On name generation and set-based analysis in Dolev-Yao model

Roberto M. Amadio *
Université de Provence, Marseille

Witold Charatonik †
MPI, Saarbrücken

January 15, 2002

Abstract

We study the control reachability problem in the Dolev-Yao model of cryptographic protocols when principals are represented by tail recursive processes with generated names. We propose a conservative approximation of the problem by reduction to a non-standard *collapsed* operational semantics and we introduce checkable syntactic conditions entailing the equivalence of the standard and the collapsed semantics. Then we introduce a conservative and decidable *set-based* analysis of the collapsed operational semantics and we characterize a situation where the analysis is exact. Finally, we describe how our framework can be used to specify secrecy and freshness properties of cryptographic protocols.
Keywords: cryptographic protocols, name generation, verification, set constraints.

1 Introduction

We study the control reachability problem for cryptographic protocols in Dolev-Yao model [DY83] which is nowadays a widely used model abstracting the behaviour of cryptographic functions.

Most cryptographic protocols are programs of finite length without loops and then the control reachability problem can be solved in NPTIME [ALV01, RT01]. However, these finite programs usually can be executed any number of times in different sessions. In every session a participant may generate fresh *names* representing, *e.g.*, nonces or keys. A number of attacks are then possible relying on messages exchanged in previous or parallel sessions. Unfortunately, introducing recursive behaviours in cryptographic protocols leads quickly to an undecidable verification problem.

It has been observed by Durgin *et al.* [DLMS99] that name generation leads to undecidability of control reachability even when the height of the messages considered is *bounded*. When name generation is *not* allowed then the control reachability problem is still undecidable in general, and decidable in certain particular cases allowing a limited use of the pairing construct (also known as *ping-pong* protocols; see [DEK82]). To complete this picture, we show in appendix B that the control reachability problem is undecidable *without pairs* and *with bounded height* messages when name generation is allowed.

It then appears that in order to obtain a decidable class of protocols we have to further restrict the use of name generation. The approach we explore in this paper is to bound the number of *parallel* sessions generating new names. This entails that there is a bound on the number of *fresh* names 'used' at any time by the principals.

Technically, we will concentrate on *tail-recursive* processes including an operation of *name generation*. In the standard operational semantics, whenever we execute a name generation instruction νc, we associate to c a fresh constant. Thus, during the execution the name c may get assigned constants from a countable set $C_0, C_1, C_2, \ldots$ In this paper, we introduce a non-standard

*Laboratoire d'Informatique Fondamentale, CMI, 39 rue Joliot-Curie, F-13453, Marseille, France. amadio@cmi.univ-mrs.fr. Partly supported by ACI VERNAM and IST PROFUNDIS.
†Max Plack Institut für Informatik, Im Stadtwald D-66123 Saarbrücken, Germany. witold@mpi-sb.mpg.de.

collapsed operational semantics for name generation whose intuition is as follows: when a principal starts a new *session* a name generated in the session gets assigned a *new* constant while names generated in previous sessions are collapsed to an *old* constant. Thus a name generator νc operates over the finite set of constants $\{C^{old}, C^{new}\}$.

In our formal model, looping back amounts to start a new session. Then the collapsed semantics is a partial formalization of the idea that names generated in previous sessions can be confused. We note that the logical *length* of a session can be suitably adapted by unfolding the program. [1] As long as the behaviour of principals does *not* depend on name inequality, the collapsed semantics 'simulates' the standard one, *i.e.*, reachability of a control point in the standard semantics implies reachability of the same control point in the concrete one. Moreover, we introduce checkable syntactic conditions that guarantee the equivalence of the standard and the collapsed semantics (section 2).

As a second contribution, we provide a set-based analysis of the collapsed semantics. From an algorithmic point of view, the fact that the collapsed semantics operates over a *finite* signature is exploited to associate to a system of processes Eq a system of set constraints Φ_{Eq} such that the reachability of a control state in Eq implies the nonemptiness of a distinguished variable in the least solution of Φ_{Eq}. It turns out that the set constraints generated are a variant of those studied in [CP97, TDT00] so that the nonemptiness problem can be solved by a suitable adaptation of standard techniques (section 3). In particular, we give a general construction that handles the renaming operators introduced by the collapsed semantics and we point out a 'linear' subclass of definite set constraints that can be solved in PTIME.

We characterize a situation (without name generation) where the set based analysis is exact and obtain a new decidable class of cryptographic protocols with a complexity ranging between simple and double exponential time. Unfortunately, the precision of the set-based analysis requires iteration which, in presence of name generation, is different from tail recursion (used in proving precision of the collapsed semantics). Therefore the obtained analysis of protocols with name generation is not exact, our results suggest only that if all our syntactic conditions are met then the loss of precision is not big. We leave open the problem whether control reachability under these conditions is decidable. Finally, we give examples of cryptographic protocols that fit our syntactic conditions, and we illustrate how our model can be used to formalize the standard property of *secrecy* as well as a more elusive property of *freshness* (section 4).

1.1 Related work

We restrict our attention to authors who considered fully automatic methods to prove correctness of recursive cryptographic protocols in Dolev-Yao model.

Monniaux [Mon99] introduces tree automata (which are related to set-constraints) to represent the knowledge of the adversary. In his study, he restricts his attention to protocols without recursion, but the approach has been generalized to handle recursion by the following authors. He already notices the lack of precision of the set based analysis pointed out in example 4.1.

At about the same time, Weidenbach [Wei99] applies the SPASS theorem prover to the analysis of a protocol by Neuman and Stubblebine. He reduces the verification problem to a proof search in monadic Horn clauses (again related to set-constraints). There is no general guarantee of termination for his method but some decidable subclasses are pointed out. Name generation is modelled by skolemisation –looking at the name generator as an existential– but no result is reported on the soundness or completeness of this modelling with respect to a standard operational semantics of name generation.

Later, Genet and Klay [GK00] and Goubault [Gou00] also rely on tree automata to produce a conservative approximation of cryptographic protocols. Genet and Klay consider a rather rough approximation where every action can be performed at any time and point out in their conclusion the need for a refined analysis of the control (a programme we carry on to some extent here).

[1] Clearly, if we adopted *iteration* rather than tail recursion, then it would be more difficult to design a sensible collapsed semantics; as shown by Stoller [Sto99] an attack may require exponentially many parallel sessions.

Goubault proposes to approximate a name generator by a superset of all values that it can generate in a run (which is quite different from the collapsed semantics studied here). In these two papers, no definite analysis is given of the complexity and the precision of the approximation schema.

Finally, Comon et al. [CCM01] introduce a special class of tree automata *with memory* that allows to decide in DEXPTIME secrecy properties for a class of cryptographic protocols strictly containing the ping-pong protocols. This work does not cover the notion of name generation and the class of protocols considered is *incomparable* with the one considered in theorem 4.3; their class goes beyond regular tree languages but, because of the so called *basicness condition*, only particular input filters are admitted.

2 Model

We begin by recalling the basic assumptions in Dolev-Yao model (section 2.1). Then we introduce a particular family of tail recursive processes, their operational semantics, and the related control reachability problem (section 2.2). We propose a certain number of syntactic conditions and exhibit related fragments of the model where the control reachability problem is still undecidable (section 2.3). Finally, we define a collapsed semantics simulating the standard one and determine a checkable condition where the collapsed semantics is precise (section 2.4).

2.1 Dolev-Yao model

We recall that in Dolev-Yao model communications are mediated by an adversary that can analyse the messages exchanged and synthesize new ones. To represent the set of messages we assume an infinite set of constants $\mathcal{N}$ and consider terms over the (infinite) signature $\Sigma = \mathcal{N} \cup \{E^2, \langle \_, \_ \rangle^2\}$. Thus we have two binary constructors: E for encryption and $\langle \_, \_ \rangle$ for pairing.

We use the following standard notation: $x, y, \ldots$ for (term) variables; V for the set of variables; $T_\Sigma(V)$ for the collection of finite terms over $\Sigma \cup V$; $t, t', \ldots$ for terms in $T_\Sigma(V)$; $\vec{t}$ for vectors of terms; $[t/x]$ for the substitution of t for x. We denote with $Var(t)$ the variables occurring in the term t.

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

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

We abbreviate a message $E(\cdots E(t, D_n), \ldots, D_1)$ with $D_1 \cdots D_n t$ thus regarding nested encryptions as *words*.

The functions S for *synthesis* and A for *analysis* are closure operators over the power set of messages $\mathcal{M}$ defined as follows:

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

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

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

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

For the sake of simplicity, in this paper we restrict our attention to *symmetric* encryption where encryption and decryption keys coincide and moreover we make the rather standard assumption that keys are atomic names, *i.e.*, a pair or an encrypted message cannot be used as a key. However, our results are not strictly dependent on these hypotheses and we expect that they can been adapted to, *e.g.*, *public* keys and *complex* symmetric keys.

2.2 Tail-recursive processes

In a message, it is useful to distinguish between *data* and *key* positions.

Definition 2.1 *Let u, v be variables or constants and let $t \in T_\Sigma(V)$ be a term. We define two predicates $Occ_{Data}(u,t)$ and $Occ_{Key}(u,t)$ that are satisfied if u occurs in t in* Data *or* Key *position, respectively:*

$$\begin{array}{ll} Occ_{Data}(u,u) & \\ Occ_{Data}(u,\langle t_1,t_2\rangle) & \textit{if } Occ_{Data}(u,t_i) \;\; i=1 \textit{ or } 2 \\ Occ_{Data}(u,E(t,v)) & \textit{if } Occ_{Data}(u,t) \\ & \\ Occ_{Key}(u,\langle t_1,t_2\rangle) & \textit{if } Occ_{Key}(u,t_i) \;\; i=1 \textit{ or } 2 \\ Occ_{Key}(u,E(t,v)) & \textit{if } u=v \textit{ or } Occ_{Key}(u,t)\ . \end{array}$$

Remark 2.2 *Note that u may occur in t without occurring in either data or key position. This may happen only when no ground instance of t can produce a message,* i.e., *when t contains a subterm $E(t_1,t_2)$ and t_2 is neither a constant nor a variable.*

We fix a finite system Eq of k equations:

$$A_i = \nu\vec{c}_i\, Q_i \quad \text{where} \quad Q_i \equiv ?t^i_1.!s^i_1 \ldots ?t^i_{l_i}.!s^i_{l_i}.A'_i \quad \text{for} \quad i=1,\ldots,k,\ l_i \geq 1$$

with distinct process identifiers A_i, $i = 1,\ldots,k$ and where A'_i is either A_i or a special state err. The following definitions of configuration and reduction will refer to *this* system. The intuitive semantics is as follows: A_i generates a vector of fresh names $\vec{c}_i$, it engages in an alternating sequence of input-output where it receives messages of the shape t^i_j and emits messages of the shape s^i_j, and finally either it loops back or it reaches an erroneous state.

We suppose that there is always a ground instance of the terms t^i_j, s^i_j that can produce a message. If such a ground instance does not exist, then the thread will be blocked at the occurrence of the term, and an equivalent system is obtained by inserting at a corresponding position a filter which can be never satisfied.

We will say that the terms t^i_j are *filters* and define the *filter variables* as

$$FVar(t^i_j) = Var(t^i_j) \backslash \bigcup_{h=1,\ldots,j-1} Var(t^i_h)\ .$$

Similarly, the *key variables* are defined by $KVar(s^i_j) = \{z \in Var(s^i_j) \mid Occ_{Key}(z, s^i_j)\}$. The *filters* t^i_j must correspond to a combination of projections and decryptions. For this purpose, we require that filter variables occur in data position. Namely, if $x \in FVar(t^i_j)$ then x cannot occur in t^i_j in key position (cf. definition 2.1). This condition forbids, *e.g.*, to set $t^i_1 = E(x,y)$ as in the operational semantics presented below this would allow to learn the contents and encryption key of any encrypted message.

Finally, we require that the variables in an output are contained in the variables of the preceding input, *i.e.*, $Var(s^i_j) \subseteq FVar(t^i_j)$. We will see next that there is no loss of generality in this assumption.

We now turn to the formal operational semantics. For every vector of generated names $\nu\vec{c}_i$, $i = 1,\ldots,k$ in Eq, we reserve the vectors of distinct constants:

$$\vec{C}^{old}_i, \quad \vec{C}^{new}_i, \quad \vec{C}^j_i, j = 0,1,2,\ldots$$

We assume that no confusion arises with constants appearing in Eq or with constants related to another thread. We will see that the standard semantics of name generation relies only on the constants $\vec{C}^j_i, j = 0,1,2,\ldots$ while the collapsed semantics presented in section 2.4 will rely on the constants $\vec{C}^{old}_i, \vec{C}^{new}_i$. We say that a thread P *relates to* the equation $A_i = \nu\vec{c}_i\ Q_i$ if P is either A_i or an instance of $?t^i_j.!s^i_j \ldots ?t^i_{l_i}.!s^i_{l_i}.A'_i$, $j \geq 1$.

Definition 2.3 (standard configuration) *A* standard configuration *is a pair* (R, T) *where:*

- $R \equiv (P_1, n_1) \mid \cdots \mid (P_k, n_k)$ *is the parallel composition of* k *pairs composed of a thread* P_i *relating to the* i^{th} *equation and a counter* n_i.
- T *is a finite set of messages representing the knowledge of the adversary,*

and such that the constants $\vec{C}_i^{old}$, $\vec{C}_i^{new}$, $\vec{C}_i^{j}$ *for* $i = 1, \ldots, k$ *and* $j > n_i$ do not *occur in* (R, T).

We assume that parallel composition is associative and commutative and feel free to write R as $(P_i, n_i) \mid R'$, where R' might be empty.

Definition 2.4 (standard reduction) *We define a reduction relation on standard configurations as follows:*

$$\begin{array}{llll}
(\mathsf{unfold}) & ((A_i, n_i) \mid R, T) \rightarrow ((\sigma_i^{n_i+1} Q_i, n_i + 1) \mid R, T), & \text{if } \sigma_i^{n_i+1} = [\vec{C}_i^{n_i+1}/\vec{c}_i] \ , \\
(\mathsf{i/o}) & ((?t.!s.P, n_i) \mid R, T) \rightarrow ((\theta P, n_i) \mid R, T \cup \{\theta s\}) & \text{if } \theta t \in S(A(T)) \text{ and } \theta s \in \mathcal{M} \ .
\end{array}$$

The first rule (unfold) expands the recursive definition, instantiates the generated names $\vec{c}_i$ with fresh constants, and increments the related counter n_i. The second rule (i/o) inputs from the adversary a message θt matching a filter and outputs a message θs. Note that if θs is *not* a message then the reduction cannot take place. We can now state the control reachability problem.

Definition 2.5 *We fix* $R_0 \equiv (A_1, 0) \mid \cdots \mid (A_k, 0)$ *as initial control,* $T_0 \neq \emptyset$ *as initial knowledge of the adversary, and write* $(R_0, T_0) \xrightarrow{*} \mathsf{err}$ *if* $(R_0, T_0) \xrightarrow{*} ((\mathsf{err}, n) \mid R', T')$ *for some* n, R', T'. *The control reachability problem amounts to determine whether* $(R_0, T) \xrightarrow{*} \mathsf{err}$.

2.3 Undecidable fragments

It is convenient to introduce some *syntactic transformations* that do not affect the control reachability problem. Consider a thread $A = \nu\vec{c}\,?t_1.!s_1 \ldots ?t_n.!s_n.A_i'$ and suppose s_i is the first output that depends on filter variables of $t_1, \ldots, t_{i-1}$, say $\vec{x}_1, \ldots, \vec{x}_{i-1}$. If $\vec{d} \equiv d_1, \ldots, d_j$ and $\vec{x} \equiv x_1, \ldots, x_j$ then $\langle \vec{d} \cdot \vec{x} \rangle$ stands for $\langle d_1 x_1, \ldots, d_j x_j \rangle \equiv \langle E(x_1, d_1), \ldots, E(x_j, d_j) \rangle$. With this convention, we can rewrite the thread as

$$\begin{array}{ll}
A & = \nu\vec{c}\,\nu\vec{d}_1, \ldots, \vec{d}_{i-1}\,?t_1.!\langle s_1, \langle \vec{d}_1 \cdot \vec{x}_1 \rangle \rangle. \ldots \\
& ?t_{i-1}.!\langle s_{i-1}, \langle \vec{d}_{i-1} \cdot \vec{x}_{i-1} \rangle \rangle. \\
& ?\langle t_i, \langle \vec{d}_1 \cdot \vec{y}_1 \rangle, \ldots, \langle \vec{d}_{i-1} \cdot \vec{y}_{i-1} \rangle \rangle ![\vec{y}_1, \ldots, \vec{y}_{i-1}/\vec{x}_1, \ldots, \vec{x}_{i-1}] s_i. \ldots \\
& ?t_n.!s_n.A_i'
\end{array}$$

where $\vec{d}_j, \vec{y}_j$ are fresh. In other terms, we store under the fresh keys $\vec{d}_j$ the parameters $\vec{x}_j$ and retrieve them again in the filter just preceding the output s_i. By iterating this transformation, we obtain an equivalent thread satisfying the condition $Var(s_j) \subseteq FVar(t_j)$.

Repeated outputs $?t.!s_1.!s_2$ can be encoded as $?t.!s_1.?x.!s_2$ with x fresh. Repeated inputs $?t_1.?t_2.!s$ can be encoded as $?t_1.!t_0.?t_2.!s$ where t_0 is a term in the initial knowledge. We will also write $?t_1.!s_1 \ldots ?t_n.!s_n.0$ to indicate that the thread stops at point 0. This can be encoded by inserting at the place of 0 a filter that can never be passed. Finally, we may generate names during a session as in $\ldots ?t.!s.\nu c\, ?t'.!s'. \ldots$ which is equivalent to generate them at the very beginning.

Next we introduce four syntactic conditions on the system Eq. We anticipate that the syntactic transformations we have presented above can be performed while satisfying these conditions.

Definition 2.6 (variable dependency) *We say that the variables* x *and* y *are* dependent *in a term* t *if for some* C *there exists a subterm* $E(t', C)$ *of* t *with two distinct occurrences of* x *and* y *(*x *depends on itself if it occurs twice in* t'*). Otherwise, we say that* x *and* y *are independent in* t.

Definition 2.7 (syntactic conditions) *The system Eq satisfies the:*

(1) LINEARITY *condition if the filters* t^i_j *are linear,* i.e., *each variable in* $FVar(t^i_j)$ *occurs exactly once in* t^i_j.

(2) LOCALITY *condition if the filters* t^i_j *do* not *depend on previous filter variables,* i.e., $Var(t^i_j) = FVar(t^i_j)$.

(3) INDEPENDENCE *condition if for every i/o action* $?t.!s$, *assuming* $\eta : KVar(s') \to \mathcal{N}$ *is any assignment and* $\langle s_1, \ldots, s_m \rangle \equiv \eta s$ *the following holds for* $l = 1, \ldots, m$: *either (i)* $\sharp Var(s_l) \leq 1$, *or or (ii)* s_l *is a linear term and the variables* $Var(s_l)$ *are independent in* ηt.

(4) DATA OR KEY *condition if every generated name occurs in the related thread either in data or key position (but not both, cf. definition 2.1), and moreover, if at least one generated name occurs in data position then every variable occurring in a filter can only occur in the related thread in data position.*[2]

Conditions (1–3) are partially motivated by the following undecidability result whose proof, given in appendix A.2, is based on a rather direct encoding of 2-counter machines. Conditions (1–3) will also play a role in the complexity and precision of the set-based analysis. Moreover, conditions (1–2) together with condition (4) will be used in section 2.4 to characterize the precision of the collapsed semantics. Examples of cryptographic protocols satisfying conditions (1–4) will be given in section 4.

Theorem 2.8 (undecidability) *There are encodings of 2-counter machines showing that violation of one of the conditions* LINEARITY, LOCALITY, INDEPENDENCE *is sufficient for the undecidability of the control reachability problem.*

Remark 2.9 *If we assume the* DATA OR KEY *condition then theorem 2.8 still holds. Moreover, if we violate the* INDEPENDENCE *condition then undecidability does not rely on name generation.*

2.4 Collapsed semantics

We introduce the notion of collapsed configuration and reduction mimicking definitions 2.3 and 2.4.

Definition 2.10 (collapsed configuration) *A* collapsed configuration *is a pair* (R, T) *where:*

- R *is the parallel composition of* k *threads* $P_1 \mid \cdots \mid P_k$ *with* P_i *relating to the* i^{th} *equation,*
- T *is a finite set of messages representing the knowledge of the adversary,*

and such that the constants $\vec{C}^j_i$ *for* $i = 1, \ldots, k$ do not *occur in* (R, T).

Definition 2.11 (collapsed reduction) *We define a reduction relation on collapsed configurations as follows:*

$$\begin{array}{llll}
(\mathsf{unfold}) & (A_i \mid R, T) & \to (\sigma^{new}_i Q_i \mid \sigma^{old}_i R\ , \sigma^{old}_i T), & \textit{if } \sigma^{new}_i = [\vec{C}^{new}_i / \vec{c}_i] \textit{ and } \sigma^{old}_i = [\vec{C}^{old}_i / \vec{C}^{new}_i] \\
(\mathsf{i/o}) & (?t.!s.P \mid R, T) & \to (\theta P \mid R, T \cup \{\theta s\}), & \textit{if } \theta t \in S(A(T)) \textit{ and } \theta s \in \mathcal{M}\ .
\end{array}$$

The notion of control reachability for collapsed configurations is an immediate adaptation of definition 2.5. Without name generation the collapsed semantics coincides with the standard semantics and therefore it follows from the remark 2.9 that the control reachability problem for the collapsed semantics is also undecidable in general.

We want to relate the control reachability problem for standard and collapsed configurations. Given any standard configuration (R, T) where $R \equiv (P_1, n_1) \mid \cdots \mid (P_k, n_k)$ we define a substitution τ_R as follows:

$$\tau_R(\vec{C}^j_i) = \begin{cases} \vec{C}^{new}_i & \text{if } j = n_i \\ \vec{C}^{old}_i & \text{if } j < n_i\ . \end{cases}$$

[2]The restriction that filter variables occur only in data position is needed to avoid that a principal uses generated nonces as keys. The condition could be omitted if we included in the model a *typing* mechanism to distinguish nonces from keys.

We extend the definition of τ_R to standard configurations as follows:

$$\tau_R(R,T) = (\tau_R P_1 \mid \cdots \mid \tau_R P_k, \tau_R T) \ .$$

We can prove the following proposition by case analysis on the reduction rules (see appendix A.3).

Proposition 2.12 (simulation) *Let (R,T) be a standard configuration.*

(1) *If $(R,T) \to (R_1,T_1)$ then $\tau_R(R,T) \to \tau_{R_1}(R_1,T_1)$.*

(2) *If $((A_1,0) \mid \cdots \mid (A_k,0),T_0) \xrightarrow{*} \mathsf{err}$ then $(A_1 \mid \cdots \mid A_k,T_0) \xrightarrow{*} \mathsf{err}$.*

The following two propositions analyse the impact of the syntactic conditions on the precision of the collapsed semantics.

Proposition 2.13 *There are examples showing that the violation of one of the conditions* LINEARITY*,* LOCALITY*,* DATA OR KEY *is sufficient to compromise the precision of the collapsed semantics (even when condition* INDEPENDENCE *is satisfied).*

PROOF HINT. The following example concerns the violation of the DATA OR KEY condition:

$$\begin{array}{ll}
A_1 & = \nu b\, ?D_0x.!D_1\langle b,x\rangle.A_1 \\
A_2 & = \nu c_1, c_2\, !D_0c_1.?D_1\langle y,c_1\rangle.!yE_1.!Dy. \\
 & \quad !D_0c_2.?D_1\langle z,c_2\rangle.!zE_2.!Dz.0 \\
A_3 & = ?Dx.!x.0 \\
A_4 & = ?E_1.?E_2.\mathsf{err}
\end{array}$$

taking $T_0 = \{C\}$ and running A_1 twice (see appendix A.4). The other cases are similar. □

On the other hand, if the three conditions hold then the control reachability problem in the collapsed semantics is equivalent to the control reachability problem in the standard one (the rather technical proof is given in appendix A.5).

Theorem 2.14 (precision of collapsed semantics) *Suppose the system Eq satisfies conditions* LINEARITY*,* LOCALITY*, and* DATA OR KEY*. Then $((A_1,0) \mid \cdots \mid (A_k,0),T_0) \xrightarrow{*} \mathsf{err}$ iff $(A_1 \mid \cdots \mid A_k,T_0) \xrightarrow{*} \mathsf{err}$.*

3 Set based analysis

In this section we perform an analysis based on *set constraints* of the control reachability problem in the collapsed semantics. In section 3.1 we introduce a particular family of set constraints tailored to our needs, in section 3.2 we show how to generate them, and in section 3.3 we explain how to solve them.

3.1 A family of set constraints

We will use a class of set constraints very close to *definite* set constraints with membership expressions [HJ90, CP97, TDT00], but there are few differences. First, we do not allow any expressions except variables on the right-hand side of inclusion, which is quite usual in set based program analysis [HJ94]; therefore, we are not interested in testing satisfiability but in computing the least solution (such constraints are always satisfiable—a trivial solution is a valuation assigning the set of all terms to each variable). Second, we use conditional inclusions of the form $\mathsf{if}\ \mathsf{ne}(S)\ \mathsf{then}\ S' \subseteq X$. This is not an essential extension since the emptiness test is an inherent part of every set constraint solving algorithm. Moreover, in a setting with more complicated expressions on the right-hand side of inclusion, such conditional inclusion is equivalent to $f(S,S') \subseteq f(S,X)$. Third, we use membership expressions as in [TDT00]. A not surprising, but probably new observation here is that if the formulas in the set comprehension part of these expressions are all linear then set constraints can be solved in polynomial time (see theorem 3.3(2)). Finally, we use a novel operation of renaming needed in the representation of the unfolding rule of the collapsed semantics (definition 2.11). To our knowledge, this operation was not present in any previous work on set constraints.

3.1.1 Syntax of set constraints

We assume that a finite signature $\Sigma = \{f, \ldots\}$ of function symbols is given. Every function symbol has a fixed arity; symbols of arity 0 are also called constants and denoted with $c, \ldots$ We will use a set of *individual variables* $V = \{x, y, \ldots\}$ ranging over terms from T_Σ and a set of *set variables* $\Xi = \{X, Y, \ldots\}$ ranging over sets of terms 2^{T_Σ}.

Set expressions are given by the grammar:

$$S \ ::= \ X \mid f(S_1, \ldots, S_n) \mid \{x \mid t \in S\} \mid \sigma S,$$

where X ranges over Ξ and f over n-ary function symbols from Σ. The meta-variable t ranges over *linear* terms in $T_\Sigma(V)$ and may contain individual variables possibly different from x. The hypothesis that terms are linear can be removed at the price of an increase in the complexity of the solving algorithm (see [CP97, TDT00]).

The renaming operator σ is a substitution of the form $[\vec{c'}/\vec{c}\,]$ that replaces all occurrences of constant symbols from $\vec{c}$ with a respective symbol in $\vec{c'}$. To ensure termination of the solving algorithm we assume that all renamings here are idempotent ($\sigma\sigma = \sigma$) and commutative ($\sigma\sigma' = \sigma'\sigma$) and come from a finite set *Sub*, so that a composition of renamings can be canonically represented by a subset of *Sub* taking conventionally the identity if the subset is empty. The assumption that the renamings operate on constant symbols is not essential—an extension to other function symbols is straightforward.

Set constraints are:

$$\Phi \ ::= \ S \subseteq X \mid \mathsf{if} \ \bigwedge_i \mathsf{ne}(S_i) \ \mathsf{then} \ S \subseteq X \mid \Phi_1 \wedge \Phi_2 \ .$$

As is usual, we identify a conjunction of constraints with the set of all conjuncts. We will require that for every inclusion $S \subseteq X$ in all subexpressions $\{x \mid t \in S'\}$ of S, the variable x occurs in the term t (this requirement is not essential, but it allows to avoid special rules in table 2 for handling the set of all terms). We will not require this in subexpressions of $\mathsf{ne}(S)$.

3.1.2 Semantics of set constraints

A *valuation* $\alpha : \Xi \to 2^{T_\Sigma}$ is a mapping assigning sets of ground terms to set variables. The semantics of set expressions relative to a valuation α is defined recursively as follows.

$$\begin{aligned}
[\![X]\!]_\alpha &= \alpha(X) \\
[\![f(S_1, \ldots, S_n)]\!]_\alpha &= \{f(t_1, \ldots, t_n) \mid t_1 \in [\![S_1]\!]_\alpha, \ldots, t_n \in [\![S_n]\!]_\alpha\} \\
[\![\{x \mid t \in S\}]\!]_\alpha &= \{t' \in T_\Sigma \mid \{\vec{y}\} = \mathit{Var}(t)\backslash\{x\} \text{ and } \exists\, \vec{s}\ [t'/x, \vec{s}/\vec{y}]t \in [\![S]\!]_\alpha\} \\
[\![\sigma S]\!]_\alpha &= \{\sigma t \mid t \in [\![S]\!]_\alpha\} \ .
\end{aligned}$$

We note that if x does not occur in t and there is an instance of t in $[\![S]\!]_\alpha$ then $[\![\{x \mid t \in S\}]\!]_\alpha$ is the set of all terms T_Σ. Similarly, if x does not occur in t and there is no instance of t in $[\![S]\!]_\alpha$ then $[\![\{x \mid t \in S\}]\!]_\alpha$ is the empty set.

A valuation α is a *solution* of a constraint Φ if for all conjuncts $S \subseteq X$ in Φ we have $[\![S]\!]_\alpha \subseteq \alpha(X)$, and for all conjuncts $\mathsf{if}\ \wedge_i \mathsf{ne}(\vec{S_i})\ \mathsf{then}\ S' \subseteq X$ we have $[\![S']\!]_\alpha \subseteq \alpha(X)$ whenever $[\![S_i]\!]_\alpha \neq \emptyset$ for all i.

Obviously every constraint Φ is satisfiable—a trivial solution is a valuation assigning T_Σ to each variable. In the following we want to find the least solution of a given constraint Φ. This *least solution* may be defined inductively by:

$$\begin{aligned}
\alpha_0(X) &= \emptyset, \\
\alpha_{i+1}(X) &= \bigcup \{[\![S]\!]_{\alpha_i} \mid S \subseteq X \in \Phi \text{ or } (\mathsf{if}\ \mathsf{ne}(\wedge_j S'_j)\ \mathsf{then}\ S \subseteq X \in \Phi, [\![S'_j]\!]_{\alpha_i} \neq \emptyset)\}, \\
\alpha(X) &= \textstyle\bigcup_{i \in \mathbb{N}} \alpha_i(X) \ .
\end{aligned}$$

(init)	$t \subseteq T_{R_0}$	(1)
(err)	$T_{\mathsf{err}\mid R} \subseteq T_{\mathsf{err}}$	(2)
(A_1)	$\{x \mid \langle x, y\rangle \in T_R\} \subseteq T_R$	(3)
(A_2)	$\{y \mid \langle x, y\rangle \in T_R\} \subseteq T_R$	(3)
(A_3)	if $\mathsf{ne}(\{y \mid C \in T_R\})$ then $\{x \mid E(x, C) \in T_R\} \subseteq T_R$	(3)
(S_1)	$\langle T_R, T_R\rangle \subseteq T_R$	(3)
(S_2)	if $\mathsf{ne}(\{y \mid C \in T_R\})$ then $E(T_R, C) \subseteq T_R$	(3)
(unfold)	$\sigma_i^{old} T_{A_i\mid R} \subseteq T_{\sigma_i^{new} Q_i \mid R}$	(4)
($\mathsf{i/o}_1$)	if $\mathsf{ne}(\{y \mid \eta t \in T_R\})$ then $T_R \subseteq T_{R'}$	(5)
($\mathsf{i/o}_2$)	if $\mathsf{ne}(\{y \mid \eta t \in T_R\})$ then $[\vec{S}/\vec{x}](\eta s) \subseteq T_{R'}$	(5)

(1) (R_0, T_0) initial configuration, $t \in T_0$.
(2) $\mathsf{err} \mid R$ control state.
(3) For all R control state and $C \in \mathcal{N}$.
(4) $A_i \mid R$ control state.
(5) $R \equiv ?t.!s.R_1 \mid R_2$ control state, $R' \equiv R_1 \mid R_2$,
$\{x_1, \ldots, x_n\} = \mathit{Var}(s) \backslash \mathit{KVar}(s)$,
$\eta : \mathit{KVar}(s) \to \mathcal{N}$, $S_i \equiv \{x_i \mid \eta t \in T_R\}$.

Table 1: Constraints generated

3.2 Constraints generation

Here we show how to generate set constraints from a system Eq satisfying the conditions LINEARITY, LOCALITY, and INDEPENDENCE. We prove that the generated constraints give a conservative approximation of the protocols. [3]

Under the syntactic conditions above a *control state* is a process R that can be decomposed in $P_1 \mid \cdots \mid P_k$ so that P_i is either A_i or err or $\sigma_i^{new}(?t_j^i.!s_j^i \ldots ?t_{l_i}^i.!s_{l_i}^i).A_i'$ For a system of k threads each comprising n i/o alternations there are at most $(n+2)^k$ control states and for a reachable collapsed configuration (R, T), R is always a control state.

For every control state R we introduce a set variable T_R. Intuitively it will represent (an approximation of) the knowledge of the adversary at this control point, so for any reachable configuration (R, T) the variable T_R will contain the set $S(A(T))$. Moreover, we introduce a set variable T_{err} that will be empty if an erroneous configuration is not reachable.

Table 1 gives the rules to generate constraints. The rules are self-explanatory: (init) is for the initial configuration, (err) for determining T_{err}, (A_{1-3}) and (S_{1-2}) for analysis and synthesis, respectively. The rule (unfold) is for the unfolding step. Here we note that every renaming occurring in the (unfold) constraint is of the form $[\vec{C}_i^{old}/\vec{C}_i^{new}]$, so for two different renamings their domains are disjoint, and every domain is disjoint from any range. This gives idempotency and commutativity as required. Finally, the ($\mathsf{i/o}_{1-2}$) rules cover the (i/o) reduction. In these rules, we get rid of the key variables in in the output by considering all their possible instances. This is conceptually simple but may lead to inefficiency. In practice, one can introduce a limited form of intersection and write set expressions such as $\{x_i \mid t \in X\} \cap \mathcal{N}$.

The set constraints provide a conservative approximation of the collapsed semantics. The proof given in appendix A.6 proceeds by induction on the length of the reduction $(R_0, T_0) \xrightarrow{*} (R, T)$. We will see in example 4.1 that due to the constraints generated by rule ($\mathsf{i/o}_2$) the analysis is not exact.

Proposition 3.1 *Let α be a solution of the constraints generated according to table 1. Then:*
(1) *If $(R_0, T_0) \xrightarrow{*} (R, T)$ then $S(A(T)) \subseteq \alpha(T_R)$.*

[3]The method can be generalized to protocols not satisfying the conditions above. To this end, it seems natural to rely on *non-linear* constraints in order to limit the loss of precision. In this case the constraints can be solved in DEXPTIME.

1. $\bigwedge_i \mathsf{ne}(X_i) \rightarrow \mathsf{ne}(f(X_1,\ldots,X_n))$

2. $\bigwedge_i \mathsf{ne}(\{x \mid t_i \in X_i\}),\ f(X_1,\ldots,X_n) \subseteq X \rightarrow \mathsf{ne}(\{x \mid f(t_1,\ldots,t_n) \in X\})$
   (provided $f(t_1,\ldots,t_n)$ occurs in Φ)

3. $\mathsf{ne}(S), S \subseteq X \rightarrow \mathsf{ne}(X)$

4. $\mathsf{ne}(X) \rightarrow \mathsf{ne}(\{x \mid y \in X\})$ $\quad$ (y can be x)

5. $S \subseteq X,\ X \subseteq Y \rightarrow S \subseteq Y$

6. $\{x \mid f(t_1,\ldots,t_i[x],\ldots,t_n) \in X\} \subseteq Y,\ f(X_1,\ldots,X_n) \subseteq X \rightarrow$
   $\mathsf{if}\ \bigwedge_{j\neq i} \mathsf{ne}(\{x \mid t_j \in X_j\})\ \mathsf{then}\ \{x \mid t_i[x] \in X_i\} \subseteq Y$

7. $\{x \mid x \in X\} \subseteq Y \rightarrow X \subseteq Y$

8. $\bigwedge_i \mathsf{ne}(S_i),\ \mathsf{if}\ \bigwedge_i \mathsf{ne}(S_i)\ \mathsf{then}\ S \subseteq X \rightarrow S \subseteq X$

9. $f(X_1,\ldots,X_n) \subseteq X \rightarrow \sigma(f)(X_1^\sigma,\ldots,X_n^\sigma) \subseteq X^\sigma$

Table 2: Saturation rules for linear constraints

(2) *If* $(R_0, T_0) \xrightarrow{*} \mathsf{err}$ *then* $\alpha(T_{\mathsf{err}}) \neq \emptyset$.

3.3 Constraints solving

We say that a constraint Φ is in a *shallow* form if in every expression of the form $f(S_1,\ldots,S_n)$, $\{x \mid t \in S\}$ or σS occurring in Φ the expressions $S, S_1,\ldots,S_n$ are set variables. Any set constraint can be transformed into an equivalent one in shallow form by replacing every occurrence of a nested subexpression S with a fresh set variable X_S and adding a conjunct $S \subseteq X_S$. For the rest of this section we assume that all constraints are in shallow form.

Given a constraint Φ in shallow form over a set of variables Ξ and a set of renamings *Sub* we construct a constraint Φ' over a set of variables $\Xi' = \{(Y,u) \mid Y \in \Xi, u \subseteq \mathit{Sub}\}$ and an *empty* set of renamings. Thus if in Φ we have n variables and k renaming we have $n \cdot 2^k$ variables in Φ'. If $X = (Y,u) \in \Xi'$ and $\sigma \in \mathit{Sub}$ then X^σ stands for $(Y, u \cup \{\sigma\})$. Then Φ' is obtained from Φ by first replacing all set variables Y by $(Y,\emptyset)$ and then all set expressions σX by X^σ. By this construction we get rid of the set expressions σS and we will see that the least solution of Φ' when restricted to the variables $(Y,\emptyset)$ gives the least solution of Φ.

We say that a constraint Φ is in a *solved* form if every conjunct in Φ is of the form $f(X_1,\ldots,X_n) \subseteq X$. A constraint in a solved form can be seen as a transition table of a tree automaton whose states are set variables. The least solution of such a constraint is then a valuation that assigns to a variable X the language recognized by this automaton with X as a final state.

Our constraints solving algorithm applies the rules in table 2 starting from Φ'. Each consequence is an inclusion between two terms from a set of bounded size; this is used to show the termination and to derive the upper bound. Then given the initial constraint Φ' we infer all consequences of Φ' under the rules and then simply remove all inclusions that are not in solved form.

The algorithm can be seen as an elimination of backward transitions (constraints $\{x \mid t \in X\} \subseteq X'$) from a two-way tree automaton, together with an elimination of ε-transitions (constraints $X \subseteq X'$) and renaming operations.[4]

Each rule in table 2 translates to a basic fact. Rules 1–4 are used to infer the non-emptiness of set expressions. In the particular case of rules 1 and 2 where $n = 0$ we obtain the empty conjunction, *i.e.*, *true*, and f is a constant symbol. This gives in particular $\mathsf{ne}(c)$ for all constant

[4] In this view, non-linear constraints would require *alternating* rather than non-deterministic tree automata.

symbols $c \in \Sigma$. The side condition in rule 2 is used to avoid infinitely many consequences, which is necessary for termination. In rule 4 the variable y is any variable in V, including x.

The non-emptiness information is needed in rule 6, the decomposition rule (in this rule we assume that t_i is the only term among $t_1, \ldots, t_n$ with an occurrence of x, which is stressed by the notation $t_i[x]$). Note that if one of the expressions $\{x \mid t_i \in X_i\}$ is empty, then $\{x \mid f(t_1, \ldots, t_n) \in f(X_1, \ldots, X_n)\}$ is the empty set regardless of the other expressions; then $\{x \mid f(t_1, \ldots, t_n) \in f(X_1, \ldots, X_n)\} \subseteq Y$ is satisfied while $\{x \mid t_i \in X_i\}$ need not be satisfied.

Rule 6 is used to eliminate membership expressions with deep terms (more formally, to make the inclusions involving these expressions redundant). Intuitively, it reduces the depth of terms in membership expressions. Rule 7 then eliminates membership expressions with terms of depth 0. In the tree-automata view, this corresponds to elimination of backward transitions. Rule 5 eliminates (that is, makes redundant) inclusions involving only variables, which corresponds to ε-transition elimination in tree automata. Rule 8 eliminates conditional inclusions.

We say that a valuation α is *renaming compatible* if $\alpha(X^\sigma) = \sigma(\alpha(X))$. The last rule 9 makes sure that the least solution is renaming compatible.

Definition 3.2 (closed and solved form) *For a constraint Φ we denote by Φ^C the least set of constraints that contains Φ and is closed under all rules in table 2, and by Φ^S the restriction of Φ^C to the constraints in a solved form $f(X_1, \ldots, X_n) \subseteq X$.*

Theorem 3.3 *Let Φ be a linear constraint over Ξ, Sub and let Φ' be the associated constraint over $\Xi' = \Xi \times 2^{Sub}$.*

(1) *The least solution of ${\Phi'}^S$ restricted to the variables $(Y, \emptyset)$ of Ξ' is the least solution of Φ.*

(2) *The least solution of Φ can be computed in time $\mathsf{poly}(n \cdot 2^k)$ where n is the size of Φ and k is the number of renaming operations in Φ.*

4 Analysis of cryptographic protocols

In section 4.1 we discuss the syntactic conditions in definition 2.7 *in practice* and hint to some family of protocols that satisfy them. Then in section 4.2, we discuss the precision of the set based analysis, and finally in section 4.3 we address the issue of specifying familiar properties in our framework.

4.1 Pragmatics of the syntactic conditions

We discuss how restrictive our syntactic conditions are in practice.

Condition LINEARITY is usually satisfied. Note that a principal can still check whether a message contains *e.g.* a nonce it previously generated.

Condition LOCALITY requires that in a principal, filters are built out of names locally generated. For instance, we can only decrypt with keys generated by the principal. If a principal wants to send to another principal an encrypted information then it has first to get the local key of the recipient. This condition forces an asymmetric use of the keys: many principals can encrypt but only one principal can decrypt.[5]

Condition INDEPENDENCE is rather restrictive except for, *e.g.*, special ping-pong protocols. Unfortunately, as pointed out in remark 2.9, without this condition the reachability problem is undecidable already without name generation.

Condition DATA OR KEY is quite restrictive on the usage of generated keys. Since generated keys cannot be transmitted in data position, the principal who generates the key is the only one that can encrypt or decrypt messages with that key. On the other hand, the condition is not

[5]Incidentally, the terminology 'locality' is inspired by the π-calculus where it means that a thread cannot perform inputs on a received channel name; in our case, keys play the role of channels. See [AM01] for an analysis of the expressive power of name generation in the setting of the π-calculus.

so restrictive on generated nonces, *i.e.* names that are *not* used for encryption. In this case a principal that did not generate the nonce can actually use it, *e.g.*, to sign a certain message.

An example of cryptographic protocol satisfying all these conditions is the Andrew Secure RPC Protocol from [CJ97]

$$\begin{array}{lll} (1) & A \to B & : & A, E(Na, Kab) \\ (2) & B \to A & : & E(\langle Na+1, Nb\rangle, Kab) \\ (3) & A \to B & : & E(Nb+1, Kab) \\ (4) & B \to A & : & E(\langle K'ab, N'b\rangle, Kab) \end{array}$$

We do not have addition in our syntax, but we take $x+1$ as an abbreviation for $\langle x, 1\rangle$ where 1 is a constant symbol. Thus, we can model the protocol with the following two equations:

$$\begin{array}{llll} A_1 = & \nu n_a & A_2 = & \nu n_b, k'_{ab}, n'_b \\ (1) & !\langle A, E(n_a, K_{ab})\rangle. & (1) & ?\langle A, E(x, K_{ab})\rangle. \\ (2) & ?E(\langle n_a+1, y\rangle, K_{ab}). & (2) & !E(\langle x+1, n_b\rangle, K_{ab})). \\ (3) & !E(y+1, K_{ab}). & (3) & ?E(n_b+1, K_{ab}). \\ (4) & ?E(\langle z, w\rangle, K_{ab}).A_1 & (4) & !E(\langle k'_{ab}, n'_b\rangle, K_{ab}).A_2 \end{array}$$

It is easy to see that this system satisfies all our syntactic conditions. Other examples include a series of Woo and Lam Π protocols in [CJ97, section 6.3.10].

4.2 Precision of the set based analysis

It turns out that in general the set based analysis is *not* precise even under the four syntactic conditions given in definition 2.7.

Example 4.1 *Consider the initial knowledge $T_0 = \{ABD, ACD\}$ and the system*

$$A_1 = ?Ax.!x.?By.?Cz.\mathsf{err}\ .$$

The constraints generated do not express that after the first filter is passed either BD or AD are known but not both. Consequently, the following two filters are passed and the erroneous state is reached.

An approach to the characterization of the set-based analysis is to look at *iterated* threads *without* name generation. [6] A thread is iterated if a new copy of the thread is spawned as soon as the first input output action is performed. A system Eq of k iterated threads is then formalized by k equations:[7]

$$A_i = ?t^i_1.!s^i_1.(A_i \mid (?t^i_2.!s^i_2 \ldots ?t^i_{l_i}.!s^i_{l_i}.U)) \tag{1}$$

where U can be either the terminated state 0 or the erroneous state err. Since we do not have generated names, the transformation given in section 2.3 does not apply and we just require that $Var(s^i_j)$ is a subset of $\bigcup_{l\leq j} FVar(t^i_l)$ rather than of $FVar(t^i_j)$.

It turns out that these programs can be *flattened* so that each thread includes just *one alternation of input and output* and it is thus expressed by a tail recursive definition $A = ?t.!s.A$. Thus the following theorem 4.3 is also a result about the precision of the set based analysis for tail-recursive definitions without name generation and with one alternation of input and output.

Definition 4.2 *The system of iterated threads (1) satisfies, respectively, the* LINEARITY *and* INDEPENDENCE *conditions if for all equations $i = 1, \ldots, k$ the term $\langle t^i_1, \ldots, t^i_{l_i}\rangle$ is linear and the i/o action $?\langle t^i_1, \ldots, t^i_{l_i}\rangle.!\langle s^i_1, \ldots, s^i_{l_i}\rangle$ is independent in the sense of definition 2.7(3).*

Theorem 4.3 *Let n be the size of the system of iterated threads (1), c be the number of different keys, and k be the number of key variables. Suppose the system satisfies the* INDEPENDENCE *condition 4.2. Then the control reachability problem is* DEXPTIME*-hard and decidable in time* $\mathsf{exp}(n \cdot c^k)$. *Moreover, under the additional* LINEARITY *condition it is decidable in time* $\mathsf{poly}(n \cdot c^k)$.

[6] *With* name generation a simple variant of the example 4.1 shows that precision is lost.

[7] This formalization is equivalent to the standard notion of replication in π-calculus.

We expect that the factor c^k can be reduced, but the exponential blowup in the non-linear case is unavoidable as we can reduce the satisfaction of unary definite set constraints (see [CPT00]) to control reachability for the class of iterated systems considered even when neither pairs nor key variables are used (see proposition A.13 in section A.8 for details).

The upper bound relies on the flattening transformation mentioned above. By this transformation every input-output action can be repeated in every reachable configuration. This property coupled with the INDEPENDENCE condition allows to match the constraint generated by the rule ($\mathsf{i/o}_2$) (a proof is given in appendix A.8).

As a particular instance of this result, one can obtain yet another polynomial time decision procedure for ping-pong protocols which satisfy LINEARITY and do not contain variables in key position (see [DEK82] and [ALV01] for another decision procedure based on prefix-rewriting). Of course, the DEXPTIME lower bound implies that the class of decidable protocols considered in the theorem 4.3 above is strictly more expressive than the class of ping-pong protocols.

4.3 Specification of properties

As stated in proposition 3.1(1), the set based analysis provides in general a conservative, *i.e.*, upper, approximation of the knowledge of the adversary at any given control point. This approximation is represented by a tree automaton that can be effectively computed and is amenable to further analysis depending on the property we are interested in.

We give two concrete examples and leave the design of a full fledged specification language for further work. We point out that certain properties such as *authentication* that require checking whether some message is *not* known by the adversary, are not easily handled as the set based analysis presented here provides only an *upper* approximation.

4.3.1 Secrecy

Secrecy of a message (also known as confidentiality) is a property that requires that a message t is not known to the adversary (at some control point). To check whether the property holds at a given point of the control of a thread we can simply insert at that point the instruction $?t.\mathsf{err}$. The thread will reach the erroneous state if and only if t can be built from the adversary (and stop otherwise). In the set based analysis, we can check the emptiness of the set $\{x \mid t \in T_R\}$ for all corresponding control points R.

Another secrecy specification is to require that a certain message t is never known by the adversary. We can model this specification as follows: (i) we reserve a key S, and (ii) whenever a thread requires a certain message t to stay secret it emits the message $E(t, S)$. The secrecy specification is violated if we can reach a configuration (R, T) where $\langle t, E(t, S)\rangle \in S(A(T))$. Correspondingly, in the set based analysis we check the emptiness of the set $\{x \mid \langle x, E(x, S)\rangle \in T_R\}$ for all control points R. For each such R this last condition can be reduced to the emptiness of the intersection of two languages recognized by tree automata.

4.3.2 Freshness

In protocols with multiple sessions a given thread may be interested in knowing that the received message is relative to the *current* session, *i.e.*, it could not have been forged in a previous session. As a concrete example, consider the protocol given in section 4.1: the message received by A at step (4) might come from a previous session and this points out a weakness of the protocol.

In order to test freshness in our framework we can exploit the imprecision of the generated set constraints. Given a thread $A_i = \nu\vec{c}_i \ldots ?t \ldots . A_i$, suppose we want to check whether a message received in the filter t is fresh. Then we replace this thread by the thread $A = \nu\vec{c}\nu k \ldots ?t \ldots ?x.!E(x, k).A$ where k is a fresh key. The effect of the constraint generated by the ($\mathsf{i/o}_2$) rule for the added action $?x.!E(x, k)$ is that at the end of a session for every message s currently known by the adversary, the message $E(s, K^{new})$ is added to the value of the set variable $T_{A|R}$. Thus when A starts a new session the set of messages s such that $E(s, K^{old})$ is in the value of the corresponding set variable gives an over approximation of the messages known by the adversary. Then, if P is the point of the control corresponding to the input $?t$ in A, we have to check the emptiness of $\{y \mid E(\sigma_i t, K^{old}) \in T_{P|R}\}$ (where $\sigma_i = [\vec{C}_i^{new}/\vec{c}_i]$ is the renaming relative to the thread i), for all R such that $P \mid R$ is a control point, noting that by construction this implies that $\{y \mid \sigma_i t \in T_{P|R}\}$ is not empty.

5 Conclusion

We have studied a class of recursive cryptographic protocols with name generation. Generated names are very important in protocol design as they model, *e.g.*, nonces and fresh keys. We have proposed an original classification of undecidable fragments (theorem 2.8) and introduced a new notion of collapsed semantics tailored to tail-recursive programs (theorem 2.14). Moreover, we have determined a suitable family of set constraints that allows for a conservative analysis of the collapsed semantics in exponential time (proposition 3.1 and theorem 3.3). This is perhaps quite a surprising result—it shows that set constraints can model in a reasonable way the scope of nonces, which contradicts a statement from the conclusion of [CCM01]. We have determined a new decidable class of protocols (theorem 4.3). Finally, we have given examples of cryptographic protocols that fit our syntactic conditions and illustrated how our model can be used to specify secrecy and freshness properties. We expect that our approach can be generalized to handle (i) complex symmetric keys and (ii) other cryptographic primitives beyond the symmetric encryption considered here. Experimentation is needed to design a user friendly specification language and to determine what is the size of the protocols that can be handled.

References

[ALV01] R. Amadio, D. Lugiez, and V. Vanackere. On the symbolic reduction of processes with cryptographic functions. Technical report, RR 4147, INRIA, Sophia-Antipolis, 2001. Under revision for *Theoretical Computer Science*. Abstract appeared in Proc. Logical Aspects of Cryptographic Protocols Verification, Electronic Notes in Theoretical Computer Science, 55(1), 2001.

[AM01] R. Amadio, C. Meyssonnier. On the decidability of fragments of the asynchronous π-calculus. Proc. EXPRESS01, Electronic Notes in Theoretical Computer Science, 52.1, 2001. Also appeared as RR-INRIA 4241.

[CP97] W. Charatonik and A. Podelski. Set constraints with intersection. In Glynn Winskel, editor, *Twelfth Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 362–372. IEEE, June 1997.

[CPT00] W. Charatonik, A. Podelski, and J.-M. Talbot. Paths vs. trees in set-based program analysis. In *27th Annual ACM Symposium on Principles of Programming Languages*, pages 330–338, Jan. 2000.

[CCM01] H. Comon, V. Cortier, and J. Mitchell. Tree automata with one memory, set constraints, and ping-pong protocols. In *Proc. ICALP*, pages 682–693. Springer Lecture Notes in Comp. Sci. 2076, 2001.

[CJ97] J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0. Unpublished manuscript available at http://www-users.cs.york.ac.uk/~jac/papers/drareview.ps.gz, 1997.

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

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

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

[GK00] T. Genet and F. Klay. Rewriting for cryptographic protocol verification. In Proc. *CADE*, Springer Lecture Notes in Comp. Sci. 1831, 2000.

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

[HJ90] N. Heintze and J. Jaffar. A decision procedure for a class of set constraints (extended abstract). In *Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 42–51, 1990.

[HJ94] N. Heintze and J. Jaffar. Set constraints and set-based analysis. In *Proceedings of the Workshop on Principles and Practice of Constraint Programming*, volume 874 of *LNCS*, pages 281–298. Springer-Verlag, 1994.

[Mon99] D. Monniaux. Abstracting cryptographic protocols with tree automata. In *Proc. Static Analysis Symposium, Springer Lect. Notes in Comp. Sci.*, 1999.

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

[Sto99] S. Stoller. A bound on attacks on authentication protocols. Technical Report 526, Indiana University, CS Dept., july 1999.

[TDT00] J.-M. Talbot, Ph. Devienne, and S. Tison. Generalized definite set constraints. *Constraints: An International Journal*, 5(1-2):161–202, January 2000.

[Wei99] C. Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In *Proc. CADE 99*. Springer Lect. Notes in Comp. Sci. (LNAI) 1632, 1999.

A Proofs

A.1 2-counter machines

We assume that a 2-counter machine contains instructions of the form ($k \in \{1,2\}$):

$$\begin{array}{lll} (1) & q & : N_k := N_k + 1;\ \mathit{goto}\ q' \\ (2) & q & : (N_k = 0) \to \mathit{goto}\ q', N_k := N_k - 1; \mathit{goto}\ q'' \end{array}$$

where N_1, N_2 denote the two counters. An instruction of type (1) increments the counter k and jumps to another point of the control. An instruction of type (2), tests whether the counter N_k is 0, and if it is the case it jumps to a control point q', otherwise it decrements the counter and jumps to control point q''.

A.2 Proof of theorem 2.8

We give three different encodings of 2-counter machine so that the problem of determining whether the initial configuration $(q_o, 0, 0)$ of the 2-counter machines reaches a configuration (q_F, n, m), for some n, m, reduces to the control reachability problem considered here.

Each encoding violates one of the conditions (LINEARITY, LOCALITY, INDEPENDENCE) while satisfying the other two plus the DATA OR KEY condition. In the following encodings we assume that the operations act on the first counter ($k = 1$). The case $k = 2$ is quite similar.

For every state q of the 2-counter machine we reserve a constant q. To represent the counters, we reserve the constants $S, 0, C, N_1, N_2$. The natural number n is represented by the message $\underline{n} \equiv S^n 0C$.

$\neg$LINEARITY A reachable configuration (q, n, m) is stored in the knowledge of the adversary as a pair of messages $q\langle C', N_1\underline{n}\rangle, q\langle C', N_2\underline{m}\rangle$ where C' is a fresh constant that is used to bind together the two messages. Thus if q_o is the initial state of the 2-counter machine, the thread

$$\nu c\, !q_o\langle c, N_1\underline{0}\rangle.!q_o\langle c, N_2\underline{0}\rangle.0$$

starts the computation by giving to the adversary the knowledge of the initial configuration $(q_o, 0, 0)$.

For every instruction of type (1) we introduce the equation:

$$\begin{array}{ll} A_q & = ?q\langle x_1, N_1 y\rangle.?q\langle x_2, N_2 z\rangle. \\ & \nu d_1, d_2\, !d_1 x_1.!d_2 x_2.?\langle d_1 x, d_2 x\rangle. \\ & \nu c\, !q'\langle c, N_1 S y\rangle.!q'\langle c, N_2 z\rangle.A_q\ . \end{array}$$

For every instruction of type (2) we introduce two equations:

$$\begin{aligned}
A^1_q \quad &= ?q\langle x_1, N_1 0y\rangle.?q\langle x_2, N_2 z\rangle.\\
&\nu d_1, d_2\ !d_1 x_1.!d_2 x_2.?\langle d_1 x, d_2 x\rangle.\\
&\nu c\ !q'\langle c, N_1 0y\rangle.!q'\langle c, N_2 z\rangle.A^1_q\ .
\end{aligned}$$

$$\begin{aligned}
A^2_q \quad &= ?q\langle x_1, N_1 Sy\rangle.?q\langle x_2, N_2 z\rangle.\\
&\nu d_1, d_2\ !d_1 x_1.!d_2 x_2.?\langle d_1 x, d_2 x\rangle.\\
&\nu c\ !q'\langle c, N_1 y\rangle.!q'\langle c, N_2 z\rangle.A^2_q\ .
\end{aligned}$$

We rely on the non-linearity of the filter to check the equality of x_1 and x_2.

$\neg$LOCALITY Basically, the non-locality of the filter gives another way to code equality. In this case, a reachable configuration (q, n, m) is stored in the knowledge of the adversary as a triple of messages $qC', N_1\langle C', \underline{n}\rangle, N_2\langle C', \underline{m}\rangle$, where C' is a fresh constant that is used to bind together the three messages. The initial knowledge of the adversary is provided by the following thread $\nu c\,!q_o c.!N_1\langle c, \underline{0}\rangle !N_2\langle c, \underline{0}\rangle.0$. For every instruction of type (1) we introduce the equation:

$$\begin{aligned}
A_q \quad &= ?qx.?N_1\langle x, y\rangle.?N_2\langle x, z\rangle.\\
&\nu x'\ !q'x'.!N_1\langle x', Sy\rangle.!N_2\langle x', z\rangle.A_q\ .
\end{aligned}$$

Instructions of type (2) are coded in a similar way.

$\neg$INDEPENDENCE A configuration (q, n, m) is stored in the knowledge of the adversary as a message $q\langle \underline{n}, \underline{m}\rangle$. For every instruction of type (1) we introduce the equation:

$$A_q \quad = ?q\langle x, y\rangle.!q'\langle Sx, y\rangle.A_q$$

Instructions of type (2) are coded in a similar way. The thread $!q_o\langle \underline{0}, \underline{0}\rangle.0$ initializes the computation. This encoding does not rely on name generation. □

A.3 Proof of proposition 2.12

In proofs, it is convenient to use the following *equivalent* iterative definitions of $S(T)$ and $A(T)$:

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

(1) Suppose (R, T) is a standard configuration and that $(R, T) \to (R_1, T_1)$. We show that $\tau_R(R, T) \to \tau_{R_1}(R_1, T_1)$ by case analysis on the definition 2.4 of standard reduction.

(unfold) Let $R \equiv (A_i, n_i) \mid R'$ and suppose $(R, T) \to (R_1, T)$ where $R_1 \equiv (\sigma_i^{n_i+1}(Q_i), n_i + 1) \mid R'$. By definition of collapsed semantics, we have:

$$(A_i \mid \tau_R(R'), \tau_R(T)) \to (\sigma_i^{new}(Q_i) \mid \sigma_i^{old}(\tau_R(R')), \sigma_i^{old}(\tau_R(T)))\ .$$

Then we have to check:

$$\begin{aligned}
&(1) \quad \tau_{R_1}(\sigma_i^{n_i+1}(Q_i)) = \sigma_i^{new}(Q_i),\\
&(2) \quad \tau_{R_1}(R') = \sigma_i^{old}(\tau_R(R')),\\
&(3) \quad \tau_{R_1}(T) = \sigma_i^{old}(\tau_R(T)))\ .
\end{aligned}$$

We note that the substitutions τ_R, τ_{R_1} act only on fresh constants $\vec{C}^j_h$ and that they only differ on the constants $\vec{C}^{n_i}_i, \vec{C}^{n_i+1}_i$.

(1) The only fresh constants that can occur in $\sigma_i^{n_i+1}(Q_i)$ are $\vec{C}_i^{n_i+1}$. Then we observe that:

$$\tau_{R_1}([\vec{C}_i^{n_i+1}/\vec{c}_i]\vec{c}_i) \quad = \tau_{R_1}(\vec{C}_i^{n_i+1}) \quad = \vec{C}_i^{new} \quad = \sigma_i^{new}(\vec{c}_i) \ .$$

(2) We note that the constants $\vec{C}_i^{n_i+1}$ do not occur in R'. Then τ_{R_1} differs from τ_R only on $\vec{C}_i^{n_i}$ which is mapped to $\vec{C}_i^{old}$ by the former and to $\vec{C}_i^{new}$ by the latter. Thus $\tau_{R_1}(R') = \sigma_i^{old}(\tau_R(R'))$. The same argument applies to (3).

(i/o) Suppose $R \equiv (?t.!s.P, n_i) \mid R'$, $\theta t \in S(A(T))$, $\theta s \in \mathcal{M}$, and $(R,T) \rightarrow (R_1, T \cup \{\theta s\})$ where $R_1 \equiv (\theta P, n_i) \mid R'$.

We note that $\tau_R = \tau_{R_1}$. Next, we prove that for any $t \in A(T)$, $\tau_R t \in A(\tau_R T)$. We use the inductive definition of analysis.

- If $t \in T$ then $\tau_R(t) \in \tau_R T \subseteq A(\tau_R T)$.
- Suppose $\langle t_1, t_2\rangle \in A(T)$ and, inductively, $\tau_R\langle t_1, t_2\rangle \equiv \langle \tau_R t_1, \tau_R t_2\rangle \in A(\tau_R T)$. Then $t_i \in A(T)$ and $\tau_R t_i \in A(\tau_R T)$, for $i = 1, 2$.
- Suppose $E(t, C), C \in A(T)$ and, inductively, $\tau_R E(t, C) \equiv E(\tau_R t, \tau_R C), \tau_R C \in A(\tau_R T)$. Then $t \in A(T)$ and $\tau_R t \in A(\tau_R T)$.

Then, using the inductive definition of synthesis above we show

$$t \in S(A(T)) \ \Rightarrow \ \tau_R t \in S(A(\tau_R T)) \ . \tag{2}$$

Now consider the collapsed configuration

$$\tau_R(R,T) = (\tau_R(?t.!s.P) \mid \tau_R R', \tau_R T) \ .$$

If $\theta = [t_1/x_1, \ldots, t_n/x_n]$ with $x_1, \ldots, x_n$ filter variables in t then we set $\theta' = [\tau_R t_1/x_1, \ldots, \tau_R t_n/x_n]$. Then by implication (2) we have:

$$\theta t \in S(A(T)) \ \Rightarrow \ \tau_R(\theta t) = \theta'(\tau_R t) \in S(A(\tau_R T)) \ .$$

Moreover $\theta'(\tau_R s) = \tau_R(\theta s) \in \mathcal{M}$ iff $\theta s \in \mathcal{M}$. Therefore

$$\tau_R(R,T) \rightarrow (\theta'(\tau_R P) \mid \tau_R R', \tau_R T \cup \{\theta' \tau_R s\}) \equiv \tau_R((\theta P, n_i) \mid R', T \cup \{\theta s\}) \ .$$

(2) It follows immediately by diagram chasing and the definition of τ_R. □

A.4 Proof of proposition 2.13

We give three different examples where the collapsed semantics is *not* precise. Each example violates one of the conditions LINEARITY, LOCALITY, DATA OR KEY while satisfying the other two plus the INDEPENDENCE condition.

We start with an example concerning the violation of the DATA OR KEY condition. We consider the parallel composition of the threads (we write repeated encryptions as a word):

$$\begin{array}{ll}
A_1 & = \nu b\, ?D_0 x.!D_1\langle b, x\rangle.A_1 \\
A_2 & = \nu c_1, c_2\, !D_0 c_1.?D_1\langle y, c_1\rangle.!yE_1.!Dy. \\
 & \quad !D_0 c_2.?D_1\langle z, c_2\rangle.!zE_2.!Dz.0 \\
A_3 & = ?Dx.!x.0 \\
A_4 & = ?E_1.?E_2.\mathsf{err}
\end{array}$$

Note that we do not satisfy condition DATA OR KEY because the generated names occur in data position and in A_2 the filter variables y, z occur in key position. In the concrete semantics, A_1 receives requests for a fresh key along D_0 and replies to them along D_1. A_2 gets two keys from A_1 and uses them to encrypt E_1 and E_2. Because of the nonces c_1, c_2, we are sure that the two keys got by A_2 are *different*. A_2 uses the two keys to encrypt E_1 and E_2 and it encrypts them with D. A_3 reveals exactly one key and consequently the adversary may learn either E_1 or E_2. In the collapsed semantics, the two keys get mapped to the same constant and therefore the adversary may learn E_1 and E_2 thus making err reachable.

Simple variants of the system above show that without the conditions LINEARITY and LOCALITY the collapsed semantics is not precise. For LINEARITY consider:

$$\begin{array}{ll} A_1 & = \nu b\, ?D_0 x.!D_1\langle b, x\rangle.A_1 \\ A_2 & = \nu c_1, c_2\, !D_0 c_1.?D_1\langle y, c_1\rangle.!E_1 y. \\ & \quad !D_0 c_2.?D_1\langle z, c_2\rangle.!E_2 z.0 \\ A_3 & = ?\langle E_1 x, E_2 x\rangle.\mathsf{err}\ , \end{array}$$

and for LOCALITY, replace the third equation by:

$$A_3 \quad = ?E_1 x.?E_2 x.\mathsf{err}\ .\ \Box$$

A.5 Proof of theorem 2.14

PROOF. Implication ($\Rightarrow$) follows by proposition 2.12. We prove the implication ($\Leftarrow$).

- We start with a general remark: in a reachable configuration (R, T) with $R \equiv (?t.!s.P_i, n_i) \mid R'$ the fresh constants occurring in the filter t can only be in $\vec{C}_i^{n_i}$ because of the LOCALITY condition. Moreover, the term t is linear because of condition LINEARITY. Note that no collapse arises when we apply the substitution τ_R to t.

- We claim that the standard reduction relation maintains the invariant that in a reachable configuration (R, T):

(1) If a generated name c_i is in ***key position*** then all occurrences of the related constants C_i^j in (R, T) are in key position.

(2) If a generated name c_i is in *data position* and for some j a corresponding constant C_i^j occurs in key position in (R, T) then $C_i^j \in A(T)$.

Conditions (1-2) hold in the initial configuration (R_0, T_0) as the constants C_i^j do not occur in it and the substitution τ_R is the identity.

(unfold) Suppose $R \equiv ((A_h, n_h) \mid R'')$ and $(R, T) \to (R', T)$ with $R' \equiv (\sigma_h^{n_h+1} Q_h, n_h + 1) \mid R''$.

(1) Constants C_i^j that occur in R'', T satisfy condition (1) by inductive hypothesis. The only constants C_i^j that can occur in $\sigma_h^{n_h+1} Q_h$ are $\vec{C}_h^{n_h+1}$ and they occur in the position of the corresponding name generators $\vec{c}_h$.

(2) Suppose the generated name c_i is in data position. If C_i^j is in key position in R'', T then $C_i^j \in A(T)$ by inductive hypothesis. On the other hand, we cannot find constants C_i^j in key position in $\sigma_h^{n_h+1} Q_h$.

$(\mathsf{i/o})$ Suppose $R \equiv (?t.!s.P, n_h) \mid R''$ and $(R, T) \to (R', T \cup \{\theta s\})$ with $R' \equiv (P, n_h) \mid R''$, $\theta t \in S(A(T))$, and $\theta s \in \mathcal{M}$ (because of the LOCALITY condition, P is not affected by θ).

(1) Suppose c_i is in key position. Then by inductive hypothesis all occurrences of C_i^j in (R, T) are in key position. Suppose $Var(t) = \{x_1, \ldots, x_m\}$ and $\theta = [t_1/x_1, \ldots, t_m/x_m]$. If C_i^j occurs in t_l then it is in key position because x_l is in data position in t and $\theta t \in S(A(T))$. Then the occurrences of C_i^j in θs satisfy condition (1).

(2) Suppose c_i is in data position. If C_i^j is in key position in s, P, R'', T then $C_i^j \in A(T) \subseteq A(T \cup \{\theta s\})$ by inductive hypothesis. Otherwise, suppose $Var(t) = \{x_1, \ldots, x_m\}$, $\theta = [t_1/x_1, \ldots, t_m/x_m]$, and C_i^j occurs in key position in t_h. This means that $E(t', C_i^j)$ is a subterm of $\theta t \in S(A(T))$. If $C_i^j \notin A(T)$ then $E(t', C_i^j)$ must be a subterm of $A(T)$ and of T. But then C_i^j would occur in T in key position and by inductive hypothesis $C_i^j \in A(T)$. Contradiction!

• Next we use properties (1-2) to show that in a configuration (R,T) reachable in the standard semantics the following holds:

$$A(\tau_R T) = \tau_R A(T) \text{ and } S(A(\tau_R T)) = \tau_R S(A(T)) \ . \tag{3}$$

$\tau_R A(T) \subseteq A(\tau_R T)$ We show by induction on n that $\tau_R A(T)_n \subseteq A(\tau_R T)$.

$A(\tau_R T) \subseteq \tau_R A(T)$ We show by induction on n that $A(\tau_R T)_n \subseteq \tau_R A(T)$. The interesting case arises with encryption. Suppose $E(t,C), C \in A(\tau_R T)_n$. By inductive hypothesis, there are $E(t_1,C'), C'' \in A(T)$ such that $\tau_R t_1 \equiv t$ and $\tau_R C' \equiv \tau_R C'' \equiv C$. Two cases can arise:

$C' \equiv C''$ Then $t_1 \in A(T)$ and $\tau_R t_1 \in \tau_R A(T)$.

$C' \not\equiv C''$ Since $\tau_R C' \equiv \tau_R C''$ it must be the case that $C' \equiv C_i^j$, $C'' \equiv C_i^{j'}$, and $j, j' < n_i$. This means that the name generator c_i occurs in data position in the system and that C_i^j occurs in key position in $A(T)$, hence in T. Then, by condition (2), $C_i^j \in A(T)$, and therefore $t_1 \in A(T)$ and $\tau_R t_1 \in \tau_R A(T)$.

$S(A(\tau_R T)) = \tau_R S(A(T))$ By the results above, it is enough to show $S(\tau_R A(T)) = \tau_R S(A(T))$. We show the two inclusions, by induction on the iterative definition of the synthesis operator.

• Finally, we show that if (R,T) is a reachable configuration and $\tau_R(R,T) \to (R'',T'')$ then $(R,T) \to (R',T')$ and $\tau_{R'}(R',T') \equiv (R'',T'')$.

(unfold) Suppose $R \equiv ((A_i,n_i) \mid R_1, T)$ and

$$\tau_R(R,T) \equiv (A_i \mid \tau_R R_1, \tau_R T) \to (\sigma_i^{new} Q_i \mid \sigma_i^{old} \tau_R R_1, \sigma_i^{old} \tau_R T) \ .$$

Then $(R,T) \to ((\sigma_i^{n_i+1} Q_i, n_i+1) \mid R_1, T) \equiv (R',T)$ and the assertion follows by checking $\tau_{R'} \sigma_i^{n_i+1} Q_i \equiv \sigma_i^{new} Q_i$, $\tau_{R'} R_1 \equiv \sigma_i^{old} \tau_R R_1$, and $\tau_{R'} T \equiv \sigma_i^{old} \tau_R T$. The arguments already developed for the proof of the simulation proposition 2.12 apply.

(i/o) Suppose $R \equiv (?t.!s.P, n_i) \mid R_1$ and

$$\tau_R(R,T) \to (\tau_R P \mid \tau_R R_1, \tau_R T \cup \{\theta \tau_R s\})$$

where $\theta \tau_R t \in S(A(\tau_R T))$ and $\theta \tau_R s \in \mathcal{M}$.

Since $S(A(\tau_R T)) = \tau_R S(A(T))$ by (3), there must be $t' \in S(A(T))$ such that $\theta \tau_R t = \tau_R t'$. Suppose $Var(t) = \{x_1, \ldots, x_m\}$ and $\theta = [t_1/x_1, \ldots, t_m/x_m]$.

Since τ_R is injective on t, there is a substitution $\theta' = [t_1'/x_1, \ldots, t_m'/x_m]$ such that $t' \equiv \theta' t$ and $\tau_R t_i' \equiv t_i$ for $i = 1, \ldots, m$. Moreover, we note that $\theta \tau_R s \in \mathcal{M}$ iff $\theta' s \in \mathcal{M}$. We then observe that $(R,T) \to (R', T \cup \{\theta' s\})$ with $R' \equiv (P, n_i) \mid R_1$. Since $\tau_{R'} = \tau_R$ it is enough to check that $\tau_R \theta' s \equiv \theta \tau_R s$. □

A.6 Proof of proposition 3.1

(1) We proceed by induction on the length of the reduction $(R_0,T_0) \xrightarrow{*} (R,T)$.

(Base case) We know $T_0 \subseteq \alpha(T_{R_0})$ by the constraint (init). By monotonicity of synthesis and analysis it follows that $S(A(T_0)) \subseteq S(A(\alpha(T_{R_0})))$, and by the constraints (A_{1-3}) and (S_{1-2}) we derive $S(A(\alpha(T_{R_0}))) \subseteq \alpha(T_{R_0})$.

(unfold) Suppose $(R_0,T_0) \xrightarrow{*} (A_i \mid R, T) \to (\sigma_i^{new} Q_i \mid R, \sigma_i^{old} T)$. Then:

$$\begin{array}{lll} (a) & T \subseteq S(A(T)) \subseteq \alpha(T_{A_i|R}) & \text{(by induction hypothesis)} \\ (b) & \sigma_i^{old} \alpha(T_{A_i|R}) \subseteq \alpha(T_{\sigma_i^{new} Q_i|R}) & \text{(by the (unfold) constraint).} \end{array}$$

By (a) and (b) it follows:

$$\sigma_i^{old} T \subseteq \sigma_i^{old} \alpha(T_{A_i|R}) \subseteq \alpha(T_{\sigma_i^{new} Q_i|R}) \ .$$

Then by monotonicity of synthesis and analysis and constraints (A_{1-3}) and (S_{1-2}) it follows:

$$S(A(\sigma_i^{old}T)) \subseteq S(A(\alpha(T_{\sigma_i^{new}Q_i|R}))) \subseteq \alpha(T_{\sigma_i^{new}Q_i|R}) \ .$$

(i/o) Suppose $R \equiv ?t.!s.R_1 \mid R_2$, $R' \equiv R_1 \mid R_2$, and

$$(R_0, T_0) \xrightarrow{*} (R, T) \to (R', T \cup \{\theta s\}),$$

where $\theta t \in S(A(T))$ and $\theta s \in \mathcal{M}$. From $\theta s \in \mathcal{M}$ we know that $\theta = \theta' \circ \eta$ where $\eta : KVar(s) \to \mathcal{N}$ and $\theta' : Var(t) \backslash KVar(s) \to \mathcal{M}$. We set $\{x_1, \ldots, x_n\} = Var(s) \backslash KVar(s)$ and $S_i \equiv \{x_i \mid \eta t \in T_R\}$. We observe:

(a)	$\theta t = \theta'(\eta t) \in S(A(T)) \subseteq \alpha(T_R)$	by inductive hypothesis,
(b)	$\theta'(x_i) \in [\![S_i]\!]_\alpha$	as $[\theta'(\vec{x})/\vec{x}]\eta t \in S(A(T))$,
(c)	$[\![\{y \mid \eta t \in T_R\}]\!]_\alpha \neq \emptyset$	
(d)	$T \subseteq \alpha(T_R) \subseteq \alpha(T_{R'})$	by (c) and constraint $(\mathsf{i/o}_1)$
(e)	$\theta s = \theta' \eta s \in [\![[\vec{S}/\vec{x}]\eta s]\!]_\alpha \subseteq \alpha(T_{R'})$	by (b,c) and constraint $(\mathsf{i/o}_2)$.

From (d) and (e) we know $T \cup \{\theta s\} \subseteq \alpha(T_{R'})$ and we conclude by applying as above the constraints on synthesis and analysis.

(2) If $(R_0, T_0) \xrightarrow{*} (\mathsf{err} \mid R, T)$ then $\emptyset \neq S(A(T_0)) \subseteq S(A(T))$ since $T_0 \neq \emptyset$ and the knowledge of the adversary can only grow. Moreover, $S(A(T)) \subseteq \alpha(T_{\mathsf{err}|R})$ by (1), and $\alpha(T_{\mathsf{err}|R}) \subseteq \alpha(T_{\mathsf{err}})$ by the constraint (err). Putting all together we conclude:

$$\emptyset \neq S(A(T_0)) \subseteq S(A(T)) \subseteq \alpha(T_{\mathsf{err}|R}) \subseteq \alpha(T_{\mathsf{err}}) \ . \ \Box$$

A.7 Proof of theorem 3.3

To prove theorem 3.3 we need soundness and completeness results for the solving algorithm; they are stated as lemmas below. The soundness result (lemma A.1) is formulated for arbitrary (renaming compatible) valuations, while the completeness (lemma A.3) is restricted to the least solution. This is because the implications in our rules are not equivalences. For example in the case of rule 6 the inverse implication does not hold for arbitrary valuations.

Lemma A.1 *Let α be a renaming compatible valuation over Ξ'. Then, for every rule in table 2, if α satisfies the premise of the rule then it also satisfies the conclusion of the rule.*

Proof. The proof is a case analysis by inspection of each rule. Rules 2 and 6 rely on the linearity of constraints while rule 9 requires that the valuation is renaming compatible. The other cases are immediately verified.

rule 2 Because $f(t_1, \ldots, t_n)$ is linear $\{Var(t_i)\}_{i=1,\ldots,n}$ is a partition of $Var(f(t_1, \ldots, t_n))$. If $Var(t_i) = \{\vec{x}_i\}$ then we know that $\exists\, \vec{s}_i\ [\vec{s}_i/\vec{x}_i]t_i \in \alpha(X_i)$ and

$$f([\vec{s}_1/\vec{x}_1]t_1, \ldots, [\vec{s}_n/\vec{x}_n]t_n) \in [\![f(X_1, \ldots, X_n)]\!]_\alpha \subseteq \alpha(X) \ .$$

Then we can conclude that $[\![\{x \mid f(t_1, \ldots, t_n) \in X\}]\!]_\alpha \neq \emptyset$.

rule 6 Because we deal with linear constraints, we can partition the variables in $f(t_1, \ldots, t_i[x], \ldots, t_n)$ as follows: $\{x, \vec{y}_i\} = Var(t_i)$ and $\{\vec{y}_j\} = Var(t_j)$ for $j \neq i$. Suppose s is a term such that $\exists\, \vec{s}_i\ [s/x, \vec{s}_i/\vec{y}_i]t_i \in \alpha(X_i)$. We know $\exists\, \vec{s}_j\ [\vec{s}_j/\vec{y}_j]t_j \in \alpha(X_j)$ for $j \neq i$. Then

$$f([\vec{s}_1/\vec{y}_1]t_1, \ldots, [s/x, \vec{s}_i/\vec{y}_i]t_i, \ldots [\vec{s}_n/\vec{y}_n]t_n) \in [\![f(X_1, \ldots, X_n)]\!]_\alpha \subseteq \alpha(X) \ ,$$

and we observe:

$$s \in [\![\{x \mid f(t_1, \ldots, t_i[x], \ldots, t_n) \in X\}]\!]_\alpha \subseteq \alpha(Y) \ .$$

rule 9 By renaming compatibility, $\alpha(X_i^\sigma) = \sigma(\alpha(X_i))$ and $\alpha(X^\sigma) = \sigma(\alpha(X))$. We observe that:

$$\begin{aligned}
[\![\sigma(f)(X_1^\sigma, \ldots, X_n^\sigma)]\!]_\alpha &= \{\sigma(f)(t'_1, \ldots, t'_n) \mid \exists\, t_1, \ldots, t_n \ t'_i = \sigma(t_i), t_i \in \alpha(X_i), i = 1, \ldots, n\} \\
&= \sigma\{f(t_1, \ldots, t_n) \mid t_i \in \alpha(X_i), i = 1, \ldots, n\} \\
&\subseteq \sigma(\alpha(X)) = \alpha(X^\sigma) \ .
\end{aligned}$$

□

Lemma A.2 *Let α be the least solution of Φ'^S. Then for all expressions S occurring in Φ'^C, if $[\![S]\!]_\alpha \neq \emptyset$ then $\mathsf{ne}(S) \notin \Phi'^C$.*

Proof. Suppose $[\![S]\!]_\alpha \neq \emptyset$. There are three cases depending on the syntax of S.

(a) S is a variable, say X. The proof goes by induction on the definition of α. In the base case there is a constraint $a \subseteq X \in \Phi'^S$ for some constant symbol a, so by rules 1 and 3, $\mathsf{ne}(X) \in \Phi'^C$. In the induction step, we have $f(X_1, \ldots, X_n) \subseteq X \in \Phi'^S$ such that $\alpha(X_i) \neq \emptyset$ for $i = 1, \ldots, n$. By induction hypothesis $\mathsf{ne}(X_i) \in \Phi'^C$, and again by application of rules 1 and 3, $\mathsf{ne}(X) \in \Phi'^C$.

(b) $S = f(X_1, \ldots, X_n)$. The result follows by case (a) above and rule 1.

(c) $S = \{x \mid t \in X\}$. The result follows by induction on the structure of t using rule 4 together with case (a) in the base case and rules 2 and 8 together with the definition of α in the induction step. □

Lemma A.3 *The least solutions of Φ'^S and Φ'^C coincide.*

Proof. Clearly every solution of Φ'^C is a solution of Φ'^S, so it is enough to show that the least solution of Φ'^S is a solution of Φ'^C. Let α be the least solution of Φ'^S. First observe that by lemma A.2 all constraints in Φ'^C involving at least one empty expression (that is, an expression whose nonemptiness is not in Φ'^C) are trivially satisfied. Hence it is enough to show that all constraints in Φ'^C involving only nonempty expressions are satisfied. Moreover, by rule 8, it is enough to show it only for constraints of the form $S \subseteq Y$ where $\mathsf{ne}(S) \in \Phi'^C$. There are two types of such constraints that are in Φ'^C but not in Φ'^S, corresponding to the cases where S is a variable and a membership expression.

(i) S is a variable, say $S = X$. We have to show that for all t, if $t \in \alpha(X)$ then $t \in \alpha(Y)$. By the definition of α there exists a constraint $f(X_1, \ldots, X_n) \subseteq X \in \Phi'^S$ such that $t \in [\![f(X_1, \ldots, X_n)]\!]_\alpha$. By rule 5 $f(X_1, \ldots, X_n) \subseteq Y \in \Phi'^S$ which shows that $t \in \alpha(Y)$.

(ii) S is a membership expression, say $S = \{x \mid t \in X\}$. The proof goes by induction on the structure of t. In the base case $t = x$ (recall our convention that the variable x must occur in t). Then, for an arbitrary term s, if $s \in [\![\{x \mid x \in X\}]\!]_\alpha$ then $s \in \alpha(X)$. By rule 7, $X \subseteq Y \in \Phi'^S$, and by the case (i) above, $s \in \alpha(Y)$.

In the induction step we have that $t = f(t_1, \ldots, t_i[x], \ldots, t_n)$. Suppose $s \in [\![\{x \mid t \in X\}]\!]_\alpha$. Then there exists a substitution θ such that $s = \theta(x)$ and $\theta(t) \in \alpha(X)$. By the definition of α there exists a constraint $f(X_1, \ldots, X_n) \subseteq X \in \Phi'^C$ such that $\theta(t) \in [\![f(X_1, \ldots, X_n)]\!]_\alpha$. Therefore, for all $i = 1, \ldots, n$, $\alpha(X_i) \neq \emptyset$. By lemma A.2 and rules 6 and 8, $\{x \mid t_i[x] \in X_i\} \subseteq Y \in \Phi'^C$. By induction hypothesis, $s \in \alpha(Y)$. □

Lemma A.4 *The least solution of Φ'^C is renaming compatible.*

Proof. Let α be the least solution of Φ'^C. We show $t \in \alpha(X) \Rightarrow \sigma(t) \in \alpha(X^\sigma)$ by induction on the definition of α (cf. section 3.1.2) using rule 9.

In the other direction, we show again by induction on the definition of α that if $t \in \alpha_n(X, u \cup \{\sigma\})$ then $\exists\, t' \in \alpha_n(X, u)\ \sigma t' = t$ (hence $\alpha(X, u \cup \{\sigma\}) \subseteq \sigma\alpha(X, u)$). To this end, we note that by construction of Φ'^C, if $f((X_1, u_1), \ldots, (X_n, u_n)) \subseteq (X, u \cup \{\sigma\}) \in \Phi'^C$ then for some g, u'_i, $f = \sigma(g)$, $u_i = u'_i \cup \{\sigma\}$, and $g((X_1, u'_1), \ldots, (X_n, u'_n)) \subseteq (X, u) \in \Phi'^C$. □

PROOF OF THEOREM 3.3(1). First we fix some notation. If $\alpha : \Xi \to 2^{\Sigma}$ is a valuation over Ξ then we define $ext(\alpha) : \Xi \times 2^{Sub} \to 2^{\Sigma}$ as

$$ext(\alpha)(Y, u) = (u)(\alpha(Y)).$$

On the other hand, if $\beta : \Xi \times 2^{Sub} \to 2^{\Sigma}$ is a valuation over Ξ' then we define $res(\beta) : \Xi \to 2^{\Sigma}$ as

$$res(\beta)(Y) = \beta(Y, \emptyset)\ .$$

Let α_0 be the least solution of Φ'^S and let $\leq$ be the order on valuations defined by pointwise inclusion.

First, observe that if a valuation β is a renaming compatible solution of Φ'^C then $res(\beta)$ is a solution of Φ. This together with lemmas A.4 and A.3 implies that $res(\alpha_0)$ is a solution of Φ. We have to show that any other solution of Φ is greater than $res(\alpha_0)$.

Take any solution α of Φ. Clearly, $ext(\alpha)$ is renaming compatible and is a solution of Φ'. By lemma A.1, $ext(\alpha)$ is a solution of Φ'^C and Φ'^S, so $\alpha_0 \leq ext(\alpha)$. The operation res is monotone, so $res(\alpha_0) \leq res(ext(\alpha))$. Since $res(ext(\alpha)) = \alpha$, we have $res(\alpha_0) \leq \alpha$. □

PROOF OF THEOREM 3.3(2). Obviously, if Φ has size n then it can contain at most n variables. If the system Φ has k renamings and n variables then the system Φ' depends on $2^k n$ variables. Thus we need to show that Φ'^C can be computed in time polynomial in $2^k n$.

Every membership expression is uniquely identified by two variables (an individual and a set one) and a term that occurs in Φ', therefore there are polynomially (in the number $2^k n$) many membership expressions. We assume that the signature Σ is fixed, so the arity of function symbols is bounded and there are only polynomially (in $2^k n$) many expressions $f(X_1, \ldots, X_n)$, so polynomially many set expressions up to renaming of the comprehension variables. There are at most 2^k times more expressions at all, and thus polynomially many inclusions. Every conditional inclusion either occurs in Φ' or is introduced by an application of rule 6 (and uniquely identified by a pair of inclusions). Hence every conjunct in Φ'^C comes from a set of polynomial (in $2^k n$) size. There are no more iterations of the fixpoint algorithm than conjuncts in Φ'^C, and every iteration does a polynomial work. □

A.8 Proof of theorem 4.3

Consider a system of iterated threads (1) satisfying the INDEPENDENCE condition. We apply a series of 4 transformations to the system that do not affect the control reachability problem.

A.8.1 Transformation (1)

We transform all recursive equations of the shape $A = ?t_1.!s_1.(A \mid ?t_2.!s_2 \ldots ?t_n.!s_n.\mathsf{err})$ into

$$A = ?t_1.!s_1.(A \mid ?t_2.!s_2 \ldots ?t_n.!\langle s_n, \mathsf{err}\rangle.0)$$

where we take err as a distinct fresh constant. Then the control reachability problem amounts to determine whether there is a reachable configuration (R, T) such that $\mathsf{err} \in S(A(T))$.

A.8.2 Transformation (2)

After step (1), all equations have the shape:

$$A = ?t_1.!s_1.(A \mid ?t_2.!s_2 \ldots ?t_n.!s_n.0) \tag{4}$$

For every such equation we introduce n tail-recursive equations with fresh identifiers $A_1, \ldots, A_n$ and exactly one alternation of input-output:

$$\begin{array}{ll} A_1 & = ?t_1.!s_1.A_1 \\ A_2 & = ?\langle t_1, t_2\rangle.!\langle s_1, s_2\rangle.A_2 \\ \cdots & = \cdots \\ A_n & = ?\langle t_1, \ldots, t_n\rangle.!\langle s_1, \ldots, s_n\rangle.A_n \end{array}$$

Let (R_0, T_0) be the initial configuration of the system (1). Let R'_0 the control obtained from R_0 by replacing each identifier A with the parallel composition of the fresh identifiers $A_1, \ldots, A_n$. Note that the resulting system has just *one control point* namely R'_0. Therefore we will just write $T \to T'$ rather than $(R'_0, T) \to (R'_0, T')$. We pause to note the following property of systems with one control state.

Lemma A.5 *If $T_0 \xrightarrow{*} T$ and $T_0 \xrightarrow{*} T'$ then $T_0 \xrightarrow{*} T \cup T'$.*

To prove the soundness and completeness of the transformation (2) we proceed as follows. Say that a configuration $(R_0 \mid P_1 \mid \cdots \mid P_m, T)$ is *admissible* if $m \geq 0$ and for $j = 1, \ldots, m$ there is an equation of the shape (4), an index $i \geq 2$, and a substitution θ satisfying $P_j \equiv \theta(?t_i.!s_i \ldots ?t_n.!s_n.0)$, $\theta\langle t_1, \ldots, t_{i-1}\rangle \in S(A(T))$, and $\theta\langle s_1, \ldots, s_{i-1}\rangle \in \mathcal{M}$.

We note that the initial configuration (R_0, T_0) is admissible and that admissible configurations are closed under reduction. Then the following lemma allows to conclude the argument.

Lemma A.6 *If (R, T) is admissible and $(R, T) \to (R', T')$ then $T \to T''$ and $S(A(T')) \subseteq S(A(T''))$. On the other hand, if $T \to T'$ and (R, T) is admissible then $(R, T) \xrightarrow{*} (R', T'')$ and $S(A(T')) \subseteq S(A(T''))$.*

A.8.3 Transformation (3)

As a result of transformation (2) we may assume that each equation has the shape $A = ?t.!s.A$. For each such equation we generate the equations:

$$A_\eta = ?\eta t.!\eta s.A_\eta \quad \eta : KVar(s) \to \mathcal{N}$$

where again A_η are fresh identifiers. Denote with $\to_3$ the reduction relation for the derived system. The following lemma entails the soundness and completeness of this transformation.

Lemma A.7 *$T \to T'$ if and only if $T \to_3 T'$.*

A.8.4 Transformation (4)

Every tail recursive equation of the shape $A = ?t.!s.A$ obtained as a result of the transformation (3) can be further simplified as follows. If $s \equiv \langle s_1, \ldots, s_n\rangle$ where s_i is not a pair then introduce the equations: [8]

$$A_i = ?t.!s_i.A_i, \quad i = 1, \ldots, n$$

with $A_1, \ldots, A_n$ fresh identifiers. Denote with $\to_4$ the reduction relation for the derived system. Then an easy induction on the length of the derivation shows the soundness and completeness of this transformation.

Lemma A.8 *If $T \to_3 T'$ then $T \xrightarrow{*}_4 T''$ and $S(A(T')) = S(A(T''))$. On the other hand, if $T \to_4 T'$ then there exists T'' such that $T \to_3 T''$ and $S(A(T')) \subseteq S(A(T''))$.*

Remark A.9 *Transformations (1), (2), and (4) are polynomial in the size of the system while transformation (3) can be exponential. As pointed out in section 3.2, we expect that more efficient methods to deal with key variables can be found.*

The following lemma specifies what is left of the INDEPENDENCE condition 4.2 after the transformations (1–4) are performed.

[8] This transformation applies even if $s_1, \ldots, s_n$ share variables.

Lemma A.10 *If the system of iterated threads (1) satisfies the* INDEPENDENCE *condition then after application of the transformations (1–4) for every tail recursive equation $A = ?t.!s.A$ in the resulting system the following holds:*

(i) if (1) satisfies LINEARITY *then t is a linear term,*

(ii) s is a term which is not a pair,

(iii) s has no key variables, and

(iv) either s contains at most one variable or it is linear and the variables $Var(s)$ are independent in t.

A.8.5 Constraints generation

We generate set constraints on the system obtained at the end of the transformation (4) according to the following rules that are just a specialization of those presented in table 1. There is just one set variable X corresponding to the unique control state.

$$\begin{array}{lll} (\mathsf{init}) & t \subseteq X & (1) \\ (\mathsf{i/o}) & \text{if } \mathsf{ne}(\{y \mid t \in X\}) \text{ then } [\vec{S}/\vec{x}](s) \subseteq X & (2) \end{array}$$

(1) for all t in the initial configuration T_0
(2) for all equation $A = ?t.!s.A$ where $Var(s) = \{x_1, \ldots, x_n\}, S_i = \{x_i \mid t \in X\}$.
We also add the constraints (A_{1-3}) and (S_{1-2}) in table 1 with T_R replaced by X.

Remark A.11 *We note that if x is independent in t then up to permutations of the tuple, t has the shape $\langle t_1(x), \ldots, t_k(x), t' \rangle$ where t_i are not pairs with only one occurrence of x and no occurrences of other variables, and x does not occur in t'. Moreover, we note that for any set of messages T and any permutation π, $\langle s_1, \ldots, s_n \rangle \in S(A(T))$ iff $\langle s_{\pi(1)}, \ldots, s_{\pi(n)} \rangle \in S(A(T))$.*

A.8.6 Precision analysis

Thus we arrive at the crucial lemma where we apply to the transformed system the independence condition derived in lemma A.10.

Lemma A.12 *Let Eq be the system obtained by applying the transformations (1–4) to a system of iterated threads satisfying the* INDEPENDENCE *condition. Let $\rightarrow$ denote the related reduction relation, let T_0 be the initial configuration, and let α be the least solution of the generated set constraint. Then:*

1. *If $T_0 \xrightarrow{*} T$ then $S(A(T)) \subseteq \alpha(X)$.*
2. *For all $t \in \alpha(X)$ there exists T such that $T_0 \xrightarrow{*} T$ and $t \in S(A(T))$.*
3. *$T_0 \xrightarrow{*} T$ and $\mathsf{err} \in S(A(T))$ if and only if $\mathsf{err} \in \alpha(X)$.*

PROOF. The proof of the implication (1) is the same as in proposition 3.1. The equivalence (3) follows from implications (1) and (2). In the following we prove implication (2).

By lemma A.10 we know that in all tail recursive equations $A = ?t.!s.A$ in the system Eq, s is a term which is not a pair, has no key variable and satisfies:

- either s contains only one occurrence of one variable, or
- s is linear and all variables from $Var(s)$ are independent in t.

The proof goes by induction on the index i in the definition of the least solution α (cf. section 3.1.2).

In the base case, for the constraint (init), it is enough to take $T = T_0$. In the induction step in the case of constraints (A_{1-3}), (S_{1-2}) and (err) the result follows immediately from the induction hypothesis, properties of synthesis and analysis, and lemma A.5. Thus the only interesting case is the constraint $(\mathsf{i/o})$.

Assume that for all $r \in \alpha_i(X)$ there exists T such that $T_0 \xrightarrow{*} T$ and $r \in S(A(T))$, that $[\![\{y \mid t \in X\}]\!]_{\alpha_i} \neq \emptyset$ and that a term r belongs to the set $[\![[\vec{S}/\vec{x}](s)]\!]_{\alpha_i}$. We want to show that there exists T such that $r \in S(A(T))$ and $T_0 \xrightarrow{*} T$. There are two cases.

(1) s contains only one occurrence of one variable x. Then $r \in [\![[\{x \mid t \in X\}/x]s]\!]_{\alpha_i}$ and there exists a substitution θ such that $r = \theta s$ and $\theta t \in \alpha_i(X)$. By the induction hypothesis there exists T' such that $T_0 \xrightarrow{*} T'$ and $\theta t \in S(A(T'))$. By reduction $(\mathsf{i/o})$ we have $T' \to T' \cup \{\theta s\}$, so taking $T = T' \cup \{\theta s\}$ we are done.

(2) s is linear and all variables from $Var(s)$ are independent in t. Let $Var(s) = \{x_1, \ldots, x_k\}$. Then there exist substitutions $\theta_1, \ldots, \theta_k$ such that for $j = 1, \ldots, k$ we have $\theta_j t \in \alpha_i(X)$ and $r = [\theta_1 x_1/x_1, \ldots, \theta_k x_k/x_k]s$. By the induction hypothesis there exist T_j such that $T_0 \xrightarrow{*} T_j$ and $\theta_j t \in S(A(T_j))$. By remark A.11 we may assume that $t = \langle \vec{t_1}(x_1), \ldots, \vec{t_k}(x_k), t'(\vec{y})\rangle$ where $\vec{t_j}(x_j)$ is a tuple of terms each of which contains only one occurrence of one variable x_j. By properties of synthesis and analysis, $\theta_j \vec{t_j}(x_j) \in T_j$. Let $T' = T_1 \cup \ldots \cup T_k$. By lemma A.5, $T_0 \xrightarrow{*} T'$. Define a new substitution θ such that $\theta(x_j) = \theta_j(x_j)$ and $\theta(y) = \theta_1(y)$ for $y \in \vec{y}$. Then, by properties of synthesis and analysis, $\theta t \in S(A(T'))$, and by reduction $(\mathsf{i/o})$ we have $T' \to T' \cup \{\theta s\}$. Since $r = \theta s$, by taking $T = T' \cup \{\theta s\}$ we are done. □

PROOF OF THEOREM 4.3. By the soundness and completeness of the transformations (1–4) and lemma A.12 the control reachability problem is reduced to the question whether $\mathsf{err} \subseteq X$ appears in the solved form of the generated constraint. By remark A.9 the generated constraint is of the size $\mathsf{poly}(n \cdot c^k)$. In the linear case the result follows immediately by theorem 3.3. In the non-linear case, since there are no renaming operations involved, the generated constraint belongs to the class of constraints from [TDT00] and can be solved in exponential time. Finally, the DEXPTIME lower bound for non-linear iterated threads relies on the following proposition A.13. □

Proposition A.13 *The control reachability problem for (non-linear) iterated threads is* DEXPTIME-*hard, even if the system of equations uses neither pairs nor key variables.*

PROOF HINT. The proof is by reduction of the satisfiability problem for unary definite set constraints, which is shown to be DEXPTIME-hard in [CPT00]. Unary definite set constraints are defined by the following grammar

$$\Phi \quad ::= \quad C \subseteq X \mid f(Y) \subseteq X \mid Y \subseteq X \mid Y \cap Z \subseteq X \mid \top \subseteq X \mid X \subseteq C \mid X \subseteq f(Y) \mid \Phi \wedge \Phi$$

where C and f range over constants and unary function symbols from a given signature Σ, the symbol $\top$ denotes the set of all terms over Σ, and X, Y, Z range over the set Ξ of variables.

Given a set constraint Φ over (Σ, Ξ) we will write a system of iterated threads over the signature $\Sigma \cup \Xi \cup \{\top\}$. The initial knowledge consists of all terms $\top(C)$ where C is a constant over Σ. Moreover, for every unary function f, we introduce the thread $?\top(x).!\top f(x)$. Thus the adversary can know all terms $\top t$ for t term over Σ. Each inclusion in Φ is translated to a thread according to the table below. Then each thread of the form $?t.!s$ or $?t_1.?t_2.!s$ is respectively transformed to an equation $A = ?t.!s.A$ or $A = ?t_1.!C.(A \mid ?t_2.!s.0)$ where A is an identifier unique for this thread and C is a constant in the initial knowledge.

$C \subseteq X$	$?C.!XC$
$f(Y) \subseteq X$	$?Y(x).!Xf(x)$
$Y \subseteq X$	$?Y(x).!X(x)$
$Y \cap Z \subseteq X$	$?Y(x).?Z(x).!X(x)$
$\top \subseteq X$	$?\top(x).!X(x)$
$X \subseteq C$	$?Xf(x).\mathsf{err}$ for all $f \in \Sigma - \{C\}$
$X \subseteq f(Y)$	$?Xf(x).!Y(x)$ and $?Xg(x).\mathsf{err}$ for all $g \in \Sigma - \{f\}$

Now Φ is unsatisfiable if and only if the control point err is reachable. □

B Undecidability of name generation without pairs and with bounded height

Proposition B.1 *There is an encoding of 2-counter machines without pairs and with messages of bounded height.*

PROOF. In the proof A.2 of theorem 2.8, we have encoded a number n as a term of the shape $S^n 0C$ which obviously leads to terms of unbounded height. Therefore, in this proof we encode a number n as a list of length n (this idea is already in [DLMS99], what is new here is that pairing is not needed). We reserve keys:

$$Top_i, Cont_i, Adj_i, Below_i, Copy_i$$

for $i = 1, 2$ and the keys $0, 1, C$ and q for every state q of the 2-counter machine. We look at instructions on the first counter. To every instruction of type (1) (increase) we associate the equation:

$$\begin{array}{ll} A_q & = ?qu.?Top_1 ux_1.?Top_2 ux_2. \\ & \nu u', y_1, y_1', y_2\, !q'u'.!Top_1 u'y_1'.!Top_2 u'y_2. \\ & !Cont_1 y_1' 1C.!Adj_1 y_1' y_1.!Below_1 x_1 y_1.!Copy_1 x_1. \\ & !Below_2 x_2 y_2.!Copy_2 x_2.A_q \ . \end{array}$$

To every instruction of type (2) (test and decrease) we associate two equations:

$$\begin{array}{ll} A_q^1 & = ?qu.?Top_1 ux_1.?Top_2 ux_2. \\ & ?Cont_1 x_1 0z. \\ & \nu u', y_1, y_2\, !q'u'.!Top_1 u'y_1.!Top_2 u'y_2. \\ & !Below_1 x_1 y_1.!Copy_1 x_1. \\ & !Below_2 x_2 y_2.!Copy_2 x_2.A_q^1, \\ & \\ A_q^2 & = ?qu.?Top_1 ux_1.?Top_2 ux_2. \\ & ?Cont_1 x_1 1z.?Adj_1 x_1 x_1'. \\ & \nu u', y_1', y_2\, !q'u'.!Top_1 u'y_1'.!Top_2 u'y_2. \\ & !Below_1 x_1' y_1'.!Copy_1 x_1'. \\ & !Below_2 x_2 y_2.!Copy_2 x_2.A_q^2 \ . \end{array}$$

In this encoding, after every reduction the lists representing the two counters need to be copied in fresh locations. This operation is performed from the following threads ($k = 1, 2$).

$$\begin{array}{ll} C_k & = ?Copy_k x.?Below_k xy.?Cont_k x0z. \\ & !Cont_k y0C.C_k \ . \\ & \\ C_k' & = ?Copy_k x.?Below_k xy.?Adj_k xx'.?Cont_k x1z. \\ & \nu y'\, !Below_k x'y'.!Adj_k yy'.!Cont_k y1z.!Copy_k x'.C_k' \ . \end{array}$$

The computation is initialized by the thread:

$$\begin{array}{ll} \nu u, x_1, x_2\, !q_o u. & !Top_1 ux_1.!Cont_1 x_1 0C. \\ & !Top_2 ux_2.!Cont_2 x_2 0C.0 \ . \end{array}$$

Because pairing is not used, the conditions LINEARITY and INDEPENDENCE are obviously satisfied in this encoding; on the other hand the conditions LOCALITY and DATA OR KEY are not. □