A Novel Approach for Reasoning about Liveness in Cryptographic Protocols and its Application to Fair Exchange

Michael Backes, Jannik Dreier, Steve Kremer and Robert Künnemann

Abstract

In this paper, we provide the first methodology for reasoning about liveness properties of cryptographic protocols in a machine-assisted manner without imposing any artificial, finite bounds on the protocols and execution models. To this end, we design an extension of the SAPiC process calculus so that it supports key concepts for stating and reasoning about liveness properties, along with a corresponding translation into the formalism of multiset rewriting that the state-of-the-art theorem prover Tamarin relies upon. We prove that this translation is sound and complete and can thereby automatically generate sound Tamarin specifications and automate the protocol analysis.

Second, we applied our methodology to two widely investigated fair exchange protocols – ASW and GJM – and to the Secure Conversation Protocol standard for industrial control systems, deployed by major players such as Siemens, SAP and ABB. For the fair exchange protocols, we not only re-discovered known attacks, but also uncovered novel attacks that previous analyses based on finite models and a restricted number of sessions did not detect. We suggest fixed versions of these protocols for which we prove both fairness and timeliness, yielding the first automated proofs for fair exchange protocols that rely on a general model without restricting the number of sessions and message size. For the Secure Conversation Protocol, we prove several strong security properties that are vital for the safety of industrial systems, in particular that all messages (e.g., commands) are eventually delivered in order.

1 Introduction

The security properties of cryptographic protocols are commonly formalized in terms of trace properties (*"all protocol traces are secure"*) or in terms of indistinguishability (*"the adversary cannot distinguish two protocol executions"*). Trace properties can be further partitioned into safety properties (*"bad things do not happen"*) and liveness properties (*"eventually, good things happen"*) [23].

An impressive number of symbolic analysis tools have been developed for automated reasoning about trace properties such as authentication and weak forms of confidentiality [26, 3, 14, 13, 29], and, more recently, also about indistinguishability properties such as anonymity and strong forms of confidentiality [6, 5, 10]. Reasoning about the class of *liveness* properties of cryptographic protocols, has, however, received considerably less attention in the literature, even though this class is vital for many security-sensitive applications. Fair exchange protocols arguably constitute the most widely known of these applications. These protocols ensure that both participants eventually reach a state that is fair, e.g., either they both receive a desired item, or no one does [4]. Further examples comprise self-healing schemes that ensure that the system eventually returns to a safe state (e.g., by providing a revocation API that ensures that compromised keys will be turned invalid [12]) and the Secure Conversation Protocol (SCP) – a security layer for industrial control systems specified in the United Architecture (UA) standard [25] that strives to ensure that all messages will eventually be received in the correct order.

While a few previous attempts on the machine-assisted verification of liveness properties exist (see the section on related work for more details), they all face one of the following two conceptual limitations.

First, the bulk of these attempts are restricted to *finite models* and hence fail to adequately model the variety of interactions that might arise in a protocol execution in concurrent environments. So far, only a bounded (typically one or two) number of concurrent sessions was considered, sometimes with

an additional bound on the size of messages. Therefore, these approaches may miss common attacks that rely on interweaving different sessions, and even more so in cases of bounded message size.

Second, the only automated security protocol verification tool that is capable of expressing liveness properties without imposing such finite bounds onto the model is the Tamarin prover [28, 29]. The protocol description language of Tamarin relies on multiset rewriting, which constitutes a rich formalism for expressing desired protocols but offers a low level of abstraction. For instance, there is no abstract notion of a local computation, of individual protocol parties, or of the communication between two parties. This lack of a suitable abstraction layer makes it cumbersome and error-prone to express common assumptions imposed on virtually all existing protocols: certain messages will be eventually delivered; honest protocol parties do not stall computation; and the existence of a mechanism that prevents protocol participants from waiting indefinitely for message delivery. Stating these assumptions is a key requirement for liveness properties in practice. Using an illustrative analogy, this is akin to stating a condition on, and reasoning about, objects within assembly code, compared to using a higher-level language instead. A potential remedy to overcome these limitations would be to provide a sound and complete embedding from a suitable higher-level language into the formalism of multiset rewriting. This would in particular leverage the impressive potential of Tamarin. However, this task contains formidable research challenges and no such prior work exists.

1.1 Our Contribution

This paper makes the following two main contributions: (i) the first methodology for reasoning about liveness properties in a machine-assisted manner without imposing any finite bounds on the models and (ii) an application of this methodology for verifying liveness properties in fair exchange protocols and in a standardized communication protocol for industrial control systems, discovering novel attacks and proving fixed versions secure.

Methodology for reasoning about liveness properties

We present the first verification toolchain for cryptographic protocols that can handle liveness properties such as fairness without the need to bound the number of sessions or the message size. To this end, we have designed an extension of the SAPiC process calculus [18, 19] such that it supports key elements for stating and reasoning about liveness, see below, along with a corresponding translation into multiset rewriting as used by Tamarin. We prove the correctness of the translation, retaining the completeness and soundness of both SAPiC and Tamarin.

Our extension of SAPiC in particular supports three important concepts: *non-deterministic choice*, *local progress* and *resilient channels*. First, extending the calculus with (external) non-deterministic choice (NDC) is essential to model scenarios in which a participant needs to either continue the main protocol or execute one of the sub-protocols. As a technical complication, we stress that, unlike internal NDC, external NDC cannot be encoded using private channels. Second, local progress ensures that honest participants execute their protocol as far as possible, which differs from executions in traditional symbolic models. This is a key aspect for reasoning about liveness properties of the form "on all traces we eventually reach a state such that ..." in a meaningful way, as we need to discard partial traces where the participant would merely stop. Finally, resilient channels ensure that a message will eventually be delivered, but may be delayed by the attacker for an arbitrary amount of time. For fair exchange protocols, the impossibility of achieving fair exchange without a TTP [15] implies that a reliable channels between the protocol participants and the trusted third party (TTP) are strictly necessary.

We have implemented all our extensions in the SAPiC/Tamarin toolchain [18, 19]. The undecidability of the underlying problem ensures that we cannot expect guaranteed termination of our tool. However, the underlying Tamarin prover allows to switch to interactive mode or to additionally specify lemmas, which are proved automatically, and may guide the tool. Interestingly, in our case studies, only one such additional lemma was needed and all our proofs are fully automatic. Implementing local progress turned out to be surprisingly involved, as it interacts with branching and operators for NDC that are possibly nested. Given a position in the current process, the next position the process needs

to progress to is neither unique nor is there a dedicated set of next positions: instead, a propositional formula describes which positions have to be reached. Fortunately, we are able to treat the original, comprehensive correctness proof for SAPiC [19] in a blackbox manner, making the argument more conveniently accessible and less prone to human error.

Application to liveness-sensitive protocols

We have used our methodology and tool for modelling and analyzing two widely investigated fair exchange / contract-signing protocols: ASW [4] and GJM [16]. Moreover, we investigated a toy example for motivating the need of the so-called timeliness property. Our analyses not only rediscovered a known attack on the ASW protocol, first found by Shmatikov and Mitchell [31], but it additionally discovered a thus far unknown variant of their attack. Whether these findings should be considered attacks has been subject to discussions in the literature already though, since they strongly depend on what precisely should be considered a formal contract. If one slightly relaxes the notion of what constitutes a contract, we are able to prove both the fairness and timeliness properties of these protocols. To the best of our knowledge, this yields the first automated proof of fair exchange protocols that considers a general model without restriction on the number of sessions and message size. We finally show that our methodology and tool is applicable beyond fair exchange protocols by investigating the Secure Conversation Protocol standard [25] for industrial control systems, employed, e.g., by Siemens, SAP and ABB. We formally prove several strong security properties for this protocol that are vital for the safety of industrial systems, pertaining to the content, order, and number of messages (e.g., commands) transmitted during communication.

1.2 Related work

Cederquist and Torabi Dashti [7] present a first symbolic model with support for liveness properties. Their model formalizes the *resilient communication channel* assumption, by means of a fairness[1] constraint: if a given message delivery is enabled infinitely often, the message must eventually be delivered. However, they use the general purpose algebra μCRL which does not have built-in support for a symbolic, Dolev-Yao-style adversary; moreover, it does not provide tool support for infinite state systems which would be necessary to analyse protocols without bounding message size or the number of sessions.

Liveness properties arise naturally in the study of optimistic fair exchange protocols (see [20] for a survey) which have been subject to many attempts of formal analyses.

Optimistic fair exchange protocols have been analysed through complex hand proofs in general settings [8, 11, 30], but these proofs have not been machine-checked. In [31], Shmatikov and Mitchell analyse the ASW and GJM protocol in the finite-state model checker Murϕ [31], using an ad-hoc encoding of processes in terms of finite-state machines. As they use a finite-state tool their model requires to bound both the message size and the number of sessions. They define fairness as a state invariant that must hold in any final state. However, they do not verify any liveness property, such as timeliness. In their analysis, they discovered a potential attack related to the precise definition of what is a contract. In this work, we were able to find a similar attack on their "fixed" protocol, which was surprisingly overlooked in their model. Kremer et al. use another finite model checker, Mocha, to analyse several fair exchange [21], contract signing [22] and multi-party contract signing [9]. They model resilient channels as fairness constraints and are able to check liveness properties. Gürgens and Rudolph [17] use the finite-state model checker SHVT to find new flaws in some fair non-repudiation protocols (flaws that were outside of the model of previous analyses). This illustrates that finite model checkers may easily miss attacks, and that there is consequently a need for a general model that is capable of reasoning about liveness properties without imposing artificial finiteness constraints on the underlying model.

[1] Fairness in this context refers to the fairness property in temporal logic, not to the security property of fair exchange protocols.

2 Preliminaries

Terms and equational theories

As usual in symbolic protocol analysis, we model messages by abstract terms. Therefore, we define an order-sorted term algebra with the sort *msg* and two incomparable subsorts *pub* and *fresh*. For each of these subsorts we assume a countably infinite set of names, *FN* for fresh names and *PN* for public names. Fresh names will be used to model cryptographic keys and nonces while public names model publicly known values. We, furthermore, assume a countably infinite set of variables for each sort s, $\mathcal{V}_s$, and let $\mathcal{V}$ be the union of the set of variables for all sorts. We write $u : s$ when the name or variable u is of sort s. Let Σ be a signature, i.e., a set of function symbols, each with an arity. We write f/n when function symbol f is of arity n. There is a subset $\Sigma_{priv} \subseteq \Sigma$ of *private* function symbols, which cannot be applied by the adversary. We denote by $\mathcal{T}_\Sigma$ the set of well-sorted terms built over Σ, *PN*, *FN* and $\mathcal{V}$. For a term t, we denote by $names(t)$, respectively $vars(t)$ the set of names, respectively variables, appearing in t. The set of ground terms, i.e., terms without variables, is denoted by $\mathcal{M}_\Sigma$. When Σ is fixed or clear from the context, we often omit it and simply write $\mathcal{T}$ for $\mathcal{T}_\Sigma$ and $\mathcal{M}$ for $\mathcal{M}_\Sigma$.

We equip the term algebra with an equational theory E, which is a finite set of equations of the form $M = N$ where $M, N \in \mathcal{T}$. From the equational theory we define the binary relation $=_E$ on terms, which is the smallest equivalence relation containing equations in E that is closed under application of function symbols, bijective renaming of names and substitution of variables by terms of the same sort. Furthermore, we require E to distinguish different fresh names, i. e., $\forall a, b \in FN : a \neq b \Rightarrow a \neq_E b$.

Example 1. *Digital signatures can be modelled using a signature*

$$\Sigma = \{\, sig/2, ver/2, pk/1, sk/1 \,\}$$

and an equational theory defined by

$$ver(sig(m, sk(i)), pk(i)) = m,$$

where i is the identity of a party. If sk is a private function symbol, this gives a very minimalistic model of a public-key infrastructure.

For the remainder of the article, we assume that E refers to some fixed equational theory and that the signature and equational theory always contain symbols and equations for pairing and projection, i.e., $\{\langle .,. \rangle, \mathsf{fst}, \mathsf{snd}\} \subseteq \Sigma$ and equations $\mathsf{fst}(\langle x, y\rangle) = x$ and $\mathsf{snd}(\langle x, y\rangle) = y$ are in E. We will sometimes use $\langle x_1, x_2, \ldots, x_n\rangle$ as a shortcut for $\langle x_1, \langle x_2, \langle \ldots, \langle x_{n-1}, x_n\rangle \ldots\rangle$.

Positions within terms are defined as usual. A position p is a sequence of positive integers and $t|_p$ denotes the subterm of t at position p.

Facts

We also assume an unsorted signature Σ_{fact}, disjoint from Σ. The set of *facts* is defined as

$$\mathcal{F} := \{F(t_1, \ldots, t_k) \mid t_i \in \mathcal{T}_\Sigma, F \in \Sigma_{fact} \text{ of arity } k\}.$$

Facts will be used both to annotate protocols by means of events and to define multiset rewrite rules. We partition the signature Σ_{fact} into *linear* and *persistent* fact symbols. We suppose that Σ_{fact} always contains a persistent, unary symbol $!\mathsf{K}$ and a linear, unary symbol Fr. Given a sequence or set of facts S we denote by $lfacts(S)$ the multiset of all linear facts in S and $pfacts(S)$ the set of all persistent facts in S. By notational convention, facts whose identifier starts with '!' will be persistent. $\mathcal{G}$ denotes the set of ground facts, i.e., the set of facts that does not contain variables. For a fact f we denote by $ginsts(f)$ the set of ground instances of f. This notation is also lifted to sequences and sets of facts as expected.

Predicates

We assume an unsorted signature Σ_{pred} of predicate symbols that is disjoint from Σ and Σ_{fact}. The set of *predicate formulas* is defined as

$$\mathcal{P} := \{pr(t_1, \ldots, t_k) \mid t_i \in \mathcal{T}_\Sigma, pr \in \Sigma_{pred} \text{ of arity } k\}.$$

Predicate formulas may be used to describe branching conditions in protocols. The semantics of a predicate is defined via a first-order formula over atoms of the form $t_1 \approx t_2$, i.e. the grammar for such formulae is

$$\langle \phi \rangle ::= t_1 \approx t_2 \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \exists x.\phi$$

where t_1, t_2 are terms and $x \in \mathcal{V}$. For an n-ary predicate symbol pr, $pr(x_1, ..., x_n)$ is defined by a formula ϕ_{pr} such that $fv(\phi_{pr}) \subseteq x_1, ..., x_n$, where fv denotes the free variables in a formula, i. e., variables $v \in \mathcal{V}$ not bound by $\exists v$. The semantics of the first-order formulae is as usual where we interpret $\approx$ as $=_E$.

Example 2. *Suppose $encSucc \in \Sigma_{pred}$ is a binary predicate symbol. We can define it as follows, so that it allows to check whether a term x_1 was encrypted using a key x_2:*

$$\phi_{encSucc} = \exists m.enc(m, x_2) \approx x_1$$

Substitutions

A substitution σ is a partial function from variables to terms. We suppose that substitutions are well-typed, i.e., they only map variables of sort s to terms of sort s, or of a subsort of s. We denote by $\sigma = \{^{t_1}/_{x_1}, \ldots, ^{t_n}/_{x_n}\}$ the substitution whose domain is $\mathbf{D}(\sigma) = \{x_1, \ldots, x_n\}$ and which maps x_i to t_i. As usual, we homomorphically extend σ to apply to terms and facts, and use a postfix notation to denote its application, e.g., we write $t\sigma$ for the application of σ to the term t. A substitution σ is grounding for a term t if $t\sigma$ is ground.

Sets, sequences and multisets

We write $\mathbb{N}_n$ for the set $\{1, \ldots, n\}$. Given a set S we denote by S^* the set of finite sequences of elements from S and by $S^\#$ the set of finite multisets of elements from S. We use the superscript $^\#$ to annotate usual multiset operations, e.g. $S_1 \cup^\# S_2$ denotes the multiset union of multisets S_1, S_2. Given a multiset S we denote by $set(S)$ the set of elements in S. The sequence consisting of elements $e_1, \ldots, e_n$ will be denoted by $[e_1, \ldots, e_n]$ and the empty sequence is denoted by []. We denote by $|S|$ the length, i.e., the number of elements of the sequence. We use $\cdot$ for the operation of adding an element either to the start or to the end, e.g., $e_1 \cdot [e_2, e_3] = [e_1, e_2, e_3] = [e_1, e_2] \cdot e_3$. Given a sequence S, we denote by $idx(S)$ the set of positions in S, i.e., $\mathbb{N}_n$ when S has n elements, and for $i \in idx(S)$ S_i denotes the ith element of the sequence. Set membership modulo E is denoted by $\in_E$ and defined as $e \in_E S$ iff $\exists e' \in S.\ e' =_E e$. $\subset_E$, $\cup_E$, and $=_E$ are defined for sets in a similar way. Application of substitutions are lifted to sets, sequences and multisets as expected. By abuse of notation we sometimes interpret sequences as sets or multisets; the applied operators should make the implicit cast clear.

Functions

We suppose that functions between terms are interpreted modulo E, i.e., if $x =_E y$ then $f(x) = f(y)$. Given function f we let $f(x) = \bot$ when $x \notin_E \mathbf{D}(x)$. When $f(x) = \bot$ we say that f is undefined for all $y =_E x$. We define the function $f := g[a \mapsto b]$ with $\mathbf{D}(f) = \mathbf{D}(g) \cup_E \{a\}$ as $f(x) := b$ for $x =_E a$ and $f(x) := g(x)$ for $x \neq_E a$.

3 Cryptographic calculus with local progress

We extend the Stateful Applied Pi calculus (SAPiC) [19] adding three necessary ingredients to show fairness in fair exchange protocols:

Local progress: each process needs to be reduced as far as possible. That is, until it is either waiting to receive a message, or until it reaches a replication (as we cannot replicate the process indefinitely).

Resilient channels: There is a resilient channel which guarantees message delivery, i.e., each trace is induced by at least one execution in which all messages sent were delivered.

External non-determinism: Any process $P + Q$ reduces to P' or Q' if either P reduces to P', or Q to Q'. Hence, if either P or Q are able to progress, then $P + Q$ *must* progress.

We first go into the syntax, before we formally define the semantics of this calculus and justify some design decisions around these new constructs.

3.1 Syntax and informal semantics

SAPiC is a variant of the applied pi calculus [1]. In addition to the usual operators for concurrency, replication, communication, and name creation, SAPiC was designed for the analysis of state-based protocols and cryptographic APIs, hence it offers several constructs for reading and updating an explicit global state. For the analysis of fair-exchange protocols, we add constructs for non-deterministic choice and communication on a reliable channel to SAPiC. The resulting grammar for processes is described in Figure 1.

$\langle P,Q \rangle ::= 0$
$\mid P \mid Q$
$\mid P + Q$
$\mid\ !\, P$
$\mid \nu n\colon \mathit{fresh};\ P$
$\mid \mathrm{out}(c, N);\ P \quad (c \in \{\text{'r'}, \text{'c'}\}\colon \mathit{pub})$
$\mid \mathrm{in}(c, N);\ P \quad (c \in \{\text{'r'}, \text{'c'}\}\colon \mathit{pub})$
$\mid$ if *Pred* then P [else Q]
$\mid$ event F ; $P \quad (F \in \mathcal{F})$
$\mid$ insert M,N; P
$\mid$ delete M; P
$\mid$ lookup M as x in P [else Q]
$\mid$ lock M; P
$\mid$ unlock M; P

Figure 1: Syntax, where $M, N \in \mathcal{T}$ and $\mathit{Pred} \in \mathcal{P}$

0 denotes the terminal process. $P \mid Q$ is the parallel execution of processes P and Q and $!P$ the replication of P allowing an unbounded number of sessions in protocol executions. $P + Q$ denotes *external* non-deterministic choice, as discussed above. The construct $\nu n; P$ binds the name $n \in FN$ in P and models the generation of a fresh, random value. The processes $\mathrm{out}(c, N);\ P$ and $\mathrm{in}(c, N);$ P represent the output, respectively input, of message N on channel $c \in \{\text{'c'}, \text{'r'}\}$. There are exactly two channels, one for reliable communication, e.g., between a protocol participant and the trusted third party, and one public channel. Messages on both channels may be intercepted and altered by the adversary, however, the reliable channel guarantees that eventually, the message that was sent arrives. Readers familiar with the applied pi calculus [1] may note that we opted for the possibility of pattern matching in the input construct, rather than merely binding the input to a variable x. The process $\mathsf{if}\ \mathit{Pred}\ \mathsf{then}\ P\ \mathsf{else}\ Q$ will execute P or Q, depending on whether *Pred* holds. For example, if $\mathit{Pred} = \mathit{equal}(M, N)$, and $\phi_{\mathit{equal}} = x_1 \approx x_2$, then $\mathsf{if}\ \mathit{equal}(M, N)\ \mathsf{then}\ P\ \mathsf{else}\ Q$ will execute P if $M =_E N$ and Q otherwise. (In the following, we will use $M = N$ as a short-hand for $\mathit{equal}(M, N)$.) The event construct is merely used for annotating processes and will be useful for stating security properties. For readability, we sometimes omit trailing 0 processes, respectively, else branches that consist of a 0 process.

Note that several semantics would be possible for the non-deterministic choice operator. One possibility would be a purely internal non-deterministic choice, whose semantics would correspond to the following reduction rules: $P_1 + P_2 \to P_i$ $(1 \leq i \leq 2)$. The other possibility, which we choose here, is an external choice whose semantics is defined by the rules

$$\frac{P_i \to Q}{P_1 + P_2 \to Q} \quad (1 \leq i \leq 2)$$

In this version $P_1 + P_2$ may only behave as P_1 or P_2 if the chosen process can indeed reduce, i.e., execute an action. While the external choice does complicate the translation towards Tamarin, this flavour of choice is required for modelling fair exchange protocols as we will illustrate in Example 3 below.

The remaining constructs are used to manipulate state and were introduced with SAPiC [18]. The construct insert M,N binds the value N to a key M. Successive inserts overwrite this binding, the delete M operation "undefines" the binding. The lookup M as x in P else Q allows for retrieving the value associated to M binding it to the variable x in P. If the mapping is undefined for M, the process behaves as Q. The lock and unlock constructs are used to gain or waive exclusive access to a resource M, in the style of Dijkstra's binary semaphores: if a term M has been locked, any subsequent attempt to lock M will be blocked until M has been unlocked. This is essential for writing protocols where parallel processes may read and update a common memory.

Example 3. *The following example models the responder in the ASW contract-signing protocol (cf. Section 8.2), simplified to use pattern m_1 and m_3 to match the first and the third message of the optimistic protocol.*

$$\mathrm{in}(\text{`}c\text{'}, m_1); \mathrm{out}(\text{`}c\text{'}, m_2); \begin{pmatrix} \mathrm{in}(\text{`}c\text{'}, m_3); \mathrm{out}(\text{`}c\text{'}, m_4) \\ +(\mathrm{out}(\text{`}r\text{'}, \langle m_1, m_2 \rangle); \ldots) \end{pmatrix}$$

The responder emits m_2, but is not sure to receive the response m_3, as the originator might be dishonest or the adversary might have intercepted this message. If m_3 arrives, then the process $\mathrm{in}(\text{`}c\text{'}, m_3); \mathrm{out}(\text{`}c\text{'}, m_4)$ *is able to transition to* $\mathrm{out}(\text{`}c\text{'}, m_4)$*. If m_3 does not arrive, the message $\langle m_1, m_2 \rangle$ is transmitted on the reliable channel, to contact the TTP. This example highlights the need for external choice, as opposed to internal choice: with internal choice, the responder could simply move to the first branch* $\mathrm{in}(\text{`}c\text{'}, m_3); \mathrm{out}(\text{`}c\text{'}, m_4)$ *and would then be unable to contact the TTP. This would result in an unsound model. Using external choice, however, moving to the first branch is only possible if m_3 is indeed available for input.*

3.2 Semantics

Frames and deduction

Before giving the formal semantics of SAPiC, we introduce the notions of frame and deduction. A *frame* consists of a set of fresh names $\tilde{n}$ and a substitution σ, and is written $\nu\tilde{n}.\sigma$. Intuitively, a frame represents the sequence of messages that have been observed by an adversary during a protocol execution and secrets $\tilde{n}$ generated by the protocol, a priori unknown to the adversary. Deduction models the capacity of the adversary to compute new messages from the observed ones.

Definition 1 (Deduction)**.** *We define the deduction relation $\nu\tilde{n}.\sigma \vdash t$ as the smallest relation between frames and terms defined by the deduction rules in Figure 2.*

Example 4. *If one key is used to encrypt a second key, then, if the intruder learns the first key, he can deduce the second. For $\tilde{n} = k_1, k_2$ and $\sigma = \{ {}^{senc(k_2,k_1)}/_{x_1}, {}^{k_1}/_{x_2} \}$, $\nu\tilde{n}.\sigma \vdash k_2$, as witnessed by the proof tree given in Figure 3.*

$$\frac{a \in FN \cup PN \quad a \notin \tilde{n}}{\nu\tilde{n}.\sigma \vdash a}\ \textsc{Dname} \qquad \frac{\nu\tilde{n}.\sigma \vdash t \quad t =_E t'}{\nu\tilde{n}.\sigma \vdash t'}\ \textsc{DEq}$$

$$\frac{x \in \mathbf{D}(\sigma)}{\nu\tilde{n}.\sigma \vdash x\sigma}\ \textsc{DFrame} \qquad \frac{\nu\tilde{n}.\sigma \vdash t_1 \cdots \nu\tilde{n}.\sigma \vdash t_n \quad f \in \Sigma^k \setminus \Sigma^k_{priv}}{\nu\tilde{n}.\sigma \vdash f(t_1, \ldots, t_n)}\ \textsc{DAppl}$$

Figure 2: Deduction rules.

$$\dfrac{\dfrac{\dfrac{x_1 \in \mathbf{D}(\sigma)}{\nu\tilde{n}.\sigma \vdash c} \quad \dfrac{x_2 \in \mathbf{D}(\sigma)}{\nu\tilde{n}.\sigma \vdash k_1}}{\nu\tilde{n}.\sigma \vdash sdec(c, k_1)} \quad sdec(c, k_1) =_E k_2}{\nu\tilde{n}.\sigma \vdash k_2}$$

Figure 3: Proof tree witnessing that $\nu\tilde{n}.\sigma \vdash k_2$, where $c = senc(k_2, k_1)$

Operational semantics

We can now define the operational semantics of our calculus. The semantics is defined by a labelled transition relation between process configurations. A *process configuration* is a 6-tuple $(\mathcal{E}, \mathcal{S}, \mathcal{P}, \sigma, \mathcal{L}, \mathcal{U})$ where

- $\mathcal{E} \subseteq FN$ is the set of fresh names generated by the processes;
- $\mathcal{S} : \mathcal{M}_\Sigma \to \mathcal{M}_\Sigma$ is a partial function modeling the store;
- $\mathcal{P}$ is a multiset of ground processes representing the processes executed in parallel;
- σ is a ground substitution modeling the messages output to the environment;
- $\mathcal{L} \subseteq \mathcal{M}_\Sigma$ is the set of currently active locks
- $\mathcal{U} \subseteq \mathcal{M}^\#_\Sigma$ is the multiset of messages pending delivery on the public (resilient) channel

The transition relation is defined by the rules in Figure 4. Transitions are labelled by sets of ground facts. For readability, we omit empty sets and brackets around singletons, i.e., we write $\to$ for $\xrightarrow{\emptyset}$ and $\xrightarrow{f}$ for $\xrightarrow{\{f\}}$. We write $\to^*$ for the reflexive, transitive closure of $\to$ (the transitions that are labelled by the empty sets) and write $\stackrel{f}{\Rightarrow}$ for $\to^* \xrightarrow{f} \to^*$. We can now define the set of traces, i.e., possible executions that a process admits. As we are interested in liveness properties we will only consider the set of *progressing* traces, that is traces that end with a *final* state. Intuitively, a state is final if all messages on resilient channels have been delivered and the process is *blocking*.

Definition 2. *Given a ground process P we define the predicate* blocking *as follows*

$$\mathsf{blocking}(P) \hat{=} \begin{cases} \top & \text{if } P = 0,\ P = !Q \text{ or } P = \mathrm{in}(c, m); Q \\ \mathsf{blocking}(P_1) \wedge \mathsf{blocking}(P_2) & \text{if } P = P_1 + P_2 \\ \bot & \text{otherwise} \end{cases}$$

Intuitively a process will always execute completely, except if it is a replication or when it blocks for external reasons, i.e., it is awaiting input on (the public or the resilient) channel.

Definition 3 (Traces of P)**.** *Given a ground process P we define the* set of progressing traces of P *as*

$$\begin{aligned} traces^{ppi}(P) = \Big\{ (F_1, \ldots, F_n) \mid (\emptyset, \emptyset, \{P\}, \emptyset, \emptyset, \emptyset) \stackrel{F_1}{\Longrightarrow}_* (\mathcal{E}_1, \mathcal{S}_1, \mathcal{P}_1, \sigma_1, \mathcal{L}_1, \mathcal{U}_1) \stackrel{F_2}{\Longrightarrow}_* \\ \ldots \stackrel{F_n}{\Longrightarrow}_* (\mathcal{E}_n, \mathcal{S}_n, \mathcal{P}_n, \sigma_n, \mathcal{L}_n, \mathcal{U}_n) \\ \wedge final(\mathcal{E}_n, \mathcal{S}_n, \mathcal{P}_n, \sigma_n, \mathcal{L}_n, \mathcal{U}_n) \Big\},\ \text{where} \end{aligned}$$

$final(\mathcal{E}_n, \mathcal{S}_n, \mathcal{P}_n, \sigma_n, \mathcal{L}_n, \mathcal{U}_n)$ *iff* $\mathcal{U}_n = \emptyset$ *and* $\mathsf{blocking}(P)$ *for all* $P \in \mathcal{P}$.

Standard operations:

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{0\},\sigma,\mathcal{L},\mathcal{U}) \longrightarrow (\mathcal{E},\mathcal{S},\mathcal{P},\sigma,\mathcal{L},\mathcal{U})$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P|Q\},\sigma,\mathcal{L},\mathcal{U}) \longrightarrow (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P,Q\},\sigma,\mathcal{L},\mathcal{U})$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P+Q\},\sigma,\mathcal{L},\mathcal{U}) \xrightarrow{A} (\mathcal{E}',\mathcal{S}',\mathcal{P}',\sigma',\mathcal{L}',\mathcal{U}')$$
$$\text{if } (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P\},\sigma,\mathcal{L},\mathcal{U})\xrightarrow{A}(\mathcal{E}',\mathcal{S}',\mathcal{P}',\sigma',\mathcal{L}',\mathcal{U}') \text{ or } (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{Q\},\sigma,\mathcal{L},\mathcal{U})\xrightarrow{A}(\mathcal{E}',\mathcal{S}',\mathcal{P}',\sigma',\mathcal{L}',\mathcal{U}')$$
$$\text{where } \mathcal{P}'=\mathcal{P} \text{ or } \mathcal{P}'=\mathcal{P}\cup^{\#}\{\,P'\,\}^{\#} \text{ for some } P'\neq P$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{!P\},\sigma,\mathcal{L},\mathcal{U}) \longrightarrow (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{!P,P\},\sigma,\mathcal{L},\mathcal{U})$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\nu a;P\},\sigma,\mathcal{L},\mathcal{U}) \longrightarrow (\mathcal{E}\cup\{a'\},\mathcal{S},\mathcal{P}\cup^{\#}\{P\{^{a'}/_{a}\}\},\sigma,\mathcal{L},\mathcal{U}) \quad \text{if } a' \text{ is fresh}$$

$$(\mathcal{E},\mathcal{S},\mathcal{P},\sigma,\mathcal{L},\mathcal{U}) \xrightarrow{K(M)} (\mathcal{E},\mathcal{S},\mathcal{P},\sigma,\mathcal{L},\mathcal{U}) \quad \text{if } \nu\mathcal{E}.\sigma\vdash M$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\text{out}(\text{`r'},N);P\},\sigma,\mathcal{L},\mathcal{U}) \to (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P\},\sigma\cup\{^{N}/_{x}\},\mathcal{L},\mathcal{U}\cup^{\#}\{\,N\,\}^{\#}) \quad \text{if } x \text{ is fresh}$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\text{in}(\text{`r'},N);P\},\sigma,\mathcal{L},\mathcal{U}) \xrightarrow{K(N\tau)} (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P\tau\},\sigma,\mathcal{L},\mathcal{U}) \quad \text{if } \exists\tau.\ \tau \text{ is grounding for } N, \nu\mathcal{E}.\sigma\vdash N\tau$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\text{in}(\text{`r'},N);P\},\sigma,\mathcal{L},\mathcal{U}) \xrightarrow{K(N\tau)} (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P\tau\},\sigma,\mathcal{L},\mathcal{U}\setminus^{\#}\{\,N'\,\}^{\#}) \quad \text{if } \exists N',\tau.\ N'\in\mathcal{U}, \tau \text{ is grounding for } N, N\tau=_E N'$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\text{out}(\text{`c'},N);P\},\sigma,\mathcal{L},\mathcal{U}) \xrightarrow{K(\text{`c'})} (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P\},\sigma\cup\{^{N}/_{x}\},\mathcal{L},\mathcal{U}) \quad \text{if } x \text{ is fresh}$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\text{in}(\text{`c'},N);P\},\sigma,\mathcal{L},\mathcal{U}) \xrightarrow{K(\langle\text{`c'},N\tau\rangle)} (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P\tau\},\sigma,\mathcal{L},\mathcal{U}) \quad \text{if } \exists\tau.\ \tau \text{ is grounding for } N, \nu\mathcal{E}.\sigma\vdash N\tau$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup\{\text{if } pr(M_1,\ldots,M_n) \text{ then } P \text{ else } Q\},\sigma,\mathcal{L},\mathcal{U}) \longrightarrow (\mathcal{E},\mathcal{S},\mathcal{P}\cup\{P\},\sigma,\mathcal{L},\mathcal{U}) \quad \text{if } \phi_{pr}\{^{M_1}/_{x_1},\ldots,^{M_n}/_{x_n}\} \text{ is satisfied}$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup\{\text{if } pr(M_1,\ldots,M_n) \text{ then } P \text{ else } Q\},\sigma,\mathcal{L},\mathcal{U}) \longrightarrow (\mathcal{E},\mathcal{S},\mathcal{P}\cup\{Q\},\sigma,\mathcal{L},\mathcal{U}) \quad \text{if } \phi_{pr}\{^{M_1}/_{x_1},\ldots,^{M_n}/_{x_n}\} \text{ is not satisfied}$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup\{\text{event}(F);\,P\},\sigma,\mathcal{L},\mathcal{U}) \xrightarrow{F} (\mathcal{E},\mathcal{S},\mathcal{P}\cup\{P\},\sigma,\mathcal{L},\mathcal{U})$$

Operations on global state:

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\text{insert } M,N;\,P\},\sigma,\mathcal{L},\mathcal{U}) \longrightarrow (\mathcal{E},\mathcal{S}[M\mapsto N],\mathcal{P}\cup^{\#}\{P\},\sigma,\mathcal{L},\mathcal{U})$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\text{delete } M;\,P\},\sigma,\mathcal{L},\mathcal{U}) \longrightarrow (\mathcal{E},\mathcal{S}[M\mapsto\bot],\mathcal{P}\cup^{\#}\{P\},\sigma,\mathcal{L},\mathcal{U})$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\text{lookup } M \text{ as } x \text{ in } P \text{ else } Q\,\},\sigma,\mathcal{L},\mathcal{U}) \longrightarrow (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P\{V/x\}\},\sigma,\mathcal{L},\mathcal{U}) \quad \text{if } \mathcal{S}(M)=_E V \text{ is defined}$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\text{lookup } M \text{ as } x \text{ in } P \text{ else } Q\,\},\sigma,\mathcal{L},\mathcal{U}) \longrightarrow (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{Q\},\sigma,\mathcal{L},\mathcal{U}) \quad \text{if } \mathcal{S}(M) \text{ is undefined}$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\text{lock } M;\,P\},\sigma,\mathcal{L},\mathcal{U}) \longrightarrow (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P\},\sigma,\mathcal{L}\cup\{\,M\,\},\mathcal{U}) \quad \text{if } M\notin_E\mathcal{L}$$

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\text{unlock } M;\,P\},\sigma,\mathcal{L},\mathcal{U}) \longrightarrow (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P\},\sigma,\mathcal{L}\setminus\{\,M'\mid M'=_E M\,\},\mathcal{U})$$

Figure 4: Operational semantics

3.3 Discussion

Our calculus supports two channels: 'c' is public and not resilient, while 'r' is public and resilient. It is easy to model an arbitrary number of resilient channels by consistently using pattern matching, e.g., one channel per session id *sid* and party A is encoded by using $\text{out}(\text{'r'}, \langle A, sid, m\rangle)$ and $\text{in}(\text{'r'}, \langle A, sid, m\rangle)$ throughout the process modelling A in session *sid*.

Unlike in the original SAPiC [19], we do not support private channels for the following reason. Suppose a process is reduced to $P = \text{out}(s, m); P'$, and s is a supposedly secret channel name. As there is no matching input $\text{in}(s, m)$, internal communication is not an option, and thus, the process can only reduce further if the adversary was able to deduce s. In this case, P should *not* be considered final, since now the secret channel should behave like a public channel. If the adversary was *not* able to deduce s in time, P should not be considered final, as there is no way to progress, and neglecting this trace could potentially miss attacks. Hence, whether P is final would depend on whether s is private, i.e., deducible by the adversary, which would significantly complicate the translation.

Consider the process

$$P''; \text{out}(s, M : pub); \text{event} A.$$

and the property $\exists i. A@i$ for all traces, i.e., all traces contain $A()$. If P'' contains no replication or input, then this property holds true iff. s stays secret, which means it is *not possible* to deduce s from P, which means there exists no trace in the set of traces produced by P that *contains* the deduction of s. But this cannot (directly) be expressed using axioms and lemmas.

4 Labelled multiset rewriting

We now recall the syntax and semantics of labelled multiset rewriting rules, which constitute the input language of the Tamarin tool [28].

Definition 4 (Multiset rewrite rule)**.** *A labelled multiset rewrite rule ri is a triple (l, a, r), $l, a, r \in \mathcal{F}^*$, written $l \;-[\; a \;]\!\to\; r$. We call $l = prems(ri)$ the premises, $a = actions(ri)$ the actions, and $r = conclusions(ri)$ the conclusions of the rule.*

Definition 5 (Labelled multiset rewriting system)**.** *A labelled multiset rewriting system is a set of labelled multiset rewrite rules R, such that each rule $l \;-[\; a \;]\!\to\; r \in R$ satisfies the following conditions:*

- *l, a, r do not contain fresh names and*
- *r does not contain* Fr*-facts.*

A labelled multiset rewriting system is called well-formed, if additionally

- *for each $l' \;-[\; a' \;]\!\to\; r' \in_E ginsts(l \;-[\; a \;]\!\to\; r)$ we have that $\cap_{r''=_E r'} names(r'') \cap FN \subseteq \cap_{l''=_E l'} names(l'') \cap FN$.*

The semantics of the rules is defined by a labelled transition relation.

Definition 6 (Labelled transition relation)**.** *Given a multiset rewriting system R we define the* labeled transition relation $\to_R \subseteq \mathcal{G}^\# \times \mathcal{P}(\mathcal{G}) \times \mathcal{G}^\#$ *as*

$$S \xrightarrow{a}_R ((S \setminus^\# lfacts(l)) \cup^\# r)$$

if and only if $l \;-[\; a \;]\!\to\; r \in_E ginsts(R \cup \text{FRESH})$, $lfacts(l) \subseteq^\# S$ and $pfacts(l) \subseteq S$.

Definition 7 (Executions)**.** *Given a multiset rewriting system R we define its set of executions as*

$$exec^{msr}(R) = \Big\{ \emptyset \xrightarrow{A_1}_R \ldots \xrightarrow{A_n}_R S_n \mid \forall a, i, j : 0 \le i \neq j < n. \\ (S_{i+1} \setminus^\# S_i) = \{\mathsf{Fr}(a)\} \Rightarrow (S_{j+1} \setminus^\# S_j) \neq \{\mathsf{Fr}(a)\} \Big\}$$

The set of executions consists of transition sequences that respect freshness, i. e., for a given name a the fact $\mathsf{Fr}(a)$ is only added once, or in other words the rule FRESH is at most fired once for each name. We define the set of traces in a similar way as for processes.

Definition 8 (Traces)**.** *The set of traces is defined as*

$$traces^{msr}(R) = \Big\{(A_1, \ldots, A_n) \mid \ \forall\, 0 \le i \le n. \\ \emptyset \overset{A_1}{\Longrightarrow}_R \ldots \overset{A_n}{\Longrightarrow}_R S_n \in exec^{msr}(R)\Big\}$$

where $\overset{A}{\Longrightarrow}_R$ *is defined as* $\xrightarrow{\emptyset}{}^*_R \xrightarrow{A}_R \xrightarrow{\emptyset}{}^*_R$ *for* $A \neq \emptyset$.

Note that both for processes and multiset rewrite rules the set of traces is a sequence of sets of facts.

5 Security Properties

In the Tamarin tool [28], security properties are described in an expressive two-sorted first-order logic. The sort *temp* is used for time points, $\mathcal{V}_{temp}$ are the temporal variables.

Definition 9 (Trace formulas)**.** *A trace atom is either false* $\bot$*, a term equality* $t_1 \approx t_2$*, a timepoint ordering* $i \lessdot j$*, a timepoint equality* $i \doteq j$*, or an action* $F@i$ *for a fact* $F \in \mathcal{F}$ *and a timepoint* i*. A trace formula is a first-order formula over trace atoms.*

As we will see in our case studies, this logic is expressive enough to analyze a variety of security properties, including liveness properties.

To define the semantics, let each sort s have a domain $\mathbf{D}(s)$. $\mathbf{D}(temp) = \mathcal{Q}$, $\mathbf{D}(msg) = \mathcal{M}$, $\mathbf{D}(fresh) = FN$, and $\mathbf{D}(pub) = PN$. A function $\theta : \mathcal{V} \to \mathcal{M} \cup \mathcal{Q}$ is a valuation if it respects sorts, i. e., $\theta(\mathcal{V}_s) \subset \mathbf{D}(s)$ for all sorts s. If t is a term, $t\theta$ is the application of the homomorphic extension of θ to t.

Definition 10 (Satisfaction relation)**.** *The satisfaction relation* $(tr, \theta) \vDash \varphi$ *between a trace* tr*, a valuation* θ*, and a trace formula* φ *is defined as follows:*

$(tr, \theta) \vDash \bot$	*never*
$(tr, \theta) \vDash F@i$	$\iff \theta(i) \in idx(tr) \wedge F\theta \in_E tr_{\theta(i)}$
$(tr, \theta) \vDash i \lessdot j$	$\iff \theta(i) < \theta(j)$

$$
\begin{aligned}
(tr,\theta) \vDash i \doteq j &\iff \theta(i) = \theta(j)\\
(tr,\theta) \vDash t_1 \approx t_2 &\iff t_1\theta =_E t_2\theta\\
(tr,\theta) \vDash \neg\varphi &\iff \textit{not } (tr,\theta) \vDash \varphi\\
(tr,\theta) \vDash \varphi_1 \wedge \varphi_2 &\iff (tr,\theta) \vDash \varphi_1 \textit{ and } (tr,\theta) \vDash \varphi_2\\
(tr,\theta) \vDash \exists x : s.\varphi &\iff \textit{there is } u \in \mathbf{D}(s)\\
&\qquad \textit{such that } (tr,\theta[x \mapsto u]) \vDash \varphi.
\end{aligned}
$$

For readability, we define $t_1 \gtrdot t_2$ as $\neg(t_1 \lessdot t_2 \vee t_1 \doteq t_2)$ and $(\trianglelefteq, \not\doteq, \trianglerighteq)$ as expected. We also use classical notational shortcuts such as $t_1 \lessdot t_2 \lessdot t_3$ for $t_1 \lessdot t_2 \wedge t_2 \lessdot t_3$ and $\forall i \leq j.\ \varphi$ for $\forall i.\ i \leq j \rightarrow \varphi$. When φ is a ground formula we sometimes simply write $tr \vDash \varphi$ as the satisfaction of φ is independent of the valuation.

Definition 11 (Validity, satisfiability)**.** *Let $Tr \subseteq (\mathcal{P}(\mathcal{G}))^*$ be a set of traces. A trace formula φ is said to be* valid *for Tr (written $Tr \vDash^\forall \varphi$) if for any trace $tr \in Tr$ and any valuation θ we have that $(tr,\theta) \vDash \varphi$.*

A trace formula φ is said to be satisfiable *for Tr, written $Tr \vDash^\exists \varphi$, if there exist a trace $tr \in Tr$ and a valuation θ such that $(tr,\theta) \vDash \varphi$.*

Note that $Tr \vDash^\forall \varphi$ iff $Tr \nvDash^\exists \neg\varphi$. Given a multiset rewriting system R we say that φ is valid, written $R \vDash^\forall \varphi$, if $traces^{msr}(R) \vDash^\forall \varphi$. We say that φ is satisfied in R, written $R \vDash^\exists \varphi$, if $traces^{msr}(R) \vDash^\exists \varphi$. Similarly, given a ground process P we say that φ is valid, written $P \vDash^\forall \varphi$, if $traces^{ppi}(P) \vDash^\forall \varphi$, and that φ is satisfied in P, written $P \vDash^\exists \varphi$, if $traces^{ppi}(P) \vDash^\exists \varphi$.

Example 5. *In Section 8, the following trace formula is used to express timeliness for the originator, i.e., whenever originator 'a' runs the protocol with 'b' on a contract 't' in session 'sid', unless 'a' will be corrupted at some point, she will eventually reach either a contract in this session or receive notification that it has been aborted.*

$$
\begin{aligned}
&\forall i : temp, a, b, t, sid : msg.\\
&\quad Start_A(a,b,t,sid)@i \Rightarrow (\exists j. Contract_A(a,b,t,sid)@j)\\
&\qquad\qquad\qquad\qquad |(\exists j. Abort_A(a,b,t,sid)@j)\\
&\qquad\qquad\qquad\qquad |(\exists j. Corrupt(a)@j)"
\end{aligned}
$$

6 A translation from processes into multiset rewrite rules

In this section, we define a translation from a process P into a set of multiset rewrite rules $[\![P]\!]$ and a translation on trace formulas such that $P \models^\forall \varphi$ if and only if $[\![P]\!] \models^\forall [\![\varphi]\!]$. Note that the result also holds for satisfiability as an immediate consequence. For a rather expressive subset of trace formulas (see [28] for the exact definition of the fragment), checking whether $[\![P]\!] \models^\forall [\![\varphi]\!]$ can then be discharged to the Tamarin prover that we use as a backend. Except for local progress, resilient channels and NDC, the other elements of the translation have been discussed in previous work [19].

6.1 Progress function

In Section 3, we have defined local progress axiomatically in terms of the final state to be reached. The progress function that we use for our translation gives a more constructive understanding. In this section, we will give an intuition of how this function works and illustrate the subtle interplay between non-deterministic choice and local progress. We postpone the formal definition to Appendix A.

When a process is in a certain position p, the progress function π defines the maximal follow-up positions that the process can reach on its own. All traces that do not reach maximal positions will later be ruled out by the means of an axiom. As we will see below, the progress function maps a position to a set of sets of positions. In the simplest case a position in a process can have a unique position it must progress to.

Example 6. *Let*

$$P = \text{event } A; \text{in}(\text{`}c\text{'}, m); \text{event } B; 0.$$

Initially, the process P must reduce to

$$P|_1 = \text{in}(\text{`}c\text{'}, m); \text{event } B; 0$$

i.e., raise A. However, the process will be blocked as it needs to wait for an input. Once P can be reduced to $P|_{111} = \text{event } B; 0$, e.g., because the adversary sends a message, it must continue to reduce to 0. We would therefore define the progress function for P such that $\pi([]) = \{\{1\}\}$ and $\pi(11) = \{\{111\}\}$

In general, a process needs to progress until it reaches a *blocking* process, as defined in Definition 2. We call a position blocking in P, if $P|_p$ is blocking. If P is clear from context, we only call the position blocking. Note that in case of a non-deterministic choice, the process $P = P_1 + P_2$ is only blocking if both P_1 and P_2 are blocking too. If one of the P_i is non-blocking, the process will progress in that branch. Therefore, progress can never reach the position directly below a $+$. In particular, if NDC operators are nested (which is useful, e.g., to express an n-fold choice), several positions need to be 'jumped over', figuratively speaking.

Moreover, in case of a parallel composition, the progress function expects the process to progress to the next blocking position in each of the parallel branches.

Example 7. *Consider the process*

$$P = (\text{event } A; 0) \mid (\text{event } B; 0)$$

This process is expected to raise both events A and B. Therefore, $\pi([]) = \{\{11\}, \{21\}\}$.

A process $P = P_1 + P_2$ allows to either execute P_1 or P_2 but not both. Unlike parallel composition where P must progress in both branches, we ensure that P processes in either P_1 or P_2.

Example 8. *Consider the process*

$$P = (\text{event } A; 0) + (\text{event } B; 0)$$

To express that P must either raise event A or event B, we define $\pi([]) = \{\{11, 21\}\}$.

We are now ready to explain why the progress function requires to return a set of sets of positions. When

$$\pi(p) = \{A_1, \ldots A_n\},$$

process P must transition p to some position in A_i for each $1 \leq i \leq n$. Intuitively, each A_i corresponds to a parallel branch. The positions in each A_i are the mutually exclusive positions due to the $+$ operator. We hence require that a process at position p executes until it reaches one blocking position for each A_i.

Example 9. *Consider the process*

$$(\text{event } A; 0 \mid \text{event } B; 0) + (\text{event } C; 0 \mid \text{event} D; 0)$$

and denote the leaf position below event A *be called p_A and similarly for other events. We have that $\pi([]) = \{\{p_A, p_C\}, \{p_A, p_D\}, \{p_B, p_C\}, \{p_B, p_D\}\}$*

Example 10. *The following example illustrates the difficulty in defining π due to non-deterministic choice.*

$$\begin{aligned} P = \quad & (\text{in}(\text{`}c\text{'}, m); \text{event } A; 0) + \\ & (\text{out}(\text{`}r\text{'}, r_1); \text{event } B; \text{in}(\text{`}r\text{'}, r_2); \text{event } C; 0) \end{aligned}$$

While the left branch starts with a blocking position, progress is possible in the right branch. As π aims to capture the guarantee that a process will progress until no action is directly available, we have

$$
\begin{array}{rcll}
\mathsf{Out}(x) & \dashv[\ ]\rightarrow & !\mathsf{K}(x) & (\text{MDOut}) \\
!\mathsf{K}(x) & \dashv[\ K(x)\ ]\rightarrow & \mathsf{In}(x) & (\text{MDIn}) \\
 & \dashv[\ ]\rightarrow & !\mathsf{K}(x : pub) & (\text{MDPub}) \\
\mathsf{Fr}(x : fresh) & \dashv[\ ]\rightarrow & !\mathsf{K}(x : fresh) & (\text{MDFresh}) \\
!\mathsf{K}(x_1), \ldots, !\mathsf{K}(x_k) & \dashv[\ ]\rightarrow & !\mathsf{K}(f(x_1, \ldots, x_k)) \text{ for } f \in \Sigma^k & (\text{MDAppl})
\end{array}
$$

Figure 5: The set of rules MD.

to consider two cases: a) If the input in the first branch is not available, P has no choice but to progress to the second branch, which is non-blocking. b) If the input in the first branch is available, then P has to reduce to the next blocking position in the first branch. Thus $\pi([]) = \{\{111, 211\}\}$, and $\pi(2111) = \{\{21111\}\}$. This situation appears in all contract-signing protocols we verify: either a message appears on the public channel and we can proceed to A, or the process eventually gives up and contacts the TTP on the reliable channel. If the TTP is implemented well, then due to local progress and the fact that messages sent on the reliable channel are eventually delivered, the position below event B (2111) can be reached, which triggers progress up to event C.

When π is the progress function for P we will denote by $\mathsf{From}(P)$ the domain of π and $\mathsf{To}(P)$ the set of positions that appear in the image of π. $\mathsf{From}(P)$ is the set of positions where progress starts, roughly, the non-blocking positions that directly follow a blocking position and possibly [] (if [] is non blocking). We also show that for any position $q \in \mathsf{To}(P)$ there is a unique position $p \in \mathsf{From}(P)$ such that q appears in $\pi(p)$ and denote the function that maps q to p by π^{-1}. The formal definitions of these sets and the progress function are rather technical and can be found in the long version.

6.2 Definition of the translation of processes

To model the adversary's message deduction capabilities, we introduce the set of rules MD defined in Figure 5. In order for our translation to be correct, we need to make some assumptions on the set of processes we allow. These assumptions are however, as we will see, rather mild and most of them without loss of generality. First we define a set of reserved variables that will be used in our translation and whose use we therefore forbid in the processes.

Definition 12 (Reserved variables and facts)**.** *The set of reserved variables is defined as the set containing the elements n_a for any $a \in FN$ and $lock_l$ for any $l \in \mathbb{N}$. The set of reserved facts $\mathcal{F}_{res}$ is defined as the set containing facts $f(t_1, \ldots, t_n)$ where $t_1, \ldots, t_n \in \mathcal{T}$ and $f \in \{$ Init, Insert, Delete, IsIn, IsNotSet, state, Lock, Unlock, Out, Fr, In, Msg, ProtoNonce, Event, InEvent, $Pred_{pr}$, $Pred\_not_{pr}$ $\mid pr \in \Sigma_{pred}$ $\}$.*

For our translation to be sound, we require that for each process, there exists an injective mapping assigning to every $\mathsf{unlock}\ t$ in a process a $\mathsf{lock}\ t$ that precedes it in the process' syntax tree. Moreover, given a process $\mathsf{lock}\ t; P$ the corresponding unlock in P shall not be under a parallel or replication. These conditions allow us to annotate each corresponding pair $\mathsf{lock}\ t$, $\mathsf{unlock}\ t$ with a unique label l. The annotated version of a process P is denoted $\overline{P}$. In case the annotation fails, i.e., P violates one of the above conditions, the process $\overline{P}$ contains $\bot$. This is similar to the hypotheses on locks made in StatVerif [2]. They precisely require that:

> *"In every branch of the syntax tree, every lock must be followed by precisely one corresponding unlock. In $\mathsf{lock}; P$, the part of the process P that occurs before the next unlock, if any, may not include parallel, replication, or lock."*

Unlike StatVerif we do not need to forbid nested locks for our results to hold, even though nested locks are not very useful as they directly lead to deadlocks.

Definition 13 (Process annotation)**.** *Given a ground process P we define the annotated ground process $\overline{P}$ as $\mathsf{ap}(P, [])$ where:*

$$\mathsf{ap}(0, A) := 0$$

$$\mathsf{ap}(P|Q, A) := \begin{cases} \mathsf{ap}(P, A)|\mathsf{ap}(Q, A) \text{ if } A = [] \\ \bot \qquad\qquad\qquad \text{otherwise} \end{cases}$$

$$\mathsf{ap}(!P, A) := \begin{cases} !\mathsf{ap}(P, A) \text{ if } A = [] \\ \bot \qquad \text{otherwise} \end{cases}$$

$$\mathsf{ap}(\textit{if } \mathit{Pred} \textit{ then } P \textit{ else } Q, A) := \textit{if } \mathit{Pred} \textit{ then } \mathsf{ap}(P, A) \textit{ else } \mathsf{ap}(Q, A)$$

$$\mathsf{ap}(\textit{lookup } M \textit{ as } x \textit{ in } P \textit{ else } Q, A) := \textit{lookup } M \textit{ as } x \textit{ in } \mathsf{ap}(P, A) \textit{ else } \mathsf{ap}(Q, A)$$

$$\mathsf{ap}(\alpha; P, A) := \alpha; \mathsf{ap}(P, A) \qquad \textit{where } \alpha \notin \{\, \text{lock } t, \text{unlock } t \colon t \in \mathcal{T} \,\}$$

$$\mathsf{ap}(\text{lock } t; P, A) := \text{lock}^l\ t; \mathsf{ap}(P, A \cdot (t, l)) \qquad \textit{where } l \in \mathbb{N} \textit{ is a fresh label}$$

$$\mathsf{ap}(\text{unlock } t; P, A) := \begin{cases} \text{unlock}^l\ t; \mathsf{ap}(P, A \setminus \{(t, l)\}) \textit{ if } \exists i.\ A_i = (t, l) \\ \qquad\qquad\qquad\qquad\qquad \textit{and } \forall l', j < i. A_j \neq (t, l') \\ \qquad\qquad\qquad\qquad\qquad \textit{for } A = (A_0, \ldots, A_m) \\ \bot \qquad\qquad\qquad\qquad\qquad \text{otherwise} \end{cases}$$

Intuitively, the function $\mathsf{ap}(P, A)$ makes a traversal of the process P and maintains the list A of pending unlocks. A pair (l, t) is in A whenever the instruction lock t was encountered, annotated by the label l and no corresponding instruction $\mathsf{unlock}\ t$ was found yet. When encountering an $\mathsf{unlock}\ t$ instruction we annotate it with the first corresponding label that was added to the list. We choose the first occurrence in the list in order to guarantee that the resulting process is uniquely defined. Remark that the Appendix of [18] contains a different but equivalent formulation of this definition.

Definition 14 (well-formed)**.** *A ground process P is well-formed if*

- *no reserved variable nor reserved fact appears in P,*
- *any bound name and variable in P cannot be rebound, i.e., if u is bound in P then u is not under the scope of a previous binder, and*
- *$\overline{P}$ does not contain $\bot$.*

A trace formula φ is well-formed if no reserved variable nor reserved fact appear in φ.

Definition 15. *Given a well-formed ground process P we define the labelled multiset rewriting system $[\![P]\!]$ as*

$$\text{MD} \cup \{\textsc{Init}, \text{MID}\} \cup [\![\overline{P}, [], []]\!]$$

where

- *the rule* Init *is defined as*

$$\textsc{Init} : [\mathsf{Fr}_{[]}] \dashv[\ \text{Init}, \mathit{ProgFrom}_{[]}\ ]\rightarrow [\mathsf{state}_{[]}()],$$

- *the rule* MID *is defined as*

$$\text{MID} : [\mathsf{Fr}(x)] \dashv[\ \ ]\rightarrow [\mathsf{MID}_{\mathsf{rcv}}(x), \mathsf{MID}_{\mathsf{snd}}(x)]$$

- $[\![P, p, \tilde{x}]\!]$ *is defined inductively for process* P*, position* $p \in \mathbb{N}^*$ *and sequence of variables* $\tilde{x}$ *in Figure 6.*

  For brevity, we use the following syntactic shortcuts:

$$ProgFrom_p \hat{=} \begin{cases} ProgFrom_p(prog_p) & \text{if } p \in \mathsf{From}(P) \\ [] & \text{otherwise} \end{cases}$$

$$ProgTo_p \hat{=} \begin{cases} ProgTo_p(prog_{\pi^{-1}(p)}) & \text{if } p \in \mathsf{To}(P) \\ [] & \text{otherwise} \end{cases}$$

$$\mathsf{Fr}_p \hat{=} \begin{cases} \mathsf{Fr}(prog_p) & \text{if } p \in \mathsf{From}(P) \\ [] & \text{otherwise} \end{cases}$$

$$\tilde{x}_p \hat{=} \begin{cases} \tilde{x} \cup^{\#} \{\, prog_p \,\}^{\#} & \text{if } p \in \mathsf{From}(P) \\ \tilde{x} & \text{otherwise} \end{cases}$$

The core of the translation builds on [19]. The message deduction rules MD consist of four rules for message output, message input, application of (non-private) function symbols, and creation of fresh values. In the definition of $[\![P, p, \tilde{x}]\!]$, we intuitively use the family of facts state_p to indicate that the process is currently at position p in its syntax tree. A fact state_p will indeed be true in an execution of these rules whenever some instance of P_p (i.e. the process defined by the subtree at position p of the syntax tree of P) is in the multiset $\mathcal{P}$ of the process configuration.

We will now comment on the main changes w.r.t. [19]. The translation of a NDC does not produce a rule, but rewrites the positions of the required state-fact in the first rules of both its child processes, with the effect that the NDC step is skipped to proceed to either one. Input and output on the resilient channel require the fact $\mathsf{MID}_{\mathsf{snd}}$, respectively $\mathsf{MID}_{\mathsf{rcv}}$, both of which can be instantiated with the rule MID. Each instantiation can be used but once and assures that each message sent has a unique identifier, even if a message with the same content has been sent before. Thus, the axiom α_{resil} can enforce that a message sent must be received, for each instance of a message. Finally, we annotate the rules with *ProgFrom* and *ProgTo* facts. The α_{prog} axiom will use these actions to enforce progress.

6.3 Definition of the translation of trace formulas

A trace formula φ is well-formed if no reserved variable nor a reserved fact appear in φ.

Definition 16. *Given a well-formed trace formula* φ *we define*

$$[\![\varphi]\!]_\forall := \alpha \Rightarrow \varphi \qquad \text{and} \qquad [\![\varphi]\!]_\exists := \alpha \wedge \varphi$$

where α *is defined in Figure 7.*

In the definition of $[\![P, p, \tilde{x}]\!]$ we intuitively use the family of facts state_p to indicate that the process is currently at position p in its syntax tree. A fact state_p will indeed be true in an execution of these rules whenever some instance of P_p (i.e. the process defined by the subtree at position p of the syntax tree of P) is in the multiset $\mathcal{P}$ of the process configuration. The translation of the zero-process, parallel and replication operators merely use state_p-facts. For instance $[\![P \mid Q, p, \tilde{x}]\!]$ defines the rule

$$[\mathsf{state}_p(\tilde{x})] \to [\mathsf{state}_{p\cdot 1}(\tilde{x}), \mathsf{state}_{p\cdot 2}(\tilde{x})]$$

which intuitively states that when a process is at position p (modelled by the fact $\mathsf{state}_p(\tilde{x})$ being true) then the process is allowed to move both to P (putting $\mathsf{state}_{p\cdot 1}(\tilde{x})$ to true) and Q (putting $\mathsf{state}_{p\cdot 2}(\tilde{x})$ to true). The translation of $[\![P \mid Q, p, \tilde{x}]\!]$ also contains the set of rules $[\![P, p \cdot 1, \tilde{x}]\!] \cup [\![Q, p \cdot 2, \tilde{x}]\!]$ expressing that after this transition the process may behave as P and Q, i.e., the processes at positions $p \cdot 1$, respectively $p \cdot 2$, in the process tree. Also note that the translation of $!P$ results in a persistent fact as $!P$ always remains in $\mathcal{P}$. The translation of the construct $\nu\, a$ translates the name a into a variable n_a, as msr rules must not contain fresh names. Any instantiation of this rule will substitute n_a by

$$
\begin{array}{rcl}
[\![0, p, \tilde{x}]\!] & = & \{[\mathsf{state}_p(\tilde{x})] \multimap[\ \ ]\to []\} \\
[\![P \mid Q, p, \tilde{x}]\!] & = & \{[\mathsf{state}_p(\tilde{x})] \multimap[\ \mathit{ProgFrom}_p, \mathit{ProgTo}_{p\cdot 1}, \mathit{ProgTo}_{p\cdot 2}\ ]\to [\mathsf{state}_{p\cdot 1}(\tilde{x}_p), \mathsf{state}_{p\cdot 2}(\tilde{x}_p)]\} \\
 & & \cup [\![P, p\cdot 1, \tilde{x}_p]\!] \cup [\![Q, p\cdot 2, \tilde{x}_p]\!] \\
[\![P + Q, p, \tilde{x}]\!] & = & [\![P, p\cdot 1, \tilde{x}]\!] \left\{ {}^{\mathsf{state}_p(\tilde{x})}/_{\mathsf{state}_{p\cdot 1}(\tilde{x})} \right\} \cup [\![Q, p\cdot 2, \tilde{x}]\!] \left\{ {}^{\mathsf{state}_p(\tilde{x})}/_{\mathsf{state}_{p\cdot 2}(\tilde{x})} \right\} \\
[\![!P, p, \tilde{x}]\!] & = & \{[\mathsf{state}_p(\tilde{x})] \multimap[\ \mathit{ProgTo}_{p\cdot 1}\ ]\to [!\mathsf{state}_p^{\mathsf{semi}}(\tilde{x})], \\
 & & [!\mathsf{state}_p^{\mathsf{semi}}(\tilde{x})] \multimap[\ \ ]\to [\mathsf{state}_{p\cdot 1}(\tilde{x})]\} \cup [\![P, p\cdot 1, \tilde{x}]\!] \\
[\![\nu a; P, p, \tilde{x}]\!] & = & \{[\mathsf{state}_p(\tilde{x}), \mathsf{Fr}(n_a : \mathit{fresh})] \multimap[\ \mathit{ProgFrom}_p, \mathit{ProgTo}_{p\cdot 1}, \mathit{ProtoNonce}(n_a : \mathit{fresh})\ ]\to \\
 & & [\mathsf{state}_{p\cdot 1}(\tilde{x}, n_a : \mathit{fresh})]\} \cup [\![P, p\cdot 1, (\tilde{x}, n_a : \mathit{fresh})]\!] \\
[\![\mathrm{out}(\text{`r'}, m); P, p, \tilde{x}]\!] & = & \{[\mathsf{state}_p(\tilde{x}), \mathsf{MID}_{\mathsf{snd}}(\mathit{mid})] \multimap[\ \mathit{ProgFrom}_p, \mathit{ProgTo}_{p\cdot 1}, \mathrm{Send}(\mathit{mid}, m)\ ]\to \\
 & & [\mathsf{state}_{p\cdot 1}(\tilde{x}, \mathit{mid}), \mathsf{Out}(m)]\} \cup [\![P, p\cdot 1, \tilde{x}]\!] \\
[\![\mathrm{in}(\text{`r'}, m); P, p, \tilde{x}]\!] & = & \{[\mathsf{state}_p(\tilde{x}), \mathsf{In}(m), \mathsf{Fr}_p, \mathsf{MID}_{\mathsf{rcv}}(\mathit{mid})] \multimap[\ \mathit{ProgTo}_{p\cdot 1}, \mathrm{Receive}(\mathit{mid}, m), \mathrm{InEvent}(m)\ ]\to \\
 & & \quad [\mathsf{state}_{p\cdot 1}(\tilde{x}_p \cup \mathit{vars}(m))]\} \\
 & & \cup [\![P, p\cdot 1, \tilde{x} \cup \mathit{vars}(m)]\!] \\
[\![\mathrm{out}(\text{`c'}, N); P, p, \tilde{x}]\!] & = & \{[\mathsf{state}_p(\tilde{x}), \mathsf{In}(\text{`c'})] \multimap[\ \mathit{ProgFrom}_p, \mathit{ProgTo}_{p\cdot 1}, \mathrm{InEvent}(\text{`c'})\ ]\to [\mathsf{state}_{p\cdot 1}(\tilde{x}), \mathsf{Out}(N)]\} \\
 & & \cup [\![P, p\cdot 1, \tilde{x}_p]\!] \\
[\![\mathrm{in}(\text{`c'}, m); P, p, \tilde{x}]\!] & = & \{[\mathsf{state}_p(\tilde{x}), \mathsf{In}(\langle \text{`c'}, m\rangle), \mathsf{Fr}_p] \multimap[\ \mathit{ProgTo}_{p\cdot 1}, \mathrm{InEvent}(\langle \text{`c'}, m\rangle)\ ]\to \\
 & & \quad [\mathsf{state}_{p\cdot 1}(\tilde{x} \cup \mathit{vars}(m))]\} \\
 & & \cup [\![P, p\cdot 1, \tilde{x} \cup \mathit{vars}(m)]\!] \\
[\![\text{if } \mathit{pr}(M_1, \ldots, M_k) & = & \{[\mathsf{state}_p(\tilde{x})] \multimap[\ \mathit{ProgFrom}_p, \mathit{ProgTo}_{p\cdot 1}, \mathrm{Pred}_{\mathit{pr}}(M_1, \ldots, M_k)\ ]\to [\mathsf{state}_{p\cdot 1}(\tilde{x})], \\
\quad \text{then } P \text{ else } Q, p, \tilde{x}]\!] & & [\mathsf{state}_p(\tilde{x})] \multimap[\ \mathit{ProgFrom}_p, \mathit{ProgTo}_{p\cdot 1}, \mathrm{Pred\_not}_{\mathit{pr}}(M_1, \ldots, M_k)\ ]\to [\mathsf{state}_{p\cdot 2}(\tilde{x})]\} \\
 & & \cup [\![P, p\cdot 2, \tilde{x}]\!] \cup [\![Q, p\cdot 2, \tilde{x}]\!] \\
[\![\text{event } F; P, p, \tilde{x}]\!] & = & \{[\mathsf{state}_p(\tilde{x})] \multimap[\ \mathit{ProgFrom}_p, \mathit{ProgTo}_{p\cdot 1}, \mathrm{Event}(), F\ ]\to [\mathsf{state}_{p\cdot 1}(\tilde{x})]\} \cup [\![P, p\cdot 1, \tilde{x}]\!] \\
[\![\text{insert } s, t; P, p, \tilde{x}]\!] & = & \{[\mathsf{state}_p(\tilde{x})] \multimap[\ \mathit{ProgFrom}_p, \mathit{ProgTo}_{p\cdot 1}, \mathrm{Insert}(s, t)\ ]\to [\mathsf{state}_{p\cdot 1}(\tilde{x})]\} \cup [\![P, p\cdot 1, \tilde{x}]\!] \\
[\![\text{delete } s; P, p, \tilde{x}]\!] & = & \{[\mathsf{state}_p(\tilde{x})] \multimap[\ \mathit{ProgFrom}_p, \mathit{ProgTo}_{p\cdot 1}, \mathrm{Delete}(s)\ ]\to [\mathsf{state}_{p\cdot 1}(\tilde{x})]\} \cup [\![P, p\cdot 1, \tilde{x}]\!] \\
[\![\text{lookup } M \text{ as } v \text{ in } P & = & \{[\mathsf{state}_p(\tilde{x})] \multimap[\ \mathit{ProgFrom}_p, \mathit{ProgTo}_{p\cdot 1}, \mathrm{IsIn}(M, v)\ ]\to [\mathsf{state}_{p\cdot 1}(\tilde{M}, v)], \\
\quad \text{else } Q, p, \tilde{x}]\!] & & [\mathsf{state}_p(\tilde{x})] \multimap[\ \mathit{ProgFrom}_p, \mathit{ProgTo}_{p\cdot 2}, \mathrm{IsNotSet}(M)\ ]\to [\mathsf{state}_{p\cdot 2}(\tilde{x})]\} \\
 & & \cup [\![P, p\cdot 1, (\tilde{x}, v)]\!] \cup [\![Q, p\cdot 2, \tilde{x}]\!] \\
[\![\mathrm{lock}^l\ s; P, p, \tilde{x}]\!] & = & \{[\mathsf{state}_p(\tilde{x}), \mathrm{Fr}(\mathrm{lock}_l)] \multimap[\ \mathit{ProgFrom}_p, \mathit{ProgTo}_{p\cdot 1}, \mathrm{Lock}(\mathit{lock}_l, s)\ ]\to [\mathsf{state}_{p\cdot 1}(\tilde{x}, \mathit{lock}_l)]\} \\
 & & \cup [\![P, p\cdot 1, \tilde{x}]\!] \\
[\![\mathrm{unlock}^l\ s; P, p, \tilde{x}]\!] & = & \{[\mathsf{state}_p(\tilde{x})] \multimap[\ \mathit{ProgFrom}_p, \mathit{ProgTo}_{p\cdot 1}, \mathrm{Unlock}(\mathit{lock}_l, s)\ ]\to [\mathsf{state}_{p\cdot 1}(\tilde{x})]\} \cup [\![P, p\cdot 1, \tilde{x}]\!]
\end{array}
$$

Figure 6: Translation of processes: definition of $[\![P, p, \tilde{x}]\!]$.

$\alpha := \alpha_{init} \wedge \alpha_{pred} \wedge \alpha_{noteq} \wedge \alpha_{in} \wedge \alpha_{notin} \wedge \alpha_{lock} \wedge \alpha_{inev} \wedge \alpha_{resil} \wedge \alpha_{prog}$ and

$$
\begin{array}{rcll}
\alpha_{init} & := & \forall i, j. & \mathrm{Init}()@i \wedge \mathrm{Init}()@j \Longrightarrow i \doteq j \\
\alpha_{pred} & := & \bigwedge_{pr \in \Sigma_{pred}} & \{\forall x_1, \ldots, x_k, i.\ \mathrm{Pred}_{pr}(x_1, \ldots, x_k)@i \Longrightarrow \phi_{pr} \mid pr \text{ is of arity k}\} \wedge \\
 & & \bigwedge_{pr \in \Sigma_{pred}} & \{\forall x_1, \ldots, x_k, i.\ \mathrm{Pred\_not}_{pr}(x_1, \ldots, x_k)@i \Longrightarrow \neg(\phi_{pr}) \mid pr \text{ is of arity k}\} \\
\alpha_{in} & := & \forall x, y, t_3. & \mathrm{IsIn}(x, y)@t_3 \Longrightarrow \exists t_2.\ \mathrm{Insert}(x, y)@t_2 \wedge t_2 \lessdot t_3 \\
 & & & \qquad \wedge \forall t_1. \quad \mathrm{Delete}(x)@t_1 \Longrightarrow (t_1 \lessdot t_2 \vee t_3 \lessdot t_1) \\
 & & & \qquad \wedge \forall t_1, y.\ \mathrm{Insert}(x, y)@t_1 \Longrightarrow (t_1 \leq t_2 \vee t_3 \lessdot t_1) \\
\alpha_{notin} & := & \forall x, y, t_3. & \mathrm{IsNotSet}(x)@t_3 \Longrightarrow (\forall t_1, y.\ \mathrm{Insert}(x, y)@t_1 \Longrightarrow t_3 \lessdot t_1) \vee \\
 & & & \qquad (\exists t_1.\ \mathrm{Delete}(x)@t_1 \wedge t_1 \lessdot t_3 \\
 & & & \qquad\qquad \wedge \forall t_2, y.\ (\mathrm{Insert}(x, y)@t_2 \wedge t_2 \lessdot t_3) \Longrightarrow t_2 \lessdot t_1) \\
\alpha_{lock} & := & \forall x, l, l', t_1, t_3. & \mathrm{Lock}(l, x)@t_1 \wedge \mathrm{Lock}(l', x)@t_3 \wedge t_1 \lessdot t_3 \\
 & & & \qquad \Longrightarrow \exists t_2.\ \mathrm{Unlock}(l, x)@t_2 \wedge t_1 \lessdot t_2 \lessdot t_3 \\
 & & & \qquad\qquad \wedge (\forall t_0.\ \mathrm{Unlock}(l, x)@t_0 \Longrightarrow t_0 \doteq t_2) \\
 & & & \qquad\qquad \wedge (\forall l', t_0.\ \mathrm{Lock}(l', x)@t_0 \Longrightarrow t_0 \leq t_1 \vee t_2 \lessdot t_0) \\
 & & & \qquad\qquad \wedge (\forall l', t_0.\ \mathrm{Unlock}(l', x)@t_0 \Longrightarrow t_0 \lessdot t_1 \vee t_2 \leq t_0) \\
\alpha_{inev} & := & \forall x, t_3. & \mathrm{InEvent}(x)@t_3 \Longrightarrow \exists t_2.\ \mathrm{K}(x)@t_2 \wedge t_2 \lessdot t_3 \\
 & & & \qquad\qquad \wedge (\forall t_0.\ \mathrm{Event}()@t_0 \Longrightarrow (t_0 \lessdot t_2 \vee t_3 \lessdot t_0)) \\
 & & & \qquad\qquad \wedge (\forall t_0, x'.\ \mathrm{K}(x')@t_0 \Longrightarrow (t_0 \leq t_2 \vee t_3 \lessdot t_0)) \\
\alpha_{resil} & := & \forall x, y, t_1. & \mathrm{Send}(x, y)@t_1 \Longrightarrow \exists t_2.\ \mathrm{Receive}(x, y)@t_2 \wedge t_1 \leq t_2 \\
\alpha_{prog} & := & \bigwedge_{a \in \mathsf{From}(P) \wedge B \in \pi(a)} & \left\{\forall l, t_1.\ \mathit{ProgFrom}_a(l)@t_1 \Longrightarrow \exists t_2.\ \bigvee_{b \in B} (\mathit{ProgTo}_b(l)@t_2)\right\}
\end{array}
$$

Figure 7: Definition of α.

a fresh name, which the Fr-fact in the premise guarantees to be new. This step is annotated with a (reserved) action *ProtoNonce*. This annotation is merely used in the proof of correctness to distinguish adversary and protocol nonces which is useful as it allows us to identify the restricted names of the process. Note that the fact $\mathsf{state}_{p,1}$ in the conclusion carries n_a, so that the following protocol steps are bound to the fresh name used to instantiate n_a. The first rules of the translation of out and in model the communication between the protocol and the adversary, and vice versa. In the case of out, the adversary must know the channel M, modelled by the fact $\mathsf{In}(M)$ in the rule's premisse, and learns the output message, modelled by the fact $\mathsf{Out}(N)$ in the conclusion. In the case of in, the knowledge of the message N is additionally required and the variables of the input message are added to the parameters of the state fact to reflect that these variables are bound. The second and third rules of the translations of out and in model an internal communication, which is synchronous. For this reason, when the second rule of the translation of out is fired, the state-fact is substituted by an intermediate, *semi-state* fact, $\mathsf{state}^{\mathsf{semi}}$, reflecting that the sending process can only execute the next step if the message was successfully received. The fact $\mathsf{Msg}(M, N)$ models that a message is present on the synchronous channel. Only with the acknowledgement fact $\mathsf{Ack}(M, N)$, resulting from the second rule of the translation of in, is it possible to advance the execution of the sending process, using the third rule in the translation of out, which transforms the semi-state *and* the acknowledgement of receipt into $\mathsf{state}_{p,1}(\ldots)$. Only now the next step in the execution of the sending process can be executed. The remaining rules essentially update the position in the state facts and add labels. Some of these labels are used to restrict the set of executions. For instance the label $\mathrm{Pred}\_pr(M_1, \ldots, M_k)$ will be used to indicate that we only consider executions in which ϕ_{pr} holds for $M_1, \ldots, M_k$. As we will see in the next section these restrictions will be encoded in the trace formula.

6.4 Discussion

While an ad-hoc encoding of these properties in Tamarin's calculus is possible, it is prone to missing attacks (indeed, we present a general method of encoding these properties). In particular for *local progress*, an ad-hoc encoding runs the risk of implying the desirable property right away. Not only is it difficult to do, see Section 6 for details, if is not even clear how to specify that, e.g., local progress is encoded correctly. As the underlying calculus (multiset rewriting, msr) has no notion of processes or protocol parties, it is hard to say what constitutes a local computation. We solve this by formalizing these properties in a higher-level calculus, which then soundly translates to the lower level calculus of Tamarin. Another benefit of such a translation is that knowledge of the protocol structure can be used to enhance the translated rules with additional information, helping Tamarin's verification procedure.

6.5 Heuristics

In order to improve the degree of automation, make use of the a priori knowledge that the msr system is an output of our translation, and employ a custom heuristic for the Tamarin prover. These heuristics can be switched on using the command line switch `--heuristic=p` and is available in the tamarin-prover development version. It alters the ranking of goals which is used to determine the next step in an automatic proof. The heuristics have no bearing on the correctness of Tamarin, but often improve automation of the verification procedure, as our case studies show (see Table 1).

The main goal is to avoid a loop in the resolution procedure, so our approach is conservative in that we only prioritize goals that do not cause other prioritized goals to appear, unless the protocol has been annotated to do that. Most heuristics have been discussed in previous work [19], in summary:

- Facts of form state_p are prioritized (less than case distinctions, but more than the following facts or actions).
- *Unlock*-actions are prioritized.
- Messages and *Insert*-actions can be prioritized or de-prioritized based on user-supplied annotations, but in the present case studies, we did not make use of this feature.

"L_" instead of "F_" achieves deprioritisation.

We have added the following heuristics specifically for this extension of SAPiC. Disjunction are still prioritized (this is the default in Tamarin's 'smart' heuristic), but disjunctions resulting from α_{prog} are explicitly excluded and prioritized less. The prioritisation of disjunctions serves the goal of applying the conclusion of helping lemmas that are already proven as soon as possible, as they are typically disjunctions. Solving *ProgTo* goals, however, is typically not as urgent; if a helping lemma can already add information to or even refute a branch, this more valuable. Receive goals are not prioritized at all, because prioritizing both *ProgTo* and Receive goals could easily resolve in a loop, e.g., if *ProgTo* triggers a Send, which triggers a Receive, which in turn activates progress in another participant. Finally, $\mathsf{MID}_{\mathsf{snd}}$ and $\mathsf{MID}_{\mathsf{rcv}}$ facts, which can always be resolved immediately by MID, are prioritized even above state-facts.

7 Proof of correctness

The correctness of our translation is stated by the following theorem.

Theorem 1. *Given a well-formed ground process P and a well-formed trace formula φ we have that*

$$traces^{ppi}(P) \vDash^{\star} \varphi \text{ iff } traces^{msr}([\![P]\!]) \vDash^{\star} [\![\varphi]\!]_{\star}$$

where $\star$ is either $\forall$ or $\exists$.

We here give an overview of the main propositions and lemmas needed to prove Theorem 1. Detailed proofs are given in the Appendix. To show the result we need two additional definitions. We first define an operation that allows to restrict a set of traces to those that satisfy the trace formula α as defined in Definition 16.

Definition 17. *Let α be a trace formula and Tr a set of traces. We define*

$$filter_{\alpha}(Tr) := \{tr \in Tr \mid \forall \theta.(tr, \theta) \vDash \alpha\}$$

The proof (detailed in Appendix D) follows directly from the definitions. Next we define the *hiding* operation which removes all reserved facts from a trace.

Definition 18 (*hide*)**.** *Given a trace tr and a set of facts $\mathcal{F}$ we inductively define $hide_{\mathcal{F}}([]) = []$ and*

$$hide_{\mathcal{F}}(F \cdot tr) := \begin{cases} hide_{\mathcal{F}}(tr) & \text{if } F \subseteq \mathcal{F} \\ (F \setminus \mathcal{F}) \cdot hide_{\mathcal{F}}(tr) & \text{otherwise} \end{cases}$$

Given a set of traces Tr we define $hide_{\mathcal{F}}(Tr) = \{hide_{\mathcal{F}}(t) \mid t \in Tr\}$.

As expected, well-formed formulas that do not contain reserved facts evaluate the same whether reserved facts are hidden or not, which is shown in Proposition 3 Appendix D.

We can now adopt the main lemma of the previous translation [19], which is relating the set of traces of a process P and the set of traces of its translation into multiset rewrite rules.

Definition 19 (Traces of P (no progress))**.** *Given a ground process P we define the* set of traces of *P as*

$$traces^{pi}(P) = \Big\{(F_1, \ldots, F_n) \mid (\emptyset, \emptyset, \{P\}, \emptyset, \emptyset, \emptyset) \xRightarrow{F_1}_{*} (\mathcal{E}_1, \mathcal{S}_1, \mathcal{P}_1, \sigma_1, \mathcal{L}_1, \mathcal{U}_1) \xRightarrow{F_2}_{*} \\ \ldots \xRightarrow{F_n}_{*} (\mathcal{E}_n, \mathcal{S}_n, \mathcal{P}_n, \sigma_n, \mathcal{L}_n, \mathcal{U}_n) \wedge \mathcal{U}_n = \emptyset\}.$$

Note that this adoption does not yet take into account the progress axiom. We chose to treat this argument separately to improve readability.

Lemma 1 (Adaptation of the previous proof)**.** *For all P well-formed with respect to this paper's Definition,*

$$traces^{pi}(P) = hide_{\mathcal{F}_{res}}(filter_{\alpha \setminus \alpha_{prog}}(traces^{msr}([\![P]\!]))).$$

Proof. The proof is largely similar to the one presented in earlier work [19]. The adapted cases are detailedin Appendix C for completeness, here, we just briefly note which cases are affected:

- The set of reserved keywords previously used is a subset of $\mathcal{F}_{res}$. As P is well-formed, no reserved word in $\mathcal{F}_{res}$ appears in P. By definition of *hide*, whenever an event was filtered in the previous proof, it still is.
- Sending and receiving messages on the resilient channels requires correct bookkeeping, i. e., at any point of the execution, the multiset of undelivered messages equals the multiset of pairs of messages and message ids (*mid*, see rule MID) for which a Send-action appears in the trace, but not a Receive-action.
- Compared to the previous translation, we handle replication differently: for $P|_p$ a replication, in the previous translation $\mathsf{state}_p(\ldots)$ is a permanent fact, so the rule can be instantiated arbitrarily often. Now, the linear fact $\mathsf{state}_p(\ldots)$ can be exchanged for a permanent fact $\mathsf{state}_p^{\mathsf{semi}}(\ldots)$. This is necessary to be able to annotate this rule with a *ProgTo*-action, i.e., have a point where the previous step was completed, but the replication is entered properly.
- We handle (possibly nested) non-deterministic choice.
- As there are only the public and the resilient channel, and thus no secret channel, internal message transmission is omitted. In particular, the notion of normal msr-execution (Definition 25 in Appendix C) is simplified compared to the previous version of the proof.

□

In order to show the same property for $traces^{ppi}$ and α including α_{prog}, we have to show that, at any point, if and only if no process can be further reduced, if α, including α_{prog}, is preserved.

Lemma 2 (Correctness of α_{prog})**.** *The following two statements are equivalent for all $E_1, \ldots, E_m$:*

$$\exists s_1, \ldots, s_m.(s_0 \xRightarrow{E_1} \cdots \xRightarrow{E_m} (\mathcal{E}_m, \mathcal{S}_m, \mathcal{P}_m, \sigma_m, \mathcal{L}_m, \mathcal{U}_m)) \wedge \forall Q \in \mathcal{P}_m.\mathsf{blocking}(Q)$$

iff.

$$\begin{aligned}\exists S_1, \ldots, S_n, F_1, \ldots, F_n.\emptyset \xrightarrow{F_1}_* \cdots \xrightarrow{F_n}_* S_n \in exec^{msr}([\![P]\!]) \\ \wedge (E_1, \ldots, E_m) = hide_{\mathcal{F}_{res}}((F_1, \ldots, F_n)) \\ \wedge ((F_1, \ldots, F_n)) \vDash \alpha.\end{aligned}$$

Proof-sketch. The proof in Appendix B.2 uses the strong invariants on $\alpha \setminus \alpha_{prog}$ in Lemma 14 (for the direction from SAPiC to msr) and Lemma 16 (for the inverse direction) to establish a corresponding msr trace, or SAPiC execution, depending on the direction to be proven. Both Lemmas can be found in Appendix C) and constitute the proof to Lemma 1 above. With the corresponding msr trace or SAPiC execution established, the argument only relies on the correctness of the progress function, which in turn relies on Lemmas 5 to 11 provided in Appendix B.1. In the first direction, if there is a process that is non-blocking, there is a *ProgFrom*-action not yet 'resolved' by a *ProgTo*-action, otherwise α_{prog} would hold. Conversely, where α_{prog} does not hold, one can point to a *ProgFrom* that is not 'resolved', which identifies a position that must be a prefix of some position in the process that has not been further resolved. In both directions, we argue about the dependency graph inducing the trace rather than the trace itself, which allows us to relate dependencies between msr rules to prefix-relations on positions. □

Combining Lemma 1 and 2, we can show our main lemma.

Lemma 3 (trace-equivalence)**.** *For all well-formed P, then*

$$traces^{ppi}(P) = hide_{\mathcal{F}_{res}}(filter_{\alpha}(traces^{msr}([\![P]\!]))).$$

Table 1: Case studies and results: ✓ denotes successful verification, while ✗ denotes we discovered an attack. ∞ means that the verification procedure diverges.

	ASW		ASW (mod.)		GJM		GJM (mod.)		toy example	
property	type	time	type	time	type	time	type	time	type	time
timeliness (A)	✓	1:40min	✓	1:38min	✓	0:46min	✓	6:08min	✗	37s
timeliness (B)	∞	—	✓	37:34min	✓	12:49min	✓	34:49h		
fairness (A)	✗[1]	8:34min	✗[1]	31:06min	✗[2]	2:22min	✓[2]	14:11min	✓[2]	5:46h
			✓[2]	0:40h						
fairness (B)			✓	14:05h[2]			✓[2]	43:52h [3]		

[1] weak notion of contract [2] strong notion of contract [3] add. helping lemma (verified in 2:38min)

Proof-sketch. The idea is to define $traces^{ppi}(P)$ in terms of the set difference between $traces^{pi}(P)$ and non-final traces. As the negation of Lemma 2 show equivalence between non-final SAPiC executions and msr executions that are filtered, the rest of the proof is a set-theoretical transformation. □

Our main theorem can now be proven by applying Lemma 3, and Proposition 3 and Proposition 2 (cf. Appendix D).

Proof of Theorem 1.

$$
\begin{aligned}
& traces^{ppi}(P) \vDash^{\star} \varphi \\
& \quad \Leftrightarrow hide_{\mathcal{F}_{res}}(filter_{\alpha}(traces^{msr}([\![P]\!]))) \vDash^{\star} \varphi && \text{(by Lemma 3)} \\
& \quad \Leftrightarrow filter_{\alpha}(traces^{msr}([\![P]\!])) \vDash^{\star} \varphi && \text{(by Proposition 3)} \\
& \quad \Leftrightarrow traces^{msr}([\![P]\!]) \vDash^{\star} [\![\varphi]\!]_{\star} && \text{(by Proposition 2)}
\end{aligned}
$$

□

The axiom α_{inev} within α has turned out to slow down verification time, which is why we have chosen to remove it. We show in Theorem 2 Appendix E, that this is sound for all security properties we are interested in.

8 Case studies

In this section we present the analyses of several case studies using our extension of SAPiC/tamarin toolchain. The obtained results, obtained using 16 2.5GhZ Intel Xeon E7-8867 cores and 1.5TB available RAM are summarised in Tables 1 and 2. The implementation and SAPiC models are part of the tamarin-prover repository [2].

8.1 A first toy protocol

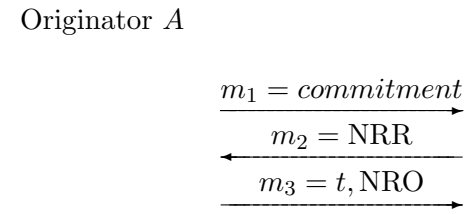
Our first case study is a fair non-repudiation protocol introduced in [20] to motivate the need for timeliness in addition to fairness. We will also use this protocol to discuss our modelling of fairness, timeliness and corruption.

Protocol description

The protocol consists of two sub-protocols: a main protocol and a recovery protocol. In the main protocol, 3 messages are exchanged.

In order to concentrate on the message flow, we do not give the details of these messages, which can be found in [20]. The first message m_1 is a commitment of A to send some text t. B replies

[2] `https://github.com/tamarin-prover/tamarin-prover`



Toy protocol: honest protocol run

$$P_A(a,b,ttp,t) = \text{new } k;\ \text{event } Start_A(a,k);\ \text{out}(\text{`c'}, m_1); \left(\begin{array}{l}\text{in}(\text{`c'}, m_2);\ \text{out}(\text{`c'}, m_3);\\ \text{event } Contract_A(a,b,t,k)\end{array}\right) + \left(\begin{array}{l}\text{in}(\text{`r'}, r_2);\\ \text{event } Contract_A(a,b,t,k)\end{array}\right)$$

$$P_B(a,b,ttp) = \text{in}(\text{`c'}, m_1);\ \text{new } sess;\ \text{event } Start_b(b,sess);\ \text{out}(\text{`c'}, m_2); \left(\begin{array}{l}\text{in}(\text{`c'}, m_3);\\ \text{event } Contract_b(a,b,t,sess)\end{array}\right) + \left(\begin{array}{l}\text{out}(\text{`r'}, r_1); \text{in}(\text{`r'}, r_3);\\ \text{event } Contract_b(a,b,t,sess);\end{array}\right)$$

$$P_T = \text{in}(\text{`r'}, r_1); \text{out}(\text{`r'}, r_2); \text{out}(\text{`r'}, r_3);$$

Figure 9: Description of the toy fair non-repudiation protocol

by sending m_2 which represents a non-repudiation of receipt (NRR) evidence. Finally, message m_3 contains t and a non-repudiation of origin (NRO) proof. An obvious fairness problem arises when B does not receive m_3. In this case, he may contact the TTP with a resolve request (which includes m_1). m_1 contains enough information for the TTP to recover t and produce a non-repudiation of origin evidence on behalf of A. The TTP sends the evidence and t to B. The TTP also sends a non-repudiation of receipt evidence to A on behalf of B: this is important as a dishonest B could otherwise request a resolve after having received m_1 without sending m_2.

The processes used to model the roles of A, B and the TTP are given in Figure 9. The definitions of the messages m_i and r_i are available in our example files. The processes P_A and P_B use the NDC operator to model the possible branching in case of a recovery. Note that, unlike the more complex ASW and GJM examples, here the model of the TTP neither requires NDC nor persistent state (to store the status of the protocol). The processes are annotated with events that allow us to define security properties. Even though the toy protocol was designed to exchange non-repudiation evidences we will refer to the items to be exchanged as contracts (these evidences may be seen as a kind of contract).

Modelling Fairness

In this section, we will discuss our formulation of the fairness property. Intuitively, fairness may be expressed as follows:

"Either both parties can receive a contract or none of them can."

Suppose that C_A and C_B are logical formulas that represent the statement *"if A (respectively B) proceeds, she will receive the expected contract"*. This can indeed be expressed using the *Contract* events that annotate the processes (see Figure 9). Then the above intuitive formulation can be expressed as

$$\begin{aligned}&(C_A \wedge C_B) \vee \neg(C_A \vee C_B)\\ \Leftrightarrow &(C_A \rightarrow C_B) \wedge (C_B \rightarrow C_A).\end{aligned}$$

The second equivalent formulation expresses both fairness for A (first disjunct) and fairness for B (second disjunct). In our complete model, we consider the cases where A or B may be dishonest (modelled through corruption described below). Suppose that D_B expresses that B has been corrupted. In

that case we do not require fairness to hold for B, but only for A. As in our calculus, protocol events can only be emitted by honest runs of the protocol, the attacker may not be able to raise the event $Contract_B$ for a corrupted B. Therefore, we model a fourth entity, a judge, which emits an event if enough evidence has been brought forward to prove that a contract was made:

$$\begin{aligned} P_J = \quad & (\text{in}(\text{`c'}, m_1);\ \text{event}\ Contract_{judge}(A, B, T)) \\ & |\cdots|(\text{in}(\text{`c'}, m_n);\ \text{event}\ Contract_{judge}(A, B, T)) \end{aligned}$$

where $m_1, \ldots, m_n$ are the messages that suffice as evidence of a contract. We assume the public variable T is part of $m_1, \ldots, m_n$ and describes the contract text. Suppose that J expresses that *"it is possible to raise event* $Contract_{judge}$*"*. Then, in the case where B is corrupted, C_B should be replaced by J, and fairness for A expressed as $J \rightarrow C_A$. Note that the judge is different from the TTP in particular, the judge never emits messages, but just an event if sufficient evidence for a contract was brought forward.

Following these ideas, and recalling that fairness is only required to hold for uncorrupted parties, we can express fairness for A in the first-order logic introduced in Section 5.

$$\begin{aligned} & \forall i : temp, a, b, t : pub.Contract_{judge}(a, b, t)@i \\ & \quad \Rightarrow \quad (\exists j : temp, k : msg.Contract_A(a, b, t, k)@j) \\ & \qquad \vee\, (\exists k : temp.Corrupt(a)@k). \end{aligned}$$

where *Corrupt* is the event raised when a party has been corrupted. Fairness for B is obtained by switching A for B. Overall, fairness is the conjunction of these two conditions. Using our tool we show that fairness indeed holds for A.

Modelling timeliness

Timeliness guarantees that no honest participant is 'left hanging', i.e., stuck in a situation where it cannot continue without the help of another participant, while fairness guarantees that no honest party ends up without a contract if the other has one. Consider again the toy example. Even though fairness holds for A, once A has sent message m_1 he needs to wait for either m_2 or r_2. He does however not know whether one of these messages will ever arrive or if B simply stopped the protocol – A is left 'hanging'. This demonstrates that fairness does not imply timeliness, while the other direction is clear: even if a participant can always terminate, he might not always obtain the contract.

Timeliness expresses that a participant can always unilaterally (i.e. without the help of the other participant, but possibly relying on the TTP) finish the protocol. Timeliness is modelled by annotating the processes with *Start* events, expressing that a session has started (cf Figure 9) in addition to the *Contract* events (and *Abort* events in the more complex ASW and GJM protocols). The arguments of these events should identify the session. Then timeliness for A is expressed as

$$\begin{aligned} \forall i : temp, a, b, t, k.Start_A(a, k)@i \Rightarrow & \\ (\exists j : temp.Contract_A(a, b, t, k)@j) & \\ \vee(\exists j : temp.Corrupt(a)@j) & \end{aligned}$$

Again, no guarantee is required when A is corrupt and timeliness for B is formulated similarly. Using our tool we confirm that timeliness does not hold for A.

Modelling resilient channels and corruption

In our case studies, we found that it is very often not clear what exactly is required from resilient channels to achieve timeliness or fairness. Suppose Alice takes the role of the initiator in two sessions, and the role of the responder in another. Do all her sessions share the same channel, do the initiator sessions share the same channel, or does every session have a separate channel?

In the ASW protocol, we found that a reply from the TTP does not identify which responder session should receive it and we chose to model:

- For the toy and ASW protocol, one resilient channel to the TTP per participant and protocol role (either originator or responder), along with the corresponding return channel.
- For the GJM protocol, one channel per session, as this protocol does not carry any session information in its messages.

Our calculus provides only a single resilient channel, but the above assumptions can be trivially modelled via pattern matching. While the assumption of a channel per participant is standard (e.g., it is necessary for fairness of the Zhou-Gollman protocol [30]), the separation by protocol role is unusual. It is justified in the case of the ASW protocol, as a participant A that has two sessions with itself and aborts the protocol in both, might receive one of the two abort messages from session one in the other session. While this does not necessarily imply an attack, it makes it much more difficult to prove timeliness for our tool, while it only amplifies the assumption.

The corruption process raises an event to mark a party corrupted, and reveals its secret key to the adversary. Additionally, for each corrupted party we add a process that inputs any messages sent over resilient channels to these parties. This is important as any trace with undelivered messages is ignored and attacks might be missed.

$$\begin{array}{l} !\ \mathrm{in}(\text{`c'}, \langle \text{'cor'}, x\rangle);\ \text{event}\ \mathit{Corrupt}(x);\ \mathrm{out}(\text{`c'}, \mathit{sk}(x));\\ \quad (!\ \mathrm{in}(\text{`r'}, \langle \text{'resp'}, x, m\rangle\ \mid\ !\ \mathrm{in}(\text{`r'}, \langle \text{'orig'}, x, m\rangle)) \end{array}$$

8.2 ASW protocol

The optimistic contract-signing protocol by Asokan, Shoup, and Waidner [4] proceeds as follows. For a contract text T, the originator A sends a signature for T and a commitment to a freshly drawn nonce n_a in the form of a hash. The responder B confirms by signing this message and a commitment on another freshly drawn nonce, n_b. Both parties then exchange their nonces. (Note that we have left out the identifiers of originator, responder and TTP in the first message.) In case that A or B

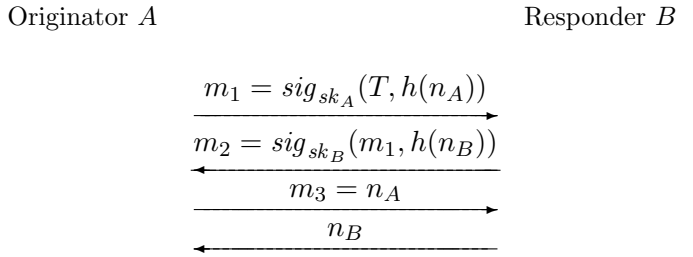
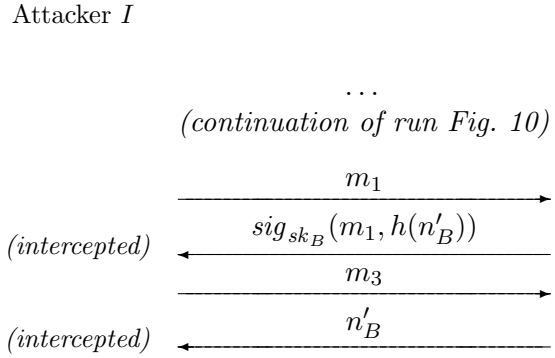


Figure 10: ASW protocol: honest protocol run

are not receiving a response in appropriate time, A may decide to abort the protocol (if the second message does not arrive), to resolve the contract with the TTP (if the fourth message does not arrive), or B may decide to resolve the contract (if the third message does not arrive). For brevity, we will only outline the parts of the corresponding abort/resolve-protocols when they are relevant to attacks below.

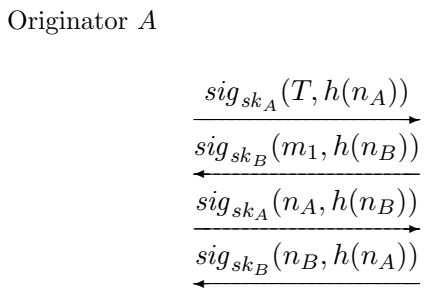
It is important to note that here *the complete transcript from the first to the fourth message constitutes the contract text*, including the nonces. As indicated by the original authors [4, Definition 3.1], each transcript of the honest protocol run identifies a different contract. In case the TTP is called, a second form of a valid contract is recognized, which consists of the TTP's signature on the first and the second message of the opportunistic protocol run, $sig_{sk_{TTP}}(m_1, m_2)$ for m_1 and m_2 of the form in Figure 10.

With this notion of contract, the following replay attack permits an attacker to create an arbitrary amount of different copies with the same contract text T for A and B, without A having any knowledge of this, nor A having any evidence that this attack took place, as the TTP is never contacted. Suppose the attacker observes the honest run above, he can commit to another contract with the responder B in the name of A, just by replaying the first and third message:



ASW protocol: Shmatikov/Mitchell attack.

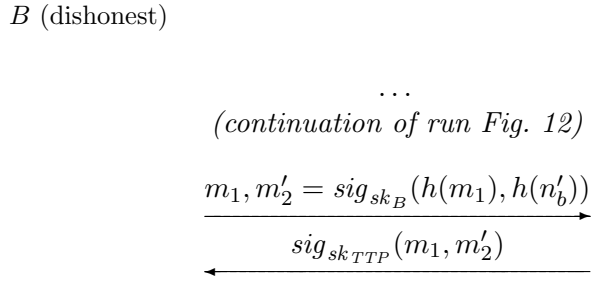
This attack was discovered in a finite model by Shmatikov and Mitchell. The weakness here is that the third message is not related to the second message in any way, so Shmatikov and Mitchell proposed the following fix for the protocol.



ASW protocol: repaired version by Shmatikov and Mitchell.

Nevertheless we were able to show timeliness for A. We were unable to show timeliness for B, as the response of the TTP to a resolve request from B does not contain enough information to identify B's session. Rather than strengthening the assumption on the channel to avoid this possible confusion, we chose to modify the protocol by adding $h(n_B)$ to the response. Note that *a*) we did not find an actual attack, as a message that arrives in the wrong session would prevent this session to pick up any other messages, hence we conjecture that, overall, each message will be successfully picked up. Yet, the additional proof effort did not seem to justify the slight gain in generality. Furthermore, *b*) for the GJM protocol, we did avoid this problem by imposing a larger number of secret channels, as there was no hope for a similar fix in that protocol. For the protocol containing our and Shmatikov/Mitchell's modification, we managed to show timeliness for both parties.

Using our tool, we also found that surprisingly simple attack on the fairness of this repaired protocol. Suppose a dishonest B has signed a contract with A and wants to have a second copy of it. B can obtain a second copy without A's consent by calling the TTP to resolve with a 'refreshed' m_2, where n_B is substituted by a freshly drawn nonce n'_B.



ASW protocol: new attack.

If we alter the judge process, so that it identifies a contract with the text committed to, and the

Table 2: OPC UA Secure Conversation results

Property	Time	Proof steps
all messages are received	1s	15
all messages were sent	7s	138
message order is respected	26s	204

two signers, but not the nonce n_a, then we are able to show fairness for both parties. We call this property *fairness for the weaker notion of a contract*. Note that this rules out certain kinds of contract, e.g., if A and B exchange IOUs, one would expect each new IOU, even if it contains the same text, to correspond to a different contract, e.g., that three contracts saying 'A owes B \$50' would amount to a debt of \$150.

8.3 GJM protocol

The fairness of the optimistic contract-signing protocol by Garay, Jakobsson, and MacKenzie (GJM) [16] was already analysed in previous work, but only in a bounded model. Under the assumption that each party has a reliable channel to and from the TTP for each session, we can automatically show timeliness for A and B. The verification proceeds automatically and without any additional lemma.

However, we immediately found an attack on fairness for A, even for the weak notion of contract. The optimistic protocol run, as well as recovery conducted by the trusted third party, will return a contract of the same form, namely

$$(sig(\langle `1', t\rangle, sk_A), sig(\langle `2', t\rangle, sk_B)),$$

where t is the contract set, and '1' and '2' just serve to distinguish these messages in a protocol run. Note that neither signature contains the identity of the respective contract partner. Hence it is easy, e.g., for a party X with a bad reputation, to obtain a contract A would only want to sign with B, just by replacing the second signature:

$$(sig(\langle `1', t\rangle, sk_A), sig(\langle `2', t\rangle, sk_X)).$$

This attack only applies if the contract text does not explicitly mention the signing parties but rather depends on the signers, e.g., "the signers agree to ...". If we require t to be of the form $\langle A, B, t'\rangle$, i.e., to contain the signing parties, we can automatically show fairness (for the weak notion of contract) for A and B. As the resulting contract does not contain fresh information shared between the parties, we have no hope of showing the strong notion of fairness for the protocol as it is. The protocol we show secure actually enjoys a small improvement: Garay et.al. assume the reliable channel to the TTP to additionally be secret. We lift this assumption, as only the responder's resolve message needs to be kept secret. Thus, we use asymmetric encryption in the transmission of this message, while the originator's resolve message and the abort message can remain unencrypted.

8.4 OPC UA Secure Conversation Protocol

To show that our approach can also be used beyond contract-signing, we analyzed the Secure Conversation Protocol, which is part of the United Architecture (UA) standard [25] developed by the OPC Foundation. The protocol implements a security layer designed for the use in industrial control systems, and aims at securing the data flow between two devices that share symmetric keys. It uses symmetric encryption and message authentication codes (MACs), and relies on sequence numbers to ensure the correct order of messages.

In the context of industrial control systems, the integrity of the data exchanged between two devices is extremely important. Modifying, injecting, or just reordering command messages, e.g., in critical infrastructure such as the power grid, can have catastrophic effects by putting the system in a state beyond its safe operation limits.

Similar to fair exchange protocols, the protocol relies on a resilient channel to ensure message delivery, yet the protocol still needs to make sure that messages cannot be injected, duplicated or reordered. Using our approach we were able to show that in the OPC UA Secure Communication protocol, all messages are received only once and in the correct order. More precisely, we prove the following properties.

- All sent messages are received:

$$\forall i : temp, A, B, t : msg.\ Send(A, B, t)@i \Rightarrow \\ (\exists j \gg i : temp.\ Recv(A, B, t)@j)$$

- All received messages were sent before and are only received once:

$$\forall i : temp, A, B, t : msg.\ Recv(A, B, t)@i \Rightarrow \\ (\exists j \ll i : temp.\ Send(A, B, t)@j \wedge \\ \neg(\exists k \not\equiv i : temp, A_2, B_2 : msg.\ Recv(A_2, B_2, t)@k))$$

- Any two messages that are received in a certain order were sent in that order:

$$\forall i, j : temp, A, B, m, n : msg. \\ Recv(A, B, m)@i \wedge Recv(A, B, n)@j \wedge i \ll j \\ \Rightarrow (\exists k, l : temp. \\ Send(A, B, m)@k \wedge Send(A, B, n)@l \wedge k \ll l)$$

9 Evaluation

Contrary to previous analyses of state-based protocols, the security properties and attacks in our case studies were derived without the need of any additional intermediate lemmas. This is promising, as it actually achieves the goal of automated verification. It also indicates that our heuristics work well.

The verification time itself was remarkably long, almost 15 hours in the case of fairness for the responder in the ASW protocol. However, this is not surprising given the complexity of these protocols.

The first protocol we modelled was the ASW protocol. As we gained experience with the correct formulation of fairness, corruption, etc. during this process, it is difficult to say how long modelling and verification took. For the GJM protocol, the verification took approximately two weeks, where the actual interaction with the prover was about a third of that time. If this paper succeeds in conveying our experiences, this should give a rough estimate of how long the verification of a fair exchange protocol would take for a dedicated analyst.

We found that the modelling language is well-suited and intuitive, and that the Tamarin prover captures causal relations within the protocol nicely. Initially, we modelled the TTP's database using global state. We then observed that the TTP, once consulted, does not need to distinguish between a contract that was resolved and one that was aborted. Hence, we could model the state of a protocol via locks that are never removed. As SAPiC's translation has specific optimisations for locks, this noticeably simplified the analysis. While timeliness lemmas could usually be proven out of the box, this restating of the TTP greatly helped in showing fairness.

We experienced the following difficulties: first, modelling fairness and corruption proved difficult. Notions introduced in previous work did not readily apply, or missed attacks; hence, we had to derive a formulation of the fairness property as outlined in Section 8.1. Similarly, a sloppy modelling of corruption can easily miss attack 8.1. The underlying issue here is that the model needs to make sure that each message sent on a reliable channel can be picked up by someone; otherwise, attacks might be missed, since any trace containing a message that does not arrive is (rightfully) discarded. While future analyses using our approach can benefit from our formulation of fairness and our modelling of corruption, this issue also concerns the protocol modelling. We advice to make sure that all valid protocol run are covered by sanity lemmas showing that messages output on the reliable channel can be received.

10 Conclusion

In this paper, we have presented a novel methodology for reasoning about liveness properties of cryptographic protocols in a machine-assisted manner without imposing artificial constraints on the size of protocol descriptions and executions as commonly done in prior work. Our findings from applying this methodology to the widely investigated class of fair exchange protocols notably demonstrate that such finiteness constraints do not constitute a purely academic limitation, but that they are responsible for not detecting actual weaknesses in such protocols.

Moreover, our approach of augmenting a higher-level calculus with key concepts for stating and reasoning about liveness properties and of subsequently designing a provably sound and complete translation into the widely used model of multiset rewriting allowed us to build upon recent advances in the automated verification of cryptographic protocols. In particular, we strongly benefit from the large degree of automation in the state-of-the-art verification tool Tamarin, and enable reasoning about liveness properties in Tamarin in a comprehensive manner through our translation.

11 Acknowledgements

This work was supported by the German Federal Ministry for Education and Research (BMBF) through funding for the Center for IT-Security, Privacy and Accountability (CISPA), the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 645865-SPOOC), as well as the Centre National de la Recherche Scientifique (CNRS) through funding for the PEPS projects ASSI and VESPA.

A Progress function

In order to define the progress function, we introduce some notation for the manipulation of sets of positions first: Given a position p_0 and a set of positions P we denote by $p_0 \cdot P$ the set of positions $\{ p_0 \cdot p \mid p \in P \}$.We first define the next possible position for a given process, i.e., the child process for all non-zero processes, except for NDC processes, where the next position jumps over blocking subprocesses.

Definition 20. *Given a ground process P we define the function next* $\mathsf{next}(P)$ *as*

$$\mathsf{next}(P) \hat{=} \begin{cases} \emptyset & \text{if } P = 0 \\ 1.\mathsf{next}(P_1) \cup 2.\mathsf{next}(P_2) & \text{if } P = P_1 + P_2 \text{ and } \mathsf{blocking}(P_1) \wedge \mathsf{blocking}(P_2) \\ 1.\mathsf{next}(P_1) \cup \{2\} & \text{if } P = P_1 + P_2 \text{ and } \mathsf{blocking}(P_1) \wedge \neg\mathsf{blocking}(P_2) \\ \{1\} \cup 2.\mathsf{next}(P_2) & \text{if } P = P_1 + P_2 \text{ and } \neg\mathsf{blocking}(P_1) \wedge \mathsf{blocking}(P_2) \\ \mathsf{children}(P) & \text{otherwise} \end{cases}$$

Next, we define the starting points of local progress. Intuitively, the set $\mathsf{From}(P)$ is the set of positions of P from where a progression starts.

Definition 21. *Given a ground process P we define the set* $\mathsf{From}(P) \hat{=} \mathit{from}(P, \top)$ *where*

$$\mathit{from}(P, b) \hat{=} \begin{cases} \{ [] \} \cup \bigcup_{p \in \mathsf{next}(P)} p \cdot \mathit{from}(P|_p, \mathsf{blocking}(P)) & \text{if } \neg\mathsf{blocking}(P) \wedge b \\ \bigcup_{p \in \mathsf{next}(P)} p \cdot \mathit{from}(P|_p, \mathsf{blocking}(P)) & \text{otherwise} \end{cases}$$

Note that the flag b in the helper function *from* indicates that the function has been called from a blocking position. A position is in From, i.e., the first case matches, iff it has been the next position from a blocking position (or if it is the initial position $[]$), and it is not blocking itself. Observe also, that the initial position in a process $!P$ is not in $\mathsf{From}(!P)$.

Now that the positions that trigger local progress are defined, we can discuss which positions need to be reached from such a position. Of this set, we only regard those that are maximal w.r.t. the prefix relation. Obviously, these positions are blocking positions. Recall Section 6.1, in particular Example 7 to 9: the progress function returns a set of sets of positions, interpreted as a propositional formula in conjunctive normal form. We extend the previous notation: given a position p_0 and a set of sets of positions $\mathcal{P}$, $p_0 \cdot \mathcal{P}$ denotes $\{ p_0 \cdot P \mid P \in \mathcal{P} \}$.

Definition 22. *Given a process P we define the progression function $\pi \colon \mathsf{From}(P) \to 2^{2^{pos(P)}}$, as $\pi(p) = p.f(P|_p)$, where f is defined inductively on the structure of P:*

$$f(P) \hat{=} \begin{cases} \{ \{ [] \} \} & \text{if } \mathsf{blocking}(P) \\ 1.f(P_1) \cup 2.f(P_2) & \text{if } P = P_1 \parallel P_2 \\ \{ \bigcup_{p \in \mathsf{next}^0(P)} S_p \mid S_p \in p.f(P|_p) \} & \text{otherwise,} \end{cases}$$

where $\mathsf{next}^0(P)$ *is defined just like* $\mathsf{next}(P)$ *(i.e., replacing any occurence of* next *in Definition 20 by* next^0*), except that* $\mathsf{next}^0(0) = \{ [] \}$.

Using the progress function we relate positions in $\mathsf{From}(P)$ to positions they move to.

Definition 23. *Let P be a process and π its progress function. We define the binary relation $\mathcal{R}_\pi$ as*

$$(p, q) \in \mathcal{R}_\pi \text{ iff } p \in \mathsf{From}(P) \text{ and } \exists A. A \in \pi(p) \text{ and } q \in A.$$

We define the set $\mathsf{To}(P)$ *to be the range of $\mathcal{R}_\pi$.*

At any point on the right-hand side of this relation, one can point at the position in $\mathsf{From}(P)$ that lead there. In the translation, this is used to annotate positions that are progressed to with a nonce drawn at the position that that triggered the progress.

Lemma 4. *Let π be the progress function of a process P. Then, $\mathcal{R}_\pi^{-1}$ is functional.*

(Proof in Appendix B.1)

B Correctness of progress function

B.1 Lemmas about progress function

We abuse notation by writing $p\pi q$ when $(p,q) \in \mathcal{R}_\pi$ and $p\pi^{-1}q$ when $(p,q) \in \mathcal{R}_\pi^{-1}$.

Lemma 5. *Let P be a process and π its progress function. We have that*

$$\mathsf{From}(P) \subseteq \{p \mid p \in pos(P) \wedge \neg\mathsf{blocking}(P|_p), \tag{1}$$

$$\mathsf{To}(P) \subseteq \{p \mid p \in pos(P) \wedge \mathsf{blocking}(P|_p), \tag{2}$$

$$\mathsf{From}(P) \cap \mathsf{To}(P) = \emptyset \quad \textit{and} \tag{3}$$

$$p\pi q \Rightarrow p \sqsubset q. \tag{4}$$

Proof. (1) and (2) follow directly from Definition 21 and 22, respectively. (3) is a direct consequence of the first two relations. We now show Item 4. The fact that $p \sqsubseteq q$ follows from Definition 22. By item 3 we have that $p \neq q$ and hence we conclude. □

Lemma 6. *Let P be a process and π its progress function. $\mathcal{R}_\pi$ is left-total, i.e., for any $p \in \mathsf{From}(P)$ there exists $q \in pos(P)$ such that $p\pi q$.*

Proof. This follows directly from the fact that the function next^0 (Definition 22) never returns the empty set. □

Lemma 7. *Let P be a process and π its progress function. We have that*

$$\min_{\sqsubset}\{\, p \mid p \in pos(P) \wedge \neg\mathsf{blocking}(P|_p) \,\} \subseteq \mathsf{From}(P).$$

Proof. The proof proceeds by structural induction on the process P.

Base case. If $P = 0$ then $\min_{\sqsubset}\{\, p \mid p \in pos(P) \wedge \neg\mathsf{blocking}(P|_p) \,\} = \emptyset = \mathsf{From}(P)$.

Inductive case. We proceed by case distinction.

- $P = !Q$ or $P = \mathrm{in}(c,m);Q$. By induction hypotesis, we have that

  $$\min_{\sqsubset}\{\, p \mid p \in pos(Q) \wedge \neg\mathsf{blocking}(Q|_p) \,\} \subseteq \mathsf{From}(Q)$$

  and hence

  $$1.\min_{\sqsubset}\{\, p \mid p \in pos(Q) \wedge \neg\mathsf{blocking}(Q|_p) \,\} \subseteq 1.\mathsf{From}(Q).$$

  As $\mathsf{blocking}(P)$ holds we have that $[] \notin \mathsf{From}(P)$ and $\mathsf{From}(P) = 1.\mathsf{From}(Q)$. As $[] \notin \mathsf{From}(P)$ we also have that $\min_{\sqsubset}\{\, p \mid p \in pos(P) \wedge \neg\mathsf{blocking}(P|_p) \,\} = 1.\min_{\sqsubset}\{\, p \mid p \in pos(Q) \wedge \neg\mathsf{blocking}(Q|_p) \,\}$ which allows us to conclude.

- $P = P_1 + P_2$. If $\neg\mathsf{blocking}(P)$ then $[] \in \mathsf{From}(P)$ and $\min_{\sqsubset}\{\, p \mid p \in pos(P) \wedge \neg\mathsf{blocking}(P|_p) \,\} = \{\, [] \,\}$ which allows us to conclude.

  Otherwise, we have that $\mathsf{blocking}(P_1)$ and $\mathsf{blocking}(P_2)$. Then for $i \in \{1,2\}$

  $$\mathsf{From}(P_i) = \bigcup_{p \in \mathsf{next}(P_i)} p.\mathit{from}(P_i|_p, \top)$$

and

$$\begin{aligned}
\mathsf{From}(P) &= \bigcup_{p \in \mathsf{next}(P)} p.\mathit{from}(P|_p, \top) \\
&= \bigcup_{p \in (1.\mathsf{next}(P_1)) \cup (2.\mathsf{next}(P_2))} p.\mathit{from}(P|_p, \top) \\
&= \bigcup_{p \in 1.\mathsf{next}(P_1)} p.\mathit{from}(P|_p, \top) \bigcup_{p \in 2.\mathsf{next}(P_2)} p.\mathit{from}(P|_p, \top) \\
&= \bigcup_{p \in \mathsf{next}(P_1)} p.\mathit{from}(P|_{1.p}, \top) \bigcup_{p \in \mathsf{next}(P_2)} p.\mathit{from}(P|_{2.p}, \top) \\
&= 1.\mathsf{From}(P_1) \cup 2.\mathsf{From}(P_2). \qquad (5)
\end{aligned}$$

By induction hypotesis, we have for $i \in \{1, 2\}$ that

$$\min_{\sqsubset}\{\, p \mid p \in \mathit{pos}(P_i) \wedge \neg\mathsf{blocking}(P_i|_p) \,\} \subseteq \mathsf{From}(P_i)$$

and hence

$$i.\min_{\sqsubset}\{\, p \mid p \in \mathit{pos}(P_i) \wedge \neg\mathsf{blocking}(P_i|_p) \,\} \subseteq i.\mathsf{From}(P_i). \qquad (6)$$

As $\mathsf{blocking}(P)$ holds we have that $\min_{\sqsubset}\{\, p \mid p \in \mathit{pos}(P) \wedge \neg\mathsf{blocking}(P|_p) \,\} =$

$$\bigcup_{1 \le i \le 2} i.\min_{\sqsubset}\{\, p \mid p \in \mathit{pos}(P_i) \wedge \neg\mathsf{blocking}(P_i|_p) \,\}$$

Combining this with Equations (5) and (6) allows us to conclude.

- Otherwise, we have that $[] \in \mathsf{From}(P)$ and $\min_{\sqsubset}\{\, p \mid p \in \mathit{pos}(P) \wedge \neg\mathsf{blocking}(P|_p) \,\} = \{\, [] \,\}$ which allows us to conclude.

□

Lemma 8. *Let P be a process and π its progress function. If $p, q \in \mathit{pos}(P)$, $p \in \mathsf{From}(P)$ and $p \sqsubset q$ then there exists $r \in \mathit{pos}(P)$ such that $p \pi r$ and either $p \sqsubset q \sqsubseteq r$ or $p \sqsubset r \sqsubseteq q$.*

In the following we will sometimes say that a property holds for next^* when holds for both next and next^0.

Lemma 9. *Let P be a process. If $p_1, p_2 \in \mathsf{next}^*(P)$ and $p_1 \neq p_2$ then p_1 and p_2 are incomparable, i.e., $p_1 \not\sqsubset p_2$ and $p_2 \not\sqsubset p_1$.*

Proof. This follows directly from Definitions 20 and 22. □

Lemma 10. *Let π be the progress function of a process P. For all $p, q, r \in \mathit{pos}(P)$ such that $p \pi q$ and $p \sqsubset r \sqsubset q$, $r \notin \mathsf{From}(P) \cup \mathsf{To}(P)$.*

Proof. Let $r_1, \ldots, r_n$ be the sequence of positions such that

$$p \sqsubset r_1 \sqsubset \ldots \sqsubset r_n \sqsubset q$$

and

- $r_1 = p.s_1$ and $s_1 \in \mathsf{next}^0(P|p)$,
- $r_i = r_{i-1}.s_i$ and $s_i \in \mathsf{next}^0(P|_{r_{i-1}})$ for $1 < i \le n$,
- $q = r_n.s_n$ and $s_n \in \mathsf{next}^0(P|_{r_n}$.

By Definition 22, as $p\pi q$, such a sequence exists, and by Lemma 9, it is uniquely defined.

We will show in a second that for all r_i, $i \in \mathbb{N}_n$, $r_i \notin \mathsf{From}(P) \cup \mathsf{To}(P)$. To show that this suffices, suppose (for contradiction) that there exists r such that $p \sqsubset r \sqsubset q$ and $r \notin \{ r_1, \ldots, r_n \}$ and $r \in \mathsf{From}(P)$ or $r \in \mathsf{To}(P)$. Then by Definition 21, respectively Definition 22, there exists a sequence

$$p \sqsubset r'_1 \sqsubset \ldots \sqsubset r'_k \sqsubset r$$

such that

- $r'_1 = p.s'_1$ and $s'_1 \in \mathsf{next}^*(P|p)$,
- $r'_i = r'_{i-1}.s'_i$ and $s'_i \in \mathsf{next}^*(P|_{r'_{i-1}})$ for $1 < i \leq k$,
- $r = r'_k.s'_k$ and $s'_k \in \mathsf{next}^*(P|_{r'_k})$.

As $r \notin \{r_1, \ldots, r_n\}$ and $r \sqsubset q$ there exists a smallest index i such that $r_i \neq r'_i$, which contradicts Lemma 9.

Thus we will now show, by induction on n, that $r_i \notin \mathsf{From}(P) \cup \mathsf{To}(P)$ for all $i \in \mathbb{N}$. When $n = 0$ the result trivially holds. Suppose that it holds for $n - 1$ and suppose by contradiction that $r_n \in \mathsf{From}(P) \cup \mathsf{To}(P)$.

First suppose $r_n \in \mathsf{From}(P)$. Then, by Definition 21 we have that $\neg\mathsf{blocking}(P|_{r_i})$, as well as $\mathsf{blocking}(P|_{r_{i-1}})$ where we define $r_0 = p$. If $i = 1$ then we have $\mathsf{blocking}(P|_p)$ and as $p \in \mathsf{From}(P)$, by (1) in Lemma 5, $\neg\mathsf{blocking}(P|_p)$ yielding a contradiction. If $i > 1$ then by Definition 22, as $\mathsf{blocking}(P|_{r_{i-1}})$, we have that $p\pi r_{i-1}$ and hence $r_{i-1} \in \mathsf{To}(P)$, contradicting the induction hypothesis.

Next, suppose $r_n \in \mathsf{To}(P)$. However $r_n \in \mathsf{To}(P)$ and $q \in \mathsf{To}(P)$ would contradict Lemma 9, as $r \sqsubset q$. □

Lemma 11 (Unblocking positions are suffixes of From-nodes)**.** *Let π be a progress function for a process P and $p \in pos(P)$. If $\neg\mathsf{blocking}(P|_p)$ then there exist $q, r \in pos(P)$ such that $q \sqsubseteq p \sqsubset r$, $q \in \mathsf{From}(P)$ and $q\pi r$.*

Proof. Let q be the largest prefix of p such that $q \in \mathsf{From}(P)$. Such a p exists by Lemma 7. We consider 2 cases:

- If $q = p$ then take any r such that $p\pi r$. As $\mathcal{R}_\pi$ is left-total (Lemma 6) such an r necessarily exists. Using (4) in Lemma 5 we conclude.
- If $q \sqsubset p$ by Lemma 4 we have that there exists r such that $q\pi r$ and either $q \sqsubset p \sqsubseteq r$ or $q \sqsubset r \sqsubseteq p$. As $r \in \mathsf{To}(P)$, by Lemma 5, we moreover have that $r \neq p$. In the first case ($q \sqsubset p \sqsubset r$) we directly conclude. In the second case $q \sqsubset r \sqsubset p$ we consider 2 cases.
  1. There exists $p' \in pos(P)$ such that $r \sqsubset p' \sqsubset p$ and $p' \in \mathsf{From}(P)$. This case is not possible as it would contradict maximality of the prefix q.
  2. For all $p' \in pos(P)$ such that $r \sqsubset p' \sqsubset p$ it holds that $p' \notin \mathsf{From}(P)$. Hence we also have that $\neg\mathsf{blocking}(P|_{p'})$. From Definition 21, as $\mathsf{blocking}(P|_r)$ and $\mathsf{blocking}(P|_{p'})$ for any $r \sqsubset p' \sqsubset p$ and $\neg\mathsf{blocking}(P|_p)$ it follows that $p \in \mathsf{From}(P)$ and we would have chosen $q = p$.

□

Lemma 4. *Let π be the progress function of a process P. Then, $\mathcal{R}_\pi^{-1}$ is functional.*

Proof. By contradiction suppose that there exist $p_1 \neq p_2$ and q, such that $p_1\pi q$ and $p_2\pi q$. By Lemma 5 (4) we have that $p_1 \sqsubset q$ and $p_2 \sqsubset q$. Hence $p_1 \sqsubset p_2$ (or the symmetric case where $p_2 \sqsubset p_1$). Therefore $p_1 \sqsubset p_2 \sqsubset q$ and $p_2 \in \mathsf{From}(P)$ which contradicts Lemma 10. □

B.2 Correctness of α_{prog}

To show the correctness of the progress axiom, it is convenient to use the notion of a dependency graph as introduced in the Ph.D. theses of Benedikt Schmidt [27] and Simon Meier [24]. In particular, we make explicit use of [27, Lemma 3.10] (which slightly different in condition DG1, corresponds to [24, Theorem 3].

Lemma 12 (Trace-equivalance of dependency graphs to msr executions.)**.** *For all set of sets of labelled multiset rewriting rules R,*

$$\mathit{trace}(\mathit{exec}^{msr}(R)) = \{\, \mathit{trace}(dg) \mid \mathit{dgraphs}_E(R) \,\}.$$

Proof. Proof by induction on the sequence of multiset rewriting steps. □

With this notion (we refer to the mentioned works for details), we can describe the causal relations between two rule instantiations. Due to the fact that our transition preserves a strict ordering in terms of position, i.e., if a fact state_p appears, then any prefix that is not pointing at a NDC , needs to appear in the dependency graph, and moreover, the is a *chain* connecting them.

Lemma 13 (Backward chain)**.** *Let (I, D) be a dependency graph, and $ri \in I$ with $ri = r\tau$ for $r \in \{\, r \in [\![P]\!]_{=p} \mid \mathsf{state}_p(\tilde{x}) \in \mathit{prems}(r)$ for some $\tilde{x}\,\}$, from some p and τ.*

Then, there exists a sequence $((ri_0, p_0), \ldots, (ri_m, p_m))$ such that, for all $i \in \mathbb{N}_m$, $(\mathit{idx}(ri_i), j) \rightarrowtail (\mathit{idx}(ri_{i+1}), 0)$ (where j is 1 if $P|_{p_i} = Q \mid Q'$ and $P|_{p_{i+1}} = Q'$ and 0 otherwise), $ri_i \in [\![P]\!]_{=p_i}\tau$, $p_i \sqsubseteq p_{i+1}$, $ri = ri_m$ and $\{\, p_0, \ldots, p_m \,\} = \{\, p' \mid p' \sqsubseteq p \wedge P|_{=p'}$ is not NDC$\,\}$.

We call this sequence backward chain*. It furthermore holds that $(\mathit{idx}(ri_{init}), 0) \rightarrowtail (\mathit{idx}(ri_0), 0)$ where $ri_{init} = \textsc{Init}\tau$ is the (unique) instance of the initialisation rule.*

Proof. Induction over the length of p. Case distinction over rules in $[\![P]\!]$. Note that the set of terms $\tilde{t}$ in each fact $\mathsf{state}_p(\tilde{t})$ grows monotonically with the length of p and contains all variables in $[\![P]\!]_{=p'}$, for p' the parent position to p. □

Now we can argue the correctness of α_{prog}. Given the strong invariants used in Lemma 14 for the direction from SAPiC to msr, respectively Lemma 16 for the other direction, we can conduct this proof in a black-box manner: if these strong correspondences between executions hold, adhering to α_{prog} implies that all processes have been reduced up to a blocking position, and vice versa.

Lemma 2 (Correctness of α_{prog})**.** *The following two statements are equivalent for all $E_1, \ldots, E_m$:*

$$\exists s_1, \ldots, s_m.(s_0 \stackrel{E_1}{\Longrightarrow} \cdots \stackrel{E_m}{\Longrightarrow} (\mathcal{E}_m, \mathcal{S}_m, \mathcal{P}_m, \sigma_m, \mathcal{L}_m, \mathcal{U}_m)) \wedge \forall Q \in \mathcal{P}_m.\mathsf{blocking}(Q)$$

iff.

$$\begin{aligned}\exists S_1, \ldots, S_n, F_1, \ldots, F_n.\, \emptyset \xrightarrow{F_1}_* \cdots \xrightarrow{F_n}_* S_n \in \mathit{exec}^{msr}([\![P]\!]) \\ \wedge\, (E_1, \ldots, E_m) = \mathit{hide}_{\mathcal{F}_{res}}((F_1, \ldots, F_n)) \\ \wedge\, ((F_1, \ldots, F_n)) \vDash \alpha.\end{aligned}$$

Proof. We first show that the first statement implies the second, and then the opposite direction.

From first to second Let (by assumption) $s_1, \ldots, s_m$ such that $s_0 \stackrel{E_1}{\Longrightarrow} \cdots \stackrel{E_m}{\Longrightarrow} s_m$, with $s_m = (\mathcal{E}_m, \mathcal{S}_m, \mathcal{P}_m, \sigma_m, \mathcal{L}_m, \mathcal{U}_m)$ and $\forall Q \in \mathcal{P}_m.\mathsf{blocking}(Q)$.

By Lemma 14 in Appendix C, there are $S_1, \ldots, S_n$ such that

$$\emptyset \xrightarrow{F_1}_* \cdots \xrightarrow{F_n}_* S_n \in \mathit{exec}^{msr}([\![P]\!])$$

and $\mathit{hide}_{\mathcal{F}_{res}}([F_1, \ldots, F_n]) = [E_1, \ldots, E_m]$, and for all $i \in \mathbb{N}_n$, the conditions in Lemma 14 hold. We can apply Lemma 12 and conclude that there is $dg \in \mathit{dgraphs}_E([\![P]\!])$ with $dg = (I, D) \in \mathit{ginsts}([\![P]\!] \cup \{\, \mathit{Fresh} \,\})^* \times (\mathbb{N}^2 \times \mathbb{N}^2)$.

We show that, in addition to the conditions in Lemma 14, $(F_1, \ldots, F_n) \vDash \alpha_{prog}$. Assuming (for contradiction) that $(F_1, \ldots, F_n) \nvDash \alpha_{prog}$, there exists a rule instance ri_f in dg with action $ProgFrom_{p_f}(n)$ for some n, but no rule instance with $ProgTo_{p_t}(n)$ for any p_t such that $p_f \pi p_t$. From the definition of $[\![P]\!]$, we have that $n = prog_{p_f}\tau$, for τ the grounding substitution used to instantiate ri_f.

Let ri_{p_t} be a rule instance for some p_t with $ri_{p_t} \in [\![P|_{p_t}]\!]\tau'$, $\tau'(prog_{p_f}) = \tau'(prog_{p_f}) = n$ and $p_f \sqsubseteq p_t$, and $\mathsf{state}_{p_t'}(\tilde{t}) \in conclusions(ri_{p_t})$ for some $\tilde{x}$ such that p_t is of maximal length. (The shortest one would be ri_f).

By Lemma 13, the backward chain from ri_{p_t} would either contain ri_f, or there would be distinct instantiation of a rule in $[\![P]\!]|_{p_f}$ containing $ProgFrom_{p_f}$, with substitution τ'. The latter case, however, would require to distinct instantiations of FRESH for $\tau(prog_{p_f}) = \tau'(prog_{p_f}$, and thus contradict DG4. Thus, the backward chain from ri_{p_t} contains ri_f. By Lemma 8, there is either a position q such that $p_f \pi q$ in the backward chain, or there is one such that $p_f \pi q$ and $p_t \sqsubset q$. In the first case, by definition of the progress function (Definition 22), there would be an action $ProgTo(n)$ in the backward chain, contrary to the assumption. Hence there is q such that $p_f \pi q$ and $p_t \sqsubset q$.

Case distinction: The conclusion $\mathsf{state}_{p_t'}(\tilde{t'})$ in $conclusions(ri_{p_t})$ has no outgoing edge. Then, $\mathsf{state}_{p_t'}(\tilde{t}) \in S_n$. By Lemma 14 in Appendix C, there is a $Q \in \mathcal{P}_m$ such that $P|_{p_t'}\tau'' = Q\rho$ for some substitution τ'' and renaming ρ. As $\mathsf{blocking}(Q)$, $p_t' \in \mathsf{To}(P)$. Hence $q = p_t'$ (otherwise, Lemma 10 would be contradicted). Therefore, by definition of $[\![P]\!]$, $ProgTo_{p_t'}(prog_{p_t'})$ with $p_f \pi p_t'$ appears in trace, contrary to the assumption.[3]

Consider the opposite case: assume there is an outgoing edge for the conclusion $\mathsf{state}_{p_t'}(\tilde{t})$. In that case, there is a rule instantiation $ri_{t'}$ that would immeadeatly contradict the assumption that p is maximal if it would have a state-fact in the conclusion. This is immediate in all but the following three cases in the definition of $[\![P]\!]$: if the translation results from replication, a zero-process, or from one or more non-deterministic choice positions directly above a replication or zero process. In the first two cases, we observe that $\mathsf{blocking}(P_{p_t'})$ and thus $q = t_t'$, leading to contradicton as in the previous case. Dito if the non-deterministic choice was blocking. Consider the case of a (sequence of) unblocking non-deterministic choices above a replication or zero process. The resulting rule instance consuming $\mathsf{state}_p(p_t')(\tilde{t})$ would have an action $ProgTo_{p_t'}(prog_{p_t'})$ with $p_f \pi p_t'$, contrary to the assumption.

From second to first Assume that $(F_1, \ldots, F_n) \vDash \alpha_{prog}$. As

$$\emptyset \xrightarrow{F_1}_* \cdots \xrightarrow{F_n}_* S_n \in exec^{msr}([\![P]\!]),$$

we can apply Lemma 12 and conclude that there is $dg \in dgraphs_E([\![P]\!])$ with $dg = (I, D) \in ginsts([\![P]\!] \cup \{\, Fresh \,\})^* \times (\mathbb{N}^2 \times \mathbb{N}^2)$.

By Lemma 15 in Appendix C w.l.o.g. we can assume $trace(dg)$ to be normal. Then, by Lemma 16 in Appendix C, there are $s_1, \ldots, s_{n'}$, such that

$$s_0 \xRightarrow{E_1}_* \cdots \xRightarrow{E_{n'}}_* s_{n'} = (\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{P}_{n'}, \sigma_{n'}, \mathcal{L}_{n'}, \mathcal{U}_{n'})$$

such that the conditions in Lemma 16 hold. Assume (for contradiction), that there is $Q \in \mathcal{P}_{n'}$ such that $\neg\mathsf{blocking}(Q)$. By the third condition of Lemma 12, we get that there is $\mathsf{state}_p(\tilde{t}) \in S_n$ for a position p, a substitution θ, a renaming ρ, and a rule instance ri such that

$$\tilde{t} = \tilde{x}\theta \tag{7}$$

$$P|_p\tau = Q\rho, \tag{8}$$

for τ defined as follows:

$$\begin{aligned} \tau(x) &:= \theta(x) && \text{if } x \text{ not a reserved variable} \\ \rho(a) &:= a' && \text{if } \theta(n_a) = a' \end{aligned}$$

[3] Observe that in case of non-deterministic choice, the substitution operation alters only the premise of any rule resulting from the translation of the child processes.

We fix $p, \tilde{t}, p, \theta, \rho$ and ri. We chose a rule instantiation ri' such that $\mathsf{state}_p(\tilde{t}) \in \mathit{conclusions} ri')$ and such that there is no outgoing edge from $(\mathit{idx}(ri'), j)$, where j is the position of the state_p-fact. As $\mathsf{state}_p(\tilde{t}) \in S_n$, we know that such an ri' exists. By Lemma 11, we have that there is $p_f \in \mathsf{From}(P)$.

By Lemma 13, there exists a backward chain from ri' including (ri_f, p_f) with $ri_f = [\![P]\!]_{=p_f}\tau$. Hence $\mathit{ProgFrom}(\mathit{prog}_{p_f}\tau)$ appears in the trace. As $(F_1, \ldots, F_n) \vDash \alpha_{prog}$, there must also be an action containing $\mathit{ProgTo}_{p_t}(\mathit{prog}_{p_f}\tau)$ in the trace, where p_t might be any position such that $p_f \pi p_t$. Let $ri_t = [\![P]\!]_{=p_t}\tau'$ be the rule instantiation that contains this action, i. e., $\tau'(\mathit{prog}_{p_f}) = \tau(\mathit{prog}_{p_f})$.

Consider the backward chain from ri_t. More precisely, its suffix starting at the first rule instance for position p_f with action $\mathit{ProgFrom}_{p_f}(\mathit{prog}_{p_f}\tau')$ (which exists by definition of $[\![P]\!]$). All premises at position 0 are linear, except for possibly the very first in the list, as we will now show. By Lemma 10, we have that for all p' such that $p_f \sqsubset p' \sqsubset p_t$, $p' \notin \mathsf{From}(P)$ and $p' \notin \mathsf{To}(P)$. This means, in particular, that by definition of π and From, for all such p', $P|_{p'}$ is no replication. As the only permanent fact is named $\mathsf{state}^{\mathsf{semi}}$ appears only in the translation of a replication, the described suffix of the backward chain has only linear premises for each element at position 0.

From this, we conclude that either the first rule of this suffix (starting with $(\mathit{ProgFrom}_{p_f}(\mathit{prog}_{p_f}\tau'))$ is ri_f and hence ri' appears in this suffix, or that the first rule of this suffix is different from ri_f. The second case can readily be excluded, as, by definition of $[\![P]\!]$, this would imply two distinct instantiations of the FRESH rule for $\mathit{prog}_{p_f}\tau = \mathit{prog}_{p_f}\tau'$ contradicting DG4.

If ri' is in this suffix however, it can only be a last element, as there is no outgoing edge from $(\mathit{idx}(ri'), j)$. This would imply that $p = p_t$, but as $\neg\mathsf{blocking}(P|_p)$, by definition of the progress function (Definition 22), $p \notin \mathsf{To}(P)$. Thus, no action $\mathit{ProgFrom}_{p_f}$ would appear in $[\![P]\!]$, contradicting the existence of ri_t. □

Lemma 3 (trace-equivalence)**.** *For all well-formed P, then*

$$\mathit{traces}^{ppi}(P) = \mathit{hide}_{\mathcal{F}_{res}}(\mathit{filter}_{\alpha}(\mathit{traces}^{msr}([\![P]\!]))).$$

Proof. By Definition 3:

$$\begin{aligned}\mathit{traces}^{ppi}(P) &= \mathit{traces}^{pi}(P) \setminus \underbrace{\{\, (F_1, \ldots, F_n) \mid \forall s_1, \ldots, s_n.(s_0 \xRightarrow{F_1}_* s_1 \ldots \xRightarrow{F_n}_* s_n) \Rightarrow \neg \mathit{final}(s_n) \,\}}_{=: Tr_{incomplete}} \\ &= \mathit{hide}_{\mathcal{F}_{res}}(\mathit{filter}_{\alpha \setminus \{\, \alpha_{prog} \,\}}(\mathit{traces}^{msr}([\![P]\!]))) \setminus Tr_{incomplete}\end{aligned}$$

Here, we use s_0 as a short-hand to $(\emptyset, \emptyset, \{P\}, \emptyset, \emptyset, \emptyset)$. Since $\mathit{filter}_{\alpha}(\mathit{traces}^{msr}(R)) = \{\, (F_1, \ldots, F_n) \mid \exists S_1, \ldots, S_n.\emptyset \xrightarrow{F_1}_* \cdots \xrightarrow{F_n}_* S_n \in \mathit{exec}^{msr}(R) \wedge \forall \theta.((F_1, \ldots, F_n), \theta) \vDash \alpha \,\}$, we have:

$$\begin{aligned}&= \mathit{hide}_{\mathcal{F}_{res}}(\{\, (F_1, \ldots, F_n) \mid \exists S_1, \ldots, S_n.\emptyset \xrightarrow{F_1}_* \cdots \xrightarrow{F_n}_* S_n \in \mathit{exec}^{msr}([\![P]\!]) \\ &\qquad \wedge \forall \theta.((F_1, \ldots, F_n), \theta) \vDash \alpha \setminus \{\, \alpha_{prog} \,\} \,\} \setminus Tr_{incomplete}\end{aligned}$$

By definition of *hide* (Definition 18)

$$\begin{aligned}&= \{\, \mathit{hide}_{\mathcal{F}_{res}}((F_1, \ldots, F_n)) \mid \exists S_1, \ldots, S_n.\emptyset \xrightarrow{F_1}_* \cdots \xrightarrow{F_n}_* S_n \in \mathit{exec}^{msr}([\![P]\!]) \\ &\qquad \wedge \forall \theta.((F_1, \ldots, F_n), \theta) \vDash \alpha \setminus \{\, \alpha_{prog} \,\} \,\} \setminus Tr_{incomplete}\end{aligned}$$

We reinsert $Tr_{incomplete}$.

$$\begin{aligned}&= \{\, \mathit{hide}_{\mathcal{F}_{res}}((F_1, \ldots, F_n)) \mid \exists S_1, \ldots, S_n.\emptyset \xrightarrow{F_1}_* \cdots \xrightarrow{F_n}_* S_n \in \mathit{exec}^{msr}([\![P]\!]) \\ &\qquad \wedge \forall \theta.((F_1, \ldots, F_n), \theta) \vDash \alpha \setminus \{\, \alpha_{prog} \,\} \\ &\qquad \wedge \exists s_1, \ldots, s_m, E_1, \ldots, E_m.(s_0 \xRightarrow{E_1}_* \cdots \xRightarrow{E_m}_* s_m) \wedge \mathit{final}(s_m) \\ &\qquad \wedge (E_1, \ldots, E_m) = \mathit{hide}_{\mathcal{F}_{res}}((F_1, \ldots, F_n)) \,\}\end{aligned}$$

This can be rewritten as follows.

$$
\begin{aligned}
&= \{\, (E_1, \ldots, E_m) \mid \\
&\quad \exists S_1, \ldots, S_n, F_1, \ldots, F_n. \\
&\qquad \emptyset \xrightarrow{F_1}_* \cdots \xrightarrow{F_n}_* S_n \in \mathit{exec}^{msr}([\![P]\!]) \\
&\qquad \wedge \forall \theta. ((F_1, \ldots, F_n), \theta) \vDash \alpha \setminus \{\, \alpha_{prog} \,\} \\
&\qquad \wedge (E_1, \ldots, E_m) = \mathit{hide}_{\mathcal{F}_{res}}((F_1, \ldots, F_n)) \\
&\quad \wedge \exists s_1, \ldots, s_m, E'_1, \ldots, E'_{m'}. \\
&\qquad (s_0 \xRightarrow{E'_1}_* \cdots \xRightarrow{E'_{m'}}_* s_{m'}) \wedge \mathit{final}(s_{m'}) \\
&\qquad \wedge (E'_1, \ldots, E'_{m'}) = \mathit{hide}_{\mathcal{F}_{res}}((F_1, \ldots, F_n)) \,\}
\end{aligned}
$$

We apply Lemma 2 and the definition of $\vDash$ for the conjunctive case.

$$
\begin{aligned}
&= \{\, (E_1, \ldots, E_m) \mid \\
&\quad \exists S_1, \ldots, S_n, F_1, \ldots, F_n. \\
&\qquad \emptyset \xrightarrow{F_1}_* \cdots \xrightarrow{F_n}_* S_n \in \mathit{exec}^{msr}([\![P]\!]) \\
&\qquad \wedge \forall \theta. ((F_1, \ldots, F_n), \theta) \vDash \alpha \setminus \{\, \alpha_{prog} \,\} \\
&\qquad \wedge (E_1, \ldots, E_m) = \mathit{hide}_{\mathcal{F}_{res}}((F_1, \ldots, F_n)) \\
&\quad \wedge \exists S'_1, \ldots, S'_{n'}, F'_1, \ldots, F'_{n'}, E'_1, \ldots, E'_{m'}. \\
&\qquad \emptyset \xrightarrow{F'_1}_* \cdots \xrightarrow{F'_{n'}}_* S'_{n'} \in \mathit{exec}^{msr}([\![P]\!]) \\
&\qquad \wedge \forall \theta. ((F'_1, \ldots, F'_{n'}), \theta) \vDash \alpha \\
&\qquad \wedge (E'_1, \ldots, E'_{m'}) = \mathit{hide}_{\mathcal{F}_{res}}((F'_1, \ldots, F'_{n'})) \\
&\qquad \wedge (E'_1, \ldots, E'_{m'}) = \mathit{hide}_{\mathcal{F}_{res}}((F_1, \ldots, F_n)) \,\} \\
&= \{\, (E_1, \ldots, E_m) \mid \\
&\quad \exists S_1, \ldots, S_n, F_1, \ldots, F_n. \\
&\qquad \emptyset \xrightarrow{F_1}_* \cdots \xrightarrow{F_n}_* S_n \in \mathit{exec}^{msr}([\![P]\!]) \\
&\qquad \wedge \forall \theta. ((F_1, \ldots, F_n), \theta) \vDash \alpha \setminus \{\, \alpha_{prog} \,\} \\
&\qquad \wedge (E_1, \ldots, E_m) = \mathit{hide}_{\mathcal{F}_{res}}((F_1, \ldots, F_n)) \\
&\quad \wedge \exists S'_1, \ldots, S'_{n'}, F'_1, \ldots, F'_{n'}. \\
&\qquad \emptyset \xrightarrow{F'_1}_* \cdots \xrightarrow{F'_{n'}}_* S'_{n'} \in \mathit{exec}^{msr}([\![P]\!]) \\
&\qquad \wedge \forall \theta. ((F'_1, \ldots, F'_{n'}), \theta) \vDash \alpha \\
&\qquad \mathit{hide}_{\mathcal{F}_{res}}((F'_1, \ldots, F'_{n'})) = \mathit{hide}_{\mathcal{F}_{res}}((F_1, \ldots, F_n)) \,\} \\
&= \{\, (E_1, \ldots, E_m) \mid \\
&\quad \exists S_1, \ldots, S_n, F_1, \ldots, F_n, S'_1, \ldots, S'_{n'}, F'_1, \ldots, F'_{n'}. \\
&\qquad \emptyset \xrightarrow{F_1}_* \cdots \xrightarrow{F_n}_* S_n \in \mathit{exec}^{msr}([\![P]\!]) \\
&\qquad \wedge \forall \theta. ((F_1, \ldots, F_n), \theta) \vDash \alpha \setminus \{\, \alpha_{prog} \,\} \\
&\qquad \wedge (E_1, \ldots, E_m) = \mathit{hide}_{\mathcal{F}_{res}}((F_1, \ldots, F_n)) \\
&\qquad \emptyset \xrightarrow{F'_1}_* \cdots \xrightarrow{F'_{n'}}_* S'_{n'} \in \mathit{exec}^{msr}([\![P]\!]) \\
&\qquad \wedge \forall \theta. ((F'_1, \ldots, F'_{n'}), \theta) \vDash \alpha \\
&\qquad \mathit{hide}_{\mathcal{F}_{res}}((F'_1, \ldots, F'_{n'})) = (E_1, \ldots, E_m) \,\}
\end{aligned}
$$

This can be simplified, as all $S'_1, \ldots, S_{n'}, F'_1, \ldots, F'_{n'}$ that satisfy the second triple of conjunctions satisfy the first triple, too.

$$\begin{aligned}
&= \{\, hide_{\mathcal{F}_{res}}((F_1, \ldots, F_n)) \mid \emptyset \xrightarrow{F_1}_* \cdots \xrightarrow{F_n}_* S_n \in exec^{msr}([\![P]\!]) \\
&\qquad \wedge \forall \theta.((F_1, \ldots, F_n), \theta) \vDash \alpha \,\} \\
&= hide_{\mathcal{F}_{res}}(filter_\alpha(traces^{msr}([\![P]\!]))).
\end{aligned}$$

□

C Proof of Lemma 1

In this Appendix, we will present the modifications to the original proof [19]. To keep the presentation concise, we abbreviate $\alpha \setminus \alpha_{prog}$ by α' in this section, i. e.,

$$\alpha' := \alpha_{init} \wedge \alpha_{pred} \wedge \alpha_{noteq} \wedge \alpha_{in} \wedge \alpha_{notin} \wedge \alpha_{lock} \wedge \alpha_{inev} \wedge \alpha_{resil}$$

We will furthermore ignore the actions *ProgTo* and *ProgFrom* altogether, as they can neither appear in a process, nor in a security property (due to the respective well-formedness condition) and are not referred to in α'.

We use $[\![P]\!]_p$ for the set of rules of the translated process at position p, which is formally defined in [19, Definition 19].

C.1 Proof that $traces^{pi}(P) \subseteq hide(filter(traces^{msr}([\![P]\!])))$

Definition 24. *Let P be a ground process, $\mathcal{P}$ be a multiset of processes and S a multiset of ground facts. We write $\mathcal{P} \leftrightarrow_P S$ if there exists a bijection between $\mathcal{P}$ and the multiset $\{\, \mathsf{state}_p(\tilde{t}) \mid \exists p, \tilde{t}.\ \mathsf{state}_p(\tilde{t}) \in^\# S \,\}^\# \cup^\# \{\, \mathsf{state}_p^{\mathsf{semi}}(\tilde{t}) \mid \exists p, \tilde{t}.\ \mathsf{state}_p^{\mathsf{semi}}(\tilde{t}) \in^\# S \,\}^\#$ such that whenever $Q \in^\# \mathcal{P}$ is mapped to $\mathsf{state}_p(\tilde{t}) \in^\# S$ or $\mathsf{state}_p^{\mathsf{semi}}(\tilde{t}) \in^\# S$, we have that*

1. *$P|_p\tau = Q\rho$, for some substitution τ and some bijective renaming ρ of fresh, but not bound names in Q, and*
2. *$\exists ri \in_E ginsts([\![P]\!]_{=p}).\ \mathsf{state}_p(\tilde{t}) \in prems(ri) \wedge \mathsf{state}_p^{\mathsf{semi}}(\tilde{t}) \in prems(ri)$.*

When $\mathcal{P} \leftrightarrow_P S$, $Q \in^\# \mathcal{P}$ and $\mathsf{state}_p(\tilde{t}) \in^\# S$ we also write $Q \leftrightarrow_P \mathsf{state}_p(\tilde{t})$ if this bijection maps Q to $\mathsf{state}_p(\tilde{t})$. Note that, if $P|_p$ is not a replication, then there is no $\mathsf{state}_p^{\mathsf{semi}}$, i. e., thus, if it is clear from context that $Q \in \mathcal{P}$ is not a replication, we can assume there to be $Q \leftrightarrow_P \mathsf{state}_p(\tilde{t})$.

Lemma 14. *Let P be a well-formed ground process. If*

$$(\mathcal{E}_0, \mathcal{S}_0, \mathcal{P}_0, \sigma_0, \mathcal{L}_0, \mathcal{U}_0) \xrightarrow{E_1} (\mathcal{E}_1, \mathcal{S}_1, \mathcal{P}_1, \sigma_1, \mathcal{L}_1, \mathcal{U}_1) \xrightarrow{E_2} \ldots \xrightarrow{E_n} (\mathcal{E}_n, \mathcal{S}_n, \mathcal{P}_n, \sigma_n, \mathcal{L}_n, \mathcal{U}_n)$$

where $(\mathcal{E}_0, \mathcal{S}_0, \mathcal{P}_0, \sigma_0, \mathcal{L}_0, \mathcal{U}_0) = (\emptyset, \emptyset, \emptyset, \{\, P \,\}, \emptyset, \emptyset)$ then there are $(F_1, S_1), \ldots, (F_{n'}, S_{n'})$ such that

$$S_0 \xrightarrow{F_1}_{[\![P]\!]} S_1 \xrightarrow{F_2}_{[\![P]\!]} \ldots \xrightarrow{F_{n'}}_{[\![P]\!]} S_{n'} \in exec^{msr}([\![P]\!])$$

and there exists a monotonic, strictly increasing function $f\colon \mathbb{N}_n \to \mathbb{N}_{n'}$ such that $f(n) = n'$, $S_0 = \emptyset$, and for all $i \in \mathbb{N}_n$

1. *$\mathcal{E}_i = \{\, a \mid ProtoNonce(a) \in \bigcup_{1 \le j \le f(i)} F_j \,\}$*
2. *$\forall t \in \mathcal{M}.\ \mathcal{S}_i(t) = \begin{cases} u & \text{if } \exists j \le f(i).\mathrm{Insert}(t,u) \in F_j \\ & \quad \wedge \forall j', u'.j < j' \le f(i) \to \mathrm{Insert}(t,u') \notin_E F_{j'} \wedge \mathrm{Delete}(t) \notin_E F_{j'} \\ \bot & \text{otherwise} \end{cases}$*
3. *$\mathcal{P}_i \leftrightarrow_P S_{f(i)}$*

4. $\{\, x\sigma_i \mid x \in \mathbf{D}(\sigma_i) \,\}^{\#} = \{\, t \mid \exists k \in \mathbb{N}_{f(i)-1}.\mathsf{Out}(t) \in S_{k+1} \setminus S_k \,\}^{\#}$

5. $\mathcal{L}_i =_E \{\, t \mid \exists j \leq f(i), u.\ \mathrm{Lock}(u,t) \in_E F_j \wedge \forall j < k \leq f(i).\mathrm{Unlock}(u,t) \notin_E F_k \,\}$

6. $(F_1, \ldots, F_{n'}) \vDash \alpha'$ *where* α' *is defined as* $\alpha \setminus \alpha_{prog}$ *(see beginning of Appendix).*

7. $\exists k.\ f(i-1) < k \leq f(i)$ *and* $E_i = F_k$ *and* $\cup_{\{\, k \mid f(i-1) < j \leq f(i), j \neq k \,\}} F_j \subseteq \mathcal{F}_{res}$

8. $\mathcal{U}_i =_E \{\, m \mid \exists j \leq f(i), mid.\ \mathrm{Send}(mid, m) \in_E F_j \wedge \forall j'.\mathrm{Receive}(mid, m) \notin_E F_{j'} \,\}^{\#}$

Proof. We proceed by induction over the number of transitions n. The base case remains unchanged, Condition 8 holds trivially. The inductive step is altered in case of:

- non-deterministic choice,
- replication, and
- message input and out on reliable channels (input/output on the public channel is a special case of the previous case for general message input/output).

Note that *ProgFrom* and *ProgTo* events are in $\mathcal{F}_{res}$ and only appear in α_{resil}, hence we ignore them for conciseness.

Assume the invariant holds for $n-1 \geq 0$. We have to show that the lemma holds for n transitions

$$(\mathcal{E}_0, \mathcal{S}_0, \mathcal{P}_0, \sigma_0, \mathcal{L}_0, \mathcal{U}_0) \xrightarrow{E_1} (\mathcal{E}_1, \mathcal{S}_1, \mathcal{P}_1, \sigma_1, \mathcal{L}_1, \mathcal{U}_1) \xrightarrow{E_2} \ldots \xrightarrow{E_n} (\mathcal{E}_n, \mathcal{S}_n, \mathcal{P}_n, \sigma_n, \mathcal{L}_n, \mathcal{U}_n)$$

By induction hypothesis, we have that there exists a monotonically increasing function from $\mathbb{N}_{n-1} \to \mathbb{N}_{n'}$ and an execution

$$\emptyset \xrightarrow{F_1}_{[\![P]\!]} S_1 \xrightarrow{F_2}_{[\![P]\!]} \ldots \xrightarrow{F_{n'}}_{[\![P]\!]} S_{n'} \in exec^{msr}([\![P]\!])$$

such that Conditions 1 to 7 hold. Let f_p be this function and note that $n' = f_p(n-1)$. Fix a bijection such that $\mathcal{P}_{n-1} \leftrightarrow_P S_{f_p(n-1)}$. We will abuse notation by writing $P \leftrightarrow_P \mathsf{state}_p(\tilde{t})$, if this bijection goes from P to $\mathsf{state}_p(\tilde{t})$.

We now proceed by case distinction over the type of transition from $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1}, \sigma_{n-1}, \mathcal{L}_{n-1}, \mathcal{U}_{n-1})$ to $(\mathcal{E}_n, \mathcal{S}_n, \mathcal{P}_n, \sigma_n, \mathcal{L}_n, \mathcal{U}_n)$. We will (unless stated otherwise) extend the previous execution by a number of steps, say s, from $S_{n'}$ to some $S_{n'+s}$, and prove that Conditions 1 to 8 hold for n (since by induction hypothesis, they hold for all $i < n$) and a function $f\colon \mathbb{N}_n \to \mathbb{N}_{n'+s}$ that is defined as follows:

$$f(i) := \begin{cases} f_p(i) & \text{if } i \in \mathbb{N}_{n-1} \\ n' + s & \text{if } i = n \end{cases}$$

Case: Replication. $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1} = \mathcal{P}' \cup \{!Q\}, \sigma_{n-1}, \mathcal{L}_{n-1}, \mathcal{U}_{n-1}) \to (\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}' \cup \{!Q, Q\}, \sigma_{n-1}, \mathcal{L}_{n-1}, \mathcal{U}_{n-1})$

Let p and $\tilde{t}$ such that $!Q \leftrightarrow_P \mathsf{state}_p(\tilde{t})$. or $!Q \leftrightarrow_P \mathsf{state}_p^{\mathsf{semi}}(\tilde{t})$. By Definition 24, there is a $ri \in ginsts([\![P]\!]_{=p})$ such that $\mathsf{state}_p(\tilde{t})$ or $\mathsf{state}_p^{\mathsf{semi}}(\tilde{t})$ is part of its premise. By definition of $[\![P]\!]_{=p}$, we can choose

$$ri = [\mathsf{state}_p(\tilde{t})] -[\ \ ]\to [\mathsf{state}_p^{\mathsf{semi}}(\tilde{t})], \text{ and} ri' = [\mathsf{state}_p^{\mathsf{semi}}(\tilde{t})] -[\ \ ]\to [\mathsf{state}_{p\cdot 1}(\tilde{t})],$$

or, in case $!Q \leftrightarrow_P \mathsf{state}_p^{\mathsf{semi}}(\tilde{t})$, only ri'. We can extend the previous execution by ri and ri', respectively ri',therefore:

$$\emptyset \xrightarrow{F_1}_{[\![P]\!]} S_1 \xrightarrow{F_2}_{[\![P]\!]} \ldots \xrightarrow{F_{n'}}_{[\![P]\!]} S_{n'} \xrightarrow{(ri)}_{[\![P]\!]} S_{n'+1} \xrightarrow{(ri')}_{[\![P]\!]} S_{n'+2} \in exec^{msr}([\![P]\!]),$$

with $S_{n'+2} = S_{n'} \setminus^{\#} \{\, \mathsf{state}_p(\tilde{t}) \,\}^{\#} \cup^{\#} \{\, \mathsf{state}_p^{\mathsf{semi}}(\tilde{t}), \mathsf{state}_{p\cdot 1}(\tilde{t}) \,\}^{\#}$. or

$$\emptyset \xrightarrow{F_1}_{[\![P]\!]} S_1 \xrightarrow{F_2}_{[\![P]\!]} \ldots \xrightarrow{F_{n'}}_{[\![P]\!]} S_{n'} \xrightarrow{(ri')}_{[\![P]\!]} S_{n'+1} \in exec^{msr}([\![P]\!]),$$

with $S_{n'+1} = S_{n'} \cup^{\#} \{\, \mathsf{state}_{p\cdot 1}(\tilde{t}) \,\}^{\#}$. Condition 3 holds because $\mathcal{P}_n = \mathcal{P}_{n-1} \cup^{\#} \{Q\}$ and $\{Q\} \leftrightarrow_P \{\mathsf{state}_{p\cdot 1}(\tilde{t})\}$ (by definition of $[\![P]\!]_{=p}$). Condition 1, Condition 2, Condition 4, Condition 5, Condition 6 Condition 7 and Condition 8 hold trivially.

Case: Non-deterministic choice. $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1} = \mathcal{P} \cup \{Q + S\}, \sigma_{n-1}, \mathcal{L}_{n-1}, \mathcal{U}_{n-1}) \rightarrow (\mathcal{E}_n, \mathcal{S}_n, \mathcal{P}_n, \sigma_n, \mathcal{L}_n, \mathcal{U}_n)$

Here, $\mathcal{P}_n$ is either $\mathcal{P} \cup^{\#} \{\, Q' \,\}^{\#}$ or $\mathcal{P}$. A necessary condition for this step is that

$$(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1} = \mathcal{P} \cup \{Q\}, \sigma_{n-1}, \mathcal{L}_{n-1}, \mathcal{U}_{n-1}) \rightarrow (\mathcal{E}_n, \mathcal{S}_n, \mathcal{P}_n, \sigma_n, \mathcal{L}_n, \mathcal{U}_n), \text{ or}$$
$$(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1} = \mathcal{P} \cup \{S\}, \sigma_{n-1}, \mathcal{L}_{n-1}, \mathcal{U}_{n-1}) \rightarrow (\mathcal{E}_n, \mathcal{S}_n, \mathcal{P}_n, \sigma_n, \mathcal{L}_n, \mathcal{U}_n).$$

If Q or S are themselves processes of form $Q' + Q''$ or $S' + S''$, the necessary condition only applies if Q', Q'', S' or S'' can make a transition. Therefore, let T, w.l.o.g., be any child process of $Q + S$ that itself is not a non-deterministic choice, and

$$(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1} = \mathcal{P} \cup \{\, T \,\}, \sigma_{n-1}, \mathcal{L}_{n-1}, \mathcal{U}_{n-1}) \rightarrow (\mathcal{E}_n, \mathcal{S}_n, \mathcal{P}_n, \sigma_n, \mathcal{L}_n, \mathcal{U}_n),$$

and $\mathcal{P}_n = \mathcal{P} \cup^{\#} \{\, T' \,\}^{\#}$ for some T' or $\mathcal{P}_n = \mathcal{P}$. In both cases, let p and $\tilde{t}$ such that $Q + S \leftrightarrow_P \mathsf{state}_p(\tilde{t})$ or $Q + S \leftrightarrow_P \mathsf{state}_p^{\mathsf{semi}}(\tilde{t})$. By Definition 24, there is a $ri \in \mathit{ginsts}(\llbracket P \rrbracket_{=p})$ such that $\mathsf{state}_p(\tilde{t})$ or $\mathsf{state}_p^{\mathsf{semi}}(\tilde{t})$ is part of its premise. In the latter case, $T = 0$, as the corresponding transition is the only one decreasing the cardinality of the process set. In that case, $\mathcal{E}_n = \mathcal{E}_{n-1}$, $\mathcal{S}_n = \mathcal{S}_{n-1}$, $\sigma_n = \sigma_{n-1}$, $\mathcal{L}_n = \mathcal{L}_{n-1}$ and $\mathcal{U}_n = \mathcal{U}_{n-1}$. By definition of $\llbracket Q + S \rrbracket_{=p}$ and the fact that $T = 0$ is the first child-process of $Q + S$ which is not a non-deterministic choice, there is a rule $[\mathsf{state}_p(\tilde{x})] \,-[\ \ ]\rightarrow\, []$. Applying this rule, Condition 1 to Condition 8 hold trivially.

If $\mathcal{P}_n = \mathcal{P} \cup^{\#} \{\, T' \,\}^{\#}$ for some T', then the case corresponding to the transition from

$$(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1} = \mathcal{P} \cup \{\, T \,\}, \sigma_{n-1}, \mathcal{L}_{n-1}, \mathcal{U}_{n-1}) \rightarrow (\mathcal{E}_n, \mathcal{S}_n, \mathcal{P} \cup^{\#} \{\, T' \,\}^{\#}, \sigma_n, \mathcal{L}_n, \mathcal{U}_n)$$

applies, with the following modifications:

- The position p is substituted by $p \cdot p'$, where p' is the position of T relative to $Q + S$. Thus, from the invariant, we have $T \leftrightarrow_P \mathsf{state}_{p \cdot p'}(\tilde{t})$. or $T \leftrightarrow_P \mathsf{state}_{p \cdot p'}^{\mathsf{semi}}(\tilde{t})$.
- Whenever a rule instance ri or ri' is chosen, state_p is in the premise (rather than $\mathsf{state}_{p \cdot p'}$).

Note that, since T is not a non-deterministic choice, and none of the other cases refers to this case, this argument is not circular.

Case: $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1} = \mathcal{P}' \cup \{\, \mathrm{out}('r', m); Q \,\}, \sigma_{n-1}, \mathcal{L}_{n-1}, \mathcal{U}_{n-1}) \rightarrow (\mathcal{E}_{n-1}, \mathcal{S}_{n-1},$
$\mathcal{P}' \cup^{\#} \{\, Q \,\}, \sigma_{n-1} \cup \{^{m}/_{x}\}, \mathcal{L}_{n-1}, \mathcal{U}_{n-1} \cup^{\#} \{\, m \,\}^{\#})$. This step requires that x is fresh and $\nu\mathcal{E}_{n-1}.\sigma \vdash t$. By Definition 24, there are p and $\tilde{t}$ such that $\{\mathrm{out}('r', m); Q\} \leftrightarrow_P \mathsf{state}_p(\tilde{t})$. and hence $ri \in \mathit{ginsts}(\llbracket P \rrbracket_{=p})$ such that $\mathsf{state}_p(\tilde{t})$ is part of its premise. From the definition of $\llbracket P \rrbracket_{=p}$ and MID, we can choose

$$ri = \quad [\mathsf{Fr}(i)] \,-[]\rightarrow\, [\mathsf{MID}_{\mathsf{rcv}}(i), \mathsf{MID}_{\mathsf{snd}}(i)] \text{ and}$$
$$ri' = \quad [\mathsf{state}_p(\tilde{t}), \mathsf{MID}_{\mathsf{snd}}(i)] \,-[\, \mathrm{Send}(i, m) \,]\rightarrow\, [\mathsf{state}_{p \cdot 1}(\tilde{t}, m), \mathsf{Out}(m)]$$

for some fresh i. From S, we can now go two steps further, using ri and ri':

$$\emptyset \xrightarrow{F_1}_{\llbracket P \rrbracket} S_1 \ldots \xrightarrow{F_{n'}}_{\llbracket P \rrbracket} S_{n'} \xrightarrow{(ri)} S_{n'+1} \xrightarrow{\mathrm{Send}(i,m)}_{\llbracket P \rrbracket} S_{n'+2} \in \mathit{exec}^{msr}(\llbracket P \rrbracket)$$

where $S_{n'+1} = S \cup^{\#} \{\, \mathsf{MID}_{\mathsf{rcv}}(i), \mathsf{MID}_{\mathsf{snd}}(i) \,\}^{\#}$ and $S_{f(n)} = S \setminus^{\#} \{\, \mathsf{state}_p(\tilde{t}) \,\} \cup^{\#} \{\, \mathsf{state}_{p \cdot 1}(\tilde{t}, i), \mathsf{Out}(m) \,\}$.

As Send is reserved, Condition 7 holds.

Since $\mathcal{P}_n = \mathcal{P}_{n-1} \setminus \{\mathrm{out}('r', m); Q\} \cup \{Q\}$ and $\{Q\} \leftrightarrow_P \{\mathsf{state}_{p \cdot 1}(\tilde{t})\}$ (by definition of $\llbracket P \rrbracket_{=p}$), we have that $\mathcal{P}_n \leftrightarrow_P S_{f(n)}$, i. e., Condition 3 holds. Condition 4 holds since m was added to σ_{n-1} and $\mathsf{Out}(m)$ added to $S_{f(n-1)}$. Condition 8 holds, since a new transition marked $\mathrm{Send}(i, m)$ was added to the trace and $m = \mathcal{U}_n \setminus \mathcal{U}_{n-1}$. Condition 1, Condition 2, and Condition 5 hold trivially.

Case: $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1} = \mathcal{P}' \cup \{\text{in}('r', N); Q\}, \sigma_{n-1}, \mathcal{L}_{n-1}, \mathcal{U}_{n-1}) \to (\mathcal{E}_{n-1}, \mathcal{S}_{n-1},$
$\mathcal{P}' \cup^{\#} \{ Q\theta \}, \sigma_{n-1}, \mathcal{L}_{n-1}), \mathcal{U}_{n-1}$. This step requires that θ is grounding for N and that $\nu\mathcal{E}_{n-1}.\sigma_{n-1} \vdash N\theta$. Using [19, Lemma 8], we have that there is an execution $\emptyset \xrightarrow{F_1} S_1 \xrightarrow{F_2} \ldots \xrightarrow{F_{f(n-1)}} S_{f(n-1)} \to^* S \in exec_E^{msr}([\![P]\!])$ such that $!\mathsf{K}(N\theta) \in_E S$ and $S_{f(n-1)} \to_R^* S$ for $R = \{ \text{MDOut}, \text{MDPub}, \text{MDFresh}, \text{MDAppl} \}$. Let $\emptyset \xrightarrow{F_1} S_1 \xrightarrow{F_2} \ldots \xrightarrow{F_{f(n-1)}} S_{f(n-1)} \to_R^* \overline{S} \in exec_E^{msr}([\![P]\!])$ with $!\mathsf{K}(N\theta) \in_E \overline{S}$. Let p and $\tilde{t}$ be such that, $\text{in}(t, N); Q \leftrightarrow_P \mathsf{state}_p(\tilde{t})$. By Definition 24 there is a $ri \in ginsts([\![P]\!]_{=p})$ such that $\mathsf{state}_p(\tilde{t})$ is part of its premise. By definition of $[\![P]\!]_{=p}$ and the fact that θ is grounding for $N\theta$, we chose two transitions as follows:

$$\begin{aligned} ri = \quad & [\mathsf{Fr}(i)] \dashrightarrow [\mathsf{MID}_{\mathsf{rcv}}(i), \mathsf{MID}_{\mathsf{snd}}(i)] \text{ and} \\ ri' = \quad & [\mathsf{state}_p(\tilde{t}), \mathsf{In}(m), \mathsf{MID}_{\mathsf{rcv}}(i)] \dashrightarrow [\text{ Receive}(i, m)] \to [\mathsf{state}_{p \cdot 1}(\tilde{t} \cup (vars(N)\theta)], \end{aligned}$$

where i is fresh.

From $S_{n'}$, we can first apply the above transition $S_{n'} \to_R^* \overline{S}$, and then, (since $!\mathsf{K}(t), !\mathsf{K}(N\theta)$, $\mathsf{state}_p(\tilde{x}) \in \overline{S}$), MDIn, ri and ri':

$$\begin{aligned} \emptyset \xrightarrow{F_1}_{[\![P]\!]} S_1 \ldots \xrightarrow{F_{n'}}_{[\![P]\!]} S_{n'} \to^*_{R \subset [\![P]\!]} \overline{S} = S_{n'+s-3} \\ \xrightarrow{K(N\theta)}_{[\![P]\!]} S_{n'+s-2} \xrightarrow{(ri)}_{[\![P]\!]} S_{n'+s-1} \xrightarrow{\text{Receive}(i,m)}_{[\![P]\!]} S_{n'+s} \in exec^{msr}([\![P]\!]) \end{aligned}$$

where

- since $S_{n'} \to_R S$, S is such that $set(S_{n'}) \setminus \{ \mathsf{Fr}(t), \mathsf{Out}(t) | t \in \mathcal{M} \} \subseteq set(S)$, $set(S) \setminus \{ !\mathsf{K}(t) | t \in \mathcal{M} \} \subseteq set(S_{n'})$, and $!\mathsf{K}(N\theta) \in S$,
- $S_{n'+s-2} = S \cup^{\#} \{ !\mathsf{K}(N\theta) \}^{\#}$,
- $S_{n'+s-1} = S \cup^{\#} \{ \mathsf{In}(N\theta) \}^{\#}$ and
- $S_{n'+s} = S \setminus^{\#} \{ \mathsf{state}_p(\tilde{t}) \} \cup^{\#} \{ \mathsf{state}_{p \cdot 1}(\tilde{t} \cup (vars(N)\theta)) \}$.

Letting $k = n' + s - 1$ we immediately have that Condition 7 holds.

We now show that Condition 3 holds. Since by induction hypothesis, $\text{in}(t, N); Q \leftrightarrow_P \mathsf{state}_p(\tilde{t})$, we have that $P|_p \tau = \text{in}(t, N); Q\rho$ for some τ and ρ. Therefore we also have that $P|_{p \cdot 1} \tau = Q\rho$. Thus $(P|_{p \cdot 1} \tau)(\theta\rho) = (Q\rho)(\theta\rho) = Q\theta\rho$. Now it is easy to see from the definition of $[\![P]\!]_{=p}$ that $\{Q\theta\} \leftrightarrow_P \{\mathsf{state}_{p \cdot 1}(\tilde{t}, (vars(N)\theta))\}$. Since $\mathcal{P}_n = \mathcal{P}_{n-1} \setminus^{\#} \{\text{in}(t, N); Q\} \cup^{\#} \{Q\theta\}$, we have that $\mathcal{P}_n \leftrightarrow_P S_{f(n)}$, i. e., Condition 3 holds.

Condition 8 holds, as i is fresh and thus no $\text{Send}(i, m)$ has previously appeared. Condition 6, Condition 1, Condition 2, Condition 4, Condition 5 and Condition 6 hold trivially.

Case: $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1} = \mathcal{P}' \cup \{\text{in}('r', N); Q\}, \sigma_{n-1}, \mathcal{L}_{n-1}, \mathcal{U}_{n-1}) \to (\mathcal{E}_{n-1}, \mathcal{S}_{n-1},$
$\mathcal{P}' \cup^{\#} \{ Q\theta \}, \sigma_{n-1}, \mathcal{L}_{n-1}), \mathcal{U}_{n-1} \setminus^{\#} \{ N' \}^{\#}$. This step requires that θ grounding for N and $N\theta \in^{\#}_E \mathcal{U}_{n-1}$. Using [19, Lemma 8], we have that there is an execution $\emptyset \xrightarrow{F_1} S_1 \xrightarrow{F_2} \ldots \xrightarrow{F_{f(n-1)}} S_{f(n-1)} \to^* S \in exec_E^{msr}([\![P]\!])$ such that $!\mathsf{K}(N\theta) \in_E S$ and $S_{f(n-1)} \to_R^* S$ for $R = \{ \text{MDOut}, \text{MDPub}, \text{MDFresh}, \text{MDAppl} \}$. Let $\emptyset \xrightarrow{F_1} S_1 \xrightarrow{F_2} \ldots \xrightarrow{F_{f(n-1)}} S_{f(n-1)} \to_R^* \overline{S} \in exec_E^{msr}([\![P]\!])$ with $!\mathsf{K}(N\theta) \in_E \overline{S}$. Let p and $\tilde{t}$ be such that, $\text{in}(t, N); Q \leftrightarrow_P \mathsf{state}_p(\tilde{t})$. By Definition 24 there is a $ri \in ginsts([\![P]\!]_{=p})$ such that $\mathsf{state}_p(\tilde{t})$ is part of its premise. As $N\theta \in^{\#}_E \mathcal{U}_{n-1}$, by Condition 8, there is a previous rule instantiation with action $\text{Send}(i, N\theta)$, for some i. By definition of the translation, there must hence be an instantiation of MID with $\mathsf{MID}_{\mathsf{rcv}}(i)$ in the conclusion, as otherwise the premise $\mathsf{MID}_{\mathsf{snd}}(i)$ would remain open. By Condition 8, there is no action $\text{Receive}(i, N\theta)$, but there might be $\text{Receive}(i, m')$ for $m' \neq N\theta$. In that case, the trace can easily be rewritten so that i is substituted by a fresh i' and an instantiation of MID for i' is added prior to the $\text{Receive}(i', m')$ – as the translation does not use i/i' anywhere else, this is possible w.l.o.g. Therefore we can safely assume that $\mathsf{MID}_{\mathsf{rcv}}(i) \in S$.

By definition of $[\![P]\!]_{=p}$ and the fact that θ is grounding for $N\theta$, we chose the following transition:

$$ri = [\mathsf{state}_p(\tilde{t}), \mathsf{In}(N\theta), \mathsf{MID}_{\mathsf{rcv}}(i)] \dashv[\ \mathrm{Receive}(i,m)\ ]\rightarrow [\mathsf{state}_{p{\cdot}1}(\tilde{t} \cup (vars(N)\theta)],$$

From $S_{n'}$, we can first apply the above transition $S_{n'} \rightarrow_R^* \overline{S}$, and then, (since $!\mathsf{K}(t), !\mathsf{K}(N\theta)$, $\mathsf{state}_p(\tilde{x}), \mathsf{MID}_{\mathsf{rcv}}(i) \in \overline{S}$), MDIn, ri and ri':

$$\emptyset \xrightarrow{F_1}_{[\![P]\!]} S_1 \ldots \xrightarrow{F_{n'}}_{[\![P]\!]} S_{n'} \rightarrow^*_{R \subset [\![P]\!]} \overline{S} = S_{n'+s-2}$$
$$\xrightarrow{K(N\theta)}_{[\![P]\!]} S_{n'+s-1} \xrightarrow{\mathrm{Receive}(i,m)}_{[\![P]\!]} S_{n'+s} \in exec^{msr}([\![P]\!])$$

where

- since $S_{n'} \rightarrow_R S$, S is such that $set(S_{n'}) \setminus \{\, \mathsf{Fr}(t), \mathsf{Out}(t) | t \in \mathcal{M} \,\} \subseteq set(S)$, $set(S) \setminus \{\, !\mathsf{K}(t) | t \in \mathcal{M} \,\} \subseteq set(S_{n'})$, and $!\mathsf{K}(N\theta) \in S$
- $S_{n'+s-1} = S \cup^{\#} \{\, \mathsf{In}(N\theta) \,\}^{\#}$,
- $S_{n'+s} = S \setminus^{\#} \{\, \mathsf{state}_p(\tilde{t}), \mathsf{MID}_{\mathsf{rcv}}(i) \,\} \cup^{\#} \{\, \mathsf{state}_{p{\cdot}1}(\tilde{t} \cup (vars(N)\theta)) \,\}$.

Letting $k = n' + s - 1$ we immediately have that Condition 7 holds.

We now show that Condition 3 holds. Since by induction hypothesis, $\mathrm{in}(t, N); Q \leftrightarrow_P \mathsf{state}_p(\tilde{t})$, we have that $P|_p\tau = \mathrm{in}(t, N); Q\rho$ for some τ and ρ. Therefore we also have that $P|_{p{\cdot}1}\tau = Q\rho$. Thus $(P|_{p{\cdot}1}\tau)(\theta\rho) = (Q\rho)(\theta\rho) = Q\theta\rho$. Now it is easy to see from the definition of $[\![P]\!]_{=p}$ that $\{Q\theta\} \leftrightarrow_P \{\mathsf{state}_{p{\cdot}1}(\tilde{t}, (vars(N)\theta))\}$. Since $\mathcal{P}_n = \mathcal{P}_{n-1} \setminus^{\#} \{\mathrm{in}(t, N); Q\} \cup^{\#} \{Q\theta\}$, we have that $\mathcal{P}_n \leftrightarrow_P S_{f(n)}$, i. e., Condition 3 holds.

Condition 8 holds, since $N\theta$ is removed from $\mathcal{U}_{n-1}$ as well as from the multiset

$$\{\, m \mid \exists j \leq f(i), mid.\ \mathrm{Send}(mid, m) \in_E F_j \wedge \forall j'.\mathrm{Receive}(mid, m) \notin_E F_{j'} \,\}^{\#},$$

since $\mathrm{Receive}(i, m)$ is added to the trace, and, as previously established, $\mathrm{Send}(i, m)$ appeared in the msr trace before step n'.

Condition 6, Condition 1, Condition 2, Condition 4, Condition 5 and Condition 6 hold trivially.

The remaining cases are treated exactly as in [19, Lemma 10], with α' in lieu of α. □

C.2 Proof that $traces^{pi}(P) \supseteq hide(filter(traces^{msr}([\![P]\!])))$

In the first step, we define the notion of a *normal* msr execution [19, Definition 21], and show that any msr execution resulting from a translation of a process has an equivalent normal execution [19, Lemma 11]. We use a slightly different normalisation in this case.

Definition 25 (normal msr execution)**.** *A msr execution $\emptyset \xrightarrow{E_1}_{[\![P]\!]} \cdots \xrightarrow{E_n}_{[\![P]\!]} S_n \in exec^{msr}([\![P]\!])$ for the multiset rewrite system $[\![P]\!]$ defined by a ground process P is* normal *if:*

1. *The first transition is an instance of the* Init *rule, i. e., $S_1 = \mathsf{state}_{[]}()$ and there is at least this transition.*
2. *$S_{n-1} \xrightarrow{E_n}_{[\![P,[],[]]\!],\mathrm{MDIn},\mathrm{Init}} S_n$*
3. *if $\mathsf{In}(t) \in (S_{i-1} \setminus^{\#} S_i)$ for some i and $t \in \mathcal{M}$, then $S_{i-2} \xrightarrow{K(t)}_{\mathrm{MDIn}} S_{i-1}$*

Intuitively, a normal execution always starts with an Init rule (Item 1), and the last action is neither the generation of a fresh name, nor a message deduction rule (Item 2. Indeed such a transition is not useful if it is the last one, as the freshly generated name, or the deduced message would not be used. Finally, if the attacker inputs a term to a process, this term is deduced at the last possible moment (Item 3).

Any execution has an equivalent normal execution, i. e., an execution that has the same labels, up to reserved facts, and preserves α'.

Lemma 15 (Normalisation)**.** *Let P be a well-formed ground process. If*

$$S_0 = \emptyset \xrightarrow{E_1}_{[\![P]\!]} S_1 \xrightarrow{E_2}_{[\![P]\!]} \dots \xrightarrow{E_n}_{[\![P]\!]} S_n \in exec^{msr}([\![P]\!])$$

and $[E_1, \dots, E_n] \vDash \alpha'$, then there exists a normal msr execution

$$T_0 = \emptyset \xrightarrow{F_1}_{[\![P]\!]} T_1 \xrightarrow{F_2}_{[\![P]\!]} \dots \xrightarrow{F_{n'}}_{[\![P]\!]} T_{n'} \in exec^{msr}([\![P]\!])$$

such that $hide([E_1, \dots, E_n]) = hide(F_1, \dots, F_{n'})$ and $[F_1, \dots, F_{n'}] \vDash \alpha'$.

Proof. We will modify $S_0 \xrightarrow{E_1}_{[\![P]\!]} \dots \xrightarrow{E_n}_{[\![P]\!]} S_n$ by applying one transformation after the other, each resulting in an msr execution that preserves satisfaction of α'.

1. If an application of the Init rule appears in $S_0 \xrightarrow{E_1}_{[\![P]\!]} \dots \xrightarrow{E_n}_{[\![P]\!]} S_n$, we move it to the front. Therefore, $S_1 = \mathsf{state}_{[]}()$. This is possible since the left-hand side of the Init rule is empty. If the rule is never instantiated, we prepend it to the trace. Since Init() $\in \mathcal{F}_{res}$, the resulting msr execution
$$S_0^{(1)} \xrightarrow{E_1^{(1)}}_{[\![P]\!]} \dots \xrightarrow{E_n^{(1)}}_{[\![P]\!]} S_{n^{(1)}}^{(1)}$$
is such that $hide([E_1, \dots, E_n]) = hide([E_1^{(1)}, \dots, E_{n^{(1)}}^{(1)}])$. Since Init() is only added if it was not present before, $[E_1^{(1)}, \dots, E_{n^{(1)}}^{(1)}] \vDash \alpha$, especially α_{init}.

2. If the last transition is in $\{\,\text{MDOut}, \text{MDPub}, \text{MDFresh}, \text{MDAppl}, \text{Fresh}\,\}$, we remove it. Repeat until fixpoint is reached and call the resulting trace
$$S_0^{(4)} \xrightarrow{E_1^{(4)}}_{[\![P]\!]} \dots \xrightarrow{E_n^{(4)}}_{[\![P]\!]} S_{n^{(4)}}^{(4)}$$
Since no rule removed during the procedure has an action, $hide([E_1, \dots, E_n]) = hide([E_1^{(4)}, \dots, E_{n^{(4)}}^{(4)}])$ and $[E_1^{(4)}, \dots, E_{n^{(4)}}^{(4)}] \vDash \alpha$.

3. If there is $\mathsf{In}(t) \in S_{n^{(4)}-1}^{(4)}$, then there is a transition where $\mathsf{In}(t)$ is produced and never consumned until $n^{(4)} - 1$. The only rule producing $\mathsf{In}(t)$ is MDIn. We can move this transition to just before $n^{(4)} - 1$ and call the resulting trace
$$S_0^{(5)} \xrightarrow{E_1^{(5)}}_{[\![P]\!]} \dots \xrightarrow{E_n^{(5)}}_{[\![P]\!]} S_{n^{(5)}}^{(5)}$$
Since $[E_1^{(4)}, \dots, E_{n^{(4)}}^{(4)}] \vDash \alpha$, especially α_{inev}, there is no action that is not in $\mathcal{F}_{res}$ between the abovementioned instance of MDIn, therefore, $hide([E_1, \dots, E_n]) = hide([E_1^{(5)}, \dots, E_{n^{(5)}}^{(5)}])$ holds. Since α_{inev} is the only part of α' that mentions K, and since the tranformation preserved α_{inev}, we have that $[E_1^{(5)}, \dots, E_{n^{(5)}}^{(5)}] \vDash \alpha$.

□

Proposition 1. *If P is a ground process and $\emptyset \xrightarrow{E_1}_{[\![P]\!]} \cdots \xrightarrow{E_n}_{[\![P]\!]} S_n \in exec^{msr}([\![P]\!])$ is a* normal *msr execution with $n \geq 2$, then there exists $m < n$ such that*

$$S_m \rightarrow^*_{\{\,\text{MDOut},\text{MDPub},\text{MDFresh},\text{MDAppl},\text{Fresh}\,\}} S_{n-1}$$

and $\emptyset \xrightarrow{E_1}_{[\![P]\!]} \cdots \xrightarrow{E_m}_{[\![P]\!]} S_m \in exec^{msr}([\![P]\!])$ is normal.

Proof. We chose the largest $m < n$ such that $S_{m-1} \xrightarrow{E_m}_{[\![P,[],[]]\!],\text{Init},\text{MDIn}} S_m$, Such an m exists since $S_0 \xrightarrow{\text{Init}()}_{[\![P]\!]} S_1$. Then, $S_m \rightarrow^*_{\{\,\text{MDOut},\text{MDPub},\text{MDFresh},\text{MDAppl},\text{Fresh}\,\}} S_{n-1}$ since otherwise there would be a larger m.

We will now show that the prefixes of the execution until m are normal.

- Item 1 of Definition 25 is preserved as in this case $m \geq 1$.
- Item 2 of Definition 25 holds trivially since $S_m \rightarrow^*_{\{\,\text{MDOut,MDPub,MDFresh,MDAppl,Fresh}\,\}} S_{n-1}$.
- Item 3 of Definition 25 hold for all parts of the trace, and therefore also for the prefix of size m.

□

Definition 26. *Let P be a ground process, $\mathcal{P}$ be a multiset of processes and S a multiset of multiset rewrite rules. We write $\mathcal{P} \leftrightsquigarrow_P S$ if there exists a bijection between $\mathcal{P}$ and the multiset*

$$\{\,\mathsf{state}_p(\tilde{t}) \mid \exists p, \tilde{t}.\ \mathsf{state}_p(\tilde{t}) \in^\# S\,\}^\# \cup^\# \{\,\mathsf{state}_p^{\mathsf{semi}}(\tilde{t}) \mid \exists p, \tilde{t}.\ \mathsf{state}_p^{\mathsf{semi}}(\tilde{t}) \in^\# S\,\}^\#$$

such that whenever $Q \in^\# \mathcal{P}$ is mapped to $\mathsf{state}_p(\tilde{t}) \in^\# S$ or $\mathsf{state}_p^{\mathsf{semi}}(\tilde{t}) \in^\# S$, then:

1. *$\mathsf{state}_p(\tilde{t}) \in_E prems(ri)$ or $\mathsf{state}_p(\tilde{t}) \in_E prems(ri)$ for $ri \in ginsts(\llbracket P \rrbracket_{=p})$.*
2. *Let θ be a grounding substitution for $state_p^*(\tilde{x}) \in prems(\llbracket P \rrbracket_{=p})$ such that $\tilde{t} = \tilde{x}\theta$. Then*

$$(P|_p\tau)\rho =_E Q$$

for a substitution τ, and a bijective renaming ρ of fresh, but not bound names in Q, defined as follows:

$$\begin{aligned} \tau(x) &:= \theta(x) && \text{if } x \text{ not a reserved variable} \\ \rho(a) &:= a' && \text{if } \theta(n_a) = a' \end{aligned}$$

When $\mathcal{P} \leftrightsquigarrow_P S$, $Q \in^\# \mathcal{P}$ and $\mathsf{state}_p(\tilde{t}) \in^\# S$ we also write $Q \leftrightsquigarrow_P \mathsf{state}_p(\tilde{t})$ if this bijection maps Q to $\mathsf{state}_p(\tilde{t})$ (and similar for $\mathsf{state}_p^{\mathsf{semi}}(\tilde{t})$).

Lemma 16. *Let P be a well-formed ground process. If*

$$S_0 = \emptyset \xrightarrow{E_1}_{\llbracket P \rrbracket} S_1 \xrightarrow{E_2}_{\llbracket P \rrbracket} \ldots \xrightarrow{E_n}_{\llbracket P \rrbracket} S_n \in exec^{msr}(\llbracket P \rrbracket)$$

is normal (see Definition 25) and $[E_1, \ldots, E_n] \vDash \alpha'$ (see Definition 16), then there are $(\mathcal{E}_0, \mathcal{S}_0, \mathcal{P}_0, \sigma_0, \mathcal{L}_0, \mathcal{U}_0), \ldots, (\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{P}_{n'}, \sigma_{n'}, \mathcal{L}_{n'}, \mathcal{U}_{n'})$ and $F_1, \ldots, F_{n'}$ such that:

$$(\mathcal{E}_0, \mathcal{S}_0, \mathcal{P}_0, \sigma_0, \mathcal{L}_0, \mathcal{U}_0) \xrightarrow{F_1} (\mathcal{E}_1, \mathcal{S}_1, \mathcal{P}_1, \sigma_1, \mathcal{L}_1, \mathcal{U}_1) \xrightarrow{F_2} \ldots \xrightarrow{F_{n'}} (\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{P}_{n'}, \sigma_{n'}, \mathcal{L}_{n'}, \mathcal{U}_{n'})$$

where $(\mathcal{E}_0, \mathcal{S}_0, \mathcal{P}_0, \sigma_0, \mathcal{L}_0, \mathcal{U}_0) = (\emptyset, \emptyset, \emptyset, \{\,P\,\}, \emptyset, \emptyset)$ and there exists a monotonically increasing, surjective function $f\colon \mathbb{N}_n \setminus \{\,0\,\} \to \mathbb{N}_{n'}$ such that $f(n) = n'$ and for all $i \in \mathbb{N}_n$

1. $\mathcal{E}_{f(i)} = \{\, a \in FN \mid ProtoNonce(a) \in_E \bigcup_{1 \leq j \leq i} E_j \,\}$
2. $\forall\, t \in \mathcal{M}.\ \mathcal{S}_{f(i)}(t) = \begin{cases} u & \text{if } \exists j \leq i. \mathrm{Insert}(t,u) \in_E E_j \\ & \quad \wedge \forall j', u'. j < j' \leq i \rightarrow \mathrm{Insert}(t,u') \notin_E E_{j'} \wedge \mathrm{Delete}(t) \notin_E E_{j'} \\ \bot & \text{otherwise} \end{cases}$
3. $\mathcal{P}_{f(i)} \leftrightsquigarrow_P S_i$
4. $\{\, x\sigma_{f(i)} \mid x \in \mathbf{D}(\sigma_{f(i)}) \,\}^\# = \{\, t \mid \exists k \in \mathbb{N}_{i-1}. \mathsf{Out}(t) \in S_{k+1} \setminus S_k \,\}^\#$
5. $\mathcal{L}_{f(i)} =_E \{\, t \mid \exists j \leq i, u.\ \mathrm{Lock}(u,t) \in_E E_j \wedge \forall j < k \leq i. \mathrm{Unlock}(u,t) \notin_E E_k \,\}$.

Furthermore,

6. *$hide([E_1, \ldots, E_n]) =_E [F_1, \ldots, F_{n'}]$ and*
7. $\mathcal{U}_{f(i)} =_E \{\, m \mid \exists j \leq i, mid.\ \mathrm{Send}(mid, m) \in F_j \wedge \forall j'. \mathrm{Receive}(mid, m) \notin_E F_{j'} \,\}^\#$

The Lemma indeed implies that $\{\, tr \in \mathit{hide}(\mathit{filter}(\mathit{traces}^{msr}([\![P]\!]))) \mid tr \text{ is normal} \,\} \subseteq \mathit{traces}^{pi}(P)$: for any normal trace $[E_1, \ldots, E_n]$ that satisfies α', i.e., in $\mathit{filter}(\mathit{traces}^{msr}([\![P]\!]))$ we show there exists a trace $[F_1, \ldots, F_{n'}] \mathit{intraces}^{pi}(P)$ such that $\mathit{hide}([E_1, \ldots, E_n]) =_E [F_1, \ldots, F_{n'}]$ (Condition 6).

Proof. We proceed by induction over the number of transitions n.

Base Case. A normal msr execution contains at least an application of the init rule, thereby the shortest normal msr execution is

$$\emptyset \rightarrow_{[\![P]\!]} S_1 = \{\, \mathsf{state}_{[]}() \,\}^{\#}$$

We chose $n' = 0$ and thus

$$(\mathcal{E}_0, \mathcal{S}_0, \mathcal{P}_0, \sigma_0, \mathcal{L}_0, \mathcal{U}_0) = (\emptyset, \emptyset, \emptyset, \{\, P \,\}^{\#}, \emptyset, \emptyset).$$

We define $f : \{\, 1 \,\} \to \{\, 0 \,\}$ such that $f(1) = 0$.

To show that Condition 3 holds, we have to show that $\mathcal{P}_0 \leftrightsquigarrow_P \{\, \mathsf{state}_{[]}() \,\}^{\#}$. Note that $\mathcal{P}_0 = \{\, P \,\}^{\#}$. We choose the bijection such that $P \leftrightsquigarrow_P \mathsf{state}_{[]}()$.

By definition of the translation at position $[]$ $[\![P]\!]_{=[]} = [\![P, [], []]\!]_{=[]}$. We see from Figure 6 that for every P we have that $\mathsf{state}_{[]}() \in \mathit{prems}(R\theta)$, for $R \in [\![P, [], []]\!]_{=[]}$ and $\theta = \emptyset$. This induces $\tau = \emptyset$ and $\rho = \emptyset$. Since $P|_{[]}\tau\rho = P$, we have $P \leftrightsquigarrow_P \mathsf{state}_{[]}()$, and therefore $\mathcal{P}_0 \leftrightsquigarrow_P S_1$.

Condition 1, Condition 2, Condition 4, Condition 5, and Condition 6 hold trivially.

Inductive step. Assume the invariant holds for $n - 1 \geq 1$. We have to show that the lemma holds for n transitions, i. e., we assume that

$$\emptyset \xrightarrow{E_1}_{[\![P]\!]} S_1 \xrightarrow{E_2}_{[\![P]\!]} \ldots \xrightarrow{E_n}_{[\![P]\!]} S_n \in \mathit{exec}^{msr}([\![P]\!])$$

is normal and $[E_1, \ldots, E_n] \vDash \alpha'$. Then it is to show that there is

$$(\mathcal{E}_0, \mathcal{S}_0, \mathcal{P}_0, \sigma_0, \mathcal{L}_0, \mathcal{U}_0) \xrightarrow{F_1} (\mathcal{E}_1, \mathcal{S}_1, \mathcal{P}_1, \sigma_1, \mathcal{L}_1, \mathcal{U}_1) \xrightarrow{F_2} \ldots \xrightarrow{F_{n'+1}} (\mathcal{E}_{n'+1}, \mathcal{S}_{n'+1}, \mathcal{P}_{n'+1}, \sigma_{n'+1}, \mathcal{L}_{n'+1}, \mathcal{U}_{n'+1})$$

fulfilling Conditions 1 to 5.

Since $\emptyset \xrightarrow{E_1}_{[\![P]\!]} \cdots \xrightarrow{E_n}_{[\![P]\!]} S_n \in \mathit{exec}^{msr}([\![P]\!])$ is normal and $n \geq 2$, by Proposition 1, there exists $m < n$ such that $S_m \rightarrow_R^* S_n$ for $R = \{\, \text{MDOut}, \text{MDPub}, \text{MDFresh}, \text{MDAppl}, \text{Fresh} \,\}$ and $\emptyset \xrightarrow{E_1}_{[\![P]\!]} \cdots \xrightarrow{E_m}_{[\![P]\!]} S_m \in \mathit{exec}^{msr}([\![P]\!])$ is normal, too. This allows us to apply the induction hypothesis on $\emptyset \xrightarrow{E_1}_{[\![P]\!]} \cdots \xrightarrow{E_m}_{[\![P]\!]} S_m \in \mathit{exec}^{msr}([\![P]\!])$. Hence there is a monotonically increasing function from $\mathbb{N}_m \to \mathbb{N}_{n'}$ and an execution such that Conditions 1 to 5 hold. Let f_p be this function and note that $n' = f_p(m)$.

In the following case distinction, we will (unless stated otherwise) extend the previous execution by one step from $(\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{P}_{n'}, \sigma_{n'}, \mathcal{L}_{n'}, \mathcal{U}_{n'})$ to $(\mathcal{E}_{n'+1}, \mathcal{S}_{n'+1}, \mathcal{P}_{n'+1}, \sigma_{n'+1}, \mathcal{L}_{n'+1}, \mathcal{U}_{n'+1})$, and prove that Conditions 1 to 6 hold for $n' + 1$. By induction hypothesis, they hold for all $i \leq n'$. We define a function $f \colon \mathbb{N}_n \to \mathbb{N}_{n'+1}$ as follows:

$$f(i) := \begin{cases} f_p(i) & \text{if } i \in \mathbb{N}_m \\ n' & \text{if } m < i < n \\ n' + 1 & \text{if } i = n \end{cases}$$

Since, $S_m \rightarrow_R^* S_n$ for $R = \{\, \text{MDOut}, \text{MDPub}, \text{MDFresh}, \text{MDAppl}, \text{Fresh} \,\}$, only $S_n \setminus^{\#} S_m$ contains only Fr-facts and !K-facts, and $S_m \setminus^{\#} S_n$ contains only Fr-facts and Out-facts. Therefore, 3 and 4 hold for all $i \leq n - 1$. Since $E_{m+1}, \ldots, E_{n-1} = \emptyset$, Condition 1, 2,5 and 6 hold for all $i \leq n - 1$.

Fix a bijection such that $\mathcal{P}_{n'} \leftrightsquigarrow_P S_m$. We will abuse notation by writing $P \leftrightsquigarrow_P \mathsf{state}_p(\tilde{t})$, if this bijection maps P to $\mathsf{state}_p(\tilde{t})$. As $S_n \setminus^{\#} S_m$ contains only Fr-facts and !K-facts, and $S_m \setminus^{\#} S_n$ contains only Fr-facts and Out-facts, we also have

$$\mathcal{P}_{n'} \leftrightsquigarrow_P S_{n-1} \tag{9}$$

We now proceed by case distinction over the last type of transition from S_{n-1} to S_n. Let $l_{linear} =_E S_{n-1} \setminus S_n$ and $r =_E S_n \setminus S_{n-1}$. l_{linear} can only contain linear facts, while r can contain linear as well as persistent facts. The rule instance ri used to go from S_{n-1} to S_n has the following form:

$$[l_{linear}, l_{persistent}] \,-[\, E_n \,]\to r$$

for some $l_{persistent} \subset^{\#}_E S_{n-1}$.

Note that l_{linear}, E_n and r uniquely identify which rule in $R \in [\![P, [], []]\!]$ ri is an instance of.

If R is uniquely determined, we fix some $ri \in ginsts(R)$.

In the following, we will only treat the cases that are different from [19, Lemma 12]. (Note that the notion of normal msr executions there is equivalent to the notion used here, if the fact symbol Ack is not used (which can easily be achieved by renaming) with one exception: the second item in [19, Definition 21] is omitted, as we treat replication differently here. Hence, these cases need to be proven, too. Each case omitted applies to the notion of normal msr executions in Definition 25.)

Case: $ri = [\mathsf{state}_p(\tilde{t})] \,-[]\to\, [!\mathsf{state}^{\mathsf{semi}}_{p\cdot 1}(\tilde{t})]$ (for some $p, \tilde{t}$). By the form of the rule, we see that $P|_p$ is a replication. We set $f(n+1)$ to $f(n)$ and do not extend the execution. By Definition 26, the bijection in Condition 3 remains unchanged. The other conditions hold trivially.

Case: $ri = [!\mathsf{state}^{\mathsf{semi}}_{p}(\tilde{t})] \,-[]\to\, [\mathsf{state}_{p\cdot 1}(\tilde{t})]$ (for some $p, \tilde{t}$). By induction hypothesis, we have $\mathcal{P}_{n'} \leftrightsquigarrow_P S_m$, and thus, as previously established, $\mathcal{P}_{n'} \leftrightsquigarrow_P S_{n-1}$ (see Equation (9)). Let $Q \in^{\#} \mathcal{P}_{n'}$ such that $Q \leftrightsquigarrow_P \mathsf{state}_p(\tilde{t})$. Let θ be a grounding substitution for $\mathsf{state}_p(\tilde{x}) \in prems([\![P]\!]_{=p})$ such that $\tilde{t} = \tilde{x}\theta$. Then θ induces a substitution τ and a bijective renaming ρ for fresh, but not bound names (in Q) such that $P|_p\tau\rho = Q$ (see Definition 26).

From the form of the rule instance ri, and since $Q = P|_p\tau\rho$, we can deduce that $Q = !Q'$for a process $Q' = P|_{p\cdot 1}\tau\rho$..

We therefore chose the following transition:

$$\cdots \xrightarrow{F'_n} (\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{P}_{n'}, \sigma_{n'}, \mathcal{L}_{n'}, \mathcal{U}_{n'}) \to (\mathcal{E}_{n'+1}, \mathcal{S}_{n'+1}, \mathcal{P}_{n'+1}, \sigma_{n'+1}, \mathcal{L}_{n'+1}, \mathcal{U}_{n'+1})$$

with $\mathcal{E}_{n'+1} = \mathcal{E}_{n'}$, $\mathcal{S}_{n'+1} = \mathcal{S}_{n'}$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \cup^{\#} \{\, Q' \,\}^{\#}$, $\sigma_{n'+1} = \sigma_{n'}$ and $\mathcal{L}_{n'+1} = \mathcal{L}_{n'}$.

We define f as on page 45. Therefore, Conditions 1 to 7 hold for $i < n-1$. It is left to show that Conditions 1 to 7 hold for n.

By definition of $[\![P]\!]$ and $[\![P]\!]_{=p}$, we have that $Q' \leftrightsquigarrow_P \mathsf{state}_{p\cdot 1}(\tilde{t})$. Therefore, and since $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \cup^{\#} \{\, Q' \,\}^{\#}$, while $S_n = S_{n-1} \cup^{\#} \{\, \mathsf{state}_{p\cdot 1}(\tilde{t}) \,\}^{\#}$, Condition 3 holds.

Conditions 1, 2, 4, 5, 6 and 7 hold trivially.

Case: $ri = [\mathsf{state}_p(\tilde{t}), \mathsf{MID}_{\mathsf{snd}}(mid)] \,-[\, \mathrm{Send}(mid, m) \,]\to\, [\mathsf{state}_{p\cdot 1}(\tilde{t}, mid), \mathsf{Out}(m)]$ (for some position p and $\tilde{t}, m, mid \in \mathcal{M}$). By induction hypothesis, we have $\mathcal{P}_{n'} \leftrightsquigarrow_P S_m$, and thus, as previously established, $\mathcal{P}_{n'} \leftrightsquigarrow_P S_{n-1}$ (see Equation (9)). Let $Q \in^{\#} \mathcal{P}_{n'}$ such that $Q \leftrightsquigarrow_P \mathsf{state}_p(\tilde{t})$. Let θ be a grounding substitution for $\mathsf{state}_p(\tilde{x}) \in prems([\![P]\!]_{=p})$ such that $\tilde{t} = \tilde{x}\theta$. Then θ induces a substitution τ and a bijective renaming ρ for fresh, but not bound names (in Q) such that $P|_p\tau\rho = Q$ (see Definition 26).

From the form of the rule instance ri, and since $Q = P|_p\tau\rho$, we can deduce that $Q = \mathrm{out}\ (\text{`}\mathsf{r}\text{'}, m); Q'$ for a process $Q' = P|_{p\cdot 1}\tau\rho$.

We therefore chose the following transition:

$$\cdots \xrightarrow{F'_n} (\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{P}_{n'}, \sigma_{n'}, \mathcal{L}_{n'}, \mathcal{U}_{n'}) \to (\mathcal{E}_{n'+1}, \mathcal{S}_{n'+1}, \mathcal{P}_{n'+1}, \sigma_{n'+1}, \mathcal{L}_{n'+1}, \mathcal{U}_{n'+1})$$

with $\mathcal{E}_{n'+1} = \mathcal{E}_{n'}$, $\mathcal{S}_{n'+1} = \mathcal{S}_{n'}$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^{\#} \{\, \mathrm{out}\ (\text{`}\mathsf{r}\text{'}, m); Q' \,\}^{\#} \cup^{\#} \{\, Q' \,\}^{\#}$, $\sigma_{n'+1} = \sigma_{n'} \cup \{{}^{m}/_{x}\}$ for some fresh x, $\mathcal{L}_{n'+1} = \mathcal{L}_{n'}$, and $\mathcal{U}_{n'+1} = \mathcal{U}_{n'} \cup^{\#} \{\, m \,\}^{\#}$.

We define f as on page 45. Therefore, Conditions 1 to 7 hold for $i < n-1$. It is left to show that Conditions 1 to 7 hold for n.

By definition of $[\![P]\!]$ and $[\![P]\!]_{=p}$, we have that $Q' \leftrightsquigarrow_P \mathsf{state}_{p\cdot 1}(\tilde{t})$. Therefore, and since $\mathrm{out}\ (\text{`}\mathsf{r}\text{'}, m); Q' \leftrightarrow \mathsf{state}_p(\tilde{t})$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^{\#} \{\, \mathrm{out}\ (\text{`}\mathsf{r}\text{'}, m); Q' \,\}^{\#} \cup^{\#} \{\, Q' \,\}^{\#}$, and $S_n = S_{n-1} \setminus^{\#} \{\, \mathsf{state}_p(\tilde{t}) \,\}^{\#} \cup^{\#} \{\, \mathsf{state}_{p\cdot 1}(\tilde{t}) \,\}^{\#}$, Condition 3 holds.

From the induction hypothesis, and the fact that ri adds $\mathsf{Out}(m)$ to the state, we have that:

$$\begin{aligned}\{\, x\sigma_{n'+1} \mid x \in \mathbf{D}(\sigma_{n'+1}) \,\}^{\#} &= \{\, x\sigma_{n'} \mid x \in \mathbf{D}(\sigma_{n'}) \,\}^{\#} \cup^{\#} \{\, m \,\}^{\#} \\ &=_E \{\, t \mid \exists k \in \mathbb{N}_{n-2}.\mathsf{Out}(t) \in S_{k+1} \setminus S_k \,\}^{\#} \cup^{\#} \{\, m \,\}^{\#} \\ &= \{\, t \mid \exists k \in \mathbb{N}_{n-1}.\mathsf{Out}(t) \in S_{k+1} \setminus S_k \,\}^{\#}\end{aligned}$$

Therefore, Condition 4 holds.

Condition 6 holds since $hide([E_1, \ldots, E_m]) =_E [F_1, \ldots, n']$, and $[E_{m+1}, \ldots, E_{n-1}] =_E [F_{n'+1}]$, since $E_{n-1} = K(t_1)$.

Conditions 1, 2, 5 and 7 hold trivially.

Case: $ri = [\mathsf{state}_p(\tilde{t}), \mathsf{In}(\langle \text{`c'}, t\rangle)] \; -[\; \mathrm{InEvent}(\langle \text{`c'}, t\rangle) \;]\to \; [\mathsf{state}_{p\cdot 1}(\tilde{t}')]$ (for some $p, \tilde{t}, \tilde{t}'$ and $t \in \mathcal{M}$). This case is an instance of the case for $\mathsf{state}_p(\tilde{t}), \mathsf{In}(\langle t_1, t_2\rangle)\mathrm{InEvent}(\langle t_1, t_2\rangle)\mathsf{state}_{p\cdot 1}(\tilde{t}, \tilde{t}')$ in [19, Lemma 12].

Case: $ri = [\mathsf{state}_p(\tilde{t}), \mathsf{In}(t), \mathsf{MID}_{\mathsf{rcv}}(mid)] \; -[\; \mathrm{Receive}(mid, t), \mathrm{InEvent}(t) \;]\to \; [\mathsf{state}_{p\cdot 1}(\tilde{t}')]$ (for some $p, \tilde{t}, \tilde{t}', t, mid \in \mathcal{M}$). As mentioned before, there is an m an $m < n$ such that $S_0 \xrightarrow{E_1}_{[\![P]\!]} \ldots \xrightarrow{E_m}_{[\![P]\!]} S_m$ is a normal trace and $S_m \to_R^* S_{n-1}$ for $R = \{\, \mathrm{MDOut}, \mathrm{MDPub}, \mathrm{MDFresh}, \mathrm{MDAppl}, \mathrm{Fresh} \,\}$.

Due to $S_0 \xrightarrow{E_1}_{[\![P]\!]} \ldots \xrightarrow{E_n}_{[\![P]\!]} S_n$ being normal, we have that $S_{n-2} \xrightarrow{E_n = K(t)} S_{n-1}$ is an instance of MDIn.

By induction hypothesis, we have $\mathcal{P}_{n'} \leftrightsquigarrow_P S_m$. Since $R = \{\, \mathrm{MDOut}, \mathrm{MDPub}, \mathrm{MDFresh}, \mathrm{MDAppl} \,\}$, Fresh and MDIn do not add or remove state-facts, $\mathcal{P}_{n'} \leftrightsquigarrow_P S_{n-1}$. Let $Q \in^{\#} \mathcal{P}_{n'}$ such that $Q \leftrightsquigarrow_P \mathsf{state}_p(\tilde{t})$. Let θ be a grounding substitution for $\mathsf{state}_p(\tilde{x}) \in prems([\![P]\!]_{=p})$ such that $\tilde{t} =_E \tilde{x}\theta$. Then θ induces a substitution τ and a bijective renaming ρ for fresh, but not bound names (in Q) such that $P|_p\tau\rho = Q$ (see Definition 26). From the form of the rule instance ri, and since $Q = P|_p\tau\rho$, we can deduce that $Q = \mathrm{in}(\text{`c'}, N); Q'$, for N a term that is not necessarily ground, and a process $Q' = P|_{p\cdot 1}\tau\rho$. Since $ri \in_E ginsts([\![P]\!]_{=p})$, we have that there is a substitution τ' such that $N\tau' =_E t$.

We chose the following transition depending on whether there is $j < m$ (and thus $j < n$) such that $\mathrm{Send}(mid, t) \in_E E_j$. In both cases, we chose,

$$\ldots \xrightarrow{F'_n} (\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{P}_{n'}, \sigma_{n'}, \mathcal{L}_{n'}, \mathcal{U}_{n'}) \xrightarrow{K(t)} (\mathcal{E}_{n'+1}, \mathcal{S}_{n'+1}, \mathcal{P}_{n'+1}, \sigma_{n'+1}, \mathcal{L}_{n'+1}, \mathcal{U}_{n'+1})$$

with $\mathcal{E}_{n'+1} = \mathcal{E}_{n'}$, $\mathcal{S}_{n'+1} = \mathcal{S}_{n'}$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^{\#} \{\, \mathrm{in}(\text{`c'}, N); Q' \,\}^{\#} \cup^{\#} \{\, Q'\tau' \,\}^{\#}$, $\sigma_{n'+1} = \sigma_{n'}$, $\mathcal{L}_{n'+1} = \mathcal{L}_{n'}$. In the case, where $j < n$ such that $\mathrm{Send}(mid, t) \in_E E_j$, we chose $\mathcal{U}_{n'+1} = \mathcal{U}_{n'} \setminus^{\#} \{\, t \,\}^{\#}$, otherwise $\mathcal{U}_{n'+1} = \mathcal{U}_{n'}$.

In both cases, we define f as follows:

$$f(i) := \begin{cases} f_p(i) & \text{if } i \in \mathbb{N}_m \\ n' & \text{if } m < i \leq n-1 \\ n'+1 & \text{if } i = n \end{cases}$$

For the first case, we need to show that $t \in_E \mathcal{U}_{n-1}$, which follows from the induction hypothesis and the assumption that there is $j < n$ such that $\mathrm{Send}(mid, t) \in_E E_j$. The contraposition would imply that $\mathrm{Receive}(mid, t) \in_E E_{j'}$ for some $j' < n$ which only occurs in rule instances which consume $\mathsf{MID}_{\mathsf{rcv}}(mid)$. As this fact may only be produced by some unique instance of MID, this would contradict $\mathsf{MID}_{\mathsf{rcv}}(mid) \in S_{n-1}$, which is necessary for ri to be applied.

For the second case, we need to show $\nu\mathcal{E}_{n'}.\sigma_{n'} \vdash t$. From the induction hypothesis, and since $E_{m+1}, \ldots, E_{n-2} = \emptyset$ and $E_{n-1} = K(\langle \text{`c'}, t\rangle)$ we have that

$$\mathcal{E}_{n'} = \{\, a \mid ProtoNonce(a) \in \bigcup_{1 \leq j \leq n-2} E_j \,\}.$$

From the induction hypothesis, and since no rule producing Out-facts is applied between step m and step $n-2$, we have that

$$\{\, x\sigma_{n'} \mid x \in \mathbf{D}(\sigma_{n'}) \,\}^{\#} = \{\, \mathsf{Out}(t) \in \cup_{k \leq n-2} S_k \,\}^{\#}. \tag{10}$$

Let $\tilde{r} = \{\, a : \mathit{fresh} \mid \mathit{RepNonce}(a) \in \bigcup_{1\leq j\leq n-2} F_j \,\}$. Since $!\mathsf{K}(t) \in \mathit{prems}(\text{MDIN}\sigma)$ for $\sigma(x) = t$, we have $!\mathsf{K}(t)_E \in S_{n-2}$. By [19, Lemma 8] and [19, Lemma 9], we have $\nu\mathcal{E}_{n'}, \tilde{r}.\sigma_{n'} \vdash t$. Therefore, $\nu\mathcal{E}_{n'}.\sigma_{n'} \vdash t$. Using DEQ and DAPPL with the function symbols *fst* and *snd*, we have $\nu\mathcal{E}_{n'}.\sigma_{n'} \vdash t$.

Therefore, Conditions 1 to 7 hold for $i \leq n-1$. It is left to show that Conditions 1 to 7 hold for n.

In both cases, Condition 6 holds since $\mathit{hide}([E_1, \ldots, E_m]) = [F_1, \ldots, n']$, and $[E_{m+1}, \ldots, E_{n-1}] = [F_{n'+1}]$, since $E_{n-1} = K(t)$.

Let θ' such that $ri = \theta' r$. As established before, we have τ' such that $N\tau' =_E t_2$. By definition of $[\![P]\!]_{=p}$, we have that $\mathsf{state}_{p\cdot 1}(\tilde{t}, \tilde{t'}) \in_E \mathit{ginsts}(P_{=p\cdot 1})$, and that $\theta' = \theta \cdot \tau'$. Since τ and ρ are induced by θ, θ' induces $\tau \cdot \tau'$ and the same ρ. We have that $Q'\tau' = (P|_{p\cdot 1}\tau\rho)\tau' = P|_p\tau\tau'\rho$ and therefore $Q'\tau \leftrightsquigarrow_P \mathsf{state}_{p\cdot 1}(\tilde{t}, \tilde{t'})$. Thus, and since in $(t_1, N); Q' \leftrightsquigarrow_P \mathsf{state}_p(\tilde{t})$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^{\#} \{\, \text{in } (t_1, N); Q' \,\}^{\#} \cup^{\#} \{\, Q'\tau' \,\}^{\#}$ and $S_n = S_{n-1} \setminus^{\#} \{\, \mathsf{In}(\langle t_1, t_2\rangle), \mathsf{state}_p(\tilde{t}) \,\}^{\#} \cup^{\#} \{\, \mathsf{state}_{p\cdot 1}(\tilde{t}, \tilde{t'}) \,\}^{\#}$, Condition 3 holds in both cases.

Condition 7 holds in the first case, where $j < n$ such that $\text{Send}(\mathit{mid}, t) \in_E E_j$, as

$$\begin{aligned}
\mathcal{U}_{n'+1} &= \mathcal{U}_{n'} \setminus^{\#} \{\, t \,\}^{\#} \\
&=_E \left\{ t' \,\middle|\, \begin{array}{l} \exists j \leq m, \mathit{mid}.\ \text{Send}(\mathit{mid}, t') \in F_j \\ \wedge\, \forall j'.\text{Receive}(\mathit{mid}, t') \notin_E F_{j'} \wedge t' \neq_E t \end{array} \right\} && \text{by IH} \\
&= \left\{ t' \,\middle|\, \begin{array}{l} \exists j \leq n-1, \mathit{mid}.\ \text{Send}(\mathit{mid}, t') \in F_j \\ \wedge\, \forall j'.\text{Receive}(\mathit{mid}, t') \notin_E F_{j'} \wedge t' \neq_E t \end{array} \right\} && \text{as } E_{m+1}, \ldots, E_{n-1} = \emptyset \\
&=_E \left\{ t' \,\middle|\, \begin{array}{l} \exists j \leq n, \mathit{mid}.\ \text{Send}(\mathit{mid}, t') \in F_j \\ \wedge\, \forall j'.\text{Receive}(\mathit{mid}, t') \notin_E F_{j'} \end{array} \right\},
\end{aligned}$$

as $\text{Receive}(\mathit{mid}, t) \in E_n$ and $\text{Send}(\mathit{mid}, t) \in_E E_j$. In the second case, where there is no such j, we have

$$\begin{aligned}
\{\, t \mid \exists j \leq n, \mathit{mid}.\ \text{Send}(\mathit{mid}, t) \in F_j \wedge \forall j'.\text{Receive}(\mathit{mid}, t) \notin_E F_{j'} \,\}^{\#} = \\
\{\, t \mid \exists j \leq n-1, \mathit{mid}.\ \text{Send}(\mathit{mid}, t) \in F_j, \wedge \forall j'.\text{Receive}(\mathit{mid}, t) \notin_E F_{j'} \,\}^{\#}.
\end{aligned}$$

Therefore

$$\begin{aligned}
\mathcal{U}_{n'+1} &= \mathcal{U}_n \\
&=_E \left\{ t' \,\middle|\, \begin{array}{l} \exists j \leq m, \mathit{mid}.\ \text{Send}(\mathit{mid}, t') \in F_j \\ \wedge\, \forall j'.\text{Receive}(\mathit{mid}, t') \notin_E F_{j'} \end{array} \right\} && \text{by IH} \\
&= \left\{ t' \,\middle|\, \begin{array}{l} \exists j \leq n-1, \mathit{mid}.\ \text{Send}(\mathit{mid}, t') \in F_j \\ \wedge\, \forall j'.\text{Receive}(\mathit{mid}, t') \notin_E F_{j'} \end{array} \right\} && \text{as } E_{m+1}, \ldots, E_{n-1} = \emptyset \\
&= \left\{ t' \,\middle|\, \begin{array}{l} \exists j \leq n, \mathit{mid}.\ \text{Send}(\mathit{mid}, t') \in F_j \\ \wedge\, \forall j'.\text{Receive}(\mathit{mid}, t') \notin_E F_{j'} \end{array} \right\},
\end{aligned}$$

showing that Condition 7 holds. Conditions 1, 2 and 4 hold trivially.

□

D Proofs about *filter* and *hide*

The following proposition states that if a set of traces satisfies the translated formula then the filtered traces satisfy the original formula.

Proposition 2. *Let Tr be a set of traces and φ a trace formula. We have that*

$$Tr \vDash^{\star} [\![\varphi]\!]_{\star} \text{ iff } \mathit{filter}(Tr) \vDash^{\star} \varphi$$

where $\star$ is either $\forall$ or $\exists$.

Proof. We first show the two directions for the case $\star = \forall$. We start by showing that $Tr \models^{\forall} [\![\varphi]\!]$ implies $\mathit{filter}(Tr) \models \varphi$.

$$\begin{aligned} Tr \models^{\forall} [\![\varphi]\!]_{\forall} &\Rightarrow \mathit{filter}(Tr) \models^{\forall} [\![\varphi]\!]_{\forall} && \text{(since } \mathit{filter}(Tr) \subseteq Tr\text{)} \\ &\Leftrightarrow \mathit{filter}(Tr) \models^{\forall} \alpha \Rightarrow \varphi && \text{(by definition of } [\![\varphi]\!]_{\forall}\text{)} \\ &\Leftrightarrow \mathit{filter}(Tr) \models^{\forall} \varphi && \text{(since } \mathit{filter}(Tr) \models^{\forall} \alpha\text{)} \end{aligned}$$

We next show that $\mathit{filter}(Tr) \models^{\forall} \varphi$ implies $Tr \models^{\forall} [\![\varphi]\!]_{\forall}$.

$$\begin{aligned} \mathit{filter}(Tr) \models^{\forall} \varphi &\Rightarrow \mathit{filter}(Tr) \models^{\forall} \alpha \wedge \varphi && \text{(since } \mathit{filter}(Tr) \models^{\forall} \alpha\text{)} \\ &\Leftrightarrow Tr \models^{\forall} \neg\alpha \vee (\alpha \wedge \varphi) && \text{(since } \mathit{filter}(Tr) \subseteq Tr \text{ and } (Tr \setminus \mathit{filter}(Tr)) \not\models^{\forall} \alpha\text{)} \\ &\Leftrightarrow Tr \models^{\forall} \alpha \Rightarrow \varphi \\ &\Leftrightarrow Tr \models^{\forall} [\![\varphi]\!]_{\forall} && \text{(by definition of } [\![\varphi]\!]_{\forall}\text{)} \end{aligned}$$

The case of $\star = \exists$ now easily follows:

$$Tr \models^{\exists} [\![\varphi]\!]_{\exists} \text{ iff } Tr \not\models^{\forall} [\![\neg\varphi]\!]_{\forall} \text{ iff } \mathit{filter}(Tr) \not\models^{\forall} \neg\varphi \text{ iff } \mathit{filter}(Tr) \models^{\exists} \varphi.$$

□

Proposition 3. *Let Tr be a set of traces and φ a well-formed trace formula. We have that*

$$Tr \models^{\star} \varphi \textit{ iff } \mathit{hide}(Tr) \models^{\star} \varphi$$

where $\star$ is either $\forall$ or $\exists$.

Proof. We start with the case $\star = \exists$ and show the stronger statement that for a trace tr

$$\forall\theta.\exists\theta'.\ \text{if } (tr,\theta) \models \varphi \text{ then } (\mathit{hide}(tr),\theta') \models \varphi$$

and

$$\forall\theta.\exists\theta'.\ \text{if } (\mathit{hide}(tr),\theta) \models \varphi \text{ then } (tr,\theta') \models \varphi.$$

We will show both statements by a nested induction on $|tr|$ and the structure of the formula. (The underlying well-founded order is the lexicographic ordering of the pairs consisting of the length of the trace and the size of the formula.)

If $|tr| = 0$ then $tr = []$ and $tr = \mathit{hide}(tr)$ which allows us to directly conclude letting $\theta' := \theta$.

If $|tr| = n$, we define $\overline{tr}$ and F such that $tr = \overline{tr} \cdot F$. By induction hypothesis we have that

$$\forall\overline{\theta}.\exists\overline{\theta}'.\ \text{if } (\overline{tr},\overline{\theta}) \models \varphi \text{ then } (\mathit{hide}(\overline{tr}),\overline{\theta}') \models \varphi$$

and

$$\forall\overline{\theta}.\exists\overline{\theta}'.\ \text{if } (\mathit{hide}(\overline{tr}),\overline{\theta}) \models \varphi \text{ then } (\overline{tr},\overline{\theta}') \models \varphi.$$

We proceed by structural induction on φ.

- $\varphi = \bot$, $\varphi = i \lessdot j$, $\varphi = i \doteq j$ or $t_1 \doteq t_2$. In these cases we trivially conclude as the truth value of these formulas does not depend on the trace and for both statements we simply let $\theta' := \theta$.
- $\varphi = f@i$. We start with the first statement. Suppose that $(tr,\theta) \models f@i$. If $\theta(i) < n$ then we have also that $\overline{tr},\theta \models f@i$. By induction hypothesis, there exists $\overline{\theta}'$ such that $(\overline{tr},\overline{\theta}') \models f@i$. Hence we also have that $(tr,\overline{\theta}') \models f@i$ and letting $\theta' := \overline{\theta}'$ allows us to conclude. If $\theta(i) = n$ we know that $f \in tr_n$. As φ is well-formed $f \notin \mathcal{F}_{res}$ and hence $f \in \mathit{hide}(tr)_{n'}$ where $n' = |\mathit{hide}(tr)|$. The proof of the other statement is similar.
- $\varphi = \neg\varphi'$, $\varphi = \varphi_1 \wedge \varphi_2$, or $\varphi = \exists x : s.\varphi'$. We directly conclude by induction hypotheses (on the structure of φ).

From the above statements we easily have that $Tr \models^{\exists} \varphi$ iff $\mathit{hide}(Tr) \models^{\exists} \varphi$. The case of $\star = \forall$ now easily follows:

$$Tr \models^{\forall} \varphi \text{ iff } Tr \not\models^{\exists} \neg\varphi \text{ iff } \mathit{hide}(Tr) \not\models^{\exists} \neg\varphi \text{ iff } \mathit{hide}(Tr) \models^{\forall} \varphi$$

□

E Getting rid of the α_{inev} axiom

While the α_{inev} axiom is useful (and necessary) for proving the correctness of the translation it decreases the performance of the tamarin prover. We will now show that for an interesting fragment of our logic the axiom is not needed.

Definition 27. *Let $\mathcal{L}$ be the set of traces formulas where all terms of sort temp belong to $\mathcal{V}_{temp}$. We denote by $\mathcal{L}^+$ (resp. $\mathcal{L}^-$) the subsets of $\mathcal{L}$ where all the occurrences of the fact K are positive (resp. negative), defined inductively as follows:*

- *If ψ is atomic then $\psi \in \mathcal{L}^+$*
- *If ψ is atomic and $\psi \neq K(t)@i$ then $\psi \in \mathcal{L}^-$*
- *If $\psi_1, \psi_2 \in \mathcal{L}^+$ (resp. $\psi_1, \psi_2 \in \mathcal{L}^-$), then $\psi_1 \wedge \psi_2$ and $\psi_1 \vee \psi_2$ belong to $\mathcal{L}^+$ (resp. $\mathcal{L}^-$)*
- *If $\psi \in \mathcal{L}^+$ (resp. $\psi \in \mathcal{L}^-$), then $\exists x.\psi, \forall x.\psi \in \mathcal{L}^+$ (resp. $\exists x.\psi, \forall x.\psi \in \mathcal{L}^-$)*
- *If $\psi \in \mathcal{L}^+$ (resp. $\psi \in \mathcal{L}^-$), then $\neg\psi \in \mathcal{L}^-$ (resp. $\neg\psi \in \mathcal{L}^+$)*

We note that $\mathcal{L}^-$ is generally expressive enough to model security protocols: a security protocol expresses that on all traces an adversary does *not* know a given term. Similarly $\mathcal{L}^+$ may be used to express attacks: there exists a trace where the adversary does know a given term.

The following theorem states formally that the translation without the axiom α_{inev} is correct for verifying validity of trace formulas in $\mathcal{L}^-$ and satisfiability of trace formulas in $\mathcal{L}^+$

Theorem 2. *Given a well-formed ground process P and a well-formed trace formula φ we have that*

$$\forall \psi \in \mathcal{L}^+.\ traces^{pi}(P) \models^{\exists} \psi \Leftrightarrow traces^{msr}([\![P]\!]) \models^{\exists} (\!|\psi|\!)_{\exists}$$

$$\forall \psi \in \mathcal{L}^-.\ traces^{pi}(P) \models^{\forall} \psi \Leftrightarrow traces^{msr}([\![P]\!]) \models^{\forall} (\!|\psi|\!)_{\forall}$$

where $(\!|\varphi|\!)_{\forall} = (\alpha \setminus \alpha_{inev}) \Longrightarrow \varphi$ and $(\!|\varphi|\!)_{\exists} = (\alpha \setminus \alpha_{inev}) \Longrightarrow \varphi$.

References

[1] Martín Abadi and Cédric Fournet. "Mobile Values, New Names, and Secure Communication". In: *28th ACM Symp. on Principles of Programming Languages (POPL'01)*. ACM, 2001, pp. 104–115.

[2] Myrto Arapinis, Eike Ritter, and Mark Ryan. "StatVerif: Verification of Stateful Processes". In: *24th IEEE Computer Security Foundations Symposium (CSF'11)*. IEEE Comp. Soc., 2011, pp. 33–47.

[3] Alessandro Armando et al. "The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications." In: *17th International Conference on Computer Aided Verification (CAV'05)*. LNCS. Springer, 2005, pp. 281–285.

[4] N. Asokan, Victor Shoup, and Michael Waidner. "Asynchronous protocols for optimistic fair exchange". In: *IEEE Symposium on Security and Privacy (S&P'98)*. IEEE Comp. Soc., 1998, pp. 86–99.

[5] David A. Basin, Jannik Dreier, and Ralf Sasse. "Automated Symbolic Proofs of Observational Equivalence". In: *22nd Conference on Computer and Communications Security (CCS'15)*. ACM, 2015, pp. 1144–1155.

[6] Bruno Blanchet, Martín Abadi, and Cédric Fournet. "Automated Verification of Selected Equivalences for Security Protocols". In: *Symposium on Logic in Computer Science (LICS'05)*. IEEE Comp. Soc., 2005, pp. 331–340.

[7] Jan Cederquist and Muhammad Torabi Dashti. "An intruder model for verifying liveness in security protocols". In: *ACM Workshop on Formal methods in security engineering, (FMSE'06)*. 2006, pp. 23–32.

[8] Rohit Chadha, Max Kanovich, and Andre Scedrov. "Inductive methods and contract-signing protocols". In: *8th ACM Conference on Computer and Communications Security*. ACM, 2001, pp. 176–185.

[9] Rohit Chadha, Steve Kremer, and Andre Scedrov. "Formal Analysis of Multi-Party Contract Signing". In: *Journal of Automated Reasoning* 36.1-2 (2006), pp. 39–83.

[10] Rohit Chadha et al. "Automated verification of equivalence properties of cryptographic protocols". In: *ACM Transactions on Computational Logic* 17.4 (2016).

[11] Rohit Chadha et al. "Contract signing, optimism, and advantage". In: *CONCUR 2003 — Concurrency Theory*. LNCS. Springer-Verlag, 2003, pp. 366–382.

[12] Véronique Cortier, Graham Steel, and Cyrille Wiedling. "Revoke and Let Live: A Secure Key Revocation API for Cryptographic Devices". In: *19th ACM Conference on Computer and Communications Security (CCS'12)*. ACM, 2012, pp. 918–928.

[13] Cas J.F. Cremers. "The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols". In: *20th Conference on Computer Aided Verification (CAV'08)*. LNCS. Springer, 2008, pp. 414–418.

[14] Santiago Escobar, Catherine Meadows, and José Meseguer. "Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties". In: *Foundations of Security Analysis and Design V*. LNCS. Springer, 2009, pp. 1–50.

[15] Shimon Even and Yacov Yacobi. *Relations among Public Key Signature Systems*. Tech. rep. 175. Technion, 1980.

[16] Juan A. Garay, Markus Jakobsson, and Philip D. MacKenzie. "Abuse-Free Optimistic Contract Signing". In: *Advances in Cryptology—Crypto'99*. LNCS. Springer, 1999, pp. 449–466.

[17] Sigrid Gürgens and Carsten Rudolph. "Security analysis of efficient (Un-)fair non-repudiation protocols". In: *Formal Asp. Comput.* 17.3 (2005), pp. 260–276.

[18] Steve Kremer and Robert Künnemann. "Automated Analysis of Security Protocols with Global State". In: *35th IEEE Symposium on Security and Privacy (S&P'14)*. IEEE Comp. Soc., 2014, pp. 163–178.

[19] Steve Kremer and Robert Künnemann. "Automated analysis of security protocols with global state". In: *Journal of Computer Security* 24.5 (2016), pp. 583–616.

[20] Steve Kremer, Olivier Markowitch, and Jianying Zhou. "An Intensive Survey of Fair Non-repudiation Protocols". In: *Computer Communications* 25.17 (2002), pp. 1606–1621.

[21] Steve Kremer and Jean-François Raskin. "A Game-Based Verification of Non-Repudiation and Fair Exchange Protocols". In: *Journal of Computer Security* 11.3 (2003), pp. 399–429.

[22] Steve Kremer and Jean-François Raskin. "Game Analysis of Abuse-Free Contract Signing". In: *Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW'02)*. IEEE Comp. Soc., 2002, pp. 206–220.

[23] Leslie Lamport. "Proving the correctness of multiprocess programs". In: *IEEE transactions on software engineering* 2 (1977), pp. 125–143.

[24] Simon Meier. "Advancing automated security protocol verification". PhD thesis. Diss., Eidgenössische Technische Hochschule ETH Zürich, Nr. 20742, 2013.

[25] *OPC Unified Architecture Specification, Part 6: Mappings, Release 1.02*. OPC Foundation. 2012.

[26] Bruno Blanchet, Ben Smyth, and Vincent Cheval. *ProVerif 1.88: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial*. 2013.

[27] Benedikt Schmidt. "Formal Analysis of Key-Exchange Protocols and Physical Protocols". PhD thesis. ETH Zürich, 2012.

[28] Benedikt Schmidt et al. “Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties”. In: *25th IEEE Computer Security Foundations Symposium (CSF'12)*. IEEE Comp. Soc., 2012, pp. 78–94.

[29] Benedikt Schmidt et al. “The TAMARIN Prover for the Symbolic Analysis of Security Protocols”. In: *25th International Conference on Computer Aided Verification (CAV'13)*. LNCS. Springer, 2013, pp. 696–701.

[30] Steve Schneider. “Formal Analysis of a Non-Repudiation Protocol”. In: *11th IEEE Computer Security Foundations Workshop (CSFW'98)*. 1998, pp. 54–65.

[31] Vitaly Shmatikov and John C. Mitchell. “Finite-state analysis of two contract signing protocols.” In: *Theor. Comput. Sci.* 283.2 (2002), pp. 419–450.