Tree automata with one memory, set constraints and ping-pong protocols*

Hubert Comon[1,2] and Véronique Cortier[2] and John Mitchell[1]

[1] Department of Computer Science, Gates 4B, Stanford University, CA 94305-9045
{comon,jcm}@theory.stanford.edu, Fax: (650) 725 4671
[2] Laboratoire Spécification et Vérification, CNRS and Ecole Normale Supérieure de Cachan,{comon,cortier}@lsv.ens-cachan.fr

Abstract. We introduce a class of tree automata that perform tests on a memory that is updated using function symbol application and projection. The language emptiness problem for this class of tree automata is shown to be in DEXPTIME. We also introduce a class of set constraints with equality tests and prove its decidability by completion techniques and a reduction to tree automata with one memory. Set constraints with equality tests may be used to decide secrecy for a class of cryptographic protocols that properly contains a class of memoryless "ping-pong protocols" introduced by Dolev and Yao.

1 Introduction

Set constraints were introduced in the eighties and have been studied thoroughly since, with applications to the analysis of programs of various styles (see [2] for a survey). Typically, the problem of interest is to decide the satisfiability of a conjunction of *set expression inclusions* $e \subseteq e'$ in which the set expressions are built from variables and various constructions, including, e.g., *projection*. Athough some set variable may occur several time in an expression, most classes of set constraints do not make it possible to write a set expression for a set of terms of the form $f(t,t)$, in which one subterm occurs more than once. One exception is the class of constraints studied in [6].

Our motivating interest is to develop classes of cryptographic protocols for which some form of secrecy is decidable. A historical class of decidable protocols are the so-called *ping-pong protocols* [10]. Although none of the protocols of [8] belongs to this class, ping-pong protocols remain a decidable class, while most larger classes of security protocols are undecidable [5]. One of the main restrictions in [11, 10] is that messages are built using unary symbols only. In contrast, many protocols of interest are written using a binary encryption symbol and a pairing function. Another restriction in [11, 10] is that each protocol participant is stateless: after a message is sent, the participant does not retain any memory of the contents of the message. This is a significant limitation since many protocols

* Partially supported by DoD MURI "Semantic Consistency in Information Exchange," ONR Grant N00014-97-1-0505, and NSF CCR-9629754.

rely on challenge-response steps, that require memory. A previous investigation of ping-pong protocols with added state led to undecidability [13].

It is insightful to observe that Dolev and Yao's result [11] can be proved using set constraints. This suggests a generalization of their approach to trees. A technical complication, though, is that the generalization to trees is less expressive than one might expect: in the case of unary functions only, a function and its inverse are set inverses of each other, in the sense that $f(f^{-1}(X))$ is precisely X. However, this is no longer true with trees: if f_1^{-1} and f_2^{-1} are the two projections corresponding to a binary function symbol f, the set $f(f_1^{-1}(X), f_2^{-1}(X))$ contains pairs $f(t_1, t_2)$ which are not necessarily in X. In order to increase the expressiveness of set constraints with binary functions, we need a "diagonal construction", enabling us to test for equalities the members of sets.

In this paper, we introduce a new class of set constraints, allowing limited diagonal constructions. This class is incomparable with the class sketched in [6]. We show that satisfiability is decidable for this class, allowing us to generalize Dolev and Yao's result to trees. Our class of set constraints does not capture all protocol concepts of interest. In particular, as can be seen from the survey [8], many authentication protocols make use of *nonces* or *time stamps*, which we cannot express. On the other hand, properties of protocols that are modeled using set constraints are decidable, while nonces and timestamps typically lead to undecidability [5]. Moreover, we can express conservative approximations of general protocols, and it is possible in principle that set constraints with equality tests provide algorithms for determining the security of some such protocols.

We prove the decidability of set constraints with equality tests by a reduction to an emptiness problem for a class of *tree automata with constraints*. Tree automata with various forms of constraints have been studied by several authors (see [9] for a survey). However, the class we consider in this paper is incomparable with known decidable classes. Roughly, we allow each state to hold one arbitrarily large memory register and restrict the use of this memory to equality tests. Since memory registers are updated using projections and function application, this class is a generalization of pushdown word (alternating) automata. Despite the generality of the class, there is a simple proof that emptiness decision is in DEXPTIME.

After discussing the correspondence between protocols and set constraints in the next section, we recall known results on definite set constraints which will be used later (section 3). Then we introduce tree automata with one memory in section 4; this section can be seen as a stand-alone decidability result and relies on definite set constraints. Next, we introduce in section 5 our class of set constraints with one equality, showing how to reduce the satisfiability of these constraints to the non-emptiness decision for tree automata with one memory. The reduction is similar to the saturation process described in [7] for set constraints with intersection, but it is slightly more complicated due to equality tests. (In fact, we obtain a doubly exponential algorithm if the maximum arity of all function symbols is not constant.) Finally, we discuss the application to security protocols in section 6.

2 Protocol motivation

Dolev and Yao [11] consider protocols in which each principal holds a single public key (which is known to everybody) and a corresponding private key that is known to them only. The principals are able to build messages using plain text, encryption e_X with the public key of X and signatures d_X appending the name of principal X. Here is a simple example from [11]:

$A \to B : e_B(d_A(e_B(s)))$	Alice sends to Bob a message encrypted using Bob's public key consisting of a signed encrypted text s
$B \to A : e_A(s)$	Bob acknowledges the reception by sending back to Alice the text s, encrypted using the public key of Alice

In this model, communication channels are insecure. This allows an intruder to intercept messages, remember them, and replace them with alternate (possibly forged) messages. The intruder may decrypt a message if the corresponding key has become known to him, may append or remove signatures, and may encrypt using any public key. The secrecy question asks whether there is a way for an intruder to get the plain text message s that is supposed to be kept secret between Alice and Bob. In the above example, the answer is yes (the protocol is insecure).

The possible use of set constraints in cryptographic protocols analysis has been suggested in several papers, e.g. [14]. It is however interesting to see that the Dolev-Yao decidability proof can be summarized using set constraints by letting I be the set of messages that can be built by the intruder (after any number of sessions). Since I can intercept any message of any run of the protocol, we write set constraints putting every protocol message in I. For the example protocol above, we have

$$e_Y(d_X(e_Y(s))) \subseteq I \qquad\qquad e_X(e_Y^{-1}(d_X^{-1}(e_Y^{-1}(I)))) \subseteq I$$

for every pair of principals X, Y, since Bob acknowledges a message m from Alice by sending $e_A(e_B^{-1}(d_A^{-1}(e_B^{-1}(m))))$. Finally, for every principal X, we express the ability of the intruder to perform operations using public information about X:

$$d_X(I) \subseteq I, \;\; e_X(I) \subseteq I, \;\; d_X^{-1}(I) \subseteq I$$

This process translates a protocol into a collection of set constraints about the set I of messages available to the intruder. Secrecy now becomes the question whether the set constraints, together with $s \notin I$, is satisfiable ? Assuming a fixed number of principals, this is decidable in polynomial time for set constraints arising from Dolev-Yao's ping-pong protocols: we can compute the minimal solution of the definite set constraint and check the membership of s.

There are several restrictions in the Dolev-Yao approach. In particular, only a fixed number of principals and, as mentioned above, only unary symbols may be used. A pairing function or a binary encryption symbol, allowing to write e.g. $e(k, m)$ instead of $e_k(m)$, i.e. allowing to consider keys as first-class objects, would considerably increase the expressive power.

3 Definite set constraints

This class of set constraints has been introduced in [15] and studied by various authors (e.g. [7]). Each constraint is a conjunction of inclusions $e_1 \subseteq e_2$ where e_1 is a *set expression* and e_2 is a *term set expression*. Term set expressions are built out of a fixed ranked alphabet of function symbols $\mathcal{F}$, the symbol $\top$ and set variables. A set expression is either a term set expression or a union of two set expressions $e_1 \cup e_2$, or an intersection of two set expressions $e_1 \cap e_2$ or the image of set expressions by some function symbol $f(e_1, \ldots, e_n)$ or a projection $f_i^{-1}(e_1)$ where f is a function symbol and $i \in [1..n]$ if n is the rank of f. Note that negation is not allowed. Here is a definite set constraint:

$$f_2^{-1}(X) \subseteq g(Y) \qquad f(f(X,Y) \cap X, X) \subseteq X \qquad g(Y) \cap Y \subseteq X \qquad a \subseteq Y$$

Set expressions denote sets of subsets of the Herbrand universe $T(\mathcal{F})$; if σ assigns each variable to some subset of $T(\mathcal{F})$, then $[\![\ ]\!]_\sigma$ is defined by:

$$\begin{array}{ll} [\![X]\!]_\sigma \stackrel{\text{def}}{=} X\sigma & [\![f(e_1,\ldots,e_n)]\!]_\sigma \stackrel{\text{def}}{=} \{f(t_1,\ldots,t_n) \mid \forall i \in [1..n], t_i \in [\![e_i]\!]_\sigma\} \\ [\![e_1 \cap e_2]\!]_\sigma \stackrel{\text{def}}{=} [\![e_1]\!]_\sigma \cap [\![e_2]\!]_\sigma & [\![f_i^{-1}(e)]\!]_\sigma \stackrel{\text{def}}{=} \{t_i \mid \exists t_1, ..., t_n . f(t_1, ..., t_n) \in [\![e]\!]_\sigma\} \\ [\![\top]\!]_\sigma \stackrel{\text{def}}{=} T(\mathcal{F}) & [\![e_1 \cup e_2]\!]_\sigma \stackrel{\text{def}}{=} [\![e_1]\!]_\sigma \cup [\![e_2]\!]_\sigma \end{array}$$

σ satisfies $e_1 \subseteq e_2$ iff, $[\![e_1]\!]_\sigma \subseteq [\![e_2]\!]_\sigma$. This extends to conjunctions of inclusions.

Theorem 1 ([7]). *The satisfiability of definite set constraints is DEXPTIME-complete and each satisfiable constraint has a least solution.*

4 Tree automata with one memory

The idea is to enrich the expressiveness of tree automata by allowing them to carry and test some information. For instance, a pushdown automaton will keep a stack in its memory and check the symbols at the top of the stack. What we do here is something similar. Our automata work on trees instead of words and may perform more general constructions and more general tests.

Informally, a tree automaton with one memory computes bottom-up on a tree t by synthesizing both a state (in a finite set of states Q) and a memory, which is a tree over some alphabet Γ. Each transition uses some particular function which computes the new memory from the memories at each direct son. Each transition may also check for equalities the contents of the memories at each son.

Given an alphabet of function symbols Γ, the set of functions Φ which we consider here (and which may be used to compute on memories) is the least set of functions over $T(\Gamma)$ which is closed by composition and containing:

- for every $f \in \Gamma$ of arity n, the function $\lambda x_1, ...x_n . f(x_1, \ldots, x_n)$
- for every n and every $1 \leq i \leq n$, the function $\lambda x_1, ..., x_n . x_i$
- for every $f \in \Gamma$ of arity n and for every $1 \leq i \leq n$, the (partial) function which associates each term $f(t_1, \ldots, t_n)$ with t_i, which we write $\lambda f(\boldsymbol{x}).x_i$.

For instance, if Γ contains a constant (empty stack) and unary function symbols, Φ is the set of functions which push or pop after checking the top of the stack.

Definition 1. *A tree automaton with one memory is a tuple* $(\mathcal{F}, \Gamma, Q, Q_f, \Delta)$ *where* $\mathcal{F}$ *is an alphabet of input function symbols,* Γ *is an alphabet of memory function symbols,* Q *is a finite set of states,* Q_f *is a subset of final states,* Δ *is a finite set of transition relations of the form* $f(q_1, ..., q_n) \xrightarrow[F]{c} q$ *where*

- $f \in \mathcal{F}$ *is called the* head symbol of the rule,
- c *is a subset of* $\{1, ..., n\}^2$*, defining an equivalence relation on* $\{1, \ldots, n\}$.
- $\lambda x_1 ..., x_k F(x_1, \ldots, x_k) \in \Phi$*, where* k *is the number of classes modulo* c
- $q_1, \ldots, q_n, q \in Q$*, (*$q$ *is the* target *of the rule).*

A *configuration* of the automaton consists of a state and a term in $T(\Gamma)$ (the memory). Then computations work as follows: if $t = f(t_1, \ldots, t_n)$ and the computation on $t_1, \ldots, t_n$ respectively yields the configurations $q_1, \tau_1, \ldots, q_n, \tau_n$, then the automaton, reading t, may move to q, τ when there is a transition rule $f(q_1, \ldots, q_n) \xrightarrow[F]{c} q$ and for every $i = j \in c$, $\tau_i = \tau_j$ and $\tau = F(\tau_{i_1}, \ldots, \tau_{i_k})$ where $i_1, \ldots, i_k$ are representatives of the equivalence classes for c. A tree t is accepted by the automaton whenever there is a computation of the automaton on t yielding a configuration q, γ with $q \in Q_f$.

Example 1. Assume that the transitions of the automaton A are (other components of the automaton are obvious from the context, $\top$ is the identity relation):

$$g(q) \xrightarrow[\lambda x_1.x_1]{\top} q \qquad f(q_a, q_a) \xrightarrow[\lambda x_1.h(x_1)]{1=2} q \qquad a \xrightarrow[b]{\top} q_a$$
$$g(q_a) \xrightarrow[\lambda x_1.h(x_1)]{\top} q \qquad f(q, q) \xrightarrow[\lambda h(x_1).x_1]{1=2} q$$

A computation of the automaton on $f(g(f(a, a)), g(a))$ is displayed on figure 1, in which the configurations reached at each node are displayed in a frame.

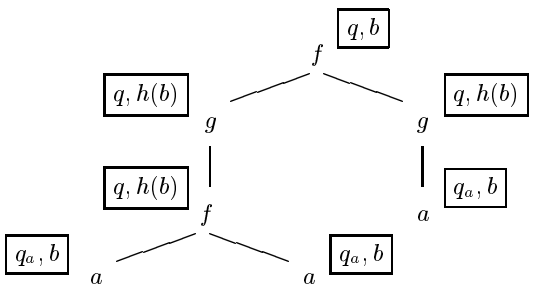

Fig. 1. A tree t and a computation of A on t

Pushdown automata (on words) perform transitions $a, q, \alpha \cdot \gamma \rightarrow q', \beta \cdot \gamma$ where a is an input symbol, q, q' are states and α, β, γ are words over the stack alphabet (the rule pops α and pushes β). Such a rule can be translated in the above formalism, viewing letters as unary symbols: $a(q) \xrightarrow[\lambda x.\beta\alpha_1^{-1}x]{} q'$. This translation does not make use of equality tests. Orthogonally, if we use the tests, but assume that $F = \lambda \boldsymbol{x}.f(\boldsymbol{x})$ for each rule headed with f, then we get tree automata with equality tests between brothers (see [4]).

Theorem 2. *The emptiness of the language recognized by a tree automaton with one memory is decidable in DEXPTIME. More generally, the reachability of a given configuration is decidable in DEXPTIME.*

Proof. (sketch) For every $q \in Q$, let M_q be the subset of $T(\Gamma)$ of memory contents m such that there is a tree t and a computation of the automaton on t yielding the configuration $< q, m >$. Then the sets M_q are the least solutions of the definite set constraint, consisting, for each transition rule $f(q_1, \ldots, q_n) \xrightarrow[F]{c} q$ of the inclusion $F(L_{q_{i_1}}, \ldots L_{q_{i_k}}) \subseteq M_q$ and $L_{q_{i_j}}$ is the intersection for all indices l equivalent (w.r.t. c) to i_j of M_l. Then the non-emptiness of the language (resp. reachability of a configuration) reduces to similar questions on definite set constraints, which are solvable in DEXPTIME.

The result can be generalized to alternating tree automata with one memory keeping the same complexity. Alternation here has to be understood as follows: we may replace the states occurring in the left hand sides of the rules with arbitrary positive Boolean combinations of states. The above proof simply works, using additional intersections and unions.

Corollary 1. *The emptiness problem of alternating tree automata with one memory is DEXPTIME-complete.*

Note however that the class of automata with one memory is neither closed under intersection nor complement (both yield undecidable models).

5 Set constraints with equality tests

We consider now definite set constraints as in section 3 with an additional construction: function symbols can be labeled with equality tests, which are conjunctions of equalities $p_1 = p_2$ between paths. The intention is to represent sets of terms t such that the subterms at positions p_1 and p_2 are identical.

More precisely, if c is a conjunction of equalities between paths (which we assume w.l.o.g. closed under transitivity and such that no strict prefix of a path in c is in c), we define

$$[\![f^c(e_1, \ldots, e_n)]\!]_\sigma \stackrel{\text{def}}{=} \{t \in [\![f(e_1, \ldots, e_n)]\!]_\sigma \mid t \models c\}$$

and $t \models c$ if, for every equality $p_1 = p_2$ in c, p_1, p_2 are positions in t and $t|_{p_1} = t|_{p_2}$. If $p_1 = p_2 \in c$, we say thatp_1, p_2 *are checked by* c. All other constructions

are the same as in section 3. In particular, right hand sides of inclusions should not contain constructions f^c. When c is empty, we may omit it or write $\top$.

Example 2. $f^{21=12}(f(Z,Y)\cap X, g(X)\cap Y) \subseteq f(Y,X)$ is an inclusion constraint. $\sigma = \{X \mapsto \{a, b, f(a,b)\}; Y \mapsto \{b, g(a), g(b), f(a,b)\}; Z \mapsto \{a,b\}\}$ is a solution of the constraint since $[\![f^{12=21}(f(Z,Y)\cap X, g(X)\cap Y)]\!]_\sigma = \{f(f(a,b), g(b))\}$

As a consequence of undecidability results on tree automata with equality tests (see e.g. [9]), the satisfiability of such general constraints is undecidable. That is why we are going to put more restrictions on the constraints.

If X is a variable of a constraint S, then let $\mathcal{R}(X)$ be the set of atomic constraints whose right hand side contains X. The set of variables having a *basic type* is the least set of variables X such that $\mathcal{R}(X)$ consists of inclusions $g_i(X_1^i, \ldots, X_{n_i}^i) \subseteq X$ and such that

- if the symbols g_i do not occur anywhere else in S and every X_j^i is either X itself or has a basic type, then X has a basic type
- if every X_j^i has a basic type, then X has a basic type.

Intuitively, the basic types correspond to data whose format is irrelevant (first case in the definition) or which can be built using a bounded number of symbols on such data (second case). For example, `Nat` and `U` have basic types in

$$\mathcal{R}(\texttt{Nat}) \cup \mathcal{R}(\texttt{U}) \stackrel{\text{def}}{=} \quad \texttt{zero} \subseteq \texttt{Nat}, \qquad s(\texttt{Nat}) \subseteq \texttt{Nat}, \qquad pn(\texttt{Nat}) \subseteq \texttt{U}$$

if `zero` and s are not used elsewhere in S.

This notion is extended to expressions: an expression e is *basic* if

- e is a basic variable or
- e is an intersection $e_1 \cap e_2$ and either e_1 or e_2 is basic
- e is an expression $e_1 \cup e_2$ and both e_1 and e_2 are basic
- e is an expression $f(e_1, \ldots, e_n)$ (or $f^c(e_1, \ldots, e_n)$) and $e_1, \ldots, e_n$ are basic

The set of *paths* in an expression e is defined as follows: $\Pi(f^c(e_1, \ldots, e_n)) \stackrel{\text{def}}{=} 1 \cdot \Pi(e_1) \cup \ldots \cup n \cdot \Pi(e_n)$, $\Pi(e_1 \cap e_2) \stackrel{\text{def}}{=} \Pi(e_1 \cup e_2) \stackrel{\text{def}}{=} \Pi(e_1) \cup \Pi(e_2)$, $\Pi(f_i^{-1}(e)) \stackrel{\text{def}}{=} \emptyset$. $e|_p$ is any of the subexpressions at position p. When there is no $\cup$ or $\cap$ symbol along a path p then $e|_p$ denotes a single expression.

The assumption: We assume that, in each subexpression $f^c(e_1, \ldots, e_n)$, for every $p_1 = p_2 \in c$, p_1 and p_2 are paths of $f(e_1, \ldots, e_n)$. (This is actually equivalent to restricting the use of projections below an f^c construction). Then, for each expression $f^c(e_1, ..., e_n)$ we require that, if $p \cdot i \cdot q$ is checked by c and p is not empty then, for $i \neq j$, either $p \cdot j = p \cdot i \cdot q \in c$ or any subexpression at position $p \cdot j$ has a basic type or any subexpression at $p \cdot i \cdot q$ has a basic type. This will be referred to as the *basicness condition.*

Example 3. If c is $12 = 21 = 11 \wedge 22 = 3$. the basicness condition imposes that either $e|_{21}$ or $e|_{22}$ has a basic type (and hence the other expressions at equivalent positions).

The resulting constraints are called *set constraints with equality tests* (*ET-constraints* for short). We can construct an ET-constraint whose least solution is the set of trees $\Delta = \{f(t,t) \mid t \in T(\mathcal{F})\}$. The only other decidable set constraint formalism which allows to express Δ is the class defined in [6], in which, however, equality tests are restricted to brother positions (which is not the case here). On the other hand, we have restrictions which are not present in [6].

5.1 Saturation

We use here a fixed point computation method which is similar to the one in [7]: the goal is to deduce enough consequences so that the inclusions whose right hand side is not a variable become redundant, hence can be discarded. Unfortunately, the first step (representation) in [7] cannot be used in the same way here, since it does not preserve the class of constraints we consider. Still, as a first step, we can get rid of projections and unions: we can compute in plynomial time an equivalent ET-constraints containing no union or projection.

Next, we normalize the expressions according to the following rule **Norm**:

$$f^c(e_1,\ldots,e_n) \cap f^{c'}(e'_1,\ldots,e'_n) \to f^{c\wedge c'}(e_1 \cap e'_1,\ldots,e_n \cap e'_n)$$

Lemma 1. *(***Norm***) transforms an expression in an equivalent one. Moreover, if the basicness condition is satisfied by the premises, then there is a (effectively computable) constraint c'' which is logically equivalent to $c \wedge c'$ and such that $f^{c''}(e_1 \cap e'_1,\ldots,e_n \cap e'_n)$ satisfies the basicness condition.*

We can get rid of basic type variables:

Lemma 2. *For each basic variable X, there is a ground expression t_X such that, if S' is the ET-constraint obtained by replacing in S each basic variable X with t_X, then σ is a solution of S' iff $\sigma \circ \sigma_X$ is a solution of S.*

We may assume now that there is no basic variable in the constraint. Then the tests can be simplified, removing positions of basic expressions.

Next, as in [7], we extend the language allowing non-emptiness preconditions in the rules, which allow to simplify (some but not all) inclusion constraints $f^c(\boldsymbol{e}) \subseteq f(\boldsymbol{e'})$. Formally, the set constraints are now clauses of the form

$$\mathsf{nonempty}(e_1),\ldots,\mathsf{nonempty}(e_n) \Rightarrow e \subseteq e'$$

where $e, e_1, \ldots, e_n$ are set expressions,e' is a set expression using only intersections, variables and function symbols (without constraints). $\mathsf{nonempty}(e_i)$ is satisfied by an assignment σ iff $[\![e_i]\!]_\sigma$ is not empty.

Then we remove constraints of the form $\phi \Rightarrow C[f^c(\boldsymbol{e}) \cap g^{c'}(\boldsymbol{e'})] \subseteq e''$ and replace constraints $\phi \Rightarrow e \subseteq C[f(\boldsymbol{e}) \cap g(\boldsymbol{e'})]$ with $\phi, \mathsf{nonempty}(e) \Rightarrow \mathbf{false}$ for every context C and every $f \neq g$. These rules are correct, by definition of the interpretation.

We also abstract out subexpressions introducing new variables, as long as this preserves the form of the constraints. For instance, for contexts $C[\ ]_p$, an

inclusion $C[f^c(\boldsymbol{e})]_p \subseteq e'$ becomes $C[X]_p \subseteq e', f^c(\boldsymbol{e}) \subseteq X$ where X is a new variable. This results in an equivalent constraint (on the original variables) in which the inclusions are $e \subseteq e'$ where e' is either an intersection of variables $X_1 \cap \ldots \cap X_n$ or an expression $f(X_1, \ldots, X_n)$ and e is either an intersection of variables or an expression $f^c(\boldsymbol{e})$ in which, at any position which is not a strict prefix of a position checked by c, there is a (non-basic) variable or a term t_X.

In addition, each time $p_1 = p_2$ appears in c in an expression $f^c(\boldsymbol{e})$, we assume that all subexpressions at positions p_1, p_2 are identical, which can be easily ensured replacing both subexpressions with their intersection.

Transitivity 1
$$\frac{\phi_1 \Rightarrow e_1 \subseteq e_2 \quad \phi_2 \Rightarrow e_2 \subseteq e_3}{\phi_1, \phi_2 \Rightarrow e_1 \subseteq e_3}$$

Transitivity 2
$$\frac{\phi \Rightarrow f^c(\boldsymbol{e})[X \cap g(\boldsymbol{e_1}) \cap e_2]_p \subseteq e_3 \quad \phi' \Rightarrow g^{c'}(\boldsymbol{e_4}) \subseteq X}{\phi, \phi' \Rightarrow f^c(\boldsymbol{e})[g^{c'}(\boldsymbol{e_4}) \cap g(\boldsymbol{e_1}) \cap e_2]_p \subseteq e_3}$$
provided p is a strict prefix of a path checked in c

Compatibility
$$\frac{\phi \Rightarrow e_1 \subseteq e_2 \quad \phi' \Rightarrow e_1' \subseteq e_2'}{\phi, \phi' \Rightarrow e_1 \cap e_1' \subseteq e_1' \cap e_2'}$$

Clash
$$\frac{\phi \Rightarrow f(\boldsymbol{e}) \subseteq g(\boldsymbol{e'})}{\phi \Rightarrow \textbf{false}} \quad \text{if } f \neq g$$

Weakenings
$$\frac{\phi \Rightarrow e_1 \subseteq e_2 \cap e_3}{\phi \Rightarrow e_1 \subseteq e_2} \qquad \frac{\phi \Rightarrow e_1 \subseteq e_2}{\phi \Rightarrow e_1 \cap e_3 \subseteq e_2}$$
If e_3 is an expression occurring somewhere in the set of constraints

Projection
$$\frac{\phi \Rightarrow f^c(e_1, \ldots, e_n) \subseteq f(e_1', \ldots, e_n')}{\phi, \mathsf{nonempty}(f^c(e_1, \ldots, e_n))) \Rightarrow e_i^{c\downarrow_i} \subseteq e_i'}$$
If the subexpression at every strict prefix of a position checked in c is of the form $g(\boldsymbol{e''})$ for some g.

Fig. 2. The saturation rules

Now, we are ready to apply the deduction rules given in figure 2, applying again abstractions and normalisation (eagerly) if necessary to keep the special form of the constraint. We use $e[e']_p$ to express either that e' is replaced by e at position p or that e' has to occur at position p. This means in particular that the subexpression at position p in e has to be defined in a unique way. $c\downarrow_i$ is defined by $(c \wedge c')\downarrow_i \stackrel{\text{def}}{=} c\downarrow_i \wedge c'\downarrow_i$, $(i \cdot p = i \cdot q)\downarrow_i \stackrel{\text{def}}{=} p = q$ and $(j \cdot p = q)\downarrow_i \stackrel{\text{def}}{=} \top$ when $i \neq j$. e^c is the expression in which the top symbol of e is constrained by c. (It is used only in a context where e must be headed with a function symbol or $c = \top$).

Lemma 3. *The inference rules in figure 2 are correct: the new constraint is a consequence of the previous ones.*

Lemma 4. *The rules of figure 2 are terminating: a fixed point is reached after finitely many steps (at most $O(2^{|S|\times|c|\times b\times 2^a})$ where a is the maximal arity of a function symbol, b is the number of basic types and $|c|$ is the maximal depth of an equality test).*

If S is an ET-constraint, let solved(S) be the clauses $\phi \to a$ in S such that either a is **false** or else a is an inclusion $f^c(\boldsymbol{e}) \subseteq e'$ where e' is an intersection of variables and $f^c(\boldsymbol{e})$ does not contain any subexpression of the form $X \cap g^c(\boldsymbol{e'})$ where X is a variable. Using a classical construction, we can show that:

Lemma 5. *solved(S) is either unsatisfiable or has a least solution.*

As in [7], the following completeness result is obtained by inspecting each clause $C \in S$ which is not in solved(S), showing that, thanks to saturatedness, the least solution of solved(S) is a solution of C. There are only some additional cases for non-flat constraints e.g. $f^c(X \cap g(\boldsymbol{e}), \boldsymbol{e'}) \subseteq f(\boldsymbol{e''})$.

Theorem 3. *If S is saturated, then either both S and solved(S) are unsatisfiable or else S has a least solution, which is the least solution of solved(S).*

5.2 The main result

We build, for each ET-constraint S, an automaton with one memory A_S such that if α is the least solution of S, for very variable X, $\alpha(X)$ is a simple homomorphic image of the set of terms accepted by A_S in state q_X. [1] The memory alphabet of the automaton is the set of function symbols used in the constraint and the alphabet $\mathcal{F}$ is the memory alphabet with some additional symbols allowing to check on auxilliary branches non emptiness conditions.

The set of states is the set of subexpressions occurring in the constraint, together with some local memory. We keep in the (unbounded) memory attached to each state the tree which will be checked later for equality. The idea is to accept in state e a term t iff there is at least one term in $[\![e]\!]_\sigma$ for every solution σ of the constraint. We have no room here to detail the construction.

As a consequence of lemma 3, lemma 4, theorem 2, theorem 3 and the above construction, we get:

Theorem 4. *The satisfiability of ET-constraints is decidable. Furthermore, given a set constraint with equality tests S, a term t and a free variable X of S, the consistency of $S \wedge t \notin X$ is equivalent to the non-reachability of the configuration q_X, t in A_S, which is decidable.*

[1] Note that we could prove the result directly, using the same trick as in the proof of theorem 2, without any reference to automata with one memory. However, we believe that theorem 2 is interesting in itself.

6 Analysis of cryptographic protocols

We sketch here a protocol example (inspired by Kerberos) that can be analyzed using ET-constraints, but is beyond the scope of [11]. We use a tupling function $< , >$, a binary encryption $\mathtt{enc}$ and several additional symbols.

1. $A \rightarrow S : A, B$
2. $S \rightarrow A : \mathtt{enc}(k(A), < B, K(A,B), \mathtt{enc}(k(B), < A, K(A,B) >) >)$
3. $A \rightarrow B : \mathtt{enc}(K(A,B), m(A,B)), \mathtt{enc}(k(B), < A, K(A,B) >)$
4. $B \rightarrow A : \mathtt{enc}(K(A,B), h(m(A,B)))$

In words, A tells the key server S that she wants to securely communicate with B. Then S sends back to A a message, encrypted using A's public key and containing a session key $K(A,B)$ together with a certificate which can be opened by B only. At the third step, A sends her message $m(A,B)$, encrypted using the key $K(A,B)$, together with the certificate, which is copied blindly from message 2. Finally, B acknowledges the reception, sending back a digest $h(m(A,B))$ of the previous message, encrypted using the shared key.

Because of lack of space, we only sketch here very briefly how to express the intruder capabilities on one hand and how the equality tests are used.

As in section 2, the intruder's knowledge is represented using a set variable I. We use a set variable S_i resp. R_i) for the set of messages sent (resp. received) at step i (see e.g. [12] for more details). Since the intruder I potentially intercepts all messages and may send any message he can forge, we have $S_i \subseteq I \subseteq R_i$. In addition, he has some capabilities to alter messages. For instance:

$$\mathtt{enc}(I, I) \subseteq I \qquad\qquad \mathtt{enc}(I, \top) \cap I \subseteq \mathtt{enc}(\top, I)$$

In words, I can encrypt a known message using a known key and I can decrypt a message whose encrypting key is known to him.

Now comes the protocol-dependent part of the constraints. The memory which is kept by the participants is modeled by sending at step i not only the ith message, but also relevant previous messages. This trick does not change anything to I's knowledge as previous messages were known to him. For instance, step 2 will consist of:

$$< R_1 \cap < \mathtt{U}, \mathtt{U} >, \mathtt{enc}(k(\mathtt{U}), < \mathtt{U}, K(\mathtt{U},\mathtt{U}), \mathtt{enc}(k(\mathtt{U}), < \mathtt{U}, K(\mathtt{U},\mathtt{U}) >) >) >^{c} \subseteq S_2$$

where $c \stackrel{\text{def}}{=} 11 = 211 = ... \wedge 12 = 221 = ...$ expresses that principal names do match. This is necessary since the intruder may be a participant in another session of the protocol: we would get a too rough overapproximation without these tests. At step 3, A includes blindly a piece of message 2, which can be expressed using an equality test on non-basic variables of the form: $< R_2 \cap \mathtt{enc}(..., ...X...), X >^{c} \subseteq S_3$ where c restricts the interpretations of the left hand side to sets of terms $< \mathtt{enc}(..., ...t...), t >$. The last message is successful only if B answers correctly the challenge, which can be expressed using an equality constraint representing A's memory: $< R_4 \cap \mathtt{enc}(X, h(Y)), S_3 \cap \mathtt{enc}(X, Y) >^{c} \subseteq S_5$ where c restricts the instances of the expression to sets of terms $< \mathtt{enc}(t_1, h(t_2)), \mathtt{enc}(t_3, t_2) >$.

We can handle an unbounded number of principals and messages may be built using any set of function symbols with any arity. However, we cannot handle nonces (randomly generated numbers) in general. In this respect, our decidability result is not more general than e.g. [3].

Considering nonces introduces several complications (which we can expect [12]): first we have to ensure that all nonces are distinct. This is possible at the price of introducing disequality tests on basic types in the set constraints, hence disequality tests in the automata. This may yield a still decidable model. A much harder issue is the freshness of the nonces. Indeed, each nonce has a lifetime, hence a scope (this becomes quite clear in spi-calculus formalizations [1]). Modeling the scope hardly fits into the (finite) set constraints formalism.

References

1. M. Abadi and A. Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 148(1), 1999.
2. A. Aiken. Introduction to set constraint-based program analysis. *Science of Computer Programming*, 35:79–111, 1999.
3. R. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In *Proc. CONCUR'00*, volume 1877 of *Lecture Notes in Computer Science*, 2000.
4. B. Bogaert and S. Tison. Equality and disequality constraints on brother terms in tree automata. In A. Finkel, editor, *Proc. 9th. Symposium on Theoretical Aspects of Comp. Science*, Cachan, France, 1992.
5. I. Cervesato, N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In P. Syverson, editor, *12-th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 1999.
6. W. Charatonik and L. Pacholski. Negative set constraints with equality. In *Proc. IEEE Symp. on Logic in Computer Science*, pages 128–136, Paris, 1994.
7. W. Charatonik and A. Podelski. Set constraints with intersection. In *Proc. IEEE Symposium on Logic in Computer Science*, Varsaw, 1997.
8. J. Clarke and J. Jacobs. A survey of authentication protocol. literature: Version 1.0. Draft paper, 1997.
9. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 1997.
10. D. Dolev, S. Even, and R. Karp. On the security of ping pong protocols. *Information and Control*, 55:57–68, 1982.
11. D. Dolev and A. Yao. On the security of public key protocols. In *Proc. IEEE Symp. on Foundations of Computer Science*, pages 350–357, 1981.
12. N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proc. Workshop on formal methods in security protocols*, Trento, Italy, 1999.
13. S. Even and O. Goldreich. On the security of multi-party ping-pong protocols. Technical Report 285, Technion, Haifa, Israel, 1983. Extended abstract appeared in IEEE Symp. Foundations of Computer Science, 1983.
14. N. Heintze and J. Tygar. A model for secure protocols and their compositions. *IEEE transactions on software engineering*, 22(1), 1996.
15. N. Heinze and J. Jaffar. A decision procedure for a class of set constraints. In *Proc. IEEE Symp. on Logic in Computer Science*, Philadelphia, 1990.