Symbolic verification of distance bounding protocols *

Alexandre Debant and Stéphanie Delaune

Univ Rennes, CNRS, IRISA, France
{alexandre.debant,stephanie.delaune}@irisa.fr

Abstract. With the proliferation of contactless applications, obtaining reliable information about distance is becoming an important security goal, and specific protocols have been designed for that purpose. These protocols typically measure the round trip time of messages and use this information to infer a distance. Formal methods have proved their usefulness when analysing standard security protocols such as confidentiality or authentication protocols. However, due to their abstract communication model, existing results and tools do not apply to distance bounding protocols.
In this paper, we consider a symbolic model suitable to analyse distance bounding protocols, and we propose a new procedure for analysing (a bounded number of sessions of) protocols in this model. The procedure has been integrated in the Akiss tool and tested on various distance bounding and payment protocols (e.g. MasterCard, NXP).

1 Introduction

In recent years, contactless communications have become ubiquitous. They are used in various applications such as access control cards, keyless car entry systems, payments, and many other applications which often require some form of authentication, and rely for this on security protocols. In addition, contactless systems aims to prevent against *relay attacks* in which an adversary mount an attack by simply forwarding messages he receives: ensuring physical proximity is a new security concern for all these applications.

Formal modelling and analysis techniques are well-adapted for verifying security protocols, and nowadays several verification tools exist, e.g. ProVerif [8], Tamarin [28]. They aim at discovering logical attacks, and therefore consider a symbolic model in which cryptographic primitives are abstracted by function symbols. Since its beginning in 80s, a lot of progress has been done in this area, and it is now a common good practice to formally analyse protocols using symbolic techniques in order to spot flaws possibly before their deployment, as it was recently done e.g. in TLS 1.3 [17, 7], or for an avionic protocol [9].

* This work has been partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 714955-POPSTAR).

These symbolic techniques are based on the so-called Dolev Yao model [20]. In such a model, the attacker is supposed to control the entire network. He can send any message he is able to build using his current knowledge, and this message will reach its final destination instantaneously. This model is accurate enough to analyse many security protocols, e.g. authentication protocols, e-voting protocols, …However, to analyse protocols that aim to prevent against relay attacks, some features need to be modelled in a more faithful way. Among them:

- *network topology*: any pair of nodes can communicate but depending on their distance, exchanging messages take more or less time. We will simply assume that the time needed is proportional to the distance between the two agents, and that messages can not travel faster than the speed of the light.
- *timing constraints*: protocols that aim to prevent against relay attacks typically rely on a rapid phase in which time measurements are performed. Our framework will allow us to model these time measurements through the use of timestamps put on each action.

There are some implications on the attacker model. Since communications take time, it may be interesting to consider several malicious nodes. We will assume that malicious nodes collaborate but again messages can not travel (even between malicious nodes) faster than the speed of the light.

Akiss in a nutshell. The procedure we present in this paper builds on previous work by Chadha et al. [12], and its implementation in the tool Akiss. Akiss allows automated analysis of privacy-type properties (modelling as equivalences) when restricted to a bounded number of sessions. Cryptographic primitives may be defined through arbitrary convergent equational theories that have the finite variant property. This class includes standard cryptographic primitives as well as less commonly supported primitives such as blind signatures and zero knowledge proofs. Termination of the procedure is guaranteed for subterm convergent theories, but also achieved in practice on several examples outside this class.

The procedure behind Akiss is based on an abstract modelling of symbolic traces into first-order Horn clauses: each symbolic trace is translated into a set of Horn clauses called *seed statements*, and a dedicated resolution procedure is applied on this set to construct a set of statements which have a simple form: the so-called *solved statements*. Once the saturation of the set of seed statements is done, it is possible to decide, based solely on those solved statements, whether processes under study are equivalent or not.

Even if we are considering reachability properties (here authentication with physical proximity), in order to satisfy timing constraints, we may need to consider recipes that are discarded when performing a classical reachability analysis. Typically, in a classical reachability analysis, there is no need to consider two recipes that deduce the same message. The main advantage of Akiss is the fact that, since its original goal is to deal with equivalence, it considers more (actually almost all possible) recipes when performing the security analysis. Moreover, even if the tool has been designed to deal with equivalence-based properties, the

first part of the Akiss procedure consists in computing a knowledge base which is in fact a finite representation of all possible traces (including recipes) executable by the process under study. We build on this saturation procedure in this work.

Our contributions. We design a new procedure for verifying reachability properties for protocols written in a calculus sharing many similarities with the one introduced in [19], and that gives us a way to model faithfully distance bounding protocols. Our procedure follows the general structure of the original one described in [12]. We first model protocols as traces (see Section 3), and then translate them into Horn clauses (see Section 4). A direct generalisation would consist of keeping the saturation procedure unchanged, and simply modifying the algorithm to check the satisfiability of our additional timing constraints at the end. However, as discussed in Section 5, such a procedure would *not* be complete for our purposes. We therefore completely redesign the update function used during the saturation procedure using a new strategy to forbid certain steps that would otherwise systematically yield to non-termination in our final algorithm. Showing these statements are indeed unnecessary requires essential changes in the proofs of completeness of the original procedure.

This new saturation procedure yields an effective method for checking reachability properties in our calculus (see Section 6). Although termination of saturation is not guaranteed in theory, we have implemented our procedure and we have demonstrated its effectiveness on various examples. We report on our implementation and the various case studies we have performed in Section 7.

As we were unable to formally establish completeness of the procedure as implemented in the original Akiss tool (due to some mismatches between the procedure described in [12] and its implementation), we decided to bring the theory closer to the practice, and this explains several differences between our seed statements and those described originally in [12].

2 Background

We start by providing some background regarding distance bounding protocols. For illustrative purposes, we present a slightly simplified version of the TREAD protocol [2] together with the attack discovered by [26] (relying on the Tamarin prover). This protocol will be used along the paper as a running example.

2.1 Distance bounding protocols

Distance bounding protocols are cryptographic protocols that enable a verifier V to establish an upper bound on the physical distance to a prover P. They are typically based on timing the delay between sending out a challenge and receiving back the corresponding response. The first distance bounding protocol was proposed by Brands and Chaum [10], and since then various protocols have been proposed. In general, distance bounding protocols are made of two or three phases, the second one being a rapid phase during which the time measurement is

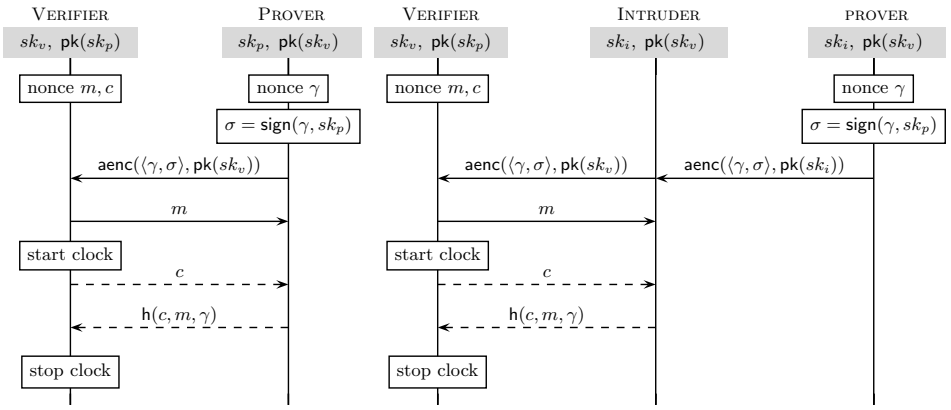


Fig. 1: TREAD protocol (left) and a mafia fraud attack (right)

performed. To improve accuracy, this challenge/response exchange during which the measurement is performed is repeated several times, and often performed at the bit level. Symbolic analysis does not allow us to reason at this level, and thus the rapid phase will be abstracted by a single challenge/response exchange, and operations done at bit level will be abstracted too.

For illustration purposes, we consider the TREAD protocol. As explained before, we ignore several details that are irrelevant to our symbolic security analysis, and we obtain the protocol described in Figure 1. First, the prover generates a nonce γ, and computes the signature σ with his own key. This signature is sent to V encrypted with the public key of V. Upon reception, the verifier decrypts the message and checks the signature. Then, the verifier sends a nonce m, and starts the rapid phase during which he sends a challenge c to the prover. The protocol ends successfully if the answer given by the prover is correct and arrived before a predefined threshold.

2.2 Attacks on distance bounding protocols

Typically, an attack occurs when a verifier is deceived into believing it is co-located with a given prover whereas it is not. Attacker may replay, relay and build new messages, as well as predict some timed challenges. Since the introduction of distance bounding protocols, various kinds of attacks have emerged, e.g. distance fraud, mafia fraud, distance hijacking attack, …For instance, a distance fraud only consider a dishonest prover who tries to authenticate remotely, whereas a distance hijacking scenario allows the dishonest prover to take advantage of honest agents in the neighbourhood of the verifier.

The TREAD protocol is vulnerable to a mafia fraud attack: an honest verifier v may end successfully a session with an honest prover p thinking that this prover p is in his vicinity whereas p is actually far away. The attack is described

in Figure 1. After learning γ and a signature $\sigma = \mathsf{sign}(\gamma, sk_p)$, the malicious agent i will be able to impersonate p. At the end, the verifier v will finish his session correctly thinking that he is playing with p (who is actually far away).

2.3 Symbolic security analysis

The first symbolic framework developed to analyse distance bounding protocols is probably the one proposed in [27]. Since then, several formal symbolic models have been proposed: *e.g.* a model based on multiset rewriting rules has been proposed in [5], another one based on strand spaces is available in [31]. However, these models do not come with a procedure allowing one to analyse distance bounding protocols in an automatic way. Recently, some attempts have been done to rely on existing automatic verification tools, e.g. ProVerif [19, 13] or Tamarin [26]. Those tools typically consider an unbounded number of sessions, and some approximations are therefore performed to tackle this problem well-known to be undecidable [21].

Here, following the long line of research on symbolic verification for a bounded number of sessions which is a problem well-known to be decidable [32, 29] and for which automatic verification tools have been developed (e.g. OFMC [6], Akiss [12]), we aim to extend this approach to distance bounding protocols.

3 A security model dealing with time and location

We assume that our cryptographic protocols are modelled using a simple process calculus sharing some similarities with the applied-pi calculus [1], and strongly inspired by the calculus introduced in [19].

3.1 Term algebra

As usual in symbolic models, we represent messages using a term algebra. We consider a set $\mathcal{N}$ of *names* split into two disjoint sets: the set $\mathcal{N}_{\mathsf{pub}}$ of *public names* which contains the set $\mathcal{A}$ of agent names, and the set $\mathcal{N}_{\mathsf{priv}}$ of *private names*. We consider the set $\mathcal{X}$ of *message variables*, denoted $x, y, \ldots$, as well as a set $\mathcal{W}$ of *handles*: $\mathcal{W} = \{\mathsf{w}_1, \mathsf{w}_2, \ldots\}$. Variables in $\mathcal{X}$ model arbitrary data expected by the protocol, while variables in $\mathcal{W}$ are used to store messages learnt by the attacker. Given a *signature* Σ, i.e. a finite set of function symbols together with their arity, and a set of atomic data $\mathcal{A}t$, we denote $\mathcal{T}(\Sigma, \mathcal{A}t)$ the set of terms built from $\mathcal{A}t$ using function symbols in Σ. Given a term u, we denote $st(u)$ the set of the subterms occurring in u, and $vars(u)$ the set of variables occurring in u. A term u is ground when $vars(u) = \emptyset$. Then, we associate an *equational theory* E to the signature Σ which consists of a finite set of equations of the form $u = v$ with $u, v \in \mathcal{T}(\Sigma, \mathcal{X})$, and induces an equivalence relation over terms denoted $=_{\mathsf{E}}$.

Example 1. $\Sigma_{\mathsf{ex}} = \{\mathsf{aenc}, \mathsf{adec}, \mathsf{pk}, \mathsf{sign}, \mathsf{getmsg}, \mathsf{check}, \mathsf{ok}, \langle\ \rangle, \mathsf{proj}_1, \mathsf{proj}_2, \mathsf{h}\}$ allows us to model the cryptographic primitives used in the TREAD protocol

presented in Section 2. The function symbols aenc and adec of arity 2 model asymmetric encryption, whereas sign, getmsg, check, and ok are used to model signature. The term $\mathsf{pk}(sk)$ represents the public key associated to the private key sk. We have function symbols to model pairs and projections, as well as a function h of arity 3 to model hashes. The equational theory E_{ex} associated to the signature Σ_{ex} is the relation induced by:

$$\begin{array}{lll} \mathsf{check}(\mathsf{sign}(x,y),\mathsf{pk}(y)) = \mathsf{ok} & \mathsf{proj}_1(\langle x,y\rangle) = x & \mathsf{adec}(\mathsf{aenc}(x,\mathsf{pk}(y)),y) = x \\ \mathsf{getmsg}(\mathsf{sign}(x,y)) = x & \mathsf{proj}_2(\langle x,y\rangle) = y & \end{array}$$

We consider equational theories that can be represented by a *convergent rewrite system*, i.e. we assume that there is a *confluent* and *terminating* rewrite system such that:

$$u =_{\mathsf{E}} v \;\Leftrightarrow\; u{\downarrow} = v{\downarrow} \text{ for any terms } u \text{ and } v$$

where $t{\downarrow}$ denotes the normal form of t. Moreover, we assume that such a rewrite system has the *finite variant property* as introduced in [16]. This means that given a sequence $t_1, \ldots, t_n$ of terms, it is possible to compute a finite set of substitutions, denoted $\mathsf{variants}(t_1, \ldots, t_n)$, such that for any substitution ω, there exist $\sigma \in \mathsf{variants}(t_1, \ldots, t_n)$ and τ such that: $t_1\omega{\downarrow}, \ldots, t_n\omega{\downarrow} = (t_1\sigma){\downarrow}\tau, \ldots, (t_n\sigma){\downarrow}\tau$. Many equational theories enjoy this property, e.g. symmetric/asymmetric encryptions, signatures and blind signatures, as well as zero-knowledge proofs.

Moreover, this finite variant property implies the existence of a finite and *complete set of unifiers* and gives us a way to compute it effectively. Given a set $\mathcal{U}$ of equations between terms, a *unifier* (modulo a rewrite system $\mathcal{R}$) is a substitution σ such that $s\sigma{\downarrow} = s'\sigma{\downarrow}$ for any equation $s = s'$ in $\mathcal{U}$. A set S of unifiers is said to be *complete* for $\mathcal{U}$ if for any unifier σ, there exists $\theta \in S$ and τ such that $\sigma = \tau \circ \theta$. We denote $\mathsf{csu}_{\mathcal{R}}(\mathcal{U})$ such a set. We will rely on these notions of $\mathsf{variants}$ and csu in our procedure (see Section 4).

Example 2. The finite variant property is satisfied by the rewrite system $\mathcal{R}_{\mathsf{ex}}$ obtained by orienting from left to right equations in E_{ex}.

Let $\mathcal{U} = \{\mathsf{check}(t_\sigma, \mathsf{pk}(sk_p)) = \mathsf{ok}\}$ with $t_\sigma = \mathsf{proj}_2(\mathsf{adec}(x, sk_v))$. We have that $\{\theta\}$ with $\theta = \{x \to \mathsf{aenc}(\langle x_1, \mathsf{sign}(x_2, sk_p)\rangle, \mathsf{pk}(sk_v))\}$ is a complete set of unifiers for $\mathcal{U}$ (modulo $\mathcal{R}_{\mathsf{ex}}$). Now, considering the variants, let $\sigma_1 = \{x \to \mathsf{aenc}(x_1, \mathsf{pk}(sk_v))\}$, $\sigma_2 = \{x \to \mathsf{aenc}(\langle x_1, x_2\rangle, \mathsf{pk}(sk_v))\}$ and id be the identity substitution, we have that $\{id, \sigma_1, \sigma_2\}$ is a finite and complete set of variants (modulo $\mathcal{R}_{\mathsf{ex}}$) for the sequence (x, t_σ).

An attacker builds her own messages by applying function symbols to terms she already knows and which are available through variables in $\mathcal{W}$. Formally, a computation done by the attacker is a *recipe*, i.e. a term in $\mathcal{T}(\Sigma, \mathcal{W} \cup \mathcal{N}_{\mathsf{pub}} \cup \mathbb{R}^+)$.

3.2 Timing constraints

To model time, we will use non-negative real numbers $\mathbb{R}^+$, and we may allow various operations (e.g. $+$, $-$, $\times$, ...). A time expression is constructed inductively

by applying arithmetic symbols to time expressions starting with the initial set $\mathbb{R}^+$ and an infinite set $\mathcal{Z}$ of *time variables*. Then, a timing constraint is typically of the form $t_1 \sim t_2$ with $\sim \in \{<, \leq, =\}$. We do not constraint the operators since our procedure is generic in this respect provided we have a way to decide whether a set of timing constraints is satisfiable or not. In practice, our tool (see Section 7) will only be able to consider simple linear timing constraints.

Example 3. When modelling distance bounding protocols, we will typically consider a timing constraint of the form $z_2 - z_1 < t$ with $z_1, z_2 \in \mathcal{Z}$ and $t \in \mathbb{R}^+$. This constraint expresses that the time elapsed between the emission of a challenge and the receipt of the corresponding answer is at most t.

3.3 Process algebra

We assume that cryptographic protocols are modelled using a simple process algebra. Following [12], we only consider a minimalistic core calculus. In particular, we do not introduce the new operator and we do not explicitly model the parallel operator. Since we only consider a bounded number of sessions (i.e. a calculus with no replication), this is at no loss of expressivity. We can simply assume that fresh names are generated from the beginning and parallel composition can be added as syntactic sugar to denote the set of all interleavings.

Syntax. We model a protocol as a finite set of traces. A *trace* T is a finite sequence (possibly empty and denoted ϵ in this case) of pairs, i.e. $T = (a_1, \mathsf{a}_1).\ldots.(a_n, \mathsf{a}_n)$ where each $a_i \in \mathcal{A}$, and a_i is an action of the form:

$$\mathsf{out}^z(u) \quad \mathsf{in}^z(x) \quad [v = v'] \quad [z := v] \quad [\![t_1 \sim t_2]\!]$$

with $x \in \mathcal{X}$, $u, v, v' \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathbb{R}^+ \cup \mathcal{X})$, $z \in \mathcal{Z}$, and $t_1 \sim t_2$ a timing constraint.

As usual, we have output and input actions. An input action acts as a binding construct for both x and z, whereas an output action acts as a binding construct for z only. For sake of clarity, we will omit the time variable z when we do not care of the precise time at which the input (resp. output) action has been performed. As usual, our calculus allows one to perform some tests on received messages, and it is also possible to extract a timestamp from a received message and perform some tests on this extracted value using timing constraints. Typically, this will allow us to model an agent that will stop executing the protocol in case an answer arrives too late.

We assume the usual definitions of *free* and *bound variables* for traces, and we assume that each variable is at most bound once. Note that, in the constructs presented above, the variables z, x are bound. Given a set $\mathcal{V}$ of variables, a trace is *locally closed w.r.t.* $\mathcal{V}$ if for any agent a, the trace obtained by considering actions executed by agent a does not contain free variables among those in $\mathcal{V}$. Such an assumption, sometimes called origination [15, 6], is always satisfied when considering traces obtained by interleaving actions of a protocol. Therefore, we will only consider traces that are locally closed w.r.t. both $\mathcal{X}$ and $\mathcal{Z}$.

Contrary to the calculus introduced in [19] which assumes that there is at most one timer per thread, we are more flexible. This generalisation is not

mandatory to analyse our case studies but it allows us to present our result on traces and greatly simplifies the theoretical development.

Example 4. Following our syntax, the trace corresponding to the role of the verifier played by v with p is modelled as follows:

$$\begin{aligned} T_{\mathsf{ex}} = {} & (v, \mathsf{in}(x)).\ (v, [\mathsf{check}(t_\sigma, \mathsf{pk}(sk_p)) = \mathsf{ok}]).\ (v, [t_\gamma = \mathsf{getmsg}(t_\sigma)]). \\ & (v, \mathsf{out}(m)). \\ & (v, \mathsf{out}^{z_1}(c)).\ (v, \mathsf{in}^{z_2}(y)).\ (v, [y = \mathsf{h}(c, m, t_\gamma)]).\ (v, [\![z_2 - z_1 < 2 \times t_0]\!]) \end{aligned}$$

where $t_\gamma = \mathsf{proj}_1(\mathsf{adec}(x, sk_v))$, $t_\sigma = \mathsf{proj}_2(\mathsf{adec}(x, sk_v))$, $x, y \in \mathcal{X}$, $z_1, z_2 \in \mathcal{Z}$, $m, c, sk_v, sk_p \in \mathcal{N}_{\mathsf{priv}}$, and $t_0 \in \mathbb{R}^+$ is a fixed threshold.

Of course, when performing a security analysis, other traces have to be considered. Typically, we may want to consider several instances of each role, and we will have to generate traces corresponding to all the possible interleavings of the actions composing these roles.

Semantics. The semantics of a trace is given in terms of a labeled transition system over configurations of the form $(T; \Phi; t)$, and is parametrised by a topology reflecting the fact that interactions between agents depend on their location.

Definition 1. *A topology is a tuple $\mathcal{T}_0 = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0)$ where $\mathcal{A}_0 \subseteq \mathcal{A}$ is the finite set of agents composing the system, $\mathcal{M}_0 \subseteq \mathcal{A}_0$ represents those that are malicious, and $\mathsf{Loc}_0 : \mathcal{A}_0 \to \mathbb{R}^3$ defines the position of each agent in the space.*

In our model, the distance between two agents is given by the time it takes for a message to travel from one to another. We have that:

$$\mathsf{Dist}_{\mathcal{T}_0}(a, b) = \frac{\|\mathsf{Loc}_0(a) - \mathsf{Loc}_0(b)\|}{c_0} \text{ for any } a, b \in \mathcal{A}_0$$

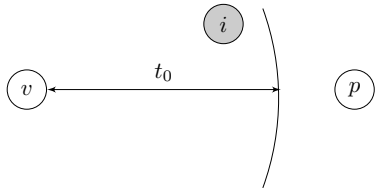
with $\|\cdot\| : \mathbb{R}^3 \to \mathbb{R}$ the Euclidean norm and c_0 the transmission speed. We suppose, from now on, that c_0 is a constant for all agents, and thus an agent a can recover, at time $t + \mathsf{Dist}_{\mathcal{T}_0}(a, b)$, any message emitted by the agent b before $t \in \mathbb{R}^+$.

Definition 2. *Given a topology $\mathcal{T}_0 = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0)$, a* configuration *over $\mathcal{T}_0$ is a tuple $(T; \Phi; t)$ where T is a trace locally closed w.r.t. $\mathcal{X}$ and $\mathcal{Z}$ composed of actions (a, a) with $a \in \mathcal{A}_0$, $t \in \mathbb{R}^+$, and $\Phi = \{\mathsf{w}_1 \xrightarrow{a_1, t_1} u_1, \ldots, \mathsf{w}_n \xrightarrow{a_n, t_n} u_n\}$ is an* extended frame*, i.e. a substitution such that $\mathsf{w}_i \in \mathcal{W}$, $u_i \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathbb{R}^+)$, $a_i \in \mathcal{A}_0$ and $t_i \in \mathbb{R}^+$ for $1 \leq i \leq n$.*

Intuitively, T represents the trace that still remains to be executed; Φ represents the messages that have been outputted so far; and t is the global time.

Example 5. Continuing Example 4, we consider the topology $\mathcal{T}_0 = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0)$ depicted on the right where $\mathcal{A}_0 = \{p, v, i\}$, and $\mathcal{M}_0 = \{i\}$.

The precise location of each agent is not relevant, only the distance between them matters. Here $\mathsf{Dist}_{\mathcal{T}_0}(v,i) < t_0$ whereas $\mathsf{Dist}_{\mathcal{T}_0}(v,p) \geq t_0$.



A possible configuration is $K_0 = (T_{\mathsf{ex}}; \Phi_0; 0)$ with

$$\Phi_0 = \{\mathsf{w}_1 \xrightarrow{i,0} \mathsf{pk}(sk_v),\ \mathsf{w}_2 \xrightarrow{i,0} sk_i,\ \mathsf{w}_3 \xrightarrow{p,0} \mathsf{aenc}(\langle \gamma, \mathsf{sign}(\gamma, sk_p)\rangle, \mathsf{pk}(sk_i))\}.$$

We have that v is playing the verifier's role with p (who is far away). We do not consider any prover's role but we assume that p (acting as a prover) has started a session with i and thus the corresponding encryption (here $\gamma \in \mathcal{N}_{\mathsf{priv}}$) has been added to the knowledge of the attacker (handle w_3). We also assume that $sk_i \in \mathcal{N}_{\mathsf{priv}}$, the private key of the agent $i \in \mathcal{M}_0$, is known by the attacker. A more realistic configuration would include other instances of the prover and the verifier roles and will probably give more knowledge to the attacker. This simple configuration is actually sufficient to retrieve the attack presented in Section 2.2.

We write $\lfloor \Phi \rfloor_a^t$ for the restriction of Φ to the agent a at time t, i.e.:

$$\lfloor \Phi \rfloor_a^t = \left\{ \mathsf{w}_i \xrightarrow{a_i,t_i} u_i \mid (\mathsf{w}_i \xrightarrow{a_i,t_i} u_i) \in \Phi \text{ and } a_i = a \text{ and } t_i \leq t \right\}.$$

Our labeled transition system is given in Figure 2 and relies on labels ℓ which can be either equal to the unobservable τ action or of the form (a, a) with $a \in \mathcal{A}$, and $\mathsf{a} \in \{\mathsf{test}, \mathsf{eq}\} \cup \{\mathsf{in}(u), \mathsf{out}(u) \mid u \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathbb{R}^+)\} \cup \{\mathsf{let}(v) \mid v \in \mathbb{R}^+\}$. The TIM rule allows time to elapse and is labeled with τ (often omitted for sake of simplicity). The OUT rule allows an output action to be executed, and the outputted term will be added to the frame. Rule EQ is used to perform some tests, and those tests are evaluated modulo the equational theory. Then, the LET rule allows us to evaluate a term that is supposed to contain a real number, and could then be used in a timing constraint through the variable z. Then, we have a rule to evaluate a timing constraint. The IN rule allows an agent a to execute an input: the received message u has been sent at time t_b by an agent b who was in possession of the message at that time. In case b is a malicious agent, i.e. $b \in \mathcal{M}_0$, the message u may have been forged through a recipe R, and b has to be in possession of all the necessary information at that time. The variable z is used to store the time at which this action has been executed.

Example 6. Continuing Example 5, we may consider the following execution which aims to mimic the trace developed in Section 2:

$$K_0 \to_{\mathcal{T}_0} \xrightarrow{v,\mathsf{in}(t_{\mathsf{aenc}})} \to_{\mathcal{T}_0} \xrightarrow{v,\mathsf{eq}} \to_{\mathcal{T}_0} \xrightarrow{v,\mathsf{eq}} \to_{\mathcal{T}_0} \xrightarrow{v,\mathsf{out}(m)} \to_{\mathcal{T}_0} K_{\mathsf{rapid}}$$

The first arrow corresponds to an application of the rule TIM with delay $\delta_0 \geq \mathsf{Dist}_{\mathcal{T}_0}(p,i) + \mathsf{Dist}_{\mathcal{T}_0}(i,v)$. Then, the IN rule is triggered considering that the message $t_{\mathsf{aenc}} = \mathsf{aenc}(\langle \gamma, \mathsf{sign}(\gamma, sk_p)\rangle, \mathsf{pk}(sk_v))$ is sent by i at time t_i such that $\mathsf{Dist}_{\mathcal{T}_0}(p,i) \leq t_i \leq \delta_0 - \mathsf{Dist}_{\mathcal{T}_0}(i,v)$. Such a message t_{aenc} can indeed be forged by i at time t_i (using recipe $R = \mathsf{aenc}(\mathsf{adec}(\mathsf{w}_3, \mathsf{w}_2), \mathsf{w}_1)$) and thus be

TIM	$(T;\Phi;t) \xrightarrow{\tau}_{\mathcal{T}_0} (T;\Phi;t+\delta)$	with $\delta \geq 0$
OUT	$((a,\mathsf{out}^z(u)).T;\Phi;t) \xrightarrow{a,\mathsf{out}(u\downarrow)}_{\mathcal{T}_0} (T\{z\to t\};\Phi \uplus \{\mathsf{w} \xrightarrow{a,t} u\downarrow\};t)$	$\mathsf{w} \in \mathcal{W}$ fresh
EQ	$((a,[u=v]).T;\Phi;t) \xrightarrow{a,\mathsf{eq}}_{\mathcal{T}_0} (T;\Phi;t)$	i $u\downarrow = v\downarrow$
LET	$((a,[z:=v]).T;\Phi;t) \xrightarrow{a,\mathsf{let}(v\downarrow)}_{\mathcal{T}_0} (T\{z\to v\downarrow\};\Phi;t)$	if $v\downarrow \in \mathbb{R}^+$
TEST	$((a,[\![t_1\sim t_2]\!]).T;\Phi;t) \xrightarrow{a,\mathsf{test}}_{\mathcal{T}_0} (T;\Phi;t)$	if $t_1 \sim t_2$ is true
IN	$((a,\mathsf{in}^z(x)).T;\Phi;t) \xrightarrow{a,\mathsf{in}(u)}_{\mathcal{T}_0} (T\{x\to u, z\to t\};\Phi;t)$	

if there exist $b \in \mathcal{A}_0$ and $t_b \in \mathbb{R}^+$ such that $t_b \leq t - \mathsf{Dist}_{\mathcal{T}_0}(b,a)$ and there exists $R \in \mathcal{T}(\Sigma, \mathcal{W} \cup \mathcal{N}_{\mathsf{pub}} \cup \mathbb{R}^+)$ such that $R\Phi\downarrow = u$. Moreover,

- if $b \in \mathcal{A}_0 \setminus \mathcal{M}_0$ then $R \in dom(\lfloor \Phi \rfloor_b^{t_b}) \cup \mathcal{N}_{\mathsf{pub}} \cup \mathbb{R}^+$;
- if $b \in \mathcal{M}_0$ then for all $\mathsf{w} \in vars(R)$, there exists $c \in \mathcal{A}_0$ such that $\mathsf{w} \in dom(\lfloor \Phi \rfloor_c^{t_b - \mathsf{Dist}_{\mathcal{T}_0}(c,b)})$.

Fig. 2: Semantics of our calculus

received by v at time δ_0. Then, tests performed by v are evaluated successfully, v outputs m, and we reach the configuration $K_{\mathsf{rapid}} = (T_{\mathsf{rapid}}; \Phi_{\mathsf{rapid}}; \delta_0)$ where:

- $T_{\mathsf{rapid}} = (v, \mathsf{out}^{z_1}(c)).(v, \mathsf{in}^{z_2}(y)).(v, [y = \mathsf{h}(c,m,\gamma)]).(v, [\![z_2 - z_1 < 2t_0]\!])$, and
- $\Phi_{\mathsf{rapid}} = \Phi_0 \uplus \{\mathsf{w}_4 \xrightarrow{v,\delta_0} m\}$.

We can pursue this execution as follows:

$$K_{\mathsf{rapid}} \xrightarrow{v,\mathsf{out}(c)}_{\mathcal{T}_0} \to_{\mathcal{T}_0} \xrightarrow{v,\mathsf{in}(\mathsf{h}(c,m,\gamma))}_{\mathcal{T}_0} \xrightarrow{v,\mathsf{eq}}_{\mathcal{T}_0} ((v, [\![\delta_0 + 2\mathsf{Dist}_{\mathcal{T}_0}(v,i) - \delta_0 < 2t_0]\!]); \Phi_{\mathsf{rapid}} \uplus \{\mathsf{w}_5 \xrightarrow{v,\delta_0} c\}; \delta_0 + 2\mathsf{Dist}_{\mathcal{T}_0}(v,i))$$

The second arrow is an application of the rule TIM with delay $2\mathsf{Dist}_{\mathcal{T}_0}(v,i)$ so that $\mathsf{h}(c,m,\gamma)$ can be received by v at time $\delta_0 + 2\mathsf{Dist}_{\mathcal{T}_0}(v,i)$. Since $\mathsf{Dist}_{\mathcal{T}_0}(v,i) < t_0$, the timing constraint is true and the last action can be executed.

The goal of this paper is to propose a new procedure for analysing a bounded number of sessions of distance bounding protocols. Once the topology is fixed, the existence of an attack can be directly encoded as a reachability property considering a finite set of traces. The following sections are thus dedicated to the study of the following problem:

Input: A trace T locally closed w.r.t. $\mathcal{X}$ and $\mathcal{Z}$, $t_0 \in \mathbb{R}^+$, and a topology $\mathcal{T}_0$.
Output: Do there exist $\ell_1, \ldots, \ell_n$, Φ, and t such that $(T;\emptyset;t_0) \xrightarrow{\ell_1 \ldots, \ell_n}_{\mathcal{T}_0} (\epsilon;\Phi;t)$?

4 Modelling using Horn clauses

Following the approach developed in Akiss [12], our procedure is based on an abstract modelling of a trace in first-order Horn clauses. Our set of seed statements is more in line with what has been implemented in Akiss for optimisation purposes rather than what is presented in [12].

OUT	$((a, \mathsf{out}^z(u)).T; \phi) \xrightarrow{a,\mathsf{out}(u\downarrow)} (T; \phi \uplus \{\mathsf{w} \to u\})$	with $\mathsf{w} \in \mathcal{W}$ fresh
EQ	$((a, [u = v]).T; \phi) \xrightarrow{a,\mathsf{eq}} (T; \phi)$	if $u\downarrow = v\downarrow$
LET	$((a, [z := v].T; \phi) \xrightarrow{a,\mathsf{let}(v\downarrow)} (T; \phi)$	
TEST	$((a, [\![t_1 \sim t_2]\!]).T; \phi) \xrightarrow{a,\mathsf{test}} (T; \phi)$	
IN	$((a, \mathsf{in}^z(x)).T; \phi) \xrightarrow{a,\mathsf{in}(u)} (T\{x \to u\}; \phi)$	if $u = R\phi\downarrow$ for some recipe R.

Fig. 3: Relaxed semantics

4.1 Preliminaries

We consider *symbolic runs* which are finite sequences of pairs with possibly a *run variable* typically denoted y at its ends. We have that each pair (a, a) is such that $a \in \mathcal{A}$ and a is an action of the form (with $u \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathbb{R}^+ \cup \mathcal{X})$):

$$\mathsf{out}(u) \qquad \mathsf{in}(u) \qquad \mathsf{eq} \qquad \mathsf{test} \qquad \mathsf{let}(u).$$

Excluding the special variable y, a symbolic run $(a_1, \mathsf{a}_1).\ldots.(a_n, \mathsf{a}_n)$, only contains variables from the set $\mathcal{X}$. We say that it is *locally closed* if whenever a variable x occurs in an output action (resp. let action) a_j, then there exists an input action a_i occurring before (i.e. $i < j$) such that $a_i = a_j$ and $x \in vars(\mathsf{a}_i)$. Symbolic runs are often denoted $w, w', \ldots$, and we write $w \sqsubseteq w'$ when the sequence w is a prefix of w'. Given a symbolic run w_0 whose sequence of outputs is $\mathsf{out}(u_1) \cdot \ldots \cdot \mathsf{out}(u_n)$, we denote $\phi(w_0) = \{\mathsf{w}_1 \to u_1, \ldots, \mathsf{w}_n \to u_n\}$.

We also consider *symbolic recipes* which are terms in $\mathcal{T}(\Sigma, \mathcal{W} \cup \mathcal{N}_{\mathsf{pub}} \cup \mathcal{Y})$ where $\mathcal{Y}$ is a set of recipe variables disjoint from $\mathcal{X}$ and $\mathcal{W}$. We use capital letters X, Y, and Z to range over $\mathcal{Y}$.

Example 7. We consider the following symbolic run:

$$w_0 = (v, \mathsf{in}(\mathsf{aenc}(\langle x', \mathsf{sign}(x', sk_p)\rangle, \mathsf{pk}(sk_v)))).(v, \mathsf{eq}).(v, \mathsf{eq}).\\ (v, \mathsf{out}(m)).(v, \mathsf{out}(c)).(v, \mathsf{in}(\mathsf{h}(c, m, x'))).(v, \mathsf{eq})$$

We have that $\phi(w_0) = \{\mathsf{w}_1 \to m, \mathsf{w}_2 \to c\}$.

Our logic is based on two predicates expressing deduction and reachability without taking into account timing constraints. More formally, given a configuration $(T; \Phi; t)$, its untimed counterpart is $(T; \phi)$ where ϕ is the untimed counterpart of Φ, i.e. a frame of the form: $\phi = \{\mathsf{w}_1 \to u_1, \ldots, \mathsf{w}_n \to u_n\}$. The relaxed semantics over untimed configurations is given in Figure 3. Since time variables (from $\mathcal{Z}$) are not instantiated during a relaxed execution, in an untimed configuration $(T; \phi)$, the trace T is only locally closed w.r.t. $\mathcal{X}$. Our predicates are:

– a *reachability predicate*: r_w holds when the run w is executable.

– a *deduction predicate*: $\mathsf{k}_w(R, u)$ holds if the message u can be built using the recipe $R \in \mathcal{T}(\Sigma, \mathcal{N}_{\mathsf{pub}} \cup \mathbb{R}^+ \cup \mathcal{W})$ by an attacker using the outputs available after the execution of w (if this execution is possible).

Formally, we have that:

– $(T_0; \phi_0) \models \mathsf{r}_{\ell_1,\ldots,\ell_n}$ if there exists $(T_n; \phi_n)$ such that $(T_0; \phi_0) \overset{\ell_1 \ldots \ell_n}{\rightsquigarrow} (T_n; \phi_n)$
– $(T_0; \phi_0) \models \mathsf{k}_{\ell_1,\ldots,\ell_n}(R, u)$ if for all $(T_n; \phi_n)$ such that $(T_0; \phi_0) \overset{\ell_1 \ldots \ell_n}{\rightsquigarrow} (T_n; \phi_n)$ we have that $R\phi_n{\downarrow} = u$.

This semantics is extended as usual to first-order formulas built using the usual connectives (e.g. conjunction, quantification, ...)

Example 8. The frame ϕ_0 below is the untimed counterpart of Φ_0:

$$\phi_0 = \{\mathsf{w}_1 \to \mathsf{pk}(sk_v),\ \mathsf{w}_2 \to sk_i,\ \mathsf{w}_3 \to \mathsf{aenc}(\langle \gamma, \mathsf{sign}(\gamma, sk_p)\rangle, \mathsf{pk}(sk_i))\}.$$

We have that $(T_{\mathsf{ex}}; \phi_0) \overset{\mathsf{tr}}{\rightsquigarrow} (\epsilon; \phi_{\mathsf{final}})$ where ϕ_{final} is the untimed counterpart of $\Phi_{\mathsf{final}} = \Phi_{\mathsf{rapid}} \uplus \{\mathsf{w}_5 \xrightarrow{v,\delta_0} c\}$, and tr is the same sequence of labels as the one developed in Example 6, i.e.

$$(v, \mathsf{in}(t_{\mathsf{aenc}}))(v, \mathsf{eq})(v, \mathsf{eq})(v, \mathsf{out}(m))(v, \mathsf{out}(c))(v, \mathsf{in}(\mathsf{h}(c, m, \gamma)))(v, \mathsf{eq})(v, \mathsf{test}).$$

4.2 Seed statements

We consider particular Horn clauses which we call *statements*.

Definition 3. *A* statement *is a Horn clause:* $H \Leftarrow \mathsf{k}_{w_1}(X_1, u_1), \ldots, \mathsf{k}_{w_n}(X_n, u_n)$ *with* $H \in \{\mathsf{r}_{w_0}, \mathsf{k}_{w_0}(R, u)\}$ *and such that:*

– $w_0, \ldots, w_n$ *are symbolic runs locally closed,* $w_i \sqsubseteq w_0$ *for any* $i \in \{1, \ldots, n\}$*;*
– $u, u_1, \ldots, u_n$ *are terms in* $\mathcal{T}(\Sigma, \mathcal{N} \cup \mathbb{R}^+ \cup \mathcal{X})$*;*
– $R \in \mathcal{T}(\Sigma, \mathcal{N}_{\mathsf{pub}} \cup \mathbb{R}^+ \cup \mathcal{W} \cup \{X_1, \ldots, X_n\}) \setminus \mathcal{Y}$*, and* $X_1, \ldots, X_n$ *are distinct variables from* $\mathcal{Y}$*.*

When $H = \mathsf{k}_{w_0}(R, u)$*, we assume in addition that* $vars(u) \subseteq vars(u_1, \ldots, u_n)$ *and* $R(\{X_i \to u_i\} \uplus \phi(w_0)){\downarrow} = u$*.*

In the above definition, we implicitly assume that all variables are universally quantified, i.e., all statements are ground. By abuse of language we sometimes call σ a grounding substitution for a statement $H \Leftarrow (B_1, \ldots, B_n)$ when σ is grounding for each of the atomic formulas $H, B_1, \ldots, B_n$. The *skeleton* of a statement f, denoted $\mathsf{skl}(f)$, is the statement where recipes are removed. Our definition of statement is in line with the original one proposed in [12] but we state an additional invariant used to establish the completeness of our procedure.

In order to define our set of seed statements, we have to fix some naming conventions. Given a trace T of the form $(a_1, \mathsf{a}_1).(a_2, \mathsf{a}_2).\ldots.(a_n, \mathsf{a}_n)$, we assume w.l.o.g. the following naming conventions:

1. $\mathsf{r}_{\ell_1\sigma\tau\downarrow\cdot\ldots\cdot\ell_n\sigma\tau\downarrow} \Leftarrow \{\mathsf{k}_{\ell_1\sigma\tau\downarrow\cdot\ldots\cdot\ell_{j-1}\sigma\tau\downarrow}(X_j, x_j\sigma\tau\downarrow)\}_{j\in\mathsf{Rcv}(n)}$
for all $\sigma \in \mathsf{csu}_{\mathcal{R}}(\{v_k = v'_k\}_{k\in\mathsf{Eq}(n)})$
for all $\tau \in \mathsf{variants}_{\mathcal{R}}(\ell_1\sigma, \ldots, \ell_n\sigma)$

2. $\mathsf{k}_{\ell_1\sigma\tau\downarrow\cdot\ldots\cdot\ell_m\sigma\tau\downarrow\mathsf{y}}(\mathsf{w}_{|\mathsf{Snd}(m)|}, u_m\sigma\tau\downarrow) \Leftarrow \{\mathsf{k}_{\ell_1\sigma\tau\downarrow\cdot\ldots\cdot\ell_{j-1}\sigma\tau\downarrow}(X_j, x_j\sigma\tau\downarrow)\}_{j\in\mathsf{Rcv}(m)}$
for all $m \in \mathsf{Snd}(n)$
for all $\sigma \in \mathsf{csu}_{\mathcal{R}}(\{v_k = v'_k\}_{k\in\mathsf{Eq}(m)})$
for all $\tau \in \mathsf{variants}_{\mathcal{R}}(\ell_1\sigma, \ldots, \ell_m\sigma)$

3. $\mathsf{k}_{\mathsf{y}}(c, c) \Leftarrow$
for all $c \in \mathcal{C}$

4. $\mathsf{k}_{\mathsf{y}}(\mathsf{f}(Y_1, \ldots, Y_k), \mathsf{f}(y_1, \ldots, y_k)\tau\downarrow) \Leftarrow \{\mathsf{k}_{\mathsf{y}}(Y_j, y_j\tau\downarrow)\}_{j\in\{1,\ldots,k\}}$
for all $\mathsf{f} \in \Sigma$ of arity k
for all $\tau \in \mathsf{variants}_{\mathcal{R}}(\mathsf{f}(y_1, \ldots, y_k))$

Fig. 4: Seed statements $\mathsf{seed}(T, \mathcal{C})$

1. if a_i is a receive action, then $\mathsf{a}_i = \mathsf{in}^{z_i}(x_i)$, and $\ell_i = (a_i, \mathsf{in}(x_i))$;
2. if a_i is a send action, then $\mathsf{a}_i = \mathsf{out}^{z_i}(u_i)$, and $\ell_i = (a_i, \mathsf{out}(u_i))$;
3. if a_i is a test action, then $\mathsf{a}_i = [v_i = v'_i]$, and $\ell_i = (a_i, \mathsf{eq})$;
4. if a_i is a let action, then $\mathsf{a}_i = [z'_i := v_i]$, and $\ell_i = (a_i, \mathsf{let}(v_i))$.
5. if a_i is a timing constraint then $\mathsf{a}_i = [\![t_i \sim t'_i]\!]$, and $\ell_i = (a_i, \mathsf{test})$.

For each $m \in \{0, \ldots, n\}$, the sets $\mathsf{Rcv}(m)$, $\mathsf{Snd}(m)$, $\mathsf{Eq}(m)$, $\mathsf{Let}(m)$, and $\mathsf{Test}(m)$ respectively denote the set of indexes of the receive, send, equality, let, and test actions amongst $\mathsf{a}_1, \ldots, \mathsf{a}_m$. We denote by $|S|$ the cardinality of S.

Given a set $\mathcal{C} \subseteq \mathcal{N}_{\mathsf{pub}} \cup \mathbb{R}^+$, the set of *seed statements* associated to T and $\mathcal{C}$, denoted $\mathsf{seed}(T, \mathcal{C})$, is defined in Figure 4. If $\mathcal{C} = \mathcal{N}_{\mathsf{pub}} \cup \mathbb{R}^+$, then $\mathsf{seed}(T, \mathcal{C})$ is said to be the set of seed statements associated to T and in this case we write $\mathsf{seed}(T)$ as a shortcut for $\mathsf{seed}(T, \mathcal{N}_{\mathsf{pub}} \cup \mathbb{R}^+)$. When computing seed statements, we compute complete sets of unifiers and complete sets of variants modulo $\mathcal{R}$. This allows us to get rid of the rewrite system in the remainder of our procedure and then only consider unification modulo the empty equational theory. In this case, it is well-known that (when it exists) $\mathsf{csu}_\emptyset(\mathcal{U})$ is uniquely defined up to some variable renaming, and we write $\mathsf{mgu}(u_1, u_2)$ instead of $\mathsf{csu}_\emptyset(\{u_1 = u_2\})$.

Example 9. Let $T^+_{\mathsf{ex}} = T_0 \cdot T_{\mathsf{ex}}$ with $T_0 = (i, \mathsf{out}(\mathsf{pk}(sk_v))).(i, \mathsf{out}(sk_i)).(p, \mathsf{out}(u))$ and $u = \mathsf{aenc}(\langle\gamma, \mathsf{sign}(\gamma, sk_p)\rangle, \mathsf{pk}(sk_i))$. The set $\mathsf{seed}(T^+_{\mathsf{ex}}, \emptyset)$ contains among others the statement f_1, f_2, f_3, and f_4 given below:

$\mathsf{r}_{T_0\cdot w_0\cdot(v,\mathsf{test})} \Leftarrow \mathsf{k}_{T_0}(X_1, \mathsf{aenc}(\langle x', \mathsf{sign}(x', sk_p)\rangle, \mathsf{pk}(sk_v))),\ \mathsf{k}_{T_0\cdot w_0^5}(X_2, \mathsf{h}(c, m, x'))$;
$\mathsf{k}_{T_0\cdot\mathsf{y}}(\mathsf{w}_3, u) \Leftarrow$;
$\mathsf{k}_{\mathsf{y}}(\mathsf{adec}(Y_1, Y_2), \mathsf{adec}(y_1, y_2)) \Leftarrow \mathsf{k}_{\mathsf{y}}(Y_1, y_1),\ \mathsf{k}_{\mathsf{y}}(Y_2, y_2)$; and its variant
$\mathsf{k}_{\mathsf{y}}(\mathsf{adec}(Y_1, Y_2), y_3) \Leftarrow \mathsf{k}_{\mathsf{y}}(Y_1, \mathsf{aenc}(y_3, \mathsf{pk}(y_2))),\ \mathsf{k}_{\mathsf{y}}(Y_2, y_2)$

where w_0 is given in Example 7, and w_0^5 is the prefix of w_0 of size 5.

Statement f_1 expresses that the trace is executable (in the relaxed semantics) as soon as we are able to deduce the two terms requested in input, f_2 says that the

attacker knows the term u as soon asT_0 has been executed. The two remaining statements model the fact that an attacker can apply the decryption algorithm on any terms he knows (statement f_3), and this will give him access to the plaintext when the right key is used (statement f_4).

4.3 Soundness and completeness

We now show that as far as the timing constraints are ignored, the set $\mathsf{seed}(T)$ is a sound and complete abstraction of a trace. Moreover, we have to ensure that the proof tree witnessing the existence of a given predicate in $\mathcal{H}(\mathsf{seed}(T))$ matches with the relaxed execution we have considered. This is mandatory to establish the completeness of our procedure.

Definition 4. *Given a set K of statements, $\mathcal{H}(K)$ is the smallest set of ground facts such that:*

$$\textsc{Conseq.}\ \frac{f = \Big(H \Leftarrow B_1, \ldots, B_n\Big) \in K \qquad B_1\sigma \in \mathcal{H}(K),\ \ldots,\ B_n\sigma \in \mathcal{H}(K) \qquad \sigma \text{ grounding for } f \qquad \mathsf{skl}(f\sigma) \text{ in normal form}}{H\sigma \in \mathcal{H}(K)}$$

Let $B_i = \mathsf{k}_{w_i}(X_i, u_i)$ for $i \in \{1, \ldots, n\}$, and w_0 the world associated to H with $v_1, \ldots, v_{k'}$ the terms occurring in input in w_0. We say that such an instance of Conseq matches with $\mathsf{exec} = (T; \emptyset) \overset{\ell_1,\ldots,\ell_p}{\rightsquigarrow} (S; \phi)$ *using $R_1, \ldots, R_k$ as input recipes if $w_0\sigma \sqsubseteq \ell_1, \ldots, \ell_p$, and there exist $\hat{R}_1, \ldots, \hat{R}_{k'}$ such that:*

- $\hat{R}_j(\{X_i \to u_i \mid 1 \leq i \leq n\} \uplus \phi(w_0))\downarrow = v_j$ *for $j \in \{1, \ldots, k'\}$; and*
- $\hat{R}_j\sigma = R_i$ *for $j \in \{1, \ldots, k'\}$.*

This notion of matching is extended to a proof tree π as expected, meaning that all the instances of Conseq used in π satisfy the property.

Actually, the completeness of our procedure will be established w.r.t. a subset of recipes, namely *uniform recipes*. We establish that an execution of a trace T_0 which only involves uniform recipes has a counterpart in $\mathcal{H}(\mathsf{seed}(T_0))$ which is uniform too.

Definition 5. *Given a frame ϕ, a recipe R is* uniform w.r.t. *ϕ if for any $R_1, R_2 \in st(R)$ such that $R_1\phi\downarrow = R_2\phi\downarrow$, we have that $R_1 = R_2$.*

Given a set K of statements, we say that a set $\{\pi_1, \ldots, \pi_n\}$ of proof trees in $\mathcal{H}(K)$ is uniform *if for any $\mathsf{k}_w(R_1, t)$ and $\mathsf{k}_w(R_2, t)$ that occur in $\{\pi_1, \ldots, \pi_n\}$, we have that $R_1 = R_2$.*

We are now able to state our soundness and completeness result.

Theorem 1. *Let T_0 be a trace locally closed w.r.t. $\mathcal{X}$.*

- $(T_0; \emptyset) \models g$ *for any $g \in \mathsf{seed}(T_0) \cup \mathcal{H}(\mathsf{seed}(T_0))$;*
- *If* $\mathsf{exec} = (T_0; \emptyset) \overset{\ell_1,\ldots,\ell_p}{\rightsquigarrow} (S; \phi)$ *with input recipes $R_1, \ldots, R_k$ that are uniform w.r.t. ϕ then*

1. $\mathsf{r}_{\ell_1,\ldots,\ell_p} \in \mathcal{H}(\mathsf{seed}(T_0))$; *and*
2. *if* $R\phi\downarrow = u$ *for some recipe* R *uniform w.r.t.* ϕ *then* $\mathsf{k}_{\ell_1,\ldots,\ell_p}(R,u) \in \mathcal{H}(\mathsf{seed}(T_0))$.

Moreover, we may assume that the proof tree witnessing these facts are uniform and match with exec *using* $R_1,\ldots,R_k$ *as input recipes.*

5 Saturation

At a high level, our procedure consists of two steps:

1. a saturation procedure which constructs a set of solved statements from the set $\mathsf{seed}(T)$; and
2. an algorithm which uses the solved statements obtained by saturation to check whether timing constraints are satisfied. This is needed to ensure that the execution obtained at step 1 is truly executable in our timed model.

5.1 Saturation procedure

We start by describing our saturation procedure. It manipulates a set of statements called a *knowledge base*.

Definition 6. *Given a statement* $f = (H \Leftarrow B_1,\ldots,B_n)$,

- f *is said to be* solved *if* $B_i = \mathsf{k}_{w_i}(X_i,x_i)$ *with* $x_i \in \mathcal{X}$ *for all* $i \in \{1,\ldots,n\}$.
- f *is said to be* well-formed *if whenever it is solved and* $H = \mathsf{k}_w(R,u)$, *we have that* $u \notin \mathcal{X}$.

A set of well-formed *statements is called a* knowledge base. *If* K *is a knowledge base,* $\mathsf{solved}(K) = \{f \in K \mid f \text{ is solved }\}$.

We restrict the use of the resolution rule and we only apply it on a selected atom. To formalise this, we assume a selection function sel which returns $\bot$ when applied on a solved statement, and an atom $\mathsf{k}_w(X,t)$ with $t \notin \mathcal{X}$ when applied on an unsolved statement. Resolution must be performed on this selected atom.

$$\textsc{Res}\ \frac{\begin{array}{c} f : H \Leftarrow \mathsf{k}_w(X,t), B_1,\ldots,B_n \in K \text{ such that } \mathsf{k}_w(X,t) = \mathsf{sel}(f) \\ g : \mathsf{k}_{w'}(R',t') \Leftarrow B_{n+1},\ldots,B_m \in \mathsf{solved}(K) \quad \sigma = \mathsf{mgu}(\mathsf{k}_w(X,t), \mathsf{k}_{w'}(R',t')) \end{array}}{h\sigma \qquad \text{where } h = \Big(H \Leftarrow B_1,\ldots,B_n,B_{n+1},\ldots,B_m\Big)}$$

Example 10. Applying resolution between f_4 and f_2 (see Example 9), we obtain:

$$\mathsf{k}_{T_0\cdot \mathsf{y}}(\mathsf{adec}(\mathsf{w}_3,Y_2),\langle\gamma,\mathsf{sign}(\gamma,sk_p)\rangle) \Leftarrow \mathsf{k}_{T_0\cdot \mathsf{y}}(Y_2,sk_i).$$

Then, we will derive $\mathsf{k}_{T_0\cdot \mathsf{y}}(\mathsf{adec}(\mathsf{w}_3,\mathsf{w}_2),\langle\gamma,\mathsf{sign}(\gamma,sk_p)\rangle) \Leftarrow$ and this solved statement (with others) will be used to perform resolution on f_1 leading (after several resolution steps) to the statement:

$$\mathsf{r}_{T_0\cdot w_0\cdot(v,\mathsf{test})} \Leftarrow \mathsf{k}_{T_0}(X_1',x'),\ \mathsf{k}_{T_0}(X_2',\mathsf{sign}(x',sk_p)),\ \mathsf{k}_{T_0\cdot w_0^5}(X_3',x')$$

Ultimately, we will derive $\mathsf{r}_{T_0\cdot w_0\sigma'\cdot(v,\mathsf{test})} \Leftarrow$ with $\sigma' = \{x' \to \gamma\}$.

During saturation, the statement obtained by resolution is given to an update function which decides whether it has to be added or not into the knowledge base (possibly after some transformations). In original Akiss, many deduction statements are discarded during the saturation procedure. This is useful to avoid non-termination issues and it is not a problem since there is no need to derive the same term (from the deduction point of view) in more than one way. Now, considering that messages need time to reach a destination, a same message emitted twice at two different locations deserves more attention.

Example 11. Let $T = (a_1, \mathsf{out}(k)).(a_2, \mathsf{out}(k)).(b, \mathsf{in}^z(x)).(b, [x = k]).(b, z < 2)$, and $\mathcal{T}_0$ be a topology such that $\mathsf{Dist}_{\mathcal{T}_0}(a_1, b) = 10$ while $\mathsf{Dist}_{\mathcal{T}_0}(a_2, b) = 1$. The configuration $(T; \emptyset; 0)$ is executable but only considering w_2 as an input recipe for x. The recipe w_1 that produces the exact same term k is not an option (even if it is outputted before w_2) since the agent a_1 who outputs it is far away from b.

Whereas the original Akiss procedure will typically discard the statement $\mathsf{k}(\mathsf{w}_2, k) \Leftarrow$ (by replacing it with an identical statement), we will keep it.

As illustrated by Example 11, we therefore need to consider more recipes (even if they deduce the same message) to accommodate timing constraints, but we have to do this in a way that does not break termination (in practice). To tackle this issue, we modified the canonicalization rule, as well as the update function to allow more deduction statements to be added in the knowledge base.

Definition 7. *The canonical form $f\Downarrow$ of a statement $f = (H \Leftarrow B_1, \ldots, B_n)$ is the statement obtained by applying the* Remove *rule given below as many times as possible.*

$$\textsc{Remove}\ \frac{H \Leftarrow \mathsf{k}_w(X,t),\ \mathsf{k}_w(Y,t),\ B_1, \ldots, B_n \qquad \textit{with } X \notin \textit{vars}(H)}{H \Leftarrow \mathsf{k}_w(Y,t),\ B_1, \ldots, B_n}$$

The intuition is that there is no need to consider several recipes (here X and Y) to deduce the same term t when such a recipe does not occur in the head of the statement.

Then, the update of K by f denoted $K \uplus \{f\}$, is defined to be K if either $\mathsf{skl}(f\Downarrow)$ is not in normal form; or $f\Downarrow$ is solved but not well-formed. Otherwise, $K \uplus \{f\} = K \cup \{f\Downarrow\}$. To initiate our saturation procedure, we start with the initial knowledge base $K_{\mathsf{init}}(S)$ associated to a set S of statements (typically $\mathsf{seed}(T, \mathcal{C})$ for some well-chosen $\mathcal{C}$). Given a set S of statements, the initial knowledge base associated to S, denoted $K_{\mathsf{init}}(S)$, is defined to be the empty knowledge base updated by the set S, i.e. $K_{\mathsf{init}}(S) = (((\emptyset \uplus f_1) \uplus f_2) \uplus \ldots f_n$ where $f_1, \ldots, f_n$ is an enumeration of the statements in S. In return, the saturation procedure produces a set $\mathsf{sat}(K)$ which is actually a knowledge base.

Then, we can establish the soundness of our saturation procedure. This is relatively straightforward and follows the same lines as the original proof.

Proposition 1. *Let T_0 be a trace locally closed w.r.t. $\mathcal{X}$, $K = \mathsf{sat}(K_{\mathsf{init}}(T_0))$. We have that $(T_0; \emptyset) \models g$ for any $g \in \mathsf{solved}(K) \cup \mathcal{H}(\mathsf{solved}(K))$.*

5.2 Completeness

Completeness is more involved. Indeed, we can not expect to retrieve all the recipes associated to a given term. To ensure termination (in practice) of our procedure, we discard some statements when updating the knowledge base, and we have to justify that those statements are indeed useless. Actually, we show that considering uniform recipes is sufficient when looking for an attack trace.

However, the notion of uniform recipe does not allow one to do the proof by induction. We therefore consider a more restricted notion that we call asap recipes. The idea is to deduce a term as soon as possible but this may depend on the agent who is performing the computation. We also rely on an ordering relation which is independent of the agent who is performing the computation, and which is compatible with our notion of asap w.r.t. any agent.

Given a relaxed execution $\mathsf{exec} = (T;\emptyset) \overset{\ell_1,\ldots,\ell_n}{\rightsquigarrow} (S;\phi)$ with input recipes $R_1,\ldots,R_k$, we define the following relations:

- $R <^{\mathsf{in}}_{\mathsf{exec}} \mathsf{w}$ when $\ell_i = a, \mathsf{in}(u)$ with input recipe R and $\ell_j = a, \mathsf{out}(u_j)$ with output recipe w for some agent a with $i < j$;
- $R' <^{\mathsf{sub}}_{\mathsf{exec}} R$ when R' is a strict subterm of R.

Then, $<_{\mathsf{exec}}$ is the smallest transitive relation over recipes built on $dom(\phi)$ that contains $<^{\mathsf{in}}_{\mathsf{exec}}$ and $<^{\mathsf{sub}}_{\mathsf{exec}}$. As usual, we denote $\leq_{\mathsf{exec}}$ the reflexive closure of $<_{\mathsf{exec}}$.

Given a timed execution $\mathsf{exec} = (T_0;\emptyset;t_0) \xrightarrow{\ell_1,\ldots,\ell_n} (S;\Phi;t)$ with $\Phi = \{\mathsf{w}_1 \xrightarrow{a_1,t_1} u_1, \ldots, \mathsf{w}_n \xrightarrow{a_n,t_n} u_n\}$, we denote by $\mathsf{agent}(\mathsf{w}_i)$ (resp. $\mathsf{time}(\mathsf{w}_i)$) the agent a_i (resp. the time t_i). The relation $<^a_{\mathsf{exec}}$ over $dom(\Phi) \times dom(\Phi)$ with $a \in \mathcal{A}$ is defined as follows: $\mathsf{w} <^a_{\mathsf{exec}} \mathsf{w}'$ when:

- either $\mathsf{time}(\mathsf{w}) + \mathsf{Dist}_{\mathcal{T}}(\mathsf{agent}(\mathsf{w}), a) < \mathsf{time}(\mathsf{w}') + \mathsf{Dist}_{\mathcal{T}}(\mathsf{agent}(\mathsf{w}'), a)$;
- or $\mathsf{time}(\mathsf{w}) + \mathsf{Dist}_{\mathcal{T}}(\mathsf{agent}(\mathsf{w}), a) = \mathsf{time}(\mathsf{w}') + \mathsf{Dist}_{\mathcal{T}}(\mathsf{agent}(\mathsf{w}'), a)$, and the output w occurs before w' in the execution exec.

This order is extended on recipes as follows: $R <^a_{\mathsf{exec}} R'$ when:

1. either $\mathsf{multi}_{\mathcal{W}}(R) <^a_{\mathsf{exec}} \mathsf{multi}_{\mathcal{W}}(R')$ where $\mathsf{multi}_{\mathcal{W}}(R)$ is the multiset of variables $\mathcal{W}$ occurring in R ordered using the multiset extension of $<^a_{\mathsf{exec}}$ on variables;
2. or $\mathsf{multi}_{\mathcal{W}}(R) = \mathsf{multi}_{\mathcal{W}}(R')$ and $|R| < |R'|$ where $|R|$ is the size (number of symbols) occurring in R;
3. or $\mathsf{multi}_{\mathcal{W}}(R) = \mathsf{multi}_{\mathcal{W}}(R')$, $|R| = |R'|$, and $|st_{\mathsf{eq}}(R)| < |st_{\mathsf{eq}}(R')|$ where $st_{\mathsf{eq}}(R) = \{(S,S') \in st(R) \times st(R) \mid S \neq S' \text{ and } S\Phi{\downarrow} = S'\Phi{\downarrow}\}$ is the set of pairs of distinct syntactic subterms of R that deduce the same term.

We have that $<^a_{\mathsf{exec}}$ is a well-founded order for any $a \in \mathcal{A}$ which is compatible with $<_{\mathsf{exec}}$, i.e. $R <_{\mathsf{exec}} R'$ implies $R <^a_{\mathsf{exec}} R'$ for any agent a.

We are now able to introduce our notion of asap recipe.

Definition 8. *Let $\mathcal{T} = (\mathcal{A}, \mathcal{M}, \mathsf{Loc})$ be a topology, and $\mathsf{exec} = (T_0;\emptyset;t_0) \xrightarrow{\ell_1,\ldots,\ell_n} (S;\Phi;t)$ be an execution. A recipe R is* asap w.r.t. $a \in \mathcal{A}$ and exec *if:*

– *either* $R \in \mathcal{N}_{\mathsf{pub}} \cup \mathbb{R}^+ \cup \mathcal{W}$ *and* $\nexists R'$ *such that* $R' <_{\mathsf{exec}} R$ *and* $R'\Phi\downarrow = R\Phi\downarrow$*;*
– *or* $R = \mathsf{f}(R_1, \ldots, R_k)$ *with* $\mathsf{f} \in \Sigma$ *and* $\nexists R'$ *such that* $R' <^a_{\mathsf{exec}} R$ *and* $R'\Phi\downarrow = R\Phi\downarrow$*.*

We may note that our definition of being asap takes care about honest agents who are not allowed to forge messages from their knowledge using recipes not in $\mathcal{W} \cup \mathcal{N}_{\mathsf{pub}} \cup \mathbb{R}^+$. Hence, a recipe $R \in \mathcal{W}$ is not necessarily replaced by a recipe R' even if $R <^a_{\mathsf{exec}} R'$ and $R'\Phi\downarrow = R\Phi\downarrow$. Actually, such a recipe R' is not necessarily an alternative to R when $a \notin \mathcal{M}_0$.

Then, we can establish completeness of our saturation procedure w.r.t. these asap recipes.

Theorem 2. *Let* $K = \mathsf{solved}(\mathsf{sat}(K_{\mathsf{init}}(T_0)))$*. Let* $\mathsf{exec} = (T_0; \emptyset; t_0) \xrightarrow{\ell_1, \ldots, \ell_p} (S; \Phi; t)$ *be an execution with input recipes* $R_1, \ldots, R_k$ *forged by* $b_1, \ldots, b_k$ *and such that each* R_j *with* $j \in \{1, \ldots, k\}$ *is asap w.r.t.* b_j *and* exec*. We have that:*

– $\mathsf{r}_{\ell_1, \ldots, \ell_p} \in \mathcal{H}(K)$ *with a proof tree matching* exec *and* $R_1, \ldots, R_k$*;*
– $\mathsf{k}_{u_0}(R, R\phi\downarrow) \in \mathcal{H}(K)$ *with a proof tree matching* exec *and* $R_1, \ldots, R_k$ *whenever* $u_0 = \ell_1, \ldots, \ell_{q-1}$ *for some* $q \in \mathsf{Rcv}(p)$ *and* R *is asap w.r.t.* $b_{|\mathsf{Rcv}(q)|}$ *and* exec*.*

Proof. (sketch) We have that asap recipes are uniform and we can therefore apply Theorem 1. This allows us to obtain a proof tree in $\mathcal{H}(\mathsf{seed}(T_0))$. Then, by induction on the proof tree, we lift it from $\mathcal{H}(\mathsf{seed}(T_0))$ to $\mathcal{H}(K)$. The difficult part is when the statement obtained by resolution is not directly added in the knowledge base. It may have been modified by the rule Remove or even discarded by the update operator. In both cases, we derive a contradiction with the fact that we are considering asap recipes. □

Example 12. Considering the relaxed execution starting from $(T_0 \cdot T_{\mathsf{ex}}, \emptyset)$ by performing the three outputs followed by the untimed version of the execution described in Example 6, we reach (ϵ, ϕ) using recipes $R_1 = \mathsf{aenc}(\mathsf{adec}(\mathsf{w}_3, \mathsf{w}_2), \mathsf{w}_1)$ and $R_2 = \mathsf{h}(\mathsf{w}_5, \mathsf{w}_4, \mathsf{proj}_1(\mathsf{adec}(\mathsf{w}_3, \mathsf{w}_2)))$. Let K be the set of solved statements obtained by saturation, we have that $\mathsf{r}_{T_0 \cdot w_0\sigma' \cdot (v, \mathsf{test})} \in \mathcal{H}(K)$ (see Example 10). Note that the symbolic run $T_0 \cdot w_0\sigma' \cdot (v, \mathsf{test})$ coincides with the labels used in the execution trace. Here, the proof tree is reduced to a leaf, and choosing $\hat{R}_1 = R_1$, $\hat{R}_2 = R_2$, gives us the matching we are looking for.

6 Algorithm

In this section, we first present our algorithm to verify whether a given timed configuration can be fully executed, and then discuss its correctness.

6.1 Description

Our procedure is given in Algorithm 1. We start with the set K of solved statements obtained by applying our saturation procedure on the trace T. We consider

Algorithm 1 Test for checking whether $(T, \emptyset, t_0)$ is executable in $\mathcal{T}_0$

1: **procedure** REACHABILITY$(K, t_0, \mathcal{T}_0)$
2: **for all** $\mathsf{r}_{\ell'_1,\ldots,\ell'_n} \Leftarrow \mathsf{k}_{w_1}(X_1, x_1), \ldots, \mathsf{k}_{w_m}(X_n, x_m) \in K$ **do**
3: let $c_1, \ldots, c_q$ be fresh public names such that
4: $\rho : vars(\ell'_1, \ldots, \ell'_n) \to \{c_1, \ldots, c_k\}$ is a bijection
5: **for all** $i \in \mathsf{Rcv}(n)$ **do**
6: **if** $\ell'_i = (a_i, \mathsf{in}(v_i))$ **then** $\overline{L_i} = \{R \mid \mathsf{k}_{\ell'_1\rho\ldots\ell'_{i-1}\rho}(R, v_i\rho) \in \mathcal{H}(K)\}$
7: **end for**
8: Let $\{i_1, \ldots, i_p\} = \mathsf{Rcv}(n)$ such that $i_1 < i_2 < \ldots < i_p$
9: **for all** $L_{i_1} \times \ldots \times L_{i_p} \in \overline{L_{i_1}} \times \ldots \times \overline{L_{i_p}}$ **do**
10: Let $\psi = \mathsf{Timing}((T; \emptyset; t_0), L_{i_1}\rho^{-1} \ldots L_{i_p}\rho^{-1}, v_{i_1}, \ldots, v_{i_p})$.
11: **if** ψ satisfiable **then return** true **end if**
12: **end for**
13: **end for**
14: **return** false
15: **end procedure**

each reachability statement in K, and after instantiating the remaining variables with fresh constants using a bijection ρ, we compute for each input $(a_i, \mathsf{in}(v_i))$ occurring in $\ell'_1 \ldots, \ell'_n$ all the possible recipes that may lead to the term $v_i\rho$ and store them in the set $\overline{L_i}$. Actually, thanks to our soundness result (Proposition 1), we know that these recipes deduce the requested terms, and it only remains to check that the timing constraints are satisfied (lines 10-11).

We consider a trace T of the form $(a_1, \mathsf{a}_1).(a_2, \mathsf{a}_2).\ldots.(a_n, \mathsf{a}_n)$ locally closed w.r.t. $\mathcal{X}$ and $\mathcal{Z}$ and we assume the naming convention given in Section 4.2. Moreover, we denote by $\mathsf{orig}(j)$ the index of the action in the trace T that performed the j^{th} output, i.e. $\mathsf{orig}(j)$ is the minimal k such that $|\mathsf{Snd}(k)| = j$. The function Timing takes as inputs the initial configuration, the recipes used to feed the inputs occurring in the trace, and the terms corresponding to these inputs. Note that all these terms may still contain variables from $\mathcal{Z}$. This function computes a formula that represents all the timing constraints that have to be satisfied to ensure the executability of the trace in our timed model. More formally, $\mathsf{Timing}((T; \emptyset; t_0)), R_{i_1} \ldots R_{i_p}, u_{i_1} \ldots u_{i_p})$ is the conjunction of the formulas:

1. $z_1 = t_0$, and $z_i \leq z_{i+1}$ for any $1 \leq i < n$;
2. $t_i \sim t'_i$ for any $i \in \mathsf{Test}(n)$ with $\mathsf{a}_i = [\![t_i \sim t'_i]\!]$;
3. $z'_i = v_i\{x_j \to u_j \mid j \in \mathsf{Rcv}(i)\}\downarrow$ for any $i \in \mathsf{Let}(n)$;
4. For any $i \in \mathsf{Rcv}(n)$, we consider the formula:
   - $z_{\mathsf{orig}(j)} + \mathsf{Dist}_{\mathcal{T}_0}(a_{\mathsf{orig}(j)}, a_i) \leq z_i$ if $R_i = \mathsf{w}_j$;
   - otherwise, we consider:

$$\bigvee_{b \in \mathcal{M}_0} \left(\bigwedge_{\{j \mid \mathsf{w}_j \in vars(R_i)\}} z_{\mathsf{orig}(j)} + \mathsf{Dist}_{\mathcal{T}_0}(a_{\mathsf{orig}}(j), b) \leq z_i - \mathsf{Dist}_{\mathcal{T}_0}(b, a_i) \right)$$

The last step of our algorithm consists in checking whether the resulting formula ψ is satisfiable or not, i.e. whether there exists a mapping from $vars(\psi)$ to $\mathbb{R}^+$ such that the formula ψ is true. Of course, even if our procedure is generic w.r.t. to timing constraints, the procedure to check the satisfiability of ψ will

depend on the constraints we consider. Actually, all the formulas encountered during our case studies are quite simple: they are expressed by equations of the form $z' - z \leq t$, and we therefore rely on the well-known Floyd-Warshall algorithm to solve them. When needed, we may rely on the simplex algorithm to solve more general linear constraints.

6.2 Termination issues

First, we may note that to obtain an effective saturation procedure, it is important to start with a finite set of seed statements. Our set $\mathsf{seed}(T)$ is infinite but as it was proved in [12], we can restrict ourselves to perform saturation using the finite set $\mathsf{seed}(T, \mathcal{C}_T)$ where $\mathcal{C}_T$ contains the public names and the real numbers occurring in the trace T. More formally, we have that:

Lemma 1. *Let $\mathcal{C}_T$ be the finite set of public names and real numbers occurring in T, and $\mathcal{C}_{all} = \mathcal{N}_{\mathsf{pub}} \cup \mathbb{R}^+$. We have that:*

$$\mathsf{sat}(K_{\mathsf{init}}(\mathsf{seed}(T, \mathcal{C}_{all}))) = \mathsf{sat}(K_{\mathsf{init}}(\mathsf{seed}(T, \mathcal{C}_T))) \cup \{\mathsf{k}_{\mathsf{y}}(c, c) \Leftarrow \mid \ c \in \mathcal{C}_{all}\}.$$

Nevertheless, the saturation may not terminate. We could probably avoid some non-termination issues by improving our update operator. However, ensuring termination in theory is a rather difficult problem (the proof of termination for the original Akiss procedure for subterm convergent theories is quite complex [12] – more than 20 pages). We would like to mention that we never encountered non-termination issues in practice on our case studies.

Another issue is that, when computing the set $\overline{L_i}$, we need to compute all the recipes R such that $\mathsf{k}_w(R, u) \in \mathcal{H}(K)$ for a given term u. This can be achieved using a simple backward search and will terminate since K only contains solved statements that are well-formed. The naive recursive algorithm will therefore consider terms $u_1, \ldots, u_n$ that are strict subterms of the initial term u. Note that statements that are not well-formed are discarded by our update operator: ensuring completeness of our saturation procedure when discarding statements that are not well-formed is the challenging part of our completeness proof.

6.3 Correctness of our algorithm

We consider a topology $\mathcal{T}_0$ and a configuration $(T; \emptyset; t_0)$ built on top of $\mathcal{T}_0$ and such that T is locally closed w.r.t. both $\mathcal{X}$ and $\mathcal{Z}$.

Theorem 3. *Let $\mathcal{C}_T \subseteq \mathcal{N}_{\mathsf{pub}} \uplus \mathbb{R}^+$ be the finite set of public names and real numbers occurring in T. Let $K = \mathsf{solved}(\mathsf{sat}(K_{\mathsf{init}}(\mathsf{seed}(T, \mathcal{C}_T))))$. We have that:*

- *if* $\textsc{Reachability}(K, t_0, \mathcal{T}_0)$ *holds then $(T; \emptyset; t)$ is executable in $\mathcal{T}_0$;*
- *if $(T; \emptyset; t_0)$ is executable in $\mathcal{T}_0$ then* $\textsc{Reachability}(K, t_0, \mathcal{T}_0)$ *holds.*

Soundness (item 1 above) is relatively straightforward. Item 2 is more involved. Of course, our algorithm does not consider all the possible recipes for inputs. Some recipes are discarded from our analysis. Actually, it is sufficient to focus our attention on asap recipes. To justify that this is not an issue regarding completeness, we first establish the following result.

Lemma 2. *Let* $\mathsf{exec} = K_0 \xrightarrow{\ell_1,\ldots,\ell_n}_{\mathcal{T}_0} (S;\Phi;t)$ *be an execution. We may assume w.l.o.g. that* exec *involves input recipes* $R_1,\ldots,R_k$ *forged by agents* $b_1,\ldots,b_k$ *and* R_i *is asap w.r.t.* b_i *and* exec *for each* $i \in \{1,\ldots,k\}$.

Then, we may apply Theorem 2 (item 1) on this "asap execution" and deduce the existence of $f = \mathsf{r}_{\ell'_1,\ldots,\ell'_n} \Leftarrow \mathsf{k}_{w_1}(X_1,x_1),\ldots,\mathsf{k}_{w_m}(X_m,x_m)$ in K and a substitution σ witnessing the fact that $\mathsf{r}_{\ell_1,\ldots,\ell_n} = \mathsf{r}_{\ell'_1\sigma,\ldots,\ell'_n\sigma} \in \mathcal{H}(K)$. Moreover, we know that f and σ match with exec and $R_1,\ldots,R_k$. Considering the symbolic recipes $\hat{R}_1,\ldots,\hat{R}_k$ witnessing this matching, and instantiating their variables with adequate fresh constants (using ρ), we can show that $\hat{R}_1\rho,\ldots,\hat{R}_k\rho$ are recipes that allow to perform the timed execution $\ell'_1\rho,\ldots,\ell'_n\rho$. Note that thanks the strong relationship we have between $R_1,\ldots,R_k$ and $\hat{R}_1,\ldots,\hat{R}_k$ (by definition of matching, $R_i = \hat{R}_i\sigma$), we know that the resulting timing constraints gathered in the formula ψ due to inputs are less restrictive, and the other ones are essentially unchanged. This allows us to ensure that the formula ψ will be satisfiable. Now, applying Lemma 2, we can assume w.l.o.g. that recipes involved in such a trace are asap, and thus according to Theorem 2 will be considered by our procedure, and put in $\overline{L_{i_1}},\ldots,\overline{L_{i_p}}$ at line 6 of Algorithm 1.

7 Implementation and case studies

We validate our approach by integrating our procedure in Akiss [12], and successfully used it on several case studies. All files related to the tool implementation and case studies are available at

http://people.irisa.fr/Alexandre.Debant/akiss-db.html.

7.1 Integration in Akiss

Our syntax is very close to the one presented in Section 3. For sake of simplicity, we sometimes omit timestamps on input/output actions. Regarding our timing constraints, our syntax only allows linear expressions of the form $z_1 - z_2 \sim z_3$ with $z_i \in \mathcal{Z} \cup \mathbb{R}^+$ and $\sim \in \{=,<,\leq\}$. These expressions are enough to model all our case studies. To ease the specification of protocols our tool support parallel composition of traces ($T_1 \mid\mid T_2$). This operator is syntactic sugar and can be translated to sets of traces in a straightforward way.

To mitigate the potential exponential blowup caused by this translation, we always favour let, equality, and test actions, as well as output actions when no timestamp occur on it. The second optimisation consists in executing input actions (without timestamps) in a raw. These optimisations will allow us to reduce the number of traces that have to be considered during our analysis, and are well-known to be sound when verifying reachability properties [30, 4].

Example 13. Let $P = (a,\mathsf{in}(x_1)).(a,\mathsf{in}(x_2).(a,\mathsf{out}(u)) \mid\mid (b,\mathsf{in}(x_3)).(b,\mathsf{out}(v))$. Computing naively all the possible interleavings will give us 10 traces to analyse. The first optimisation will allow us to reduce this number to 3, and together with the second optimisation, this number falls to 2.

7.2 Case studies

In this section we demonstrate that our tool can be effectively used to analyse distance bounding protocols and payment protocols. Our experiments have been done on a standard laptop and the results obtained confirm termination of the saturation procedure when analysing various protocols ($\times$ stands for attack, ✓ means that the protocol has been proved secure). We indicate the number of roles (running in parallel) we consider and the number of traces (due to all the possible interleaving of the roles) that have been analysed by the tool in order to conclude. Our algorithm stops as soon as an attack is found, and thus the number of possible interleavings is not relevant in this case.

We only consider two distinct topologies: one to analyse mafia fraud scenarios (2 honest agents far away with a malicious agent close to each honest agent) and one to analyse distance hijacking for which 3 agents are considered (malicious agent in the neighbourhood of the verifier on which the security property is encoded is not allowed). This may seem restrictive but it has been shown to be sufficient to capture all the possible attacks [19]. Our results are consistent with the ones obtained in [14, 26, 19, 13].

Distance bounding protocols. As explained in Section 2 on the TREAD protocol, we ignore several details that are irrelevant to a security analysis performed in the symbolic model. Moreover, our procedure is not yet able to support the exclusive-or operator and thus it has been modelled in an abstract way when analysing the protocols BC and Swiss-Knife. When no attack was found for 2 roles, we consider more roles (and thus more traces). The fact that the performances degrade when considering additional roles is not surprising and is clearly correlated with the number of traces that have to be considered.

Protocol	Mafia fraud			Distance hijacking		
	roles/ tr	time	status	roles/ tr	time	status
TREAD-Asym [2]	2 / –	1s	$\times$	2 /–	1s	$\times$
SPADE [11]	2 / –	2s	$\times$	2 / –	4s	$\times$
TREAD-Sym [2]	4 / 7500	18 min	✓	2 / –	1s	$\times$
BC [10]	4/ 5635	37 min	✓	2 / –	1s	$\times$
Swiss-Knife [25]	3 / 1080	25s	✓	3 / 7470	4min	✓
HK [23]	3/ 20	1s	✓	3/ 20	1s	✓
	4/ 3360	58s	✓	4/ 3360	47s	✓
	5 / 30240	14 min	✓	5 / 30240	12 min	✓

Payment protocols. We have also analysed three payment protocols (and some of their variants) w.r.t. mafia fraud – the only relevant scenario for this kind of application (see [13]). It happens that these protocols are more complex to analyse than traditional distance bounding protocols. They often involve more complex messages, and a larger number of message exchanges. Moreover, in

protocols MasterCard RRP and NXP, the threshold is not fixed in advance but received during the protocol execution. Due to this, these protocols fall outside the class of protocols that can be analysed by [19, 26]. To our knowledge only [13] copes with this issue by proposing a security analysis in two steps: they first establish that the value of the threshold can not be manipulated by the attacker, and then analyse the protocol considering a fixed threshold. Such a protocol can be encoded in a natural way in our calculus using the let instruction $[z := v]$ that allows one to extract a timing information from a message. We analysed these protocols considering one instance of each role.

Protocol	# tr	time	status
NXP [24]	126	4s	✓
MasterCard RRP [22]	35	6min	✓
PaySafe [14]	4	308s	✓
PaySafe-V2 [14]	–	26s	×
PaySafe-V3 [14]	–	149s	×

8 Conclusion

We presented a novel procedure for reasoning about distance bounding protocols which has been integrated in the Akiss tool. Even though termination is not guaranteed, the tool did terminate on all practical examples that we have tested.

Directions for future work include improving performances of our tool and this can be achieved by parallelising our algorithm (each trace can actually be analysed independently) and/or proposing new techniques to reduce the number of interleavings. Another interesting direction would be to add the exclusive-or operator which is often used in distance bounding protocols. This will require a careful analysis of the completeness proof developed in [3] to check whether their resolution strategy is compatible with the changes done here to accommodate timing constraints.

References

1. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. 28th Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115. ACM Press, 2001.
2. G. Avoine, X. Bultel, S. Gambs, D. Gerault, P. Lafourcade, C. Onete, and J.-M. Robert. A terrorist-fraud resistant and extractor-free anonymous distance-bounding protocol. In *Proc. 12th ACM Asia Conference on Computer and Communications Security (AsiaCCS'17)*, pages 800–814. ACM, 2017.
3. D. Baelde, S. Delaune, I. Gazeau, and S. Kremer. Symbolic verification of privacy-type properties for security protocols with xor. In *Computer Security Foundations Symposium (CSF), 2017 IEEE 30th*, pages 234–248. IEEE, 2017.
4. D. Baelde, S. Delaune, and L. Hirschi. A reduced semantics for deciding trace equivalence. *Logical Methods in Computer Science*, 13(2), 2017.

5. D. Basin, S. Capkun, P. Schaller, and B. Schmidt. Formal reasoning about physical properties of security protocols. *ACM Transactions on Information and System Security (TISSEC)*, 14(2):16, 2011.
6. D. A. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, 2005.
7. K. Bhargavan, B. Blanchet, and N. Kobeissi. Verified models and reference implementations for the TLS 1.3 standard candidate. In *Proc. 38th IEEE Symposium on Security and Privacy (S&P'17)*, pages 483–502, 2017.
8. B. Blanchet. Modeling and verifying security protocols with the applied pi calculus and proverif. *Foundations and Trends in Privacy and Security*, 1(1-2):1–135, 2016.
9. B. Blanchet. Symbolic and computational mechanized verification of the arinc823 avionic protocols. In *Proc. 30th IEEE Computer Security Foundations Symposium (CSF'17)*, pages 68–82. IEEE, 2017.
10. S. Brands and D. Chaum. Distance-bounding protocols. In *Workshop on the Theory and Application of Cryptographic Techniques*, pages 344–359. Springer, 1993.
11. X. Bultel, S. Gambs, D. Gerault, P. Lafourcade, C. Onete, and J. Robert. A prover-anonymous and terrorist-fraud resistant distance-bounding protocol. In *Proc. 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks, WISEC 2016, Darmstadt, Germany, July 18-22, 2016*, pages 121–133. ACM, 2016.
12. R. Chadha, V. Cheval, Ş. Ciobâcă, and S. Kremer. Automated verification of equivalence properties of cryptographic protocol. *ACM Transactions on Computational Logic*, 23(4), 2016.
13. T. Chothia, J. de Ruiter, and B. Smyth. Modelling and analysis of a hierarchy of distance bounding attacks. In *Proc. 27th USENIX Security Symposium (USENIX'18)*, pages 1563–1580. USENIX Association, 2018.
14. T. Chothia, F. D. Garcia, J. de Ruiter, J. van den Breekel, and M. Thompson. Relay cost bounding for contactless EMV payments. In *Proc. 19th International Conference on Financial Cryptography and Data Security (FC'15)*, volume 8975 of *Lecture Notes in Computer Science*, pages 189–206. Springer, 2015.
15. H. Comon-Lundh, V. Cortier, and E. Zălinescu. Deciding security properties for cryptographic protocols. application to key cycles. *ACM Transactions on Computational Logic*, 11(2), Jan. 2010.
16. H. Comon-Lundh and S. Delaune. The finite variant property: How to get rid of some algebraic properties. In *Proc. 16th International Conference on Rewriting Techniques and Applications (RTA'05)*, volume 3467 of *LNCS*, pages 294–307. Springer, 2005.
17. C. Cremers, M. Horvat, J. Hoyland, S. Scott, and T. van der Merwe. A comprehensive symbolic analysis of TLS 1.3. In *Proc. 24th ACM Conference on Computer and Communications Security (CCS'17)*, pages 1773–1788, 2017.
18. A. Debant and S. Delaune. Symbolic verification of distance bounding protocols. Research report, Univ Rennes, CNRS, IRISA, France, Feb. 2019.
19. A. Debant, S. Delaune, and C. Wiedling. A symbolic framework to analyse physical proximity in security protocols. In *Proc. 38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'18)*. Lipics, 2018.
20. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.
21. N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proc. Workshop on Formal Methods and Security Protocols (FMSP'99)*, Trento (Italy), 1999.

22. EMVCo. EMV contactless specifications for payment systems, version 2.6, 2016.
23. G. P. Hancke and M. G. Kuhn. An RFID distance bounding protocol. In *Proc. 1st International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM'05)*, pages 67–73. IEEE, 2005.
24. P. Janssens. Proximity check for communication devices, Oct. 31 2017. US Patent 9,805,228.
25. C. H. Kim, G. Avoine, F. Koeune, F.-X. Standaert, and O. Pereira. The Swiss-Knife RFID distance bounding protocol. In *International Conference on Information Security and Cryptology*, pages 98–115. Springer, 2008.
26. S. Mauw, Z. Smith, J. Toro-Pozo, and R. Trujillo-Rasua. Distance-bounding protocols: Verification without time and location. In *Proc. 39th IEEE Symposium on Security and Privacy (S&P'18))*, pages 152–169, 2018.
27. C. Meadows, R. Poovendran, D. Pavlovic, L. Chang, and P. Syverson. Distance bounding protocols: Authentication logic analysis and collusion attacks. In *Secure localization and time synchronization for wireless sensor and ad hoc networks*, pages 279–298. Springer, 2007.
28. S. Meier, B. Schmidt, C. Cremers, and D. Basin. The Tamarin Prover for the Symbolic Analysis of Security Protocols. In *Proc. 25th International Conference on Computer Aided Verification (CAV'13)*, volume 8044 of *LNCS*, pages 696–701. Springer, 2013.
29. J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. 8th ACM Conference on Computer and Communications Security (CCS'01)*. ACM Press, 2001.
30. S. Mödersheim, L. Viganò, and D. A. Basin. Constraint differentiation: Search-space reduction for the constraint-based analysis of security protocols. *Journal of Computer Security*, 18(4):575–618, 2010.
31. V. Nigam, C. Talcott, and A. A. Urquiza. Towards the automated verification of cyber-physical security protocols: Bounding the number of timed intruders. In *Proc. 21st European Symposium on Research in Computer Security (ESORICS'16)*, pages 450–470. Springer, 2016.
32. M. Rusinowitch and M. Turuani. Protocol insecurity with a finite number of sessions and composed keys is NP-complete. *Theoretical Computer Science*, 299(1-3):451–475, 2003.

A Proof of Theorem 1

In this section, we consider a trace T_0 which is locally closed w.r.t. $\mathcal{X}$.

Proposition 2. *We have that* $\mathsf{seed}(T_0)$ *is a set of statements.*

Proof. We consider each type of statement separately and show that each item of the definition of being a statement is satisfied. Note that T_0 is locally closed w.r.t. $\mathcal{X}$, and thus we have that the symbolic run $\ell_1 \cdot \ldots \cdot \ell_m$ with $0 \leq m \leq n$ is locally closed. Input actions are of the form $\mathsf{in}(x_i)$, and thus the property of being locally closed is still satisfied by $\ell_1\sigma\tau\downarrow \cdot \ldots \cdot \ell_m\sigma\tau\downarrow$. Hence, the result. □

As the construction of the seed is rather similar to the one in [12], the proof of Theorem 1 closely follows the original proof. However, some adaptations have been done in the completeness part. In our setting, completeness is more involved. First, it only holds when considering uniform recipes and we have to ensure an additional property (match with exec) to be able to lift it later on from $\mathcal{H}(\mathsf{seed}(T_0))$ to $\mathcal{H}(\mathsf{solved}(\mathsf{sat}(\mathsf{seed}(T_0))))$

The soundness part of Theorem 1 is a direct consequence of Lemma 3 and Lemma 4 that can be proved following the original proofs.

Lemma 3. *We have that* $(T_0; \emptyset)) \models g$ *for any statement* $g \in \mathsf{seed}(T_0)$.

Lemma 4. *Let* S *be a set of statements such that for all* $g \in S$ *we have that* $(T_0; \emptyset) \models g$. *We have that* $(T_0; \emptyset) \models g$ *for any* $g \in \mathcal{H}(S)$.

The completeness part of Theorem 1 is a direct consequence of Lemma 5. Regarding completeness, we focus on uniform recipes and establish the existence of a proof tree which is uniform too.

Lemma 5. *Let* $\mathsf{exec} = (T_0; \emptyset) \overset{\ell_1,\ldots,\ell_p}{\rightsquigarrow} (S; \phi)$ *be an execution with input recipes* $R_1, \ldots, R_k$ *that are uniform w.r.t.* ϕ. *We have that*

1. $\mathsf{r}_{\ell_1,\ldots,\ell_p} \in \mathcal{H}(\mathsf{seed}(T_0))$; *and*
2. *if* $R\phi\downarrow = u$ *for some* R *uniform w.r.t.* ϕ *then* $\mathsf{k}_{\ell_1,\ldots,\ell_p}(R, u) \in \mathcal{H}(\mathsf{seed}(T_0))$.

Moreover, we may assume that the proof trees witnessing these facts are uniform and match with exec *with input recipes* $R_1, \ldots, R_k$.

Proof. We follow the original proof but it remains to show that the proof tree witnessing these facts are uniform and match with exec using input recipes $R_1, \ldots, R_k$. We first explain how we construct a proof tree that is matching exec and then we show that it is uniform too.

When using an instance of a statement of type 3 (resp. type 4), the proof tree trivially matches with exec with input recipes $R_1, \ldots, R_k$ since $k' = 0$. When considering an instance of a statement of type 1 (resp. type 2) it is sufficient to consider $X_1, \ldots, X_{k'}$ for $\hat{R}_1, \ldots, \hat{R}_{k'}$ to obtain a proof tree matching exec with $R_1, \ldots, R_k$.

Actually, the resulting proof tree is uniform. Indeed, when considering a statement of type 4 we will always have that the recipe in the head is uniform w.r.t. ϕ (either a subterm of the initial recipe R or a subterm of an input recipe) and thus the premises will respect the uniformity. In addition, we will always construct the same proof tree when considering a deduction fact corresponding to an input of the execution. □

B Proof of Theorem 2 (soundness part)

First, we need to establish that our resolution procedure only produces statement. In particular, we have to establish the invariant we added in the definition of statement.

Lemma 6. *Let f and g be two statements, and h be defined as in the* RES *rule. If $\mathsf{skl}(h\sigma)$ is in normal form then $h\sigma$ is a statement.*

Proof. Let $H = \mathsf{k}_{w_0}(R_0, t_0)$, and $B_i = \mathsf{k}_{u_i}(X_i, t_i)$ for $i \in \{1, \ldots, m\}$. The only non trivial point is to establish that $h\sigma = \Big(H \Leftarrow B_1, \ldots, B_n, B_{n+1}, \ldots, B_m\Big)\sigma$ satisfies the following property:

$$(R_0\sigma)(\{X_i \to t_i\sigma \mid 1 \leq i \leq m\} \uplus \phi(w_0\sigma))\downarrow = t_0\sigma$$

where $\sigma_X = \{X_i \to t_i\sigma \mid 1 \leq i \leq m\}$.

We first consider the RES rule. Since f and g are two statements we know that:

- $R_0(\{X \to t\} \uplus \{X_i \to t_i \mid 1 \leq i \leq n\} \uplus \phi(w_0))\downarrow = t_0$; and
- $R'(\{X_i \to t_i \mid n+1 \leq i \leq m\} \uplus \phi(w'))\downarrow = t'$.

Therefore, we have that:

$$\begin{aligned}
&R_0\sigma(\sigma_X \uplus \phi(w_0\sigma))\downarrow \\
&= R_0\{X \to R'\}(\sigma_X \uplus \phi(w_0\sigma))\downarrow \\
&= R_0(\{X \to R'(\sigma_X \uplus \phi(w_0\sigma))\} \uplus \sigma_X \uplus \phi(w_0\sigma))\downarrow \\
&= R_0(\{X \to R'(\{X_i \to t_i \mid 1 \leq i \leq m\} \uplus \phi(w'))\sigma\} \uplus \sigma_X \uplus \phi(w_0\sigma))\downarrow \\
&= R_0(\{X \to t'\sigma\} \uplus \sigma_X \uplus \phi(w_0\sigma))\downarrow \\
&= (R_0(\{X \to t\} \uplus \{X_i \to t_i \mid 1 \leq i \leq m\} \uplus \phi(w_0)))\sigma\downarrow \\
&= t_0\sigma\downarrow = t_0\sigma
\end{aligned}$$

This concludes the proof. □

Lemma 7. *If f is a statement then $f\Downarrow$ is a statement*

Proof. Let $f = (H \Leftarrow \mathsf{k}_w(X, t),\ \mathsf{k}_w(Y, t),\ B_1, \ldots, B_n)$ be a statement such that $X \notin vars(H)$. We denote $B_i = \mathsf{k}_{w_i}(X_i, t_i)$ for all $i \in \{1, \ldots, n\}$ and $H \in \{\mathsf{r}_{w_0}, \mathsf{k}_{w_0}(R_0, t_0)\}$. We have that:

– $w_0, \ldots, w_n, w$ are symbolic runs locally closed, $w_i \sqsubseteq w_0$ for any $i \in \{1, \ldots, n\}$ and $w \sqsubseteq w_0$;
– $t, t_0, \ldots, t_n$ are terms in $\mathcal{T}(\Sigma, \mathcal{N} \cup \mathbb{R}^+ \cup \mathcal{X})$;
– $R_0 \in \mathcal{T}(\Sigma, \mathcal{N}_{\mathsf{pub}} \cup \mathbb{R}^+ \cup \{X_1, \ldots, X_n, X, Y\}) \setminus \mathcal{Y}$ and $X, Y, X_1, \ldots, X_n \in \mathcal{Y}$.

Let $g = (H \Leftarrow \mathsf{k}_w(Y,t),\ B_1, \ldots, B_n)$ the resulting formula after one application of the rule Remove on the statement f. Since $X \notin vars(H)$, it is easy to see that g is also a statement. □

Proposition 3. *Given a set S of statements, $K_{\mathsf{init}}(S)$ (resp. $\mathsf{sat}(K_{\mathsf{init}}(S))$) is a knowledge base.*

Proof. Lemma 6 ensures that Horn clauses generated during the saturation procedure are statements as soon as their skeleton is in normal form. Actually, the update function will discard a statement when its skeleton is not in normal form, thus during saturation we only consider statements. Now, it is easy to see that those statements are well-formed since the update function discard those that are not. Finally, Lemma 7 ensures that the canonised form of a statement is a statement too. □

The following lemma allows us to establish the soundness of the statement resulting from an application of the Res rule. We consider a trace T_0 locally closed w.r.t. $\mathcal{X}$.

Lemma 8. *Let $f = \Big(H \Leftarrow \mathsf{k}_w(X,t), B_1, \ldots, B_n\Big)$ be a statement such that $\mathsf{k}_w(X,t) = \mathsf{sel}(f)$, $g = \Big(\mathsf{k}_{w'}(R,t') \Leftarrow B_{n+1}, \ldots, B_m\Big)$ be a solved statement, and $\sigma = \mathsf{mgu}(\mathsf{k}_w(X,t), \mathsf{k}_{w'}(R,t'))$. If $(T_0;\emptyset) \models f$ and $(T_0;\emptyset) \models g$ then $(T_0;\emptyset) \models h$ where:*

$$h = \Big(H \Leftarrow B_1, \ldots, B_n, B_{n+1}, \ldots, B_m\Big)\sigma.$$

Proof. Let τ be a grounding substitution for h such that $(T_0;\emptyset) \models B_i\sigma\tau$ for all $i \in \{1, \ldots, m\}$. Since $(T_0;\emptyset) \models B_i\sigma\tau$ for all $i \in \{n+1, \ldots, m\}$ we deduce that $(T_0;\emptyset) \models \mathsf{k}_{w'}(R,t')\sigma\tau$. We have that $\mathsf{k}_{w'}(R,t')\sigma\tau = \mathsf{k}_w(X,t)\sigma\tau$. Therefore we have $(T_0;\emptyset) \models \mathsf{k}_w(X,t)\sigma\tau$. Therefore, we conclude that $(T_0;\emptyset) \models H\sigma\tau$ and thus $(T_0;\emptyset) \models h$. □

Lemma 9. *Let T be a trace locally closed w.r.t. $\mathcal{X}$. If $(T,\emptyset) \models h$ then $(T,\emptyset) \models h{\Downarrow}$.*

Proof. This result can be proved by induction on the number n of application of the rule Remove. Below, we show that the result is true for $n = 1$.

Let $f = (H \Leftarrow \mathsf{k}_w(X,t),\ \mathsf{k}_w(Y,t),\ B_1, \ldots, B_n)$ be a statement such that $X \notin vars(H)$, and $g = (H \Leftarrow \mathsf{k}_w(Y,t),\ B_1, \ldots, B_n)$ the statement obtained after the application of the rule Remove.

Let τ be a grounding substitution for g such that $(T,\emptyset) \models \mathsf{k}_w(Y,t)\tau$ and $(T,\emptyset) \models B_i\tau$ for all $i \in \{1, \ldots, n\}$. Let $\tau' = \tau \cup \{X \mapsto Y\tau\}$. We have that all the antecedents of $f\tau'$ are true in $(T,\emptyset)$, and since $(T,\emptyset) \models f$ by hypothesis, we deduce that $(T,\emptyset) \models H\tau' = H\tau$ (remember that $X \notin vars(H)$). This allows us to conclude. □

Now, we can establish the soundness of our saturation procedure.

Proposition 1. *Let T_0 be a trace locally closed w.r.t. $\mathcal{X}$, $K = \mathsf{sat}(K_{\mathsf{init}}(T_0))$. We have that $(T_0; \emptyset) \models g$ for any $g \in \mathsf{solved}(K) \cup \mathcal{H}(\mathsf{solved}(K))$.*

Proof. We first establish that $(T_0; \emptyset) \models g$ for any $g \in K$ by induction on the number of resolution step needed to produce g. If $g \in K_{\mathsf{init}}(T_0)$ then $g \in \mathsf{seed}(T_0)$ and thus applying Theorem 1 (soundness), we conclude that $(T_0; \emptyset) \models g$ (seed statements are already in canonical form). Otherwise, such a statement $g = g'\Downarrow$ with g' a statement obtained through the Res rule. We therefore conclude by applying Lemma 8 and Lemma 9 on g'.

Now, let $g \in \mathcal{H}(\mathsf{solved}(K))$. The fact that $(T_0; \emptyset) \models g$ is a direct consequence of Lemma 4. □

C Proof of Theorem 2 (completeness part)

In this section, we consider a trace T_0 which is locally closed w.r.t. $\mathcal{X}$.

C.1 Some useful results regarding asap recipes

Lemma 10. *Let $\mathcal{T} = (\mathcal{A}, \mathcal{M}, \mathsf{Loc})$ be a topology, and $\mathsf{exec} = (T_0; \emptyset; t_0) \xrightarrow{\ell_1, \ldots, \ell_n} (S; \Phi; t_n)$ be an execution with input recipes $R_1, \ldots, R_k$. Let R and R' be two recipes such that $R <_{\mathsf{exec}} R'$. We have that $R <^a_{\mathsf{exec}} R'$ for any $a \in \mathcal{A}$.*

Proof. Let $a \in \mathcal{A}$ and R, R' be two recipes such that $R <_{\mathsf{exec}} R'$. There exists a chain $R = R_0 <_{\mathsf{exec}} \ldots <_{\mathsf{exec}} R_n = R'$ such that each step corresponds to a step of $<^{\mathtt{in}}_{\mathsf{exec}}$ or $<^{\mathtt{sub}}_{\mathsf{exec}}$. We proof this lemma by induction on the length of this chain, and we therefore show that the property holds for one step. We distinguish two cases depending if $<^{\mathtt{in}}_{\mathsf{exec}}$ or $<^{\mathtt{sub}}_{\mathsf{exec}}$ has been applied:

- Case $R <^{\mathtt{in}}_{\mathsf{exec}} R'$. We know that $R' = \mathsf{w}$ and R is a recipe used to feed an input performed by $\mathsf{agent}(\mathsf{w})$. Observing that for all $\mathsf{w}' \in \mathit{vars}(R)$, we have that $\mathsf{time}(\mathsf{w}') + \mathsf{Dist}_{\mathcal{T}}(\mathsf{agent}(\mathsf{w}'), \mathsf{agent}(\mathsf{w})) \leq \mathsf{time}(\mathsf{w})$ we deduce that for all $\mathsf{w}' \in \mathit{vars}(R)$, we have that:

$$\begin{aligned}\mathsf{time}(\mathsf{w}') + \mathsf{Dist}_{\mathcal{T}}(\mathsf{agent}(\mathsf{w}'), \mathsf{agent}(\mathsf{w})) + \mathsf{Dist}_{\mathcal{T}}(\mathsf{agent}(\mathsf{w}), a)\\ \leq \mathsf{time}(\mathsf{w}) + \mathsf{Dist}_{\mathcal{T}}(\mathsf{agent}(\mathsf{w}), a).\end{aligned}$$

  Thus, we have that $\mathsf{time}(\mathsf{w}')+\mathsf{Dist}_{\mathcal{T}}(\mathsf{agent}(\mathsf{w}'), a) \leq \mathsf{time}(\mathsf{w})+\mathsf{Dist}_{\mathcal{T}}(\mathsf{agent}(\mathsf{w}), a)$ i.e. $\mathsf{w}' <^a_{\mathsf{exec}} \mathsf{w}$, and thus $R <^a_{\mathsf{exec}} R'$.
- Case $R <^{\mathtt{sub}}_{\mathsf{exec}} R'$. We know that R is a strict subterm of and R'. Thus, we have that $R <^a_{\mathsf{exec}} R'$. □

Note that $<^a_{\mathsf{exec}}$ is a well-founded relation, and thus we deduce that $<_{\mathsf{exec}}$ is also a well-founded relation.

Lemma 11. *Let $\mathcal{T} = (\mathcal{A}, \mathcal{M}, \mathsf{Loc})$ be a topology, $\mathsf{exec} = (T_0; \emptyset; t_0) \xrightarrow{\ell_1,\ldots,\ell_n} (S; \Phi; t_n)$ be an execution, $a \in \mathcal{A}$ and R a recipe that is asap w.r.t. a and exec. We have that R is uniform w.r.t. Φ.*

Proof. To conclude, it is sufficient to show that $|st_{\mathsf{eq}}(R)| = 0$. Assume by contradiction that $|st_{\mathsf{eq}}(R)| > 0$, and consider one of the deepest subterm S of R such that $|st_{\mathsf{eq}}(S)| > 0$ i.e. there exist $R'', R' \in st(S)$ with $R'' \neq R'$ and $R''\Phi\downarrow = R'\Phi\downarrow$. Let p'' (resp p') be the position at which R'' (resp. R') occurs in R. We distinguish 3 cases:

1. Case $\mathsf{multi}_{\mathcal{W}}(R'') <^{a}_{\mathsf{exec}} \mathsf{multi}_{\mathcal{W}}(R')$. Let $\widetilde{R} = R[R'']_{p'}$. We have that $\widetilde{R}\Phi\downarrow = R\Phi\downarrow$ since $R''\Phi\downarrow = R'\Phi\downarrow$ and $\mathsf{multi}_{\mathcal{W}}(\widetilde{R}) <^{a}_{\mathsf{exec}} \mathsf{multi}_{\mathcal{W}}(R)$. This contradicts the fact that R is asap w.r.t. a and exec.
2. Case $\mathsf{multi}_{\mathcal{W}}(R'') = \mathsf{multi}_{\mathcal{W}}(R')$ and $|R''| < |R'|$. Let $\widetilde{R} = R[R'']_{p'}$. We have that $\widetilde{R}\Phi\downarrow = R\Phi\downarrow$ since $R''\Phi\downarrow = R'\Phi\downarrow$. Moreover, we have that $\mathsf{multi}_{\mathcal{W}}(\widetilde{R}) = \mathsf{multi}_{\mathcal{W}}(R)$, and $|\widetilde{R}| < |R|$. This contradicts the fact that R is asap w.r.t. a and exec.
3. Case $\mathsf{multi}_{\mathcal{W}}(R'') = \mathsf{multi}_{\mathcal{W}}(R')$ and $|R''| = |R'|$ and $|st_{\mathsf{eq}}(R'')| < |st_{\mathsf{eq}}(R')|$. Therefore we have that R'' and R' are two distinct strict subterms of S and $|st_{\mathsf{eq}}(R')| \geq 1$. This contradicts the choice of S.

This concludes the proof. □

Lemma 12. *Let $\mathcal{T} = (\mathcal{A}, \mathcal{M}, \mathsf{Loc})$ be a topology, $\mathsf{exec} = (T_0; \emptyset; t_0) \xrightarrow{\ell_1,\ldots,\ell_n} (S; \Phi; t_n)$ be an execution, $a \in \mathcal{A}$ and R a recipe that is asap w.r.t. a and exec. For all $S \in st(R)$, we have that S is asap w.r.t. a and exec.*

Proof. Suppose by contradiction that there exists $R' \in st(R)$ such that R' is not asap w.r.t. a and exec. Let p' be the position in R such that $R|_{p'} = R'$. We distinguish two cases:

1. Case 1: $R' \in \mathcal{W}$ and there exists R'' such that $R'' <_{\mathsf{exec}} R'$ and $R'\Phi\downarrow = R''\Phi\downarrow$.
2. Case 2: $R' = \mathsf{f}(R'_1, \ldots, R'_n)$ and there exists R'' such that $R'' <^{a}_{\mathsf{exec}} R'$ and $R'\Phi\downarrow = R''\Phi\downarrow$.

Note that, in both cases, thanks to Lemma 10, we have that $R'' <^{a}_{\mathsf{exec}} R'$. We distinguish 3 cases:

1. Case $\mathsf{multi}_{\mathcal{W}}(R'') <^{a}_{\mathsf{exec}} \mathsf{multi}_{\mathcal{W}}(R')$. Let $\widetilde{R} = R[R'']_{p'}$. We have that $\widetilde{R}\Phi\downarrow = R\Phi\downarrow$ and $\mathsf{multi}_{\mathcal{W}}(\widetilde{R}) <^{a}_{\mathsf{exec}} \mathsf{multi}_{\mathcal{W}}(R)$. This contradicts the fact that R is asap w.r.t. a and exec.
2. Case $\mathsf{multi}_{\mathcal{W}}(R'') = \mathsf{multi}_{\mathcal{W}}(R')$ and $|R''| < |R'|$. Considering $\widetilde{R} = R[R'']_{p'}$ also allows us to contradict the fact that R is asap w.r.t. a and exec.
3. Case $\mathsf{multi}_{\mathcal{W}}(R'') = \mathsf{multi}_{\mathcal{W}}(R')$ and $|R''| = |R'|$ and $|st_{\mathsf{eq}}(R'')| < |st_{\mathsf{eq}}(R')|$. This will imply that $|st_{\mathsf{eq}}(R')| > 0$ and thus implies that R' is not uniform w.r.t. Φ leading to a contradiction with the result obtained by applying Lemma 11.

This concludes the proof. □

C.2 Completeness

Lemma 13. *Let* $\mathsf{exec} = (T_0; \emptyset) \overset{\ell_1,\ldots,\ell_p}{\rightsquigarrow} (S; \phi)$ *be an execution with input recipes* $R_1, \ldots, R_k$*. Let* $g = (\mathsf{k}_{u_0}(R, t) \Leftarrow B_1, \ldots, B_n)$ *be a solved statement where* $B_i = \mathsf{k}_{u_i}(X_i, x_i)$*,* u_0 *is locally closed, and* $v_1, \ldots, v_{k'}$ *denote the terms occurring in input in* u_0*. Let* σ *grounding for* g *such that* $\mathsf{skl}(g\sigma)$ *is in normal form. Moreover, we assume that* g *and* σ *match with* exec *and* $R_1, \ldots, R_k$*.*

For all $1 \leq j \leq k'$*, for all* $x \in vars(v_j)$*, there exists* $i_0 \in \{1, \ldots, n\}$ *such that* $x_{i_0} = x$ *and* $X_{i_0}\sigma \leq_{\mathsf{exec}} \hat{R}_j\sigma$*.*

Proof. We have that $\mathsf{skl}(g\sigma)$ is in normal form, and since g and σ match with exec and $R_1, \ldots, R_k$, we know that $u_0\sigma \sqsubseteq \ell_1, \ldots, \ell_p$ and there exist $\hat{R}_1, \ldots, \hat{R}_{k'}$ such that:

- $\hat{R}_j\sigma = R_i$ for $j \in \{1, \ldots, k'\}$; and
- $\hat{R}_j(\{X_i \to t_i \mid 1 \leq i \leq n\} \uplus \phi(u_0))\downarrow = v_j$ for $j \in \{1, \ldots, k'\}$.

We show this result by induction on j.

Base case: $j = 1$*, i.e.* v_j *is the first input occurring in* u_0*.* We have that $\hat{R}_1\sigma = R_1$ and since R_1 only uses frame elements occurring before the first input, this is the same for $\hat{R}_1$ and thus $\phi(u_0)$ will not introduce any variable. Therefore, if $x \in vars(v_1)$, we know that x has been introduced by $X_{i_0} \to x_{i_0}$ for some $i_0 \in \{1, \ldots, n\}$, i.e. $x = x_{i_0}$ for some $i_0 \in \{1, \ldots, n\}$ such that $X_{i_0} \in vars(\hat{R}_1)$, and therefore $X_{i_0}\sigma \leq_{\mathsf{exec}} \hat{R}_1\sigma$ (subterm relation).

Induction step: $j > 1$. Let $x \in vars(v_j)$ with $j > 1$. We have that $\hat{R}_j\sigma = R_j$ and since R_j only uses frame elements occurring before the j^{th} input, this is the same for $\hat{R}_j$, and thus either x has been introduced by $X_{i_0} \to x_{i_0}$ for some $i_0 \in \{1, \ldots, n\}$, and we conclude as in the previous case. Otherwise, the variable x is introduced through a frame element $\{w \to t\}$ with x in t, and $w \in vars(\hat{R}_j)$, and thus $w \leq_{\mathsf{exec}} R_j = \hat{R}_j\sigma$ (subterm relation). Because u_0 is locally closed, x occurs before in an input v_i performed by the same agent, thus $R_i = \hat{R}_i\sigma \leq^{\mathsf{in}}_{\mathsf{exec}} w$. Applying our induction hypothesis on $i < j$, there exists i_0 such that $x_{i_0} = x$ and $X_{i_0}\sigma \leq \hat{R}_i\sigma$. Relying on transitivity, we easily conclude. □

Lemma 14. *Let* K *be a set of statements. Let* $g = (H \Leftarrow B_1, \ldots, B_n)$ *be a statement and* σ *be a substitution such that for all* $i \in \{1, \ldots, n\}$*,* $B_i\sigma \in \mathcal{H}(K)$ *with a proof tree* π_i *such that* $\{\pi_1, \ldots, \pi_n\}$ *is uniform. If* g *and* σ *match with* exec *and* $R_1, \ldots, R_k$ *then* $g\Downarrow$ *and* σ *match with* exec *and* $R_1, \ldots, R_k$*.*

Proof. For $i \in \{1, \ldots, n\}$, we denote by $\mathsf{k}_{w_i}(X_i, t_i)$ each B_i. We show the result by induction on the number p of applications of the rule Remove. First, in case $p = 0$, we have that $g = g\Downarrow$ and the result is immediate. Now, we assume that there exist $i_1, i_2 \in \{1, \ldots, n\}$ such that $t_{i_1} = t_{i_2}$ with $X_{i_1} \notin vars(H)$.

By hypothesis we have that g and σ match with exec and $R_1, \ldots, R_k$ and thus there exist recipes $\hat{R}_1, \ldots, \hat{R}_{k'}$ such that:

– $\hat{R}_j(\{X_i \to t_i \mid 1 \leq i \leq n\} \uplus \phi(u_0))\downarrow = v_j$ for $j \in \{1, \ldots, k'\}$;
– $\hat{R}_j\sigma = R_i$ for $j \in \{1, \ldots, k'\}$; and
– $u_0\sigma \sqsubseteq \ell_1, \ldots, \ell_p$ with u_0 the underlying world of H.

where $v_1, \ldots, v_{k'}$ are the terms occurring in input in u_0 (the world of H).

Let $\hat{R}'_j = \hat{R}_j\{X_{i_1} \mapsto X_{i_2}\}$ for any $j \in \{1, \ldots, n\}$. Since $t_{i_1} = t_{i_2}$ and $w_{i_1} = w_{i_2}$, we have that $t_{i_1}\sigma = t_{i_2}\sigma$ and $w_{i_1}\sigma = w_{i_2}\sigma$, and by definition of uniformity, we deduce that $X_{i_1}\sigma = X_{i_2}\sigma$ and thus $\hat{R}'_j\sigma = \hat{R}_j\sigma$ for any $j \in \{1, \ldots, n\}$. Moreover $\hat{R}'_i\{X_i \to t_i \mid 1 \leq i \leq n \wedge i \neq i_1\} = \hat{R}_i\{X_i \to t_i \mid 1 \leq i \leq n\}$. We have that the set $\{\pi_1, \ldots, \pi_{i_1-1}, \pi_{i_1+1}, \ldots, \pi_n\}$ is uniform, and thus we easily conclude by applying our induction hypothesis. □

Lemma 15. *Let* $\mathsf{exec} = (T_0; \emptyset; t_0) \xrightarrow{\ell_1, \ldots, \ell_p} (S; \Phi; t)$ *be an execution with input recipes* $R_1, \ldots, R_k$ *forged by* $b_1, \ldots, b_k$ *and such that each* R_j *with* $j \in \{1, \ldots, k\}$ *is uniform w..r.t.* Φ. *Let* $K = \mathsf{sat}(K_{\mathsf{init}}(T_0))$, *and* $g = (H \Leftarrow B_1, \ldots, B_n) \in K$ *be such that* u_0, *the underlying world of* H, *is locally closed. Let* σ *be a substitution grounding for* g *such that* $\mathsf{skl}(g\sigma)$ *is in normal form, and* g *and* σ *match with* exec *and* $R_1, \ldots, R_k$.

Moreover, in case H *is of the form* $H = \mathsf{k}_{u_0}(R_H, t_H)$, *we assume that* $u_0\sigma = \ell_1, \ldots, \ell_{q-1}$ *for some* $q \in \mathsf{Rcv}(p)$ *and* $R_H\sigma$ *is asap w.r.t.* $b_{|\mathsf{Rcv}(q)|}$ *and* exec.

Assuming that $B_i\sigma \in \mathcal{H}(\mathsf{solved}(K))$ *with a proof tree* π_i *matching* exec *and* $R_1, \ldots, R_k$ *for each* $i \in \{1, \ldots, n\}$, *and* $\{\pi_1, \ldots, \pi_n\}$ *is uniform, then we have that* $H\sigma \in \mathcal{H}(\mathsf{solved}(K))$ *with a proof tree* π' *matching with* exec *and* $R_1, \ldots, R_k$, *and such that* $\mathsf{nodes}(\pi') \subseteq \bigcup_{i \in \{1, \ldots, n\}} \mathsf{nodes}(\pi_i) \cup \{H\sigma\}$.

Proof. We prove this result by induction on the sum of the sizes of the proof trees witnessing that $B_1\sigma, \ldots, B_n\sigma \in \mathcal{H}(\mathsf{solved}(K))$. If g is solved, then since $g \in K$, we conclude by choosing π' to be $\pi_1, \ldots, \pi_n$ on which we apply Conseq with g and σ.

Otherwise, i.e. g is not solved. Let $B_j = \mathsf{sel}(g) = \mathsf{k}_{u_j}(X_j, t_j)$. By hypothesis, we have that $B_j\sigma \in \mathcal{H}(\mathsf{solved}(K))$ with a proof tree π_j matching with exec and $R_1, \ldots, R_k$ as input recipes. Therefore, π_j is ending with a statement

$$h = \mathsf{k}_{u'_0}(R_0, t_0) \Leftarrow B'_1, \ldots, B'_m \in \mathsf{solved}(K)$$

and a substitution σ' grounding for h such that $\mathsf{k}_{u'_0}(R_0, t_0)\sigma' = \mathsf{k}_{u_j}(X_j, t_j)\sigma$ and $B'_i\sigma' \in \mathsf{solved}(K)$ for $i \in \{1, \ldots, m\}$ with a proof tree π'_i (subtree of π_j) matching with exec and $R_1, \ldots, R_k$.

Moreover, we have that the sum of the size of the proof tree witnessing that $B'_i\sigma' \in \mathsf{solved}(K)$ for $i \in \{1, \ldots, m\}$ is smaller than the size of the proof tree π_j. Let $H_0 = \mathsf{k}_{u'_0}(R_0, t_0)$. We apply the Resolution rule between g and h. Since $\sigma \uplus \sigma'$ unifies H_0 and $\mathsf{k}_{u_j}(X_j, t_j)$, there is $\omega = \mathsf{mgu}(H_0, \mathsf{k}_{u_j}(X_j, t_j))$ and τ such that $\sigma \uplus \sigma' = \omega\tau$. Let g' be the resulting statement. We have that

$$g' = H\omega \Leftarrow B_1\omega, \ldots, B_{j-1}\omega, B_{j+1}\omega, \ldots, B_n\omega, B'_1\omega, \ldots, B'_m\omega.$$

In order to conclude relying on our induction hypothesis, we distinguish two cases.

Case 1: $g'\Downarrow$ is added to the knowledge base by the update function. We conclude relying on our induction hypothesis considering $g'\Downarrow$ and τ. We have that $u_0\omega$ is locally closed. We have that $\mathsf{skl}(g'\Downarrow\tau)$ is in normal form since $\mathsf{skl}(g\sigma)$ and $\mathsf{skl}(h\sigma')$ are in normal form. The recipe occurring in $H\omega\tau = H\sigma$ (if any) is asap w.r.t. $b_{|\mathsf{Rcv}(q)|}$ and exec and all the antecedents of $g'\Downarrow\tau$ are in $\mathcal{H}(\mathsf{solved}(K))$ with a proof tree matching with exec and $R_1, \ldots, R_k$. Finally we have that $\{\pi_1, \ldots, \pi_{j-1}, \pi_{j+1}, \ldots, \pi_n, \pi'_1, \ldots, \pi'_m\}$ is uniform. It remains to show that $g'\Downarrow$ and τ match exec and $R_1, \ldots, R_k$. To do so we first show that g' and τ match exec and $R_1, \ldots, R_k$ to be able to apply Lemma 14.

Given a world u, i.e. a sequence of actions possibly followed by a variable, we denote by $|u|$ the number of actions in the sequence u. By definition of the RES rule and the form of the statement, we have that :

- either $|u_j\omega| = |u_j|$, thus $|u_0\omega| = |u_0|$;
- or $|u_j\omega| > |u_j|$, and in such a case, we have that $u_0\omega = u_j\omega = u'_0\omega$.

We consider these two cases separately. In the first case, we will rely on the fact that g and σ match with exec and $R_1, \ldots, R_k$, whereas in the second case, we will rely on the fact that h and σ' match with exec and $R_1, \ldots, R_k$,

Case a: $|u_0\omega| = |u_0|$. By hypothesis, we have that g and σ match with exec and $R_1, \ldots, R_k$, thus we know that there exist recipes $\hat{R}_1, \ldots, \hat{R}_{k'}$ such that:

- $\hat{R}_j(\{X_i \to t_i \mid 1 \leq i \leq n\} \uplus \phi(u_0))\downarrow = v_j$ for $j \in \{1, \ldots, k'\}$;
- $\hat{R}_j\sigma = R_i$ for $j \in \{1, \ldots, k'\}$; and
- $u_0\sigma \sqsubseteq \ell_1, \ldots, \ell_p$

where $v_1, \ldots, v_{k'}$ are the terms occurring in input in u_0.

To establish that g' and τ match with exec and $R_1, \ldots, R_k$, we consider $\hat{R}_1\omega, \ldots, \hat{R}_{k'}\omega$. Let $v_1, \ldots, v_{k'}$ be the terms occurring in input in $u_0\omega$, and let us denote $B'_i = \mathsf{k}\_(X'_i, x'_i)$ for $1 \leq i \leq m$. The only difficult point is to show that:

$$\hat{R}_j\omega(\{X_i \to t_i\omega \mid 1 \leq i \leq n \text{ and } i \neq j\} \uplus \{X'_i \to x'_i\omega \mid 1 \leq i \leq m\} \uplus \phi(u_0\omega))\downarrow = v'_j$$

Let $j \in \{1, \ldots, k'\}$. By hypothesis, $\hat{R}_j(\{X_i \to t_i \mid 1 \leq i \leq n\} \uplus \phi(u_0))\downarrow = v_j$, and thus, since $v_j\omega$ is in normal form, we have that:

$$(\hat{R}_j\{X_j \to t_j\omega\})(\{X_i \to t_i\omega \mid 1 \leq i \leq n \text{ and } i \neq j\} \uplus \phi(u_0\omega))\downarrow = v_j\omega$$

By definition of being a statement (invariant applied on h), we have that:

$$R_0(\{X'_i \to x'_i \mid 1 \leq i \leq m\} \uplus \phi(u'_0))\downarrow = t_0$$

Applying ω (note that $R_0\omega = R_0$), we deduce that:

$$R_0(\{X'_i \to x'_i\omega \mid 1 \leq i \leq m\} \uplus \phi(u'_0\omega))\downarrow = t_0\omega$$

Therefore, we have that:

$$\begin{aligned}
&\hat{R}_j\omega(\{X_i \to t_i\omega \mid 1 \leq i \leq n \text{ and } i \neq j\} \uplus \{X'_i \to x'_i\omega \mid 1 \leq i \leq m\} \uplus \phi(u_0\omega))\downarrow \\
&= (\hat{R}_j\{X_j \to R_0\})(\{X_i \to t_i\omega \mid 1 \leq i \leq n \text{ and } i \neq j\} \uplus \{X'_i \to x'_i\omega \mid 1 \leq i \leq m\} \uplus \phi(u_0\omega))\downarrow \\
&= (\hat{R}_j\{X_j \to R_0\{X'_i \to x'_i\omega \mid 1 \leq i \leq m\}\})(\{X_i \to t_i\omega \mid 1 \leq i \leq n \text{ and } i \neq j\} \uplus \phi(u_0\omega))\downarrow \\
&= (\hat{R}_j\{X_j \to t_0\omega\})(\{X_i \to t_i\omega \mid 1 \leq i \leq n \text{ and } i \neq j\} \uplus \phi(u_0\omega))\downarrow \\
&= (\hat{R}_j\{X_j \to t_j\omega\})(\{X_i \to t_i\omega \mid 1 \leq i \leq n \text{ and } i \neq j\} \uplus \phi(u_0\omega))\downarrow \\
&= v_j\omega = v'_j
\end{aligned}$$

We have that g' and τ match with exec and $R_1, \ldots, R_k$. Moreover, because $\pi'_1, \ldots, \pi'_m$ are subtrees of π_j we have that $\{\pi_1, \ldots, \pi_{j-1}, \pi_{j+1}, \ldots, \pi_n, \pi'_1, \ldots, \pi'_m\}$ is uniform. We can thus apply Lemma 14 to obtain that $g'\Downarrow$ and τ match with exec and $R_1, \ldots, R_k$. Our induction hypothesis applies and we obtain that $H\omega\tau \in \mathcal{H}(\mathsf{solved}(K))$ with a proof tree π' matching exec and $R_1, \ldots, R_k$ and such that $\mathsf{nodes}(\pi') \subseteq \bigcup_{i\in\{1,\ldots,n\}\setminus\{j\}} \mathsf{nodes}(\pi_i) \cup \bigcup_{i\in\{1,\ldots,m\}} \mathsf{nodes}(\pi'_i) \cup \{H\omega\tau\}$. Hence we have that $\mathsf{nodes}(\pi') \subseteq \bigcup_{i\in\{1,\ldots,n\}} \mathsf{nodes}(\pi_i) \cup \{H\sigma\}$ and this concludes the first case of our proof.

Case b: $u_0\omega = u'_0\omega$. By hypothesis, we know that h and σ' match with exec and $R_1, \ldots, R_k$, thus we know that there exist recipes $\hat{R}'_1, \ldots, \hat{R}'_{k'}$ such that:

- $\hat{R}'_j(\{X'_i \to x'_i \mid 1 \leq i \leq m\} \uplus \phi(u'_0))\downarrow = v_j$ for $j \in \{1, \ldots, k'\}$;
- $\hat{R}'_j\sigma' = R_i$ for $j \in \{1, \ldots, k'\}$; and
- $u'_0\sigma' \sqsubseteq \ell_1, \ldots, \ell_p$.

where $v_1, \ldots, v_{k'}$ are the terms occurring in input in u'_0 and $B'_i = \mathsf{k\_}(X'_i, x'_i)$ for $1 \leq i \leq m$. To establish that g' and τ match with exec and $R_1, \ldots, R_k$, we consider $\hat{R}'_1\omega, \ldots, \hat{R}'_{k'}\omega$. Let $v'_1, \ldots, v'_{k'}$ be the terms occurring in input in $u_0\omega$. The only difficult part is to show that:

$$\hat{R}_j\omega(\{X_i \to t_i\omega \mid 1 \leq i \leq n \text{ and } i \neq j\} \uplus \{X'_i \to x'_i\omega \mid 1 \leq i \leq m\} \uplus \phi(u_0\omega))\downarrow = v'_j$$

By hypothesis, we have that $\hat{R}_j(\{X'_i \to x'_i \mid 1 \leq i \leq m\} \uplus \phi(u'_0))\downarrow = v_j$, and thus, since $v_j\omega$ is in normal form, $X_i \notin \mathit{vars}(\hat{R}_j)$ for all $i \in \{1, \ldots, n\}$, and $\mathit{dom}(\omega) \cap \mathcal{Y} = \{X_j\}$, we have that:

$$\begin{aligned}
&\hat{R}_j\omega(\{X_i \to t_i\omega \mid 1 \leq i \leq n \text{ and } i \neq j\} \uplus \{X'_i \to x'_i\omega \mid 1 \leq i \leq m\} \uplus \phi(u_0\omega))\downarrow \\
&= \hat{R}_j(\{X'_i \to x'_i\omega \mid 1 \leq i \leq m\} \uplus \phi(u'_0\omega))\downarrow = v_j\omega = v'_j
\end{aligned}$$

We have that g' and τ match with exec and $R_1, \ldots, R_k$. Moreover, because $\pi'_1, \ldots, \pi'_m$ are subtrees of π_j we have that $\{\pi_1, \ldots, \pi_{j-1}, \pi_{j+1}, \ldots, \pi_n, \pi'_1, \ldots, \pi'_m\}$ is uniform. We can thus apply Lemma 14 to obtain that $g'\Downarrow$ and τ match with exec and $R_1, \ldots, R_k$. Our induction hypothesis applies and we obtain that $H\omega\tau \in \mathcal{H}(\mathsf{solved}(K))$ with a proof tree π' matching exec and $R_1, \ldots, R_k$ and such that $\mathsf{nodes}(\pi') \subseteq \bigcup_{i\in\{1,\ldots,n\}\setminus\{j\}} \mathsf{nodes}(\pi_i) \cup \bigcup_{i\in\{1,\ldots,m\}} \mathsf{nodes}(\pi'_i) \cup \{H\omega\tau\}$. Hence we have that $\mathsf{nodes}(\pi') \subseteq \bigcup_{i\in\{1,\ldots,n\}} \mathsf{nodes}(\pi_i) \cup \{H\sigma\}$ and this concludes

our proof.

Case 2: $g'\Downarrow$ *is not added to the knowledge base by the update function.*

Let $H = \mathsf{k}_{u_0}(R_H, t_H)$. In such a case, we know that $g'\Downarrow$ is a solved deduction statement, and since such a statement has been discarded, it means that $t_H\omega$ is a variable x. Let $g'\Downarrow = \mathsf{k}_{u_0\omega}(R_H\omega, x) \Leftarrow \mathsf{k}\_(Z_1, z_1), \ldots, \mathsf{k}\_(Z_q, z_q)$. By definition of being a statement, we know that $R_H\omega(\{Z_i \to z_i \mid 1 \leq i \leq q\} \uplus \phi(u_0\omega))\downarrow = x$. Either, the variable x has been introduced by $\{Z_i \to z_i\}$ for some i such that $z_i = x$, and Z_i occurring in $R_H\omega$. In such a case, we have that Z_i is a strict subterm of $R_H\omega$ since $R_H\omega$ is not a variable, and therefore $Z_i\tau <_{\mathsf{exec}} R_H\omega\tau$. Otherwise, x is introduced by a frame element $\mathsf{w} \to t$ with $x \in vars(t)$, and $\mathsf{w} \in vars(R_H\omega)$. Therefore, we have that $\mathsf{w} \leq_{\mathsf{exec}} R_H\omega\tau$. Because $u_0\omega$ is locally closed, we know that x occurs in an input in $u_0\omega$ (the action ℓ_i), and we have that $R_i <_{\mathsf{exec}} \mathsf{w}$. Lemma 13 applies considering $g'\Downarrow$, τ, $\hat{R}_1\omega, \ldots, \hat{R}_{k'}\omega$. Similarly to Case 1, we can show that the hypotheses are satisfied. There exists $i_0 \in \{1, \ldots, q\}$ such that $x = z_{i_0}$ and $Z_{i_0}\tau \leq_{\mathsf{exec}} \hat{R}_i\omega\tau = R_i <_{\mathsf{exec}} \mathsf{w} \leq_{\mathsf{exec}} R_H\omega\tau$.

Therefore, in both cases, we have that there exists $i_0 \in \{1, \ldots, q\}$ such that $x = z_{i_0}$ and $Z_{i_0}\tau <_{\mathsf{exec}} R_H\omega\tau$. Thanks to Proposition 1, we know that $Z_{i_0}\tau$ is a recipe for $x\tau$. If $R_H\omega\tau \in \mathcal{W}$ we immediately contradict that $R_H\omega\tau = R_H\sigma$ is asap w.r.t. $b_{|\mathsf{Rcv}(q)|}$ and exec. Otherwise, applying Lemma 10 we obtain that $Z_{i_0}\tau <_{\mathsf{exec}}^{b_{|\mathsf{Rcv}(q)|}} R_H\omega\tau$ and this leads to a contradiction with the fact that $R_H\omega\tau = R_H\sigma$ is supposed to be asap w.r.t. $b_{|\mathsf{Rcv}(q)|}$ and exec. □

Lemma 16. *Let* $\mathsf{exec} = (T_0; \emptyset; t_0) \xrightarrow{\ell_1, \ldots, \ell_p} (S; \Phi; t)$ *be an execution with input recipes* $R_1, \ldots, R_k$ *forged by* $b_1, \ldots, b_k$ *and such that each* R_j *with* $j \in \{1, \ldots, k\}$ *is asap w.r.t.* b_j *and* exec. *Let* $K = \mathsf{solved}(\mathsf{sat}(K_{\mathsf{init}}(T_0)))$, *and* $H \in \mathcal{H}(\mathsf{seed}(T_0))$ *with a uniform proof tree* π *matching with* exec *and* $R_1, \ldots, R_k$. *Moreover, in case* H *is of the form* $H = \mathsf{k}_{u_0}(R, t)$, *we assume that* $u_0 = \ell_1, \ldots, \ell_{q-1}$ *for some* $q \in \mathsf{Rcv}(p)$ *and* R *is asap w.r.t.* $b_{|\mathsf{Rcv}(q)|}$ *and* exec.

We have that $H \in \mathcal{H}(K)$ *with a proof tree* π' *matching with* exec *and* $R_1, \ldots, R_k$, *and such that* $\mathsf{nodes}(\pi') \subseteq \mathsf{nodes}(\pi)$.

Proof. We do the proof by induction on π.
Base case: In such a case, we have that there is $f \in \mathsf{seed}(T_0)$ of the form $f = (H \Leftarrow)$ and σ grounding for f such that $\mathsf{skl}(f\sigma)$ is in normal form. Moreover, if $H = \mathsf{k}_u(R_0, t_0)$ we know that $vars(t_0) = \emptyset$, thus t_0 is not a variable, and has been added to the knowledge base. We have that $f \in \mathsf{solved}(K)$. Then f and σ trivially match with exec and $R_1, \ldots, R_k$ because the underlying world of H is either a variable (statement of type 3 or 4) or does not contain any input (statement of type 1 or 2). Finally, $\mathsf{nodes}(\pi') = \mathsf{nodes}(\pi)$ because we keep the same proof tree.

Induction step: In such a case, we know that there exists $f \in \mathsf{seed}(T_0)$ of the form $H \Leftarrow B_1, \ldots, B_n$ and σ grounding for f such that $\mathsf{skl}(f\sigma)$ is in normal form,

and for all $i \in \{1, \ldots, n\}$, $B_i\sigma \in \mathcal{H}(\mathsf{seed}(T_0))$ with a proof tree π_i matching exec and $R_1, \ldots, R_k$ and since π is uniform we have that $\{\pi_1, \ldots, \pi_n\}$ is uniform.

Let us distinguish two cases depending on the type of the statement f:

- if f is a statement of type 1 or 2 then for all $i \in \{1, \ldots, n\}$, there exists $j \in \mathsf{Rcv}(p)$ such that $B_i\sigma = \mathsf{k}_{\ell_1,\ldots,\ell_{j-1}}(R_{|\mathsf{Rcv}(j)|}, u_i)$ for some term u_i. Therefore, by hypothesis we have that $R_{|\mathsf{Rcv}(j)|}$ is asap w.r.t. $b_{|\mathsf{Rcv}(j)|}$ and exec. Moreover π_i (the proof of $B_i\sigma \in \mathcal{H}(\mathsf{seed}(T_0))$) is uniform (as a subtree of π). Our induction hypothesis applies with $B_i\sigma$.
- if f is a statement of type 4 then for all $i \in \{1, \ldots, n\}$, $B_i\sigma = \mathsf{k}_{\ell_1,\ldots,\ell_{j-1}}(S_i, u_i)$ for some term u_i, with S_i a strict subterm of R. Since R is asap w.r.t. $b_{|\mathsf{Rcv}(j)|}$ and exec, we deduce that S_i is asap w.r.t. $b_{|\mathsf{Rcv}(j)|}$ and exec (by Lemma 12). We still have that π_i (the proof of $B_i\sigma \in \mathcal{H}(\mathsf{seed}(T_0))$) is uniform (as a subtree of π). Our induction hypothesis applies with $B_i\sigma$.

Therefore in both cases, we have that for all $i \in \{1, \ldots, n\}$, $B_i\sigma \in \mathcal{H}(K)$ with a proof tree π'_i matching with exec and $R_1, \ldots, R_k$, and $\mathsf{nodes}(\pi'_i) \subseteq \mathsf{nodes}(\pi_i)$. Because $\{\pi_1, \ldots, \pi_n\}$ is uniform, we have that $\{\pi'_1, \ldots, \pi'_n\}$ is uniform too.

Since the rule Remove cannot be applied on seed statement, we can distinguish two cases:

1. f is added in the knowledge base, i.e. $f \in K$. Then, we conclude that $H\sigma \in \mathcal{H}(\mathsf{solved}(K))$ thanks to Lemma 15 applied on f and σ, because f and σ match with exec and $R_1, \ldots, R_k$ using recipes $\hat{R}_j = X_j$ for all $i \in \{1, \ldots, |w|\}$ with w is the underlying world of H. In addition we have that $\{\pi'_1, \ldots, \pi'_n\}$ is uniform.
2. f is not added in the knowledge base by the update function. Note that $\mathsf{skl}(f)$ is in normal form since $\mathsf{skl}(f\sigma)$ is in normal form. Thus, we have that f is solved but not well-formed. Let $H = \mathsf{k}_w(R_0, u_0)$. Since f is not well-formed, we know that $u_0 = x$, and by definition of being a statement, we have that $R_0(\{X_i \to x_i \mid 1 \leq i \leq n\} \uplus \phi(w))\downarrow = x$. Either there exists $i_0 \in \{1, \ldots, n\}$ such that $x = x_{i_0}$, and X_i occurs in R_0 (which is not a variable). Thus, by soundness of the saturation (Proposition 1), we have that $(X_{i_0}\sigma)\phi\downarrow = x_{i_0}\sigma = x\sigma$ and $X_{i_0}\sigma <_{\mathsf{exec}} R_0\sigma = R$. Otherwise, x is introduced by $\{\mathsf{w} \to u\} \in \phi(w)$ with w occurring in R_0, i.e. $R_0 = \mathsf{w}$ since f is a seed statement. In such a case, the variable x occurs in an input before, thus we apply Lemma 13. Note that, similarly to the previous case, the hypotheses are satisfied. We obtain that there exists $i_0 \in \{1, \ldots, n\}$ such that $x_{i_0} = x$ and $X_{i_0}\sigma \leq_{\mathsf{exec}} \hat{R}_j\sigma = R_j$ where R_j is the recipe used to fill the j^{th} action which is an input. Thus because $R_j <_{\mathsf{exec}} \mathsf{w}$, we have that $X_{i_0}\sigma <_{\mathsf{exec}} \mathsf{w} = R_0$.

   Finally in both cases we have that there exists i_0 such that $X_{i_0}\sigma <_{\mathsf{exec}} R_0\sigma = R$. If $R \in \mathcal{W}$ we immediately contradict that R is asap w.r.t. b_m and exec. Otherwise we apply Lemma 10 to obtain that $X_{i_0}\sigma <^{b_m}_{\mathsf{exec}} R$ which is a contradiction too. This conclude the proof. □

Theorem 2. *Let* $K = \mathsf{solved}(\mathsf{sat}(K_{\mathsf{init}}(T_0)))$. *Let* $\mathsf{exec} = (T_0; \emptyset; t_0) \xrightarrow{\ell_1,\ldots,\ell_p} (S; \Phi; t)$ *be an execution with input recipes* $R_1, \ldots, R_k$ *forged by* $b_1, \ldots, b_k$ *and such that each* R_j *with* $j \in \{1, \ldots, k\}$ *is asap w.r.t.* b_j *and* exec. *We have that:*

- $\mathsf{r}_{\ell_1,\ldots,\ell_p} \in \mathcal{H}(K)$ *with a proof tree matching* exec *and* $R_1, \ldots, R_k$*;*
- $\mathsf{k}_{u_0}(R, R\phi\downarrow) \in \mathcal{H}(K)$ *with a proof tree matching* exec *and* $R_1, \ldots, R_k$ *whenever* $u_0 = \ell_1, \ldots, \ell_{q-1}$ *for some* $q \in \mathsf{Rcv}(p)$ *and* R *is asap w.r.t.* $b_{|\mathsf{Rcv}(q)|}$ *and* exec.

Proof. Applying Lemma 11 we obtain that that $R_1, \ldots, R_k$ and R are uniform w.r.t. ϕ. Therefore we can apply Theorem 1 to obtain that $\mathsf{r}_{\ell_1,\ldots,\ell_p} \in \mathcal{H}(\mathsf{seed}(T_0))$ and $\mathsf{k}_{u_0}(R, R\phi\downarrow) \in \mathcal{H}(\mathsf{seed}(T_0))$ with proof trees that are uniform and matching with exec using $R_1, \ldots, R_k$ as input recipes. We conclude applying Lemma 16 to obtain that $\mathsf{r}_{\ell_1,\ldots,\ell_p} \in \mathcal{H}(K)$ and $\mathsf{k}_{u_0}(R, R\phi\downarrow) \in \mathcal{H}(K)$ with proof trees that are still matching with exec using $R_1, \ldots, R_k$ as input recipes. □

D Our Procedure

D.1 Existence of an asap execution

Lemma 2. *Let* $\mathsf{exec} = K_0 \xrightarrow{\ell_1,\ldots,\ell_n}_{\mathcal{T}_0} (S; \Phi; t)$ *be an execution. We may assume w.l.o.g. that* exec *involves input recipes* $R_1, \ldots, R_k$ *forged by agents* $b_1, \ldots, b_k$ *and* R_i *is asap w.r.t.* b_i *and* exec *for each* $i \in \{1, \ldots, k\}$.

Proof. We consider that the execution exec is done with input recipes $R_1, \ldots, R_k$, forged by agents $b_1, \ldots, b_k$. We assume that for all $i \in \{1, \ldots, k\}$:

- if $b_i \in \mathcal{M}_0$ then there is no recipe R that can fill the input (i.e. satisfying the domain restrictions and the timing constraints of the input) and such that $R\Phi\downarrow = R_i\Phi\downarrow$ and $R <^{b_i}_{\mathsf{exec}} R_i$;
- if $b_i \notin \mathcal{M}_0$ then there is no recipe R that can fill the input and such that $R\Phi\downarrow = R_i\Phi\downarrow$ and $R <_{\mathsf{exec}} R_i$.

We prove that $R_1, \ldots, R_k$ are asap recipes w.r.t. $b_1, \ldots, b_k$ and exec by induction on the length n of the execution.

Base case: $n = 0$. In such a case, the result is immediate.

Induction step. In such a case, we have

$$\mathsf{exec} = (T; \emptyset; t_0) \xrightarrow{\ell_1,\ldots,\ell_{n-1}}_{\mathcal{T}_0} (S'; \Phi'; t') \xrightarrow{\ell_n}_{\mathcal{T}_0} (S; \Phi; t)$$

together with recipes $R_1, \ldots, R_k$ forged by $b_1, \ldots, b_k$ that satisfy our assumption. We note exec_0 the sub execution from $(T; \emptyset; t_0)$ to $(S'; \Phi'; t')$. We distinguish two cases depending on the action ℓ_n.

Case ℓ_n is not an input. Thanks to our induction hypothesis, we know that R_i $(1 \leq i \leq k)$ is asap w.r.t. b_i and exec_0. Then, we complete this execution exec_0 performing the action ℓ_n, and we obtain $\mathsf{exec} = (T; \emptyset; t_0) \xrightarrow{\ell_1,\ldots,\ell_n}_{\mathcal{T}_0} (S; \Phi; t)$

with input recipes $R_1, \ldots, R_k$ such that R_i is asap w.r.t. b_i and exec_0 for any $i \in \{1, \ldots, k\}$. It remains to show that $R_1, \ldots, R_k$ are still asap when considering the full execution exec. . Let us distinguish two cases:

If $R_i \in \mathcal{W}$: Since the relation $<_{\mathsf{exec}_0}$ induced by exec_0 is the same as the one induced by exec on recipes built using $dom(\Phi')$, we have that R_i is still asap w.r.t. b_i and exec.

If $R_i \notin \mathcal{W}$: for all recipe R such that $R\phi\downarrow = R_i\Phi\downarrow$ we have that if $vars(R) \subseteq dom(\Phi')$ then $R_i \leq^{b_i}_{\mathsf{exec}_0} R$ and thus $R_i \leq^{b_i}_{\mathsf{exec}} R$. Otherwise, we have that there exists a unique $\mathsf{w} \in vars(R) \cap (dom(\Phi) \setminus dom(\Phi'))$ and $\mathsf{time}(\mathsf{w}) = t'$. Such a w corresponds to the handle bound by ℓ_n when ℓ_n is an output action. For any $\mathsf{w}' \in vars(R_i)$, we have that: $\mathsf{time}(\mathsf{w}') + \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}), b_i) \leq \mathsf{time}(\mathsf{w})$, and thus $\mathsf{time}(\mathsf{w}') + \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}), b_i) \leq \mathsf{time}(\mathsf{w}) + \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}'), b_i)$. To conclude it is sufficient to notice that either it is a strict inequality and thus we immediately have that $\mathsf{w}' <^{b_i}_{\mathsf{exec}} \mathsf{w}$ or we have an equality but since w' has been outputted before w in exec we have that $\mathsf{w}' <^{b_i}_{\mathsf{exec}} \mathsf{w}$ too. Finally we have that such a recipe R which contains w can not be smaller than R_i, i.e. $R \not<^{b_i}_{\mathsf{exec}} R_i$.

Note that, we do not change the configurations involved in the execution but only the underlying recipes, thus timing constraints trivially hold for the new execution.

Case ℓ_n is an input, i.e. $\ell_n = (a, \mathsf{in}^z(u))$. Thanks to our induction hypothesis, we know that R_i $(1 \leq i \leq k-1)$ is asap w.r.t. b_i and exec_0. Then, we complete this execution exec_0 performing the action $\ell_n = (a, \mathsf{in}^z(u))$ with recipe R_k forged by b_k, and we obtain

$$\mathsf{exec} = (T; \emptyset; t_0) \xrightarrow{\ell_1, \ldots, \ell_n}_{\mathcal{T}_0} (S; \Phi; t)$$

with input recipes $R_1, \ldots, R_{k-1}, R_k$. First because $\Phi = \Phi'$ we have that $<_{\mathsf{exec}_0}$ and $<^a_{\mathsf{exec}_0}$ for any $a \in \mathcal{A}$, the relations induced by exec_0, are the same as the ones induced by exec. Therefore we have that R_i $(1 \leq i \leq k-1)$ is still asap w.r.t. b_i and exec. To conclude, it remains to establish that R_k is asap w.r.t. b_k and exec.

Following the semantics of the IN rule, we know that there exists $t_b \in \mathbb{R}^+$ such that $t_b \leq t - \mathsf{Dist}_{\mathcal{T}_0}(b_k, a)$ and

- if $b_k \in \mathcal{A}_0 \setminus \mathcal{M}_0$ then $R_k \in \mathcal{W} \uplus \mathcal{N}_{\mathsf{pub}} \uplus \mathbb{R}^+$. In addition, if $R_k = \mathsf{w}$ then $\mathsf{w} \in dom(\lfloor \Phi' \rfloor^{t_b}_{b_k})$;
- if $b_k \in \mathcal{M}_0$ then for all $\mathsf{w} \in vars(R_k)$, there exists $c \in \mathcal{A}_0$ such that $\mathsf{w} \in dom(\lfloor \Phi' \rfloor^{t_b - \mathsf{Dist}_{\mathcal{T}_0}(c, b_k)}_{c})$.

Let us assume that R_k is not asap w.r.t. b_k and exec.

- Case $b_k \in \mathcal{M}_0$. We have that $R_k \notin \mathcal{N}_{\mathsf{pub}} \cup \mathbb{R}^+$ and there exists R'_k such that $R'_k\Phi\downarrow = R_k\Phi\downarrow$ and either $R'_k <_{\mathsf{exec}} R_k$ or $R'_k <^{b_k}_{\mathsf{exec}} R_k$. Applying Lemma 10 we obtain that in both cases $R'_k <^{b_k}_{\mathsf{exec}} R_k$ and thus $\mathsf{multi}_{\mathcal{W}}(R'_k) \leq^{b_k}_{\mathsf{exec}} \mathsf{multi}_{\mathcal{W}}(R_k)$. By definition of the multiset order, we know that for all $\mathsf{w}' \in vars(R'_k)$, there exists $\mathsf{w} \in vars(R_k)$ such that $\mathsf{w}' \leq^{b_k}_{\mathsf{exec}} \mathsf{w}$, and thus

$$\mathsf{time}(\mathsf{w}') + \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}'), b_k) \leq \mathsf{time}(\mathsf{w}) + \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}), b_k).$$

Since we have that $\mathsf{w} \in dom(\lfloor \Phi' \rfloor^{t_b - \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}), b_k)}_{\mathsf{agent}(\mathsf{w})})$, we know that $\mathsf{time}(\mathsf{w}) \leq t_b - \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}), b_k)$. Therefore, we have that for all $\mathsf{w}' \in vars(R'_k)$, there exists $\mathsf{w} \in vars(R_k)$ such that:

$$\begin{array}{c}\mathsf{time}(\mathsf{w}') + \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}'), b_k) \leq t_b - \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}), b_k) + \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}), b_k) \\ \Rightarrow \mathsf{time}(\mathsf{w}') \leq t_b - \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}'), b_k)\end{array}$$

and hence the IN rule can be executed using R'_k forged by b_k at time t_b. This contradicts the assumption on R_k.

– Case $b_k \in \mathcal{A}_0 \setminus \mathcal{M}_0$. In that case, $R_k \in \mathcal{W} \cup \mathcal{N}_{\mathsf{pub}} \cup \mathbb{R}^+$ and since R_k is not asap, we know that there exists $R'_k <_{\mathsf{exec}} R_k$ such that $R'_k\Phi'{\downarrow} = R_k\Phi'{\downarrow}$. We consider the chain $R'_k <_{\mathsf{exec}} \ldots <_{\mathsf{exec}} R_k$ (each step corresponding to a step of $<^{\mathsf{in}}_{\mathsf{exec}}$ or $<^{\mathsf{sub}}_{\mathsf{exec}}$ under a context) witnessing the fact that $R'_k <_{\mathsf{exec}} R_k$. Let $\mathsf{w}' \in \mathcal{W}$ be the smallest variable w.r.t. $<^{\mathsf{in}}_{\mathsf{exec}}$ such that $\mathsf{w}' <^{\mathsf{in}}_{\mathsf{exec}} \ldots \mathsf{w}'' <^{\mathsf{in}}_{\mathsf{exec}} \mathsf{w}$. In case such a w' does not exist, we consider that $\mathsf{w}' = \mathsf{w}$.
We show, by induction on the length l of $\mathsf{w}' <^{\mathsf{in}}_{\mathsf{exec}} \ldots \mathsf{w}'' <^{\mathsf{in}}_{\mathsf{exec}} \mathsf{w}$ that if $\mathsf{w} \in dom(\lfloor \Phi' \rfloor^{t_1}_{b_1})$ for some b_1 and t_1 then $\mathsf{w}' \in dom(\lfloor \Phi' \rfloor^{t_b - \mathsf{Dist}_{\mathcal{T}_0}(b_2, b_1)}_{b_2})$ for some b_2. Indeed, if $l = 0$ then choosing $b_2 = b_1$, we immediately conclude. Otherwise, since w is outputted at time t_1 (at least) by b_1 then the input recipe w'' has been built by some agent b'' at time $t''_b \leq t_1 - \mathsf{Dist}_{\mathcal{T}_0}(b'', b_1)$. We have that $\mathsf{w}'' \in dom(\lfloor \Phi' \rfloor^{t_1 - \mathsf{Dist}_{\mathcal{T}_0}(b'', b_1)}_{b''})$. We apply the induction hypothesis using $t_1 - \mathsf{Dist}_{\mathcal{T}_0}(b'', b_1)$, w'' and b'' to obtain that $\mathsf{w}' \in dom(\lfloor \Phi' \rfloor^{t_1 - \mathsf{Dist}_{\mathcal{T}_0}(b'', b_1) - \mathsf{Dist}_{\mathcal{T}_0}(b_2, b'')}_{b_2})$ for some b_2. Therefore we have that $\mathsf{w}' \in dom(\lfloor \Phi' \rfloor^{t_1 - \mathsf{Dist}_{\mathcal{T}_0}(b_2, b_1)}_{b_2})$.
Applying this property to $\mathsf{w}' <^{\mathsf{in}}_{\mathsf{exec}} \ldots \mathsf{w}'' <^{\mathsf{in}}_{\mathsf{exec}} \mathsf{w}$, t_b and b_k, we obtain that $\mathsf{w}' \in dom(\lfloor \Phi' \rfloor^{t_b - \mathsf{Dist}_{\mathcal{T}_0}(b', b_k)}_{b'})$ for some b'. Therefore, because the message u is received by the agent a at time t' and u is forged by b_k at time t_b, we have that:

$$\begin{array}{rl} & t_b \leq t' - \mathsf{Dist}_{\mathcal{T}_0}(b_k, a) \\ \Rightarrow & t_b - \mathsf{Dist}_{\mathcal{T}_0}(b', b_k) \leq t' - (\mathsf{Dist}_{\mathcal{T}_0}(b_k, a) + \mathsf{Dist}_{\mathcal{T}_0}(b', b_k)) \\ \Rightarrow & t_b - \mathsf{Dist}_{\mathcal{T}_0}(b', b_k) \leq t' - \mathsf{Dist}_{\mathcal{T}_0}(b', a)\end{array}$$

If $R'_k = \mathsf{w}'$ then this last inequality give us that the rule IN can be triggered with the recipe w' considering the output is performed by b' at time $t_b - \mathsf{Dist}_{\mathcal{T}_0}(b', b_k)$. This contradicts that R_k is asap w.r.t. b_k and exec.
Otherwise, we have that there exists $R'' \notin \mathcal{W}$ such that $R'_k <_{\mathsf{exec}} R'' <^{\mathsf{in}}_{\mathsf{exec}} \mathsf{w}'$. This input received by b' (the same agent as the one who sent w') has been built by some b'' at time t''_b. Since R'', built by b'', is received by b' before outputting w' (available in b' at time $t_b - \mathsf{Dist}_{\mathcal{T}_0}(b', b_k)$), we have that $t''_b + \mathsf{Dist}_{\mathcal{T}_0}(b'', b') \leq t_b - \mathsf{Dist}_{\mathcal{T}_0}(b', b_k)$. Moreover, we have that for all $\mathsf{w} \in vars(R'')$, there exists $c \in \mathcal{A}_0$ such that $\mathsf{w} \in dom(\lfloor \Phi' \rfloor^{t''_b - \mathsf{Dist}_{\mathcal{T}_0}(c, b'')}_{c})$. Thanks to Lemma 10 we have that $R'_k <^{b''}_{\mathsf{exec}} R''$ and thus $\mathsf{multi}_{\mathcal{W}}(R'_k) \leq^{b''}_{\mathsf{exec}} \mathsf{multi}_{\mathcal{W}}(R'')$. Therefore, for all $\mathsf{w}_k \in vars(R'_k)$, there exists $\mathsf{w}'' \in vars(R'')$ such that $\mathsf{w}_k \leq^{b''}_{\mathsf{exec}} \mathsf{w}''$. We thus have that $\mathsf{time}(\mathsf{w}_k) + \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}_k), b'') \leq$

$\mathsf{time}(\mathsf{w}'')+\mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}''), b'')$ and thus $\mathsf{w}_k \in dom(\lfloor \Phi' \rfloor^{t_b''-\mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}_k),b'')}_{\mathsf{agent}(\mathsf{w}_k)})$ because $\mathsf{time}(\mathsf{w}'') \leq t_b'' - \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}''), b'')$.
This allows us to obtain that the rule IN can be triggered with recipe $R_k' \notin \mathcal{W}$ considering the message is built by b'' and time t_b''. Indeed, we have that:

$$\begin{array}{rl} & t_b \leq t' - \mathsf{Dist}_{\mathcal{T}_0}(b_k, a) \\ \Rightarrow & t_b - \mathsf{Dist}_{\mathcal{T}_0}(b_k, b') \leq t' - \mathsf{Dist}_{\mathcal{T}_0}(b', a) \text{ as before} \\ \Rightarrow & t_b'' + \mathsf{Dist}_{\mathcal{T}_0}(b'', b') \leq t' - \mathsf{Dist}_{\mathcal{T}_0}(b', a) \\ \Rightarrow & t_b'' \leq t' - \mathsf{Dist}_{\mathcal{T}_0}(b'', a). \end{array}$$

This contradicts the assumption on R_k because $R_k' <_{\mathsf{exec}} R_k$.

In conclusion, in all cases we obtain a contradiction with the initial assumption. Therefore we conclude that R_k is asap w.r.t. b_k and exec. □

D.2 Main theorem

We consider a trace T locally closed w.r.t. $\mathcal{X}$ and $\mathcal{Z}$. Before proving the main result, we establish the following lemma allowing us to apply saturation on a finite set of seed statements.

Lemma 1. *Let $\mathcal{C}_T$ be the finite set of public names and real numbers occurring in T, and $\mathcal{C}_{all} = \mathcal{N}_{\mathsf{pub}} \cup \mathbb{R}^+$. We have that:*

$$\mathsf{sat}(K_{\mathsf{init}}(\mathsf{seed}(T, \mathcal{C}_{all}))) = \mathsf{sat}(K_{\mathsf{init}}(\mathsf{seed}(T, \mathcal{C}_T))) \cup \{\mathsf{k}_{\mathsf{y}}(c, c) \Leftarrow \mid \ c \in \mathcal{C}_{all}\}.$$

Proof. Let $K_{\mathsf{ext}} = \{\mathsf{k}_{\mathsf{y}}(c, c) \Leftarrow \mid c \in (\mathcal{N}_{\mathsf{pub}} \cup \mathbb{R}^+) \setminus \mathcal{C}_T\}$.

We have that $K_{\mathsf{init}}(\mathsf{seed}(T, \mathcal{N}_{\mathsf{pub}} \cup \mathbb{R}^+)) = K_{\mathsf{init}}(\mathsf{seed}(T, \mathcal{C}_T)) \uplus K_{\mathsf{ext}}$. Then, it is quite easy to see that the resolution steps performed starting with the set $K_{\mathsf{init}}(\mathsf{seed}(T, \mathcal{N}_{\mathsf{pub}} \cup \mathbb{R}^+))$ do not involved statements in K_{ext}, and thus all these steps can also be performed starting with $K_{\mathsf{init}}(\mathsf{seed}(T, \mathcal{C}_T))$ leading to the expected result. □

Theorem 3. *Let $\mathcal{C}_T \subseteq \mathcal{N}_{\mathsf{pub}} \uplus \mathbb{R}^+$ be the finite set of public names and real numbers occurring in T. Let $K = \mathsf{solved}(\mathsf{sat}(K_{\mathsf{init}}(\mathsf{seed}(T, \mathcal{C}_T))))$. We have that:*

- *if* Reachability$(K, t_0, \mathcal{T}_0)$ *holds then $(T; \emptyset; t)$ is executable in $\mathcal{T}_0$;*
- *if $(T; \emptyset; t_0)$ is executable in $\mathcal{T}_0$ then* Reachability$(K, t_0, \mathcal{T}_0)$ *holds.*

Proof. We prove each item separately.

First, we assume that our algorithm returns true when considering $\ell_1' \dots \ell_n'$ and recipes $L_{i_1}, \dots, L_{i_n}$. Actually, soundness of our saturation procedure (Proposition 1) gives us that $(T; \emptyset) \overset{\ell_1'\rho \dots \ell_n'\rho}{\rightsquigarrow} (\epsilon; \phi)$ (relaxed semantics) using recipes $L_{i_1}, \dots, L_{i_n}$. Then, the formula ψ will gather all the timing constraints that have to be satisfied, in particular those to ensure that messages needed when performing the computation L_{i_j} are available in time. Thus, satisfiability of ψ gives us that $(T; \emptyset; t_0)$ is executable in $\mathcal{T}_0$. This concludes the proof for the first item.

Now, we assume that $(T;\emptyset;t_0)$ is executable in $\mathcal{T}_0$, it follows that there exists an execution exec such that:

$$\mathsf{exec} = (T;\emptyset;t_0) \xrightarrow{\ell_1} (T_1;\Phi_1;t_1) \xrightarrow{\ell_2} \ldots \xrightarrow{\ell_n} (T_n;\Phi_n;t_n) = (\epsilon;\Phi;t)$$

First, thanks to Lemma 2, we can assume w.l.o.g. that there exist recipes $R_1,\ldots,R_k$ forged by agents $b_1,\ldots,b_k$ that can be used as inputs in exec and such that R_j is asap w.r.t. b_j and exec for any $j \in \{1,\ldots,k\}$. Therefore, applying Theorem 2 we obtain that $\mathsf{r}_{\ell_1,\ldots,\ell_n} \in \mathcal{H}(K)$ with a proof tree π matching with exec using $R_1,\ldots,R_k$ as input recipes.

We deduce that π ends with a solved statement

$$h = \mathsf{r}_{\ell'_1,\ldots,\ell'_n} \Leftarrow \mathsf{k}_{w_1}(Y_1,y_1),\ldots,\mathsf{k}_{w_m}(Y_m,y_m) \in K$$

instantiated with a substitution τ such that $(\ell'_1,\ldots,\ell'_n)\tau = (\ell_1,\ldots,\ell_n)$. Moreover, because π matches with exec with input recipes $R_1,\ldots,R_k$, there exist recipes $\hat{R}_1,\ldots,\hat{R}_k$ such that:

- $\hat{R}_j(\{Y_i \to y_i \mid 1 \le i \le m\} \uplus \phi(\ell'_1,\ldots,\ell'_n))\downarrow = v'_j$ for $j \in \{1,\ldots,k\}$.
- $\hat{R}_j\tau = R_j$ for $j \in \{1,\ldots,k\}$

where $v'_1,\ldots,v'_k$ are the terms occurring in input in $\ell'_1,\ldots,\ell'_n$.

This statement h is the one that will be considered in our procedure but we first have to ground it it with public constants.

We define ρ as follows:

- $\rho_1 : vars(\ell'_1,\ldots,\ell'_n) \to \{c_1,\ldots,c_q\}$ is a bijection
- $\rho_2 : \{Y_1,\ldots,Y_m\}$ is such that $Y_i\rho_2 = y_i\rho_1$ for all $i \in \{1,\ldots m\}$
- $\rho = \rho_1 \cup \rho_2$

First we can note that applying Proposition 1 (soundness of the saturation) we obtain that $\mathsf{exec}_\rho = (T;\emptyset) \overset{\ell'_1\rho}{\rightsquigarrow} \ldots \overset{\ell'_n\rho}{\rightsquigarrow} (\epsilon;\varphi_\rho)$ using $\hat{R}_1\rho,\ldots,\hat{R}_k\rho$ as input recipes. Actually we have that this trace can be executed in the timed model since $\hat{R}_j\tau = R_j$ and thus the timing constraints induced by $\hat{R}_j\rho$ are less restrictive than those induced by R_j. Hence we have:

$$\mathsf{exec}_\rho^{\mathsf{timed}} = (T;\emptyset;t_0) \xrightarrow{\ell'_1\rho} \ldots \xrightarrow{\ell'_n\rho} (\epsilon;\varphi_\rho;t)$$

using $\hat{R}_1\rho,\ldots,\hat{R}_k\rho$ as input recipes.

To conclude we have to show that our procedure will consider recipes that can fill the inputs and satisfy the timing constraints. Applying Lemma 2 we know that there exist recipes $\hat{S}_1,\ldots,\hat{S}_k$ and agents $b'_1,\ldots,b'_k$ such that for each $j \in \{1,\ldots,k\}$, $\hat{S}_j$ is asap w.r.t. b'_j and $\mathsf{exec}_\rho^{\mathsf{timed}}$ and such that $\mathsf{exec}_\rho^{\mathsf{timed}}$ with input recipes $\hat{S}_1,\ldots,\hat{S}_k$ forged by $b'_1,\ldots,b'_k$ is an execution. We conclude the proof applying Theorem 2 (item 2) to obtain that the recipes $\hat{S}_j$ will be considered in our procedure.

□