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 [4, 12, 13] 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 [10]. 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 [9] 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 [10], 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 [4] Amadio, Lugiez and Vanackère extended the result of [9] 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 [3, 8].

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 [10, 9] 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. 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.

A process algebra for recursive ping-pong protocols was introduced in [12, 13] where it was proved that the resulting model (without any notion of an attacker) is Turing powerful.

Novel contribution. The results from [12, 13] 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.

Our main decidability result is consistent with the results on tail-recursive cryptographic protocols from [3]. Indeed the necessary (but not sufficient) conditions defined in [3] (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 [17, 14], or weak Process Rewrite Systems (wPRSs) [16]. 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 [19].

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 [5], 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 [19] can be naturally encoded into our MSP formalism.

Note: A full version of the paper, including complete proofs and examples of the modelling power of ping-pong protocols, is available as a BRICS technical report at `http://www.brics.dk/publications/` .

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., [6, 11]). *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.

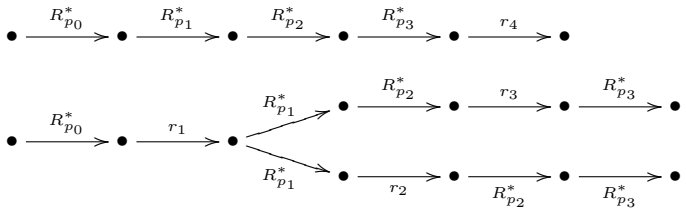

Fig. 1. A π-scheme for $\pi = p_0 r_1 \dots r_4 p_4$

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.*

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.*

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 recognizing 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* $pv \longrightarrow qw$*.*

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* $(|Q| + m + |\Gamma|)^{O(c)} \cdot a$*.*

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 [4, 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 [3] 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.

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 $\alpha = [?v \triangleright .!w\triangleright]$. 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 mentioned 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 $T^\bot \subseteq T'^\bot$.*

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* $T \subseteq \mathcal{S}(\mathcal{A}(T'))$.

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.

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 $\mathcal{K} = \{C_1,\ldots,C_k,\perp\}$. 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.

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

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, $1 \le i \le n$, 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 [19]. 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.

$$A \ ::= stop \ \mid \ \Sigma_{i=1}^{k}\, Act_i$$
$$Act ::= ask(v \cdot x) \to p \ \mid \ ask(v \cdot x) \to tell(w \cdot x) \to p$$

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., $P = A_1 \mid\mid \ldots \mid\mid A_\ell$. We assume that $\mid\mid$ 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 2. The seemingly nonstandard action $ask(v \cdot x) \rightarrow tell(w \cdot x) \rightarrow p$ can be in full ccp encoded as $ask(v \cdot x) \rightarrow \exists n.\big(tell(w \cdot x \;\&\; tok(n)) \mid\mid ask(tok(n)) \rightarrow p\big)$ 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 [12, 13]. 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 [7, 2].

We have used our results to prove (in the full version of the paper, available as a BRICS technical report) 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 [18] 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 [15]. 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 [4] 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.

Since our technique does not over-approximate the semantics, it is strictly more powerful than that of [4], at the price of a higher complexity (the algorithm of [4] runs in polynomial time), and it is incomparable with the techniques of [18, 15]. 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.

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.

Acknowledgments. 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. P.A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996.
3. 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.
4. 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.
5. 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.
6. J.R. Büchi. Regular canonical systems. *Arch. Math. Logik u. Grundlagenforschung*, 6:91–111, 1964.
7. 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.
8. 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.
9. D. Dolev, S. Even, and R.M. Karp. On the security of ping-pong protocols. *Information and Control*, 55(1–3):57–68, 1982.
10. D. Dolev and A.C. Yao. On the security of public key protocols. *Transactions on Information Theory*, IT-29(2):198–208, 1983.
11. 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.
12. 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.
13. 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.
14. O. Kupferman and M. Vardi. Weak alternating automata are not that weak. *ACM Transactions on Computational Logic*, 2(3):408–429, 2001.
15. 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.
16. 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.
17. 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.
18. 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.
19. V.A. Saraswat. *Concurrent Constraint Programming.* The MIT Press, Cambridge, Massachusetts, 1993.