Tree automata with one memory, set constraints and cryptographic protocols *

Hubert Comon†
Department of Computer Science,
Gates 4B, Stanford University,
CA 94305-9045
and
Laboratoire Spécification et Vérification,
CNRS and Ecole Normale Supérieure de Cachan
61 Avenue du Président Wilson
94235 Cachan cedex, France
comon@lsv.ens-cachan.fr

Véronique Cortier
Laboratoire Spécification et Vérification,
CNRS and Ecole Normale Supérieure de Cachan
61 Avenue du Président Wilson
94235 Cachan cedex, France
cortier@lsv.ens-cachan.fr

December 28, 2001

*This is an extended version of a paper whose abstract appeared in Proc. ICALP 2001.

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

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.

Finally, we show how to apply these results to cryptographic protocols. We introduce a class of cryptographic protocols and show the decidability of secrecy for an arbitrary number of agents and an arbitrary number of (concurrent or successive) sessions, provided that only a bounded number of new data is generated. The hypothesis on the protocol (a restricted copying ability) is shown to be necessary: without this hypothesis, we prove that secrecy is undecidable, even for protocols without nonces.

Contents

1 Introduction **5**

2 Protocol motivation **7**
2.1 Dolev-Yao's result 7
2.2 A more expressive model 8
2.2.1 Messages 9
2.2.2 Events and Global States 11
2.2.3 Inductive Relations. 12
2.2.4 Protocols 13
2.2.5 Global State Transitions. 15
2.2.6 Secrecy Policy 15
2.2.7 An undecidability result 16

3 Definite set constraints **19**
3.1 Definite set constraints and intersection constraints 19
3.2 Intersection constraints with non-emptiness guards 21

4 Tree automata with one memory **24**

5 Set constraints with equality tests **28**
5.1 Definition of the class 28
5.1.1 General set constraints with equality tests 28
5.1.2 A complete deduction system 29
5.1.3 An undecidability result 30
5.1.4 Basic variables and expressions 31
5.1.5 Our assumptions 33
5.2 Saturation 35
5.2.1 Normalization 36
5.2.2 Abstractions 38
5.2.3 Getting rid of basic variables 39
5.2.4 Complexity issues in eliminating the basic variables . . . 44
5.2.5 Simplifying again the expressions 45
5.2.6 Deduction rules 49
5.3 Connection with automata with one memory 56

6 Analysis of cryptographic protocols **58**
6.1 A decidable class of protocols 59
6.2 Proof of Theorem 57 62

7 Conclusion **68**

A Petri nets **72**

B Correspondence between Reset Petri nets and protocols **74**

C Proof of lemma 29 **77**

D SET-constraints and Automata with one memory **93**

E Protocols and Horn Clauses **95**

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* [13]. 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 [14, 13] 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 [14, 13] 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 rely on challenge-response steps, that require memory. A previous investigation of ping-pong protocols with added state led to undecidability [18].

It is insightful to observe that Dolev and Yao's result [14] 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. More precisely, we define a class of cryptographic protocols whose decidability does not assume any bound on the number of sessions (whether concurrent or not), improving over former decision results, e.g. [3, 26, 24] (see [12] for a survey on decidability results for cryptographic

protocols). We also allow compound keys. Protocols in the class assume a limited copying capability for the agents. More precisely, we assume that an agent can only blindly copy one piece of the received message in the message (s)he sends. By "blindly" we mean here, without any type knowledge; this notion will be made precise in the paper. Let us emphasize that this restriction is satisfied by almost protocols that we found in the literature. We also prove that this restriction is necessary: secrecy becomes undecidable if we allow two blind copies.

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 (more precisely, we have to assume that there is a bounded number of nonces produced by each principal in any combination of sessions). 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.

We start in section 2.1 by introducing Dolev and Yao result and its formulation in terms of set constraints. In section 2.2, we we recall (one possible) formal semantics of cryptographic protocols. We also prove that, even in the absence of nonces, secrecy is undecidable.

In section 3.1, we recall classical results on definite set constraints and generalize them to set constraints with non-emptyness guards in section 3.2. The results of this last sections are used in the following sections.

In section 4, we introduce tree automata with one memory and we prove some decidability results, relying on definite set constraints with non-emptyness guards. This can be seen as a stand-alone decidability result.

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 more complicated due to equality tests.

In section 6.1 we define our class of cryptographic protocols and show how to apply the results of the previous sections to prove that secrecy is decidable for this class.

Several technical proofs, which are not interesting by themselves, are pushed to appendices.

2 Protocol motivation

2.1 Dolev-Yao's result

Dolev and Yao [14] 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 [14]:

Example 1 *([14]):*

$A \to B:$	$e_B(d_A(e_B(s)))$	*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). For example, Dolev and Yao give the following attack: After a first session of the protocol, the intruder, I, who overhears the messages exchanged during that session, sends to A the message $e_A(d_I(e_A(s)))$, which he can build using the reply from Bob, and receives $e_I(s)$ in return.

The possible use of set constraints in cryptographic protocols analysis has been suggested in several papers, e.g. [19]. 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 an automaton accepting 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. Such a model is presented below.

2.2 A more expressive model

We start from a model inspired by Paulson [25] and developed by Millen and Ruess in [23]. However, we do not use the trace model as in [23] or [25], but a new state-transition model similar to the MSR model proposed by Mitchell *et al* [5] or those presented in [10]. Such models are much too expressive to be decidable, thus we consider in this paper a restricted model which does not allow nonce creation but on the other hand we add an arbitrary number of function symbols. In particular, we add compound keys and hashing. If a limited number of nonces is allowed for each pair of principals, nonce creation can be simulated beforehand, using additional binary function symbols $N_1, \ldots, N_k$ whose arguments are agent names.

In this section, it will be shown that this restricted model is still undecidable but not so far from decidability : sections 4 and 5 develop a decidable class of set constraints which will be used as a tool to extract a decidable fragment (see section 6.1) of the model described below.

2.2.1 Messages

They are built from a set of function symbols $\mathcal{F}$. Symbols of $\mathcal{F}$ are split into several sets:

agent's names: we assume that $\mathcal{F}$ contains constants and function symbols which allow to built agent's names. We assume that the set of agent's names is infinite. Furthermore, we distinguish an infinite subset of *honest agents* $\mathcal{A}_h$.

invertible symbols which, intuitively, correspond to constructions whose components can be computed by an intruder. Typically, the *pairing* functions belong to this set of symbols since it is assumed that an intruder can retrieve each component u, v from a pair $< u, v >$. Such symbols can be applied to any term.

one way symbols which, intuitively, correspond to constructions whose components cannot be computed by an intruder. Typically, *hash functions* belong to this set. Such function symbols can be applied to any term. In addition, we assume that there are two special function symbols with one argument: $\mathsf{pub}()$ and $\mathsf{priv}()$. Intuitively, $\mathsf{pub}()$ and $\mathsf{priv}()$ return respectively a public and a private key when they are applied to agents names.

partially invertible symbols which intuitively correspond to constructions whose component can be computed by an intruder, subject to some knowledge of the intruder. More specifically, we will consider only one such function: encryption. (This is the only relevant example we can think of, but we could generalize to more symbols in this set). For such a binary function, which takes as argument a term k (a key) and a term u and whose application will be written $\{u\}_k$, the intruder can build $\{u\}_k$ when he knows u and k and can retrieve u when he knows $\{u\}_k$ and the inverse key k^{-1}. A priori, the encryption function can be applied to any pair of terms so that we are not restricted to so-called "atomic keys". However, we will assume that the inverse of a key is the key itself, except for expressions $\mathsf{pub}(a)$ and $\mathsf{priv}(a)$ which are inverse of each other.

Orthogonally, $\mathcal{F}$ is split into three sets of function symbols: those which are known publicly $\mathcal{PF}$ (for instance $\mathsf{pub}()$, $< _, _ >$, $\{_\}_$), those which are cannot used by the public $\mathcal{UF}$, but only by specific agents (for instance a key construction function, which is known to a specific server only) and finally those which can be used by an intruder $\mathcal{AF}$, only with specific arguments. This last notion is the dual of partially invertible symbols. $\mathsf{priv}()$ is an example of such a symbol, which can be used by an intruder i, with the argument i only. We will

see later more examples. To summarize, the set of function symbols consists of

$$\mathcal{F} = \mathcal{AG} \uplus \mathcal{IF} \uplus \mathcal{OF} \uplus \{\{_\}_\} = \mathcal{PF} \uplus \mathcal{UF} \uplus \mathcal{AF}$$

where $\mathsf{pub}() \in \mathcal{PF}$, $\mathsf{priv}() \in \mathcal{AF}$, $\{_\}_ \in \mathcal{PF}$. For each partially constructible symbol in $\mathcal{AF}$, it must be specified which of the arguments must be specific and which are unrestricted. The only argument of $\mathsf{priv}()$ is restricted.

Moreover, we assume a finite set of *sorts* containing in particular the sorts `Agent`, `Ah`, `Ad`, `Message` such that the set of messages of sort `Ah` is exactly $\mathcal{A}_h$ and the set of messages of sort `Ad` is exactly $\mathcal{AG} - \mathcal{A}_h$. Elements of $\mathcal{AG}$ are constants or function symbols returning agent's names of sort `Agent`, $\mathsf{pub}(), \mathsf{priv}(), \{_\}_ :$ `Message` $\times$ `Message` $\rightarrow$ `Message`. The type of other symbols has to be specified with the protocol. Finally `Ah` and `Ad` are subsorts of `Agent` and `Agent` is a subsort of `Message`, the sort of all messages.

The set of messages is the set of (ground) terms $T(\mathcal{F})$ built over the above described signature and whose sort is `Message`.

As an example of an additional sort, we could consider *nonces*. Note however that, in our (un)decidability results, we will always assume that there is a bounded number of nonces; it is then possible to represent them as messages of the form e.g. $n_i(a, b)$ where a, b are agents (in which case $n_i \in \mathcal{AF} \cap \mathcal{OF}$ and it is restricted in its first argument, meaning that only a can generate $n_i(a, b)$, for any b).

Describing protocols and the behavior of honest participants requires variables ranging either over messages or over agents. Variables ranging over agents are usually called *roles*. *Message schemes* are terms of sort `Message`, built over $\mathcal{F}$ and possibly variables.

Example 2 *We present here a protocol example (inspired by Kerberos), which will be used as a running example through the paper.*

1. $A \rightarrow S \quad : \quad A, B$
2. $S \rightarrow A \quad : \quad \{< B, K(A,B), \{< A, K(A,B) >\}_{\mathsf{shr}(B)} >\}_{\mathsf{shr}(A)}$
3. $A \rightarrow B \quad : \quad < \{m(A,B)\}_{K(A,B)}, \{< A, K(A,B) >\}_{\mathsf{shr}(B)} >$
4. $B \rightarrow A \quad : \quad \{h(m(A,B))\}_{K(A,B)}$

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 key that she shares with the server 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.

We are going to see in more details how this protocol is formally described in the model. For the moment, let us only precise the components of the signature.

We assume here six sorts: `Nat`, `Agent`, `Ah`, `Ad`, `Message`, `Key`. *The last sort is protocol specific. Introducing such a sort means that the agents are assumed to be able to see whether a message is a key or not (we will discuss this hypothesis later on).*

There is a specific constant s (the server) of sort `Agent`. *The way other agent's names are built is irrelevant. We could, for instance, use natural numbers together with a label for (dis)honest participants:* $0 :\rightarrow$ `Nat`, $succ$: `Nat` $\rightarrow$ `Nat`, ha : `Nat` $\rightarrow$ `Ah`, da : `Nat` $\rightarrow$ `Ad`. *For simplicity, in what follows, we will use the notation $a_1, a_2, ...$ for honest agents (i.e agent of sort* `Ah`*) and $i_1, i_2, ...$ for dishonest agents (i.e. the other agents).*

Then, we use $\mathcal{IF} = \{< _, _ >, < _, _, _ >\}$. *These tupling functions take arbitrary messages as arguments and return messages.*

$$\mathcal{OF} = \{h, m, K, \mathsf{shr}, succ, 0, ha, da\}$$

with h : `Message` $\rightarrow$ `Message`, m : `Agent` $\times$ `Agent` $\rightarrow$ `Message`, K : `Agent` $\times$ `Agent` $\rightarrow$ `Key`, shr : `Agent` $\rightarrow$ `Message`.

Now, the following are public symbols:

$$\mathcal{PF} = \{< _, _ >, < _, _, _ >, \mathsf{pub}, \{_\}_, h, 0, succ, da, ha\}$$

In particular, anybody can know every agent name and every agents public key. Now, K can only be used by the server

$$\mathcal{UF} = \{K\}$$

Finally,

$$\mathcal{AF} = \{m, \mathsf{prv}, \mathsf{shr}\}$$

where both symbols are restricted in their first argument.

2.2.2 Events and Global States

There are three kinds of events: *message*, *spell*, and *state* events. A message event is simply a term (of sort `Message`) representing the content of the message. A spell event is an expression $C = S \ddagger L$ where S is a finite set of messages and L is a finite set of agents. It generates the *book*, or session-specific set of basic secrets $Book(C) = S$, which are shared among the set $Cabal(C) = L$ of agents, the *cabal*. The set of *secrets* of a spell consists of its book plus the long-term secrets of its cabal:

$$Sec(C) = Book(C) \cup ltk(Cabal(C))$$

Intuitively, a spell $C = S \ddagger L$ is a secrecy policy where S is required to be known at most by the agents of L. A state event is of the form $Q = S(\mathsf{A}, n, X)$ where A is a ground term of sort `Agent`, n is a natural number that represents the step of the protocol, and $X = Mem(Q)$ is a tuple of messages representing the memory held by the state. A *state scheme* is built in the same way, except that the agent can be abstracted (using a role) and the messages are replaced with message schemes.

A *global state* is a set (not a multiset) of events. The *content* of a global state is its set of messages, written:

$$Cont(H) \stackrel{\text{def}}{=} H \cap Messages$$

Example 3 *(example 2 continued)*
The messages i_1 or $m(i_1, a_2)$ can be build from the formalism described in our running example. $S(i_1, 1, < i_1, a_1, s >)$ is a state event. Let $\mathcal{A}_h$ be the set of honest agents (i.e. $\{s\} \cup \{a_n \mid n \in \mathbb{N}\}$),

$$H_0 = \{\{m(a,b)\} \ddagger \{a,b\} \mid a,b \in \mathcal{A}_h\} \cup \{\{K(a,b)\} \ddagger \{a,b,s\} \mid a,b \in \mathcal{A}_h\} \cup \{\{\mathsf{shr}(a)\} \ddagger \{a,s\} \mid a \in \mathcal{A}_h$$

is a global state consisting in spell events only.

2.2.3 Inductive Relations.

Given a term $t = f(t_1, \ldots, t_n)$, $\mathsf{parts}(t)$ is defined inductively as follows:

- if $f \in \mathcal{OF} \cup \mathcal{AG}$, then $\mathsf{parts}(t) \stackrel{\text{def}}{=} \{t\}$,
- if $f \in \mathcal{IF}$, then $\mathsf{parts}(t) \stackrel{\text{def}}{=} \{t\} \cup \bigcup_{i=1}^{n} \mathsf{parts}(t_i)$,
- if $f = \{_\}_$, then $\mathsf{parts}(\{t_1\}_{t_2}) \stackrel{\text{def}}{=} \{t\} \cup \mathsf{parts}(t_1)$.

Given a set of terms S, $\mathsf{parts}(S)$ is the set of parts of all terms in S.

$\mathsf{analz}(S)$ is the subset of $\mathsf{parts}(S)$ consisting of only those subterms that are accessible to an attacker: $\mathsf{analz}(S)$ is the least set S' containing S and such that:

- if $f(t_1, \ldots, t_n) \in S'$ and $f \in \mathcal{IF}$, then $t_1, \ldots, t_n \in S'$,
- if $\{t_1\}_{t_2} \in S'$ and $t_2^{-1} \in S'$, then $t_1 \in S'$.

Conversely, an attacker may use any available function to build new messages. $\mathsf{synth}(S)$ is the least set of messages S' containing S and such that

- If $f \in \mathcal{PF}$ and $t_1, \ldots, t_n \in S'$, then $f(t_1, \ldots, t_n) \in S'$

- If $f \in \mathcal{AF}$, f is restricted w.r.t. its arguments $j_1, \ldots, j_k$, $t_1, \ldots, t_n \in S'$, and $t_{j_1}, \ldots, t_{j_k} \in \mathcal{A}_d$, then $f(t_1, \ldots, t_n) \in S'$, where $\mathcal{A}_d$ is the set of dishonest agents ($\mathcal{A}_d = \{i_n \mid n \in \mathbb{N}\}$).

The intruder in our model synthesizes faked messages from analyzable parts of a set of available terms and he can iterate the process. This motivates the following definition: $\mathsf{fake}(S)$ is the least set containing S and such that $\mathsf{synth}(S) \subseteq \mathsf{fake}(S)$ and $\mathsf{analz}(S) \subseteq \mathsf{fake}(S)$. Note that $\mathsf{fake}(S)$ is not necessary equal to $\mathsf{synth}(\mathsf{analz}(S))$ if we do not assume atomic keys: for instance if an intruder knows $t_1, \{t\}_{<t_1,t_2>}, t_2$, he can build t by first constructing $< t_1, t_2 >$ and then decrypt the message.

Example 4 *(example 3 continued)*
Assume that (in some state), the intruder holds the following messages:

$$S_1 = \{\{< m(a_1, i_1), a_2 >\}_{h(K(a_1,a_2))}, \{h(K(a_1, a_2))\}_{\mathsf{pub}(i_1)}, \{m(a_1, a_2)\}_{K(a_1,a_2)}\}$$

Then $\mathsf{analz}(S_1 \cup S_2)$ *contains for instance* $h(K(a_1, a_2)), m(a_1, i_1)$ *but not* $m(a_1, a_2)$. $\mathsf{fake}(S_1 \cup S_2)$ *contains for instance* $\{< a_2, h(K(a_1, a_2)) >\}_{h(m(a_1,i_1))}$.

2.2.4 Protocols

A protocol transition t is of the form $Pre(t) \longrightarrow Post(t)$, where $Pre(t)$ and $Post(t)$ are (finite) sets of messages and states. (Unlike in [10], there is not any new spell : the secrecy policy can only be specified in the initial global state as presented later.) Such transitions specify a possible global state change in a way to be explained below. A transition t shows a state change for one agent. Formally, $Pre(t)$ and $Post(t)$ contain at most one state event and $Pre(t)$ contains one state event if and only if $Post(t)$ contains one state event.

A *protocol* is simply a set of protocol transitions and an initial global state H_0. Both the protocol transitions and the initial global state are infinite. They are however represented by means of instances of a finite number of terms: typically, the protocol is given by a finite set of rules $u_i \rightarrow v_i$ where u_i and v_i are finite sets of message schemes and state schemes. Such rules represent the infinite set $u_i\sigma \rightarrow v_i\sigma$ where σ is any substitution compatible with the types.

The initial global state is given by a finite set of spell event scheme

$$t_1, \ldots, t_n \ddagger x_1, \ldots, x_k, u_1, \ldots, u_m$$

where $x_1, \ldots, x_k$ are roles and are the free variables of the message schemes $t_1, \ldots, t_n$ and $u_1, \ldots, u_m$ are ground terms of sort *Agent*.

H_0 corresponds to the secrecy policy of the protocol for the "honest agents" of $\mathcal{A}_h$. This represents the set of instances $t_1\sigma, \ldots, t_n\sigma \ddagger a_{i_1}, \ldots, a_{i_k}$ where σ assigns each x_j some honest agent a_{i_j}.

$$\emptyset \longrightarrow \left\{ \begin{array}{l} S(A,1,<A,B,s>), \\ S(B,1,<B,s>), \\ S(s,1,s) \end{array} \right\} \quad (0)$$

$$\{S(A,1,<A,B,s>)\} \longrightarrow \{S(A,2,<A,B,s>),<A,B>\} \quad (1)$$

$$\left\{ \begin{array}{l} S(s,1,s) \\ <A,B> \end{array} \right\} \longrightarrow \left\{ \begin{array}{l} S(s,2,s), \\ \{B,K(A,B),\{A,K(A,B)\}_{\mathsf{shr}(B)}\}_{\mathsf{shr}(A)} \end{array} \right\} \quad (2)$$

$$\left\{ \begin{array}{l} S(A,2,<A,B,s>), \\ \{B,X,Y\}_{\mathsf{shr}(A)} \end{array} \right\} \longrightarrow \left\{ \begin{array}{l} S(A,3,<A,B,s,X,m(A,B)>), \\ <\{m(A,B)\}_X,Y> \end{array} \right\} \quad (3)$$

$$\left\{ \begin{array}{l} S(B,1,<B,s>), \\ <\{Z\}_X,\{A,X\}_{\mathsf{shr}(B)}> \end{array} \right\} \longrightarrow \left\{ \begin{array}{l} \{S(B,1,<B,s,A,Z,X>), \\ \{h(Z)\}_X\} \end{array} \right\} \quad (4)$$

Figure 1: Rules of the protocol

Example 5 *(example 4 continued)*

The protocol, as described in example 2 is a bit sloppy. We used there the standard notations, but, if we want to be more precise, we have to specify for instance in message 3 how Alice retrieves the different components of the message she sends. Typically in such protocols, A, B are roles, not agent's names. The "B" in message 3 can be either the name sent in message 1 or the name passed in message 2 (It does not make a difference in this particular example. But it does make a difference in other situations, as shown by the attack on the Needham-Schroeder protocol [21]).

Following example 3, H_0 is specified by:

$$\{m(A,B)\} \ddagger \{A,B\}, \{K(A,B)\} \ddagger \{A,B,s\}, \{\mathsf{shr}(A)\} \ddagger \{A,s\}$$

where A, B are roles. Intuitively, this tells that $m(A,B)$ is a secret which must be shared by A and B only, that $K(A,B)$ is a secret that must be shared by A, B and s and that $\mathsf{shr}(A)$ is a secret which must be shared by A and s only.

Next, the protocol rules are given in figure 1. The rule 0 says that, at any time, a new session can be started (the precondition is an empty set). After applying this rule to an instance a, b, the agents a, b, s are ready to act as participants of a protocol session.

The rule 1 corresponds to the first step of the protocol: any agent a who is ready to act as A in the protocol can send the message $< a, b >$ to s and switch to a state in which she remembers having completed the first step (hence the second argument is 2) and having sent the message $< a, b >$ to s.

Rule 2 corresponds to the second step of the protocol: if s is ready to serve a key and if the message $< a, b >$ has been sent, then the server switches, generates the key $K(a,b)$ and sends the expected message. Note that the variables A, B are local *to the rule, hence the instances are not necessary the same as in the previous step: an intruder can very well perform the first step of the protocol, in which*

case there are two $< a, b >, < a', b' >$ *in the global state and the second instance may be used instead of the first one.*

In the rule 3, the agent a*, who completed the first steps of the protocol expects a message of the form* $\{b, \alpha, \beta\}_{\mathsf{shr}(a)}$*. She can check that the message is an encrypted message containing three components and that the first component is an agent's name, with whom she started a session. However, she cannot check that the second component is indeed* $K(a, b)$ *and, similarly, she cannot open the third component (the ticket). Hence these two components are left as local variables of the rules which can be instantiated in an arbitrary way, provided that* Y *gets a term of sort* `Message` *and* X *gets a term of sort* `Key`*. (We assume here that* a *is able to recognize whether a term has type* `Key` *or not.)*

Similarly, in the last rule, the expected instance of Z *is* $m(a, b)$*, but it could be any faked message: there is no way to check this.*

This formal specification of the protocol gives more precisions on the abilities of each agent. We precise here what is expected by each participant and what is his behavior.

2.2.5 Global State Transitions.

Given a protocol P and a set of initial knowledge I (of the intruder), the *global succession* relation transforms a state H to a new state H'. A succession is either *honest*, i.e. it corresponds to an action by an agent following the protocol, or it is *faked* by the intruder.

- H' is an *honest* successor of H, denoted by $honest(P)(H, H')$, if there exists an applicable transition t in P such that $H' = (H \backslash (Pre(t) \cap States)) \cup Post(t)$.
- H' is a *fake* successor of H, denoted by $fake(I)(H, H')$, if there exists a field $X \in \mathsf{fake}(Cont(H) \cup I)$ such that $H' = H \cup \{X\}$.

In the honest case, a transition t is *applicable* in H if $Pre(t) \subseteq H$. In the fake case, the intruder is restricted to adding only messages that can be inferred from the content of the current state and the initial knowledge. In either case, we write $global(P, I)(H, H')$. This relation determines a logical transition system with the initial global state H_0 as its initial state. The set of reachable states of this transition system is denoted by $reachable(P, I)$.

2.2.6 Secrecy Policy

A spell is *compatible* with an initial knowledge set I that does not mention its associated basic secrets.

$$compatible(I) \stackrel{\text{def}}{=} \{C \mid Sec(C) \cap \mathsf{parts}(I) = \emptyset\}$$

Given the intruder's initial knowledge I, a global state H is called *I-secret* if $\mathsf{fake}(Cont(H) \cup I) \cap Book(C) = \emptyset$ for all I-compatible spells $C \in H$; these states are collected in the set $secret(I)$. Now, a protocol P is called *secret* if $secret(I)$ is an invariant of the transition relation associated with P; i.e. for all I, $reachable(P, I)$ is a subset of $secret(I)$.
Remark : Since the spells are fixed by H_0, it is sufficient to prove that $secret(I_0)$ is an invariant for I_0 the maximal set compatible with H_0 :

$$I_0 = \{m \,|\, \forall C \in H_0 \quad \mathsf{parts}(m) \cap Sec(C) = \emptyset\}.$$

This definition is slightly different from the one given in [10] but it matches more precisely the idea of secrecy while the definition given in [10] was an over-approximation of secrecy in order to allow inductive proofs.

2.2.7 An undecidability result

We present now an undecidability result. Let us emphasize that we do not have here the nonce construction. Hence, the result is stronger than the undecidability result of [17].

Theorem 6 *It is undecidable whether or not a protocol P is secret.*

A similar undecidability result was proved by S. Even and O. Goldreich in [18] by reducing secrecy to the Post Correspondence Problem. But the rules of their protocol can be played in any order (the second participant doesn't need to receive a message before responding) and moreover for each instance of PCP with no solution, the corresponding protocol does not have necessarily one honest instance. Undecidability could also be proved by a simple encoding of the reachability problem of Turing machines like in [10], but we prefer to present here a reduction such that the corresponding protocol is a "real" protocol in the sense that each rule of the protocol can be played in the given order : the first rule, than the second and so on, i.e., there is at least an honest instance of the protocol.

Also, there will be no conditional rule nor any test. Moreover, we will use standard cryptographic primitives only, namely pairing and encryption with symmetric keys and a fixed number of roles (actually only one role) and a finite number of protocol rules. In other words, we presented a general setting in the previous sections, because our decidability results can be proved in such a general setting, but the undecidability result does not make use of, e.g., composed keys. The undecidability proof also works if we assume a fixed number of agents: the server, one honest agent and the intruder. Finally, it must be emphasized that the result still holds when the intruder's capabilities are much more restricted

than those described in the previous sections: the proof works even if the intruder is unable to build any new message.

The remainder of this paragraph is devoted to a proof sketch of this result (the full proof is given in appendix B).

We use the following undecidability result :

Theorem 7 *Given a Reset Petri net N and an initial configuration c_0, the reachability of the empty configuration (where all places are empty) is undecidable.*

This is, for example, a straightforward consequence of Theorem 11 in [15].

Given a reset Petri net N and an initial configuration c_0, we construct a protocol P in the following way : let $\mathcal{A}_h$ be a subset of agents and $H_0 = \{\{n(a), \mathsf{shr}(a)\} \ddagger a \,|\, \forall a \in \mathcal{A}_h\}$, the initial rule of the protocol is

$$\emptyset \longrightarrow \{S(A, 1, A, s), S(s, 1, s)\}$$

i.e., A and the server are ready to start.

We use a pairing function $< _, _ >$ to encode tuples. For simplicity we write $< \alpha_1^{k_1}, \ldots, \alpha_m^{k_m} >$ for the the binary tree $t \stackrel{\text{def}}{=} t_1$ whose internal nodes are labeled with $< _, _ >$ and defined by induction:

- $u_{i,1} \stackrel{\text{def}}{=} \alpha_i$ for all i
- $u_{i,j+1} \stackrel{\text{def}}{=} < \alpha_i, u_{i,j} >$
- $t_m = u_{m,k_m}$
- $t_{i-1} = < u_{i-1,k_{i-1}}, t_i >$ for all $i \geq 2$

We also use the following notation:

$$\alpha^m \star x \stackrel{\text{def}}{=} < \underbrace{\alpha, \ldots, < \alpha}_{m}, x > \ldots >$$

The initial configuration $c_0 = (c_0^1, \ldots, c_0^p)$ (p stands for the number of places of the Petri net) is encoded by the second rule of the protocol P :

$$\{S(A, 1, A, s)\} \longrightarrow \left\{ S(A, 2, A, s, n(A)), [\left\{ \{n(A)\}_{\mathsf{shr}(A)} \ddots_{\mathsf{shr}(A)} , \overline{c_0} \right\}_{\mathsf{shr}(A)}] \right\}$$

where $n(A)$ is encoded $p+1$ times by $\mathsf{shr}(A)$ and $\overline{c_0}$ stands for the message

$$A, A^{c_0^1+1}, A^{c_0^2+1}, \ldots, A^{c_0^p+1}, A.$$

We assume that each transition of N can be applied (at least) once in a given order from c_0. (We show in the Appendix A why we can do such an assumption.) We consider the transitions numbered in this order. After this first phase, any configuration of the Petri net is encoded by $A, A^{k_1}, \ldots, A^{k_p}, A$ in which $k_1, \ldots, k_p$ are respectively the number of tokens in the places $1, \ldots, p$. The odd transitions will correspond to rules of the protocol on the form: A sends something to the server and the even transitions will correspond to rules of the protocol on the form: the server sends something to A. Therefore A and the server will respond to each other.

A transition is either a reset transition or a transition on the form : n_1 tokens are taken from place i_1, ..., n_r tokens are taken from place i_r, m_1 tokens are added to place j_1, ... m_s tokens are added to place j_s.

A reset transition is simply encoded by (assume that place number j is reset and assume, for example, that this reset transition corresponds to the i^{th} even transition):

$$\begin{aligned}\{S(A, i+1, A, s, N), [\{X, A, X_1, X_2, \ldots, X_p, A\}_{\mathsf{shr}(A)}]\} \\ \longrightarrow \{S(A, i+2, A, s, N), [\{X, A, X_1, \ldots, , X_{i-1}, \\ A, X_{i+1}, \ldots, X_p, A\}_{\mathsf{shr}(A)}]\}\end{aligned}$$

i.e. X_i is replaced with A, where $N, X, X_1, \ldots, X_p$ are variables of sort `Message`.

For the sake of simplicity, we give the encoding of the second kind of transitions on an example only. Assume the transition is : 3 tokens are taken from place i_1, 2 tokens are taken from place i_2 and 4 tokens are added to place i_3. Assume that this transition corresponds to the i^{th} odd transition. The corresponding rule is:

$$\begin{aligned}\{S(s, i, s, A, N), [\{X, A, X_1, \cdots, A^3 \star X_{i_1}, \ldots, \\ X_{i_3}, \ldots, A^2 \star X_{i_2}, \ldots, X_p, A\}_{\mathsf{shr}(A)}]\} \\ \longrightarrow \{S(s, i+1, s, A, N), [\{X, A, X_1, \cdots, X_{i_1}, \ldots, \\ A^4 \star X_{i_3}, \ldots, X_{i_2}, \ldots, X_p, A\}_{\mathsf{shr}(A)}]\}\end{aligned}$$

Intuitively, the protocol rules simulate the Petri net transitions: if we can reach a configuration $\overline{c}$, and we want to apply the transition t_i from $\overline{c}$, then it suffices to open a new session until step i and, at that step, the intruder forwards $\overline{c}$ instead of the configuration reached after $i-1$ steps.

It remains us to describe the final rule of the protocol: (assume the Petri net N has $2n$ transitions)

$$\begin{aligned}\{S(A, n+2, A, s, N), [\{\{X\}_{\mathsf{shr}(A)}, < A, Y >\}_{\mathsf{shr}(A)}]\} \\ \longrightarrow \{S(A, n+3, A, s, N), [\{X, Y\}_{\mathsf{shr}(A)}, X]\} \quad (5)\end{aligned}$$

The final rule removes one level of encryption each time the first place of the net is empty. If it is possible to reach a configuration A^{p+2} in which all places are empty, then it suffices to open sufficiently many sessions so as to reach successively p times the state $S(a, n+2, a, s, n(A))$. Each time this state is reached, the last rule is activated by the intruder with an instance A^i, and i levels of encryption of the secret. Finally, after $p+1$ activations, the intruder gets the secret.

More formally, we prove in the Appendix B the following lemma:

Lemma 8 *Let N be a Petri net and c_0 its initial configuration, let P the corresponding protocol constructed as described above then N, c_0 reaches the empty configuration if and only if P is not secret.*

Theorem 6 is a straightforward combination of Theorem 7 and Lemma 8. At this stage we have a model which allows to express pairing, encryption and hashing but, even if nonce creation is forbidden, this model is still undecidable. This motivates a small restriction, which is satisfied by all protocols we found in the literature, and which yields decidability. Informally, the restriction forbids to copy blindly two distinct dependent parts of the same message in the next one. This will be precised in section 6.1 where we give our decidability results, after a long detour through automata and set constraints.

3 Definite set constraints

3.1 Definite set constraints and intersection constraints

This class of set constraints has been introduced in [20] 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 symbols $\top, \bot$ 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.

Example 9 *Here is a definite set constraint:*

$$\begin{array}{rcl} f_2^{-1}(X) & \subseteq & g(Y) \\ g(Y) \cap Y & \subseteq & X \end{array} \qquad \begin{array}{rcl} f(f(X,Y) \cap X, X) & \subseteq & X \\ a & \subseteq & Y \end{array}$$

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:

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

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

Example 10 *(example 9 continued)*
Let $\sigma \stackrel{def}{=} [X \to \{a\}, Y \to \emptyset]$ *satisfies the set constraints dscribed in example 9.*

Following a standard translation (see e.g. [7]), the definite set constraints can be rewritten (in polynomial time) into *intersection constraints* which are conjunction of inclusions of one of the forms:

$$\begin{array}{lcllcl}
X & \subseteq & Y & f(X_1,\ldots,X_n) & \subseteq & X \\
X & \subseteq & f(Y_1,\ldots,Y_m) & f(X_1,\ldots,X_n) & \subseteq & g(Y_1,\ldots,Y_m)
\end{array}$$

where $X, X_1, \ldots, X_n, Y, Y_1, \ldots, Y_m$ are intersections of set variables. In other words, the constraints can be flattened and union and projections eliminated thanks (in particular) to the equivalence:

$$f_i^{-1}(X) \subseteq Y \quad \Leftrightarrow \quad X \cap f(\top,\ldots,\top) \subseteq f(\top,\ldots,Y,\top,\ldots)$$

where the Y is in ith position.

The translation τ from definite set constraints to intersection constraints may require the introduction of new variables. Formally, τ preserves the solutions:

Lemma 11 *σ is a solution of the definite set constraint C if and only if there exists σ', solution of the intersection constraint $\tau(C)$, such that σ is the restriction of σ' to the variables of C.*

Theorem 12 ([7]) *The satisfiability of intersection constraints (resp. definite set constraints) is DEXPTIME-complete and each satisfiable constraint has a least solution which is accepted by a finite tree automaton.*

Moreover, the decision procedure provides effectively the finite tree automaton accepting the least solution.

3.2 Intersection constraints with non-emptiness guards

Now, we consider a slight extension of intersection constraints, yielding a result similar to theorem 12. If e is a set expression, let $\mathsf{nonempty}(e)$ be a statement which is satisfied by σ iff $[\![e]\!]_\sigma$ is not empty.

We extend the formalism as follows. A *flat expression* is either an intersection of set variables or a set expression $f(X_1, \ldots, X_n)$ where $X_1, \ldots, X_n$ are intersections of set variables. An *intersection constraint with non-emptiness guards* is a conjunction of clauses

$$\mathsf{nonempty}(e'_1), ..., \mathsf{nonempty}(e'_n) \Rightarrow e_1 \subseteq e_2$$

where $e'_1, \ldots, e'_n, e_1, e_2$ are flat expressions.

The interpretation of such constraints is the expected one. Note that, of course, they extend intersection constraints. However, the algorithm given in [7] can be applied with slight changes only.

In other words, enriching the intersection constraints with clauses of the above form, we still have the same result as in theorem 12, as a corollary of [7]:

Theorem 13 *The satisfiability of intersection constraints with non-emptiness guards is DEXPTIME-complete and each satisfiable constraint has a least solution which is effectively accepted by a finite tree automaton.*

Proof: If we want to be as self-contained as possible, we need to reproduce, at least partly, the proof of [7]. In the next section, we will also rely on this proof.

First, we can assume that all expressions occurring in the guards also occur as members of inclusions (if necessary, add $e \subseteq e$ and flatten again).

Now, according to [7], we saturate the constraints using the inference rules given in figure 2.

In this figure, $X, X_1, \ldots, X_n, X', X'_1, \ldots, X'_n$ are intersection of set variables and $e, e', e_1, e_2, e'_1, e'_2$ are any flat set expressions. If these rules are applied to intersection constraints, we get as conclusions intersection constraints again, with the convention that expressions $f(e_1, \ldots, e_n) \cap f(e'_1, \ldots, e'_n)$ are eagerly normalize into $f(e_1 \cap e'_1, \ldots, e_n \cap e'_n)$.

As shown in [7] the rules of figure 2 are correct and applying the inference rules saturates the set constraint in deterministic exponential time (assuming that **reflexivity** and **weakening** do not introduce new variables).

For every constraint γ, we let γ^C be the saturated set. As in [7] again, let γ^S be the *solved form* of ϕ:

$$\gamma^S = \{f(e_1, \ldots, e_n) \subseteq X \in \gamma^C \mid \mathsf{nonempty}(f(e_1, \ldots, e_n)) \in \gamma^C\}$$

where X is a variable.

Reflexivity

$$\frac{}{e \subseteq e}$$

Transitivity

$$\frac{e_1 \subseteq e_2 \quad e_2 \subseteq e_3}{e_1 \subseteq e_3}$$

Weakening

$$\frac{}{X_1 \cap \ldots \cap X_n \subseteq X_i}$$

Compatibility

$$\frac{e_1 \subseteq e_2 \quad e_1' \subseteq e_2'}{e_1 \cap e_1' \subseteq e_2 \cap e_2'}$$

Propagation 1

$$\frac{\mathsf{nonempty}(e) \quad e \subseteq e'}{\mathsf{nonempty}(e')}$$

Propagation 2

$$\frac{\mathsf{nonempty}(X_1), \ldots, \mathsf{nonempty}(X_n)}{\mathsf{nonempty}(f(X_1, \ldots, X_n))}$$

Projection

$$\frac{\mathsf{nonempty}(f(X_1, \ldots, X_n)) \quad f(X_1, \ldots, X_n) \subseteq f(X_1', \ldots, X_n')}{X_i \subseteq X_i'}$$

Incompatibility

$$\frac{\mathsf{nonempty}(e) \quad e \subseteq f(X_1, \ldots, X_n) \quad e \subseteq g(X_1', \ldots, X_m')}{false} \quad \text{If } f \neq g$$

Cut

$$\frac{\mathsf{nonempty}(e) \quad \mathsf{nonempty}(e), \phi \Rightarrow e' \subseteq e''}{\phi \Rightarrow e \subseteq e'}$$

Figure 2: Inference rules for intersection constraints

γ^S is essentialy the definition of a tree automaton whose states are set variables. Let σ be the substitution, assigning to each variable X, the language recognized by this tree automaton in state X. We are going to prove that either $\mathit{false} \in \gamma^C$ or else σ is the least solution of γ. The minimality of σ comes from automata theory. Let us concentrate on the fact that σ is a solution of γ.

The proof that σ satisfies all inclusions $e_1 \subseteq e_2$ in ϕ^C is identical to [7]. Consider a clause $\mathsf{nonempty}(e_1), \ldots, \mathsf{nonempty}(e_n) \Rightarrow e \subseteq e'$ with $n \geq 1$. If there is an i such that $[\![e_i]\!]_\sigma$ is empty, then the clause is trivially satisfied. Otherwise, we may assume that every e_i is an intersection variable since σ satisfies $\mathsf{nonempty}(f(Y_1, \ldots, Y_n))$ if and only if it satisfies $\mathsf{nonempty}(Y_1), \ldots, \mathsf{nonempty}(Y_n)$. Then let e_i be the intersection $X_i^1 \cap \ldots \cap X_i^{k_i}$. For every i, there is a term t_i which is accepted by the tree automaton in every state X_i^j. We prove below that, if t is accepted in all states $X_1, \ldots, X_n$, then $\mathsf{nonempty}(X_1 \cap \ldots \cap X_n) \in \gamma^C$. Let us assume this for the moment. Then, by the rules **Cut** and **Propagation 2**, $e \subseteq e' \in \gamma^C$, which proves that σ satisfies $e \subseteq e'$, thanks to [7].

We prove by induction on the size of t that, if t is accepted in all states $X_1, \ldots, X_n$, then $\mathsf{nonempty}(X_1 \cap \ldots X_n) \in \gamma^C$.

- If t is a constant, by definition of the automaton, $t \subseteq X_i \in \gamma^C$ for every i. Then, by **Compatibility** (applied $n-1$ times), $t \subseteq X_1 \cap \ldots \cap X_n \in \gamma^C$
- If $t = f(t_1, \ldots, t_m)$. By definition of the automaton, there are inclusions $f(e_1^i, \ldots, e_m^i) \subseteq X_i \in \phi^C$ such that, for every $j \in [1..m]$, for every i, $t_j \in [\![e_j^i]\!]_\sigma$. Now, we apply the induction hypothesis: for every j, $\mathsf{nonempty}(e_j^1 \cap \ldots \cap e_j^n) \in \gamma^C$. By **Propagation 2**, $\mathsf{nonempty}(f(e_1^1 \cap \ldots \cap e_1^n, \ldots, e_m^1 \cap \ldots \cap e_m^n)) \in \gamma^C$. Now, by **Compatibility**,

  $$f(e_1^1 \cap \ldots \cap e_1^n, \ldots, e_m^1 \cap \ldots \cap e_m^n)) \subseteq X_1 \cap \ldots \cap X_n \in \gamma^C$$

  and, by **Propagation 1**, we conclude that $\mathsf{nonempty}(X_1 \cap \ldots \cap X_n) \in \gamma^C$.

To summarize: the assignment σ defined by the solved form γ^S also satisfies the conditional inclusions of γ^C, which means that γ^C is satisfiable whenever $\mathit{false} \notin \gamma^C$ and σ is then the minimal solution of γ.

□

In the proof of the last result, we have seen en passant that $\mathsf{nonempty}(X)$ is a logical consequence of the constraint iff it belongs to the solved form. It follows that:

Corollary 14 *Deciding whether the minimal solution of a definite set constraint with non-emptiness guards assigns the empty set to X is DEXPTIME-complete*

Actually, the DEXPTIME-hardness of this corollary is missing so far. But we can reduce the non-emptiness problem of the intersection of n tree automata (which is DEXPTIME-complete) by translating the definitions of the automata into intersection constraints and adding a clause

$$\mathsf{nonempty}(X_1 \cap \ldots \cap X_n) \Rightarrow a \subseteq X$$

where $X_1, \ldots, X_n$ are the set variables corresponding to the accepting states of the n automata respectively, X is a new variable and a is a constant.

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. We will see later as an example how to express pushdown automata in our formalism.

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 $\lambda f(\vec{x}).x_i$.

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 15 *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*

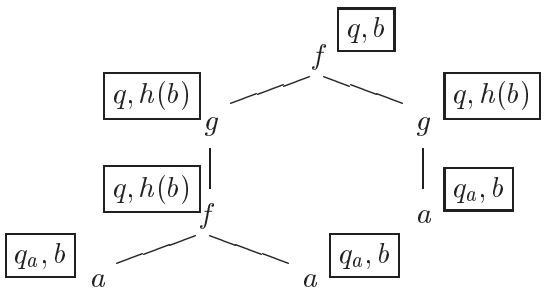


Figure 3: A tree t and a computation of A on t

- $f \in \mathcal{F}$ *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 equivalence classes w.r.t.* 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 16 *Assume that the transitions of the automaton A are (other components of the automaton are obvious from the context,* $\top$ *is the identity relation):*

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

A computation of the automaton on $f(g(f(a, a)), g(a))$ *is displayed on figure 3, in which the configurations reached at each node are displayed in a frame.*

Pushdown automata (on words) perform transitions $a, q, \alpha \cdot \gamma \to q', \beta \cdot \gamma$ 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: $a(q) \xrightarrow[\lambda x.\beta\alpha^{-1}x]{} q'$. If $w = a_1(\ldots a_k(_)\ldots)$, we use here the notation $w^{-1}(x)$ for $a_k^{-1}(\ldots(a_1^{-1}(x)))$, the additional subscript 1 being implicit for each letter, which has only one argument.

This translation does not make use of equality tests. Orthogonally, it is possible to simulate tree automata with equality tests between brothers [4]. This requires some coding, because the function F can refer to one representative for each class only, hence we cannot keep directly in the memory the subtree recognized so far. However, it is possible to show that any language recognized by an automaton with equality tests between brothers (and, more generally, with non-overlapping equality tests) is also accepted by an automaton with one memory. We don't need the projections here.

In some respect, our definition is a generalization of both models: we can both use a stack and check for equality, and keep record of deep subtrees. We avoid overlapping tests, which yield undecidability [22, 9], because we allow only one representative of each class in the function in the body of F.

Theorem 17 *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: 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 $< q, m >$. We prove that the sets M_q are the least solutions of the definite set constraint with non-emptiness guards C_A, consisting, for each transition rule $f(q_1, \ldots, q_n) \xrightarrow[F]{c} q$ of the inclusion

$$\mathsf{nonempty}(e_{q_{i_1}}), \ldots, \mathsf{nonempty}(e_{q_{i_k}}) \Rightarrow F(e_{q_{i_1}}, \ldots e_{q_{i_k}}) \subseteq X_q$$

and $e_{q_{i_j}}$ is the intersection for all indices l equivalent (w.r.t. c) to i_j of X_{q_l}.

C_A can be assumed to be an intersection constraint with non-emptiness guards (see section 3).

First, the assignment σ_0 which maps every X_q to M_q is a solution of the constraint. Indeed, consider any clause of the above form with $F = \lambda x_1, \ldots, x_k.G$ and assume (for simplicity) that $x_1, \ldots, x_r$ do not occur in G, while $x_{r+1}, \ldots, x_k$ occur (once) in G.

If $[\![e_{q_{i_j}}]\!]_{\sigma_0} \neq \emptyset$ for every i_j, then it is possible to reach configurations $< q_1, m_1 >, \ldots, < q_n, m_n >$ such that $m_j \in [\![e_{q_{i_j}}]\!]_{\sigma_0}$, i.e $m_i = m_j$ whenever $i = j \in c$. Now, consider any terms $m'_1, \ldots, m'_{k-r}$ respectively in $M_{q_{r+1}}, \ldots, M_k$. There are trees $t_1, \ldots, t_r, \ldots, t_k$ such that there are computations of the automaton on this trees yielding respectively the configurations $< q_1, m_1 >, \ldots, < q_r, m_r >, <$

$q_{r+1}, m'_1 >, \ldots, < q_k, m'_{k-r} >$. From these configurations, reading the input f, the automaton can move to the configuration $< q, G(m'_1, \ldots, m'_{k-r}) >$, hence $G(m'_1, \ldots, m'_{k-r}) \in M_q$.

Conversely, we have to prove that any solution σ of the constraint is larger than σ_0. Let $m \in M_q$. There is a computation of the automaton on some tree t, yielding the configuration $< q, m >$. We prove, by induction on t, that $m \in [\![X_q]\!]_\sigma$.

- If t is a constant, then there must be a rule $a \underset{F}{\longrightarrow} q$ and $F = m \in T(\Gamma)$. By definition, there is a constraint $F \subseteq X_q$. Hence $m \in [\![X_q]\!]_\sigma$.

- Now, let $t = f(t_1, \ldots, t_n)$ and let $f(q_1, \ldots, q_n) \underset{F}{\xrightarrow{c}} q$ be the last rule applied in the computation yielding $< m, q >$. Let moreover $< q_1, m_1 >, \ldots, < q_n, m_n >$ be the configurations corresponding to computations on $t_1, \ldots, t_n$. By definition, $m_i = m_j$ whenever $(i, j) \in c$ and $m = F(m_{i_1}, \ldots, m_{i_k})$. By induction hypothesis, for every i, $m_i \in [\![X_i]\!]_\sigma$ and, because of the equality constraints, $m_i \in [\![e_{q_{i_l}}]\!]_\sigma$ if $(i_l, i) \in c$. It follows that σ satisfies $\mathsf{nonempty}(e_{q_{i_j}})$ for all j and, since σ satisfies the clause associated with the rule, it satisfies $F(e_{q_{1_1}}, \ldots, e_{q_{i_k}}) \subseteq X_q$. In particular, $m \in [\![X_q]\!]_\sigma$.

This completes the proof that the sets M_q are the least solutions of the constraint C_A.

Then the non-emptiness of the language recognized by A reduces to the problem of deciding whether at least one of some designated variables gets a non-empty set in the least solution of the constraint. This is DEXPTIME-complete, thanks to corollary 14.

□

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 18 *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

5.1 Definition of the class

5.1.1 General set constraints with equality tests

We consider now definite set constraints as in section 3, with non-emptiness constraints and 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. We assume, without loss of generality, that there is no union and no projection symbol, which, as we have seen, is not a restriction (provided that the equality tests do not overlap projection symbols).

We use the standard notations on terms [16]. Let us recall some of them. A *position* will be a string of non-negative integers. A term t labeled with $\mathcal{F}$ can be seen as a mapping from the set $Pos(t)$ of its positions to $\mathcal{F}$. The subterm of t at position p is written $t|_p$.

An *equality constraint* c is an equivalence relation on a finite set of positions $P(c)$. We assume that no strict prefix of a position in $P(c)$ does belong to $P(c)$ (this restriction will be dropped in section 5.2.1). We will often write equality constraints as finite sets (or finite conjunctions) of expressions $p_1 = p_2$ where p_1, p_2 are positions. Then, it must be understood that c is the least equivalence relation containing the pairs (p_1, p_2) on the set of positions occurring in some of the equalities. We also say that a position p is *checked by* c when $p \in P(c)$.

A term t *satisfies* c, which we write $t \models c$, if every path in $P(c)$ is a position of t and moreover, $t|_{p_1} = t|_{p_2}$ (the subterms of t at positions p_1 and p_2 are identical).

We enrich the set expressions of section 3 with the construction $f^c(e_1, \ldots, e_n)$ where c is an equality constraint. These expressions are interpreted as follows:

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

The set of *paths* in an expression e is defined as follows:

$$\begin{aligned} \Pi(f^c(e_1, \ldots, e_n)) &\stackrel{\text{def}}{=} \{\epsilon\} \cup 1 \cdot \Pi(e_1) \cup \ldots \cup n \cdot \Pi(e_n) \\ \Pi(e_1 \cap e_2) &\stackrel{\text{def}}{=} \Pi(e_1) \cup \Pi(e_2) \\ \Pi(X) &\stackrel{\text{def}}{=} \{\epsilon\} \end{aligned}$$

Let $p \in \Pi(e)$. We let $e|_p$ be the set of subexpressions at position p:

$$\begin{array}{rcl} e|_\epsilon & \stackrel{\text{def}}{=} & \{e\} \\ (e_1 \cap e_2)|_{i\cdot p} & \stackrel{\text{def}}{=} & e_1|_{i\cdot p} \cup e_2|_{i\cdot p} \\ (f^c(e_1,\ldots,e_n))|_{i\cdot p} & \stackrel{\text{def}}{=} & e_i|_p \\ X|_{i\cdot p} & \stackrel{\text{def}}{=} & \emptyset \end{array}$$

When $e|_p$ contains only one element, we confuse $e|_p$ with this element and say that $e|_p$ is *the subexpression of e at p*.

We will assume that, in every expression $f^c(e_1,\ldots,e_n)$,

$$P(c) \subseteq \Pi(f^c(e_1,\ldots,e_n))$$

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 19 *$f^{21=12}(f(Z,Y) \cap X, g(X) \cap Y) \subseteq f(Y, g(X))$ 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 $[\![f^{12=21}(f(Z,Y) \cap X, g(X) \cap Y)]\!]_\sigma = \{f(f(a,b), g(b))\}$*

5.1.2 A complete deduction system

We first design a complete deduction system and show that every satisfiable set constraint has a least solution. These results are not meant to be practical.

Let S be a set constraint as in the previous section, whose variables are $Var(S) = \{X_1,\ldots,X_n\}$. Let Σ be the subset of $(2^{T(\mathcal{F})})^n$ of assignments σ mapping every variable X_i to a finite set.

We may assume in this section that, in every clause

$$\phi \Rightarrow e \subseteq e'$$

the expression e' does not contain intersection symbols. This is not a restriction as a clause $\phi \Rightarrow e \subseteq e'[e_1 \cap e_2]$ is equivalent to the two clauses $\phi \Rightarrow e \subseteq e'[e_i]$ for $i = 1, 2$.

With this assumption, we can associate with each right hand side of an inclusion e' a term $t_{e'} \in T(\mathcal{F}, \mathcal{X})$ such that every variable occurs only once in $t_{e'}$ and e' is obtained from $t_{e'}$ by substituting set variables to the variables of $t_{e'}$.

We define the *one step deduction relation* T_S on $\Sigma \cup \{\Box\}$ as follows:

$T_S(\Box) \stackrel{\text{def}}{=} \Box$

If there is a clause $\phi \Rightarrow e \subseteq e'$ in S such that $\sigma \models \phi$ and there is a $t \in [\![e]\!]_\sigma$ such that t is not an instance of $t_{e'}$, then $T_S(\sigma) = \Box$.

Otherwise, for each clause $C = \phi \Rightarrow e \subseteq e'$ in S

- If $\sigma \not\models \phi$, then we let θ_C be the assignments mapping every set variable to the empty set
- If $\sigma \models \phi$, then for every term $t \in [\![e]\!]_\sigma$, and every set variable X, we let $\theta_{t,C}(X)$ be the set of terms $t|_p$ such that $e'|_p = X$.
  Finally, we define $T_S(\sigma)$ by:

$$[\![X_i]\!]_{T_S(\sigma)} \stackrel{\text{def}}{=} [\![X_i]\!]_\sigma \cup \{\theta_{t,C}(X_i) \mid C \in S, t \in [\![e]\!]_\sigma\}$$

We let $\sigma_\emptyset$ be the assignment mapping every set variable to the empty set and we define σ^ω as the least fixed point of $\vdash$:

$$\sigma^\omega(X_j) = \bigcup_{i=1}^{\infty} T_S^i(\sigma_\emptyset)(X_j)$$

if $T_S^i(\sigma_\emptyset) \neq \Box$ for every i and $\sigma^\omega = \Box$ otherwise.

Proposition 20 *$\sigma^\omega = \Box$ iff S is not satisfiable.*
If $\sigma^\omega \neq \Box$, it is the least solution of S.

Proof: It is similar to the standard result that the least fixed point of the direct consequence operator of a Horn clause set is the least model of the program.

If $\sigma^\omega \neq \Box$, then σ^ω is contained in any solution of S (by induction on i, $T_S^i(\sigma_\emptyset)$ is contained in any solution of S).

Now, if $\sigma^\omega \neq \Box$, then σ^ω is a solution of S: this is a routine verification. $\Box$

5.1.3 An undecidability result

As a consequence of undecidability results on tree automata with equality tests (see e.g. [9]), the satisfiability of such general constraints is undecidable, because of possible overlapping tests.

Proposition 21 *The satisfiability of such general constraints (even without non-emptiness preconditions) is undecidable.*

Note that such a result is consistent with theorem 6 and the translation of security problems into set constraints as given in section 6.1. We sketch the proof of this proposition, because, even if the reader should already be convinced, the proof sheds some lights on the restrictions we take later on.

Proof: (sketch) We encode Turing machine computations. A configuration is represented as a triple containing the state, the part of the tape on the left of the

head (including the head position) and the part of the tape on the right of the head. Tape contents are encoded using unary symbols (one for each element in the tape alphabet), in such a way that symbols which are close to the head appear on the top of the terms. For instance a tape content $\overset{abaabaab}{\uparrow}$ is represented by the words $b(a(a(b(a(0)))))$, $a(a(b(0)))$. We use a binary tupling symbol $< _, _, _ >$ to put together the two components of the tape and the state. Now, for instance, with each transition rule $< q, a > \rightarrow < q', b, left >$ we associate the constraint:

$$f^{12=2121,131=213}(< q', X, b(Y) >, Z \cap f(< q, a(X), Y >, \top)) \subseteq Z$$

X, Y being tape contents, the equality tests ensure that we keep the same remaining tape contents when we move from one state to another.

The idea is that the least solution σ_0 of the constraint will assign to Z the (encoding of the) set of computations of the Turing machine. Adding

$$Z \cap f(< q_f, \top, \top >, \top) \subseteq X_0$$

for the final states and

$$< q_0, 0, 0 > \subseteq Z$$

for the initial state, the emptiness of $\sigma_0(X_0)$ is equivalent to the halting problem (i.e. the reachability of the state q_f). □

5.1.4 Basic variables and expressions

That is why we are going to put more restrictions on the constraints. The idea is to divide the set variables into two sets: the basic ones which correspond to sets of terms whose only a fixed part can be seen. This corresponds to non-invertible symbols in section 2.2. We do not impose any restrictions on the equality tests for such basic variables since, intuitively, the non-invertible symbols impose a boarder in the terms under which no test takes place, hence limiting the overlaps of equalities which yield the undecidability result.

For non-basic variables, we impose a restriction, which, roughly, allows to check the equalities using one memory only. The goal is of course to use the results of section 4.

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.

We introduce now *one-way function symbols* of a constraint S. This notion is of course related to the one-way function symbols of section 2 (it is a generalization). Intuitively, a symbol is one-way in a constraint S if, in each of its applications, there is no way to look at the subterms. "Looking at the subterms"

occur in two cases: when we apply a "projection" (i.e. when there is an inclusion whose right member is headed with that symbol) and when we check for equality some subterms.

Definition 22 *A function symbol g is* one-way *in a set constraint S if*

- *it does not occur on the right of an inclusion constraint of S*
- *in any expression $e = f^c(e_1, \ldots, e_n)$ occurring in S, for every $\pi \in P(c)$ and for every strict prefix π' of π, $e|_{\pi'}$ does not contain any expression headed with g.*

Let $\mathcal{OF}(S)$ be the set of one-way function symbols in S.

Definition 23 *The* basic variables *of a set constraint γ is the largest set of variables occurring in γ such that*

- *If X is basic then $\mathcal{R}(X)$ only contains one-way symbols and basic variables.*
- *If X is basic then*
  - *either $\mathcal{R}(X)$ contains only one clause $\phi \Rightarrow e \subseteq X$ such that X does not occur in e.*
  - *or every function symbol occurring in $\mathcal{R}(X)$ occurs (possibly) only in $\mathcal{R}(Y)$ where Y is basic.*

Intuitively, the function symbols used recursively to construct basic variables cannot be used for non-basic variables.

Example 24 *The following example is inspired by examples from section 2. Let* Nat, A, DA, HA, M, Key, In *be set variables and γ consist of:*

$$\begin{array}{rcl@{\qquad}rcl} 0 & \subseteq & \mathsf{Nat} & succ(\mathsf{Nat}) & \subseteq & \mathsf{Nat} \\ da(\mathsf{Nat}) & \subseteq & \mathsf{DA} & ha(\mathsf{Nat}) & \subseteq & \mathsf{HA} \\ \mathsf{DA} & \subseteq & \mathsf{A} & \mathsf{HA} & \subseteq & \mathsf{A} \\ K(\mathsf{A},\mathsf{A}) & \subseteq & \mathsf{Key} & \mathsf{shr}(\mathsf{A}) & \subseteq & \mathsf{M} \\ \mathsf{A} & \subseteq & \mathsf{M} & \mathsf{Key} & \subseteq & \mathsf{M} \\ <\mathsf{M},\mathsf{M}> & \subseteq & \mathsf{M} & \{\mathsf{M}\}_{\mathsf{M}} & \subseteq & \mathsf{M} \end{array}$$

Intruder capabilities such as:

$$\begin{array}{rcl@{\qquad}rcl} <\mathsf{In},\mathsf{In}> & \subseteq & \mathsf{In} & \{\mathsf{In}\}_{\mathsf{In}} & \subseteq & \mathsf{In} \\ \mathsf{In} \cap <\top,\top> & \subseteq & <\mathsf{In},\mathsf{In}> & \mathsf{In} \cap \{\top\}_{\mathsf{In}} & \subseteq & \{\mathsf{In}\}_{\mathsf{In}} \\ \mathsf{A} & \subseteq & \mathsf{In} & \mathsf{shr}(\mathsf{DA}) & \subseteq & \mathsf{In} \\ \mathsf{In} \cap <\top,\top,\top> & \subseteq & <\mathsf{In},\mathsf{In},\mathsf{In}> & & & \end{array}$$

And protocol-specific constraints such as

$$\begin{array}{rcl} < \mathsf{A}, \mathsf{A} > & \subseteq & \mathsf{In} \\ <>^c (\{< \mathsf{A}, \mathsf{Key}, \mathsf{M} >\}_{\mathsf{shr(A)}} \cap \mathsf{In}, \{m(\mathsf{A}, \mathsf{A})\}_{\mathsf{Key}}, \mathsf{M}) & \subseteq & \mathsf{In} \end{array}$$

where c stands here for $121 = 211 \wedge 111 = 212 \wedge 112 = 22 \wedge 113 = 3$. (We will see in section 6.1 how to translate cryptographic protocol into set constraints and, in particular, we will develop a full example). In this example, all function symbols are one-way, except the tupling $< _, _ >$ and $< _, _, _ >$ and encryption $\{_\}__$, because of the intruder's constraints.

Then In *and* M *are not basic while all other variables are basic.*

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 $f(e_1, \ldots, e_n)$ (or $f^c(e_1, \ldots, e_n)$) and $e_1, \ldots, e_n$ are basic

5.1.5 Our assumptions

Definition 25 (Basicness condition) *An expression e (resp. an equality test c in an expression $f^c(e_1, \ldots, e_n)$)* satisfies the basicness condition *(w.r.t. a set of basic variables) if for each expression $f^c(e_1, ..., e_n)$*

$$\left.\begin{array}{r} p \cdot i \cdot q \sim_c p' \\ i \neq j \\ p' \not\geq_{pref} p \\ \forall w,\ p \cdot j \cdot w \not\sim_c p \cdot i \cdot q \end{array}\right\} \Rightarrow \begin{array}{l} \textit{There are positions } p_1, p_2 \textit{ such that} \\ p_1 \sim_c p', p_2 \sim_c p \cdot j \textit{ and} \\ \textit{either } e|_{p_1} \textit{ or } e|_{p_2} \\ \textit{contains basic expressions only} \end{array}$$

where $\geq_{pref}$ is the prefix ordering on positions.

The situation is depicted on figure 4: one of the three terminal positions on the picture should hold basic expressions only.

Example 26 *Let us examine the last constraint which is displayed in example 24.*

$$<>^c (\{< \mathsf{A}, \mathsf{Key}, \mathsf{M} >\}_{\mathsf{shr(A)}} \cap \mathsf{In}, \{m(\mathsf{A}, \mathsf{A})\}_{\mathsf{Key}}, \mathsf{M}) \subseteq \mathsf{In}$$

where c stands for $121 = 211 \wedge 111 = 212 \wedge 112 = 22 \wedge 113 = 3$. At positions 121,111, there is only one subexpression A*, which is basic. At position 112, there is only one subexpression:* Key*, which is also basic. Remain the positions* $3, 113$*, which do not hold basic expressions.*

In the definition, p' can only be 3 and $p \in \{1, 11\}$. The basicness condition reduces to check that subexpressions at positions $12, 111, 113$ are basic, which is the case here.

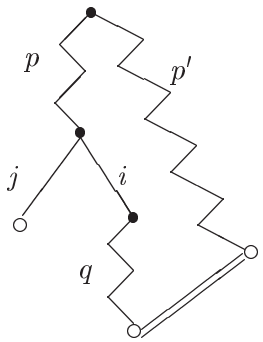


Figure 4: The basicness condition

Note that, if there are only two occurrences of non-basic variables in the expression, then the basicness condition is always satisfied.

The basicness condition looks a bit complicated, but let us give more intuition.

From tree automaton point of view, while computing on the trees bottom-up, we want to be able to check the equalities without carrying more than one memory at each node. The difficulty is that we need the stability under intersection of this property.

Consider for instance the following constraints:

$$\begin{array}{lll} f^{12=2}(X \cap f(X,Y),Y) & \subseteq & Y \\ f^{11=2}(Y \cap f(X,Y),X) & \subseteq & X \end{array}$$

Only one memory is sufficient to recognize the instances of any of the two constraints. Putting them together, however, we can derive

$$f^{12=2}(f^{11=2}(Y \cap f(\mathbf{X},Y),X) \cap f(X,Y),Y) \subseteq Y.$$

Now, we need 2 memories to accept the instances of the left hand side since, when reaching $\mathbf{X}$ we must keep this term in the memory (it is checked for equality higher up) and we must also keep in the memory $f(\mathbf{X},Y)$, which is also checked for equality higher up. Note that here X, Y are not basic since f is not a one-way function symbol.

Actually, a more natural, weaker, condition would be to assume that, in any c, if p, q are in two different equivalence classes, then they do not share any prefix. Imposing such a condition only yields an undecidable class of constraints.

On the other hand, if we intersect a basic expression with any expression, the result is a basic expression. Hence, the basicness condition expresses roughly that on the sides of a path checked for equality, we only find basic expressions,

preventing to have to keep additional information when we intersect with another expression.

The basicness condition is also relevant for our application, as we will see.

The constraints satisfying the basicness condition are called *set constraints with equality tests* (*ET-constraints* for short). Let us summarize:

Definition 27 *An* ET-constraint *is a finite conjunction of clauses*

$$\phi \Rightarrow e \subseteq e'$$

in which e, e' *are set expressions built using*

- *Set variables*
- *The constant symbol* $\bot$
- *Intersection*
- *Function symbol application* $f^c(\ldots)$ *or* $f(\ldots)$.

We assume:

- *That right hand sides (the expression* e' *above) do not make use of the constructions* f^c *with a non-empty* c.
- *For every construction* $f^c(e_1, \ldots, e_n)$, $P(c) \subseteq \Pi(f^c(e_1, \ldots, e_n))$
- *The basicness condition*

ET-constraints contain properly intersection constraints since 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].

5.2 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.

** description of the structure of the section **

We start with some simplifications of the constraints.

5.2.1 Normalization

For every expression e, let us define two notions of size:

- $|e|_{\mathcal{F}}$ is the cardinal of $\bigcup_{p\in\Pi(e)} e|_p$. This is proportional to the memory size, which is required to store the expression, regardless to the equality tests.
- $|e|_t$ is the sum, for every expression $f^c(e_1,\ldots,e_n) \in \bigcup_{p\in\Pi(e)} e|_p$ of the size of c. The size of an individual test c is the sum of sizes of positions checked by c.

The goal of the first transformation step (Normalization) is to reduce the expression to a normal form.

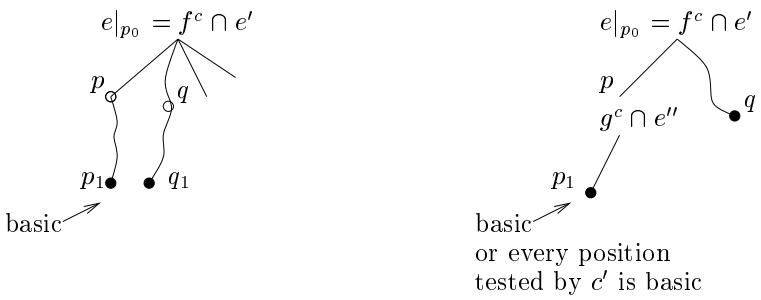
Definition 28 *An expression e is* normal *if the following conditions are satisfied for e:*

1. *All subexpressions of e satisfy the basicness condition*
2. *If $f^c(e_1,\ldots,e_n) \cap e' \in e|_{p_0}$, $p \sim_c q$ and $p\cdot p_1 \sim_c q_1$ for a non-trivial p_1, then $e|_{p_0\cdot p\cdot p_1}$ is a basic expression. (*"For ancestor positions, the lowest one is basic"*).*
3. *For every $p \in \Pi(e)$, if $g^c(e_1,\ldots,e_n) \cap e' \in e|_p$, then, for every $p_1 \sim_c p_2$, the subexpressions at positions p_1 and p_2 in $g^c(e_1,\ldots,e_n)$ are identical.*
4. *for every equality test c occurring in e, every equivalence class of c contains at least two positions which do not share any non-trivial prefix*
5. *For every $p \in \Pi(e)$, $e|_p$ is either an intersection of variables or an intersection $g^c(e_1,\ldots,e_n) \cap X_1 \cap \ldots \cap X_m$, in which case, for every $p \in \Pi(e)$, $e|_p$ is a singleton.*
6. *If $f^c(e_1,\ldots,e_n) \cap e' \in e|_{p_0}$, $p\cdot p_1 \sim_c q$ for non-empty p, p_1, $g^{c'}(e'_1,\ldots,e'_m) \cap e'' \in e|_{p_0\cdot p}$, then either $e|_{p_0\cdot p\cdot p_1}$ is a basic expression, or else for every $p' \sim_{c'} q'$, $e|_{p_0\cdot p\cdot p'}$ is a basic expression (*"For overlapping tests, the lowest one is basic "*).*

Conditions 1 and 2 are satisfied initially. Actually, even a property stronger than condition 2 is initially satisfied since, so far, any two distinct positions in $P(c)$ are incomparable w.r.t. the prefix ordering. We need however this weaker property to keep it invariant.

Properties 2, 6 are illustrated on figure 5.

The main result of this section, whose proof is quite long and technical and is given in appendix C is the following:



Property 2 Property 6

Figure 5: The properties 2 and 6

Lemma 29 *Every expression e which satisfies the basicness condition can be transformed in a normal expression e' such that, for every σ, $[\![e]\!]_\sigma = [\![e']\!]_\sigma$.*

We also conjecture that the transformation, as described in the appendix, yields an expression e' such that $|e'|_{\mathcal{F}}$ and $|e'|_t$ are polynomially bounded by $|e|_t$ and $|e|_F$.

As a side consequence, the subexpression at a given position is now defined in a unique way:

Lemma 30 *If e is an expression satisfying condition 5, then for every $p \in \Pi(e)$, $e|_p$ is a singleton.*

Proof: We prove that $e|_p$ is a singleton for every $p \in \Pi(e)$ by induction on e. If e is a variable or a constant, then $e|_p$ is a singleton consisting in e itself.

Now, assuming e satisfies condition 5, e is either an intersection of variables or an expression $f^c(e_1, \ldots, e_n) \cap X_1, \ldots \cap X_n$. In the first case $\Pi(e) = \{\epsilon\}$ and $e|_p = \{e\}$ by definition. In the latter case, if $p = i \cdot p'$, then $e|_p = e_i|_{p'} \cup X_1|_{i \cdot p'} \ldots = e_i|_{p'}$ by definition. And, by induction hypothesis, $e_i|_{p'}$ is a singleton. □

So, now, we can use the terminology "the subexpression at position p", as well as replacement at position p: $C[e]_p$ means either (this will be unambiguous from the context) that e occurs at position p in the expression $C[e]_p$ or that we have replaced the subexpression at position p with e.

The normalization is extended to constraints: every expression occurring in the constraint can be assumed normal thanks to lemma 29.

5.2.2 Abstractions

We abstract out subexpressions introducing new variables, as long as this preserves the form of the constraints. For instance, for contexts $C[\ ]_p$, an inclusion $e \subseteq C[f(\vec{e'})]_p$ becomes $C[X]_p \subseteq e, f(\vec{e'}) \subseteq X$ 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 $f(X_1, \ldots, X_n)$.

More formally, we use the following rules, assuming that $n \geq 2$ and p is not the root position:

$$\begin{array}{lrcl}
(A_1) & \phi \Rightarrow f^c(\vec{e}) \cap e_1 \subseteq e' & \rightarrow & (\exists X) f^c(\vec{e}) \subseteq X, \phi \Rightarrow X \cap e_1 \subseteq e' \\
(A_2) & \phi \Rightarrow e \subseteq C[f(\vec{e'})]_p & \rightarrow & (\exists X) \phi \Rightarrow e \subseteq C[X]_p, f(\vec{e'}) \subseteq X, X \subseteq f(\vec{e'}) \\
(A_3) & \phi \Rightarrow e \subseteq C[e_1 \cap \ldots \cap e_n]_p & \rightarrow & (\exists X) \phi \Rightarrow e \subseteq C[X]_p, e_1 \cap \ldots \cap e_n \subseteq X
\end{array}$$

In these rules, X is a new variable: we assume that there is no capture. The following lemma is a consequence of the definitions:

Lemma 31 *Applying abstraction does terminate on any constraint S, resulting in a constraint S' such that the solutions of S are the restrictions of solutions of S' to the free variables of S. Moreover, if S is an ET-constraint, then so is S' and if every expression is normal in S, then every expression is normal in S'.*

We can also abstract out in the conditions of the inclusions. However, using such a rule in an unrestricted way would lead to non termination of the saturation. That is why we are going to use it only once, to simplify the original constraint and forget it afterwards:

$$\begin{array}{ll}
(A_4) & \mathsf{nonempty}(e), \phi \Rightarrow e_1 \subseteq e_2 \\
& \rightarrow (\exists Y) e \subseteq Y \quad \wedge \quad \mathsf{nonempty}(Y), \phi \Rightarrow e_1 \subseteq e_2
\end{array}$$

In this rule, e is assumed not to be a variable. It is also assumed that there is no capture (Y is a new variable).

Lemma 32 (A_4) *preserves the solutions of the constraint.*

Proof: Assume $S \rightarrow S'$ using the rule (A_4). If σ is a solution of S, extending σ with $Y \mapsto [\![e]\!]_\sigma$ yields a solution of S'.

Conversely, if σ' is a solution of S', then its restriction σ to variables other than Y is a solution of S: either $\sigma \not\models \mathsf{nonempty}(e), \phi$, in which case σ satisfies $\mathsf{nonempty}(e), \phi \Rightarrow e_1 \subseteq e_2$, or else $[\![e]\!]_\sigma$ is non-empty. In the latter case, $[\![Y]\!]_{\sigma'}$ is non-empty (because of the constraint $e \subseteq Y$) and $[\![e_1]\!]_\sigma = [\![e_1]\!]_{\sigma'} \subseteq [\![e_2]\!]_{\sigma'} = [\![e_2]\!]_\sigma$. $\square$

Inspecting the normal forms w.r.t. $(A_1), (A_2), (A_3), (A_4)$, together with our hypotheses, the atomic constraints are now of the form $\phi \Rightarrow e \subseteq e'$ where ϕ is a conjunction of $\mathsf{nonempty}(X)$, e' is of the form $f(X_1, \ldots, X_n)$ or $X_1 \cap \ldots \cap X_n$ and e is either $X_1 \cap \ldots \cap X_n$ or $f^c(\vec{e})$.

5.2.3 Getting rid of basic variables

Next, we can get rid of basic type variables. The main idea is that we can replace each basic variable with a suitably chosen finite set, while keeping the desirable properties. This is described in the next lemmas.

We let $\mathcal{B}(S)$ be the set of basic variables of S. We split each ET-constraint into two parts $S = S_B \uplus S_{NB}$: S_B is the union of $\mathcal{R}(X)$ for $X \in \mathcal{B}(S)$ and S_{NB} is the remaining constraint.
Remember that for any basic variable, either every function symbols of $\mathcal{R}(X)$ occurs only in S_B (first type) or $\mathcal{R}(X)$ contains only one clause on the form $\phi \Rightarrow C_X(X_1, \ldots, X_k) \subseteq X$ where X is distinct from the X_i (second type). We first get ride of the basic variables X of second type by replacing them by the clause $C_X(X_1, \ldots, X_k)$.

Lemma 33 *Given an ET-constraint S, let*

$$S' \stackrel{def}{=} S[X \mapsto C_X(X_1, \ldots, X_k)]_{X \text{ of second type}}.$$

Then S' is an ET-constraint and S is satisfiable if and only if S' is satisfiable.

Proof: Since $C_X(X_1, \ldots, X_k)$ contains only one-way function symbols and basic variables, $C_X(X_1, \ldots, X_k)$ is a basic expression thus S' is an ET-constraint.
If σ is a solution of S', then σ extended to the basic variables of second type by $\sigma(X) = [\![C_X(X_1, \ldots, X_k)]\!]_\sigma$ is clearly a solution of S.
Conversely, if S is satisfiable, then S has a minimal solution σ. By minimality of σ, we have necessarily $[\![X]\!]_\sigma = [\![C_X(X_1, \ldots, X_k)]\!]_\sigma$. Thus σ is solution of S'.
□
From now on, we consider only ET-constraints with only basic variables of second type. In particular, the functions symbols occurring in S_B do not occur in S_{NB}.

Example 34 *We consider some of the constraints presented in Example 24.*

$$S_B \quad \left[\begin{array}{rcl@{\qquad}rcl} 0 & \subseteq & \mathsf{Nat} & succ(\mathsf{Nat}) & \subseteq & \mathsf{Nat} \\ da(\mathsf{Nat}) & \subseteq & \mathsf{DA} & ha(\mathsf{Nat}) & \subseteq & \mathsf{HA} \\ \mathsf{DA} & \subseteq & \mathsf{A} & \mathsf{HA} & \subseteq & \mathsf{A} \\ K(\mathsf{A},\mathsf{A}) & \subseteq & \mathsf{Key} & & & \end{array}\right.$$

$$S_{NB} \quad \left[\begin{array}{rcl@{\qquad}rcl} \mathsf{A} & \subseteq & \mathsf{M} & \mathsf{Key} & \subseteq & \mathsf{M} \\ <\mathsf{M},\mathsf{M}> & \subseteq & \mathsf{M} & \{\mathsf{M}\}_{\mathsf{M}} & \subseteq & \mathsf{M} \\ \mathsf{shr}(\mathsf{A}) & \subseteq & \mathsf{M} & & & \end{array}\right.$$

Then there is one basic variable of second type: Key. *Thus we transform our ET-constraint following lemma 33:*

$$S_B \quad \left[\begin{array}{rcl@{\qquad}rcl} 0 & \subseteq & \mathsf{Nat} & succ(\mathsf{Nat}) & \subseteq & \mathsf{Nat} \\ da(\mathsf{Nat}) & \subseteq & \mathsf{DA} & ha(\mathsf{Nat}) & \subseteq & \mathsf{HA} \\ \mathsf{DA} & \subseteq & \mathsf{A} & \mathsf{HA} & \subseteq & \mathsf{A} \end{array}\right.$$

$$S_{NB} \quad \left[\begin{array}{rcl@{\qquad}rcl} \mathsf{A} & \subseteq & \mathsf{M} & K(\mathsf{A},\mathsf{A}) & \subseteq & \mathsf{M} \\ <\mathsf{M},\mathsf{M}> & \subseteq & \mathsf{M} & \{\mathsf{M}\}_{\mathsf{M}} & \subseteq & \mathsf{M} \\ \mathsf{shr}(\mathsf{A}) & \subseteq & \mathsf{M} & & & \end{array}\right.$$

Lemma 35 *S_B has a least solution σ_m. It is possible to compute a finite tree automaton $\mathcal{A}_m$, whose states are finite sets of variables in S_B and such that $\sigma_m(X)$ is the set of trees accepted in the state $\{X\}$.*

Proof:

Since one-way function symbols do not occur on the right of inclusions, in any constraint $\phi \Rightarrow e \subseteq e'$, e' is an intersection of variables. Then S_B is satisfiable: assigning every variable to $T(\mathcal{OF})$ is a solution. Then, by proposition 20 there is a minimal solution σ_m.

We can also easily construct the minimal solution in an effective way, applying e.g. the saturation rules of figure 2 to this particular case: because there are only one-way functions in S_B, there is no construction $f^c(\ldots)$ here. The saturated constraint coincides here with the solved form (since there is no function symbols on the right).

As in section 3.2, the solved form corresponds to a tree automaton $\mathcal{A}_m$ whose states are set variables. □

Let $\approx$ be any equivalence relation on $T(\mathcal{OF})$. $\approx$ is extended to the least congruence relation on $T(\mathcal{F})$, which we write again $\approx$. Then, every assignment σ from the set of variables to $2^{T(\mathcal{F})}$ is extended into the assignment $\sigma_\approx$ defined by:

$$\sigma_\approx(X) \stackrel{\text{def}}{=} \{t \in T(\mathcal{F}) \mid \exists u \in T(\mathcal{F}), t \approx u, u \in \sigma(X)\}.$$

in other words, σ is saturated by $\approx$.

Lemma 36 *For every equivalence relation $\approx$ on $T(\mathcal{OF})$, if σ is a solution of an ET-constraint S, in normal form w.r.t.* **Norm**, $(A_1), (A_2), (A_3), (A_4)$, *then $\sigma_\approx$ is a solution of S.*

Proof: Assume $\phi \Rightarrow e \subseteq e' \in S$. Since ϕ only consists of formulas $\mathsf{nonempty}(X)$ where X is a variable, $\sigma \models \phi$ if and only if $\sigma_\approx \models \phi$.

Let $t \in [\![e]\!]_\sigma$ and $u \approx t$.

We prove, by induction on the size of e' that

$$t \in [\![e']\!]_\sigma \Rightarrow u \in [\![e']\!]_{\sigma\approx}$$

- If e' is a variable, the implication follows from the congruence property of $\approx$.
- If $e' = e_1 \cap e_2$, this is straightforward
- if $e' = f(e_1, \ldots, e_n)$ then $f \notin \mathcal{OF}$ (by definition of one-way symbols). From $t \in [\![e']\!]$, it follows that $t = f(t_1, \ldots, t_n)$. Then $u = f(u_1, \ldots, u_n)$ with $t_i \approx u_i$ for every i, since $f \notin \mathcal{OF}(S)$. By induction hypothesis, for every i, $t_i \in [\![e_i]\!]_\sigma$, hence $u_i \in [\![e_i]\!]_{\sigma\approx}$, therefore $u \in [\![e']\!]_{\sigma\approx}$.

□

Now, the idea is to construct a finite index relation $\approx$ such that we may interpret the basic variables in a set of representatives modulo $\approx$.

Lemma 37 *There is a congruence $\approx$ and an assignment σ^0 to the variables of S such that $\sigma^0_\approx = \sigma_m$ and $\sigma^0(X)$ is finite for every variable X.*

Proof: We use the automaton describing σ_m and we consider $\approx$ defined by $t \approx u$ iff t and u are accepted in exactly the same states of the automaton. Let R be a set of representatives for $\approx$ such that if $f(t_1, \ldots, t_n)$ is in R then, $t_1, \ldots, t_n$ are also in R. σ^0 assigns $R \cap [\![X]\!]_{\sigma_m}$ to X. □

Remark : Note that every element of R is recognized by at least one state X of the automaton.

Example 38 *We consider again the constraints presented in Example 34. The automaton $\mathcal{A}_m$ associated with S_B is the following one:*

$$\begin{array}{rcl@{\qquad}rcl} 0 & \to & q_{\mathsf{Nat}} & succ(q_{\mathsf{Nat}}) & \to & q_{\mathsf{Nat}} \\ da(q_{\mathsf{Nat}}) & \to & q_{\mathsf{DA}} & ha(q_{\mathsf{Nat}}) & \to & q_{\mathsf{HA}} \\ q_{\mathsf{DA}} & \to & q_{\mathsf{A}} & q_{\mathsf{HA}} & \to & q_{\mathsf{A}} \end{array}$$

Thus the equivalence classes are:

$$\{succ^n(0) | n \in \mathbb{N}\}, \{da(succ^n(0)) | n \in \mathbb{N}\}, \{ha(succ^n(0)) | n \in \mathbb{N}\}$$

We choose $R = \{0, ha(0), da(0)\}$ and $\sigma^0(\mathsf{Nat}) = \{0\}$, $\sigma^0(\mathsf{HA}) = \{ha(0)\}$, $\sigma^0(\mathsf{DA}) = \{da(0)\}$, $\sigma^0(\mathtt{Agent}) = \{ha(0), da(0)\}$.

If σ is an assignment of basic set variables to finite set of terms in $T(\mathcal{OF})$, then $\sigma(S_{NB})$ is the set constraint obtained, replacing each basic variable X with $\bigcup_{t\in\sigma(X)} t$. Since basic variables do not occur on the right hand sides of inclusions in S_{NB}, $\sigma(S_{NB})$ can be normalized in an ET-constraint, removing unions occurring on the left or in the conditions by duplicating the constraints.

Example 39 *In our example 38, $\sigma^0(S_{NB})$ is equal to:*

$$\begin{array}{rcl@{\qquad}rcl}
ha(0) & \subseteq & \mathsf{M} & da(0) & \subseteq & \mathsf{M} \\
K(ha(0),ha(0)) & \subseteq & \mathsf{M} & K(ha(0),da(0)) & \subseteq & \mathsf{M} \\
K(da(0),ha(0)) & \subseteq & \mathsf{M} & K(da(0),da(0)) & \subseteq & \mathsf{M} \\
< \mathsf{M},\mathsf{M} > & \subseteq & \mathsf{M} & \{\mathsf{M}\}_{\mathsf{M}} & \subseteq & \mathsf{M} \\
\mathsf{shr}(ha(0)) & \subseteq & \mathsf{M} & \mathsf{shr}(da(0)) & \subseteq & \mathsf{M}
\end{array}$$

Lemma 40 *Let σ^0 be the restriction to basic variables of the assignment defined as in lemma 37. Let $\approx$ be the congruence as defined in lemma 37. Then S is satisfiable iff $\sigma^0(S_{NB} \cup S_{OF})$ is satisfiable.*

Proof: First assume that S is satisfiable and σ is a solution of S. Thanks to lemma 36, we can assume w.l.o.g. that $\sigma = \sigma_{\approx}$. Let us construct σ' such that $\sigma'_{\approx} = \sigma$ and σ' is a solution of $\sigma^0(S_{NB})$.

For every term $t \in T(F)$, let $t\!\downarrow$ be its representative for $\approx$. We define

$$\sigma'(X) = \{t\!\downarrow \mid t \in \sigma(X)\}$$

Let us prove that σ' is a solution of $\sigma^0(S_{NB})$. Let $\phi \Rightarrow e \subseteq e'$ in S_{NB}. If $\sigma' \not\models \phi$ then σ' satisfies the clause. Otherwise, $\sigma \models \phi$ and therefore $[\![e]\!]_\sigma \subseteq [\![e']\!]_\sigma$. Let $t \in [\![e]\!]_\sigma$ (and hence $t \in [\![e']\!]_\sigma$). We want to prove that $t\!\downarrow \in [\![e']\!]_{\sigma'}$. By abstraction, e' is either an intersection of variables or an expression $f(X_1,\ldots,X_n)$. In the first case, $t \in [\![e']\!]_\sigma$ implies, by definition of σ', $t\!\downarrow \in [\![e']\!]_{\sigma'}$. In the second case, $f \notin \mathcal{OF}$, by definition of one-way function symbols. Then $t \in [\![e']\!]_\sigma$ implies that $t = f(t_1,\ldots,t_n)$ and $t\!\downarrow = f(t_1\!\downarrow,\ldots,t_n\!\downarrow)$. Then $t_i \in [\![X_i]\!]_\sigma$ implies, by definition of σ', that $t_i\!\downarrow \in [\![X_i]\!]_{\sigma'}$, hence $t \in [\![e']\!]_{\sigma'}$.

Conversely, assume that σ' is the *minimal* solution of $\sigma^0(S_{NB})$. We extend σ' with σ^0 to basic variables. Let us prove that $\sigma'_{\approx}$ is a solution of S. We first need to establish some properties on σ':

Lemma 41 *If $u \in [\![X]\!]_{\sigma'}$, then $u = u\!\downarrow$.*

Proof: For every term $t \in T(F)$, let $t\!\downarrow$ be its representative for $\approx$. $t\!\downarrow = C[t_1\!\downarrow, \ldots, t_n\!\downarrow]$ for some context C, such that $t = C[t_1,\ldots,t_n]$ and for every term $u \approx t$, $u = C[u_1,\ldots,u_n]$ with $t_i \approx u_i$. The maximal context C verifying the property above is called the *canonical context* of t. Note that since the $t_i\!\downarrow$ are

representatives of the minimal solution of S_B, then the function symbols of the $t_i\downarrow$ do not occur in S_{NB}. In addition, the t_i are equivalent (modulo $\approx$) to the $t_i\downarrow$, thus we have also that the function symbols of the t_i do not occur in S_{NB}. Let us first prove by induction on e that:

> for every σ, if for every term t and every set variable X, $t \in [\![X]\!]_\sigma$ implies $t = t\downarrow$, then for every expression e occurring in $\sigma^0(S_{NB})$, $t \in [\![e]\!]_\sigma$ implies $t = t\downarrow$.

Indeed, assume that for every term t and every set variable X, $t \in [\![X]\!]_\sigma$ implies $t = t\downarrow$ and consider e occurring in $\sigma^0(S_{NB})$ and $u \in [\![e]\!]_\sigma$, $u = C[u_1, \ldots, u_n]$ where C is the canonical context of u. For every i, let us split up both e and C:

- either there exists p_i and $C_i \leq C$ such that $C_i[u_i] \in [\![e|_{p_i}]\!]_\sigma$ and $e|_{p_i} = e'' \cap Y$. In particular, $C_i[u_i] \in [\![Y]\!]_\sigma$, thus by hypothesis, $C_i[u_i] = C_i[u_i]\downarrow$. By construction of the context, $C_i[u_i]\downarrow = C_i[u_i\downarrow]$ thus $u_i = u_i\downarrow$.
- or there exists p_i such that $u_i \in [\![e|_{p_i}]\!]_\sigma$. Since $e = \sigma^0(e_1)$ for some e_1 occurring in S_{NB}, we have to consider again two cases:
  - either p_i is not a path in e_1, i.e., there exists $q_i < p_i$ such that $e_1|_{q_i} = X$ where X is a basic variable and $e|_{q_i} \in \sigma^0(X)$. Thus $u|_{q_i} \in \sigma^0(X) \subseteq R$ and $u|_{q_i} = C_i[u_i]$. Since $u|_{q_i} \in R$, $C_i[u_i] = C_i[u_i]\downarrow = C_i[u_i\downarrow]$, thus $u_i = u_i\downarrow$.
  - either p_i is a path in e_1: Since the function symbols of u_i does not occur in S_{NB} and $u_i \in [\![e|_{p_i}]\!]_\sigma$, $e_1|_{p_i}$ is necessarily an intersection of variables: $e_1|_{p_i} = X_1 \cap \ldots \cap X_n$. If one of the variable, say X_1, is basic then $\sigma^0(e_1) \in \sigma^0(X)$ and we conclude like above. Else $u_i \in [\![X_1 \cap \ldots \cap X_n]\!]_\sigma$ and we conclude by hypothesis.

We are now ready to prove Lemma 41 by induction on the fixed point of our deduction system: assume that for every $n' < n$, then $u \in [\![X]\!]_{T^{n'}(\sigma_\emptyset)}$ implies $u = u\downarrow$. and let us show that $u \in [\![X]\!]_{T^n(\sigma_\emptyset)}$ implies $u = u\downarrow$. Let $\phi \Rightarrow e \subseteq e' \in \sigma^0(S_{NB})$ be the clause which generated u (we assume w.l.o.g. that e' does not contain intersection symbols like in section 5.1.2):

if $e' = X$ then $u \in [\![e]\!]_{T^{n-1}(\sigma_\emptyset)}$ and we conclude by induction.

if $e' = f(X_1, \ldots, X_n)$ **and** $X = X_i$ then there exists $v \in [\![e]\!]_{T^{n-1}(\sigma_\emptyset)}$ such that $v = f(\ldots, u, \ldots)$. Since f does not occurs in $\mathcal{R}(X)$ for X basic variable, v can not be accepted in any state X where is a basic variable, thus $v\downarrow = f(\ldots, u\downarrow, \ldots)$. By induction and the property we have just demonstrated, we know that $v = v\downarrow$ which implies $u = u\downarrow$.

□

We are now ready to prove lemma 40. $\sigma'_{\approx}$ is a solution of S_B, by definition of $\approx$ and σ'. Let us consider a clause $\phi \Rightarrow e \subseteq e'$, which does not belong to S_B. Since ϕ only contains atomic formulas of the form $\mathsf{nonempty}(X)$, $\sigma' \models \phi$ iff $\sigma'_{\approx} \models \phi$. Assume that $\sigma' \models \phi$ (if it is not the case, then $\sigma'_{\approx}$ trivially satisfies the clause).

By induction on e, any term $t \in [\![e]\!]_{\sigma'_{\approx}}$ is equivalent, modulo $\approx$ to a term $u \in [\![e]\!]_{\sigma'}$ such that $u \in [\![e']\!]_{\sigma'}$. Indeed, if e is a variable, this is true by definition of $\approx$. If $e = e_1 \cap e_2$, then, by induction, there exists $u_1 \in [\![e_1]\!]_{\sigma'}$ and $u_2 \in [\![e_2]\!]_{\sigma'}$ such that $t \approx u_1$ and $t \approx u_2$. By lemma 41, $u_1 = u_1{\downarrow}$, $u_2 = u_2{\downarrow}$, thus $u_1 = u_2$ and $u_1 \in [\![e]\!]_{\sigma'}$. If $e = f(e_1, \ldots, e_n)$ (last case), then $t = f(t_1, \ldots, t_n)$, $t_i \in [\![e_i]\!]_{\sigma'_{\approx}}$. By induction, there exists $u_i \approx t_i$ such that $u_i \in [\![e_i]\!]_{\sigma'}$, thus $u \stackrel{\text{def}}{=} f(u_1, \ldots, u_n) \in [\![e]\!]_{\sigma'}$ and $u \approx t$.

Now, either e' is the intersection of the variables X_i, in which case there exists $u_i \in [\![X_i]\!]_{\sigma'}$ for every i such that $u_i \approx t$, hence $t \in [\![X_i]\!]_{\sigma'_{\approx}}$ for every i, or else $e' = f(X_1, \ldots, X_n)$. In the latter case, $f \notin \mathcal{OF}$ and therefore $t = f(t_1, \ldots, t_n), u = f(u_1, \ldots, u_n)$ with $t_i \approx u_i$ and $u_i \in [\![X_i]\!]_{\sigma'}$. Again, this implies that $t_i \in [\![X_i]\!]_{\sigma'_{\approx}}$ for every i, hence $t \in [\![e']\!]_{\sigma'_{\approx}}$.

□

Thanks to lemma 40, and as far as satisfiability is concerned, we can now restrict our attention to the constraint $\sigma_0(S_{NB})$ in which there is no longer any basic variable.

From the cryptographic protocols point of view, if we assume that the set of principal names correspond in the set constraint formalism to a basic variable N (which is the case in all formalism we know), lemma 40 shows that, if there is an attack, then there is an attack with a bounded number of principals. The bound is given by the cardinal of $\sigma_0(N)$. Again, in any description of principals that we can think of, $\sigma_0(N)$ will contain at most two elements. Then, the result shows that, if there is an attack, then there is an attack involving two distinct principals only (a honest one and a dishonest one).

5.2.4 Complexity issues in eliminating the basic variables

Before going any further, let us comment on the complexity of $\sigma_0(S_{NB})$ with respect to S.

First consider the computation of σ_m. Following theorem 13, the computation of $\mathcal{A}_m$ requires deterministic exponential time in general, since it is quite easy to encode the emptiness problem for the intersection of tree automata [27].

On the other hand, we want to point out a particular case which can be relevant to the application to cryptographic protocols. S_B often satisfies additional properties, which we describe below.

For every basic variable X, let $Head(X)$ be the least set of (one-way) function symbols such that

- if $f(...) \subseteq X$ is an inclusion of $\mathcal{R}(X)$, then $f \in Head(X)$
- if $X_1 \cap \ldots \cap X_n \subseteq X \in \mathcal{R}(X)$, then $Head(X_1) \cap \ldots \cap Head(X_n) \subseteq Head(X)$

Lemma 42 *If for every two basic variables X, Y, either $X \subseteq Y \in S_B$ or $Y \subseteq X \in S_B$ or $Head(X) \cap Head(Y) = \emptyset$, then it is possible to compute in polynomial time a finite tree automaton whose states contain the basic variables and which accepts $\sigma_m(X)$ in state X.*

Proof: We can compute $Head(X)$ in polynomial time. Then, while saturating S_B, we replace every intersection with either $\emptyset$ or the largest variable, preventing the combinatorial explosion. □

A second source of complexity comes from the computation of an ET-constraint out of $\sigma_0(S_{NB})$: eliminating the disjunctions may lead to an exponential blow-up in general. However, with the same hypothesis as above, the cardinal of $\sigma_0(X)$ is smaller or equal to the number of inclusions of the form $Y \subseteq X$. In particular, in our running example, only $\sigma_0(A)$ contains more than one element: $\sigma_0(A) = \{ha(0), da(0)\}$. In addition, if we assume that there is no inclusion between basic variables as it was the case in our previous version [11], then $\sigma_0(X)$ assigns each basic variable either the empty set of a singleton set and therefore $\sigma_0(S_{NB})$ is smaller in size than S itself.

5.2.5 Simplifying again the expressions

The goal of this section is to achieve further simplifications. In particular we show that, after eliminating the basic variables, we can get rid of nested constructions $f^c(\ldots)$.

Thanks to lemma 40, we can now restrict our attention to the constraint $\sigma_0(S_{NB})$. In such a constraint, there is no longer any basic variable, which allows for several simplifications. First, we can abstract one the left side of inclusions such that the inclusions are $e \subseteq e'$ where e is either an intersection of variables or an expression $f^c(\vec{e})$ in which, at any position which is not a strict prefix of a position checked by c, there is a variable.

Formally, we use the following rules, assuming that $n \geq 2$ and p is not the root position: and that p is not a strict prefix of any path checked (higher) in C:

$$\begin{array}{lrcl} (A_4) & \phi \Rightarrow C[f^c(\vec{e})]_p \subseteq e' & \rightarrow & (\exists X)\phi \Rightarrow C[X]_p \subseteq e', f^c(\vec{e}) \subseteq X \\ (A_5) & \phi \Rightarrow C[e_1 \cap \ldots \cap e_n]_p \subseteq e' & \rightarrow & (\exists X)\phi \Rightarrow C[X]_p \subseteq e', e_1 \cap \ldots \cap e_n \subseteq X \end{array}$$

Simplification

(N_8)	$t \cap e_1$	$\rightarrow$	$\bot$ If t is ground and $\Pi(e_1) \not\subseteq \Pi(t)$
(N_9)	$\phi \Rightarrow e[t \cap e_0]_p \subseteq e'$	$\rightarrow (\exists Y)$	$\phi, \mathsf{nonempty}(Y) \Rightarrow e[t]_p \subseteq e'$, $t \cap e_0 \subseteq Y$ If t is ground and $\Pi(e_0) \subseteq \Pi(t)$
(N_{10})	$f^{p=q \wedge c}(\vec{e})$	$\rightarrow$	$f^c(\vec{e})$ If $f^c(\vec{e})\|_p = f^c(\vec{e})\|_q$ are ground
(N_{11})	$\phi \Rightarrow e[f^{c \wedge p=q}(\vec{e})]_p \subseteq e'$	$\rightarrow$	$true$ If $t \in f^{c \wedge p=q}(\vec{e})\|_{p \cdot p_1}$ is ground, $u \in f^{c \wedge p=q}(\vec{e})\|_{q \cdot p_1}$ is ground for some p_1 and $t \neq u$

Figure 6: Simplification rules

In these rules, X is a new variable: we assume that there is no capture. The following lemma is a consequence of the definitions:

Lemma 43 *Applying abstraction does terminate on any constraint S, resulting in a constraint S' such that the solutions of S are the restrictions of solutions of S' to the free variables of S. Moreover, if S is an ET-constraint, then so is S' and if every expression is normal in S, then every expression is normal in S'.*

In addition, in the equality tests, if $f^{c \wedge p=q}(\vec{e})$ is an expression such that the subexpression at position p (or q) is basic then the expressions at positions p, q must contain the same ground term. This is also sufficient: the equality test $p = q$ can then be removed if the appropriate inclusions $t \subseteq X$ (t is ground) are added. Formally, we use the rules displayed in figure 6.

Lemma 44 *The simplification rules displayed in figure 6 are terminating. If S is an ET-constraint in which all expressions are normal, then the result S' of simplifying and abstracting $\sigma_0(S_{NB})$ is an ET-constraint in which all expressions are normal and which is satisfiable iff S is satisfiable. Moreover, in any expression $f^c(e_1, \ldots, e_n)$ occurring in S', $e_1, \ldots, e_n$ do not contain a construction $g^{c'}(\ldots)$.*

Proof: The correctness of the rules is a routine verification. let us only consider the rule (N_9). If $\sigma \not\models \phi$, it suffices to assign $T(\mathcal{F})$ to Y and both sides are satisfied by σ. If $\sigma \models \phi$ and $[\![e[t \cap e_0]]\!]_\sigma \subseteq [\![e']\!]_\sigma$, then extending σ with $Y \mapsto [\![t \cap e_0]\!]_\sigma$ we get a solution of the right hand side:

- either $[\![t \cap e_0]\!]_\sigma = \emptyset$ and this is straightforward
- or else $[\![t \cap e_0]\!]_\sigma = \{t\}$, since t is a ground term, in which case $[\![e[t \cap e_0]]\!]_\sigma = [\![e[t]]\!]_\sigma$

Conversely, if σ is a solution of the right hand side, either $[\![Y]\!]_\sigma$ is empty, which means that $t \notin [\![e_0]\!]_\sigma$ and the left hand side is satisfied by σ or $[\![Y]\!]_\sigma = \{t\}$, in which case $[\![e[t \cap e_0]]\!]_\sigma = [\![e[t]]\!]_\sigma \subseteq [\![e']\!]_\sigma$.

Thanks to lemma 40, it only remains to show that all expressions are normal in S' whenever all expressions are normal in S and that, moreover, there is no longer any nested equality test.

For every expression $f^c(\vec{e})$, $P(c) \subseteq \Pi(f^c(\vec{e}))$. Only the case of (N_9) is not trivial. The property is ensured by the side condition.

Condition 5 is satisfied σ^0 may replace variables with ground terms, hence replace expressions $g^c(\vec{e}) \cap X_1 \cap \ldots X_n$ with $g^c(\vec{e}) \cap t_1 \cap \ldots \cap t_n$. However, each t_i is either a variable or is ground. If at least one of them is ground, we can apply either (N_8) or (N_9).

Condition 3 is satisfied There are two situations in which property 3 is not trivially preserved: first when, while removing disjunctions in $\sigma^0(S_{NB})$, we do not keep the consistency with equality tests: in an expression $f^{p=q \wedge c}(\vec{e})$, X has been replaced with t at a position $p \cdot p_1$, while X has been replaced with u at the position $q \cdot p_1$. This case is handled by rule (N_{11}).

The second situation in which condition 3 may not be preserved is when we apply the rule (N_9). However, in this case, applying the rule to all identical expressions restores condition 3.

Condition 1 is satisfied Thanks to (N_{10}) and (N_{11}), we cannot have $p \sim_c q$, $e|_p$ ground and $e|_q$ not ground. So, a repeated application of (N_{10}) consists in removing an equivalence class, which preserves condition 1, thanks to lemma 77. The rules other than (N_{10}) trivially preserve condition 1.

Conditions 4, 2 and 6 are satisfied Again, the only rule to be considered is (N_{10}) since this is the only rule in the set which modifies the tests without removing them entirely. As above, since its repeated application removes a class, properties 4, 2 and 6 are preserved.

There is no nested test Assume that there are nested tests: $f^c(\vec{e})|_p = g^{c'}(\vec{e'}) \cap e''$.

First, if there are $p_1 \sim_c p_2$ such that p is a prefix of p_1, by properties 6 and 1, for every $p' \sim_{c'} q'$, $g^{c'}(\vec{e'})|_{p'}$ must be a basic expression, hence a ground term. Then, the rules $(N_8), (N_9), (N_{10}), (N_{11})$ ensure that c' is empty.

On the other hand, if this is not the case and if there are $p_1 \sim_c p_2$ such that p_1 shares a non-trivial prefix with p, then, by property 1, $f^c(\vec{e})|_p$ must be a basic expression, hence a ground term. In this last case c' must be empty again.

Remains only the case in which, for every $p_1 \sim_c p_2$, p_1 is either incomparable with p or a prefix of p. Then, **Abstract** can be applied, which contradicts the hypothesis on S_{NB}.

□

We use a final simplification rule, abstracting away some more expressions:

$$(N_{12}) \quad \phi \Rightarrow e[X \cap g(e'_1, \ldots, e'_m)]_p \subseteq e' \rightarrow \begin{array}{l} (\exists Y_1, \ldots, Y_m) \\ X \cap g(\top, \ldots, \top) \subseteq g(Y_1, \ldots, Y_m) \\ \phi \Rightarrow e[g(Y_1 \cap e'_1, \ldots, Y_m \cap e'_m)]_p \subseteq e' \end{array}$$

If p is non-empty.

The rule assumes that there is a variable $\top$ which captures all terms (this is easy to define).

The correctness of the rule as well as the preservation of all properties is quite straightforward. Let us now inspect the constraints we have still to consider.

Definition 45 *The* SET-constraints *(Simplified Equality Tests constraints) are a subclass of ET-constraints in which, for every clause*

$$\mathsf{nonempty}(e'_1), \ldots, \mathsf{nonempty}(e'_m) \Rightarrow e \subseteq e'$$

(resp. $\mathsf{nonempty}(e'_1), \ldots, \mathsf{nonempty}(e'_m) \Rightarrow \mathbf{false}$*)*

1. *each of* $e, e'_1, \ldots, e'_m$ *is either an intersection of variables or an expression* $f^c(e_1, \ldots, e_n)$ *such that* $e_1, \ldots, e_n$ *do not contain any equality tests nor expressions* $X \cap g(\ldots)$.

2. *For every* $p \in \Pi(e)$*, except the root, either* p *is a strict prefix of some* $q \in P(c)$*, or else* $e|_p$ *is a (basic) ground term or* $p \in P(c)$ *and* $e|_p$ *is an intersection of variables.*

3. *If* $p \sim_c q$, $e|_p = e|_q$.

4. e' *is either a variable or an expression* $f'(X_1, \ldots, X_n)$ *where* $X_1, \ldots, X_n$ *are variables.*

Lemma 46 *The simplification rules displayed in figure 6 are terminating. If* S *is an SET-constraint in which all expressions are normal, then the result* S' *of simplifying and abstracting* $\sigma_0(S_{NB})$ *is an ET-constraint*

Proof: Let us show that S' verifies the four conditions of SET-constraints.

1. Assume e or one of the e'_i is of the form $f^c(e_1, \ldots, e_n)$ and that one of the e_j contains an equality test c_j. Then, by abstracting, it must be that c overlaps c_j which is not possible since S' is normalized.

2. After abstraction, if p is not a strict prefix of some $q \in P(c)$ and p is in $\Pi(e)$, then $p \in P(c)$ and $e|_p$ is a variable. After simplifying, if p is not a strict prefix of some $q \in P(c)$, then $e|_p$ is either a variable or a ground term.

Conditions 3 and 4 are consequences of the definitions. □

5.2.6 Deduction rules

Now, we are ready to apply the deduction rules given in figure 7. $c\downarrow_i$ 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 $c = \top$). Finally, X denotes a variable in these rules.

Lemma 47 *The inference rules in figure 7 are correct: the new constraint is a consequence of the previous ones.*

Proof: Only **Projection** and **Deduction** are not trivially correct. Let us start with **Projection**.

We want to prove that, if σ is a solution of $\phi \Rightarrow f^c(e_1, \ldots, e_n) \subseteq f(e'_1, \ldots, e'_n)$, then σ is a solution of $\phi, \mathsf{nonempty}(f^c(e_1, \ldots, e_n)) \Rightarrow e_i^{c\downarrow i} \subseteq e'_i$.

Assume $\sigma \models \phi, \mathsf{nonempty}(f^c(e_1, \ldots, e_n))$. Then there is an $u \in [\![f^c(e_1, \ldots, e_n)]\!]_\sigma$ and $[\![f^c(e_1, \ldots, e_n)]\!]_\sigma \subseteq [\![f(e'_1, \ldots, e'_n)]\!]_\sigma$.

Let $t \in [\![e_i^{c\downarrow i}]\!]_\sigma$. We build v as follows: v is the term u in which

- $u|_i$ is replaced with t

Transitivity

$$\frac{\phi_1 \Rightarrow e_1 \subseteq X \qquad \phi_2 \Rightarrow X \subseteq e_2}{\phi_1, \phi_2 \Rightarrow e_1 \subseteq e_2}$$

Compatibility

$$\frac{\phi \Rightarrow X \cap e_1 \subseteq e_1' \qquad \phi' \Rightarrow e_2 \subseteq X}{\phi, \phi' \Rightarrow e_1 \cap e_2 \subseteq e_1'}$$

If both e_1 and e_2 are intersections of variables

Clash

$$\frac{\phi \Rightarrow f(\vec{e}) \subseteq g(\vec{e'})}{\phi \Rightarrow \textbf{false}} \qquad \text{if } f \neq g$$

Projection

$$\frac{\phi \Rightarrow f^c(e_1, \ldots, e_n) \subseteq f(e_1', \ldots, e_n')}{\phi, \mathsf{nonempty}(f^c(e_1, \ldots, e_n))) \Rightarrow e_i^{c\downarrow_i} \subseteq e_i'}$$

Deduction

$$\frac{\begin{array}{l}\phi_1 \Rightarrow f^{c_1}(e_1^1, \ldots, e_n^1) \subseteq X_1 \\ \vdots \\ \phi_k \Rightarrow f^{c_k}(e_1^k, \ldots, e_n^k) \subseteq X_k \\ \phi \Rightarrow X_1 \cap \ldots \cap X_k \subseteq e\end{array}}{\begin{array}{l}\exists X_1^1, \ldots X_1^n, \ldots, X_k^1, \ldots, X_k^n. \\ \phi \Rightarrow f(X_1^1 \cap \ldots \cap X_k^1, \ldots, X_1^n \cap \ldots \cap X_k^n) \subseteq e \\ \phi_1 \Rightarrow f^{c_1}(e_1^1, \ldots, e_n^1) \subseteq f(X_1^1, \ldots, X_n^1) \\ \vdots \\ \phi_k \Rightarrow f^{c_k}(e_1^k, \ldots, e_n^k) \subseteq f(X_1^k, \ldots, X_n^k)\end{array}}$$

The clause $\phi \Rightarrow f(X_1^1 \cap \ldots \cap X_k^1, \ldots, X_1^n \cap \ldots \cap X_k^n) \subseteq e$ in the conclusion of **Deduction** is marked so that it cannot be used as a premisse of **Deduction**. In addition, if $\phi_1 \Rightarrow e_1 \subseteq X$ is a marked clause, then for every clause $\phi_2 \Rightarrow X \subseteq e_2$, then clause $\phi_1, \phi_2 \Rightarrow e_1 \subseteq e_2$ is also a marked clause.

Figure 7: The saturation rules

- for every $i \cdot p \sim_c j \cdot q$ with $i \neq j$, $u|_{j \cdot q}$ is replaced with $t|_p$.

Let us show that $v \in [\![f^c(e_1, \ldots, e_n)]\!]_\sigma$.

- First, $v \in [\![f(e_1, \ldots, e_n)]\!]_\sigma$: let $v = f(v_1, \ldots, v_n)$. $v_i = t \in [\![e_i]\!]_\sigma$ and we prove by induction on $|c|$ that, for every $i \neq j$, $v_j \in [\![e_j]\!]$.

  If c is empty, or if c does not contain any equation $i \cdot p = j \cdot q$, then $v_j = u_j$ and therefore $v_j \in [\![e_j]\!]_\sigma$.

  If $i \cdot p \sim_c j \cdot q$, by induction hypothesis, $w_j = v_j[u|_{j \cdot q}]_q \in [\![e_j]\!]_\sigma$. Moreover, by property 3, $e_j|_q = e_i|_p$, hence $t|_p \in [\![e_j|_q]\!]_\sigma$. Now, since we assumed in the condition of the projection rule, that there is no intersection symbol along the path $j \cdot q$, $v_j \in [\![e_j]\!]_\sigma$.

- We have to prove now that $v \models c$:
  - For the tests $i \cdot p = i \cdot q \in c$, $v \models i \cdot p = i \cdot q$ follows from $t \models p = q$ (since $t \models c \downarrow i$).
  - The tests $i \cdot p = j \cdot q$ with $i \neq j$ are satisfied by construction.
  - For the tests $j_1 \cdot p = j_2 \cdot q$, either there is a r such that $j_1 \cdot p \sim_c i \cdot r$ and we are back to the previous case, or else $u|_{j_1 \cdot p} = v|_{j_1 \cdot p}$ and $u|_{j_2 \cdot q} = v|_{j_2 \cdot q}$, which implies $v \models j_1 \cdot p = j_2 \cdot q$ since $u \models j_1 \cdot p = j_2 \cdot q$.

Now, $v \in [\![f^c(e_1, \ldots, e_n)]\!]_\sigma$ implies that $v \in [\![f(e'_1, \ldots, e'_n)]\!]_\sigma$, hence $t \in [\![e'_i]\!]_\sigma$.

Now consider **Deduction**. The rule is actually a combination of several rules which are all correct: for every i, we introduce

$$X_i \cap f(\top, \ldots, \top) = f(X_1^i, \ldots, X_n^i) \tag{1}$$

Then, we may intersect both sides of $X_1 \cap \ldots \cap X_k \subseteq e$ with $f(\top, \ldots, \top)$ and use a compatibility. We get:

$$\phi \Rightarrow f(X_1^1, \ldots, X_n^1) \cap \ldots \cap f(X_1^k, \ldots, X_n^k) \subseteq e$$

Normalizing the left hand side, we get the clause:

$$\phi \Rightarrow f(X_1^1 \cap \ldots \cap X_k^1, , \ldots, X_1^n \cap \ldots \cap X_k^n) \subseteq e$$

Now, for every i, from (1) and the inclusion $\phi_i \Rightarrow f^{c_i}(e_1^i, \ldots, e_n^i) \subseteq X_i$, intersecting again both sides with $f(\top, \ldots, \top)$, we deduce by transitivity and since $f^c(\ldots) \subseteq f(\top, \ldots \top)$:

$$\phi_i \Rightarrow f^{c_i}(e_1^i, \ldots, e_n^i) \subseteq f(X_1^i, \ldots, X_n^i)$$

□

Lemma 48 *Every transformation rule transforms a SET-constraint into a SET-constraint.*

Proof: Only the projection rule has to be considered: we need to check that $e_i^{c\downarrow_i}$ satisfies the conditions of SET-constraints, which follows from the fact that $p \sim_{c\downarrow_i} q$ iff $p \cdot i \sim_c q \cdot i$. □

Now, we consider the termination problem. The main problem is to control the creation of new variables.

Lemma 49 *The number of variables created during the saturation procedure can be bounded by $|S|_t \times a$.*

Proof: Only **Deduction** introduces new variables. It is simpler to see the **Deduction** rule as a variable introduction (rule (1) in the proof of lemma 47) combined with other deduction rules, which preserve the semantics and do not introduce variables. As far as variables creation is concerned, we can, w.l.o.g, assume that the conclusion of **Deduction** are the clauses $X_i \cap f(\top, \ldots, \top) \subseteq f(X_1^i, \ldots, X_n^i)$. In particular, if the rule is applied twice with the same variable X_i and the same function symbol f, we can use the same variables X_j^i.

The main problem is that these new variables X_j^i may trigger again the introduction of new variables. However, if we trace the origin of such variables, we observe that introducing the X_j^i is subject to the presence of a constraint $\phi_i \Rightarrow f^{c_i}(\ldots) \subseteq X_i$. Now, if later a constraint $\phi_i' \Rightarrow f^{c_i'}(\ldots) \subseteq X_j^i$ triggers the introduction of new variables again, the expression $f^{c_i'}(\ldots)$ must be the projection of some expression occurring at the previous level. And since we can only perform a bounded number of projections on a given expression, we can bound the number of successive variables generation. Let us formalize this.

Let us associate first with each variable a *level*: the variables occurring in the original constraint have level 0, and, each time we introduce new variables with:

$$(1) \quad X \cap f(\top, \ldots, \top) \subseteq f(X_1, \ldots, X_n)$$

the level of every variable X_i is one plus the level of X.

We prove, by induction on the level m of a variable X, that, if (1) is applied to X, then there is an expression e in the original set constraint and clauses $\phi \Rightarrow e' \subseteq X'$, $\phi_1 \Rightarrow X' \subseteq X_1$, $\ldots$, $\phi_{n+1} \Rightarrow X_n \subseteq X$ such that e' is obtained by at least m successive projections of e (we say that e' is a projection of e if $e = f^c(e_1, \ldots, e_n)$ and $e' = e_i^{c\downarrow_i}$ for some i).

When $m = 0$, observe that clauses $\phi \Rightarrow f^c(\ldots) \subseteq X'$ are either in the original set constraint, or obtained by projection, or obtained by **Transitivity** or obtained by **Deduction** itself. In case **Transitivity** has been applied, there

exist an other clause $\phi' \Rightarrow f^c(\ldots) \subseteq X''$ and a clause $\phi'' \Rightarrow X'' \subseteq X'$. Thus, by a simple induction, there exist a clause $\phi' \Rightarrow f^c(\ldots) \subseteq X''$ and clauses $\phi_1 \Rightarrow X'' \subseteq X_1$, $\ldots$, $\phi_{n+1} \Rightarrow X_n \subseteq X'$ such that $\phi' \Rightarrow f^c(\ldots) \subseteq X''$ has been obtained by projection or by **Deduction** itself. However, in the latter case, we explicitly prevented using the resulting clause as a premisse of **Deduction**.

When $m > 0$, observe that the variables X_j^i created by **Deduction** appear in only one clause on the right of an inclusion: the clause $\phi_i \Rightarrow f^{c_i}(e_1^i, \ldots, e_n^i) \subseteq f(X_1^i, \ldots, X_n^i)$. Only **Projection**, applied to this clause, may produce a clause in which X_j^i occurs on the right of an inclusion: there exists a clause $\phi_i \Rightarrow e_j^{i\,c_i\downarrow j} \subseteq X_j^i$ such that $e_j^{i\,c_i\downarrow j}$ is the projection of $f^{c_i}(e_1^i, \ldots, e_n^i)$. Then we apply the induction hypothesis to $\phi_i \Rightarrow f^{c_i}(e_1^i, \ldots, e_n^i) \subseteq X_i$ since X_i is of level $m-1$.

Now, how many expressions $f^c(\ldots)$ can be derived by projection from a given expression ? Note first that no projection can be applied to a ground (basic) expression since one-way symbols do not occur on the right of inclusions. Then, by definition of expressions occurring in a SET-constraint, the number of expressions $f^c(\ldots)$ which can be derived by projection from an expression $g^{c'}(\ldots)$ is the number of strict prefixes of positions in $P(c')$. It follows that the number of new variables is bounded by $|S|_t \times a$.

□

Lemma 50 *The rules of figure 7 are terminating: a fixed point is reached after finitely many steps (at most $O(|S|^a \times 2^{(a+1)|S|})$ where a is the maximal arity of a function symbol and $|S| = |S|_t + |S|_{\mathcal{F}}$ is the size of the original constraint.*

Proof: We are going to show that only a finite number of distinct clauses can be generated by the rules.

As we have seen in the proof of lemma 49, the number of distinct expressions $f^c(\ldots)$ occurring on the left of an inclusion is bounded by $|S|_t$, plus the number of times **Deduction** is applied, which is itself bounded by $|S|_t \times a$ thanks to lemma 49. The other left sides of inclusions are intersection of variables, hence there are at most $2^{a|S|_t + |V_S|}$ such possible left hand sides, thanks to lemma 49.

The right sides of inclusions are variables or a function symbol applied to variables, which gives a bound of $a|S|_t + |V_S| + |\mathcal{F}| \times (a|S|_t + |V_S|)^a$.

Finally, we have to analyze the possible preconditions. They are conjunctions of

- $\mathsf{nonempty}(X)$ where $X \in V_S$
- $\mathsf{nonempty}(f^c(e_1, \ldots, e_n))$ where $f^c(e_1, \ldots, e_n)$ occurs as a left member of an inclusion constraint

Thanks to lemma 49, this gives the following bound for the number of possible distinct preconditions:

$$2^{|V_S|+|S|_t}$$

Now, putting everything together, at most

$$((a+1)\times|S|_t + 2^{a|S|_t+|V_S|}) \times (a|S|_t + |V_S| + |\mathcal{F}| \times (a|S|_t + |V_S|)^a) \times 2^{|V_S|++|S|_t}$$

distinct clauses can be generated. Since $|S|_t+|V_S| \leq |S|$ and $a|S|_t+|V_S| \leq a\times|S|$, we get the bound $O(|S|^a \times 2^{(a+1)|S|})$.

□

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 $f^c(\vec{e}) \subseteq X$ where X is a variable.

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. $f^c(X \cap g(\vec{e}), \vec{e'}) \subseteq f(\vec{e''})$.

Theorem 51 *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).*

Proof: If solved(S) is unsatisfiable, then S, which contains solved(S), is unsatisfiable. Now, assume solved(S) is satisfiable and let α be its least solution. We show that α is a solution of S.

We prove, by induction on $n + size(t)$ that, for every clause $\phi \Rightarrow e \subseteq e'$ in S such that $\alpha \models \phi$, and for every $t \in [\![e]\!]_{T^n_{\text{solved}(S)}(\emptyset)}$ (which we abbreviate $t \in [\![e]\!]_n$), $t \in [\![e']\!]_\alpha$.

The result will follow, by minimality of α.

There are only three kinds of clauses $\phi \Rightarrow e \subseteq e'$, which are possibly in S and not in solved(S). We study each of them (e' is either a variable or an expression $f(X_1, \ldots, X_n)$).

$\phi \Rightarrow X \subseteq g(Y_1, \ldots, Y_m)$, $t \in [\![X]\!]_n$, hence there is a clause $\phi' \Rightarrow e_0 \subseteq X$ in solved(S) such that $\alpha \models \phi'$ and $t \in [\![e_0]\!]_{n-1}$. By **Transitivity**, there is a clause $\phi, \phi' \Rightarrow e_0 \subseteq g(Y_1, \ldots, Y_m)$ in S. Since $\alpha \models \phi, \phi'$ and $t \in [\![e_0]\!]_{n-1}$, we apply the induction hypothesis and get $t \in [\![g(Y_1, \ldots, Y_m)]\!]_\alpha$.

$\phi \Rightarrow X_1 \cap \ldots \cap X_p \subseteq e$ We use an induction on the multiset $M(X_1 \cap \ldots \cap X_p) \stackrel{\text{def}}{=} \{k_1, \ldots, k_p\}$ of integers k_i such that $t \in [\![X_i]\!]_{k_i}$ and $t \notin [\![X_i]\!]_{k_i-1}$. The maximum of $k_1, \ldots, k_p$ is, by hypothesis, smaller or equal to n. If it is strictly smaller than n, we use directly the induction hypothesis. If this multiset is equal to $\{n\}$, then we are back to the first case. Hence, let us

assume now that the multiset is strictly larger than $\{n\}$, which means in particular that $p \geq 2$.

Since $t \in [\![X_i]\!]_{k_i}$ for every i, there are clauses $\phi_i \Rightarrow e_0^i \subseteq X_i$ in solved(S) such that $\alpha \models \phi_i$ and $t \in [\![e_0^i]\!]_{k_i-1}$. If one of the expressions e_0^i is an intersection of variables, then by **Compatibility**, there is a clause $\phi_i, \phi' \Rightarrow X_1 \cap \ldots \cap X_{i-1} \cap e_0^i \cap X_{i+1} \cap \ldots \cap X_p \subseteq e$ in S. Moreover, $\alpha \models \phi'$ and $t \in [\![X_1 \cap \ldots \cap X_{i-1} \cap e_0^i \cap X_{i+1} \cap \ldots \cap X_p]\!]_n$. Finally, $M(X_1 \cap \ldots \cap X_{i-1} \cap e_0^i \cap X_{i+1} \cap \ldots \cap X_p)$ is obtained replacing k_i in $M(X_1 \cap \ldots, \cap X_p)$ with a multiset of strictly smaller numbers. Hence we get a strictly smaller multiset and we may apply the induction hypothesis: $t \in [\![e]\!]_\alpha$.

Now, if none of the expressions e_0^i is an intersection of variables: $t = f(t_1, \ldots, t_n)$ and $e_0^i = f^{c_i}(e_1^i, \ldots, e_n^i)$. If one of the clauses

$$\phi_i \Rightarrow f(e_1^i, \ldots, e_n^i) \subseteq X_i$$

is marked, then there exist a clause $\phi_i' \Rightarrow f(e_1^i, \ldots, e_n^i) \subseteq X_i'$ marked by **Deduction** and clauses $\psi_1 \Rightarrow X_i' \subseteq X^1$, $\ldots$, $\psi_{n+1} \Rightarrow X^n \subseteq X_i$ such that $\phi_i = \phi_i', \psi_1, \ldots, \psi_{n+1}$. Thus, there is a clause $\phi_i' \Rightarrow Z_1 \cap \ldots \cap Z_m \subseteq X_i'$ which triggered the application of **Deduction**. Then, applying repeatedly **transitivity**, there is a clause $\phi_i \Rightarrow Z_1 \cap \ldots \cap Z_m \subseteq X_i$. In such a case, for every k, $e_k^i = Z_k^1 \cap \ldots \cap Z_k^m$. Now, $t \in [\![e_0^i]\!]_{k_i-1}$, hence, for every k, j, $t_k \in [\![Z_k^j]\!]_{k_i-1}$. It follows that, for every j, $t \in [\![Z_j]\!]_{k_i-1}$. Now, by **Compatibility**, there is a clause $\phi, \phi_i \Rightarrow X_1 \cap \ldots Z_1 \cap \ldots \cap Z_m \ldots \cap X_n \subseteq e$ in S. And, as before, we get an intersection of variables with a strictly smaller multiset. We conclude thanks to the induction hypothesis.

Remains only the case where none of the clauses $\phi_i \Rightarrow f(e_1^i, \ldots, e_n^i) \subseteq X_i$ is marked. In this last case, we can apply **Deduction**. For every clause $\phi_i \Rightarrow f^{c_i}(e_1^i, \ldots, e_n^i) \subseteq f(X_1^i, \ldots, X_n^i)$, $\alpha \models \phi_i$ and $t \in [\![f^{c_i}(e_1^i, \ldots, e_n^i)]\!]_{k_i-1}$, hence $t_j \in [\![X_j^i]\!]_{k_i}$ for every i, j, by induction hypothesis.

If t is a constant, then $\phi \Rightarrow t \subseteq e$ is a clause of S and we can conclude. Otherwise, if e is a variable, then

$$\phi \Rightarrow f(X_1^1 \cap \ldots \cap X_p^1, \ldots, X_1^n \cap \ldots \cap X_p^n) \subseteq X$$

is a clause of solved(S). Hence $t = f(t_1, \ldots, t_n) \in [\![X]\!]_{n+1}$.
If e is not a variable, then e must be $f(Y_1, \ldots, Y_n)$ and, by **Projection**, for every j,

$$\phi, \mathsf{nonempty}(f(X_1^1 \cap \ldots \cap X_p^1, \ldots, X_1^n \cap \ldots \cap X_p^n)) \Rightarrow X_1^j \cap \ldots \cap X_p^j \subseteq Y_j$$

is a clause of S. α satisfies the premisse since t is a witness for the second part of the precondition. Moreover, for every j, $t_j \in [\![X_1^j \cap \ldots \cap X_p^j]\!]_n$

and $M(X_1^j \cap \ldots \cap X_p^j)$ is smaller or equal to $M(X_1 \cap \ldots, X_p)$. Then, by induction hypothesis, $t_j \in [\![Y_j]\!]_\alpha$ for every j. Hence $t \in [\![e]\!]_\alpha$.

$\phi \Rightarrow f^c(e_1, \ldots, e_p) \subseteq g(X_1, \ldots, X_m)$. If $f \neq g$, then, by **Clash**, the clause $\phi \Rightarrow$ **false** is in S, hence in solved(S), which contradicts $\alpha \models \phi$.

Assume now $f = g$. Then $t = f(t_1, \ldots, t_n)$. By **Projection**, there is a clause

$$C_i \stackrel{\text{def}}{=} \phi, \mathsf{nonempty}(f^c(e_1, \ldots, e_p)) \Rightarrow e^{c\downarrow_i} \subseteq X_i$$

in S, for every i. Since $t \in [\![f^c(e_1, \ldots, e_n)]\!]_\alpha$,

$$\alpha \models \phi, \mathsf{nonempty}(f^c(e_1, \ldots, e_n)).$$

For every i, $t_i \in [\![e_i]\!]_n$ since $t \in [\![f(e_1, \ldots, e_p)]\!]_n$ and $t_i \models c \downarrow_i$ since $t \models c$ (and by definition of $c \downarrow_i$). Thus $t_i \in [\![e_i^{c\downarrow i}]\!]_n$ which implies $t_i \in [\![X_i]\!]_\alpha$ by induction., thus $f(t_1, \ldots, t_p) \in [\![f(Y_1, \ldots, Y_p)]\!]_\alpha$.

□

We are now reduced to prove that the satisfiability of *solved*(S) is decidable.

5.3 Connection with automata with one memory

Theorem 52 *For every satisfiable SET-constraint S, there is an (effectively computable) alternating automaton with one memory A_S and an homomorphism H such that A_S accepts t in the state X iff $H(t) \in [\![X]\!]_\alpha$ where α is the least solution of solved(S).*

Proof:

The memory alphabet of the automaton is the set of function symbols used in the constraint and the alphabet $\mathcal{F}_{A_S}$ is the memory alphabet with some additional symbols allowing to check on auxiliary branches non emptiness conditions. More precisely,

- the states of A_S consist of
  - the variables of S. We write them q_X such states, for X a variable of S
  - the states q_ϕ for every $\phi = \mathsf{nonempty}(e_1), \ldots, \mathsf{nonempty}(e_m)$ such that $\mathsf{nonempty}(e_1), \ldots, \mathsf{nonempty}(e_m), \mathsf{nonempty}(e_{m+1}), \ldots, \mathsf{nonempty}(e_n)$ is a precondition of a clause in S.
  - for every expression $e = f^c(e_1, \ldots, e_n)$ and for every non-leaf position p of such an expression, a state $q_{e,p}$

– a state q_a for every constant $a \in \mathcal{F}$

- The memory alphabet is $\mathcal{F}$
- The set of function symbols $\mathcal{F}_{A_S}$ consists in an auxiliary binary symbol E and, for every symbol $f \in \mathcal{F}$ a symbol f^{+1} whose arity is one plus the arity of f.

Let H be the homomorphism:

$$\begin{array}{rcl} H(f(t_1,\ldots,t_n)) & \stackrel{\text{def}}{=} & f(H(t_1),\ldots,H(t_n)) \\ H(f^{+1}(t_0,t_1,\ldots,t_n)) & \stackrel{\text{def}}{=} & f(H(t_1),\ldots,H(t_n)) \\ H(E(t_1,t_2)) & \stackrel{\text{def}}{=} & a \end{array}$$

where a is any constant in $\mathcal{F}$. $\alpha(X)$ will be the image by H of the trees recognized in state q_X by A_S.

By convention, if p is a leaf position of $f^c(e_1,\ldots,e_n)$, we let $q_{e,p}$ be the conjunction $q_{X_1} \wedge \ldots \wedge q_{X_m}$ if $e|_p = X_1 \cap \ldots \cap X_m$ and $q_{e,p} = q_a$ if $e|_p$ is the constant a.

The transition rules of A_S consist of

1. $a \xrightarrow[a]{\top} q_a$.
2. $f(q_{e,1},\ldots,q_{e,n}) \xrightarrow[\lambda x_1,\ldots,x_n.a]{\gamma} q_{f^c(e_1,\ldots,e_n)}$ for every literal $\mathsf{nonempty}(f^c(e_1,\ldots,e_n))$ occurring in S. γ is the equality test $\tilde{c}$ defined as follows: $i \cdot \widetilde{p = j} \cdot q \stackrel{\text{def}}{=} i = j$ and $\widetilde{c_1 \wedge c_2} \stackrel{\text{def}}{=} \widetilde{c_1} \wedge \widetilde{c_2}$.
3. $g(q_{e,p\cdot 1},\ldots,q_{e,p\cdot k}) \xrightarrow[F]{\widetilde{c\downarrow p}} q_{e,p}$ in which $e = f^c(e_1,\ldots,e_n)$ occurs in S, p is a non-root position of e, $e|_p$ is headed with g. $F = \lambda x_1,\ldots,x_k.x_i$ if $p \cdot i$ is a prefix of a position in $P(c)$. Note that if there are several such indices, lemmas 29 and 44 imply that the choice is irrelevant. $c \downarrow p$ is defined as in the projection rule: $c \downarrow \epsilon \stackrel{\text{def}}{=} c$, $c_1 \wedge c_2 \downarrow i \cdot p \stackrel{\text{def}}{=} (c_1 \downarrow i \cdot p) \wedge (c_2 \downarrow i \cdot p)$, $(j \cdot p_1 = k \cdot p_2) \downarrow i \cdot p \stackrel{\text{def}}{=} \top$ if either $i \neq j$ or $k \neq i$ and $(i \cdot p_1 = i \cdot p_2) \downarrow i \cdot p \stackrel{\text{def}}{=} (p_1 = p_2) \downarrow p$.
4. $E(q_e, q_{e'}) \xrightarrow[\lambda x_1,x_2.x_1]{\top} q_{\mathsf{nonempty}(e),\mathsf{nonempty}(e')}$ for every $\mathsf{nonempty}(e), \mathsf{nonempty}(e')$ which is an initial sequence of a precondition of a clause of S.
5. $E(q_\phi, q_e) \xrightarrow[\lambda x_1,x_2.x_1]{\top} q_{\phi,\mathsf{nonempty}(e)}$ for every $\phi, \mathsf{nonempty}(e)$ which is an initial sequence of a precondition of a clause of S.

6. $f^{+1}(q_\phi, q_{e,1}, \ldots, q_{e,n}) \xrightarrow[F]{\tilde{c}} q_X$ for every clause $\phi \Rightarrow e \subseteq X$ in S such that $e = f^c(e_1, \ldots, e_n)$. $F = \lambda x_{i_1}, \ldots, x_{i_k}.t$ where t is the expression e in which, at each $e|_{j \cdot p}$ such that $j \cdot p \in P(c)$ is replaced with x_{i_j} where $i_j = j \in \tilde{c}$.

The following intermediate results are proved in appendix D:

- If a term is accepted in state q_ϕ where ϕ is a precondition, then $\alpha \models \phi$
- If $\alpha \models \phi$, then there is a term accepted in state q_ϕ
- If a term t is accepted in state q_X, then $H(t) \in \alpha(X)$
- If $t \in \alpha(X)$, then there exists t' is accepted in state q_X such that $H(t') = t$.

From these lemmas, it follows that A_S accepts the least solution α of solved(S) in the sense that t is accepted in the state X iff $H(t) \in [\![X]\!]_\alpha$.

□

Remark: We conjecture that the minimal solution of a SET-constraint is recognized by an alternating tree automata with one memory. However, to prove this would require more saturation rules to get rid of non-emptiness conditions (as in section 3).

As a consequence of theorem 52 we get:

Theorem 53 *The satisfiability of ET-constraints is decidable.*

Proof: As a consequence of lemma 47, lemma 50, theorem 51, we can assume that S is a solved form.

Then, consider all clauses $\phi \Rightarrow$ **false** in S and let S' be the rest of S.

S' is satisfiable, then, thanks to theorem 52, there is an automaton A_S such that t is accepted in q_X iff $H(t) \in [\![X]\!]_\alpha$ where α is the minimal solution of S'.

Let Q_f be the set of states q_ϕ such that $\phi \Rightarrow$ **false** is in S. Then S is satisfiable iff, the automaton A_S with final states Q_f does not accept any tree, which is decidable thanks to theorem 17. □

6 Analysis of cryptographic protocols

We present here a decidable fragment of the class of protocols described in Section 2.2 and we illustrate the relevance of this fragment by an example (inspired by Kerberos).

6.1 A decidable class of protocols

As we have seen in section 2, the specification of a protocol and its secrecy policy rely not only on the rules of the protocol, but also on the signature. In particular, we must say what are the expected types of each argument of a function symbol. This is far from being innocent, since this corresponds to the ability of each agent to recognize different data types. If all function symbols are assumed to take messages or agents as arguments and return messages or agents, then the principals are assumed to distinguish only agents from other messages. For instance, a pair of agents can be taken as a key in this case. Since many attacks are due to type confusion [8], such a signature specification would allow to detect many more attacks. On the other hand, if typing information is available, then deciding secrecy is easier.

One specificity of our model is that both the signature, hence the available type information, and the protocol itself are parameters. The defining conditions for our class will therefore depend on both the signature and the protocol and be more restrictive when the typing policy is more sloppy.

A rule of a protocol is of the form

$$\{S(A, i, M), M_1, \ldots, M_p\} \longrightarrow \{S(A, i+1, M'), M'_1, \ldots, M'_q\}$$

where M_i are messages. As we have seen in section 2, the secrecy for general protocols of this form is undecidable. To obtain a decidable class, we consider protocols such that, for each rule, the variables which are shared by $M_1, \ldots, M_p, M'_1, \ldots, M'_q$ satisfy a "basicness condition". Roughly, such a condition will state that only one variable may occur several times in different contexts without being of basic type. For instance, if we don't assume any special ability of agents to recognize data types, then repeated variables occurring in different contexts must be agents names, except for possibly one such variable.

In order to express our condition, let X_s be a variable for each sort s in the signature and let $\mathcal{C}_{\mathtt{msg}}$ be the union of the definite set constraints

$$f(X_{\mathtt{sort}_1}, \ldots, X_{\mathtt{sort}_n}) \subseteq X_{\mathtt{sort}},$$

for every function symbol f of type $\mathtt{sort}_1 \times \ldots \times \mathtt{sort}_n \to \mathtt{sort}$, and the definite set constraints

$$X_{\mathtt{sort}} \subseteq X_{\mathtt{sort}'}$$

if $\mathtt{sort}$ is a subsort of $\mathtt{sort}'$. The *basic sorts* are defined as in definition 23: this is the largest set of sorts such that

- If s is basic then $\mathcal{R}(X_s)$ only contains one-way symbols and basic sorts.

- If s is basic then
  - either $\mathcal{R}(X_s)$ contains only one clause $e \subseteq X_s$ such that X_s does not occur in e.
  - or every function symbol occurring in $\mathcal{R}(X_s)$ occurs (possibly) only in $\mathcal{R}(X_{s'})$ where s' is basic.

For instance, we have seen in example 24 that, in our running example, all sorts are basic except `Message`. Let $\mathcal{B}_{\texttt{msg}}$ be the set of basic sorts.

Now, for each rule

$$rl = \{S(A, i, M), M_1, \ldots, M_p\} \longrightarrow \{S(A, i+1, M'), M'_1, \ldots, M'_q\},$$

for each variable Y occurring in rl, let

$$t_{rl} \stackrel{\text{def}}{=} <<A, M>, M_1, \ldots, M_p, <A, M'>, M'_1, \ldots, M'_q>,$$

and $S_{rl,Y} \stackrel{\text{def}}{=} \{p \text{ such that } t_{rl}|_p = Y\}$. Let $c_{rl,Y}$ be the equality constraint $p_1 = \ldots = p|_k$ for $p_j \in S_{rl,Y}$. In the particular case where $S_{rl,Y}$ is a singleton, $c_{rl,Y}$ is the empty constraint. Finally, let c_{rl} be the conjunction of the $c_{rl,Y}$ for all variables Y occurring in rl.

Example 54 *The t_{rl} and c_{rl} corresponding to our running example (see figure 1, example 5) are displayed in figures 8 and 9.*

For each term t, let $\widetilde{t}$ be the expression obtained by replacing in t each variable of sort s with X_s. Finally, let u_{rl} be the expression (with equality constraints) obtained from $\widetilde{t_{rl}}$ by adding (at the top) the constraint c_{rl}.

We are now ready to define the basicness condition.

Definition 55 *A protocol P satisfies the* basicness condition *if, for each rule rl of P, u_{rl} as defined above satisfies the basicness condition w.r.t. the set of basic sorts.*

We tried to give here a definition which is as general as possible, hence might be a bit difficult to grasp. Let us give a simple sufficient condition:

Proposition 56 *A protocol satisfies the basicness condition if, for each rule rl of the protocol, one of the following holds:*

- *There is at most one variable occurring at least twice in rl and whose sort is not basic.*
- *There is a decomposition of tl into $C[t_1, \ldots, t_n]$ such that every variable which is not of sort* `Agent` *occurs in at most one t_i.*

For the rule

$$\emptyset \longrightarrow \left\{ \begin{array}{l} S(A,1,<A,B,s>), \\ S(B,1,<B,s>), \\ S(s,1,s) \end{array} \right\}$$

we get the term and constraints:

$$\begin{array}{rcl} t_{rl_0} & \stackrel{\text{def}}{=} & << A, < A, B, s >>, < B, < B, s >>, < s, s >> \\ c_{rl_0} & \stackrel{\text{def}}{=} & 11 = 121 \wedge 122 = 21 = 221 \end{array}$$

For the rule 1:

$$\{S(A,1,<A,B,s>)\} \longrightarrow \{S(A,2,<A,B,s>), <A,B>\}$$

$$\begin{array}{rcl} t_{rl_1} & \stackrel{\text{def}}{=} & <<A, <A,B,s>>, <A, <A,B,s>>, <A,B>>, \\ c_{rl_1} & \stackrel{\text{def}}{=} & 11 = 121 = 212 = 221 \wedge 122 = 222 \end{array}$$

For the rule 2:

$$\left\{ \begin{array}{l} S(s,1,s) \\ <A,B> \end{array} \right\} \longrightarrow \left\{ \begin{array}{l} S(s,2,s), \\ \{B, K(A,B), \{A, K(A,B)\}_{\mathsf{shr}(B)}\}_{\mathsf{shr}(A)} \end{array} \right\}$$

$$\begin{array}{rcl} t_{rl_2} & \stackrel{\text{def}}{=} & <<s,s>, <A,B>, <s,s>, \\ & & \quad <\{B, k_1(A,B), \{A, k_1(A,B)\}_{\mathsf{shr}(B)}\}_{\mathsf{shr}(A)}>> \\ c_{rl_2} & \stackrel{\text{def}}{=} & 21 = 4121 = 41311 = 413121 = 42 \wedge 22 = 411 = 4122 = 413122 = 4132 \end{array}$$

For the rule 3:

$$\left\{ \begin{array}{l} S(A,2,<A,B,s>), \\ \{B,X,Y\}_{\mathsf{shr}(A)} \end{array} \right\} \longrightarrow \left\{ \begin{array}{l} S(A,3,<A,B,s,X,m(A,B)>), \\ <\{m(A,B)\}_X, Y> \end{array} \right\}$$

$$\begin{array}{rcl} t_{rl_3} & \stackrel{\text{def}}{=} & <<A, <A,B,s>>, \{B,X,Y\}_{\mathsf{shr}(A)}, \\ & & \quad <A, <A,B,s,X,n_1(A,B)>>, <\{n_1(A,B)\}_X, Y>> \\ c_{rl_3} & \stackrel{\text{def}}{=} & 11 = 121 = 22 = 31 = 321 = 3251 = 4111 \wedge \\ & & \quad 122 = 211 = 322 = 3252 = 4112 \wedge 212 = 324 = 412 \wedge 213 = 42 \end{array}$$

Figure 8: Terms and equality constraints corresponding to the protocol (rules 0 to 3).

For the rule 4:

$$\left\{ \begin{array}{l} S(B,1,<B,s>), \\ <\{Z\}_X, \{A,X\}_{\mathsf{shr}(B)}> \end{array} \right\} \longrightarrow \left\{ \begin{array}{l} \{S(B,1,<B,s,A,Z,X>), \\ \{h(Z)\}_X\} \end{array} \right\}$$

$$\begin{array}{rcl} t_{rl_4} & \stackrel{\text{def}}{=} & <<B,<B,s>>,<\{Z\}_X,\{A,X\}_{\mathsf{shr}(B)}>, \\ & & \quad <B,<B,s,A,Z,X>>,\{H(Z)\}_X\} \\ c_{rl_4} & \stackrel{\text{def}}{=} & 11 = 121 = 222 = 31 = 321 \wedge 211 = 324 = 411 \wedge 212 = 2212 = 325 = 42 \end{array}$$

Figure 9: Terms and equality constraints corresponding to the protocol (rule4).

For instance, in figures 8 and 9, only Y, Z have a non-basic sort, hence the first condition above is met by every rule.

For simplicity, we often write A and Msg instead of respectively $X_{\texttt{Agent}}$ and $X_{\texttt{Message}}$.

Theorem 57 *If P satisfies the basicness condition, then the secrecy of P is decidable.*

In particular, our running example satisfies the basicness condition. Indeed, there are at most two occurences of non-basic variables in each expressions. However, note that giving the ability for agents to recognize messages of the form $K(a,b)$ was not innocent: if we replace our key variable K by a message variable X then our protocol does not remain in our decidable class.

We prove the theorem in next section.

6.2 Proof of Theorem 57

The proof of Theorem 57 proceeds in two steps. First, we show that every protocol can be "translated" in Horn clauses such that a message can be sent if and only if a corresponding formula can be derived from the Horn clauses. Then, we show that if a protocol satisfies the basicness condition then the corresponding Horn clauses can be expressed as ET-constraints, thus secrecy is decidable.

Step 1

Lemma 58 *Let P be a protocol with its initial state H_0. Let I_0 the maximal set of initial knowledge of the intruder (compatible with the initial state H_0 of P) mentionned in section 2. Then, there exists a set $\mathcal{H}$ of Horn clauses with a special predicate I such that $I(m)$ is derivable from $\mathcal{C}$ (where m is a message), iff there exists a reachable H such that $m \in H$.*

Proof: $\mathcal{H}$ is built in the following way : it is the union of four sets: $\mathcal{H}_{\mathtt{msg}}$, $\mathcal{H}_{I_0}$, $\mathcal{H}_I$ and $\mathcal{H}_P$.
$\mathcal{H}_{\mathtt{msg}}$ corresponds to the construction of the messages with their sort: $\mathcal{H}_{\mathtt{msg}}$ is the union of the clauses

$$\frac{P_{\mathtt{sort}_1}(x_1) \ldots P_{\mathtt{sort}_n}(x_n)}{P_{\mathtt{sort}}(f(x_1, \ldots, x_n))}$$

for every function symbol f of type $\mathtt{sort}_1 \times \ldots \times \mathtt{sort}_n \rightarrow \mathtt{sort}$, where the $P_{\mathtt{sort}}$ are new predicates. We also add to $\mathcal{H}_{\mathtt{msg}}$ the union of the clauses

$$\frac{P_{\mathtt{sort}}(x)}{P_{\mathtt{sort}'}(x)}$$

for every $\mathtt{sort}, \mathtt{sort}'$ such that $\mathtt{sort}$ is a subsort of $\mathtt{sort}'$. Moreover, we need to distinguish between symmetric and public keys: every term is symmetric except the terms of the form $\mathsf{pub}(t)$ or $\mathsf{prv}(t)$. Thus we add to $\mathcal{H}_{\mathtt{msg}}$ the clauses

$$\frac{P_{\mathtt{Message}}(x_1) \ldots P_{\mathtt{Message}}(x_n)}{Sym(f(x_1, \ldots, x_n))}$$

for every function symbol f, $f \neq \mathsf{pub}$ and $f \neq \mathsf{prv}$. Then it is easy to prove that $P_{\mathtt{sort}}(m)$ is derivable from $\mathcal{H}_{\mathtt{msg}}$ if and only if m is a message of sort $\mathtt{sort}$ and $Sym(m)$ is derivable from $\mathcal{H}_{\mathtt{msg}}$ if and only if m is a symmetric term.
Then, by the following lemma (proved in Appendix E), there exists a set of Horn clauses $\mathcal{H}_{I_0}$ such that $I_0(m)$ is derivable from $\mathcal{H}_{I_0} \cup \mathcal{H}_{\mathtt{msg}}$ if and only if $m \in I_0$.

Lemma 59 *Let $t_1, \ldots, t_n$ be message schemes with the free variables $x_1, \ldots, x_k$. Then, there exists a set of Horn clauses $\mathcal{H}_{I_0}$ with two predicates I_0 and $P_{\geq p}$ such that $I_0(m)$ is derivable from $\mathcal{H}_{I_0} \cup \mathcal{H}_{\mathtt{msg}}$ if and only if $\mathsf{parts}(m) \cap \{t_i\sigma \mid 1 \leq i \leq n, \sigma(x_i) \in \mathcal{A}_h\} = \emptyset$.*

The clauses of $\mathcal{H}_I$ are described Figure 10. They simulate the capabilities of the intruder.

To simulate the protocol rules, we first define the set of expressions $types(t)$ generated by a message scheme t by induction on t. If t is a constant, then $types(t) = \emptyset$. If t is a variable of sort $\mathtt{sort}$, then $types(t) = \{P_{\mathtt{sort}}(t)\}$. If t is a term of the form $f(t_1, \ldots, t_n)$, then $types(t) = \bigcup_{1 \leq i \leq n} types(t_i)$. Intuitively $types(t)$ is the set of constraints corresponding to the sorts of the variables of t. We sometimes write $types(t_1, t_2)$ instead of $types(t_1) \cup types(t_2)$.

Example 60 *Let us consider the message scheme* $\{B, K, X\}_K$ *of our running example. Then*

$$types(\{B, X, Y\}_{\mathsf{shr}(A)}) = \{P_{\mathtt{Agent}}(B), P_{\mathtt{Key}}(X), P_{\mathtt{Message}}(Y), P_{\mathtt{Agent}}(A)\}$$

ensures that A *and* B *stand for agent variables,* X *stands for a key variable and* Y *for a message variable.*

Then $\mathcal{H}_P$ is the union, for each rule

$$\{S(A, i, M), M_1, \ldots, M_p\} \longrightarrow \{S(A, i+1, M'), M'_1, \ldots, M'_q\}$$

of the protocol P, of the clauses described in Figure 11.
We prove by induction in Appendix E that $\mathcal{H}$ verifies the required property, which concludes the proof of Lemma 58 □

Remark: The predicate symbols of $\mathcal{H}$ are the $P_{\mathtt{sort}}$, $P_{\geq p}$ introduced by Lemma 59 and three distinct predicate symbols: I, I_0 and Sym.

Example 61 *We present here the clauses* $\mathcal{H}_P$ *corresponding to our running example. For simplicity, we factorize the rules whose premisses are identical, letting them contain several conclusions (though it must be kept in mind that these are Horn clauses).*

Initialization rule:

$$\frac{P_{\mathtt{Agent}}(A) \quad P_{\mathtt{Agent}}(B)}{I(s_1(A, <A, B, s>)) \quad I(s_1(B, <B, s>)), \quad I(s_1(s, s))}\ .$$

Rule 1:

$$\frac{P_{\mathtt{Agent}}(A) \quad P_{\mathtt{Agent}}(B) \quad I(s_1(A, <A, B, s>))}{I(s_2(A, <A, B, s>)) \quad I(<A, B>)}$$

Rule 2:

$$\frac{P_{\mathtt{Agent}}(A) \quad P_{\mathtt{Agent}}(B) \quad I(s_1(s, s)) \quad I(<A, B>)}{I(s_2(s, s)) \quad I(\{B, K(A, B), \{x_A, K(A, B)\}_{\mathsf{shr}(B)}\}_{\mathsf{shr}(A)})}$$

Rule 3:

$$\frac{\begin{array}{c} P_{\mathtt{Agent}}(A)\ P_{\mathtt{Agent}}(B)\ P_{\mathtt{Key}}(X)\ P_{\mathtt{Message}}(Y) \\ I(s_2(A, <A, B, s>))\ I(\{B, X, Y\}_{\mathsf{shr}(A)}) \end{array}}{I(s_3(A, <A, B, s, X, m(A, B)>)) \quad I(<\{m(A, B)\}_X, Y>)}$$

and similarly for Rule 4.

Initial knowledge

$$\frac{I_0(x)}{I(x)}$$

Analyzis

$$\frac{I(f(x_1,\ldots,x_n))}{I(x_i)} \qquad f \in \mathcal{IF},\ \ 1 \le i \le n$$

$$\frac{I(\{x_1\}_{x_2}) \quad I(x_2) \quad Sym(x_2)}{I(x_1)}$$

$$\frac{I(\{x_1\}_{\mathsf{pub}(x_2)}) \quad I(\mathsf{prv}(x_2))}{I(x_1)} \qquad \frac{I(\{x_1\}_{\mathsf{prv}(x_2)}) \quad I(\mathsf{pub}(x_2))}{I(x_1)}$$

Synthesis

$$\frac{I(x_1) \cdots I(x_n)}{I(f(x_1,\ldots,x_n))} \qquad f \in \mathcal{PF}$$

$$\frac{I(x_1) \cdots I(x_n) \quad P_{\mathtt{Ad}}(x_{j_1}) \ldots P_{\mathtt{Ad}}(x_{j_k})}{I(f(x_1,\ldots,x_n))} \qquad \begin{array}{l} f \in \mathcal{AF}, \\ f \text{ restricted v.s.} \\ j_1,\ldots,j_k \end{array}$$

Figure 10: Horn clauses for the intruder capabilities

$$\frac{I(s_i(A,M)) \quad I(M_1) \quad \cdots I(M_p) \quad types(A,M,M_1,\ldots,M_p,M'_1,\ldots,M'_q)}{I(s_{i+1}(A,M'))}$$

$$\frac{I(s_i(A,M)) \quad I(M_1) \quad \cdots I(M_p) \quad types(A,M,M_1,\ldots,M_p,M'_1,\ldots,M'_q)}{I(M'_i)}$$

For $i = 1,\ldots,q$

Figure 11: Horn clauses corresponding to the rule $\{S(A,i,M), M_1,\ldots,M_p\} \longrightarrow \{S(A,i+1,M'), M'_1,\ldots,M'_q\}$

Step 2 We can start the second part of the proof of Theorem 57. We write $\mathcal{H} \vdash^\star E$ when E is derivable from $\mathcal{H}$.
Let P be a protocol. Its initial global state is a set of spell events

$$\{t_1^1, \ldots, t_{n_1}^1\} \ddagger \{x_1^1, \ldots, x_{k_1}^1, u_1^1, \ldots, u_{m_1}^1\}, \ldots \\ \{t_1^n, \ldots, t_{n_s}^s\} \ddagger \{x_1^s, \ldots, x_{k_s}^s, u_1^s, \ldots, u_{m_s}^s\}.$$

By definition of secrecy, P is not secret iff there exists a reachable H and a spell $C \in H$ such that $\mathsf{fake}(Cont(H) \cup I_0) \cap Book(C) = \emptyset$, i.e.,
iff there exists a reachable H' such that $Cont(H') \cap Book(C) = \emptyset$, i.e.,
iff there exists a reachable H', $\exists i, j, \exists a_{i_1}, \ldots, a_{i_p} \in \mathcal{A}_h$ such that $t_i^j(a_{i_1}, \ldots, a_{i_p}) \in H'$
iff, by Lemma 58, $\exists i, j, \exists a_{i_1}, \ldots, a_{i_p} \in \mathcal{A}_h$ such that $\mathcal{H} \vdash^\star I(t_i^j(a_{i_1}, \ldots, a_{i_p}))$.

Thus, we are left to prove that $\exists i, j, \exists a_{i_1}, \ldots, a_{i_p} \in \mathcal{A}_h$ such that $\mathcal{H} \vdash^\star I(t_i^j(a_{i_1}, \ldots, a_{i_p}))$ is decidable.

We express the set $\mathcal{H}$ of Horn clauses as a set constraint: the set of definite set constraint corresponding to $\mathcal{H}_{\mathtt{msg}}$ is the set $\mathcal{C}_{\mathtt{msg}}$ described Section 6.1 augmented with the inclusions:

$$f(Msg, \ldots, Msg) \subseteq X_{Sym}$$

for every function symbol $f \in \mathcal{F}$, $f \neq \mathsf{pub}$, $f \neq \mathsf{prv}$.
The definite set constraint $\mathcal{C}_I$ corresponding to $\mathcal{H}_I$ is described in figure 12. The set of definite set constraint $\mathcal{C}_{I_0}$ corresponding to $\mathcal{H}_{I_0}$ is constructed similarly.

We associate with each Horn clause described in Figure 11 the following ET-constraint:

$$< s_i(A, \widetilde{M}) \cap I, \widetilde{M_1} \cap I, \ldots, \widetilde{M_p} \cap I, s_{i+1}(A, \widetilde{M'}), \widetilde{M_1'}, \ldots, \widetilde{M_q'} >^{c_{rl}} \subset I \quad (6)$$

where c_{rl} and $\widetilde{\cdot}$ are defined in section 6.1. The union of these ET-constraint is denoted by $\mathcal{C}_P$.

Example 62 *The Horn clause corresponding to the rule 3 of our running example is expressed by:*

$$< s_2(A, < A, A, s >) \cap I, \{A, X_{\mathtt{Key}}, Msg\}_{\mathsf{shr}(A)} \cap I, \\ s_3(A, < A, A, s, X_{\mathtt{Key}}, m(A, A) >), < \{m(A, A)\}_{X_{\mathtt{Key}}}, Msg >>^{c_{rl_3}} \subseteq I$$

where c_{rl_3} is the equality constraint described in Figure 8, Section 6.1.

Let $\mathcal{C}$ be the union of $\mathcal{C}_{\mathtt{msg}}$, $\mathcal{C}_{I_0}$, $\mathcal{C}_I$ and $\mathcal{C}_P$. The set variables of $\mathcal{C}$ are the variables $X_{\mathtt{sort}}$ for each sort $\mathtt{sort}$ and the four additional variables: $X_{\geq p}$, which corresponds to $P_{\geq p}$, I, I_0 and X_{Sym}.

$\mathcal{C}$ is a faithful representation of $\mathcal{H}$:

Initial knowledge	$I_0 \subseteq I$	
Analyzis	$f(I,\ldots,I) \subseteq I$	$f \in \mathcal{IF}$
	$\{\}_1^{-1}(\{Msg\}_{I \cap Sym}) \subseteq I$	
	$\{\}_1^{-1}(\{Msg\}_{\mathsf{prv}(\mathsf{pub}^{-1}(I))}) \subseteq I$	$\{\}_1^{-1}(\{Msg\}_{\mathsf{pub}(\mathsf{prv}^{-1}(I))}) \subseteq I$
Synthesis	$f_i^{-1}(I) \subseteq I$	$f \in \mathcal{PF}, i \leq arity(f)$
	$f(\alpha_1,\ldots,\alpha_n) \subseteq I$	$f \in \mathcal{AF}$, f restricted v.s. $j_1,\ldots,j_k$ $\alpha_m = X_{\mathtt{Ad}}$ if $\exists i, m = j_i$, $\alpha_m = I$ otherwise.

Figure 12: Set constraints corresponding to the intruder capabilities

Lemma 63 *Let $\mathcal{M}$ be a collection of sets S_Q for every (unary) predicate symbol Q. Then $\mathcal{M}$ is a model of $\mathcal{H}$ iff the substitution $\sigma_{\mathcal{M}}$ assigning X_Q to S_Q is a solution of $\mathcal{C}$.*

Proof: Let $\mathcal{M}$ be a model of $\mathcal{H}$ and let us show that $\sigma_{\mathcal{M}}$ satisfies $\mathcal{C}$.

The only non obvious part of the proof is to show that S_I satisfies the set constraint defined in Equation 6. Let rl be a rule of the protocol, let

$$t \in < s_i(A, \widetilde{M}\sigma) \cap S_I, \widetilde{M_1\sigma} \cap S_I, \ldots, \widetilde{M_p}\sigma \cap S_I, s_{i+1}(A, \widetilde{M'}\sigma), \widetilde{M'_1}\sigma, \ldots, \widetilde{M'_q}\sigma >$$

such that t satisfies c_{rl}. $t =< s_i(a,m), m_1, \ldots, m_p, s_{i+1}(a,m'), m'_1, \ldots, m'_q >$ and $s_i(a,m), m_1, \ldots, m_p \in S_I$. Since t satisfies c_{rl}, by applying the clause defined in Figure 11, we deduce $s_{i+1}(a,m'), m'_1, \ldots, m'_q \in S_I$, thus, by applying the clause $\dfrac{I(m_1)\, I(m_2)}{I(<m_1, m_2>)}$, t is in S_I, thus S_I satifies the constraint 6.

Conversely, let $\sigma_{\mathcal{M}}$ be a model of $\mathcal{C}$. The only non obvious part of the proof is to show that S_I satisfies the Horn clause defined in Figure 11. Let $m = M\sigma', m = M'\sigma$, $m_i = M_i\sigma', m'_i = M'_i\sigma$, $a = A\sigma'$ such that $s_i(a,m), m_1, \ldots, m_p \in S_I$ and σ' satisfies the conditions $cond(A, M, M_1, \ldots, M_p, M'_1, \ldots, M'_q)$. Thus for every variable X of $A, M, M_1, \ldots, M_p, M'_1, \ldots, M'_q$, $\sigma'(X) \in \mathcal{I}(S_{\hat{X}})$: σ' respects the

type of the variables. Let $t =< s_i(a,m), m_1, \ldots, m_p, s_{i+1}(a,m'), m'_1, \ldots, m'_q >$, then

$$t \in < s_i(A, \widetilde{M}\sigma) \cap S_I, \widetilde{M_1\sigma} \cap S_I, \ldots, \widetilde{M}_p\sigma \cap S_I, \\ s_{i+1}(A, \widetilde{M'}\sigma), \widetilde{M}'_1\sigma, \ldots, \widetilde{M}'_q\sigma >$$

and, by construction of c_{rl}, t satisfies c_{rl}. Thus t is in S_I. Applying the set constraint $<>_i^{-1}(S_I) \subseteq S_I$, we get $s_{i+1}(a,m'), m'_1, \ldots, m'_q \in S_I$, thus S_I satisfies the Horn clause defined in Figure 11. □

Then, applying Lemma 63, it is easy to verify that

$$\exists i, \exists a_{i_1}, \ldots, a_{i_p} \in \mathcal{A}_h \text{ such that } \mathcal{H} \vdash^\star I(t_i(a_{i_1}, \ldots, a_{i_p}))$$

if and only if

$$\mathcal{C} \bigcup_i I \cap t_i(X_{\mathtt{Ah}}, \ldots, X_{\mathtt{Ah}}) \subseteq \bot$$

is satisfiable.

Assume now that P satisfies the basicness condition as in theorem 57, then the set $\mathcal{C}$ as constructed above is a set of ET-constraints. Thus

$$\mathcal{C} \bigcup_i I \cap t_i(X_{\mathtt{Ah}}, \ldots, X_{\mathtt{Ah}}) \subseteq \bot$$

is also a set of ET-constraints. Then, thanks to theorem 53, the satisfiability of this constraint is decidable, which completes the proof proof of theorem 57.

7 Conclusion

Let us summarize the contributions of the paper (roughly in increasing order of significance) and discuss their meaning and possible further developments.

1. The secrecy of a protocol P is undecidable, even for a restricted class in which there are no nonces, no compound keys and there is at least one honest instance of P. This is theorem 6. This shows that the source of undecidability does not come from nonces, but from the memorization and copying facilities of the agents.

2. The satisfiability of intersection constraints with non-emptiness guards is DEXPTIME-complete. This is theorem 13. It is a slight extension of results about set constraints.

3. We introduced the new class of tree automata with one memory and we showed that the emptiness is DEXPTIME-complete for this class. This is theorem 17. This result is interesting in itself. One open question is its generalization with disequality tests (and not only equality tests between memory contents).

4. We introduced a class of set constraints with equality tests, in which the tests are not restricted to brother positions. We showed the decidability of constraints in this class by a reduction to tree automata with one memory. This is theorem 53. It must be emphasized that we did not use the full power of automata with one memory here.

   Interpreting lemma 40 in the context of cryptographic protocols, it shows that, for basic variables, we may restrict our attention to finitely many instances (the representatives w.r.t. an appropriate equivalence relation). This shows in particular that we can assume w.l.o.g. that there is a bounded number of principals (the bound is given by the index of the equivalence relation).

   One possible research direction is to investigate generalizations of this lemma, for instance in the context of nonces: is there an equivalence relation (preserving the solutions) which reduces the general case to the case of finitely many nonces ? Such a result would not necessarily contradict the undecidability result of [17] since the protocol resulting from the coding of that paper does not satisfy the basicness hypothesis. In other words, as suggested by theorem 6, the key for deciding the secrecy of cryptographic protocols might be to limit the copying facilities of the agents, not the number of sessions or nonces they generate.

5. We showed the decidability of secrecy for a class of cryptographic protocols, without any assumptions on the number of sessions (whether parallel or not). This is theorem 57. This result is obtained by a reduction to set constraints with equality tests, but we did not use the full power of such constraints.

   The use of set constraints, abstracting away the order in which messages have been sent over the network is proved to be relevant. Also, the ability of agents to recognize different types of data appeared clearly as a simplification factor, which can be tuned so that we fall in or out of the decidable class. We have showed the relationship between this ability and the copying facilities of the agents: the more they are able to distinguish between different data types, the more they are allowed to copy blindly pieces of messages, without escaping from the decidable class.

There are still several weaknesses in our paper. First, the constraint solving technique is too complex: we conjecture that our algorithm is in DEXPTIME, though we only showed a doubly exponential upper bound. It is also too complicated for the applications we have in mind. That is mainly because we tried to be as general as possible. However, most of the time, we don't need such general constraints. In particular, we can avoid the most complicated step (lemma 29) simply by designing normalized constraints only.

Finally the big open question is the extension of these results to an unbounded number of nonces.

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] V. Cortier and J. Millen and Harald Rueß Proving Secrecy is easy enough *14th IEEE Computer Security Foundations Workshop*, pages 97-108, Cape Breton, 2001.

[11] H. Comon and V. Cortier and J. Mitchell Tree automata with one memory, set constraints and ping-pong protocols In *Proc. 28th Int. Coll. Automata, Languages, and Programming (ICALP'01)*, Crete, Greece, July 2001.

[12] H. Comon and V. Shmatikov Is it possible to decide whether a cryptographic protocl is secure or not ? Draft paper, 2001.

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

[14] 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.

[15] C. Dufourd, A. Finkel, and Ph. Schnoebelen Reset nets between decidability and undecidability. *Proc. 25th Int. Coll. Automata, Languages, and Programming (ICALP'98)*, Aalborg, Denmark, July 1998, volume 1443 of Lecture Notes in Computer Science, pages 103-115. Springer, 1998.

[16] N. Dershowitz and J. Jouannaud. *Handbook of Theoretical Computer Science*, volume B, chapter Rewrite Systems, pages 243–320. Elsevier, 1990.

[17] 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.

[18] 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.

[19] N. Heintze and J. Tygar. A model for secure protocols and their compositions. *IEEE transactions on software engineering*, 22(1), 1996.

[20] 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.

[21] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS)* (Margaria and Steffen, eds.), vol. 1055 of *Lecture Notes in Computer Science*, Springer-Verlag, pages 147–166, 1996.

[22] J. Mongy. *Transformation de noyaux reconnaissables d'arbres. Forêts RATEG.* PhD thesis, Laboratoire d'Informatique Fondamentale de Lille, Université des Sciences et Technologies de Lille, Villeneuve d'Ascq, France, 1981.

[23] J. Millen and H. Rueß Protocol-Independent Secrecy *Proc. of the Workshop on Formal Methods in Computer Security*, Chicago, 2000.

[24] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. To appear in *Proc. 8th ACM Conference on Computer and Communications Security*, 2001.

[25] L. Paulson The inductive approach to verifying cryptographic protocols *Journal of Computer Security*, 1998, volume 6, pages 85-128.

[26] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *Proc. 14th IEEE Computer Security Foundations Workshop*, pages 174–190, 2001.

[27] H. Seidl. Haskell overloading is DEXPTIME-complete. *Inf. Process. Lett.*, 52:57–60, 1994.

A Petri nets

In Section 2, we need to use Theorem 7 in a restricted way:

Theorem 64 *Given a Reset Petri net N and an initial configuration c_0 such that each transition of N can be applied (at least) once in a given order from c_0, the reachability of the empty configuration (where all places are empty) is undecidable.*

To prove this theorem, it is sufficient to prove the following lemma:

Lemma 65 *Given a Reset Petri net N and an initial configuration c_0, there exists a Reset Petri net N' and an initial configuration c'_0 such that each transition of N' can be applied (at least) once in a given order from c'_0 and such that N', c'_0 reaches the empty configuration if and only if N, c_0 reaches the empty configuration.*

We rename the transitions of N such that $t_1, \ldots, t_n$ are the classical transitions and $r_1, \ldots, r_p$ are the reset transitions. The places of N are $p_1, \ldots, p_m$. The idea is to add first α tokens to each of these places (let c_1 be the new configuration) such that for each transition t, t can be applied from c_1, to play each transition and its inverse, then to remove α tokens from the places and then to let the classical and reset transitions play normally.

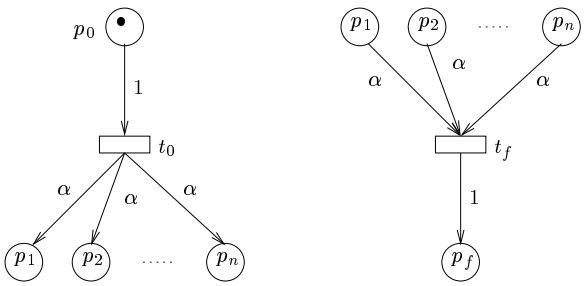


Figure 13: t_0 and t_f

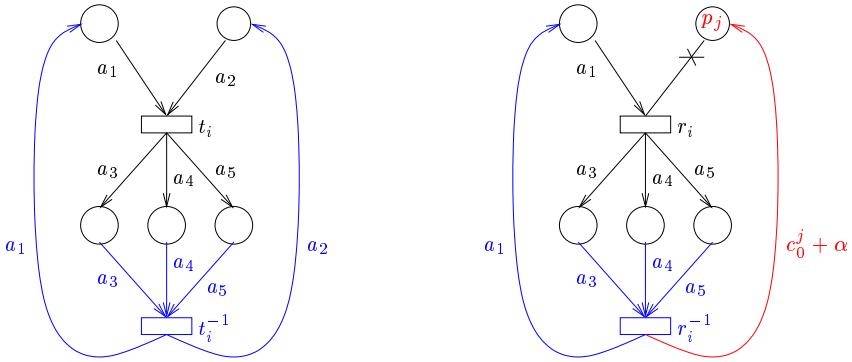


Figure 14: t_i and its inverse t_i^{-1}, r_i and its inverse r_i^{-1}

Let p_0 be a new place, initially with one token and let t_0 be a new transition which removes one token from p_0 and adds α tokens in each place $p_1, \ldots, p_n$, where α is the maximal number of tokens removed by the transitions t_i. Let t_f be the inverse transition with a new place p_f which is initially empty: t_f removes α tokens from each place $p_1, \ldots, p_n$ and adds one token to p_f, see Figure 13. Note that t_0 can be played only once.

For each transition t_i $(1 \leq i \leq n)$, let t_i^{-1} be the inverse transition. For each transition r_i $(1 \leq i \leq p)$, let r_i^{-1} be the "inverse" transition: if r_i resets the place p_j which has initially $c_1^j = c_0^j + \alpha$ tokens, then the inverse transition adds c_1^j tokens to the place p_j. (see Figure 14)

Now, each transition can be played in a given order:

$$t_0, t_1, t_1^{-1}, \ldots, t_n, t_n^{-1}, r_1, r_1^{-1}, \ldots, r_p, r_p^{-1}, t_f$$

but we need to force this order and then to prevent the new transitions $t_0, t_1^{-1}, \ldots, r_p^{-1}, t_f$.

For each transition t except t_0, let p_t be a new control place, initially empty. These places are fed in the following way: for each place p_t, a token is added to p_t by the transition preceding t and a token is removed by the transition t.

At this step, the transitions $t_0, t_1, t_1^{-1}, \ldots, t_n, t_n^{-1}, r_1, r_1^{-1}, \ldots, r_p, r_p^{-1}, t_f$ can be applied at most once, in this order. We need to free the transitions $t_1, \ldots, t_n$, $r_1, \ldots, r_p$: let t_{free} be a new transition, which removes a token from p_f and adds a token to p_f and to each place p_{t_i}, p_{r_j}, $1 \leq i \leq n, 1 \leq j \leq p$, see Figure 15.

Therefore, the only possibility for N', c_0' constructed above is to first play t_0, then $t_1, t_1^{-1}, \ldots, t_n, t_n^{-1}, t_f, t_{free}$. At this step, we recover the initial configuration and each play of N can be simulated in N' by inserting the transition t_{free}. Thus N', c_0' reaches the empty configuration if and only if N, c_0 reaches the empty configuration and each transition of N' can be applied (at least) once in a given order from c_0'.

B Correspondence between Reset Petri nets and protocols

We prove here the Lemma 8 used in Section 2.

Lemma 8. *Let N be a Petri net and c_0 its initial configuration, let P the corresponding protocol constructed like described in Section 2 then N, c_0 reaches the empty configuration if and only if P is not secret.*

We introduce first some notations: if c is a configuration $c = (c_1, \ldots, c_p)$, let $\overline{c}$ denote the message $A, A^{c_1}, A^{c_2}, \ldots, A^{c_p}, A$. If m is a message and k is a key, let $\{m\}_{k^n}$ denote the message m encrypted n times ($n \geq 0$) by k.

One way of this equivalence is obvious: assume N, c_0 reaches the empty configuration and consider $\{n(a), \mathsf{shr}(a)\} \ddagger a \in H_0$. We instantiate the first rule of P with $A = a$ and we simulate the transitions of the Petri Net on P by using the corresponding rules of the protocol until we obtain a reachable global state H such that the message $\{\{n(a)\}_{\mathsf{shr}(a)^{p+1}}, \overline{0}\}_{\mathsf{shr}(a)}$ is in H. (where 0 denotes the empty configuration). Then, by applying $p + 1$ times the final rule and a fake transition where the intruder forwards only the first part of the message, we obtain a reachable H' such that $n(a) \in H'$, thus P is not secret.

Assume now that P is not secret, i.e., there exists a reachable H and a cast C in H such that $\mathsf{fake}(Cont(H) \cup I_0) \cap Book(C) \neq \emptyset$. $C \in H$, thus $C \in H_0$, thus $C = \{n(a)\,\mathsf{shr}(a)\} \ddagger a$ for some a, honest agent: $a \in \mathcal{A}_h$. Let H be the smallest global state such that $\mathsf{fake}(Cont(H) \cup I_0) \cap Book(C) \neq \emptyset$, i.e, for all predecessor H' of H, $\mathsf{fake}(Cont(H') \cup I) \cap Book(C) = \emptyset$.

Few lemmas are useful to prove that N, c_0 reaches the empty configuration.

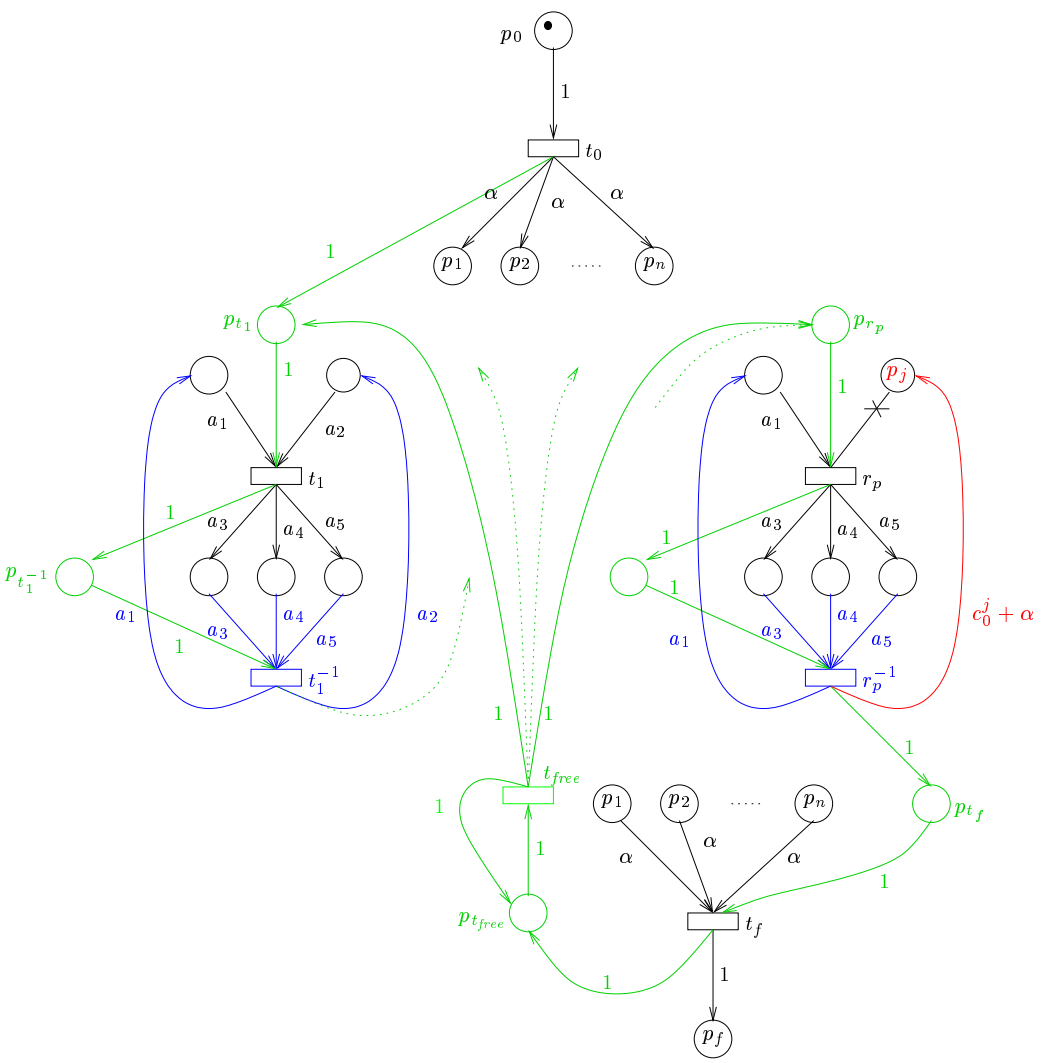


Figure 15: the control states

Lemma 66 *For all honest agent $a \in \mathcal{A}_h$, for all reachable H, $\mathsf{shr}(a) \notin \mathsf{fake}(H \cup I_0)$. In addition, if $\{m\}_{\mathsf{shr}(a)} \in \mathsf{fake}(H \cup I_0)$, then m is of the form*

$$< \{n_1(a)\}_{\mathsf{shr}(a)^k}, w >$$

where w is of the form $\overline{c}$ for some $c = (c_1, \ldots, c_n)$ or $m = \{n_1(a)\}_{\mathsf{shr}(a)^k}$ for some k.

Proof: This property is proved by induction: it is true for H_0 and it is preserved by application of a fake transition. All the rules of the protocol, except the final rule, clearly preserve the property. Assume H verifies the property and let H' be an honest successor by application of the final rule: there exists $m \in H$ such that $m = \{\{m_1\}_{\mathsf{shr}(b)}, b, m_2\}_{\mathsf{shr}(b)}$ and $Cont(H') = Cont(H) \cup \{\{m_1, m_2\}_{\mathsf{shr}(b)}, m_1\}$. Since H verifies the property, m is of the form $\{\{n_1(b)\}_{\mathsf{shr}(b)^k}, w, b\}_{\mathsf{shr}(b)}$ where w is of the form $\overline{c}$ for some $c = (c_1, \ldots, c_n)$.

Assume H' verifies $\mathsf{shr}(a) \in \mathsf{fake}(H' \cup I_0)$ for some honest agent $a \in \mathcal{A}_h$. Since $\mathsf{shr}(a) \notin \mathsf{fake}(H \cup I_0)$ (H verifies the property), we have $\mathsf{shr}(a) \in \mathsf{fake}(\{\{\{n(b)\}_{\mathsf{shr}(b)^k}, w\}_{\mathsf{shr}}(b), \{n(b)\}_{\mathsf{shr}(b)^{k-1}}\} \cup I_0)$. But $\mathsf{shr}(a) \notin \mathsf{fake}(I_0)$ since a is an honest agent, thus $b = a$, $k = 1$ and $n(b) = \mathsf{shr}(a)$, contradiction (nonces and keys are distinct).

Let $m \in H'$ be a message such that $\{m\}_{\mathsf{shr}(a)} \in \mathsf{fake}(H' \cup I_0)$. Either $\{m\}_{\mathsf{shr}(a)} \in \mathsf{fake}(H \cup I_0)$, thus the second part of the property is verified. Or $\{m\}_{\mathsf{shr}(a)} \notin \mathsf{fake}(H \cup I_0)$ and $\{m\}_{\mathsf{shr}(a)} \in \mathsf{fake}(\{\{\{n(b)\}_{\mathsf{shr}(b)^k}, w\}_{\mathsf{shr}(b)}, \{n(b)\}_{\mathsf{shr}(b)^{k-1}}\} \cup I_0)$, thus $b = a$ and either $m = \{n(a)\}_{\mathsf{shr}(a)^{k-2}}$, $k \geq 2$ or $m = \{n(a)\}_{\mathsf{shr}(a)^k}, w$ where w is of the form $\overline{c}$ for some $c = (c_1, \ldots, c_n)$. Thus, in any case, the second part of the property is verified, which concludes the proof. □

Lemma 67 *Let H be a reachable global state in P.*
If $\{\{n(a)\}_{\mathsf{shr}(a)^{p+1}}, \overline{c}\}_{\mathsf{shr}(a)} \in H$ for some $a \in \mathcal{A}_h$, then N, c_0 reaches the configuration c.

Proof: This property is true for H_0. It is preserved by honest transitions and by the fake transition since $\mathsf{shr}(a) \notin \mathsf{fake}(H \cup I_0)$ by lemma 66. Thus, by induction, this property is true for all reachable H.

Let's return to the proof of Lemma 8. Let H' be the first predecessor of H: $\mathsf{fake}(Cont(H) \cup I_0) \cap Book(C) \neq \emptyset$ and $\mathsf{fake}(Cont(H') \cup I_0) \cap Book(C) = \emptyset$. By Lemma 66, $\mathsf{shr}(a) \notin \mathsf{fake}(H \cup I_0)$, we have necessarily $n(a) \in \mathsf{fake}(Cont(H) \cup I_0)$. Since $n(a) \notin \mathsf{fake}(Cont(H') \cup I_0)$, H is an honest successor of H'. By inspection of the rule of P and since, $\mathsf{shr}(a) \notin \mathsf{fake}(H \cup I_0)$, the last rule has been applied, i.e., there exists a message w such that $\{\{n(a)\}_{\mathsf{shr}(a)}, a, w\} \in H'$. Using again

Lemma 66, w is necessarily the empty message, i.e., $\{\{n_a\}_{\mathsf{shr}(a)}, a\} \in H'$. By applying p times this argument, we conclude $\{\{n_1(a)\}_{\mathsf{shr}(a)^{p+1}}, a, a^{p+1}\} \in H'$. By Lemma 67, N, c_0 is reachable. □

C Proof of lemma 29

Lemma 29 *Every expression e which satisfies the basicness condition can be transformed in a normal expression e' such that, for every σ, $[\![e]\!]_\sigma = [\![e']\!]_\sigma$ and, moreover $|e'|_{\mathcal{F}}$ and $|e'|_t$ are polynomially bounded by $|e|_t$ and $|e'|_{\mathcal{F}}$.*

Note first that condition 2 is initially satisfied by all expressions since we assumed that any two expressions in $P(c)$ are incomparable with respect to the prefix ordering.

We are going to perform successive transformations, verifying more and more conditions, while preserving those which have already been reached. Initially, we only assume that condition 1 is satisfied, as stated in the hypothesis of the lemma.

Step 1 The goal of this step is to ensure, in addition to properties 1 and 2 a property, which implies condition 3.

If c is an equality test and p is a position, we write $p \cdot c$ the equality test $\bigwedge_{q \sim_c r} p \cdot q = p \cdot r$. In addition of condition 3, we want to ensure that in any expression $g^c(\ldots, e_i, \ldots)$ such that, for some c_1, $i \cdot c_1 \subseteq c$, if $e_i = f^{c'}(\vec{e'}) \cap e''$, then $P(c_1) \subseteq \Pi(f^{c'}(\vec{e'}))$.

To define our rule, we first need to introduce a new equivalence relation:

Definition 68 *Given an equality test c such that c is satisfiable, we define $\simeq_c$ and S_c to be the least set S and equivalence relation $\simeq$ such that:*

$$
\begin{array}{l}
p \sim_c q \Rightarrow p, q \in S \text{ and } p \simeq q \\
p \sim_{c'} q \Rightarrow p, q \in S \text{ and } p \simeq q \\
p \simeq q \Rightarrow q \simeq p \\
p \simeq q, q \simeq r \Rightarrow p \simeq r \\
\left.\begin{array}{l} p \cdot q \in S \\ p \simeq p' \end{array}\right\} \Rightarrow \left\{\begin{array}{l} p' \cdot q \in S \\ p \cdot q \simeq p' \cdot q \end{array}\right.
\end{array}
$$

Lemma 69 *The fixed point for $\simeq_c$ and S_c is reached after a finite number of steps. In addition, there exists an order $\leq_c$ on the equivalence classes of $\simeq_c$ such that $u \leq_c u'$ implies that no position of u is a prefix of a position of u'.*

Proof: (sketch) First, if there are two non-empty positions such that $p \cdot q \simeq p$ then c is unsatisfiable: by induction (on the fixed point computation of $S, \simeq$), if

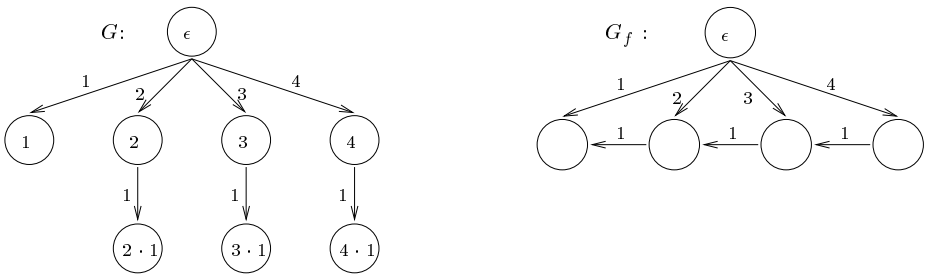


Figure 16: G and G_f for c

$t \models c \wedge c'$, then for every $p \simeq q$, $t \models p = q$. Since we are only considering finite terms, we cannot have $t|_p = t|_{p \cdot q}$.

Consider the DAG G whose vertices are elements of $P(c) \cup P(c')$ and (labeled) edges $p \xrightarrow{i} p \cdot i$. Then, for each $p \sim_{c_i} p'$, merge the two corresponding vertices. We get a new graph G_f, whose set of vertices is contained in the original set of vertices. Then, by induction (on the fixed point computation of $S, \simeq$), S is included in the set of paths of in G_f and if $p \simeq q$, then the path labelled by p and those labelled by q leads to the same vertice (starting from the vertice ϵ). As we have seen above, G_f is acyclic, thus S and $\simeq$ are finite, thus the fixed point for S, $\simeq$ is reached after a finite number of steps. Note that the number of vertices of G_f is smaller or equal to the number of vertices of G.

Since G_f is acyclic, G_f induces an order $\leq$ on its vertices such that if $v \leq v'$ than no path leading to the vertice v is a prefix of a path leading to the vertice v'. Each equivalence class u of $\simeq$ is included in the set of paths of one of the vertice v_u of the graph. We first order arbitrarily the equivalence classes which lead to the same vertices and then we extend this order by $u \leq_c u'$ if $v_u \leq v_{u'}$ and $v_u \neq v_{u'}$. Then $u \leq_c u'$ implies that no position of u is a prefix of a position of u'. □

Example 70 *Consider* $c \stackrel{def}{=} 1 = 21 \wedge 3 = 41 \wedge 2 = 31$. *Then, the graphs* G *and* G_f *are pictured in figure 16 and*

$$\simeq_c = \{1 = 21, 3 = 41, 2 = 31, 21 = 311, 31 = 411, 311 = 4111\}.$$

In addition, we define the *equality test* c_e *of an expression* e by induction on e by:

$$\begin{aligned} c_{e_1 \wedge \ldots \wedge e_n} &= c_{e_1} \wedge \ldots \wedge c_{e_n} \\ c_{f^c(e_1, \ldots, e_n)} &= c \wedge 1 \cdot c_{e_1} \wedge \ldots \wedge n \cdot c_{e_n} \wedge 1 = 1 \wedge \ldots \wedge n = n \end{aligned}$$

Let $\simeq_{c_e}$ be the equivalence relation corresponding to c_e and $u_1, \ldots, u_n$ be its equivalence classes numbered in such a way that $u_i \leq_{c_e} u_j$ implies $i \leq j$ (this is possible thanks to in lemma 69).

Then, (N_1) is the successive application of $N_{u_1}, \ldots, N_{u_n}$ where

$$(N_u) \quad e \to e[\bigcap_{\substack{p' \in u \cap \Pi(e) \\ e'' \in e|_{p'}}} e'']_{p \in u \cap \Pi(e)}$$

Note: The rule (N_1) is obviously terminating since it requires at most as many steps as the number of classes modulo $\simeq_{c_e}$. However, its complexity is unclear. We conjecture for instance that $|e'|_{\mathcal{F}}$ is plynomially bounded by $|e|_{\mathcal{F}}$ and $|e|_t$.

Lemma 71 *(N_1) preserves the semantics as well as properties 1 and 2.*

Proof: For every equivalence class u and for every $p_1, p_2 \in u \cap \Pi(e)$, by construction, $c_e \models p_1 = p_2$. Hence the rule preserves the semantics.

Now, it preserves properties 1 and 2 since it consists in repeatedly replacing an expression e' with the intersection of e' and other expressions, without changing the tests and, if an expression is basic, then its intersection with any other expression is also basic. □

Lemma 72 *If e' is the result applying (N_1) to e, then:*

- *$\forall p \in S_{c_e} \cap \Pi(e')$, $e'|_p$ is a singleton.*
- *$\forall p \simeq_{c_e} p'$, $e'|_p = e'|_{p'}$.*
- *if $e'|_p = f^c(e_1, \ldots, e_n) \cap e''$, then $p \cdot c \subseteq \simeq_{c_e}$ and for all $1 \leq i \leq n$, $p \cdot i \in S_{c_e}$.*
- *if $p \in \Pi(e')$ and $p > p'$ for some $p' \in u_i$ where u_i is minimal for $\leq_c$, then $e'|_p \subseteq \bigcup_{q \in \Pi(e)} e|_q$.*
- *if $e'|_p = f^c(e_1, \ldots, e_n) \cap f^{c'}(e'_1, \ldots, e'_n) \cap e''$ then for every $i = 1, .., n$, $e_i = e'_i$.*

Proof: We prove by induction on k that if e' is the result of applying $N_{u_1} \ldots N_{u_k}$ to e, then for all $i \leq k$,

1. $\forall p \in u_i \cap \Pi(e')$, $e'|_p$ is a singleton.
2. $\forall p, p' \in u_i \cap \Pi(e')$, $e'|_p = e'|_{p'}$.

3. if $e'|_p = f^c(e_1, \ldots, e_n) \cap e''$, then $p \cdot c \subseteq \simeq_{c_e}$ and for all $1 \leq i \leq n$, $p \cdot i \in S_{c_e}$.

4. if $p \in \Pi(e')$ and $p > p'$ for some $p' \in u_i \cap \Pi(e'), i \leq k$ where u_i is minimal for $\leq_c$, then $e'|_p \subseteq S(e)$, where $S(e) \stackrel{\text{def}}{=} \bigcup_{q \in \Pi(e)} e|_q$.

If $k = 0$, i.e., no rule has been applied, then 1, 2 and 4 are true. 3 is true by construction of S_{c_e} and $\simeq_{c_e}$.
Assume it is proved for k and let us prove the property for $k+1$. We consider e'' the result of e' by $N_{u_{k+1}}$ where e' is the result of e by the application of $N_{u_1} \ldots N_{u_k}$.

$$e'' = e'[\bigcap_{\substack{p' \in u_{k+1} \cap \Pi(e') \\ e''' \in e'|_{p'}}} e''']_{p \in u_{k+1} \cap \Pi(e')}.$$

Consider $q, q' \in u_i \cap \Pi(e'')$, $i < k+1$.

- either q is incomparable with the paths of $u_{k+1} \cap \Pi(e)$, then $e''|_q = e'|_q$ and, by induction hypothesis, $e|_q$ is a singleton.

- or there exists $p \in u_{k+1} \cap \Pi(e)$ such that $q \geq p$, i.e. $q = p \cdot q_1$, then

$$e''|_q = e'[\bigcap_{\substack{p' \in u_{k+1} \cap \Pi(e') \\ e'''|_{q_1} \in e'|_{p'}}} e''']_{p \in u_{k+1} \cap \Pi(e')}.$$

  By construction of our equivalence relation, since $p \simeq p'$ and $p \cdot q_1 \in u_i$, we have $p' \cdot q_1 \in u_i$. Thus, by induction hypothesis $e'|_{p' \cdot q_1} = e'|_{p \cdot q_1}$ and is a singleton,thus $e''|_q = e'|_q$.

- or there exists $p \in u_{k+1} \cap \Pi(e)$ such that $q < p$, which is impossible by the choice of the order on the equivalence classes.

Conclusion: in any cases, we have that $e''|_q = e'|_p$ is a singleton and $e''|_q = e''|_{q'}$.
Assume $p > q$ for some $p \in \Pi(e'')$ and assume u_i is minimal w.r.t. $\leq_c$, then $e''|_q = e'|_p \subseteq S(e)$ by induction.
Consider $q, q' \in u_{k+1} \cap \Pi(e'')$. Assume $q \notin \Pi(e')$. Then q is a path created by application of $N_{u_{k+1}}$. This means that there exists $p \in u_{k+1} \cap \Pi(e')$ such that p is a strict prefix of q which contradicts G_f acyclic.
Thus q, q' are in $u_{k+1} \cap \Pi(e')$, thus by construction

$$e''|_q = e''|_{q'} = \bigcap_{\substack{p' \in u_{k+1} \cap \Pi(e') \\ e''' \in e'|_{p'}}} e'''$$

is a singleton. Assume $p > q$, i.e. $p = q \cdot p_1$, for some $p \in \Pi(e'')$ and assume u_i is minimal w.r.t. $\leq_c$, then for all $p' \in u_{k+1} \cap \Pi(e')$, we have $e'|_{p'} = e|_{p'}$, since by minimality of u_{k+1} no rule can have been applied above p' for $p' \in u_{k+1} \cap \Pi(e')$. Thus

$$e''|_p = \bigcup_{\substack{p' \in u_{k+1} \cap \Pi(e) \\ e''' \in e|_{p'}}} e'''|_{p_1} \subseteq S(e)$$

It remains to prove 3: assume $e''|_q = f^c(e_1, \ldots, e_n) \cap e^4$. Then, either q is incomparable with the paths of $u_{k+1} \cap \Pi(e')$, then $e''|_q = e'|_q$ and we can apply the induction hypothesis. Or $q < p$ for some $p \in u_{k+1} \cap \Pi(e')$, then $e'|_q = f^c(e'_1, \ldots, e'_n) \cap e^{4'}$) and 3 is ensured by induction. Or (last case) $q \geq p$ for some $p \in u_{k+1} \cap \Pi(e')$, then $q = p \cdot q_1$ and there exists $p' \in u_{k+1} \cap \Pi(e')$ such that $e''|_q = e'|_{p' \cdot q_1}$. Thus, by induction hypothesis $p' \cdot c \in \simeq_{c_e}$ and for every $1 \leq i \leq n$, $p' \cdot i \in S_{c_e}$. Now, by construction of $\simeq_{c_e}$ and S_{c_e}, since $p \simeq_{c_e} p'$, we have $p \cdot c \in \simeq_{c_e}$ and for every $1 \leq i \leq n$, $p \cdot i \in S_{c_e}$.

Now consider the last property. c_e contains the identities $p = p$ for $p \in \Pi(e)$. Hence, if $f^c(e_1, \ldots, e_n) \cap f^{c'}(e'_1, \ldots, e'_n) \cap e'' \in e'|_p$, then $e_i = e'_i$ is the intersection of expressions $e|_{p'}$ such that $p \cdot i \simeq_{c_e} p'$ (note that $e_i, e'_i \in e'|_{p \cdot i}$.

□

Thanks to lemma 72, we have the required properties:

Corollary 73 *If e' is the result of e by the application of (N_1), then for every equality test, if $p \sim_c q$ then the expressions at positions p and q in e' are identical. In addition, in any expression $g^c(\ldots, e_i, \ldots)$ such that, for some c_1, $i \cdot c_1 \subseteq c$, if $e_i = f^{c'}(\vec{e'}) \cap e''$, then $P(c_1) \subseteq \Pi(f^{c'}(\vec{e'}))$.*

Example 74 *Consider*

$$e = f^{1=21 \wedge 3=41}(X, g(X), Y, g(Y)) \cap f^{2=31}(X_1, Z, g(Z), X_2)$$

Then $c_e \stackrel{def}{=} 1 = 21 \wedge 3 = 41 \wedge 2 = 31$ and the equivalence classes of $\simeq_{c_e}$ are:

$$u_1 = \{1, 2 \cdot 1, 3 \cdot 1 \cdot 1, 4 \cdot 1 \cdot 1 \cdot 1\}, u_2 = \{2, 3 \cdot 1, 4 \cdot 1 \cdot 1\}, u_3 = \{3.4 \cdot 1\}, u_4 = \{4\}$$

The successive applications of N_{u_1}, N_{u_2}, N_{u_3} and N_{u_4} are described figure 17.

Note that, now, "every expression in $e|_p$ is basic" is equivalent to "there is an expression in $e|_p$ which is basic" since the intersection of a basic expression with any other expression yields a basic expression. That is why, from now on,

$$
\begin{array}{rcl}
e & \overset{N_{u_1}}{\to} & e_1 \overset{\text{def}}{=} f^{1=21\wedge 3=41}(X\cap X_1, g(X\cap X_1), Y, g(Y)) \\
 & & \quad \cap f^{2=31}(X\cap X_1, Z, g(Z), X_2) \\
e_1 & \overset{N_{u_2}}{\to} & e_2 \overset{\text{def}}{=} f^{1=21\wedge 3=41}(X\cap X_1, Z\cap g(X\cap X_1), Y, g(Y)) \\
 & & \quad \cap f^{2=31}(X\cap X_1, Z\cap g(X\cap X_1), g(Z\cap g(X\cap X_1)), X_2) \\
e_2 & \overset{N_{u_3}}{\to} & e_3 \overset{\text{def}}{=} f^{1=21\wedge 3=41}(X\cap X_1, Z\cap g(X\cap X_1), Y\cap g(Z\cap g(X\cap X_1)), \\
 & & \qquad\qquad g(Y\cap g(Z\cap g(X\cap X_1)))) \\
 & & \quad \cap f^{2=31}(X\cap X_1, Z\cap g(X\cap X_1), Y\cap g(Z\cap g(X\cap X_1)), X_2) \\
e_3 & \overset{N_{u_3}}{\to} & e_3 \overset{\text{def}}{=} f^{1=21\wedge 3=41}(X\cap X_1, Z\cap g(X\cap X_1), Y\cap g(Z\cap g(X\cap X_1)), \\
 & & \qquad\qquad X_2\cap g(Y\cap g(Z\cap g(X\cap X_1)))) \\
 & & \quad \cap f^{2=31}(X\cap X_1, Z\cap g(X\cap X_1), Y\cap g(Z\cap g(X\cap X_1)), \\
 & & \qquad\qquad X_2\cap g(Y\cap g(Z\cap g(X\cap X_1))))
\end{array}
$$

Figure 17: Reduction of e by (N_1)

we may say, by abuse of language that "$e|_p$ is basic" to mean either of the two above versions.

We consider, in addition to the rule (N_1), the following "cleaning" rules:

$$(N_2)\quad f^c(e_1,\ldots,e_n)\cap f(e'_1,\ldots,e'_n)\to f^c(e_1\cap e'_1,\ldots,e_n\cap e'_n)$$

$$(N_3)\quad f^c(e_1,\ldots,e_n)\cap g^{c'}(e'_1,\ldots,e'_m)\to\perp \text{ if } f\neq g.$$

Lemma 75 *The rules $(N_2), (N_3)$, applied to normal forms w.r.t. (N_1), preserve the semantics as well as properties 1, 2, 3 and the properties described in lemma 72.*

Proof: First, by lemma 72, in any application of (N_2), we must have $e_i = e'_i$. Then, it sufficient to notice that c_e is unchanged by application of (N_2), hence its application does not trigger (N_1) and preserves the properties of lemma 72. □

Step 2 We start with some properties of equality tests.

Lemma 76 *Let $j\cdot c_1$ be the subset of c_2 containing all equalities whose both sides are prefixed by j. If c_2 satisfies the basicness condition in e and e satisfies condition 3, then c_1 satisfies the basicness condition in every expression belonging to $e|_j$.*

Proof: Let $p \cdot i_1 \cdot q \sim_{c_1} p'$, $i_1 \neq i_2$ and $p' \not\geq_{pref} p$. Then $j \cdot p \cdot i_1 \cdot q \sim_{c_2} j \cdot p'$ and $j \cdot p' \not\geq_{pref} j \cdot p$. By basicness of c_2, either $e|_{j \cdot p'}$ is basic or $e|_{j \cdot p \cdot i_2}$ contains basic expressions only or else $j \cdot p \cdot i_2 \cdot w \sim_{c_2} j \cdot p'$ for some w, which implies $p \cdot i_2 \cdot w \sim_{c_1} p'$ □

Lemma 77 *If c satisfies the basicness condition in e and c_1 is an equivalence class of c, then $c \setminus c_1$ satisfies the basicness condition in e.*

Proof: Let $p \cdot i_1 \cdot q \sim_{c \setminus c_1} p'$, $i_1 \neq i_2$ and $p' \not\geq_{pref} p$. Then $p \cdot i_1 \cdot q \sim_c p'$ and, by basicness of c, either $e|_{p'}$ or every expression in $e|_{p \cdot i_2}$ is basic or else $p \cdot i_2 \cdot w \sim_c p'$for some w. In the latter case, since c_1 is an equivalence class, $p \cdot i_2 \cdot w \sim_{c \setminus c_1} p'$. □

Now, we use the following two transformation rules:

$$(N_4) \quad f^{c \wedge j \cdot c_1}(\ldots, e_j, \ldots) \to f^{c \wedge j \cdot c_1}(\ldots, e_j \cap g^{c_1}(e_j^1, \ldots, e_j^{k_j}), \ldots)$$

if $e_j = g^{c'}(e_j^1, \ldots, e_j^{k_j}) \cap e''$ and c does not contain any test whose both sides are prefixed by j.

$$(N_5) \quad f^{c \wedge p_1 \cdot c_1 \wedge \ldots \wedge p_n \cdot c_n}(\vec{e}) \to f^c(\vec{e})$$

if $p_1 \cdot c_1 \cup \ldots p_n \cdot c_n$ is a union of equivalence classes, every p_i is non empty and, for every i, there is an expression e_i at position p_i in $f^{c \wedge \cdots}(\vec{e})$ such that $e_i = g^{c_i'}(\vec{e_i'}) \cap e_i''$ and $c_i' \models c_i$. (In words: we may remove classes which are consequences of equality tests lower in the expression).

Lemma 78 *(N_4) and (N_5) preserve conditions 3, 1 and 2. An expression which is unchanged by application of these two rules satisfies condition 4. Moreover, the the size (w.r.t. $\mathcal{F}$) is preserved by the two rules, the size (w.r.t. t) is reduced by the second rule and, using repeatedly the first rule in an expression e results in an expression e' such that $|e'|_t \leq |e|_t^2$.*

Proof: (sketch) By lemmas 76 and 77, these transformations preserve condition 1 and they preserve trivially condition 3.

Condition 2 is also preserved since we did not merge any equality test so far.

The satisfaction of condition 4 follows from an inspection of the expressions which are left unchanged by any application of these rules.

The preservation of $|e|_{\mathcal{F}}$ follows from the definition. If c yields a new test c', possibly after repeated applications of (N_4), then there is a p_1 such that c' consists in equalities $p = q$ such that $p_1 \cdot p \sim_c p_1 \cdot q$. If we fix the size of p_1, then the sum of sizes of such c' is bounded by $|c|$. Hence the total size of the new tests is bounded by $|c|^2$. □

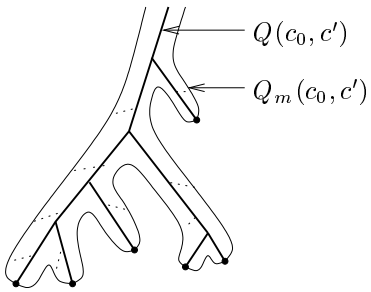


Figure 18: A representation of $Q(c_0, c')$ and $Q_m(c_0, c')$

Step 3 The purpose of this step is to show how to satisfy in addition the condition 5, while preserving properties 3, 1,4, 2. In what follows, integers $i, j...$ are always assumed range over a finite set $1..n$ which is consistent with the arity of function symbols.

Let e be an expression $f^c(\vec{e_1}) \cap f^{c'}(\vec{e_2}) \cap \ldots$. If e is in normal form w.r.t. (N_i), $i \leq 5$, then we may assume that $\vec{e_1} = \vec{e_2}$. Indeed, this is true of normal forms w.r.t. $(N_1), (N_2), (N_3)$ thanks to lemmas 72 and 75, and such a property is trivially preserved by the rules $(N_4), (N_5)$. We will however only assume in what follows the weaker property $\Pi(f^c(\vec{e_1})) = \Pi(f^{c'}(\vec{e_2}))$.

We define $c \sqcap c'$ (relatively to e) as follows: first, if $c \wedge c'$ is unsatisfiable, we replace it with $\bot$. Otherwise, for every non-trivial equivalence class c_0 for c, let

$$Q(c_0, c') = \{w \mid \exists p \in P(c_0),\ p \cdot w \in P(c'),\ e|_{p \cdot w} \text{ not basic } \}$$

and

$$Q_m(c_0, c') = \{w \cdot i \mid \exists w' \in Q(c_0, c'),\ w <_{pref} w',\ \forall w' \in Q(c_0, c'), w \cdot i \not<_{pref} w'\}.$$

Intuitively, $Q_m(c_0, c')$ is the border of $Q(c_0, c')$ with its maximal elements. See figure 18.

Each time $Q(c_0, c')$ is empty, we let $\overline{c_0} = c_0$ and, otherwise:

$$\overline{c_0}^{c'} \stackrel{\text{def}}{=} \bigwedge_{p \sim_{c_0} q} \bigwedge_{w \in Q_m(c_0, c')} p \cdot w = q \cdot w$$

Then

$$\overline{c}^{c'} \stackrel{\text{def}}{=} \bigwedge_{c_0 \text{a class of } c} \overline{c_0}^{c'}$$

We define now the sequence c_n as follows: $c_1 = c$, $c'_1 = c'$ and $c_{n+1} \stackrel{\text{def}}{=} \overline{c_n}^{c'_n}$ and $c'_{n+1} \stackrel{\text{def}}{=} \overline{c'_n}^{c_n}$.

Note that if $p \cdot w = q \cdot w \in \overline{c}^{c'}$, then $p \cdot w, q \cdot w \in \Pi(f^c(\vec{e_1})) = \Pi(f^{c'}(\vec{e_2})))$. Moreover, if $c \neq \overline{c}$, then $|\overline{c}| > |c|$. It follows that the sequences c_n and c'_n are ultimately stationary: let c_∞ and c'_∞ be the respective limit values of c_n and c'_n. We define

$$c \sqcap c' \stackrel{\text{def}}{=} c_\infty \wedge c'_\infty.$$

Example 79 *Let us consider* $c \stackrel{def}{=} 1 = 2$ *and* $c' \stackrel{def}{=} 211 = 3 \wedge 11 = 4$ *and assume that every position of* c *or* c' *is not basic. Then* $c \sqcap c'$ *is computed in two steps:*

step 1 $Q(c, c') = \{11, 1\}$ *thus* $Q_m(c, c') = \{11\}$.
$Q(c'^1, c) = Q(c'^2, c) = \emptyset$ *where* $c'^1 = 211 = 3$ *and* $c'^2 = 11 = 4$.
Thus $c_2 \stackrel{def}{=} \overline{c}^{c'} = 111 = 211$ *and* $c'_2 \stackrel{def}{=} \overline{c'}^{c} = c' = 211 = 3 \wedge 11 = 4$.

step 2 $Q(c_2, c'_2) = \emptyset$.
Let $c_2^{1\prime} \stackrel{def}{=} 211 = 3$ *and* $c_2^{2\prime} \stackrel{def}{=} 11 = 4$, *then* $Q(c_2^{1\prime}, c_2) = \emptyset$ *and* $Q(c_2^{2\prime}, c_2) = \{1\}$ *thus* $Q_m(c_2^{2\prime}, c_2) = \{1\}$.
Thus $c_3 \stackrel{def}{=} \overline{c_2}^{c'_2} = c_2 = 111 = 211$ *and* $c'_3 \stackrel{def}{=} \overline{c'_2}^{c_2} = 211 = 3 \wedge 111 = 41$.

$\overline{c_3}^{c'_3} = c_3$, $\overline{c'_3}^{c_3} = c'_3$, *thus* $c \sqcap c' \stackrel{def}{=} 111 = 211 \wedge 211 = 3 \wedge 111 = 41$.

Let us analyze a bit more precisely the complexity: for every pair $p \sim_{c \sqcap c'} q$, there is a position $r \in P(c) \cup P(c')$ such that $p = p_1 \cdot i$ and $r = p_1 \cdot r_1$. Then, if a is the maximal arity of a function symbol, then

$$|P(c \sqcap c')| \leq (|c| + |c'|) \times a$$

where a is the maximal arity of a function symbol and since the number of possible choices for p_1 is bounded by $|c| + |c'|$. It follows that

$$|c \sqcap c'| \leq (|c| + |c'|) \times M(c, c') \times a \leq a \times (|c| + |c'|)^2$$

where $M(c, c')$ is the maximal length of a position in $P(c) \cup P(c')$.

Then we use the following transformation rule:

$$(N_6) \quad f^c(e_1, \ldots, e_n) \cap f^{c'}(e'_1, \ldots, e'_n) \to f^{c \sqcap c'}(e_1 \cap e'_1, \ldots, e_n \cap e'_n)$$

if $\Pi(f^c(\vec{e})) = \Pi(f^{c'}(\vec{e'}))$.

First, $P(c), P(c') \subseteq \Pi(f(\vec{e}))$, thanks to property 3. It follows that $P(c \sqcap c') \subseteq \Pi(f(\vec{e}))$. The semantics is also preserved since, as long as all positions $p \cdot i$ and $q \cdot i$ are in $\Pi(f^c(\vec{e}))$, and the top symbols at positions p and q are identical, an equality test $p = q \in c$ is equivalent to the conjunction of equality tests $p \cdot i = q \cdot i$.

Condition 3 is also trivially satisfied. Remains to verify the preservation of the other ones: condition 4 is shown to be preserved in lemma 80, condition 1 in lemma 81, condition 2 in lemma 82.

Lemma 80 *The rule preserves property 4.*

Proof: Actually, every $\overline{c_0}$ and $\overline{c'_0}$ satisfy property 4. Indeed, if, in the class c_0 of p (in c), q does not share any prefix with p, then in the class of $p \cdot w$ (w.r.t. $\overline{c_0}$), $q \cdot w$ does not share any prefix with $p \cdot w$. □

Lemma 81 *The transformation preserves the basicness condition.*

Proof: It suffices to show that $\overline{c} \wedge \overline{c'}$ satisfies the basicness condition, whenever c, c' do. Then we use an induction on the fixed point computation for $c \sqcap c'$.

Assume $p \cdot i \cdot q \sim_{\overline{c} \wedge \overline{c'}} p'$, $p' \not\geq_{pref} p$, $j \neq i$ and, for every w, $p \cdot j \cdot w \not\sim_{\overline{c} \wedge \overline{c'}} p'$. Then p is not empty. Assume w.l.o.g that $p \cdot i \cdot q, p' \in P(\overline{c_0})$ where c_0 is an equivalence class of c. (If this is not the case, exchange the roles of c and c'). Then, by lemma 80, there is a $p'_1 \sim_{\overline{c_0}} p \cdot i \cdot q$ such that p is not a prefix of p'_1. Hence we may assume w.l.o.g. that p is not a prefix of p' (possibly after replacing p' with some p'_1).

If $p \cdot i \cdot q \sim_c p'$, then the result follows from the basicness property of c: in such a case, we must have $\overline{c_0} = c_0$ (since, in any case, either $\overline{c_0} = c_0$ or $P(\overline{c_0}) \cap P(c_0) = \emptyset$), hence $p \cdot j \cdot w \sim_c p'$ iff $p \cdot j \cdot w \sim_{\overline{c} \wedge \overline{c'}} p'$.

Let us assume now that this is not the case: $\overline{c_0} \neq c_0$, $p \cdot i \cdot q = p_0 \cdot w_0 \cdot i_0$ with $w_0 \cdot i_0 \in Q_m(c_0, c')$ and $p_0 \sim_c q_0$.

$$\begin{cases} p \cdot i \cdot q = p_0 \cdot w_0 \cdot i_0 \\ p' = q_0 \cdot w_0 \cdot i_0 \\ p_0 \sim_c q_0 \sim_c p'_0 \\ p'_0 \cdot p_1 \sim_{c'} q_1 \\ e|_{p'_0 \cdot p_1} \text{ is not basic} \end{cases} \quad \text{and} \quad \begin{cases} w_0 <_{pref} p_1 \\ w_0 \cdot i_0 \not<_{pref} p_1 \end{cases}$$

In addition, we may assume, thanks to property 4 again that p_0 and q_0 on one hand and p'_0 and q_1 on the other hand do not share any non-trivial prefix.

First case: $p <_{pref} p_0$. There is a q' such that $p_0 = p \cdot i \cdot q'$. We use the basicness property of c, considering the equivalence $p \cdot i \cdot q' \sim_c q_0$: either $e|_{p \cdot j}$ is basic or $e|_{p_0}$ is basic or $p \cdot j \cdot w \sim_c p_0$ for some w. In the first two cases, we get what we want ($e|_{p_0}$ basic implies $e_{p \cdot i \cdot q}$ basic). In the last case, $p \cdot j \cdot w \cdot w_0 \cdot i_0 \sim_{\overline{c_0}} p'$ by construction, hence contradicting the hypothesis $\forall w'. p \cdot j \cdot w' \not\sim_{\overline{c} \wedge \overline{c'}} p'$.

Second case: $p_0 \cdot w_0 >_{pref} p \geq_{pref} p_0$. Since $w_0 \cdot i_0 \not<_{pref} p_1$ and $w_0 \leq_{pref} p_1$, $p_1 = w_0 \cdot j_0 \cdot \alpha$ for some $j_0 \neq i_0$ and α. We apply now the basicness property of c', considering the equivalence $p'_0 \cdot w_0 \cdot j_0 \cdot \alpha \sim_{c'} q_1$ (recall that $p'_0 \cdot p_1 \sim_{c'} q_1$). Since $e|_{p'_0 \cdot p_1}$ is not basic, only two cases can occur:

Case 2.1: $e|_{p'_0 \cdot w_0 \cdot i_0}$ **is basic** which implies $e|_{p_0 \cdot w_0 \cdot i_0}$ basic, hence the desired conclusion

Case 2.2: $p'_0 \cdot w_0 \cdot i_0 \cdot w \sim_{c'} q_1 \sim_{c'} p'_0 \cdot p_1$ for some w. Let $p = p_0 \cdot p_2, w_0 = p_2 \cdot i \cdot w_1$. Then $p'_0 \cdot p_2 \cdot i \cdot w_1 \cdot i_0 \cdot w \sim_{c'} q_1$ and, since $i \neq j$, by the basicness of c', either $e_{p'_0 \cdot p_2 \cdot j}$ is basic or $e|_{q_1}$ is basic or else there is a w_2 such that $p'_0 \cdot p_2 \cdot j \cdot w_2 \sim_{c'} q_1$.

In the first case, $e|_{p_0 \cdot p_2 \cdot j}$ is also basic (i.e. $e|_{p \cdot j}$ is basic) and we conclude.

The second case contradicts the hypothesis that $e|_{p'_0 \cdot p_1}$ is not basic.

In the third case, $p'_0 \cdot p_2 \cdot j \cdot w_2 \sim_{c'} p'_0 \cdot p_1 \sim_{c'} p'_0 \cdot w_0 \cdot i_0$. Then, by construction, $p_0 \cdot w_0 \cdot i_0 \sim_{\overline{c} \wedge \overline{c'}} p_0 \cdot p_2 \cdot j \cdot w_2 = p \cdot j \cdot w_2$, which contradicts again the hypothesis.

Third case: $p = p_0 \cdot w_0$**,** $i = i_0$ **and** q **is empty** . $p_1 = w_0 \cdot j_0 \cdot \alpha$ with $j_0 \neq i_0$. If $j = j_0$, from $e|_{p'_0 \cdot p_1}$ is not basic, we conclude the desired result. Otherwise, $j \neq j_0$ and we use the basicness property of c', considering again $p'_0 \cdot w_0 \cdot j_0 \cdot \alpha \sim_{c'} q_1$, $j \neq j_0$. Either $e|_{p'_0 \cdot w_0 \cdot j}$ is basic (in which case we also conclude) or there is w'_1 $p'_0 \cdot p_1 \sim_{c'} p'_0 \cdot w_0 \cdot j \cdot w'_1$.

In the latter case, we use again the basicness property on c', considering $p'_0 \cdot w_0 \cdot j_0 \cdot \alpha \sim_{c'} q_1$, $i \neq j_0$. Either $e|_{p'_0 \cdot w_0 \cdot i_0}$ is basic, in which case we conclude, or else there is a w'_2 such that $p'_0 \cdot p_1 \sim_{c'} p'_0 \cdot w_0 \cdot i_0 \cdot w'_2$. Now, let us recall that, by construction, $\forall w' \in Q(c_0, c')$, $w_0 \cdot i_0 \not<_{pref} w'$. Since $w_0 \cdot i_0 \cdot w'_1 \in Q(c_0, c')$, we must have w'_1 empty.

$e|_{p'_0 \cdot w_0 \cdot i_0}$ is basic (in which case we conclude) or

It follows that $p'_0 \cdot w_0 \cdot i_0 \sim_{c'} p'_0 \cdot w_0 \cdot j \cdot w'_2$, which contradicts $p \cdot i \not\sim_{\overline{c} \wedge \overline{c'}} p \cdot j \cdot w'_2$.

□

Lemma 82 *The transformation* (N_6) *preserves property 2.*

Proof: We prove, by induction on n that

1. for every $p \cdot q, p \in P(c_n) \cup P(c'_n)$ such that q is not empty, either $e|_{p \cdot q}$ is a basic expression or else $c_{n+1} \neq c_n$ or $c'_{n+1} \neq c'_n$.

2. c_n and c'_n (individually) satisfy property 2

The lemma will follow.

For every n, if $p \in P(c_n)$ and if $p \cdot q \in P(c'_n)$ and if $e|_{p \cdot q}$ is not basic, then $q \in Q(\gamma_0, c'_n)$ by definition (γ_0 is the equivalence class of p) and, since q is not empty, a suffix of q belongs to $Q_m(\gamma_0, c'_n)$. It follows that $\overline{\gamma_0}^{c'_n} \neq \gamma_0$ and therefore

$c_{n+1} \neq c_n$. So, from now on we may assume w.l.o.g that both p and $p \cdot q$ are in $P(c_n)$.

In the base case, $c_n = c$ and $c'_n = c'$ and c, c' satisfy condition 2. Then, if $p, p \cdot q \in P(c)$, $e|_{p \cdot q}$ is basic.

Consider now the induction step and let $p \in P(\overline{\gamma_0}^{c'_n})$, $p \cdot q \in P(\overline{\gamma_1}^{c'_n})$, q is not empty and γ_0, γ_1 are classes of c_n.

Assume first that $\overline{\gamma_0}^{c'_n} \neq \gamma_0$. Then, according to the definition of c_{n+1}, there are positions such that:

$$\begin{cases} p = p_1 \cdot w \cdot i \\ w <_{pref} w' \\ p_2 \sim_{c_n} p_1 \\ p_2 \cdot w' \in P(c'_n) \\ e|_{p_2 \cdot w'} \text{ is not basic} \\ \forall w'' \in Q(\gamma_0, c'_n), w \cdot i \not<_{pref} w'' \end{cases}$$

We consider now several cases for $p \cdot q$.

case 1: $p \cdot q \in P(\overline{\gamma_1}^{c'_n})$, $\overline{\gamma_1}^{c'_n} = \gamma_1 \subseteq c_n$. Then $p_1 <_{pref} p \cdot q$ and we conclude using property 2 on c_n (Induction hypothesis).

case 2: $p \cdot q \in P(\overline{\gamma_1})$, $\overline{\gamma_1} \neq \gamma_1 \subseteq c_n$. Then, by definition, there are positions such that:

$$\begin{cases} p \cdot q = p_3 \cdot w_1 \cdot i_1 \\ w_1 <_{pref} w'_1 \\ p_4 \sim_{c_n} p_3 \\ p_4 \cdot w'_1 \in P(c'_n) \\ e|_{p_4 \cdot w'_1} \text{ is not basic} \\ \forall w''_1 \in Q(\gamma_1, c'_n), w_1 \cdot i_1 \not<_{pref} w''_1 \end{cases}$$

In this case, $p \cdot q = p_3 \cdot w_1 \cdot i_1 = p_1 \cdot w \cdot i \cdot q$, hence p_3 and p_1 must be comparable w.r.t. the prefix ordering. If they are distinct, assume p_k is the largest one, then, by property 2 on c_n, $e|_{p_k}$ must be basic, hence $e|_p$ or $e|_{p \cdot q}$ must be basic and we conclude. Otherwise, $p_1 = p_3$ and $\gamma_0 = \gamma_1$. By hypothesis, $\forall w'' \in Q(\gamma_0, c'_n), w \cdot i \not<_{pref} w''$, thus $w'' = w \cdot i \cdot q \notin Q(\gamma_0, c'_n)$, which can only occur when $e|_{p_1 \cdot w \cdot i \cdot q} = e|_{p \cdot q}$ is basic, and we conclude again.

Assume now $\overline{\gamma_0}^{c'_n} = \gamma_0$. If $\overline{\gamma_1}^{c'_n} = \gamma_1$, we conclude by the induction hypothesis: property 2 holds on c_n.

Otherwise, there are positions such that:

$$\begin{cases} p \cdot q = p_1 \cdot w_1 \cdot i_1 \\ w_1 <_{pref} w_1' \\ p_2 \sim_{c_n} p_1 \\ p_2 \cdot w_1' \in P(c_n') \\ e|_{p_2 \cdot w_1'} \text{ is not basic} \\ \forall w_1'' \in Q(\gamma_1, c_n'), w_1 \cdot i_1 \not<_{pref} w_1'' \end{cases}$$

$\overline{\gamma_0}^{c_n'} = \gamma_0$ implies $p_1 \not<_{pref} p$. If $p_1 >_{pref} p$, from $p_1 \sim_{c_n} p_2$, we conclude that $e|_{p_1}$ is basic (hence $e|_{p \cdot q}$), thanks to the induction hypothesis.

We are left to the case $p = p_1$. Then $p_2 \sim_{c_n} p$ and $p_2 \cdot w_1' \in P(c_n')$ and w_1' is not empty, hence $\overline{\gamma_0}^{c_n'} \neq \gamma_0$, a contradiction.

□

Lemma 83 *Property 5 is satisfied for normal forms w.r.t.* (N_6).

Proof: At step 1, we ensured that, for every $p \in \Pi(e)$, $e|_p$ only contains expressions of the form $f^{c_1}(\vec{e}) \cap \ldots \cap f^{c_n}(\vec{e}) \cap X_1 \cap \ldots \cap X_m$. The rule (N_6) imposes $n = 1$, hence property 5. □

In this step, we did not increase $|e|_{\mathcal{F}}$ and $|c \sqcap c'| \leq a \times (|c| + |c'|)^2$, hence $|e'|_t \leq a \times |e|_t^2$.

Step 4 . The purpose of this last step is to rearrange the equality tests so that there is no overlapping tests except possibly for basic expressions. (More formally, we need to ensure condition 6).

Let us show first some examples of what we want.

Example 84 $e \stackrel{def}{=} f^{11=12=2}(f^{11=12=21=22}(f(X,X), f(X,X)), f(X,X))$. *$e$ contains overlapping tests. We can however use first the rules* $(N_4), (N2)$ *and get*

$$f^{11=12=2}(f^{11=12=21=22}(f^{1=2}(X,X), f^{1=2}(X,X)), f(X,X))$$

Now, it turns out that the intermediate test is a consequence of the top one and the lowest ones, and it can be removed, yielding (after normalization w.r.t. $(N2)$*) :*

$$f^{11=12=2}(f(f^{1=2}(X,X), f^{1=2}(X,X)), f^{1=2}(X,X))$$

for which there is no overlapping test.
In this example, pictured in figure 19, we see that we do not need to change the tests but only to reorganize them.

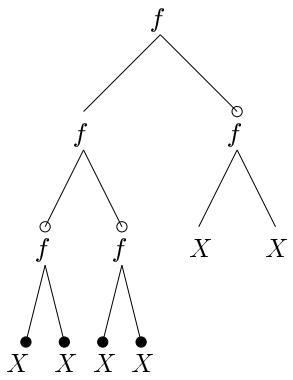


Figure 19: the tests of example 84

Example 85 *Let*

$$e = f^{111=121=112=122=2}(f^{1=2}(g(X,X,Y),g(X,X,Y)),X),$$

which contains overlapping tests. Using the rule (N_4) *we get*

$$f^{c}(f^{1=2}(g(X,X,Y),g(X,X,Y))\cap f^{11=12=21=22}(g^{1=2}(X,X,Y),g^{1=2}(X,X,Y)),X)$$

with $c \stackrel{def}{=} 111 = 121 = 112 = 122 = 2$*. Using rule* (N_6) *we get*

$$f^{111=121=112=122=2}(f^{11=12=21=22\wedge 13=23}(g^{1=2}(X,X,Y),g^{1=2}(X,X,Y)),X)$$

Now, the class $11 = 12 = 21 = 22$ *is a consequence of the top and low tests and it can be removed:*

$$f^{111=121=112=122=2}(f^{13=23}(g^{1=2}(X,X,Y),g^{1=2}(X,X,Y)),X)$$

Finally the low tests can also be removed since they are consequences of the top one, yielding:

$$f^{111=121=112=122=2}(f^{13=23}(g(X,X,Y),g(X,X,Y)),X)$$

in which there is a remaining overlapping test. However, in e*,* Y *must be basic (thanks to the basicness condition) and thus the lower positions* 13 *and* 23 *correspond to basic expressions.*
In this example, pictured in figure 20, we need to push some tests down.

So, the idea is to first inherit the constraints thanks to rule (N_4) (this has been done at step 2), next normalize w.r.t. (N_6) (this has been done at step 3) and finally remove useless tests, which we do now.

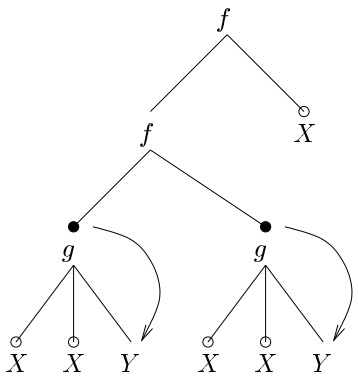


Figure 20: the tests of example 85

$$(N_7)\quad f^{c_1}(\vec{e_1})[g^{c_2\wedge c_0}(\vec{e_2})\cap e'_2]_{p_1} \to f^{c_1}(\vec{e_1})[g^{c_2}(\vec{e_2})\cap e'_2]_{p_1}$$

If

- c_0 is an equivalence class in $c_2 \wedge c_0$
- $c_1 \wedge p_1 \cdot \bigwedge_{\substack{r\in\Pi(g^{c_2\wedge c_0}(\vec{e_2}))\\ r\neq\epsilon \\ g^{c_2\wedge c_0}(\vec{e_2})|_r = h^{c_r}(\vec{e_r})\cap e'_r}} r\cdot c_r \models_e c_0$
- $e = f^{c_1}(\vec{e_1})[g^{c_2\wedge c_0}(\vec{e_2})\cap e'_2]_{p_1}$
- $\models_e$ is the consequence relation according to the following rules:
  - reflexivity, symmetry and transitivity
  - right compatibility: $p = q \models_e p\cdot r = q\cdot r$
  - folding (w.r.t. e): if $e|_p = f^c(\vec{e'})\cap e''$ and f has arity n, then $p\cdot 1 = q\cdot 1 \wedge \ldots \wedge p\cdot n = q\cdot n \models_e p = q$.
  - conjunction introduction: $\left.\begin{array}{l} c_1 \models_e c'_1 \\ c_2 \models_e c'_2 \end{array}\right\} \Rightarrow c_1\wedge c_2 \models_e c'_1 \wedge c'_2$.

We must be careful on how to apply this rule. Consider the following example

Example 86 $e \stackrel{def}{=} f^{111=12=2}(f^{11=12=2}(f^{1=2}(g^\top(X), g^\top(X)), g^\top(X)), g^\top(X))$. *This expression is in normal form w.r.t. the previous transformations. There are two ways of applying rule* (N_7)*: we can remove the constraint* $1 = 2$ *since* $11 = 12 = 2 \wedge 11\cdot\top \wedge 12\cdot\top \models 1\cdot(1 = 2)$. *Then the expression is in normal form for* N_7 *and there are still some overlapping tests. The other possibility*

is to apply (N_7) *to* $11 = 12 = 2$*:* $111 = 12 = 2 \wedge 1 \cdot 1 \cdot (1 = 2) \models 1 \cdot (11 = 12 = 2)$ *and there is no longer any overlapping tests in the expression.*

We assume that the previous steps have been completed and use the rule (N_7) top-down.

Lemma 87 (N_7) *(applied top-down) is terminating, it preserves the semantics and the properties 5, 3, 1, 4, 2.*

Proof: The termination is straightforward: the size of the expression is strictly decreasing (and the resulting expression e' satisfies $|e'|_{\mathcal{F}} = |e|_{\mathcal{F}}$ and $|e'|_t \leq |e|_t$.)

The condition of the rule ensures the preservation of interpretations. Property 5 is preserved since we do not change the term structure of the expression. Property 3 is not necessarily preserved by one-step application of (N_7). However, if $p_0 \cdot p_1 = p_0 \cdot p'_1$ is checked higher up in the expression, then the expressions at positions p_1 and p'_1 must be identical (by property 3) and the rule (N_7) will be applied twice to these expressions, yielding removal of c_0 for both occurrences.

Lemma 77 ensures the preservation of property 1. Also, property 4 is preserved since we remove an equivalence class and property 2 is preserved since we remove some tests. □

Lemma 88 *Normal forms w.r.t. rule* (N_7) *(applied top-down) satisfy condition 6.*

Proof: Assume that we are in the situation of property 6: $e = f^c(e_1, \ldots, e_n) \cap e'$, $p_1 \cdot p_2 \sim_c q$, p_1, p_2 are non-empty, $f^c(e_1, \ldots, e_n)|_{p_1} = g^{c'}(e'_1, \ldots, e'_m) \cap e''$ and $p' \in P(c')$.

Assume that there is no position p'_2 such that $p_1 \cdot p'_2 \sim_c p_1 \cdot p_2$ and p'_2 is comparable with p' w.r.t. $\geq_{pref}$. Let $p_1 \cdot p'_2 \sim_c p_1 \cdot p_2$ such that p'_2 is the position which shares the longest prefix with p' . Then we can write $p' = w \cdot j \cdot w'$ and $p'_2 = w \cdot i \cdot w''$ with $i \neq j$. By condition 1 (for c), considering $p_1 \cdot w \cdot i \cdot w'' \sim_c p_1 \cdot p_2$, either $e|_{p_1 \cdot p_2}$ is basic or $e|_{p_1 \cdot p'}$ is basic or $\exists w_1, p_1 \cdot p_2 \sim_c p_1 \cdot w \cdot j \cdot w_1$. This last case contradicts the maximal shared prefix hypothesis.

We are left to the case where p' is comparable w.r.t. $\geq_{pref}$ with some position p'_2 such that $p_1 \cdot p'_2 \sim_c p_1 \cdot p_2$.

If $p' \sim_{c'} q'$ and q' does not share any non-trivial prefix with p' (this is possible thanks to property 4 on c'), then a similar property holds for q': we assume now that $p' \sim_{c'} q'$ and there are positions p'_2 and p''_2 such that $p_1 \cdot p_2 \sim_c p_1 \cdot p'_2 \sim_c p_1 \cdot p''_2$ and p' and p'_2 on one hand and q' and p''_2 on the other hand are comparable w.r.t. the prefix ordering. p'_2 must be distinct from p''_2 since p' and q' do not share any prefix.

By rules $(N_4), (N_6)$, $c' = c_1 \sqcap \ldots \sqcap c_n$ and $c_1 \models p'_2 = p''_2$. Now, we consider a number of cases, depending on the comparisons between p', q', p'_2, q'_2:

Case 1: $p' <_{pref} p'_2$.

In this case, by definition of $\sqcap$, $e|_{p_1 \cdot p'_2}$ must be basic: c' has to contain a suffix of p'_2 and we use property 2 on c'. It follows that $e|_{p_1 \cdot p_2}$ is basic.

Case 2: $q' <_{pref} p''_2$. This is similar to the first case.

Case 3: $p' \geq_{pref} p'_2$ **and** $q' \geq_{pref} p''_2$.

Let $p' = p'_2 \cdot q'_2$. By definition of $\sqcap$, $p' \sim_{c'} p''_2 \cdot q'_2$. Then q' must be equal to $p''_2 \cdot q'_2$ and $c \models p_1 \cdot (p' = q')$. Next, equalities $p' \sim_{c'} q''$, in which p' and q'' share a non-trivial prefix must be consequences (w.r.t. $\models_e$) of equality tests on subexpressions: this is true for normal forms w.r.t. (N_4) and this is an invariant of (N_6) since $c \sqcap c' \models_e c \wedge c'$.

Then, rule N_7 can be applied (contradiction).

□

D SET-constraints and Automata with one memory

We consider a satisfiable SET-constraint S and we assume that A_S is constructed as described in section 5.3.

We can first note that if $e = f^c(e_1, \ldots, e_n)$ occurs in S and if p is a non-root position of e, then $c \downarrow p$ has only one non-trivial equivalence class. This is ensured by conditions 1, 4 and 2 of normal expressions (see definition 28.

We prove by induction on the size of t that, if α is the solution of $solved(S)$, then for every t accepted in q_X, $H(t) \in \alpha(X)$.

For sake of simplicity, we will say shortly that "t is accepted in $< q, m >$" instead of "there is a computation of the automaton on t yielding the configuration $< q, m >$.

Lemma 89

- *if t is accepted in $< q_{e,p}, \tau >$, then $H(t) \in \alpha(e|_p)$, $H(t) \models c \downarrow p$ and $\tau = H(t)|_{p \cdot p'}$ for some p' tested by $c \downarrow p$ (since $H(t) \models c \downarrow p$ and since $c \downarrow p$ has only one non-trivial equivalence class, for every p', p'' tested by $c \downarrow p$, we have $H(t)|_{p \cdot p'} = H(t)|_{p \cdot p''}$).*
- *if t is accepted in $< q_e, \tau >$, then $H(t) \in \alpha(e)$ and $\tau = a$.*
- *if t is accepted in $< q_X, \tau >$, then $H(t) \in \alpha(X)$ and $\tau = H(t)$.*

- *if t is accepted in $< q_\phi, \tau >$, then $\alpha \models \phi$.*
- *if t is accepted in $< q_a, \tau >$, then $t = a$ and $\tau = a$.*

Proof: (sketch) If $|t| = 1$, then $t = b$ for some constant in $\mathcal{F}$ and the only transition leading to b is $b \xrightarrow[b]{\top} q_b$.

Assume Lemma 89 is satisfied for every t of size $\leq n$ and consider t such that $|t| = n + 1$.
If t is accepted in $< q_{e,p}, \tau >$, then $t = g(t_1, \ldots, t_k)$ such that t_i is accepted in $< q_{e,p\cdot i}, \tau_i >$. By induction hypothesis, $H(t_i) \in \alpha(e|_{p\cdot i})$, $H(t_i) \models c \downarrow p \cdot i$ and $\tau = H(t_i)|_{p\cdot i\cdot p'}$ for some p' tested by $c \downarrow p \cdot i$. $H(t) = g(H(t_1), \ldots, H(t_n))$, thus $H(t) \in \alpha(e|_p)$. We have also that $H(t) \models \widetilde{c \downarrow p}$, thus $H(t) \models c \downarrow p$. Finally, $\tau = \tau_i$ for some i such that $p \cdot i$ is a position checked by c, thus $\tau = H(t)|_{p\cdot p'}$ for some p' tested by $c \downarrow p$.
The other cases are proved similarly. □

Conversely, if $t \in \alpha(X)$ where α is the minimal solution of $solved(S)$, then there exists t' such that $H(t) = t'$ and t' is accepted in state q_X.

Lemma 90

1. *if $t \in \alpha(X)$, then there exists t' such that $H(t') = t$ and t' is accepted in $< q_X, t' >$.*
2. *if $t \in \alpha(e|_p)$ and $t \models c \downarrow p$, then there exists t' such that $H(t') = t$ and t' is accepted in $< q_{e,p}, t|_{p\cdot p'} >$ for some p' tested by $c \downarrow p$.*
3. *if $t \in \alpha(e)$, then there exists t' such that $H(t') = t$ and t' is accepted in $< q_e, a >$.*

Proof: (sketch) We prove that if $T^n_{solved(S)}(\emptyset)$ satisfies the properties of lemma 90, then $T^{n+1}_{solved(S)}(\emptyset)$ also satisfies the properties of lemma 90. The result follows by minimality of α.
Assume $T^n_{solved(S)}(\emptyset)$ satisfies the properties of lemma 90. First, we can verify that if $T^n_{solved(S)}(\emptyset) \models \phi$ then there exists t' such that t' is accepted in state q_ϕ.
Assume now that $T^{n+1}_{solved(S)}(\emptyset)$ satisfies property 1, then, by well-founded induction on $\geq$ (the reverse prefix order), we show that $T^{n+1}_{solved(S)}(\emptyset)$ satisfies property 2 and 3.
Thus, it is sufficient to prove that $T^{n+1}_{solved(S)}(\emptyset)$ satisfies property 1: assume $t \in [\![X]\!]_n$, then there exists a clause $\phi \Rightarrow e \subseteq X$ such that $T^n_{solved(S)}(\emptyset) \models \phi$ and $t \in [\![e]\!]_n$. Applying the induction hypothesis and the rules of the automaton, we deduce that there exists t' such that $H(t') = t$ and t' is accepted in $< q_X, t' >$.

□

E Protocols and Horn Clauses

Let P be a protocol. We assume $\mathcal{H}_{Msg}, \mathcal{H}_I$ and $\mathcal{H}_P$ constructed as described in Section 6.2.

We first construct $\mathcal{H}_{I_0}$ such that the maximal initial knowledge of the intruder I_0 is a minimal interpretation of the predicate I_0 which satisfies $\mathcal{H}_{I_0}$.

Lemma 59 *Let $t_1, \ldots, t_n$ be message schemes with the free variables $x_1, \ldots, x_k$. Then, there exists a set of Horn clauses $\mathcal{H}_{I_0}$ such that $I_0(m)$ is derivable from $\mathcal{H}_{I_0} \cup \mathcal{H}_{\mathtt{msg}}$ if and only if $\mathsf{parts}(m) \cap \{t_i\sigma \mid 1 \leq i \leq n, \sigma(x_i) \in \mathcal{A}_h\} = \emptyset$.*

Proof: Let p be the maximal depth of the terms $t_1, \ldots, t_n$. Let $S = \{t_i\sigma \mid 1 \leq i \leq n, \sigma(x_i) \in \mathcal{A}_h\}$. We introduce a new predicate $P_{\geq p}$ such $P_{\geq p}$ accepts the terms of depth larger or equal to p. The clauses $\mathcal{H}_{I_0}$ for $P_{\geq p}$ and I_0 are described figure 21. $|m|_d$ denotes the depth of the term m. Let $\mathcal{H}' = \mathcal{H}_{Msg} \cup \mathcal{H}_{I_0}$.
By construction, $\mathcal{H}' \vdash P_{\geq p}(m)$ if and only if the depth of m is greater or equal to p.
Let us show by induction on the number of rules which have been applied that if $\mathcal{H}' \vdash I_0(m)$ then $\mathsf{parts}(m) \cap S = \emptyset$. Let m be a term such that $\mathcal{H}' \vdash I_0(m)$ and let us consider the last rule which has been applied:

rule (1): $\mathsf{parts}(m) \cap S = \emptyset$ by definition of rule (1).

rule (4): $m = f(m_1, \ldots, m_n$ such that $f \in \mathcal{IF}$, $\mathcal{H}' \vdash I_0(m_1), \ldots, I_0(m_n)$ and there exists i such that $\mathcal{H}' \vdash P_{\geq p}(m_i)$.

$$\mathsf{parts}(m) = \{m\} \cup \bigcup_{1 \leq i \leq n} \mathsf{parts}(m_i).$$

By induction hypothesis, $\mathsf{parts}(m_i) \cap S = \emptyset$. In addition, there exists i such that $|m_i|_d \geq p$, thus $|m|_d > p$ which implies that $m \notin S$. Thus $\mathsf{parts}(m) \cap S = \emptyset$.

rule (5): $m = f(m_1, \ldots, m_n$ such that $f \in \mathcal{OF}$ and there exists i such that $\mathcal{H}' \vdash P_{\geq p}(m_i)$, thus $|m|_d > p$. Since $\mathsf{parts}(m) = \{m\}$ and $|m|_d > p$, we have $\mathsf{parts}(m) \cap S = \emptyset$.

rule (6): this case is similar to the previous ones.

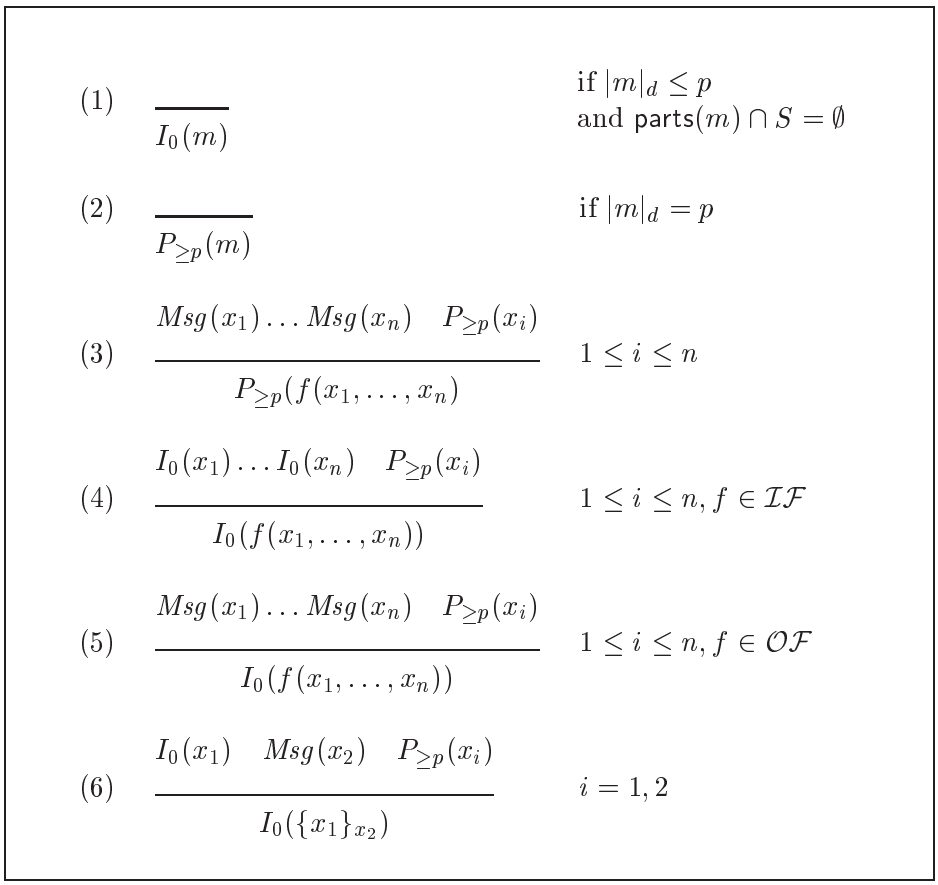


Figure 21: Horn clauses for $P_{\geq p}$ and I_0

Conversely, an induction on the depth of m proves that if $\mathsf{parts}(m) \cap S = \emptyset$, then $\mathcal{H}' \vdash I_0(m)$. □

We prove here Lemma 58 by proving the following stronger lemma.

Lemma 91 *Let m be a message and a an agent, $\mathcal{C} \vdash^\star I(\overline{m})$ iff there exists a reachable H such that $[m] \in H$ and*
$\mathcal{C} \vdash^\star I(s_i(\overline{a}, \overline{m}))$ iff there exists a reachable H such that $S(a, i, m) \in H$.

To prove this, we need few lemmas:

Lemma 92 *If there exists a reachable H_1 such that $m_1 \in H_1$ where m_1 is a message and if there exists a reachable H_2 such that $e_2 \in H_2$ where e_2 is either a message or a state, then there exists a reachable H such that $m_1, e_2 \in H$.*

A transition t of a protocol is applicable in H provided $Pre(t) \subseteq H$. Thus, if t is applicable in H, then t is applicable in H', for all $H' \supseteq H$. In the same way, if $X \in \mathsf{fake}(Cont(H) \cup I)$ then $X \in \mathsf{fake}(Cont(H') \cup I)$, for all $H' \supseteq H$. Therefore, let H be the global state obtained from H_1 by applying all the transitions used to obtain H_2. e_2 is in H and m_1 is still in H since the transitions do not remove any message.

Lemma 93 *Let S a set of messages such that $\forall m \in S \quad \mathcal{C} \vdash^\star \overline{m}$. Then $\forall m \in \mathsf{fake}(S), \quad \mathcal{C} \vdash^\star \overline{m}$.*

Pairing, unpairing, encryption and decryption are simulated by the clauses in Figure 10.

For Lemma 58, we first prove by induction on n that if $\mathcal{C} \vdash^n I(\overline{m})$ then there exists a reachable H such that $[m] \in H$ and if $\mathcal{C} \vdash^n I(s_i(\overline{a}, \overline{m}))$ then there exists a reachable H such that $S(a, i, m) \in H$.

For $n = 0$, it is true,

Assume the hypothesis is verified for n, and assume $\mathcal{C} \vdash^{n+1} I(\overline{m})$. The last deduction rule is either one of those presented Figure 10, in this case, by inspection of the deduction rules, using the induction hypothesis and Lemma 92, we conclude that there exists a reachable H such that $[m] \in H$. Or the last deduction rule is one of those presented Figure 11. Then,

$$\mathcal{C} \vdash^n I(s_i(\overline{a}, \overline{m_0})), Msg(\overline{m_0}), I(\overline{m_1}), Msg(\overline{m_1}), \ldots, I(\overline{m_p}), Msg(\overline{m_p})$$

where $\overline{m_i} = \overline{M_i}\sigma'$, $m_0 = \overline{M_0}\sigma'$ and σ' preserves the type : if $\sigma'(x_A) = \overline{t}$ where A is an agent variable, then $\mathcal{C} \vdash^n A(\overline{t})$, thus t is an agent. By application of the induction hypothesis and applying Lemma 92, there exists a reachable H such that $m, m_1, \ldots, m_p \in H$. Thus, the transition

$$t = \{S(A\sigma', i, M\sigma'), M_1\sigma', \ldots, M_p\sigma'\} \longrightarrow \{S(A\sigma', i+1, M'\sigma'), M'_1\sigma', \ldots, M'_q\sigma'\}$$

is applicable in H. Let H' the global state obtained from H by applying t, m is in H'.

Assume $\mathcal{C} \vdash^{n+1} I(s_i(\overline{a}, \overline{m}))$. The only choice for the last deduction rule is one those presented Figure 11. The same reasoning as above allow us to conclude that there exists a reachable H such that $S(a, i, m) \in H$.

Conversely, we prove by induction on n that if there exists a n-reachable H such that $[m] \in H$ or $S(a, i, m) \in H$ then $\mathcal{C} \vdash^\star I(\overline{m})$ or $\mathcal{C} \vdash^\star I(s_i(\overline{a}, \overline{m}))$ where *n-reachable* stands for "reachable with n global transitions".

For $n = 0$, $H = H_0$ and H_0 does not contain any message or state.

Assume the hypothesis is verified for n, and assume there exists a $n+1$-reachable H such that $[m] \in H$. Thus, there exists a n-reachable H_1 such that H is an honest or fake successor of H'. If $[m] \in H'$, we conclude immediately. Assume $[m] \notin H'$:

honest successor Let t the applicable transition such that $H = (H' \backslash (Pre(t) \cap H') \cup Post(t))$. By application of the induction hypothesis and applying the clause described in Figure 11, we conclude $\mathcal{C} \vdash^\star I(\overline{m})$.

fake successor If $H = H' \cup \{m\}$ where H' is n-reachable and $m \in \mathsf{fake}(Cont(H) \cup I)$. Lemma 93 and the induction hypothesis allows us to conclude.