 0$ is trivial. Let $n > 0$ which means that the derivation in Δ can be written as

$$(P_0, T_0) \longrightarrow_\Delta^{n-1} (P_1, T_1) \longrightarrow_\Delta (P, T).$$

By induction hypothesis we know that

$$(\langle\{P_0\},\emptyset,\emptyset\rangle, T_0^{\perp}) \longrightarrow_R^* (\langle S_1, E_1, C_1\rangle, T_1'^{\perp})$$

such that $P_1 \in S_1$ and $T_1^{\perp} \subseteq T_1'^{\perp}$. From the definition we know that the step $(P_1, T_1) \longrightarrow_\Delta (P, T)$ is because $(P_1, [?v \triangleright .!w\triangleright], P) \in \mathcal{T}$ such that $vt \in \mathcal{S}(\mathcal{A}(T_1))$ for some t and $T = T_1 \cup \{wt\}$. Starting from $(\langle S_1, E_1, C_1\rangle, T_1'^{\perp})$ we can update the third component C_1 to the set of compromised keys $C(T_1')$ by repeatedly using the rules (analz) and (learn) and reach a configuration $(\langle S_1, E_1, C(T_1')\rangle, T_1''^{\perp})$. Now we can build the message vt using the rules (analz) and (synth) and reach a configuration $(\langle S_1, E_1, C(T_1')\rangle, T_1'''^{\perp})$ such that $vt\perp \in T_1'''^{\perp}$ and $T_1' \subseteq T_1'''$. Finally, by one application of the rule (comm), we get

$$(\langle S_1, E_1, C(T_1')\rangle, T_1'''^{\perp}) \longrightarrow_R (\langle \mathcal{S}cc(P, E), E, C(T_1')\rangle, T'^{\perp})$$

where $E = E_1 \cup \{(P_1, [?v \triangleright .!w\triangleright], P)\}$, $T'^{\perp} = T_1'''^{\perp} \cup \{wt\perp\}$ and of course $P \in \mathcal{S}cc(P, E)$ and $T^{\perp} \subseteq T'^{\perp}$. Hence

$$(\langle\{P_0\},\emptyset,\emptyset\rangle, T_0^{\perp}) \longrightarrow_R^* (\langle \mathcal{S}cc(P, E), E, C(T_1')\rangle, T'^{\perp})$$

as required. □

We will now proceed to prove the other implication. In order to do that we will need the following straightforward proposition which essentially says that (i) messages are persistent and once a certain step from a process constant P in the protocol was possible in the past then it is permanently enabled also in any future configuration in the control location P, and (ii) that the set C in the control state is always a subset of the compromised keys.

Proposition 5. *If* $(\langle\{P_0\},\emptyset,\emptyset\rangle, T_0^{\perp}) \longrightarrow_R^* (\langle S, E, C\rangle, T^{\perp})$ *for some* S, E, C *and* T *then (i) for any* $(P, \alpha, P') \in E$ *there is some* T' *such that* $(P, T) \longrightarrow_\Delta (P', T')$ *by using the transition* (P, α, P')*, and (ii)* $C \subseteq C(T)$*.*

Lemma 5. *Let* $P_0 \in \mathcal{C}onst$ *and* $T_0 \subseteq \mathcal{K}^*$*. If we have* $(\langle\{P_0\},\emptyset,\emptyset\rangle, T_0^{\perp}) \longrightarrow_R^* (\langle S, E, C\rangle, T^{\perp})$ *for some* S, E, C *and* T *then for all* $P \in S$ *also* $(P_0, T_0) \longrightarrow_\Delta^* (P, T')$ *such that* $T \subseteq \mathcal{S}(\mathcal{A}(T'))$*.*

Proof. By induction on the length of derivation in R we shall prove that for all natural numbers n it holds that if $(\langle\{P_0\},\emptyset,\emptyset\rangle, T_0^{\perp}) \longrightarrow_R^n (\langle S, E, C\rangle, T^{\perp})$ for some S, E, C and T then for all $P \in S$ also $(P_0, T_0) \longrightarrow_\Delta^* (P, T')$ such that $T \subseteq \mathcal{S}(\mathcal{A}(T'))$.

The case $n = 0$ is trivial. Let $n > 0$ which means that the derivation in R can be written as

$$(\langle\{P_0\},\emptyset,\emptyset\rangle, T_0^{\perp}) \longrightarrow_R^{n-1} (\langle S_1, E_1, C_1\rangle, T_1^{\perp}) \longrightarrow_R (\langle S, E, C\rangle, T^{\perp})$$

and by the induction hypothesis we can assume that for all $P_1 \in S_1$ there is some T_1' such that $T_1 \subseteq \mathcal{S}(\mathcal{A}(T_1'))$ and $(P_0, T_0) \longrightarrow_\Delta^* (P_1, T_1')$. There are now four cases according to the type of rule used in the last derivation step in R:

– If (analz), (synth) or (learn) was used in the last step then no additional transition is needed to match this sequence in Δ as $T \subseteq \mathcal{S}(\mathcal{A}(T_1'))$ (see also Proposition 5 part (ii) and Lemma 3).
– Assume now that (comm) was used in the last derivation step. As P_1 was freely selected from the set S_1 we can assume that the application of (comm) was due to some $(P_1, \alpha, P_2) \in \mathcal{T}$. By definition $S = \mathcal{S}cc(P_2, E)$. Let us consider an arbitrary $P \in \mathcal{S}cc(P_2, E)$ and we will show that this process constant is reachable from (P_1, T_1'). Let the first transition be $(P_1, T_1') \longrightarrow_\Delta (P_2, T_1'')$ and we know that $T \subseteq \mathcal{S}(\mathcal{A}(T_1''))$. Because $P_2 \Longrightarrow_E P$, by a repeated application of Proposition 5 part (i) we get that $(P_2, T_1'') \longrightarrow_\Delta^* (P, T')$ such that $T_1'' \subseteq T'$ and hence also $T \subseteq \mathcal{S}(\mathcal{A}(T'))$.

□

The next theorem states the correctness of our reduction and follows directly from Lemma 4 and Lemma 5.

Theorem 2. *Let $P_0, P \in \mathcal{C}onst$ and $T_0 \subseteq \mathcal{K}^*$. It holds that $(P_0, T_0) \longrightarrow_\Delta^* (P, T)$ for some T if and only if $(\langle \{P_0\}, \emptyset, \emptyset \rangle, T_0^\perp) \longrightarrow_R^* (\langle S, E, C \rangle, T'^\perp)$ for some S, E, C and T' such that $P \in S$.*

Hence control state reachability for recursive ping-pong protocols is reducible to control state reachability for monotonic set-extended prefix rewriting systems, which is decidable by Theorem 1. We also obtain the following complexity upper bound.

Corollary 1. *Control state reachability for recursive ping-pong protocols is decidable in deterministic time $2^{O(n^4)} \cdot a$ where n is the size of the protocol written as a string and a is the size of a nondeterministic automaton representing the pool T_0.*

Proof. Observe that for a protocol Δ of size n (written as a string) we create an MSP with $2^{O(n)}$ control states (states consist of three components $\langle S, E, C \rangle$ where S, E and C are subsets of the sets $\mathcal{C}onst$, $\mathcal{T}$ and $\mathcal{K}$, respectively, and $|\mathcal{C}onst|, |\mathcal{T}|, |\mathcal{K}| \leq n$). Observe also that the longest $\leq$-chain in the resulting MSP is of length $O(n^3)$. This is due to the fact that the components E and C grow monotonically (see the rules (learn) and (comm)) and for a fixed component E there are at most $|\mathcal{C}onst| \leq n$ strongly connected components in the first coordinate S and they form a directed acyclic graph.

All together in combination with Proposition 4 we get the $2^{O(n^4)}$ complexity upper bound for one query on the MSP and according to Theorem 2 we issue at most $2^{O(n)}$ of such queries, which gives the total $2^{O(n^4)}$ upper bound. □

Finally, we show that control state reachability for recursive ping-pong protocols is at least NP-hard.

Theorem 3. *Control state reachability of recursive ping-pong protocols is NP-hard.*

Proof. By reduction from the satisfiability problem of boolean formulae in CNF. Let $C = C_1 \wedge C_2 \wedge \ldots \wedge C_k$ be a formula over boolean variables $x_1, \ldots, x_n$ such that for all i, $1 \leq i \leq k$, C_i is a disjunction of literals. We shall construct a ping-pong protocol Δ where $\mathcal{C}onst \stackrel{\text{def}}{=} \{X_1, \ldots, X_{n+1}, Y_1, \ldots, Y_{k+1}\}$ and $\mathcal{K} = \{C_1, \ldots, C_k, \bot\}$. Let for all i, $1 \leq i \leq n$, t_i be the sequence of keys $C_{i_1} C_{i_2} \cdots C_{i_\ell}$ such that $1 \leq i_1 < i_2 < \cdots < i_\ell \leq k$ and $C_{i_1}, C_{i_2}, \ldots, C_{i_\ell}$ are all the clauses where x_i occurs positively, and let f_i be the sequence of keys $C_{i_1} C_{i_2} \cdots C_{i_\ell}$ such that $1 \leq i_1 < i_2 < \cdots < i_\ell \leq k$ and $C_{i_1}, C_{i_2}, \ldots, C_{i_\ell}$ are all the clauses where x_i occurs negatively. The set Δ of process definitions is given as follows.

$$
\begin{array}{ll}
X_i \stackrel{\text{def}}{=} [?\bot \rhd .!t_i \bot \rhd].X_{i+1} + [?\bot \rhd .!f_i \bot \rhd].X_{i+1} & \text{for all } i,\ 1 \leq i \leq n \\
X_{n+1} \stackrel{\text{def}}{=} [?\bot \rhd .!\bot \rhd].Y_1 & \\
Y_i \stackrel{\text{def}}{=} [?C_i \rhd .!\rhd].Y_{i+1} + \sum_{1 \leq j < i} [?C_j \rhd .!\rhd].Y_i & \text{for all } i,\ 1 \leq i \leq k
\end{array}
$$

It is now easy to observe that the given formula C is satisfiable if and only if $(X_1, \{\bot\}) \longrightarrow^* (Y_{k+1}, T)$ for some T. The computation from $(X_1, \{\bot\})$ starts by going through the sequence of control constants $X_1, \ldots, X_{n+1}$ where for every i, $1 \leq i \leq n$, there is a choice, whether $t_i \bot$ or $f_i \bot$ (but not both) is added to the pool of messages. This corresponds to selecting a truth assignment. Then the control constant is changed from X_{n+1} to Y_1 without modifying the pool and the second (verification) phase starts. The move from Y_i to Y_{i+1} is possible only if the key C_i is present somewhere in the pool (which means that the corresponding clause is satisfied). The second summand in the definition of Y_i enables to remove duplicate clauses from the messages in order to access C_i. The control constant is not changed if the second summand of Y_i is used. Observe that the operations of analysis and synthesis cannot add any of the keys $C_1, \ldots, C_k$ to the pool, unless the protocol does it itself. Hence we can reach the control constant Y_{k+1} if and only if it was possible to satisfy all the clauses by the given truth assignment generated during the first phase. □

6 MSP and Concurrent Constraint Programming

We shall now outline some further applicability of our model of monotonic set-extended prefix rewriting. The MSP model shares some similarities with the ccp (concurrent constraint programming) language [22]. The ccp language is based on the notion of a monotonic store which is used by a collection of agents as a common blackboard to communicate by means of two primitives: *ask* to query the store without removing information, and *tell* to add information to the store.

This feature of the ccp semantics is similar in spirit to the way we defined the semantics of MSP. In an MSP configuration (p, T) the component T can be viewed as the current store. Since prefix rules never remove information from T, we can view them as a special case of the *ask* and *tell* operations. To make the connection between ccp and MSP more informal, we define next a fragment of ccp whose semantics can be directly encoded in MSP.

For this purpose, given a finite alphabet A, we will consider an instance of the ccp framework in which the constraint store is a set of strings $T \subseteq A^*$. Furthermore, we consider only one type of constraint formula of the form $v \cdot x$ where v is a string and x is a variable. If T is a set of strings (the current store), then $T \models v \cdot x$ via the binding $x \rightsquigarrow w$ if $vw \in T$.

Concerning the syntax of our ccp instance, we will restrict ourselves to processes defined as follows. A process declaration is defined as $p \leftarrow A$ where p is a process constant taken from a finite set P, and A is an agent. Agents (and actions) are defined by the following grammar.

$$\begin{array}{l} A \;\; ::= stop \;\; | \;\; \Sigma_{i=1}^{k}\, Act_i \\ Act ::= ask(v \cdot x) \rightarrow p \;\; | \;\; ask(v \cdot x) \rightarrow tell(w \cdot x) \rightarrow p \end{array}$$

Given a finite set of declarations $\mathcal{D} = \{D_1, \ldots, D_n\}$, a process P is defined as the (bounded) parallel compositions of ℓ agents, i.e., $P = A_1 \;||\; \ldots \;||\; A_\ell$. We assume that $||$ is associative and commutative. The operational semantics of a process P is defined in accordance with the semantics of ccp. Configurations are pairs $\langle P, T\rangle$ where P is a process and T is a store. The transition relation is defined as follows.

1. $\langle P_1 || P_2, T\rangle \rightarrow \langle P_1' || P_2, T'\rangle$ *if* $\langle P_1, T\rangle \rightarrow \langle P_1', T'\rangle$
2. $\langle p, T\rangle \rightarrow \langle A, T\rangle$ *if* $p \leftarrow A \in \mathcal{D}$
3. $\langle \Sigma_{i=1}^{k} Act_i, T\rangle \rightarrow \langle p, T\rangle$ *if* $Act_i = ask(v \cdot x) \rightarrow p$ *and* $vz \in T$ *for* $1 \leq i \leq k$
4. $\langle \Sigma_{i=1}^{k} Act_i, T\rangle \rightarrow \langle p, T \cup \{wz\}\rangle$ *if* $Act_i = ask(v \cdot x) \rightarrow tell(w \cdot x) \rightarrow p$ *and* $vz \in T$ *for* $1 \leq i \leq k$

Remark 3. The seemingly nonstandard action $ask(v \cdot x) \rightarrow tell(w \cdot x) \rightarrow p$ can be in full ccp encoded as $ask(v \cdot x) \rightarrow \exists n.\big(tell(w \cdot x \;\&\; tok(n)) \;||\; ask(tok(n)) \rightarrow p\big)$ where $tok(x)$ is a new type of constraint with one argument x.

Following the reduction schemes of the recursive definition of ping-pong processes, we know that we can extract a set of partially ordered locations from the parallel control flow graph of n recursive processes (by using the idea of strongly connected components). Under this assumption, we can focus our attention on the way we can model ccp agents and actions. Actions can be naturally mapped into prefix rules:

- The definition $a \leftarrow ask(v \cdot x) \rightarrow b$ for the i-th thread is mapped to a rule like $pv \rightarrow qv$ in which p and q are related by the change of the local state of the i-th thread from a to b.
- The definition $a \leftarrow ask(v \cdot x) \rightarrow tell(w \cdot x) \rightarrow b$ for the i-th thread is mapped to a rule like $pv \rightarrow qw$ in which p and q are related by the change of the local state of the i-th thread from a to b.

Although quite limited with respect to the original ccp model (e.g. it is not possible to spawn new processes), this instance is still nontrivial since the constraint store can grow unboundedly during the execution of a process.

The decidability of the control reachability problem for this instance of the ccp framework follows then from our result for MSP. Further extensions of the restricted ccp formalism are left for future work.

7 Conclusion

We proved that the control state reachability problem for recursive ping-pong protocols with Dolev-Yao attacker is decidable in deterministic exponential time. This result may seem surprising when one observes that recursive ping-pong protocols without any attacker are Turing powerful [13, 14]. However, a similar phenomenon occurs in FIFO-channel systems (automata whose transitions may add or retrieve items from channels, modelled as unbounded queues): if the channels are perfect, then the model is Turing powerful, but if one assumes that the channels are lossy, i.e., that the queues can spontaneously lose messages, then several important verification problems become decidable [8, 3].

We have used our results to prove the authenticity of Woo and Lam's protocol; to find a flaw in Otway and Rees' key distribution protocol and prove secrecy of a corrected version for arbitrarily many sessions; and to prove secrecy of Bull and Otway's recursive authentication protocol. To the best of our knowledge, no other method in the literature can deal simultaneously with these three problems in a fully automatic way. The approach of Rusinowitch and Turuani [21] can be used to prove authenticity of Woo and Lam's protocol, and Küsters has used regular transducers to automatically verify Bull and Otway's protocol [16]. However, these techniques can only deal with a bounded number of protocol sessions. In order to find the flaw in Otway and Rees' protocol they have to guess the right number of sessions, and they cannot directly prove secrecy of the corrected version. The replicative calculus of Amadio, Lugiez and Vanackère [5] can be used to model protocols with an unbounded number of sessions. However, the model over-approximates the semantics, i.e., there are executions of the model that do not correspond to executions of the protocol. Due to this over-approximation the secrecy or authenticity analysis can report false attacks. Appendix D explains this point in more detail.

Since our technique does not over-approximate the semantics, it is strictly more powerful than that of [5], at the price of a higher complexity (the algorithm of [5] runs in polynomial time), and it is incomparable with the techniques of [21, 16]. On the one hand, it provides an exact analysis for an arbitrary number of sessions; on the other hand, it is restricted to prefix rewriting, which can only deal with very restricted forms of pairing. Our model also allows only a bounded number of nonces. The distinguishing feature of our technique seems to be the possibility to model open-ended protocols with messages of unbounded length, in combination with an unrestricted (cyclic) communication structure.

The overall aim of the paper was to explore the verification boundaries for protocols with an unbounded number of rounds and to show the difference between protocols with and without an explicit Dolev-Yao attacker. We have therefore chosen a rather simple calculus to demonstrate our ideas. The contribution

of the paper should be understood as the first step towards the development of more general calculi which include a wider range of cryptographic primitives as outlined in the appendix. In particular, we think that decidability may then be proved by moving from regular word languages to regular tree languages (which would allow to introduce a more general notion of pairing into the language).

Our work also opens several venues for further research. MSPs are a rather natural computational model, which may have further applications, in particular in the area of coordination-based languages. To demonstrate this, we have presented an encoding of a fragment of the ccp language into MSP.

Acknowledgements. The second and the third author acknowledge a support from the Alexander von Humboldt Foundation.

References

1. M. Abadi and A.D. Gordon. A bisimulation method for cryptographic protocols. *Nordic Journal of Computing*, 5(4):267–303, 1998.
2. M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, 1996.
3. P.A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996.
4. R.M. Amadio and W. Charatonik. On name generation and set-based analysis in the Dolev-Yao model. In *Proceedings of the 13th International Conference on Concurrency Theory (CONCUR'02)*, volume 2421 of *LNCS*, pages 499–514. Springer-Verlag, 2002.
5. R.M. Amadio, D. Lugiez, and V. Vanackère. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science*, 290(1):695–740, October 2002.
6. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proceedings of the 8th International Conference on Concurrency Theory (CONCUR'97)*, volume 1243 of *LNCS*, pages 135–150. Springer-Verlag, 1997.
7. J.R. Büchi. Regular canonical systems. *Arch. Math. Logik u. Grundlagenforschung*, 6:91–111, 1964.
8. G. Cécé, A. Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(1):20–31, 1996.
9. H. Comon, V. Cortier, and J. Mitchell. Tree automata with one memory, set constraints, and ping-pong protocols. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP'01)*, volume 2076 of *LNCS*, pages 682–693. Springer-Verlag, 2001.
10. D. Dolev, S. Even, and R.M. Karp. On the security of ping-pong protocols. *Information and Control*, 55(1–3):57–68, 1982.
11. D. Dolev and A.C. Yao. On the security of public key protocols. *Transactions on Information Theory*, IT-29(2):198–208, 1983.
12. J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *LNCS*, pages 232–247. Springer-Verlag, 2000.

13. H. Hüttel and J. Srba. Recursive ping-pong protocols. In *Proceedings of the 4th International Workshop on Issues in the Theory of Security (WITS'04)*, pages 129–140, 2004.
14. H. Hüttel and J. Srba. Recursion vs. replication in simple cryptographic protocols. In *Proceedings of the 31st Annual Conference on Current Trends in Theory and Practice of Informatics (SOFSEM'05)*, volume 3381 of *LNCS*, pages 175–184. Springer-Verlag, 2005.
15. O. Kupferman and M. Vardi. Weak alternating automata are not that weak. *ACM Transactions on Computational Logic*, 2(3):408–429, 2001.
16. R. Küsters. On the decidability of cryptographic protocols with open-ended data structures. In *Proceedings of the 13th International Conference on Concurrency Theory (CONCUR'02)*, volume 2421 of *LNCS*, pages 515–530. Springer-Verlag, 2002.
17. M. Křetínský, V. Řehák, and J. Strejček. Extended process rewrite systems: Expressiveness and reachability. In *Proceedings of the 15th International Conference on Concurrency Theory (CONCUR'04)*, volume 3170 of *LNCS*, pages 355–370. Springer-Verlag, 2004.
18. R. Mayr. Process rewrite systems. *Information and Computation*, 156(1):264–286, 2000.
19. D.E. Muller, A. Saoudi, and P.E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *Proceedings of the 3rd Annual IEEE Symposium on Logic in Computer Science (LICS'88)*, pages 422–427. IEEE Computer Society Press, 1988.
20. L.C. Paulson. Mechanized proofs for a recursive authentication protocol. In *10th Computer Security Foundations Workshop*, pages 84–95. IEEE Computer Society Press, 1997.
21. M. Rusinowitch and M. Turuani. Protocol insecurity with a finite number of sessions and composed keys is NP-complete. *TCS: Theoretical Computer Science*, 299, 2003.
22. V.A. Saraswat. *Concurrent Constraint Programming*. The MIT Press, Cambridge, Massachusetts, 1993.
23. T. Truderung. Selecting theories and recursive protocols. In *Proceedings of the 16th International Conference on Concurrency Theory (CONCUR'05)*, volume 3653 of *LNCS*, pages 217–232. Springer-Verlag, 2005.
24. T.Y.C. Woo and S.S. Lam. Authentication for distributed systems. *Computer*, 25(1):39–52, 1992.

Appendix

We shall now outline a few encoding techniques to show that recursive ping-pong protocols reach beyond the pure ping-pong behaviour and allow us to encode a larger range of cryptographic protocols. In the first example we consider Woo and Lam's protocol, on which we demonstrate the basic encoding of communication channels, symmetric encryption, a limited notion of pairing and in particular we show that we can verify for authenticity properties. In the second example, we consider a protocol for key distribution by Otway and Rees on which we demonstrate that recursive definitions might be necessary to discover subtle flaws in some protocols. The third example describes modelling of the recursive authentication protocol with open-ended messages by Bull and Otway. The fourth example highlights the difference between ping-pong processes with replicative and recursive definitions. We finish this appendix by providing a short discussion.

A Authentication Protocol by Woo and Lam

Woo and Lam presented in [24, pages 42–43] an authentication protocol based on symmetric-key cryptography. A participant C wants to communicate with participants A and B and make sure that the communication is authentic. The protocol uses a trusted server S which shares with A, B and C secret symmetric keys k_{AS}, k_{BS}, and k_{CS}, respectively. The communication protocol to ensure that C talks exclusively with A is as follows (the protocol for authentication between B and C is according to the same general scheme):

1. $A \rightarrow C: \ A$
2. $C \rightarrow A: \ N_A$
3. $A \rightarrow C: \ \{N_A\}_{k_{AS}}$
4. $C \rightarrow S: \ \{A, \{N_A\}_{k_{AS}}\}_{k_{CS}}$
5. $S \rightarrow C: \ \{N_A\}_{k_{CS}}$

Here N_A is a nonce and $\{m\}_k$ denotes that a message m is encrypted by a key k. In the first message, A claims his identity and C responds by returning a nonce challenge N_A; A then encrypts this nonce by the key k_{AS} and passes the message back to C; the participant C adds A's identity to the received message and forwards it, encrypted by the key k_{CS}, to the server S for verification; S first decrypts the message to obtain A and $\{N_A\}_{k_{AS}}$ and if the key k_{AS} is indeed the shared key with A, it returns the nonce N_A encrypted by k_{CS} back to C; finally C decrypts the message and in case that the nonce he received is the same as the challenge sent to A, then C is convinced that it was A who responded to the challenge N_A. The participant C is running the same protocol also with B but it uses a different nonce N_B to authenticate B.

We shall now present how the protocol can be modelled using our recursive calculus for ping-pong protocols. The set of keys we shall be using is defined by

$$\mathcal{K} \stackrel{\text{def}}{=} \{c_{X\to Y} \mid X, Y \in \{A, B, C, S\}\} \cup \{k_{AS}, k_{BS}, k_{CS}\} \cup$$
$$\{A, B, C, S, N_A, N_B, \mathit{start}, \mathit{end}\}.$$

The intuitive meaning is that whenever $c_{X\to Y}$ is appended as the outer-most key of some message, it signals the claimed sender X and receiver Y of the message, i.e., it represents a communication channel from X to Y; k_{AS}, k_{BS}, k_{CS} are the symmetric keys shared between the server and the corresponding agent; A, B, C, S are the names of the participants; N_A and N_B are two different nonces. The special keys *start* and *end* are added only for technical reasons to initiate the run of the protocol and to signal the end of an execution of a particular agent, respectively.

We shall start by describing the defining equations for the participant A.

$$A \stackrel{\text{def}}{=} [?\mathit{start} \rhd .!c_{A\to C}\, A \rhd].A_1 \tag{3}$$
$$A_1 \stackrel{\text{def}}{=} [?c_{C\to A} \rhd .!c_{A\to C}\, k_{AS} \rhd].A_{end} \tag{4}$$

The meaning is that A can read the message *start* (which shall always be present in the initial pool of messages) and output on the channel from A to C that the agent A wants to start the authentication protocol with the agent C. After that the agent changes into A_1. Once A_1 receives a nonce challenge from C, he returns it encrypted by his symmetric key k_{AS} (shared only with the server S) and finishes the protocol. Symmetric definitions are added also for the participant B.

$$B \stackrel{\text{def}}{=} [?\mathit{start} \rhd .!c_{B\to C}\, B \rhd].B_1 \tag{5}$$
$$B_1 \stackrel{\text{def}}{=} [?c_{C\to B} \rhd .!c_{B\to C}\, k_{BS} \rhd].B_{end} \tag{6}$$

Since the agent C has to be able to handle both authentication with A and B, it consists of two (parallel) parts C^A and and C^B. We shall first define C^A.

$$C^A \stackrel{\text{def}}{=} [?c_{A\to C}\, A \rhd .!c_{C\to A}\, N_A \rhd].C_1^A \tag{7}$$
$$C_1^A \stackrel{\text{def}}{=} [?c_{A\to C} \rhd .!c_{C\to S}\, k_{CS}\, A \rhd].C_2^A \tag{8}$$
$$C_2^A \stackrel{\text{def}}{=} [?c_{S\to C}\, k_{CS}\, N_A \rhd .!\mathit{end} \rhd].C_{end}^A \tag{9}$$

On receiving a communication request from A, the agent C^A responds by sending a nonce N_A back to A and changes into C_1^A. Next, the message received on the channel from A to C is, together with the identity of the agent A, encrypted by the key k_{CS} and sent to the server S for verification. The state of C_1^A changes to C_2^A. Now if C_2^A receives a message encrypted by the key shared with the server, it compares it with the initially issued nonce N_A and in case of a match, it enters the state C_{end}^A meaning that C is now sure that the communication happened exclusively with the participant A.

Remark 4. Our ping-pong formalism enables to encode a restricted notion of pairing, like the one in step 4. of the Woo and Lam's protocol. More generally, a message of the form $\{A_1, \ldots, A_n, \{m\}_{k_2}\}_{k_1}$ where $A_1, \ldots, A_n$ are single keys which are by default contained in the initial pool (e.g. the names of the participants in the protocol), and $\{m\}_{k_2}$ is an arbitrary message m encrypted by the key k_2, can be encoded as $k_1 A_1\, A_2 \ldots A_n\, k_2\, m$. Assuming now that the intruder knows the key k_1, he is able to analyze the message and in particular to do a projection on any component of the encrypted tuple. In the case of Woo and Lam's protocol, we encode the message that C sends to S in step 4. by $k_{CS}\, A\, k_{AS}\, N_A$.

The agent C^B for handling the authentication with B is completely symmetric to C^A.

$$C^B \stackrel{\text{def}}{=} [?c_{B\to C}\, B \rhd .!c_{C\to B}\, N_B \rhd].C_1^B \tag{10}$$

$$C_1^B \stackrel{\text{def}}{=} [?c_{B\to C} \rhd .!c_{C\to S}\, k_{CS}\, B \rhd].C_2^B \tag{11}$$

$$C_2^B \stackrel{\text{def}}{=} [?c_{S\to C}\, k_{CS}\, N_B \rhd .!end \rhd].C_{end}^B \tag{12}$$

Under the agent C we shall understand the parallel composition $C^A \parallel C^B$ (which could be, of course, defined also by a single agent using the cross product construction of Remark 1).

Finally, the behaviour of the server S can be described by the following recursive definition.

$$S \stackrel{\text{def}}{=} [?c_{C\to S}\, k_{CS}\, A\, k_{AS} \rhd .!c_{S\to C}\, k_{CS} \rhd].S + \tag{13}$$

$$[?c_{C\to S}\, k_{CS}\, B\, k_{BS} \rhd .!c_{S\to C}\, k_{CS} \rhd].S \tag{14}$$

The intuition is that the server accepts requests only from C by verifying that the message is encrypted by the shared key k_{CS} and checks whether the claimed identity (either A or B) really corresponds with the shared key (either k_{AS} or k_{BS}) by which the nonce is encrypted. If this is the case, S sends to C the rest of the message (in this case a nonce) encrypted by the key k_{CS} and is ready to accept another request.

It was showed in [2] that the protocol is flawed. In case that a symmetric key of one of the participants, let us say B, is compromised then the protocol does not guarantee authenticity for A (even if the key k_{AS} is assumed to be secure). Indeed, also in our modelling of the protocol, we can discover the flaw by showing that we can reach a configuration where the first component of C is in the state C_{end}^A while A did not participate in any communication with C (i.e. it is still in its initial state A). This is demonstrated by the following sequence of reductions.

We start from the initial configuration (P_0, T_0) where $P_0 = A \parallel B \parallel C \parallel S$ and $T_0 = \{c_{X\to Y} \mid X, Y \in \{A, B, C, S\}\} \cup \{A, B, C, S, start\} \cup \{k_{BS}\}$. (The parallel composition P_0 can in fact be expressed as a single process without any parallel composition due to Remark 1.) Note that we assume that the shared key k_{BS} of B is compromised and hence contained in the initial knowledge T_0. We shall now describe a possible attack on the protocol as published in [2].

First, the (implicit) intruder claims to be A and sends a request to C, on which C answers by issuing a nonce challenge N_A. This corresponds to the following transition in our protocol

$$\frac{\text{by } c_{A\to C}\, A \in \mathcal{S}(\mathcal{A}(T_0)) \text{ and (7)}}{(A \parallel B \parallel C \parallel S,\ T_0) \longrightarrow (A \parallel B \parallel C_1^A \parallel C^B \parallel S,\ T_1)}$$

where $T_1 = T_0 \cup \{c_{C\to A}\, N_A\}$.

Next, the intruder claims the identity of B and initiates the protocol acting as B. Agent C answers by providing a nonce challenge N_B. The corresponding transition is

$$\frac{\text{by } c_{B\to C}\, B \in \mathcal{S}(\mathcal{A}(T_1)) \text{ and (10)}}{(A \parallel B \parallel C_1^A \parallel C^B \parallel S,\ T_1) \longrightarrow (A \parallel B \parallel C_1^A \parallel C_1^B \parallel S,\ T_2)}$$

where $T_2 = T_1 \cup \{c_{C\to B}\, N_B\}$.

The intruder now synthesizes $\{N_A\}_{k_{BS}}$ because he knows the key k_{BS} and forwards this message twice to C. First time acting as A and second time as B. The agent C reacts on these two messages by asking the server for verification. This is described by the following two rules

$$\frac{\text{by } c_{A\to C}\, k_{BS}\, N_A \in \mathcal{S}(\mathcal{A}(T_2)) \text{ and (8)}}{(A \parallel B \parallel C_1^A \parallel C_1^B \parallel S,\ T_2) \longrightarrow (A \parallel B \parallel C_2^A \parallel C_1^B \parallel S,\ T_3)}$$

where $T_3 = T_2 \cup \{c_{C\to S}\, k_{CS}\, A\, k_{BS}\, N_A\}$, and

$$\frac{\text{by } c_{B\to C}\, k_{BS}\, N_A \in \mathcal{S}(\mathcal{A}(T_2)) \text{ and (11)}}{(A \parallel B \parallel C_2^A \parallel C_1^B \parallel S,\ T_3) \longrightarrow (A \parallel B \parallel C_2^A \parallel C_2^B \parallel S,\ T_4)}$$

where $T_4 = T_3 \cup \{c_{C\to S}\, k_{CS}\, B\, k_{BS}\, N_A\}$.

The server cannot, of course, confirm the first message, however, it can confirm the second one and return $\{N_A\}_{k_{CS}}$ to the agent C by the following rule

$$\frac{\text{by } c_{C\to S}\, k_{CS}\, B\, k_{BS}\, N_A \in \mathcal{S}(\mathcal{A}(T_2)) \text{ and (14)}}{(A \parallel B \parallel C_2^A \parallel C_2^B \parallel S,\ T_4) \longrightarrow (A \parallel B \parallel C_2^A \parallel C_2^B \parallel S,\ T_5)}$$

where $T_5 = T_4 \cup \{c_{S\to C}\, k_{CS}\, N_A\}$.

Finally, the agent C receives the message from the server and because it has the expected format, C wrongly believes that he was communicating with A and enters the state C_{end}^A (while A is still in its initial state and did not at all participate in the communication) by the rule

$$\frac{\text{by } c_{S\to C}\, k_{CS}\, N_A \in \mathcal{S}(\mathcal{A}(T_2)) \text{ and (9)}}{(A \parallel B \parallel C_2^A \parallel C_2^B \parallel S,\ T_5) \longrightarrow (A \parallel B \parallel C_{end}^A \parallel C_2^B \parallel S,\ T_6)}$$

where $T_6 = T_5 \cup \{end\}$.

The main decidability result of our paper moreover enables to find such a transition sequence algorithmically.

In [2] the authors suggested a fix by adding the agent's identity to the last message sent from the server to C so that it looks as follows.

5'. $S \to C : \ \{A, N_A\}_{k_{CS}}$

This can be also modelled in our formalism by changing the definition of the server S into

$$S \stackrel{\text{def}}{=} [?c_{C\to S}\, k_{CS}\, A\, k_{AS} \rhd .!c_{S\to C}\, k_{CS}\, A \rhd].S + \\ [?c_{C\to S}\, k_{CS}\, B\, k_{BS} \rhd .!c_{S\to C}\, k_{CS}\, B \rhd].S$$

and the equations (9) and (12) into

$$C_2^A \stackrel{\text{def}}{=} [!c_{S\to C}\, k_{CS}\, A\, N_A \rhd .!end \rhd].C_{end}^A$$

and

$$C_2^B \stackrel{\text{def}}{=} [!c_{S\to C}\, k_{CS}\, B\, N_B \rhd .!end \rhd].C_{end}^A \quad .$$

Indeed, the previously described attack is not possible any more and one can show this in an algorithmic way by checking that it is impossible to reach a configuration where the first component is still in its initial state A (or A_1), while the third component is in C_{end}^A.

B Key Distribution Protocol by Otway and Rees

Let us have a look at further modelling possibilities of our recursive ping-pong formalism. In this example we aim to demonstrate the convenient use of recursive definitions. We consider a protocol by Otway and Rees for the distribution of shared symmetric keys by a trusted server, or rather its secure optimization by Abadi and Needham [2]. The protocol by which A and B want to obtain a joint symmetric key generated by a trusted server S is as follows.

1. $A \to B : \ A, B, N_A$
2. $B \to S : \ A, B, N_A, N_B$
3. $S \to B : \ \{A, B, k_{AB}, N_A\}_{k_{AS}}, \ \{A, B, k_{AB}, N_B\}_{k_{BS}}$
4. $B \to A : \{A, B, k_{AB}, N_A\}_{k_{AS}}$

We consider the following variant where messages sent in steps 2. and 3. are split into two parts dealing with A and B separately, but with some additional encryptions which could allow us to remove the identity of agents from the encrypted messages (indeed, in the original protocol with the flaw the identity of the agents was not connected enough to the corresponding nonces).

1. $A \to B:\ A, B, \{N_A\}_{k_{AS}}$
2. $B \to S:\ A, B, \{N_A\}_{k_{AS}}$
3. $S \to B:\ \{A, \{k_{AB}, N_A\}_{k_{AS}}\}_{k_{BS}}$
4. $B \to A:\ \{k_{AB}, N_A\}_{k_{AS}}$
5. $B \to S:\ \{N_B\}_{k_{BS}}$
6. $S \to B:\ \{k_{AB}, N_B\}_{k_{BS}}$

We can model this protocol using similar tricks that we used in Woo and Lam's protocol. In particular, the definition of the server S might look as follows (where $\mathcal{N}$ is the set of names of protocol participants and the notation is as in the previous example).

$$S \stackrel{\text{def}}{=} \sum_{X,Y \in \mathcal{N}} [?c_{Y \to S}\, X\, Y\, k_{XS}.!c_{S \to Y}\, k_{YS}\, X\, k_{XS}\, k_{XY}]. \tag{15}$$

$$[?c_{Y \to S} k_{YS}.!c_{S \to Y}\, k_{YS}\, k_{XY}].S \tag{16}$$

Note that in our definition of S we use (for the notational convenience) sequencing of two prefixes which can, however, easily be encoded into the standard format. The server behaves so that on receiving a message from Y of the form $c_{Y \to S}\, X\, Y\, k_{XS}\, N_X$ it returns $c_{S \to Y}\, k_{YS}\, X\, k_{XS}\, k_{XY}\, N_X$ and on the successive request from Y of the form $c_{Y \to S}\, k_{YS}\, N_Y$ it answers by sending $c_{S \to Y}\, k_{YS}\, k_{XY}\, N_Y$. The other participants A and B of the protocol can be defined in a similar way as before and it is assumed that there is one more agent's name C with a compromised shared key k_{CS}.

The protocol is flawed. An intruder can make the participants A and B believe that they share a secret symmetric key k_{AB}, while in fact they both share a key together with the intruder (A shares with C the key k_{AC} and B shares with C the key k_{BC}).

The flaw can be discovered only if the server is defined recursively — even though the participants A and B make only one run of the protocol (including two requests on the server), the server must be asked five times in total to make the flaw appear. A possible attack can look as follows.

1. $A \to B:\ A, B, \{N_A\}_{k_{AS}}$
2. $B \to S:\ A, B, \{N_A\}_{k_{AS}}$
3. $S \to B:\ \{A, \{k_{AB}, N_A\}_{k_{AS}}\}_{k_{BS}}$
4. $B \to A:\ \{k_{AB}, N_A\}_{k_{AS}}$ (intercepted by the intruder)
5. $B \to S:\ \{N_B\}_{k_{BS}}$
6. $S \to B:\ \{k_{AB}, N_B\}_{k_{BS}}$ (intercepted by the intruder)
7. $C \to S:\ A, C, \{N_A\}_{k_{AS}}$ (intruder acts as C)
8. $S \to C:\ \{A, \{k_{AC}, N_A\}_{k_{AS}}\}_{k_{CS}}$
9. $B' \to A:\ \{k_{AC}, N_A\}_{k_{AS}}$ (intruder knows k_{CS} and acts as B)
10. $C \to S:\ \{N_C\}_{k_{CS}}$ (intruder acts as C and sends any nonce)
11. $S \to C:\ \{k_{AC}, N_C\}_{k_{CS}}$ (intercepted by the intruder)
12. $C \to S:\ B, C, \{N_B\}_{k_{BS}}$ (intruder acts as C)

13. $S \to C:\ \{B, \{k_{BC}, N_B\}_{k_{BS}}\}_{k_{CS}}$ (intercepted by the intruder)
14. $S' \to B:\ \{k_{BC}, N_B\}_{k_{BS}}$ (intruder knows k_{CS} and acts as S)

Now both A and be B followed the protocol and could not see any problem but the key they received is known to the intruder who can now listen to any private communication between A and B in the following way: A sends a message $\{m\}_{k_{AC}}$ to B; it is intercepted by the intruder, decrypted, changed into $\{m\}_{k_{BS}}$ and forwarded to B; similarly in the opposite direction.

Observe that a non-recursive definition of the server is not enough to discover the flaw in the protocol (assuming that we do not know in advance how many times the server has to be able to answer requests — in our case it was five times but it could be more times in other protocols).

C Recursive Authentication Protocol by Bull and Otway

Bull and Otway proposed an extension to the Otway-Rees authentication protocol, called the Recursive Authentication (RA) protocol (see [20]), to establish session keys between an arbitrary number of principals in one protocol run.

The RA protocol operates over an arbitrarily long chain of protocol agents $A, B, C \ldots$ terminating with a key-server S that shares long-terms keys with the principals. Every agent in the chain appends a request for a session key to the message received by the agent on his left, and sends the new message to the agent to the right. The recursive definition is terminated when the message is sent to the server. At that point the server inspects the (potentially unbounded) structure of the message and distributes fresh session keys to the principals, encrypted with their long-term keys. Notice that the request message is an open-ended structure (it has an unbounded number of fields). The server must be defined in an inductive way. To formalize the protocol, we need some further notation.

Let $h(m)$ be the hash of message m, and $h_k(m)$ be the messages $h(km)m$, where km is the message obtained from k and m by concatenation, and $h(km)m$ is the concatenation of $h(km)$ and m. If K_a is a long-term key shared between A and S, $hash_{K_a}(m)$ is used by the server to identify principal A. If principal A wants to establish a session with B, he/she sends a request message to B:

$$A \longrightarrow B : hash_{K_a}(A, B, N_a, -)$$

where K_a is a long-term key shared between A and S, N_a is a fresh nonce, and $-$ indicates that this message started the protocol run. Let M be the message $hash_{K_a}(A, B, N_a, -)$. Differently from Otway-Rees, B can attach his request for a session key with C to the message M received from A and send the resulting message to C, i.e.,

$$B \longrightarrow C : hash_{K_b}(B, C, N_b, M)\ .$$

This step can be repeated an arbitrary number of times. This phase of the protocol is terminated when a principal contacts the server S. For instance, if

C is the last principal in the chain, then he sends the message, say M', received from B to S.

$$C \longrightarrow S : hash_{K_c}(C, S, N_c, M')$$

The server now has to process the complete message

$$hash_{K_c}(C, S, N_c, hash_{K_b}(B, C, N_b, hash_{K_a}(A, B, N_a, -))) \ .$$

Namely, by looking at the two outer hashes the server generates two fresh keys K_{cs} (which is redundant but makes the protocol easier to write) and K_{bc} to be used as session keys between C and S and between B and C, respectively. These keys are encrypted using the long-term keys of principal C, i.e., S prepares the certificates $enc_{K_c}(K_{cs}, S, N_c)$ and $enc_{K_c}(K_{bc}, S, N_c)$. After this step, S proceeds with the remaining part of the message performing similar operations for B (and, in the general case, for all other agents of the chain). Finally, to process the message $hash_{K_a}(A, B, N_a, -)$, S only needs to prepare the message with session key K_{ab} (the same sent to B) to agent A, and then terminate the operation. After this second phase, S sends all certificates to C.

$$\begin{aligned} S \longrightarrow C : & enc_{K_c}(K_{cs}, S, N_c) enc_{K_c}(K_{bc}, S, N_c) enc_{K_b}(K_{bc}, S, N_b) \\ & enc_{K_b}(K_{ab}, S, N_b) enc_{K_a}(K_{ab}, S, N_a) \end{aligned}$$

C accepts the first two certificates, and forwards the rest to B, and so on for all the principals in the chain, i.e.,

$$\begin{aligned} & C \longrightarrow B : enc_{K_b}(K_{bc}, S, N_b) enc_{K_b}(K_{ab}, S, N_b) enc_{K_a}(K_{ab}, S, N_a) \\ & B \longrightarrow A : enc_{K_a}(K_{ab}, S, N_a) \ . \end{aligned}$$

We model the RA protocol by considering only the hashes of the messages exchanged in the protocol, i.e., instead of $hash_{K_a}(m)$ (defined as $hash(K_a m)m$) we consider the hash $hash(K_a m)$ without the plain text copy m. The nonces are sent in clear together with the hash, too.

The set of keys $\mathcal{K}$ used in our model is obtained as the union of the following finite sets:

- $\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$ to model principal's identities where $p_m = S$ (the server),
- $\mathcal{N} = \{n_1, \ldots, n_m\}$ to model nonces,
- $\mathcal{L} = \{k_1, \ldots, k_m\}$ to model long-term keys,
- $\mathcal{S} = \{s_{ij} | 1 \leq i, j \leq m\}$ to model session keys,
- $\mathcal{C} = \{c_{i \to j} \mid 1 \leq i, j \leq m, \ i \neq j\}$ to model communication channels between principals,
- $\mathcal{O} = \{end, srv, h, start\}$ contains the keys *end* to model the terminator of the protocol messages, *srv* to hide the internal steps of the server, h to model hashes, and *start* to start the protocol.

The initial knowledge of the intruder contains $\mathcal{C} \cup \mathcal{P}$. If P_i is a malicious agent, then we can include k_i, n_i into the initial intruder's knowledge. The intruder

has the additional capability of building the hash of any message in the current knowledge by using the following recursive definition.

$$HASH \stackrel{\text{def}}{=} [?\,\triangleright\,.!h\,\triangleright].HASH$$

In our model we assume that at most m agents can participate in the protocol. Thus, we put a bound on the number of receive-send messages executed by the principals in one session of the protocol. The intruder can, however, contribute to building of messages of potentially unbounded size. Hence the server has to be able to handle messages of arbitrary size.

To model the behaviour of the principal P_i, we use the following set of process definitions.
For any $i, j, l \in I = \{1, \ldots, m\}$ and $a \in \mathcal{S}$:

$$P(i) \stackrel{\text{def}}{=} \begin{cases} \sum\limits_{j \in I \setminus \{i\}} [?start\ \triangleright\ .\ !c_{i\to j}\ h\ k_i\ p_i\ p_j\ n_i\ end\ \triangleright].KEY(i,j) \\ + \\ \sum\limits_{j \neq l \in I \setminus \{i\}} \sum\limits_{a \in \mathcal{L}} [?c_{j\to i}\ \triangleright\ .\ !c_{i\to l}\ h\ k_i\ p_i\ p_l\ n_i\ \triangleright].KEY(i,j,l) \end{cases}$$

$$KEY(i,j) \stackrel{\text{def}}{=} \sum_{a \in \mathcal{S}} [?k_i\ a\ p_m\ n_i\ \triangleright\ .!\triangleright].EndP(i,j,a)$$

$$KEY(i,j,l) \stackrel{\text{def}}{=} \sum_{a \in \mathcal{S}} [?k_i\ a\ p_m\ n_i\ \triangleright\ .!\triangleright].KEY(i,j,l,a)$$

$$KEY(i,j,l,a) \stackrel{\text{def}}{=} \sum_{b \in \mathcal{S}} [?k_i\ b\ p_m\ n_i\ \triangleright\ .!\triangleright].HALT(i,j,l,a,b)$$

To model the server, we use the following set of recursive definition for all $a \in \mathcal{S}$.

$$SRV \stackrel{\text{def}}{=} \sum_{i \in I} \sum_{a \in \mathcal{N}} [?c_{i\to m}\ h\ k_i\ p_i\ p_m\ b\ \triangleright\ .\ !k_i\ s_{im}\ p_m\ b\ srv\ \triangleright].SRV(a)$$

$$SRV(a) \stackrel{\text{def}}{=} \sum_{i,j \in I} \sum_{b \in \mathcal{N}} [?srv\ h\ k_i\ p_i\ p_j\ b\ \triangleright\ .\ !k_j\ s_{ij}\ p_m\ a\ k_i\ s_{ij}\ p_m\ b\ srv\ \triangleright].SRV(b)$$

$$SRV(a) \stackrel{\text{def}}{=} [?srv\ end\ \triangleright\ .\ !end\triangleright].HALT_S$$

If the intruder learns a secret key $s \in \mathcal{S}$ during the execution of the protocol, then he/she can forge a message using s as encryption key. Thus, in order to verify that secret keys are never exposed to the intruder we can add an observer process defined as follows

$$OBS \stackrel{\text{def}}{=} \sum_{s \in \mathcal{S}} [?s\ \triangleright\ .!\triangleright].ERR\ .$$

Thus, the initial configuration of the system is defined by

$$HASH \parallel P(1) \parallel P(2) \parallel \ \ldots\ \parallel P(m) \parallel SRV \parallel OBS$$

and by the initial message *start*. Since the reachability of the control location ERR of the process OBS identifies a violation of the secrecy of one of the keys in $\mathcal{S}$, our algorithm can be used to verify secrecy for our model of the protocol.

Other approaches to verification of the recursive authentication protocol by using transducers can be found in [16, 23].

D Comparison of Recursive and Replicative Definitions

The aim of this section is to demonstrate that recursive ping-pong protocols can model the execution of a protocol more faithfully than the replicative ones, like e.g. the calculus of Amadio, Lugiez and Vanackère [5]. For this purpose, consider the following simple protocol.

$A \stackrel{\text{def}}{=} [?init \rhd .!k_A\ init \rhd].A1$
$A1 \stackrel{\text{def}}{=} [?k_A \rhd .!k_A\ ok \rhd].A2$

$B \stackrel{\text{def}}{=} [?init \rhd .!k_B\ init \rhd].B1$
$B1 \stackrel{\text{def}}{=} [?k_B \rhd .!k_B\ ok \rhd].B2$

$C \stackrel{\text{def}}{=} [?k_A \rhd .!k_A\ secret \rhd].C1 + [?k_B \rhd .!k_B\ secret \rhd].C1$
$C1 \stackrel{\text{def}}{=} [?k_A\ ok\ secret \rhd .!\rhd].CA + [?k_B\ ok\ secret \rhd .!\rhd].CB$

The idea is that after the agent A or B initiates the protocol, the agent C can communicate with the initiator and send a secret message *secret* encrypted by the other agent's private (symmetric) key (k_A or k_B). Once C started the communication with one of the agents, it is expecting an *ok* confirmation from the same agent. (Note that C never again executes the first step once it already sent the key *secret*). Normally, the protocol is supposed to be run from the initial configuration

$$(A \parallel B \parallel C,\ \{init, ok\})$$

and it is easy to see that if C reaches the state CA, then only A knows the secret. Moreover, even if the agent B is compromised, i.e., we run the protocol from the initial configuration

$$(A \parallel B \parallel C,\ \{init, ok, k_B\})$$

where the attacker knows B's private key, once the agent C enters the state CA, there is a guarantee that the message *secret* cannot be known to the attacker. The reason is that after C finishes the first step of the protocol, C will never ever again forward the secret to anybody else.

In [5] the authors suggest a possible encoding of sequences of actions in the replicative variant of the calculus. This, to a certain extend, enables to model e.g. our example protocol in their calculus. The main difference is, however, that their approach provides an over-approximation of the real protocol behaviour

(and would return a false positive in the example protocol with the compromised key k_B). The reason is, that is not possible to disable performing repeatedly the first step of C in the replicative calculus. The modelling of the protocol in the recursive ping-pong formalism is more precise, as it corresponds exactly to the intuitive semantics. On the other hand, the price we pay for it is a worse complexity upper bound.

E Final Remarks

Let us finish our examples with a few observations. First of all, as already noted in [5], ping-pong protocols have the capability to model public key cryptography. This can be done by a simple syntactic restriction so that a private key of an agent A can only appear in agent's own input prefix but any other agent (including A) can use the key in any of the output prefixes. Originally, ping-pong protocols were studied as a communication scheme between two memory-less agents [10, 11]. In [5] the formalism was extended to handle any (finite) number of participants, moreover with the possibility of a replicated behaviour. Nevertheless, the agents in [5] were still memory-less and once a certain prefix became executable, it remained so also anytime in the future. Checking for authenticity was not possible in this setting.

In our recursive calculus (which subsumes both of the above mentioned ones), the agents can moreover remember a finite amount of information (they behave essentially as finite-state machines) plus they have the possibility of changing behaviour. For example once a certain key gets compromised, an agent can switch to another key and reject any future communication using the compromised key. We can also use a (fixed) finite number of nonces; allowing an unbounded number of nonces would cause undecidability of the reachability problem. Finally, allowing the agents to remember their current states provides the possibility to verify not only for security properties but also for authenticity. Moreover, the formalism enables to start from an infinite (but regular) pool of initially known messages, which could be used for parameterized reasoning about protocols.

Finally, we showed that our calculus can have further applications in modelling of protocols which are defined recursively and the length of exchanged messages is potentially unbounded, like for example the recursive authentication protocol of Bull and Otway.

Recent BRICS Report Series Publications

RS-06-14 **Giorgio Delzanno, Javier Esparza, and Jiří Srba.** ***Monotonic Set-Extended Prefix Rewriting and Verification of Recursive Ping-Pong Protocols*****. July 2006. 31 pp. To appear in ATVA '06.**

RS-06-13 **Jiří Srba.** ***Visibly Pushdown Automata: From Language Equivalence to Simulation and Bisimulation*****. July 2006. 21 pp. To appear in CSL '06.**

RS-06-12 **Kristian Støvring.** ***Higher-Order Beta Matching with Solutions in Long Beta-Eta Normal Form*****. June 2006. 13 pp. To appear in** ***Nordic Journal of Computing*****, 2006.**

RS-06-11 **Kim G. Larsen, Ulrik Nyman, and Andrzej Wasowski.** ***An Interface Theory for Input/Output Automata*****. June 2006. 40 pp. Appears in Misra, Nipkow and Sekerinski, editors,** ***Formal Methods: 14th International Symposium*****, FM '06 Proceedings, LNCS 4085, 2006, pages 82–97.**

RS-06-10 **Christian Kirkegaard and Anders Møller.** ***Static Analysis for Java Servlets and JSP*****. June 2006. 23 pp. Full version of paper presented at SAS '06.**

RS-06-9 **Claus Brabrand, Robert Giegerich, and Anders Møller.** ***Analyzing Ambiguity of Context-Free Grammars*****. April 2006. 19 pp.**

RS-06-8 **Christian Kirkegaard and Anders Møller.** ***Static Analysis for Java Servlets and JSP*****. April 2006. 22 pp.**

RS-06-7 **Petr Jančar and Jiří Srba.** ***Undecidability Results for Bisimilarity on Prefix Rewrite Systems*****. April 2006. 20 pp. Presented at** ***FoSSaCS 2006*****, LNCS 3921:277–291.**

RS-06-6 **Luca Aceto, Willem Jan Fokkink, Anna Ingólfsdóttir, and Bas Luttik.** ***A Finite Equational Base for CCS with Left Merge and Communication Merge*****. March 2006. 22 pp.**

RS-06-5 **Kristian Støvring.** ***Extending the Extensional Lambda Calculus with Surjective Pairing is Conservative*****. March 2006. 18 pp. To appear in** ***Logical Methods in Computer Science*****. Supersedes RS-05-35.**