Automated Analysis of Security Protocols with Global State

Steve Kremer, Robert Künnemann

Journal of Computer Security 0 (0) 1
IOS Press

Steve Kremer [a] and Robert Künnemann [b]
[a] *Inria Nancy – Grand Est and LORIA, France*
[b] *CISPA and Saarland University, Germany*

Automated analysis of security protocols with global state

Abstract. Security APIs, key servers and protocols that need to keep the status of transactions, require to maintain a global, non-monotonic state, e.g., in the form of a database or register. However, most existing automated verification tools do not support the analysis of such stateful security protocols – sometimes because of fundamental reasons, such as the encoding of the protocol as Horn clauses, which are inherently monotonic. A notable exception is the recent tamarin prover which allows specifying protocols as multiset rewrite (msr) rules, a formalism expressive enough to encode state. As multiset rewriting is a "low-level" specification language with no direct support for concurrent message passing, encoding protocols correctly is a difficult and error-prone process.

We propose a process calculus which is a variant of the applied pi calculus with constructs for manipulation of a global state by processes running in parallel. We show that this language can be translated to msr rules whilst preserving all security properties expressible in a dedicated first-order logic for security properties. The translation has been implemented in a prototype tool which uses the tamarin prover as a backend. We apply the tool to several case studies among which a simplified fragment of PKCS#11, the Yubikey security token, and an optimistic contract signing protocol.

Keywords: Automated verification, Stateful security protocols, Security APIs

1. Introduction

Automated analysis of security protocols has been extremely successful. Using automated tools, flaws have been for instance discovered in the Google Single Sign On Protocol [5], in commercial security tokens implementing the PKCS#11 standard [10], and one may also recall Lowe's attack [23] on the Needham-Schroeder public key protocol 17 years after its publication. While efficient tools such as ProVerif [7], AVISPA [4] or Maude-NPA [14] exist, these tools are generally not suitable to analyze protocols that require *non-monotonic global state*, i.e., some database, register or memory location that can be read and altered by different parallel threads. The input language of the AVISPA tool offers support for this kind of state but only supports a bounded number of sessions. This is particularly restrictive when analysing security APIs where attacks typically require several keys and API calls, which are difficult to bound a priori. ProVerif, one of the most efficient and widely used protocol analysis tools for an unbounded number of sessions, relies on an abstraction that encodes protocols in first-order Horn clauses. This abstraction is well suited for the monotonic knowledge of an attacker (who never forgets), makes the tool extremely efficient for verifying an unbounded number of protocol sessions and allows to build on existing techniques for Horn clause resolution. However, Horn clauses are inherently monotonic: once a

fact is true it cannot be set to false anymore. As a result, even though ProVerif's input language, a variant of the applied pi calculus [2], allows a priori encodings of a global memory, the abstractions performed by ProVerif introduce false attacks. In the ProVerif user manual [8, Section 6.3.3] such an encoding of memory cells and its limitations are indeed explicitly discussed:

> *"Due to the abstractions performed by ProVerif, such a cell is treated in an approximate way: all values written in the cell are considered as a set, and when one reads the cell, ProVerif just guarantees that the obtained value is one of the written values (not necessarily the last one, and not necessarily one written before the read)."*

Some work [3,24,13] has nevertheless used ingenious encodings of mutable state in Horn clauses, but these encodings have limitations that we discuss below.

A prominent example where non-monotonic global state appears are security APIs, such as the RSA PKCS#11 standard [27], IBM's CCA [11] or the trusted platform module (TPM) [34]. They have been known to be vulnerable to logical attacks for some time [22,9] and formal analysis has shown to be a valuable tool to identify attacks and find secure configurations. One promising paradigm for analyzing security APIs is to regard them as a participant in a protocol and use existing analysis tools. However, Herzog [18] already identified not accounting for mutable global state as a major barrier to the application of security protocol analysis tools to verify security APIs. Apart from security APIs many other protocols need to maintain databases: key servers need to store the status of keys, in optimistic contract signing protocols a trusted party maintains the status of a contract, RFID protocols maintain the status of tags and more generally websites may need to store the current status of transactions.

Our contributions We propose a tool for analyzing protocols that may involve non-monotonic global state, relying on Schmidt et al.'s tamarin tool [29,30] as a backend. We designed a new process calculus that extends the applied pi calculus by defining, in addition to the usual constructs for specifying concurrent processes, constructs for explicitly manipulating global state. This calculus serves as the tool's input language. The heart of our tool is a translation from this extended applied pi calculus to a set of multiset rewrite rules that can then be analyzed by tamarin which we use as a backend. We prove the correctness of this translation and show that it preserves all properties expressible in a dedicated first order logic for expressing security properties. As a result, relying on the tamarin prover, we can analyze protocols without bounding the number of sessions, nor making any abstractions. Moreover it allows to model a wide range of cryptographic primitives by the means of equational theories. As the underlying verification problem is undecidable, tamarin may not terminate. However, it offers an interactive mode with a GUI which allows to manually guide the tool in its proof. Our specification language includes support for private channels, global state and locking mechanisms (which are crucial to write meaningful programs in which concurrent threads manipulate a common memory). The translation has been carefully engineered in order to favor termination by tamarin, including a goal ranking method tailored to the output of the translation. Several case studies illustrate the tool's capability: a simple security API in the style of PKCS#11, a complex case study of the Yubikey security token, as well as several examples analyzed by other tools that aim at analyzing stateful protocols. In all of these case studies we were able to avoid restrictions that were necessary in previous works.

Related work The most closely related work is the StatVerif tool by Arapinis et al. [3]. They propose an extension of the applied pi calculus, similar to ours, which is translated to Horn clauses and analyzed by the ProVerif tool. Their translation is sound but allows for false attacks, limiting the scope of protocols that can be analyzed. Moreover, StatVerif can only handle a finite number of memory cells: when

analyzing an optimistic contract signing protocol this appeared to be a limitation and only the status of a single contract was modeled, providing a manual proof to justify the correctness of this abstraction. In important case studies, e. g. key-management APIs like PKCS#11 or the Yubikey, an unbounded amount of memory is required to avoid artificially bounding the number of keys or Yubikey devices. Finally, StatVerif is limited to the verification of secrecy properties. As illustrated by the Yubikey case study, our work is more general and we are able to analyze complex injective correspondence properties.

Mödersheim [24] proposed a language with support for sets together with an abstraction where all objects that belong to the same sets are identified. His language, which is an extension of the low level AVISPA intermediate format, is compiled into Horn clauses that are then analyzed, e. g., using ProVerif. His approach is tightly linked to this particular abstraction, limiting the scope of applicability, e. g., when keys may be compromised (all keys with the same attributes are abstracted to one and the same, thus either all are revealed, or none) or when the set of states a key or value is not bounded a priori (as in the Yubikey case study). Mödersheim also discusses the need for a more high-level specification level which we provide in this work.

There has also been work tailored to particular applications. In [12], Delaune et al. show by a dedicated hand proof that for analyzing PKCS#11 one may bound the message size. Their analysis still requires to artificially bound the number of keys. Similarly in spirit, Delaune *et al.* [13] give a dedicated result for analyzing protocols based on the TPM and its registers. However, the number of reboots (which reinitialize registers) needs to be limited.

Guttman [17] also extended the strand space model by adding support for state. While the protocol execution is modeled using the classical strand spaces model, state is modeled by a multiset of facts, and manipulated by multiset rewrite rules. The extended model has been used for analyzing by hand an optimistic contract signing protocol. In a more recent paper Ramsdell et al. [26] propose another approach also based on the strand space model. Using the CPSA tool they obtain a symbolic representation (called skeletons) of all possible attacks. However, as their model analyzed by CPSA encodes the state in a message passing style, the tool may consider false attacks. They therefore import the CPSA result, as an axiom, in the theorem prover PVS and, based on a more precise model of the possible state transitions, refine their analysis to exclude the false attacks. The approach has been applied to the so-called envelope protocol, which was also analysed (in a slightly more restrictive model) in [13].

In the goal of relating different approaches for protocol analysis Bistarelli et al. [6] also proposed a translation from a process algebra to multiset rewriting: they do however not consider private channels, have no support for global state and assume that processes have a particular structure. These limitations significantly simplify the translation and its correctness proof. Moreover their work does not include any tool support for automated verification.

Obviously any protocol that we are able to analyze can be directly analyzed by the tamarin prover [29,30] as the rules produced by our translation could have been given directly as an input to tamarin. Indeed, tamarin has already been used for analyzing a model of the Yubikey device [21], the case studies presented with Mödersheim's abstraction, as well as those presented with StatVerif. It is furthermore able to reproduce the aforementioned results on PKCS#11 [12] and the TPM [13] – moreover, it does so without bounding the number of keys, security devices, reboots, etc. Contrary to ProVerif, tamarin sometimes requires additional *typing lemmas* which are used to guide the proof. These lemmas need to be written by hand (but are proved automatically). In our case studies we also needed to provide a few such lemmas manually. In our opinion, an important disadvantage of tamarin is that protocols are modeled as a set of multiset rewrite rules. This representations is very low level and far away from actual protocol implementations, making it very difficult to model a protocol adequately. Encoding private

channels, nested replications and locking mechanisms directly as multiset rewrite rules is a tricky and error prone task. As a result we observed that, in practice, the protocol models tend to be simplified. For instance, locking mechanisms are often omitted, modeling protocol steps as a single rule and making them effectively atomic. Such more abstract models may obscure issues in concurrent protocol steps and increase the risk of implicitly excluding attacks in the model that are well possible in a real implementation, e. g., race conditions. Using a more high-level specification language, such as our process calculus, arguably eases protocol specification and overcomes some of these risks. Examples in which the explicit modelling of locking mechanisms in SAPIC improved the protocol and/or the analysis include the Yubikey case study presented in Section 7. In our modelling of the Yubikey the server can handle several requests from different devices in parallel, which was not possible in the direct modelling in [21]. Another example is the model of the enhanced authorization mechanism introduced in the TPM 2.0 specification by Shao et al. [32]. In this work, a model of the TPM that executes API commands sequentially is compared to one that executes them in parallel, finding flaws in the parallel version. The TPM model in the tamarin example files models TPM commands as atomic steps. While an explicit modelling of locking steps is possible in tamarin, judging from existing models, it is not widely used, although protocols and analysis could benefit from it.

Since the first prototype of this translation was presented [19], subsequent work has demonstrated and extended its scope. The present calculus and verification method have been used to verify a configuration of the key-management API PKCS#11 [20] and was extended with loops to allow for the analysis of the streaming protocol TESLA [25]. In [32], Shao et al. used our tool to analyse the enhanced authorization mechanism introduced in the TPM 2.0 specification.

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. 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, that 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. Symmetric encryption can be modelled using a signature $\Sigma = \{ senc/2, sdec/2 \}$ and an equational theory defined by $sdec(senc(m, k), k) = m$.

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 $\langle x_1, \langle x_2, \langle \ldots, \langle x_{n-1}, x_n\rangle \ldots\rangle$.

We suppose the usual notion of positions for terms. 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 the means of events, and for defining 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 will 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. 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. Given function g we let $g(x) = \perp$ when $x \notin \mathbf{D}(x)$. When $g(x) = \perp$ we say that g is undefined for x. We define the function $f := g[a \mapsto b]$ with $\mathbf{D}(f) = \mathbf{D}(g) \cup \{\, a \,\}$ as $f(a) := b$ and $f(x) := g(x)$ for $x \neq a$.

$\langle P,Q\rangle$ ::= 0
| $P \mid Q$
| $!\,P$
| $\nu n\colon \mathit{fresh}; P$
| out$(M, N); P$
| in$(M, N); P$
| if Pred then P [else Q]

$\langle P,Q\rangle$ ::= *(continued)*
| event $F\,; P \quad (F \in \mathcal{F})$
| insert M,$N; P$
| delete $M; P$
| lookup M as x in P [else Q]
| lock $M; P$
| unlock $M; P$

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

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 operation, e.g. $S_1 \cup^\# S_2$ denotes the multiset union of multisets S_1, S_2. Given a multiset S we denote by $\mathit{set}(S)$ 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 $\mathit{idx}(S)$ the set of positions in S, i.e., $\mathbb{N}_n$ when S has n elements, and for $i \in \mathit{idx}(S)$ 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 $\exists e' \in S.\ e' =_E e$. $\subset_E$ 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.

3. A cryptographic pi calculus with explicit state

3.1. Syntax and informal semantics

Our calculus, dubbed SAPiC (Stateful Applied Pi calculus) is a variant of the applied pi calculus [2]. In addition to the usual operators for concurrency, replication, communication and name creation, it offers several constructs for reading and updating an explicit global state. The grammar for processes is described in Figure 1.

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. The construct $\nu n; P$ binds the name $n \in FN$ in P and models the generation of a fresh, random value. The processes out(M, N); P and in(M, N); P represent the output, respectively input, of message N on channel M. Readers familiar with the applied pi calculus [2] 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 if Pred then P 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 if $\mathit{equal}(M, N)$ then P else Q will execute P if $M =_E N$ and Q otherwise. (In the following, we will use $M = N$ as 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 to write else Q when Q is 0, as well as trailing 0 processes.

The remaining constructs are used for manipulating state and are new compared to the applied pi calculus. The construct insert M,N binds the value N to a key M. Successive inserts allow changing this binding. We emphasise that we have only one value bound to a key, and that successive inserts update

the binding. The delete M operation simply "undefines" the mapping for the key M. 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 Djkstra'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.

In the following example, which will serve as our running example, we model a security API that, even though much simplified, illustrates the most salient issues that occur in the analysis of security APIs such as PKCS#11 [12,10,15].

Example. We consider a security device that allows the creation of keys in its secure memory. The user can access the device via an API. If he creates a key, he obtains a handle, which he can use to let the device perform operations on his behalf. For each handle the device also stores an attribute which defines what operations are permitted for this handle. The goal is that the user can never gain knowledge of the key, as the user's machine might be compromised. We model the device by the following process (we use $\mathsf{out}(m)$ as a shortcut for $\mathsf{out}(c, m)$ for a public channel c):

$!P_{new} \mid !P_{set} \mid !P_{dec} \mid !P_{wrap}$, where

$P_{new} := \nu h;\ \nu k;$ event NewKey(h,k); insert $\langle$'key',$h\rangle$,k; insert $\langle$' att ',$h\rangle$, 'dec'; $\mathsf{out}(h)$

In the first line, the device creates a new handle h and a key k and, by the means of the event NewKey(h, k), logs the creation of this key. It then stores the key that belongs to the handle by associating the pair $\langle$'key',$h\rangle$ to the value of the key k. In the next line, $\langle$'att',$h\rangle$ is associated to a public constant 'dec'. Intuitively, we use the public constants 'key' and 'att' to distinguish two databases. The process

$P_{set} :=$ in(h); insert $\langle$'att',$h\rangle$, 'wrap'

allows the attacker to change the attribute of a key from the initial value 'dec' to another value 'wrap'. If a handle has the 'dec' attribute set, it can be used for decryption:

$P_{dec} :=$ in($\langle h,c\rangle$); lookup $\langle$'att',$h\rangle$ as a in if a='dec' then
lookup $\langle$'key',$h\rangle$ as k in if *encSucc*(c,k) then
event DecUsing(k,*sdec*(c,k)); out(*sdec*(c,k))

The first lookup stores the value associated to $\langle$'att',$h\rangle$ in a. The value is compared against 'dec'. If the comparison and another lookup for the associated key value k succeeds, we check whether decryption succeeds and, if so, output the plaintext.

If a key has the 'wrap' attribute set, it might be used to encrypt the value of a second key, e. g., to export the key for external storage:

$P_{wrap} :=$ in($\langle h_1,h_2\rangle$); lookup $\langle$'att',$h_1\rangle$ as a_1 in if a_1='wrap' then
lookup $\langle$'key',$h_1\rangle$ as k_1 in lookup $\langle$'key', $h_2\rangle$ as k_2 in
event Wrap(k_1,k_2); out(*senc*(k_2,k_1))

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

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

Fig. 2. Deduction rules.

The bound names of a process are those that are bound by νn. We suppose that all names of sort $fresh$ appearing in the process are under the scope of such a binder. Free names must be of sort pub. A variable x can be bound in two ways: *(i)* by the construct lookup M as x, or *(ii)* $x \in vars(N)$ in the construct in(M, N) and x is not under the scope of a previous binder, While the construct lookup M as x always acts as a binder, the input construct does not rebind an already bound variable but performs pattern matching. For instance in the process

$$P = \mathsf{in}(\mathsf{c}, f(x)); \mathsf{in}(\mathsf{c}, g(x))$$

x is bound by the first input and pattern matched in the second. It might seem odd that lookup acts as a binder, while input does not. We justify this decision as follows: as P_{dec} and P_{wrap} in the previous example show, lookups appear often after input was received. If lookup were to use pattern matching, the following process

$$P = \mathsf{in}(c, x); \mathsf{lookup}\ \text{`store'}\ \mathsf{as}\ x\ \mathsf{in}\ P'$$

might unexpectedly perform a check if 'store' contains the message given by the adversary, instead of binding the content of 'store' to x, due to an undetected clash in the naming of variables.

A process is ground if it does not contain any free variables. We denote by $P\sigma$ the application of the homomorphic extension of the substitution σ to P. As usual we suppose that the substitution only applies to free variables. We sometimes interpret the syntax tree of a process as a term and write $P|_p$ to refer to the subprocess of P at position p (where $|$, if and lookup are interpreted as binary symbols, all other constructs as unary). Our tool supports additional syntactic sugar: else-branches consisting of the 0-Process can be omitted, as well as let-construct for terms (let $m = dec(c, k)$ in out(m)) and processes (let $P = \ldots$ in $!P$) perform simple substitution.

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.*

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

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

Example. If one key is used to wrap 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.

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 5-tuple $(\mathcal{E}, \mathcal{S}, \mathcal{P}, \sigma, \mathcal{L})$ 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 acquired locks.

The transition relation is defined by the rules described 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.

Definition 2 (Traces of P)**.** *Given a ground process P we define the* set of traces of P *as*

$$traces^{pi}(P) = \Big\{ [F_1, \ldots, F_n] \mid (\emptyset, \emptyset, \{P\}, \emptyset, \emptyset) \stackrel{F_1}{\Longrightarrow} (\mathcal{E}_1, \mathcal{S}_1, \mathcal{P}_1, \sigma_1, \mathcal{L}_1) \stackrel{F_2}{\Longrightarrow} \ldots \stackrel{F_n}{\Longrightarrow} (\mathcal{E}_n, \mathcal{S}_n, \mathcal{P}_n, \sigma_n, \mathcal{L}_n) \Big\}$$

Example. In Figure 5 we display the transitions corresponding to the creation of a key on the security device in our running example and witness that $[\mathrm{NewKey}(h', k')] \in traces^{pi}(P)$.

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 [29].

Definition 3 (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 4 (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:*

Standard operations:

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

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

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

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

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

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\mathrm{out}(M,N);P\},\sigma,\mathcal{L}) \xrightarrow{K(M)} (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P\},\sigma\cup\{^{N}/_{x}\},\mathcal{L}) \quad \text{if } x \text{ is fresh and } \nu\mathcal{E}.\sigma\vdash M$$

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

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\mathrm{out}(M,N);P,\mathrm{in}(M',N');Q\},\sigma,\mathcal{L}) \longrightarrow (\mathcal{E},\mathcal{S},\mathcal{P}\cup\{P,Q\tau\},\sigma,\mathcal{L}) \quad \text{if } M=_E M' \text{ and } N=_E N'\tau \text{ and } \tau \text{ grounding for } N'$$

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

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

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

Operations on global state:

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

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

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

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

$$(\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{\text{lock } M;P\},\sigma,\mathcal{L}) \longrightarrow (\mathcal{E},\mathcal{S},\mathcal{P}\cup^{\#}\{P\},\sigma,\mathcal{L}\cup\{M\}) \quad \text{if } M\not\in_E\mathcal{L}$$

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

Fig. 4. Operational semantics

- 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* $\cap_{r''=_E r'} names(r'') \cap FN \subseteq \cap_{l''=_E l'} names(l'') \cap FN$.

We define one distinguished rule FRESH which is the only rule allowed to have Fr-facts on the right-hand side

$$\text{FRESH} : [] -[]\to [\mathsf{Fr}(x : fresh)]$$

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

$$(\emptyset,\emptyset,\{\,!P_{new}|\underbrace{!P_{set}\;|!P_{dec}\;|!P_{wrap}}_{=:\mathcal{P}'}\,\}^{\#},\emptyset,\emptyset) \to (\emptyset,\emptyset,\{\,P_{new}\,\}^{\#}\cup^{\#}\mathcal{P}',\emptyset,\emptyset)$$

$$\to\ (\emptyset,\emptyset,\{\,\nu\,h;\nu\,k;\mathsf{event}\,\mathrm{NewKey}(h,k);\ldots\}^{\#}\cup^{\#}\mathcal{P}',\emptyset,\emptyset)$$

$$\to^{*}(\{\,h',k'\,\},\emptyset,\{\,\mathsf{event}\,\mathrm{NewKey}(h',k');\ldots\}^{\#}\cup^{\#}\mathcal{P}',\emptyset,\emptyset)$$

$$\xrightarrow{\mathrm{NewKey}(h',k')}\ (\{\,h',k'\,\},\emptyset,\{\,\mathsf{insert}\,\langle\text{`}key',h'\rangle,\ k';\ldots\}^{\#}\cup^{\#}\mathcal{P}',\emptyset,\emptyset)$$

$$\to^{*}(\{\,h',k'\,\},\mathcal{S},\{\,\mathsf{out}(h');0\,\}^{\#}\cup^{\#}\mathcal{P}',\emptyset,\emptyset)\to^{*}\ (\{\,h',k'\,\},\mathcal{S},\mathcal{P}',\{\,{}^{h'}/_{x_1}\,\},\emptyset)$$

where $\mathcal{S}(\langle\text{`}key',h'\rangle)=k'$ and $\mathcal{S}(\langle\text{`}att',h'\rangle)=$ 'dec'.

Fig. 5. Example of transitions modelling the creation of a key on a PKCS#11-like device

Definition 5 (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 $l -[\,a\,]\to r \in_E ginsts(R\cup \textsc{Fresh})$, $lfacts(l)\subseteq^{\#} S$ *and* $pfacts(l)\subseteq S$.

Definition 6 (Executions)**.** *Given a multiset rewriting system R we define its set of executions as*

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

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

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

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

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 [29] 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 8 (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 complex injective correspondence 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 9 (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:*

$$
\begin{array}{ll}
(tr,\theta) \vDash \bot & \textit{never} \\
(tr,\theta) \vDash F@i & \textit{iff } \theta(i) \in idx(tr) \textit{ and } F\theta \in_E tr_{\theta(i)} \\
(tr,\theta) \vDash i \lessdot j & \textit{iff } \theta(i) < \theta(j) \\
(tr,\theta) \vDash i \doteq j & \textit{iff } \theta(i) = \theta(j) \\
(tr,\theta) \vDash t_1 \approx t_2 & \textit{iff } t_1\theta =_E t_2\theta \\
(tr,\theta) \vDash \neg\varphi & \textit{iff not } (tr,\theta) \vDash \varphi \\
(tr,\theta) \vDash \varphi_1 \wedge \varphi_2 & \textit{iff } (tr,\theta) \vDash \varphi_1 \textit{ and } (tr,\theta) \vDash \varphi_2 \\
(tr,\theta) \vDash \exists x : s.\varphi & \textit{iff there is } u \in \mathbf{D}(s) \textit{ such that } (tr, \theta[x \mapsto u]) \vDash \varphi
\end{array}
$$

For readability, we define $t_1 \gtrdot t_2$ as $\neg(t_1 \lessdot t_2 \vee t_1 \doteq t_2)$ and $(\leq, \not\doteq, \geq)$ 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 $\forall i.\ i \leq j \to \varphi$. When φ is a ground formula we sometimes simply write $tr \vDash \varphi$ as the satisfaction of φ is independent of the valuation.

Definition 10 (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 $traces^{pi}(P) \vDash^\forall \varphi$, and that φ is satisfied in P, written $P \vDash^\exists \varphi$, if $traces^{pi}(P) \vDash^\exists \varphi$.

Example. The following trace formula expresses secrecy of keys generated on the security API, which we introduced in Section 3.

$$\neg(\exists h, k\colon msg,\ i, j\colon temp.\ \mathrm{NewKey}(h,k)@i \wedge \mathrm{K}(k)@j)$$

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 [29] 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.

$$\begin{array}{rcll}
\mathsf{Out}(x) & -[\ ]\rightarrow & !\mathsf{K}(x) & (\text{MDOUT}) \\
!\mathsf{K}(x) & -[\ K(x)\ ]\rightarrow & \mathsf{In}(x) & (\text{MDIN}) \\
 & -[\ ]\rightarrow & !\mathsf{K}(x : pub) & (\text{MDPUB}) \\
\mathsf{Fr}(x : fresh) & -[\ ]\rightarrow & !\mathsf{K}(x : fresh) & (\text{MDFRESH}) \\
!\mathsf{K}(x_1), \ldots, !\mathsf{K}(x_k) & -[\ ]\rightarrow & !\mathsf{K}(f(x_1, \ldots, x_k)) \text{ for } f \in \Sigma^k & (\text{MDAPPL})
\end{array}$$

Fig. 6. The set of rules MD.

6.1. Definition of the translation of processes

To model the adversary's message deduction capabilities, we introduce the set of rules MD defined in Figure 6. 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 11 (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 [3]. 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 12 (Process annotation). *Given a ground process P we define the annotated ground process $\overline{P}$ as* $\mathsf{ap}(P, [])$ *where:*

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

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

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

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

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

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

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

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

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 $\mathrm{lock}\ t$ 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 [19] contains a different but equivalent formulation of this definition.

Definition 13 (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.

The two first restrictions of well-formed processes can be assumed without loss of generality as processes and formulas can be consistently renamed to avoid reserved variables and α-converted to avoid binding names or variables several times. Also note that the second condition is not necessarily preserved during an execution, e.g. when unfolding a replication, $!P$ and P may bind the same names. We only require this condition to hold on the initial process for our translation to be correct.

The annotation of locks restricts the set of protocols we can translate, but allows us to obtain better verification results, since we can predict which unlock is "supposed" to close a given lock. This additional information is helpful for tamarin's backward reasoning. We think that our locking mechanism captures

all practical use cases. Obviously, locks can be modelled both in tamarin's multiset rewriting calculus (this is actually what the translation does) and Mödersheim's set rewriting calculus [24]. However, protocol steps typically consist of a single input, followed by several database lookups, and finally an output. In practice, they tend to be modelled as a single rule, and are therefore atomic. Real implementations are however different, as several entities might be involved, database lookups could be slow, etc. In this case, such simplified models could, e. g., miss race conditions. To the best of our knowledge, StatVerif is the only comparable tool that models locks explicitly and it has stronger restrictions.

Definition 14. *Given a well-formed ground process P we define the labelled multiset rewriting system $[\![P]\!]$ as*

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

where the rule INIT *is defined as*

$$\textsc{Init} : [] \; -[\; \mathrm{Init}() \;]\rightarrow [\mathsf{state}_{[]}()] \text{ and}$$

$[\![P, p, \tilde{x}]\!]$ *is defined inductively for process P, position $p \in \mathbb{N}^*$ and sequence of variables $\tilde{x}$ in Figure 7. For a position p in P we define state_p to be persistent if $P|_p = !Q$ for some process Q; otherwise state_p is linear.*

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

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

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 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 $\mathsf{state}_{p\cdot 1}$ 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

$$
\begin{aligned}
[\![0, p, \tilde{x}]\!] &= \{[\mathsf{state}_p(\tilde{x})] \to []\} \\
[\![P \mid Q, p, \tilde{x}]\!] &= \{[\mathsf{state}_p(\tilde{x})] \to [\mathsf{state}_{p\cdot 1}(\tilde{x}), \mathsf{state}_{p\cdot 2}(\tilde{x})]\} \\
&\quad \cup [\![P, p\cdot 1, \tilde{x}]\!] \cup [\![Q, p\cdot 2, \tilde{x}]\!] \\
[\![!P, p, \tilde{x}]\!] &= \{[!\mathsf{state}_p(\tilde{x})] \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})] \,{-\![}\ \mathit{ProtoNonce}(n_a : \mathit{fresh})\ ]\!{\to} \\
&\quad [\mathsf{state}_{p\cdot 1}(\tilde{x}, n_a : \mathit{fresh})]\} \cup [\![P, p\cdot 1, (\tilde{x}, n_a : \mathit{fresh})]\!] \\
[\![\mathsf{Out}(M, N); P, p, \tilde{x}]\!] &= \{[\mathsf{state}_p(\tilde{x}), \mathsf{In}(M)] \,{-\![}\ \mathrm{InEvent}(M)\ ]\!{\to}\ [\mathsf{Out}(N), \mathsf{state}_{p\cdot 1}(\tilde{x})], \\
&\quad [\mathsf{state}_p(\tilde{x})] \to [\mathsf{Msg}(M, N), \mathsf{state}_p^{\mathsf{semi}}(\tilde{x})], \\
&\quad [\mathsf{state}_p^{\mathsf{semi}}(\tilde{x}), \mathsf{Ack}(M, N)] \to [\mathsf{state}_{p\cdot 1}(\tilde{x})]\} \cup [\![P, p\cdot 1, \tilde{x}]\!] \\
[\![\mathsf{In}(M, N); P, p, \tilde{x}]\!] &= \{[\mathsf{state}_p(\tilde{x}), \mathsf{In}(\langle M, N\rangle)] \,{-\![}\ \mathrm{InEvent}(\langle M, N\rangle)\ ]\!{\to} \\
&\quad [\mathsf{state}_{p\cdot 1}(\tilde{x} \cup \mathit{vars}(N))], [\mathsf{state}_p(\tilde{x}), \mathsf{Msg}(M, N)] \to \\
&\quad [\mathsf{state}_{p\cdot 1}(\tilde{x} \cup \mathit{vars}(N)), \mathsf{Ack}(M, N)]\} \\
&\quad \cup [\![P, p\cdot 1, \tilde{x} \cup \mathit{vars}(N)]\!] \\
[\![\text{if } pr(M_1, \ldots, M_k) \text{ then } P \text{ else } Q, p, \tilde{x}]\!] &= \begin{array}{l}\{[\mathsf{state}_p(\tilde{x})] \,{-\![}\ \mathrm{Pred}_{pr}(M_1, \ldots, M_k)\ ]\!{\to}\ [\mathsf{state}_{p\cdot 1}(\tilde{x})], \\ \quad [\mathsf{state}_p(\tilde{x})] \,{-\![}\ \mathrm{Pred\_not}_{pr}(M_1, \ldots, M_k)\ ]\!{\to}\ [\mathsf{state}_{p\cdot 2}(\tilde{x})]\}\end{array} \\
&\quad \cup [\![P, p\cdot 1, \tilde{x}]\!] \cup [\![Q, p\cdot 2, \tilde{x}]\!] \\
[\![\text{event } F; P, p, \tilde{x}]\!] &= \{[\mathsf{state}_p(\tilde{x})] \,{-\![}\ \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})] \,{-\![}\ \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})] \,{-\![}\ \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 \text{ else } Q, p, \tilde{x}]\!] &= \{[\mathsf{state}_p(\tilde{x})] \,{-\![}\ \mathrm{IsIn}(M, v)\ ]\!{\to}\ [\mathsf{state}_{p\cdot 1}(\tilde{M}, v)], \\
&\quad [\mathsf{state}_p(\tilde{x})] \,{-\![}\ \mathrm{IsNotSet}(M)\ ]\!{\to}\ [\mathsf{state}_{p\cdot 2}(\tilde{x})]\} \\
&\quad \cup [\![P, p\cdot 1, (\tilde{x}, v)]\!] \cup [\![Q, p\cdot 2, \tilde{x}]\!] \\
[\![\text{lock}^l\ s; P, p, \tilde{x}]\!] &= \{[\mathrm{Fr}(\mathrm{lock}_l), \mathsf{state}_p(\tilde{x})] \,{-\![}\ \mathrm{Lock}(\mathit{lock}_l, s)\ ]\!{\to}\ [\mathsf{state}_{p\cdot 1}(\tilde{x}, \mathit{lock}_l)]\} \\
&\quad \cup [\![P, p\cdot 1, \tilde{x}]\!] \\
[\![\text{unlock}^l\ s; P, p, \tilde{x}]\!] &= \{[\mathsf{state}_p(\tilde{x})] \,{-\![}\ \mathrm{Unlock}(\mathit{lock}_l, s)\ ]\!{\to}\ [\mathsf{state}_{p\cdot 1}(\tilde{x})]\} \cup [\![P, p\cdot 1, \tilde{x}]\!]
\end{aligned}
$$

Fig. 7. Translation of processes: definition of $[\![P, p, \tilde{x}]\!]$

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 $\mathsf{state}_{p\cdot 1}(\ldots)$. 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

$$
\begin{array}{rcl}
[] & -[\ \mathrm{Init}()\ ]\to & [!\mathsf{state}_{[]}()] \\
[!\mathsf{state}_{[]}()] & -[\ \ ]\to & [\mathsf{state}_{[1]}()] \\
[\mathsf{state}_{[1]}(), \mathsf{Fr}(h)] & -[\ \ ]\to & [\mathsf{state}_{[11]}(h)] \\
[\mathsf{state}_{[11]}(h), \mathsf{Fr}(k)] & -[\ \ ]\to & [\mathsf{state}_{[111]}(k,h)] \\
[\mathsf{state}_{[111]}(k,h)] & -[\ \mathrm{Event}(), \mathrm{NewKey}(h,k)\ ]\to & [\mathsf{state}_{[1111]}(k,h)] \\
[\mathsf{state}_{[1111]}(k,h)] & -[\ \mathrm{Insert}(\langle\text{'key'}, h\rangle, k)\ ]\to & [\mathsf{state}_{[11111]}(k,h)] \\
[\mathsf{state}_{[11111]}(k,h)] & -[\ \mathrm{Insert}(\langle\text{'att'}, h\rangle, \text{'dec'})\ ]\to & [\mathsf{state}_{[111111]}(k,h)] \\
[\mathsf{state}_{[111111]}(k,h)] & -[\ \ ]\to & [\mathsf{Out}(h), \mathsf{state}_{[1111111]}(k,h)]
\end{array}
$$

Fig. 8. The set of multiset rewrite rules $[\![!P_{new}]\!]$ (omitting the rules in MD)

facts and add labels. Some of these labels are used to restrict the set of executions. For instance the label 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.

Example. Figure 8 illustrates the above translation by presenting the set of msr rules $[\![!P_{new}]\!]$ (omitting the rules in MD already shown in Figure 6).

A graph representation of an example trace, similar to the one generated by the tamarin tool, is depicted in Figure 9. Every node stands for the application of a multiset rewrite rule, where the premises are at the top, the conclusions at the bottom, and the actions (if any) annotate the node. Every premise needs to have a matching conclusion, visualized by the arrows, to ensure the graph depicts a valid msr execution. (This is a simplification of the dependency graph representation tamarin uses to perform backward-induction [29,30].) We also note that in the current example $!\mathsf{state}_{[]}()$ is persistent and can therefore be used multiple times as a premise. As $\mathsf{Fr}(\,)$ facts are generated by the FRESH rule which has an empty premise and action, we omit instances of FRESH and leave those premises, but only those, disconnected.

Remark 1. *One may note that, while for all other operators, the translation produces well-formed multiset rewriting rules (as long as the process is well-formed itself), this is not the case for the translation of the* lookup operator, i. e., it violates the well-formedness condition from Definition 4. Tamarin's constraint solving algorithm requires all rules, with the exception of FRESH, to be well-formed. We show however that, under these specific conditions, the solution procedure is still correct. See Appendix A for the proof.

6.2. Definition of the translation of trace formulas

We can now define the translation for formulas.

Definition 15. *Given a well-formed trace formula φ we define*

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

where α is defined in Figure 10.

The formula α uses the actions of the generated rules to filter out executions that we wish to discard:

– α_{init} ensures that the init rule is only fired once.

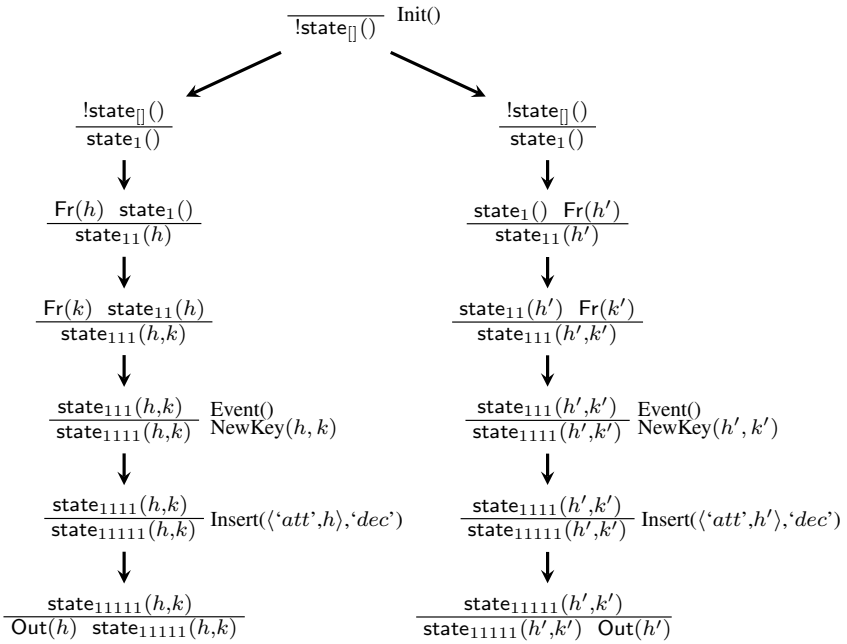

Fig. 9. Example trace for the translation of $!P_{new}$.

- α_{pred} ensures that we only consider traces where for all positive and negative branches in conditionals the corresponding predicate formula, respectively its negation, hold.
- α_{in} and α_{notin} ensure that a successful lookup was preceded by an insert that was neither revoked nor overwritten while an unsuccessful lookup was either never inserted, or deleted and never reinserted.
- α_{lock} checks that between each two matching locks there must be an unlock. Furthermore, between the first of these locks and the corresponding unlock, there is neither a lock nor an unlock.
- α_{inev} ensures that whenever an instance of MDIN is required to generate an In-fact, it is generated as late as possible, i. e., there is no visible event between the action $K(t)$ produced by MDIN, and a rule that requires $\mathsf{In}(t)$.

We also note that $Tr \vDash^{\forall} [\![\varphi]\!]_{\forall}$ iff $Tr \nvDash^{\exists} [\![\neg\varphi]\!]_{\exists}$.

6.3. Discussion of design choices

There exist certainly other ways of correctly translating our calculus into msr rules. Most of our choices were guided by the way tamarin internally works. To better appreciate our choices we will give a high-level overview of the procedure implemented in tamarin. A detailed review of the procedure is however out of scope of this paper and we refer the reader to [29] for a detailed description.

A short overview of tamarin. Tamarin basically applies a backward reasoning approach to try to find a trace which satisfies a given formula. (Validity claims are first translated to satisfiability claims.) This is reminiscent to the reasoning when proving protocol correctness in the strand space model [33]. More precisely, rather than reasoning about traces, tamarin reasons about *dependency graphs*, an enriched representation of traces. Dependency graphs are DAGs, where each node corresponds to a ground instance

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

$$\alpha_{init} := \quad \forall i, j.\ \text{Init}()@i \wedge \text{Init}()@j \Longrightarrow i \doteq j$$

$$\alpha_{pred} := \quad \bigwedge_{pr \in \Sigma_{pred}} \{\forall x_1, \ldots, x_k, i.\ \text{Pred}_{pr}(x_1, \ldots, x_k)@i \Longrightarrow \phi_{pr} \mid pr \text{ is of arity k}\} \wedge$$
$$\bigwedge_{pr \in \Sigma_{pred}} \{\forall x_1, \ldots, x_k, i.\ \text{Pred\_not}_{pr}(x_1, \ldots, x_k)@i \Longrightarrow \neg(\phi_{pr}) \mid pr \text{ is of arity k}\}$$

$$\begin{aligned}\alpha_{in} := \quad & \forall x, y, t_3.\ \text{IsIn}(x, y)@t_3 \Longrightarrow \exists t_2.\ \text{Insert}(x, y)@t_2 \wedge t_2 \lessdot t_3 \\ & \wedge \forall t_1.\quad \text{Delete}(x)@t_1 \Longrightarrow (t_1 \lessdot t_2 \vee t_3 \lessdot t_1) \\ & \wedge \forall t_1, y.\ \text{Insert}(x, y)@t_1 \Longrightarrow (t_1 \leq t_2 \vee t_3 \lessdot t_1)\end{aligned}$$

$$\begin{aligned}\alpha_{notin} := \quad & \forall x, t_3.\ \text{IsNotSet}(x)@t_3 \Longrightarrow (\forall t_1, y.\ \text{Insert}(x, y)@t_1 \Longrightarrow t_3 \lessdot t_1) \vee \\ & (\exists t_1.\ \text{Delete}(x)@t_1 \wedge t_1 \lessdot t_3 \\ & \quad \wedge \forall t_2, y.\ (\text{Insert}(x, y)@t_2 \wedge t_2 \lessdot t_3) \Longrightarrow t_2 \lessdot t_1)\end{aligned}$$

$$\begin{aligned}\alpha_{lock} := \forall x, l, l', t_1, t_3.\ & \text{Lock}(l, x)@t_1 \wedge \text{Lock}(l', x)@t_3 \wedge t_1 \lessdot t_3 \\ & \Longrightarrow \exists t_2.\ \text{Unlock}(l, x)@t_2 \wedge t_1 \lessdot t_2 \lessdot t_3 \\ & \quad \wedge (\forall t_0.\ \text{Unlock}(l, x)@t_0 \Longrightarrow t_0 \doteq t_2) \\ & \quad \wedge (\forall l', t_0.\ \text{Lock}(l', x)@t_0 \Longrightarrow t_0 \leq t_1 \vee t_2 \lessdot t_0) \\ & \quad \wedge (\forall l', t_0.\ \text{Unlock}(l', x)@t_0 \Longrightarrow t_0 \lessdot t_1 \vee t_2 \leq t_0)\end{aligned}$$

$$\begin{aligned}\alpha_{inev} := \quad & \forall x, t_3.\ \text{InEvent}(x)@t_3 \Longrightarrow \exists t_2.\ \text{K}(x)@t_2 \wedge t_2 \lessdot t_3 \\ & \quad \wedge (\forall t_0.\ \text{Event}()@t_0 \Longrightarrow (t_0 \lessdot t_2 \vee t_3 \lessdot t_0)) \\ & \quad \wedge (\forall t_0, x'.\ \text{K}(x')@t_0 \Longrightarrow (t_0 \leq t_2 \vee t_3 \lessdot t_0))\end{aligned}$$

Fig. 10. Definition of α.

of an msr rule and the edges represent the causal dependencies among these rules. For every premise of a rule there is an incoming edge from another rule with a conclusion that matches the premise. Moreover, linear facts may have at most one outgoing edge and fresh rules are unique. Every topological ordering then corresponds to a trace.

Tamarin's backward search is formalised by a constrained solving algorithm. The solutions of a constraint system are the dependency graphs whose traces satisfy the constraints. The initial constraint system is simply the formula to be satisfied. The procedure then applies simplification rules which preserve all solutions. If the constraint system reaches $\bot$ the formula is unsatisfiable. In case no more rules can be applied the system is solved, and the dependency graphs that are the solutions of the constraint system can be directly constructed.

Slightly simplifying, a typical rule in the constraint solving algorithm would state that if the formula is of the form $a@i$ then the dependency graph must contain a node corresponding to a rule $\ell \xrightarrow{b} r$ with an action b that matches a. Next, it will try to solve each premise in ℓ by adding a constraint that this rule must be preceded by a node corresponding to rules with a fact in its conclusion matching this premise.

Another example of a simplification rule is the following, which reasons about the uniqueness of fresh names: when the constraint system contains both $\mathsf{Fr}(n)@i$ and $\mathsf{Fr}(n)@j$ it concludes that $i \doteq j$.

The constraint simplification procedure may of course enter a loop and not terminate. This is natural given that the underlying problem is undecidable. The algorithm can nevertheless be guided by heuristics to avoid some of these loops and use previously proven lemmas and axioms to prune otherwise infinite branches.

Design choices. The axioms in the translation of the formula are designed to work hand in hand with the translation of the process into rules. They express the correctness of traces with respect to our calculus' semantics, but are also meant to guide tamarin's constraint solving algorithm. The use of axioms, rather than other possible encodings, often helps the algorithm to enforce termination as they can be used to cut branches that are not consistent with the axioms. We will discuss the axioms related to state manipulation.

Let us first consider the axioms related to lock actions. A naïve axiomatization would postulate that *"every lock is preceded by an unlock and no lock or unlock in between, unless it is the first lock."* This would however cause tamarin to loop, as we will see below. We will first illustrate how the axiom α_{lock} avoids this caveat because it only applies to pairs of locks carrying the same annotations.

Consider the constraint solving procedure for the following process

$$P := !(\mathsf{lock}\ \text{`s'};\ \mathsf{lookup}\ \text{`visited'}\ \mathsf{as}\ v\ \mathsf{in}\ \mathsf{unlock}\ \text{`s'}\ \mathsf{else}\ \mathsf{event}\ \mathit{Visit}\ ();\ \mathsf{insert}\ \text{`visited'}, \text{`s'};\ \mathsf{unlock}\ \text{`s'})$$

and the trace formula $\forall i, j.\, \mathit{Visit}()@i \wedge \mathit{Visit}()@j \Longrightarrow i \doteq j$. The msr rules generated by our translation are depicted in Figure 11.

$$\begin{array}{rcl}
[] & -[\ \mathrm{Init}()\ ]\rightarrow & [!\mathsf{state}()] \\
[!\mathsf{state}()] & -[\ \ ]\rightarrow & [\mathsf{state}_1()] \\
[\mathsf{state}_1(), \mathsf{Fr}(l)] & -[\ \mathrm{Lock}(l, \textit{`s'})\ ]\rightarrow & [\mathsf{state}_{11}(l)] \\
[\mathsf{state}_{11}(l)] & -[\ \mathrm{IsIn}(\textit{`visited'}, v)\ ]\rightarrow & [\mathsf{state}_{111}(l, v)] \\
[\mathsf{state}_{111}(l, v)] & -[\ (l, \textit{`s'})\ ]\rightarrow & [\mathsf{state}_{1111}(l, v)] \\
[\mathsf{state}_{11}(l)] & -[\ \mathrm{IsNotSet}(\textit{`visited'})\ ]\rightarrow & [\mathsf{state}_{112}(l)] \\
[\mathsf{state}_{112}(l)] & -[\ \mathrm{Event}(), \mathrm{Visit}()\ ]\rightarrow & [\mathsf{state}_{1121}(l)] \\
[\mathsf{state}_{1121}(l)] & -[\ \mathrm{Unlock}(\textit{`visited'}, \textit{`s'})\ ]\rightarrow & [\mathsf{state}_{11211}(l)] \\
[\mathsf{state}_{11211}(l)] & -[\ \mathrm{Unlock}(l, \textit{`s'})\ ]\rightarrow & [\mathsf{state}_{112111}(l)]
\end{array}$$

Fig. 11. Translation of process P

$$i : \mathrm{Visit}() \qquad j : \mathrm{Visit}() \qquad i \lessdot j$$

Fig. 12. Constraint system resulting from the negation of $\forall i, j.\, \mathit{Visit}()@i \wedge \mathit{Visit}()@j \Longrightarrow i \doteq j$.

1. Tamarin shows validity of the trace formula by showing that its negation $\exists i, j.\, \mathit{Visit}()@i \wedge \mathit{Visit}()@j \wedge (i \lessdot j \vee j \lessdot i)$ is not satisfiable. Two symmetrical constraint systems need to be refuted, we focus on the one pictured in Figure 12, i. e., the case where $i \lessdot j$.

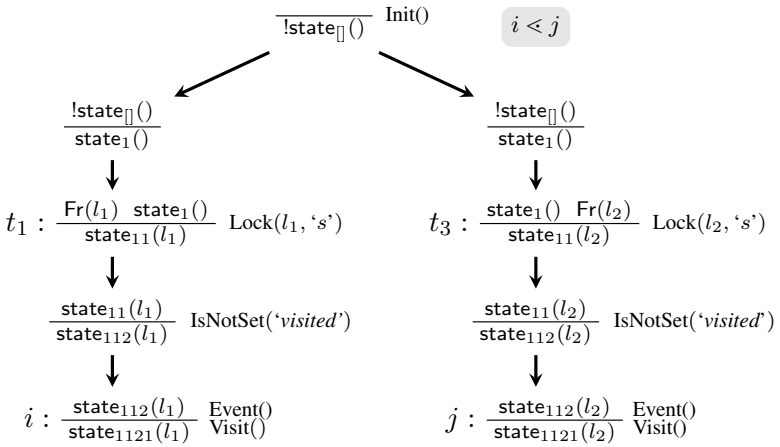

Fig. 13. All state-premises have exactly one matching conclusion and are resolved up to a (unique) instance of INIT.

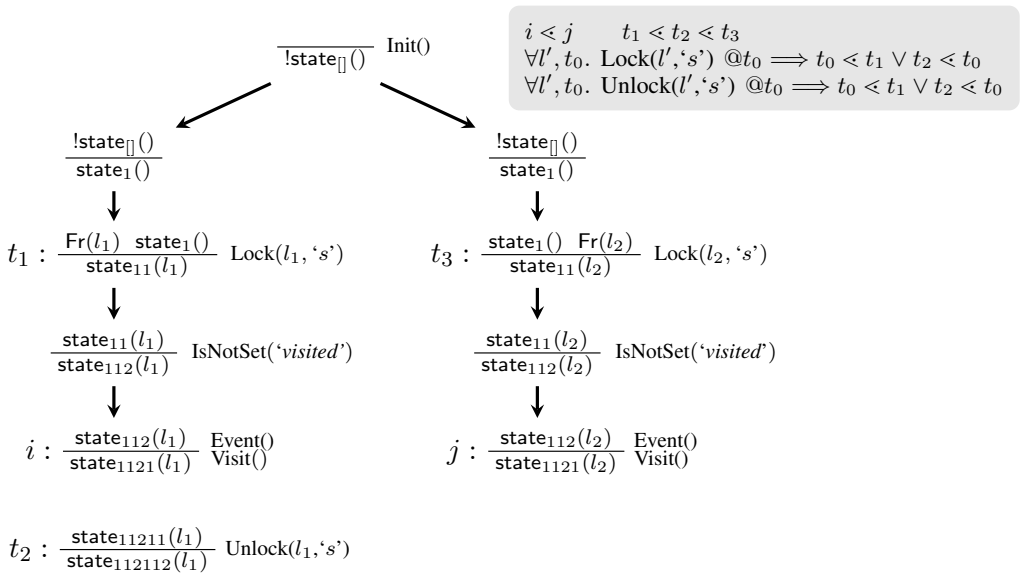

Fig. 14. By α_{lock}, there exists node Unlock(l_1,'s') at position t_2 such that $t_1 \lessdot t_2 \lessdot t_3$ without any matching lock or unlock for 's' between t_1 and t_2.

2. As all state-premises have exactly one rule with a matching conclusion, there are two chains of rule instances from i and j up to the INIT rule, which is unique by α_{init}. Both are recovered in this step, see Figure 13. As tamarin pre-computes chains of rule instantiations whose open premises can be uniquely resolved, this is done in two steps, one for each chain.
3. Now α_{lock} is applied, which adds the constraint that the first lock needs to have a matching unlock, i. e., a node Unlock(l_1,'s') has to appear at some position t_2 between positions t_1 and t_3 as sketched in Figure 14. More precisely, we require the existence of an unlock for *'s'* annotated with l_1, and no lock or unlock for 's' in between. The axiom itself contains only one case, so the only case distinction that takes place is over which rule produces the matching Unlock-action. Due to the

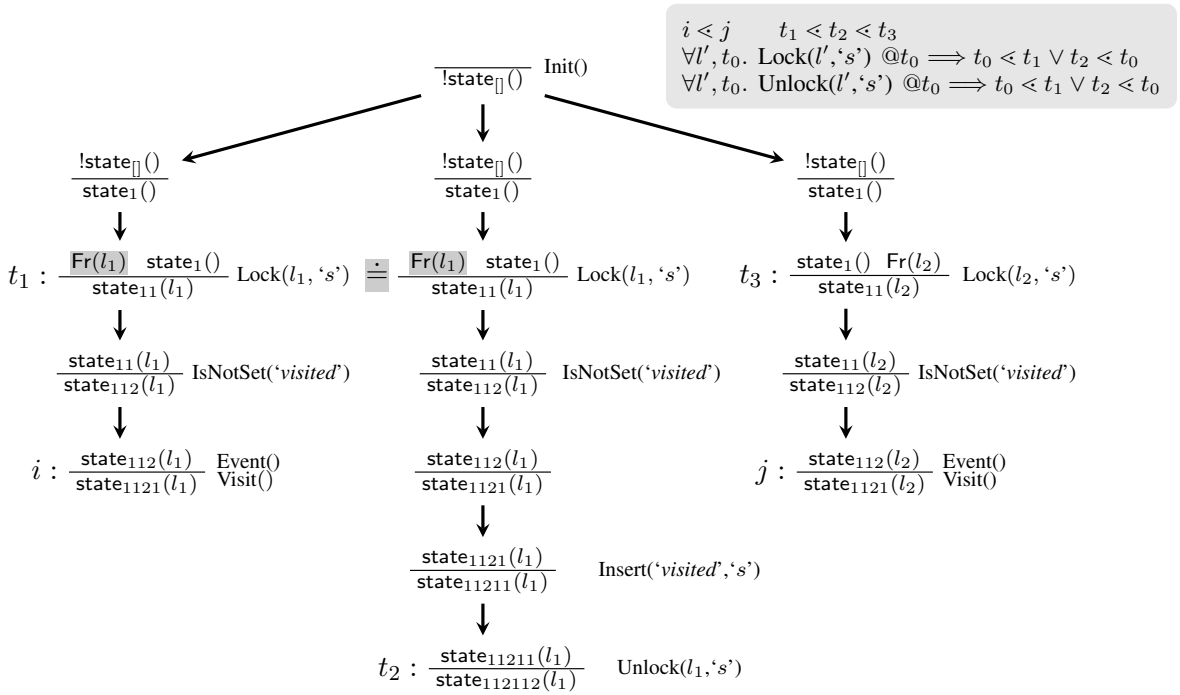

Fig. 15. state-premise at position t_2 can be resolved up to INIT. Same fresh value l_1 is generated at positions t_1 and t_2.

annotation, however, all but one are refuted immediately in the next step, as two nodes containing the same fact $\mathsf{Fr}(l_1)$ in the premise are unified immediately.

4. Due to the annotation, the fact $\mathsf{state}_{11211}(l_1)$ contains the same fresh name l_1 that instantiates the annotation variable in Unlock(l_1,'s') at t_1. Every fact $\mathsf{state}_{p'}(\ldots)$ for some position p' that is a prefix of p and a suffix of the position of the corresponding lock contains this fresh name. Furthermore, every rule instantiation that is an ancestor of a node in the dependency graph corresponds to the execution of a command that is an ancestor in the process tree. Therefore, the backward search eventually reaches the matching lock, including the annotation, which is determined to be l_1, and hence appears in the Fr-premise (Figure 15).
5. Because of the common premise $\mathsf{Fr}(l_1)$, both subgraphs are merged. The result is a sequence of nodes from the first lock to the corresponding unlock, and graph constraints restricting the second lock to not take place between the first lock and the unlock. We note that the axiom α_{lock} is only instantiated once per pair of locks, since it requires that $i \lessdot j$, thereby fixing their order.
   If we would not annotate locks with fresh names, these two subgraphs would not be merged, as they could be different. In fact, the axion α_{lock} would apply again, e. g., for Lock(l_1,'s') (or rather Lock('s')) at t_1 and the newly created rule instantiation with the same action. We would thus run in a loop.
6. We have achieved a total ordering on all rule instantiations that appear in the constrain system. Now α_{notin} can be applied for the rule instantiation at k as pictured in Figure 16. Since $t_2 \lessdot t_3$, it holds that $i' \lessdot k$ and thus the first case can be refuted. The second case is also refuted right away, as there is no rule with action Delete in the translation of P.

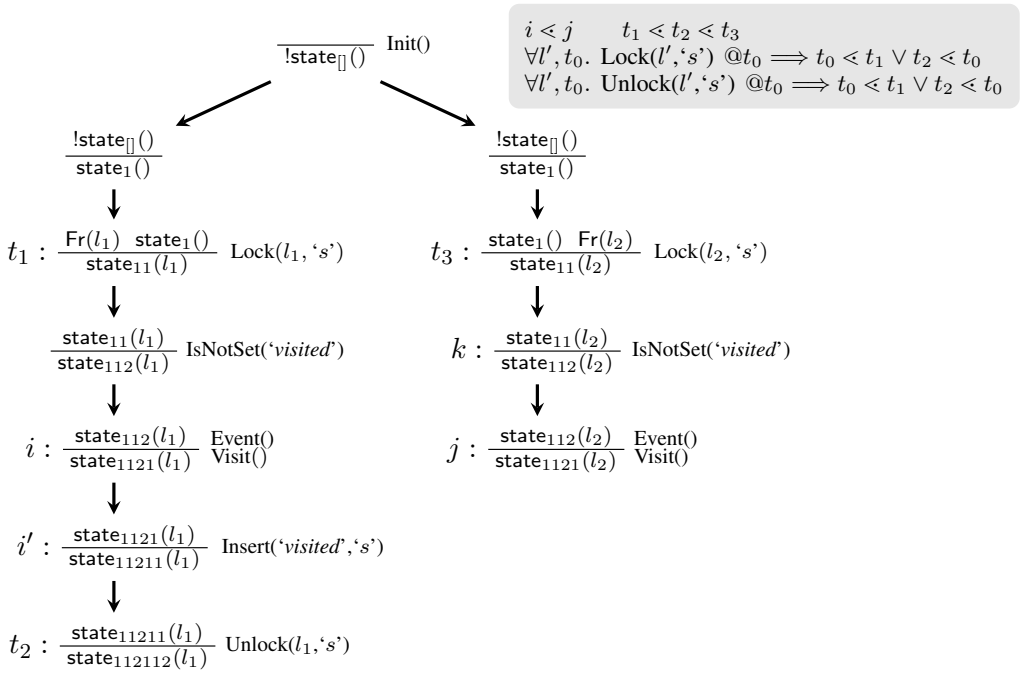

Fig. 16. Because of the identical premise $\mathsf{Fr}(l_1)$ in both chains leading to t_i and t_2, and as all state-facts below position $[1]$ are linear, both subgraphs are merged.

In contrast, consider now the naïve formulation of α_{lock} (*"every lock is preceded by an unlock and no lock or unlock in between, unless it is the first lock"*):

$$\begin{aligned}\alpha'_{lock} = \forall t_1, l, s.Lock(l,s)@t_1 \Longrightarrow (\exists t_0, l'.\ & Unlock(l',s)@t_0 \wedge t_0 \lessdot t_1 \\ & \wedge(\forall t_i, l_i.Lock(l_i,s)@t_i \Rightarrow (t_i \lessdot t_0) \vee (t_1 \lessdot t_i)) \\ & \wedge(\forall t_i, l_i.Unlock(l_i,s)@t_i \Rightarrow (t_i \lessdot t_0) \vee (t_1 \lessdot t_i))) \\ \vee(\forall t_i, l_i.Lock(l_i,s)@t_i & \Rightarrow t_0 \lessdot t_i)\end{aligned}$$

Even if annotations are employed, this would easily provoke a loop: applied after the second step, to the Lock-node at t_3 (see Figure 13), the first case would require a node Unlock(l','s') at position t_0 with $t_0 \lessdot t_3$. Similar to the second step, a chain of rule instances from this node to the unique instantiation of the INIT rule would be created in one step, pictured in Figure 17. Observe that the rule instantiation at position t'_0 has an action Lock(l','s'). As l' is not necessarily equal to l_1 or l_2, this chain of rule instantiations cannot be merged with any other subgraphs. Hence the Lock-action at position t'_0 needs is considered to be new, and thus α'_{lock} can be applied again, resulting in a loop. This loop is triggered whenever an action Lock(l,'s') appears.

In summary, a careful formulation of this axiom was necessary to avoid loops. The annotation helps distinguishing which unlock is expected between two locks, vastly improving the speed of the backward search. This optimisation, however, required us to put restrictions on the locks. The axiom is formulated in a way that links the lock with the corresponding unlock by means of this annotation. The equivalence between α_{lock} and the naïve formulation is non-trivial, but shown in the proof of Lemma 10 in Appendix B.

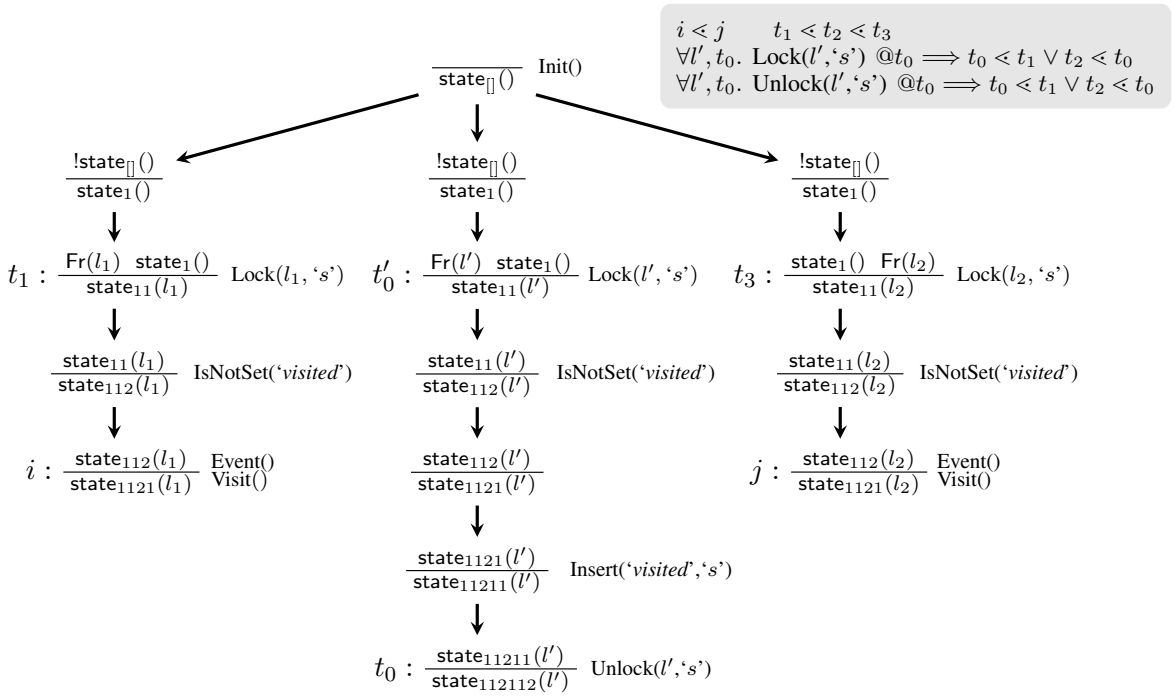

Fig. 17. The naïve formulation α'_{lock} provokes a loop: t_1 and t'_0 are possibly distinct, thus α'_{lock} applies to Lock(l','s') at t'_0.

Similarly, the axioms α_{in} and α_{notin} are designed to work well with tamarin's constraint solving algorithm: when a constraint with the action IsIn is created, by definition of the translation, this corresponds to a lookup command. The existential in α_{in} translates into a graph constraint that postulates the existence of an insert node for the value fetched by the lookup, and three formulas assuring that *(i)* this insert node appears before the lookup, *(ii)* is uniquely defined, i. e., it is the last insert to the corresponding key, and *(iii)* there is no delete in between. Due to these conditions, α_{notin} only adds one Insert node per IsIn node – the case where an axiom postulates a node, which itself allows for postulating yet another node needs to be avoided, as tamarin runs into loops otherwise. The technique of enforcing correctness of the translation through rewriting the formula via these axioms additionally allows us to convey information on the nature of our rules resulting from the translation to the constraint solving algorithm.

6.4. Correctness of the translation

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*

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

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

We here give an overview of the main propositions and lemmas needed to prove Theorem 1. 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 15.

Definition 16. *Let α be the trace formula as defined in Definition 15 and Tr a set of traces. We define*

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

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

Proposition 1. *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 $Tr \vDash^\forall [\![\varphi]\!]$ implies $\mathit{filter}(Tr) \vDash \varphi$.

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

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

$$\begin{aligned}
\mathit{filter}(Tr) \vDash^\forall \varphi &\Rightarrow \mathit{filter}(Tr) \vDash^\forall \alpha \wedge \varphi && \text{(since } \mathit{filter}(Tr) \vDash^\forall \alpha\text{)}\\
&\Leftrightarrow Tr \vDash^\forall \neg\alpha \vee (\alpha \wedge \varphi) && \text{(since } \mathit{filter}(Tr) \subseteq Tr \text{ and } (Tr \setminus \mathit{filter}(Tr)) \nvDash^\forall \alpha\text{)}\\
&\Leftrightarrow Tr \vDash^\forall \alpha \Rightarrow \varphi\\
&\Leftrightarrow Tr \vDash^\forall [\![\varphi]\!]_\forall && \text{(by definition of } [\![\varphi]\!]_\forall\text{)}
\end{aligned}$$

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

$$Tr \vDash^\exists [\![\varphi]\!]_\exists \text{ iff } Tr \nvDash^\forall [\![\neg\varphi]\!]_\forall \text{ iff } \mathit{filter}(Tr) \nvDash^\forall \neg\varphi \text{ iff } \mathit{filter}(Tr) \vDash^\exists \varphi.$$

□

Next we define the *hiding* operation which removes all reserved facts from a trace.

Definition 17 ($hide$)**.** *Given a trace tr and a set of facts F we inductively define $hide([]) = []$ and*

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

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

As expected well-formed formulas that do not contain reserved facts evaluate the same whether reserved facts are hidden or not.

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

$$Tr \vDash^\star \varphi \text{ iff } hide(Tr) \vDash^\star \varphi$$

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

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

and

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

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 $tr = hide(tr)$ 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

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

and

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

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 $(tr, \theta) \vDash f@i$. If $\theta(i) < n$ then we have also that $\overline{tr}, \theta \vDash f@i$. By induction hypothesis, there exists $\overline{\theta}'$ such that $(\overline{tr}, \overline{\theta}') \vDash f@i$. Hence we also have that $(tr, \overline{\theta}') \vDash f@i$ 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 $f \in hide(tr)_{n'}$ where $n' = |hide(tr)|$. 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 $Tr \vDash^\exists \varphi$ iff $hide(Tr) \vDash^\exists \varphi$. The case of $\star = \forall$ now easily follows:

$$Tr \vDash^\forall \varphi \text{ iff } Tr \nvDash^\exists \neg\varphi \text{ iff } hide(Tr) \nvDash^\exists \neg\varphi \text{ iff } hide(Tr) \vDash^\forall \varphi$$

□

We can now state our main lemma which is relating the set of traces of a process P and the set of traces of its translation into multiset rewrite rules.

Lemma 1. *Let P be a well-formed ground process. We have that*

$$traces^{pi}(P) = hide(\mathit{filter}(traces^{msr}([\![P]\!]))).$$

The proof is given in Appendix B. Our main theorem can now be proven by applying Lemma 1, Proposition 2 and Proposition 1.

Proof of Theorem 1.

$$\begin{aligned} traces^{pi}(P) \vDash^{\star} \varphi &\Leftrightarrow hide(\mathit{filter}(traces^{msr}([\![P]\!]))) \vDash^{\star} \varphi && \text{(by Lemma 1)} \\ &\Leftrightarrow \mathit{filter}(traces^{msr}([\![P]\!])) \vDash^{\star} \varphi && \text{(by Proposition 2)} \\ &\Leftrightarrow traces^{msr}([\![P]\!]) \vDash^{\star} [\![\varphi]\!]_{\star} && \text{(by Proposition 1)} \end{aligned}$$

□

7. Case studies and dedicated heuristics

In the following we will briefly overview some case studies we performed. These case studies include a simple security API similar to PKCS#11 [27], the Yubikey security token, the optimistic contract signing protocol by Garay, Jakobsson and MacKenzie (GJM) [16] and a few other examples discussed in Arapinis et al. [3] and Mödersheim [24]. We do not detail all the formal models of the protocols and properties that we studied, and sometimes present slightly simplified versions. All files of our prototype implementation and our case studies are available at

`http://sapic.gforge.inria.fr/`

In addition to the syntax of the calculus described in Section 3 our tool also allows the user to fall back to labelled msr rules inside of processes. The treatment of this extension is described in the conference version [19]. Having an access to the underlying formalism may sometimes be convenient, but as we do not use it in the examples described in this paper we chose to omit this feature to clarify the presentation.

Related work complements these case studies with an analysis of a more complete model of PKCS#11 [20], and the enhanced authorisation mechanism in the TPM 2.0 [32], as well as an extension of SAPIC that allows for the analysis of stream protocols such as TESLA [25].

We will also discuss a dedicated heuristics we developed that favours termination of tamarin on msr systems produced by our tool. The results are summarized in Table 1. For each case study we provide the number of typing lemmas that were needed by the tamarin prover and whether manual guidance of the tool was required. In case no manual guidance is required we also give execution times.

Example	Typing Lemmas	Automated run (w/o heuristics)*	Automated run (w/ heuristics)*
Security API à la PKCS#11	4	no	yes ($2m1s$)
Needham-Schroeder-Lowe [23]	1	yes ($1, 4s$)	yes ($17, 7s$)
Yubikey Protocol [21,35]	5	no	no
GJM protocol [3,16]	0	yes ($11, 5s$)	yes ($9, 9s$)
Mödersheim's example [24]	0	no	yes($0, 8s$)
Security Device [3]	1	yes ($3, 5s$)	yes ($8, 7s$)

* (Running times on Intel i7-4770 CPU 3.40GHz (8 Cores) and 8 GB RAM)

Table 1
Case studies.

7.1. Security API à la PKCS#11

This example illustrates how our modelling might be useful for the analysis of Security APIs in the style of the PKCS#11 standard [27]. Indeed, Künnemann [20] used our tool to perform an automated analysis of PKCS#11 v2.20. In addition to the processes presented in the running example in Section 3 the actual case study models the following two operations: *(i) encryption:* given a handle and a plaintext, the user can request an encryption under the key the handle points to. *(ii) unwrap* given a ciphertext $senc(k_2, k_1)$, and a handle h_1, the user can request the ciphertext to be *unwrapped*, i.e. decrypted, under the key pointed to by h_1. If decryption is successful, the result is stored on the device, and a handle pointing to k_2 is returned. Moreover, contrary to the running example, at creation time keys are assigned the attribute 'init', from which they can move to either 'wrap', or 'unwrap'. Furthermore, the database maps handles to pairs of keys and attributes. See the following snippet:

```
1 in(⟨'set_dec', h⟩); lock h;
2     lookup h as v in
3         if att(v)='init' then
4             event DecKey(h,key(v));
5                 insert h, ⟨key(v), 'dec'⟩;
6                 unlock h
```

Note that, in contrast to the running example, it is necessary to encapsulate the state changes between lock and unlock. Otherwise an adversary can stop the execution after line 3, set the attribute to 'wrap' in a concurrent process and produce a wrapping. After resuming operation at line 4, he can set the key's attribute to 'dec', even though the attribute is set to 'wrap'. Hence, the attacker is allowed to decrypt the wrapping he has produced and can obtain the key. Such subtleties can produce attacks that our modeling allows to detect. If locking is handled correctly, we show secrecy of keys produced on the device, proving the property introduced in Example 6. If locks are removed the attack described before is found. The conference version [19] mistakenly reported that the verification of this example was fully automated, but the verified model contained a typo, where P_{set_wrap} wrote to $\langle attr, h\rangle$ rather than $\langle att, h\rangle$, effectively disabling unwrapping altogether. Using the new heuristics, it is again possible to verify this example automatically.

7.2. Yubikey

The Yubikey [35] is a small hardware device designed to authenticate a user against network-based services. Manufactured by Yubico, a Swedish company, the Yubikey itself is a low cost ($25), thumb-sized USB device. In its typical configuration, it generates one-time passwords based on encryptions of a secret value, a running counter and some random values using a unique AES-128 key contained in the device. The Yubikey authentication server accepts a one-time password only if it decrypts under the correct AES key to a valid secret value containing a counter larger than the last counter accepted. The counter is thus a means to prevent replay attacks. To date, over a million Yubikeys have been shipped to more than 50,000 customers including governments, universities and enterprises, e.g. Google, Microsoft and Facebook [36].

The following process $P_{Yubikey}$ models a single Yubikey, as well as its initial configuration, where an entry in the server's database for the public id pid is created. This entry contains a tuple consisting of the Yubikey's secret id, AES key, and an initial counter value.

$P_{Yubikey} =$
$\nu\ k;\ \nu\ pid;\ \nu\ secretid\ ;$
insert $\langle$'Server', $pid\rangle$, $\langle\ secretid\ , k,$ 'zero'$\rangle$;
insert $\langle$'Yubikey', $pid\rangle$, ('zero'+'one');
event $Init\ (pid,\ secretid\ , k)$;
out(pid); $!P_{Plugin} \mid !P_{ButtonPress}$

Here, the processes $!P_{Plugin}$ and $!P_{ButtonPress}$ model the Yubikey being unplugged and plugged in again (possibly on a different computer), and the emission of the one-time password. We will only discuss $P_{ButtonPress}$ here. When the user presses the button on the Yubikey, the device outputs a one-time password consisting of a counter tc, the secret id $secretid$ and additional randomness npr encrypted using the AES key k. For readability, we leave out events that are only used in helping lemmas as well as message input from the adversary that is included in the model to force him to provide the next counter (which he always can, as it is public).

$P_{ButtonPress} =$
lock pid; lookup $\langle$'Yubikey', $pid\rangle$ as tc in
  insert $\langle$'Yubikey', $pid\rangle$, tc + 'one';
  $\nu\ nonce;\ \nu\ npr;$
  event YubiPress$(pid,\ secretid\ , k, tc)$;
  out$(\langle pid, nonce, senc(\langle\ secretid\ , tc\ , npr\rangle, k)\rangle)$;
unlock pid

The one-time password $senc(\langle secretid, tc, npr\rangle, k)$ can be used to authenticate against a server that shares the same secret key, which we model in the process P_{Server}. The process receives the encrypted one-time password along with the public id pid of a Yubikey and a $nonce$ that is part of the protocol, but is irrelevant for the authentication of the Yubikey on the server. The server then looks up the secret id and the AES key associated to the public id, as well as the last recorded counter value otc. If the key and secret id used in the request match the values retrieved from the database, then the event $\mathrm{Login}(pid, k, tc)$ is logged, marking a successful login of the Yubikey pid with key k for the counter value tc. Afterwards, the old tuple $\langle secretid, k, otc\rangle$ is replaced by $\langle secretid, k, tc\rangle$, to update the latest counter value received.

$P_{Server} =$
! in($\langle pid, nonce, senc(\langle secretid, tc, npr\rangle, k)\rangle$);
lock $\langle$'Server', $pid\rangle$; lookup $\langle$'Server', $pid\rangle$ as $tuple$ in
  if $fst(tuple) = secretid$ then
    if $fst(snd(tuple)) = k$ then
      in (otc); if $snd(snd(tuple)) = otc$ then
        if $smaller(otc, tc)$ then
          event Login(pid, k, tc);
          insert $\langle$'Server', $pid\rangle$, $\langle secretid, k, tc\rangle$;
  unlock $\langle$'Server', $pid\rangle$

Note that, in our modelling, the server keeps one lock per public id, which means that it is possible to have several active instances of the server thread in parallel as long as all requests concern different Yubikeys.

We model the counter as a multiset only consisting of the symbols "one" and "zero". The multiplicity of 'one' in the multiset is the value of the counter. A counter value is considered smaller than another one, if the first multiset is included in the second, therefore

$$\phi_{smaller}(x_1, x_2) := \exists z. x_1 + z = x_2$$

The process we analyse models a single authentication server (that may run arbitrarily many threads) and an arbitrary number of Yubikeys, i. e., $P_{Server} \mid !P_{Yubikey}$. Among other properties, we show by the means of an injective correspondence property that an attacker that controls the network cannot perform replay attacks, and that each successful login was preceded by a user "pressing the button", formally:

$$\forall\, pid, k, x, t_2.\mathrm{Login}(pid, k, x)@t_2 \Rightarrow$$
$$\exists sid, t_1.\mathrm{YubiPress}(pid, sid, k, x)@t_1 \wedge t_1 \lessdot t_2 \wedge \forall t_3.\mathrm{Login}(pid, k, x)@t_3 \Rightarrow t_3 \doteq t_2$$

Besides injective correspondence, we show the absence of replay attacks and the property that a successful login invalidates previously emitted one-time passwords. All three properties follow more or less directly from a stronger invariant, which itself can be proven in 516 steps. To find theses steps, tamarin needs some additional human guidance (17 steps), which can be provided using the interactive mode. This mode still allows the user to complement his manual efforts with automated backward search. The example files contain the modelling in our calculus, the complete proof, and the manual part of the proof which can be verified by tamarin without interaction.

Our analysis makes three simplifications: First, in P_{Server}, we use pattern matching instead of decryption as demonstrated in the process P_{dec} we introduced in Section 3. Second, we omit the CRC checksum and the time-stamp that are part of the one-time password in the actual protocol, since they do not add to the security of the protocol in the symbolic setting. Third, the Yubikey has actually two counters instead of one, a session counter, and a token counter. We treat the session and token counter on the Yubikey as a single value, which we justify by the fact that the Yubikey either increases the session counter and resets the token counter, or increases only the token counter, thereby implementing a complete lexicographical order on the pair $(session\ counter, token\ counter)$.

A similar analysis has already been performed by Künnemann and Steel, using tamarin's multiset rewriting calculus [21]. However, the model in our new calculus is more fine-grained and we believe more

readable. Security-relevant operations like locking and tests on state are written out in detail, resulting in a model that is closer to the real-life operation of such a device. The modeling of the Yubikey takes approximately 38 lines in our calculus, which translates to 49 multiset rewrite rules. The model of [21] contains only four rules, but they are quite complicated, resulting in 23 lines of code. More importantly, the gap between their model and the actual Yubikey protocol is larger – in our calculus, it becomes clear that the server can treat multiple authentication requests in parallel, as long as they do not claim to stem from the same Yubikey. An implementation on the basis of the model from Künnemann and Steel would need to implement a global lock accessible to the authentication server and all Yubikeys. This is however unrealistic, since the Yubikeys may be used at different places around the world, making it unlikely that there exist means of direct communication between them. While a server-side global lock might be conceivable (albeit impractical for performance reasons), a real global lock could not be implemented for the Yubikey as deployed.

7.3. The GJM contract signing protocol [16]

A contract signing protocol allows two parties to sign a contract in a fair way: none of the participants should be bound to the contract without the other participant being bound as well. A straightforward solution is to use a trusted party that collects both signatures on the contract and then sends the signed contracts to each of the participants. Optimistic protocols have been designed to avoid the use of a trusted party whenever possible (optimizing efficiency, and avoiding the potential cost of a trusted party). In these protocols the parties first try to simply exchange the signed contracts; in case of failure, or cheating behavior of one of the parties, the trusted party can be contacted. Depending on the situation, the trusted party may either *abort* the contract, or *resolve* it. In case of an abort decision the protocol ensures that none of the parties obtains a signed contract, while in case of a resolve the protocol ensures that both participants obtain the signed contract. For this the trusted party needs to maintain a database with the current status of all contracts (aborted, resolved, or no decision has been taken). In our calculus the status information is naturally modelled using our insert and lookup constructs. The use of locks is also crucial here to avoid the status to be changed between a lookup and an insert.

This protocol was also studied by Arapinis et al. [3]. They showed the crucial property that a same contract can never be both aborted and resolved. However, due to the fact that StatVerif only supports a finite number of memory cells, they have shown this property for a single contract and provide a manual proof to lift the result to an unbounded number of contracts. We directly prove this property for an unbounded number of contracts.

7.4. Further Case Studies

We investigated the case study presented by Mödersheim [24], a key-server example, as well as a simple security device which served as an example for StatVerif [3]:the device is initialized once, either to left or right. Later on, it accepts pairs of encryptions and decrypts either the left component of the pair or the right component, but not both. As the input language of StatVerif is very similar to ours their model could be easily adapted to our tool. In fact, we were able to remove the restriction to a single security device. Finally, we also illustrate the tool's ability to analyze classical security protocols by analyzing the Needham Schroeder Lowe protocol [23].

7.5. Heuristics

In order to improve our results on the case studies presented in the conference version [19], we have altered the heuristics of the tamarin-prover. We make use of the a priori knowledge that the msr system is an output of our translation. These heuristics can be switched on using the command line switch `--heuristic=p` and alter 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). These heuristics also allowed an automated proof of the PKCS#11 case study [20].

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. The heuristic alters tamarin's standard "smart" heuristic in the following way: state-facts are resolved right away. As state-goals can only be solved by exactly one rule (except for message transmission), and state predicates in the premise of a rule are indexed with a position that is a prefix of the position of the state predicate in the conclusion, loops are impossible and case distinctions rare. Moreover, tamarin precomputes chains and is hence often able to resolve the chain until state_0 in one step. Goals for Unlock-actions are solved right away. As these goals are produced by α_{lock}, they identify the correct unlock using the annotations introduced in Definition 12. By reformulating α_{lock} (compared to the conference article), we were able to avoid the repeated application of this rule. We removed the prioritisation of goals for adversarial deduction of fresh values, as it is counter-productive in the case of handles. They are fresh values that can usually be derived from protocol output, so a case distinction on all possible ways of deriving them is sometimes misleading. Another addition prioritizes goals for Insert-actions when the first element of the key is prefixed "F_", so the user can prioritize the reasoning on lookups to keys like $\langle$'`F_database`'$, p\rangle$. Adversarial deduction for fresh values can be prioritized in the same way, using "L_" instead of "F_" achieves deprioritisation.

7.6. Proof effort

A comparison between the effort needed to derive a proof for a protocol in our calculus and a protocol modelled via multiset rewrite rules is only sound when both model the same thing. Whenever the direct encoding is simplified, e. g., in the Yubikey model, the proof is obviously simpler, but on the other hand, as we have already discussed in Section 7.2, it may be oversimplified. Whenever models were relatively close, our experiments suggested that the same kind of lemmas are needed. In particular for the GJM contract signing protocol, the simple security device and the Needham-Schroeder-Lowe protocol, the lemmas were literally the same. This suggests that these helping lemmas prove properties beyond the level of representation, i.e., properties of the protocol itself.

Our dedicated heuristics discussed in the previous section also improve termination. One may note that tamarin also includes several heuristics that can be chosen from and combined in several ways to help termination. Some of the case studies, e.g., the group protocols analysed in [31], also required the development of dedicated heuristics. Our heuristics benefit from the fact that the msr rules are generated and, therefore, are more restricted than the arbitrary msr rules that may be given to tamarin using a direct msr rule modelling.

When these heuristics fail, or the user wishes to inspect the proof, tamarin's interactive mode allow manual inspection and selection of the proof goals that are chosen at each step. To make use of this, in addition to the working of the tamarin interactive mode, a basic understanding of our translation (but

not of the correctness proof) is necessary. A tight integration of SAPIC into tamarin would surely aid in this regard, but requires significant engineering effort. Such an integration could additionally provide information given by the process description. Relations between locks, lookup and inserts could be highlighted and protocol roles (often defined as abbreviations by protocol designers) distinguished.

For protocols which have complicated control flow or structure (e. g. group protocols [31]), a direct encoding may actually be better suited. We provide a mechanism for embedding labelled msr rules directly inside processes (described in the conference version [19]), which may useful is some circumstances, and such a mixed model might sometimes give the user "the best of the two approaches".

8. Conclusion

We present a process calculus which extends the applied pi calculus with constructs for accessing a global, shared memory together with an encoding of this calculus in labelled msr rules which enables automated verification using the tamarin prover as a backend. Our prototype verification tool, automating this translation, has been successfully used to analyze several case studies. As future work we plan to increase the degree of automation of the tool by automatically generating helping lemmas. To achieve this goal we can exploit the fact that we generate the msr rules, and hence control their form. We also plan to use the tool for more complex case studies, specifically contract signing protocols.

Acknowledgments. This work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 645865-SPOOC), and from the German Federal Ministry of Education and Research (BMBF) within EC SPRIDE.

References

[1] Martín Abadi and Véronique Cortier. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science*, 387(1-2):2–32, 2006.

[2] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Proc. 28th ACM Symp. on Principles of Programming Languages (POPL'01)*, pages 104–115. ACM Press, 2001.

[3] Myrto Arapinis, Eike Ritter, and Mark Ryan. Statverif: Verification of stateful processes. In *Proc. 24th IEEE Computer Security Foundations Symposium (CSF'11)*, pages 33–47. IEEE Press, 2011.

[4] Alessandro Armando, David A. Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Paul Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Michaël Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In *Proc. 17th International Conference on Computer Aided Verification (CAV'05)*, LNCS, pages 281–285. Springer, 2005.

[5] Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuellar, and Llanos Tobarra Abad. Formal analysis of saml 2.0 web browser single sign-on: Breaking the saml-based single sign-on for google apps. In *Proc. 6th ACM Workshop on Formal Methods in Security Engineering (FMSE'08)*, pages 1–10, 2008.

[6] Stefano Bistarelli, Iliano Cervesato, Gabriele Lenzini, and Fabio Martinelli. Relating multiset rewriting and process algebras for security protocol analysis. *Journal of Computer Security*, 13(1):3–47, 2005.

[7] Bruno Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proc. 14th Computer Security Foundations Workshop (CSFW'01)*, pages 82–96. IEEE Press, 2001.

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

[9] M. Bond and R. Anderson. API level attacks on embedded systems. *IEEE Computer Magazine*, pages 67–75, October 2001.

[10] Matteo Bortolozzo, Matteo Centenaro, Riccardo Focardi, and Graham Steel. Attacking and fixing PKCS#11 security tokens. In *Proc. 17th ACM Conference on Computer and Communications Security (CCS'10)*, pages 260–269. ACM Press, 2010.

[11] CCA Basic Services Reference and Guide. *CCA Basic Services Reference and Guide*, October 2006. Available online.
[12] Stéphanie Delaune, Steve Kremer, and Graham Steel. Formal analysis of PKCS#11 and proprietary extensions. *Journal of Computer Security*, 18(6):1211–1245, November 2010. doi: 10.3233/JCS-2009-0394. URL http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/DKS-jcs09.pdf.
[13] Stéphanie Delaune, Steve Kremer, Mark D. Ryan, and Graham Steel. Formal analysis of protocols based on TPM state registers. In *Proc. 24th IEEE Computer Security Foundations Symposium (CSF'11)*, pages 66–82. IEEE Press, 2011.
[14] Santiago Escobar, Catherine Meadows, and José Meseguer. Maude-npa: Cryptographic protocol analysis modulo equational properties. In *Foundations of Security Analysis and Design V*, volume 5705 of *LNCS*, pages 1–50. Springer, 2009.
[15] Sibylle B. Fröschle and Nils Sommer. Reasoning with past to prove PKCS#11 keys secure. In *Proc. 7th International Workshop on Formal Aspects in Security and Trust (FAST'10)*, volume 6561 of *LNCS*, pages 96–110, 2010.
[16] Juan A. Garay, Markus Jakobsson, and Philip D. MacKenzie. Abuse-free optimistic contract signing. In *Advances in Cryptology—Crypto'99*, volume 1666 of *LNCS*, pages 449–466. Springer, 1999.
[17] Joshua D. Guttman. State and progress in strand spaces: Proving fair exchange. *J. Autom. Reasoning*, 48(2):159–195, 2012.
[18] Jonathan Herzog. Applying protocol analysis to security device interfaces. *IEEE Security & Privacy Magazine*, 4(4): 84–87, July-Aug 2006.
[19] Steve Kremer and Robert Künnemann. Automated analysis of security protocols with global state. In *Proc. 35th IEEE Symposium on Security and Privacy (S&P'14)*, pages 163–178. IEEE Computer Society Press, 2014. doi: 10.1109/SP.2014.18.
[20] Robert Künnemann. Automated backward analysis of PKCS#11 v2.20. In *Proc. 4th Conference on Principles of Security and Trust (POST'15)*, volume 9036 of *LNCS*, pages 219–238. Springer, 2015.
[21] Robert Künnemann and Graham Steel. YubiSecure? Formal security analysis results for the Yubikey and YubiHSM. In *Proc. 8th Workshop on Security and Trust Management (STM'12)*, volume 7783 of *LNCS*, pages 257–272, 2012.
[22] Dennis Longley and Simon Rigby. An automatic search for security flaws in key management schemes. *Computers and Security*, 11(1):75–89, March 1992.
[23] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. 2nd International Workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAS'96)*, volume 1055 of *LNCS*, pages 147–166. Springer, 1996.
[24] Sebastian Mödersheim. Abstraction by set-membership: verifying security protocols and web services with databases. In *Proc. 17th ACM Conference on Computer and Communications Security (CCS'10)*, pages 351–360. ACM, 2010.
[25] Itsaka Rakotonirina. Vérification automatique de protocoles de sécurité avec mémoire globale et boucles. Intership report, September 2014. URL http://www.dptinfo.ens-cachan.fr/~irakoton/stagel3/rapportl3.pdf.
[26] John D. Ramsdell, Daniel J. Dougherty, Joshua D. Guttman, and Paul D. Rowe. A hybrid analysis for security protocols with state. In *Proc. 11th International Conference on Integrated Formal Methods (IFM'14)*, volume 8739 of *LNCS*, pages 272–287. Springer, 2014. doi: 10.1007/978-3-319-10181-1.
[27] *PKCS #11: Cryptographic Token Interface Standard.* RSA Security Inc., v2.20, June 2004.
[28] Benedikt Schmidt. *Formal Analysis of Key-Exchange Protocols and Physical Protocols*. PhD thesis, ETH Zürich, November 2012.
[29] Benedikt Schmidt, Simon Meier, Cas Cremers, and David Basin. Automated analysis of Diffie-Hellman protocols and advanced security properties. In *Proc. 25th IEEE Computer Security Foundations Symposium (CSF'12)*, pages 78–94. IEEE Press, 2012.
[30] Benedikt Schmidt, Simon Meier, Cas Cremers, and David Basin. The tamarin prover for the symbolic analysis of security protocols. In *Proc. 25th International Conference on Computer Aided Verification (CAV'13)*, volume 8044 of *LNCS*, pages 696–701. Springer, 2013.
[31] Benedikt Schmidt, Ralf Sasse, Cas Cremers, and David A. Basin. Automated verification of group key agreement protocols. In *Proc. 35th IEEE Symposium on Security and Privacy (S&P'14)*, pages 179–194. IEEE Computer Society Press, 2014. doi: 10.1109/SP.2014.19.
[32] Jianxiong Shao, Yu Qin, Dengguo Feng, and Weijin Wang. Formal analysis of enhanced authorization in the TPM 2.0. In *Proc. 10th ACM Symposium on Information, Computer and Communications Security (ASIA CCS '15)*, pages 273–284. ACM, 2015.
[33] F. Javier Thayer Fabrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2/3):191–230, 1999.
[34] Trusted Computing Group. TPM Specification version 1.2. Parts 1–3, revision 103, 2007. Available at http://www.trustedcomputinggroup.org/resources/tpm_main_specification.
[35] *The YubiKey Manual - Usage, configuration and introduction of basic concepts (Version 2.2), available at: http://www.yubico.com/documentation.* Yubico AB, Kungsgatan 37, 111 56 Stockholm Sweden, June 2010.
[36] Yubico AB. Yubico customer list, 2014. URL https://www.yubico.com/about/reference-customers/. Accessed: Do 13 Nov 2014 08:33:34 CET.

Appendix

A. Correctness of tamarin's solution procedure for translated rules

The multiset rewrite system produced by our translation for a well-formed process P could actually contain rewrite rules that are not valid with respect to Definition 4, because they violate the third condition, which is: for each $l' \ -[\ a'\ ]\to\ r' \in R \in_E \mathit{ginsts}(l\ -[\ a\ ]\to\ r)$ we have that $\cap_{r''=_E r'} \mathit{names}(r'') \cap \mathit{FN} \subseteq \cap_{l''=_E l'} \mathit{names}(l'') \cap \mathit{FN}$.

This does not hold for rules in $[\![P]\!]_{=p}$ where p is the position of the lookup-operator. The right hand-side of this rule can be instantiated such that, assuming the variable bound by the lookup is named v, this variable v is substituted by a names that does not appear on the left-hand side. In the following, we will show that the results from [29] still hold. In practice, this means that the tamarin-prover can be used for verification, despite the fact that it outputs well-formedness errors for each rule that is a translation of a lock.

First we give some intuition. The third condition assures that rules do not introduce new names on the right-hand side of rules, i. e., all new names originate from the FRESH rule. While the condition is indeed violated by the appearance of v, v cannot be instantiated to a new name, since α_{in} guarantees that each value v retrieved in a lookup has appeared in an insert before. We show that one could as well introduce a dummy fact $!\mathsf{Dum}(v)$ at the left-hand side of the rewrite rule corresponding to the lookup, and consequently at the right-hand side of the rewrite rule corresponding to the previous insert, which *must exist* in each trace satisfying α_{in}. This modified translation would not violate the third condition anymore, but as these dummy facts constitute overhead and rather a proof argument than a real necessity, we will use the modified translation to show that the actual translation is correct despite the warning.

We will introduce some notation first. We re-define $[\![P]\!]$ to contain the INIT rule and $[\![\overline{P}, [], []]\!]$, but not MD (which is different to Definition 14). We furthermore define a translation with dummy-facts, denoted $[\![P]\!]^D$, that contains INIT and $[\![\overline{P}, [], []]\!]^D$, which is defined as follows:

Definition 18. *We define* $[\![P]\!]^D := \text{INIT} \cup [\![\overline{P}, [], []]\!]^D$*, where* $[\![\overline{P}, [], []]\!]^D$ *is defined just as* $[\![\overline{P}, [], []]\!]$*, with the exception of two cases,* $P =$ *lookup* M *as* v *in* P *else* Q *and* $P =$ *insert* $s, t; P$*, where it is defined as follows:*

$$
\begin{aligned}
[\![\textit{lookup } M \textit{ as } v \textit{ in } P \textit{ else } Q, p, \tilde{x}]\!]^D = \{&[\mathsf{state}_p(\tilde{x}), !\mathsf{Dum}(v)]\ -[\ \mathrm{IsIn}(M, v)\ ]\to [\mathsf{state}_{p\cdot 1}(\tilde{M}, v)],\\
&[\mathsf{state}_p(\tilde{x})]\ -[\ \mathrm{IsNotSet}(M)\ ]\to [\mathsf{state}_{p\cdot 2}(\tilde{x})]\}\\
&\cup [\![P, p\cdot\ 1, (\tilde{x}, v)]\!]^D \cup [\![Q, p\cdot 2, \tilde{x}]\!]^D\\
[\![\textit{insert } s, t; P, p, \tilde{x}]\!]^D = \{\ &[\mathsf{state}_p(\tilde{x})]\ -[\ \mathrm{Insert}(s, t)\ ]\to [\mathsf{state}_{p\cdot 1}(\tilde{x}), !\mathsf{Dum}(t)]\ \}\\
&\cup [\![P, p\cdot 1, \tilde{x}]\!]^D
\end{aligned}
$$

The only difference between $[\![P]\!]$ and $[\![P]\!]^D$ is therefore, that $[\![P]\!]^D$ produces a permanent fact !Dum for every value v that appears in an action $\mathrm{insert}(k, v)$, which is a premise to every rule instance with an action $\mathrm{IsIn}(k', v)$. We see that $[\![P]\!]^D$ contains now only valid multiset rewrite rules.

In the following, we would like to show that the tamarin-prover's solution algorithm is correct for $[\![P]\!]$. To this end, we make use of the proof of correctness of tamarin as presented in Benedikt Schmidt's

Ph.D. thesis [28]. We will refer to Lemmas, Theorems and Corollaries in this work by their numbers. We will use the notation of this work, to make it easier to the reader to compare our statements against the statements there. In particular, $\overline{trace(execs(R))}$ is $traces^{msr}(R)$ in our notation.

Again, we give some intuition first. The crucial step in the correctness proof is Lemma A.12, as the third condition is used only once in the overall correctness proof, namely in Lemma A.14 (which is used by Lemma A.12). In this step the switch is made from dependency graphs to normalized dependency graphs, which enforce normalized message deduction, e. g., that messages are not deduced twice, that a message deduced is not unnecessarily deconstructed, and that rule instances are normal with respect a rewriting system that simplifies message deduction for Diffie-Hellman groups ($\mathcal{RDH}_e$). Normalized dependency graphs are much more constrained than dependency graphs and thus necessary for efficient backward analysis. Lemma A.12 shows that normal dependency graphs have the same observable traces as traces generated using multiset rewriting. For this step it is necessary to show that all accessible factors (or their inverses) that appear in the normalized dependency graph are known to the adversary (Lemma A.14). In general, if the third condition is not met, a new factor could appear 'out of nowhere', when the right-hand side of the rule is instantiated. However, as Lemma A.14 holds for the dummy translation, it holds for the actual translation, too, given a trace that satisfies α_{in}. This will be detailed below, but first we recall the overall proof structure.

Formally, we have to show that:

Lemma 2. *For all well-formed process P and guarded trace properties ϕ,*

$$trace(execs([\![P]\!] \cup \mathrm{MD}) \vDash_{\mathcal{DH}_e} \neg\alpha_{in} \vee \phi$$

if and only if

$$trace(ndgraphs([\![P]\!])) \vDash_{ACC} \neg\alpha_{in} \vee \phi.$$

Proof. The proof proceeds similar to the proof to Theorem 3.27. We refer to results in [28], whenever their proofs apply despite the fact that the rules in $[\![P]\!]$ do not satisfy the third condition of multiset rewrite rules.

$$\begin{aligned}
& trace(execs([\![P]\!] \cup \mathrm{MD}) \vDash_{\mathcal{DH}_e} \neg\alpha_{in} \vee \phi \\
\Leftrightarrow\ & \overline{trace(execs([\![P]\!] \cup \mathrm{MD})} \vDash_{\mathcal{DH}_e} \neg\alpha_{in} \vee \phi && \text{(Lemma 3.7 (unaltered))} \\
\Leftrightarrow\ & \overline{trace(execs([\![P]\!] \cup \mathrm{MD}))} \downarrow_{\mathcal{RDH}_e} \vDash_{\mathcal{DH}_e} \neg\alpha_{in} \vee \phi && \text{(Definition of } \vDash_{\mathcal{DH}_e}\text{)} \\
\Leftrightarrow\ & \overline{trace(dgraphs_{\mathcal{DH}_e}([\![P]\!] \cup \mathrm{MD}))} \downarrow_{\mathcal{RDH}_e} \vDash_{\mathcal{DH}_e} \neg\alpha_{in} \vee \phi && \text{(Lemma 3.10 (unaltered))} \\
\Leftrightarrow\ & \overline{trace(\{dg \mid dg \in dgraphs_{\mathcal{ACC}}(\lceil [\![P]\!] \cup \mathrm{MD} \rceil^{\mathcal{RDH}_e}_{insts}) } \\
& \overline{\wedge dg \downarrow_{\mathcal{RDH}_e} \text{-normal}\})} \vDash_{\mathcal{DH}_e} \neg\alpha_{in} \vee \phi && \text{(Lemma 3.11 (unaltered))} \\
\Leftrightarrow\ & \overline{trace(ndgraphs([\![P]\!]))} \vDash_{\mathcal{DH}_e} \neg\alpha_{in} \vee \phi && \text{(Lemma A.12 (*))} \\
\Leftrightarrow\ & trace(ndgraphs([\![P]\!])) \vDash_{\mathcal{ACC}} \neg\alpha_{in} \vee \phi && \text{(Lemma 3.7 and A.20(both unaltered))}
\end{aligned}$$

It is only in Lemma A.12 where the third condition is used: The proof to this lemma applies Lemma A.14, which says that all factors (or their inverses) are known to the adversary. We will quote Lemma A.14 here:

Lemma 3 (Lemma A.14 in [28]). *For all $ndg \in ndgraphs(P)$, conclusions (i, u) in ndg with conclusion fact f and terms $t \in afactors(f)$, there is a conclusion (j, v) in ndg with $j < i$ and conclusion fact $\mathsf{K}^d(m)$ such that $m \in_{ACC} \{ t, (t^{-1}) \downarrow_{\mathcal{RBP}_e} \}$.*

If there is $ndg \in ndgraphs([\![P]\!])$, such that $trace(ndg) \vDash_{ACC} \alpha_{in}$, then

$$\begin{aligned} & trace(ndgraphs([\![P]\!])) \vDash_{\mathcal{ACC}} \neg\alpha_{in} \vee \phi \\ \Leftrightarrow & \forall ndg \in ndgraphs([\![P]\!]) \text{ s.t. } trace(ndg) \vDash_{ACC} \alpha_{in} \\ & \quad trace(ndg) \vDash_{\mathcal{ACC}} \phi \end{aligned}$$

Since for the empty trace, $[] \vDash_{ACC} \alpha_{in}$, we only have to show that Lemma A.14 holds for $ndg \in ndgraphs([\![P]\!])$, such that $trace(ndg) \vDash_{ACC} \alpha_{in}$.

For every $ndg \in ndgraphs([\![P]\!])$, such that $trace(ndg) \vDash_{ACC} \alpha_{in}$, there is a trace equivalent $ndg' \in ndgraphs([\![P]\!]^D)$, since the only difference between $[\![P]\!]$ and $[\![P]\!]^D$ lies in the dummy conclusion and premises, and α_{in} requires that any v in an action $\mathrm{IsIn(u, v)}$ appeared previously in an action $\mathrm{Insert(u, v)}$ (equivalence modulo ACC). Therefore, ndg' has the same K^d-conclusions ndg has, and every conclusion in ndg is a conclusion in ndg'.

We have that Lemma A.14 holds for $[\![P]\!]^D$, since all rules generated in this translation are valid multiset rewrite rules. Therefore, Lemma A.14 holds for all $ndg \in ndgraphs([\![P]\!])$, such that $trace(ndg) \vDash_{ACC} \alpha_{in}$, too, concluding the proof by showing the marked (*) step.

□

B. Proof of Lemma 1

We will first show a few useful preliminary lemmas. Then we will show each of the two directions of Lemma 1.

B.1. Preliminary definitions and lemmas

In order to prove Lemma 1, we need a few additional lemmas.

We say that a set of traces Tr is prefix closed if for all $tr \in Tr$ and for all tr' which is a prefix of tr we have that $tr' \in Tr$.

Lemma 4 ($filter$ is prefix-closed). *Let Tr be a set of traces. If Tr is prefix closed then $filter(Tr)$ is prefix closed as well.*

Proof. It is sufficient to show that for any trace $tr = tr' \cdot a$ we have that if $\forall\theta.(tr, \theta) \vDash \alpha$ then $\forall\theta.(tr', \theta) \vDash \alpha$. This can be shown by inspecting each of the conjuncts of α. □

We next show that the translation with dummy facts defined in Definition 18 produces the same executions as $[\![P]\!]$, excluding executions not consistent with the axioms. For this we define the function d which removes any dummy fact from an execution, i.e.,

$$d(\emptyset \xrightarrow{F_1} S_1 \xrightarrow{F_2} \dots \xrightarrow{F_n} S_n) = \emptyset \xrightarrow{F_1} S_1' \xrightarrow{F_2} \dots \xrightarrow{F_n} S_n'$$

where $S_i' = S_i \setminus^{\#} \cup_{t \in \mathcal{T}} !\mathsf{Dum}(t)$. To state this property we lift the function *filter* (Definition 16) from traces to executions, i.e. for a set of msr executions Ex we define

$$\mathit{filter}(Ex) = \{\emptyset \xrightarrow{F_1} S_1 \xrightarrow{F_2} \dots \xrightarrow{F_n} S_n \in Ex \mid [F_1, \dots, F_n] \Vdash \alpha\}$$

where α is defined in Definition 15.

Lemma 5. *Given a ground process P, we have that*

$$\mathit{filter}(\mathit{exec}^{msr}([\![P]\!])) = \mathit{filter}(d(\mathit{exec}^{msr}([\![P]\!]^D \cup \mathrm{MD})))$$

Proof. The only rules in $[\![P]\!]^D$ that differ from $[\![P]\!]$ are translations of insert and lookup. The first one only adds a permanent fact, which by the definition of d, is removed when applying d. The second one requires a fact $!\mathsf{Dum}(t)$, whenever the rule is instantiated such the actions equals $\mathsf{IsIn}(s,t)$ for some s. Since the translation is otherwise the same, we have that

$$\mathit{filter}(d(\mathit{exec}^{msr}([\![P]\!]^D \cup \mathrm{MD}))) \subseteq \mathit{filter}(\mathit{exec}^{msr}([\![P]\!]))$$

For any execution in $\mathit{filter}(d(\mathit{exec}^{msr}([\![p]\!] \cup \mathrm{MD})))$ and any action $\mathsf{IsIn}(s,t)$ in this execution, there is an earlier action $\mathsf{Insert}(s',t')$ such that $s = s'$ and $t = t'$, as otherwise α_{in} would not hold. Therefore the same execution is part of $\mathit{filter}(d(\mathit{exec}^{msr}([\![p]\!]^D \cup \mathrm{MD})))$, as this means that whenever $!\mathsf{Dum}(t)$ is in the premise, $!\mathsf{Dum}(t')$ for $t = t'$ has previously appeared in the conclusion. Since it is a permanent fact, it has not disappeared and therefore

$$\mathit{filter}(d(\mathit{exec}^{msr}([\![P]\!]^D \cup \mathrm{MD}))) \supseteq \mathit{filter}(\mathit{exec}^{msr}([\![P]\!]))$$

□

Lemma 6. *Let P be a ground process and $\emptyset \xrightarrow{F_1} S_1 \xrightarrow{F_2} \dots \xrightarrow{F_n} S_n \in \mathit{filter}(\mathit{exec}^{msr}([\![P]\!]))$. For all $1 \leq i \leq n$, if $\mathsf{Fr}(a) \in S_i$ and $F(t_1, \dots, t_k) \in S_i$ for any $F \in \Sigma_{fact} \setminus \{\mathsf{Fr}\}$, then $a \notin \cap_{t =_E t'} \mathit{names}(t')$, for any $t \in \{t_1, \dots, t_k\}$.*

Proof. The translation with the dummy fact introduced in Appendix A will make this proof easier as for $[\![P]\!]^D \cup \mathrm{MD}$, we have that the third condition of Definition 4 holds, namely,

$$\forall l' -[\, a' \,]\to r' \in_E \mathit{ginsts}(l -[\, a \,]\to r) : \cap_{r'' =_E r'} \mathit{names}(r'') \cap FN \subseteq \cap_{l'' =_E l'} \mathit{names}(l'') \cap FN \quad (1)$$

We will show that the statement holds for all $\emptyset \xrightarrow{F_1} S_1 \xrightarrow{F_2} \dots \xrightarrow{F_n} S_n \in \mathit{filter}(\mathit{exec}^{msr}([\![P]\!]^D \cup \mathrm{MD}))$, which implies the claim by Lemma 5. We proceed by induction on n, the length of the execution.

– Base case, $n = 0$. We have that $S_0 = \emptyset$ and therefore the statement holds trivially.
– Inductive case, $n \geq 1$. We distinguish two cases.

1. A rule that is not FRESH was applied and there is a fact $F(t_1, \ldots, t_k) \in S_n$, such that $F(t_1, \ldots, t_k) \notin S_{n-1}$, and $\mathsf{Fr}(a) \in S_n$ such that $a \in \cap_{t_i =_E t'} names(t')$ for some t_i. (If there are no such $F(t_1, \ldots, t_k)$ and $\mathsf{Fr}(a)$ we immediately conclude by induction hypothesis.) By Equation 1, $a \in t'_j$ for some $F'(t'_1, \ldots, t'_l) \in S_{n-1}$. Since FRESH is the only rule that adds a Fr-fact and $\mathsf{Fr}(a) \in S_n$, it must be that $\mathsf{Fr}(a) \in S_{n-1}$, contradicting the induction hypothesis. Therefore this case is not possible.
2. The rule FRESH was applied, i. e., $\mathsf{Fr}(a) \in S_n$ and $\mathsf{Fr}(a) \notin S_{n-1}$. If there is no $a \in \cap_{t_i =_E t'} names(t')$ for some t_i, and $F(t_1, \ldots, t_k) \in S_n$, then we conclude by induction hypothesis. Otherwise, if there is such a $F(t_1, \ldots, t_k) \in S_n$, then, by Equation 1, $a \in t'_j$ for some $F'(t'_1, \ldots, t'_l) \in S_i$ for $i < n$. We construct a contradiction to the induction hypothesis by taking the prefix of the execution up to i and appending the instantiation of the FRESH rule to its end. Since $d(exec^{msr}(\llbracket P \rrbracket^D \cup \mathrm{MD}))$ is prefix closed by Lemma 4 we have that $\emptyset \xrightarrow{F_1} S'_1 \xrightarrow{F_2} \ldots \xrightarrow{F_i} S_i \in \mathit{filter}(d(exec^{msr}(\llbracket P \rrbracket^D \cup \mathrm{MD})))$. Moreover as rule FRESH was applied adding $\mathsf{Fr}(a) \in S_n$ it is also possible to apply the same instance of FRESH to the prefix (by Definition 6) and therefore

$$\emptyset \xrightarrow{F_1} S'_1 \xrightarrow{F_2} \ldots \xrightarrow{F_i} S_i \longrightarrow S_i \cup \{\, \mathsf{Fr}(a) \,\} \in \mathit{filter}(d(exec^{msr}(\llbracket P \rrbracket^D \cup \mathrm{MD})))$$

contradicting the induction hypothesis.

□

Lemma 7. *For any frame $\nu\tilde{n}.\sigma$, $t \in \mathcal{M}$ and $a \in FN$, if $a \notin st(t)$, $a \notin st(\sigma)$ and $\nu\tilde{n}.\sigma \vdash t$, then $\nu\tilde{n}, a.\sigma \vdash t$.*

Proof. In [1, Proposition 1] it is shown that $\nu\tilde{n}.\sigma \vdash t$ if and only if $\exists M. fn(M) \cap \tilde{n} = \emptyset$ and $M\sigma =_E t$. Define M' as M renaming a to some fresh name, i.e., not appearing in $\tilde{n}, \sigma, t$. As $a \notin st(\sigma, t)$ and the fact that equational theories are closed under bijective renaming of names we have that $M'\sigma =_E t$ and $fn(M') \cap (\tilde{n}, a) = \emptyset$. Hence $\nu\tilde{n}, a.\sigma \vdash t$. □

Lemma 8. *Let P be a ground process and $\emptyset \xrightarrow{F_1} S_1 \xrightarrow{F_2} \ldots \xrightarrow{F_n} S_n \in \mathit{filter}(exec^{msr}(\llbracket P \rrbracket))$. Let*

$$\tilde{n} = \{a : \mathit{fresh} \mid \mathit{ProtoNonce}(a) \in \bigcup_{1 \leq j \leq n} F_j\},$$

$$\{t_1, \ldots, t_m\} = \{t \mid \mathsf{Out}(t) \in_{1 \leq j \leq n} S_j\}.$$

Let $\sigma = \{^{t_1}/_{x_1}, \ldots, ^{t_m}/_{x_m}\}$. We have that

1. *if $!\mathsf{K}(t) \in S_n$ then $\nu\tilde{n}.\sigma \vdash t$;*
2. *if $\nu\tilde{n}.\sigma \vdash t$ then there exists S such that*
   – $\emptyset \xrightarrow{F_1} S_1 \xrightarrow{F_2} \ldots \xrightarrow{F_n} S_n \longrightarrow^* S \in \mathit{filter}(exec^{msr}_E(\llbracket P \rrbracket))$,
   – $!\mathsf{K}(t) \in_E S$ *and*

– $S_n \to_R^* S$ *for* $R = \{\,\text{MDOUT}, \text{MDPUB}, \text{MDFRESH}, \text{MDAPPL}, \text{FRESH}\,\}$.

Proof. We prove both items separately.

1. The proof proceeds by induction on n, the number of steps of the execution.

   Base case: n=0. This case trivially holds as $S_n = \emptyset$.

   Inductive case: n>0. By induction we suppose that if $!\mathsf{K}(t) \in S_{n-1}$ then $\nu\tilde{n}'.\sigma' \vdash t$ where $\tilde{n}', \sigma'$ are defined in a similar way as $\tilde{n}, \sigma$ but for the execution of size $n-1$. We proceed by case analysis on the rule used to extend the execution.

   – MDOUT. Suppose that $\mathsf{Out}(u) \dashv[\quad]\to !\mathsf{K}(u) \in \mathit{ginsts}(\text{MDOUT})$ is the rule used to extend the execution. Hence $\mathsf{Out}(u) \in S_{n-1}$ and by definition of σ there exists x such that $x\sigma = u$. We can apply deduction rule DFRAME and conclude that $\nu\tilde{n}.\sigma \vdash u$. If $!\mathsf{K}(t) \in S_n$ and $t \neq u$ we conclude by induction hypothesis as $\tilde{n} = \tilde{n}', \sigma = \sigma'$.
   – MDPUB. Suppose that $\dashv[\quad]\to \mathsf{K}(a : pub) \in \mathit{ginsts}(\text{MDPUB})$ is the rule used to extend the execution. As names of sort pub are never added to $\tilde{n}$ we can apply deduction rule DNAME and conclude that $\nu\tilde{n}.\sigma \vdash a$. If $K(t) \in S_n$ and $t \neq a$ we conclude by induction hypothesis as $\tilde{n} = \tilde{n}', \sigma = \sigma'$.
   – MDFRESH. Suppose that $\mathsf{Fr}(a : \mathit{fresh}) \dashv[\quad]\to \mathsf{K}(a : \mathit{fresh}) \in \mathit{ginsts}(\text{MDFRESH})$ is the rule used to extend the execution. By definition of an execution we have that $\mathsf{Fr}(a : \mathit{fresh}) \neq (S_{j+1} \setminus S_j)$ for any $j \neq n-1$. Hence $n \notin \tilde{n}$. We can apply deduction rule DNAME and conclude that $\nu\tilde{n}.\sigma \vdash a$. If $!\mathsf{K}(t) \in S_n$ and $t \neq a$ we conclude by induction hypothesis as $\tilde{n} = \tilde{n}', \sigma = \sigma'$.
   – MDAPPL. Suppose that $!\mathsf{K}(t_1), \ldots, !\mathsf{K}(t_k) \dashv[\quad]\to !\mathsf{K}(u) \in \mathit{ginsts}(\text{MDAPPL})$ is the rule used to extend the execution. We have that $K(t_1), \ldots, K(t_k) \in S_{n-1}$ and $u =_E f(t_1, \ldots, t_k)$. By induction hypothesis, $\nu\tilde{n}'.\sigma' \vdash t_i$ for $1 \leq i \leq k$. As $\tilde{n} = \tilde{n}', \sigma = \sigma'$ we have that $\nu\tilde{n}.\sigma \vdash t_i$ for $1 \leq i \leq k$. We can apply deduction rule DAPPL and conclude that $\nu\tilde{n}.\sigma \vdash f(t_1, \ldots, t_k)$. Hence, $\nu\tilde{n}.\sigma \vdash u$ by rule DEQ. If $K(t) \in S_n$ and $t \neq f(t_1, \ldots, t_k)$ we conclude by induction hypothesis as $\tilde{n} = \tilde{n}', \sigma = \sigma'$.
   – If $S_{n-1} \xrightarrow{\mathit{ProtoNonce}(a)} S_n$ we have that $\mathsf{Fr}(a) \in S_{n-1}$. By Lemma 6, we obtain that if $!\mathsf{K}(t) \in S_{n-1}$ then there exist t' and σ'' such that $t' =_E t$, $\sigma'' =_E \sigma'$ and $a \notin \mathit{st}(t')$ and $a \notin \mathit{st}(\sigma'')$. For each $!\mathsf{K}(u) \in S_n$ there is $!\mathsf{K}(u) \in S_{n-1}$, and by induction hypothesis, $\nu\tilde{n}'.\sigma' \vdash u$. By Lemma 7 and the fact that $\sigma'' =_E \sigma'$ we have that $\nu\tilde{n}', a.\sigma' \vdash u$. As $\tilde{n}', a = \tilde{n}$ and $\sigma' = \sigma$ we conclude.
   – All other rules do neither add $!\mathsf{K}(\ )$-facts nor do they change $\tilde{n}$ and may only extend σ. Therefore we conclude by the induction hypothesis.

2. Suppose that $\nu\tilde{n}.\sigma \vdash t$. We proceed by induction on the proof tree witnessing $\nu\tilde{n}.\sigma \vdash t$.

   Base case. The proof tree consists of a single node. In this case one of the deduction rules DNAME or DFRAME has been applied.

   – DNAME. We have that $t \notin \tilde{n}$. If $t \in PN$ we use rule MDPUB and we have that $S_n \to S = S_n \cup \{!\mathsf{K}(t)\}$. In case $t \in FN$ we need to consider 3 different cases: *(i)* $!\mathsf{K}(t) \in S_n$ and we immediately conclude (by letting $S = S_n$), *(ii)* $\mathsf{Fr}(t) \in S_n$ and applying rule MDFRESH we have that $S_n \to S = S_n \cup \{!\mathsf{K}(t)\}$, *(iii)* $\mathsf{Fr}(t) \notin S_n$. By inspection of the rules we see that $\mathsf{Fr}(t) \notin S_i$ for any $1 \leq i \leq n$: the only rules that could remove $\mathsf{Fr}(t)$ are MDFRESH which

would have created the persistent fact $!\mathsf{K}(t)$, or the $ProtoNonce$ rules which would however have added t to $\tilde{n}$. Hence, applying successively rules FRESH and MDFRESH yields a valid extension of the execution $S_n \to S_n \cup \{\mathsf{Fr}(t)\} \to S = S_n \cup \{!\mathsf{K}(t)\}$.
– DFRAME. We have that $x\sigma = t$ for some $x \in \mathbf{D}(\sigma)$, that is, $t \in \{t_1, \ldots, t_m\}$. By definition of $\{t_1, \ldots, t_m\}$, $\mathsf{Out}(t) \in S_i$ for some $i \leq n$. If $\mathsf{Out}(t) \in S_n$ we have that $S_n \to S = (S_n \setminus \{\mathsf{Out}(t)\}) \cup \{!\mathsf{K}(t)\}$ applying rule MDOUT. If $\mathsf{Out}(u) \notin S_n$, the fact that the only rule in $[\![P]\!]$ that allows to remove an Out-fact is MDOUT, suggests that it was applied before, and thus $!\mathsf{K}(u) \in S$.

Inductive case. We proceed by case distinction on the last deduction rule which was applied.

– DAPPL. In this case $t = f(t_1, \ldots, t_k)$, such that $f \in \Sigma^k$ and $\nu\tilde{n}\tilde{r}.\sigma \vdash t_i$ for every $i \in \{1, \ldots, k\}$. Applying the induction hypothesis we obtain that there are k transition sequences $S_n \to_R^* S^i$ for $1 \leq i \leq k$ which extend the execution such that $t_i \in S^i$. All of them only add $!\mathsf{K}$ facts which are persistent facts. If any two of these extensions remove the same $\mathsf{Out}(t)$-fact or the same $\mathsf{Fr}(a)$-fact it also adds the persistent fact $!\mathsf{K}(t)$, respectively $!\mathsf{K}(a)$, and we simply remove the second occurrence of the transition. Therefore, applying the same rules as for the transitions $S_n \to^* S^i$ (and removing duplicate rules) we have that $S_n \to^* S'$ and $!\mathsf{K}(t_1), \ldots, !\mathsf{K}(t_k) \in S'$. Applying rule MDAPPL we conclude.
– DEQ. By induction hypothesis there exists S as required with $!\mathsf{K}(t') \in_E S$ and $t =_E t'$ which allows us to immediately conclude that $!\mathsf{K}(t) \in_E S$.

□

Lemma 9. *If $\nu\tilde{n}.\sigma \vdash t$, $\tilde{n} =_E \tilde{n}'$, $\sigma =_E \sigma'$ and $t =_E t'$, then $\nu\tilde{n}'.\sigma' \vdash t'$.*

Proof. Assume $\nu\tilde{n}.\sigma \vdash t$. Since an application of DEQ can be appended to the leafs of its proof tree, we have $\nu\tilde{n}.\sigma' \vdash t$. Since DEQ can be applied to its root, we have $\nu\tilde{n}.\sigma' \vdash t'$. Since $\tilde{n}, \tilde{n}'$ consist only of names, $\tilde{n} = \tilde{n}'$ and thus $\nu\tilde{n}'.\sigma' \vdash t'$. □

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

To state our next lemma we need two additional definitions.

Definition 19. *Let P be a well-formed ground process and p_t a position in P. We define the set of multiset rewrite rules generated for position p_t of P, denoted $[\![P]\!]_{=p_t}$ as follows:*

$$[\![P]\!]_{=p_t} := [\![P, [], []]\!]_{=p_t}$$

where $[\![\cdot, \cdot, \cdot]\!]_{=p_t}$ is defined in Figure 18.

The next definition will be useful to state that for a process P every fact of the form $\mathsf{state}_p(\tilde{t})$ in a multiset rewrite execution of $[\![P]\!]$ corresponds to an active process in the execution of P which is an instance of the subprocess $P|_p$.

Definition 20. *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 $\{\mathsf{state}_p(\tilde{t}) \mid \exists p, \tilde{t}.\ \mathsf{state}_p(\tilde{t}) \in^\# S\}^\#$ such that whenever $Q \in^\# \mathcal{P}$ is mapped to $\mathsf{state}_p(\tilde{t}) \in^\# S$ we have that*

$$
\begin{aligned}
[\![0, p, \tilde{x}]\!]_{=p_t} &= \{\, [\mathsf{state}_p(\tilde{x})] \to [] \,\}_{p \stackrel{?}{=} p_t} \\
[\![P \mid Q, p, \tilde{x}]\!]_{=p_t} &= \{\, [\mathsf{state}_p(\tilde{x})] \to [\mathsf{state}_{p\cdot 1}(\tilde{x}), \mathsf{state}_{p\cdot 2}(\tilde{x})] \,\}_{p \stackrel{?}{=} p_t} \\
&\quad \cup [\![P, p \cdot 1, \tilde{x}]\!]_{=p_t} \cup [\![Q, p \cdot 2, \tilde{x}]\!]_{=p_t} \\
[\![!P, p, \tilde{x}]\!]_{=p_t} &= \{\, [!\mathsf{state}_p(\tilde{x})] \to [\mathsf{state}_{p\cdot 1}(\tilde{x})] \,\}_{p \stackrel{?}{=} p_t} \cup [\![P, p \cdot 1, \tilde{x}]\!]_{=p_t} \\
[\![\nu a; P, p, \tilde{x}]\!]_{=p_t} &= \{\, [\mathsf{state}_p(\tilde{x}), \mathsf{Fr}(n_a : \mathit{fresh})] \multimap [\, \mathit{ProtoNonce}(n_a : \mathit{fresh}) \,]\to \\
&\quad [\mathsf{state}_{p\cdot 1}(\tilde{x}, n_a : \mathit{fresh})] \,\}_{p \stackrel{?}{=} p_t} \cup [\![P, p \cdot 1, (\tilde{x}, n_a : \mathit{fresh})]\!]_{=p_t} \\
[\![\mathsf{Out}(M, N); P, p, \tilde{x}]\!]_{=p_t} &= \{\, [\mathsf{state}_p(\tilde{x}), \mathsf{In}(M)] \multimap [\, \mathrm{InEvent}(M) \,]\to [\mathsf{Out}(N), \mathsf{state}_{p\cdot 1}(\tilde{x})], \\
&\quad [\mathsf{state}_p(\tilde{x})] \to [\mathsf{Msg}(M, N), \mathsf{state}_p^{\mathsf{semi}}(\tilde{x})], \\
&\quad [\mathsf{state}_p^{\mathsf{semi}}(\tilde{x}), \mathsf{Ack}(M, N)] \to [\mathsf{state}_{p\cdot 1}(\tilde{x})] \,\}_{p \stackrel{?}{=} p_t} \cup [\![P, p \cdot 1, \tilde{x}]\!]_{=p_t} \\
[\![\mathsf{In}(M, N); P, p, \tilde{x}]\!]_{=p_t} &= \{\, [\mathsf{state}_p(\tilde{x}), \mathsf{In}(\langle M, N \rangle)] \multimap [\, \mathrm{InEvent}(\langle M, N \rangle) \,]\to \\
&\quad [\mathsf{state}_{p\cdot 1}(\tilde{x} \cup \mathit{vars}(N))], [\mathsf{state}_p(\tilde{x}), \mathsf{Msg}(M, N)] \to \\
&\quad [\mathsf{state}_{p\cdot 1}(\tilde{x} \cup \mathit{vars}(N)), \mathsf{Ack}(M, N)] \,\}_{p \stackrel{?}{=} p_t} \\
&\quad \cup [\![P, p \cdot 1, \tilde{x} \cup \mathit{vars}(N)]\!]_{=p_t} \\
[\![\text{if } pr(M_1, \ldots, M_k) \text{ then } P \text{ else } Q, p, \tilde{x}]\!]_{=p_t} &= \{\, [\mathsf{state}_p(\tilde{x})] \multimap [\, \mathsf{Pred}_{pr}(M_1, \ldots, M_k) \,]\to [\mathsf{state}_{p\cdot 1}(\tilde{x})], \\
&\quad [\mathsf{state}_p(\tilde{x})] \multimap [\, \mathsf{Pred\_not}_{pr}(M_1, \ldots, M_k) \,]\to [\mathsf{state}_{p\cdot 2}(\tilde{x})] \,\}_{p \stackrel{?}{=} p_t} \\
&\quad \cup [\![P, p \cdot 1, \tilde{x}]\!]_{=p_t} \cup [\![Q, p \cdot 2, \tilde{x}]\!]_{=p_t} \\
[\![\text{event } F; P, p, \tilde{x}]\!]_{=p_t} &= \{\, [\mathsf{state}_p(\tilde{x})] \multimap [\, \mathrm{Event}(), F \,]\to [\mathsf{state}_{p\cdot 1}(\tilde{x})] \,\}_{p \stackrel{?}{=} p_t} \cup [\![P, p \cdot 1, \tilde{x}]\!]_{=p_t} \\
[\![\text{insert } s, t; P, p, \tilde{x}]\!]_{=p_t} &= \{\, [\mathsf{state}_p(\tilde{x})] \multimap [\, \mathrm{Insert}(s, t) \,]\to [\mathsf{state}_{p\cdot 1}(\tilde{x})] \,\}_{p \stackrel{?}{=} p_t} \cup [\![P, p \cdot 1, \tilde{x}]\!]_{=p_t} \\
[\![\text{delete } s; P, p, \tilde{x}]\!]_{=p_t} &= \{\, [\mathsf{state}_p(\tilde{x})] \multimap [\, \mathrm{Delete}(s) \,]\to [\mathsf{state}_{p\cdot 1}(\tilde{x})] \,\}_{p \stackrel{?}{=} p_t} \cup [\![P, p \cdot 1, \tilde{x}]\!]_{=p_t} \\
[\![\text{lookup } M \text{ as } v \text{ in } P \text{ else } Q, p, \tilde{x}]\!]_{=p_t} &= \{\, [\mathsf{state}_p(\tilde{x})] \multimap [\, \mathrm{IsIn}(M, v) \,]\to [\mathsf{state}_{p\cdot 1}(\tilde{M}, v)], \\
&\quad [\mathsf{state}_p(\tilde{x})] \multimap [\, \mathrm{IsNotSet}(M) \,]\to [\mathsf{state}_{p\cdot 2}(\tilde{x})] \,\}_{p \stackrel{?}{=} p_t} \\
&\quad \cup [\![P, p \cdot 1, (\tilde{x}, v)]\!]_{=p_t} \cup [\![Q, p \cdot 2, \tilde{x}]\!]_{=p_t} \\
[\![\text{lock}^l\ s; P, p, \tilde{x}]\!]_{=p_t} &= \{\, [\mathrm{Fr}(\mathrm{lock}_l), \mathsf{state}_p(\tilde{x})] \multimap [\, \mathrm{Lock}(\mathit{lock}_l, s) \,]\to [\mathsf{state}_{p\cdot 1}(\tilde{x}, \mathit{lock}_l)] \,\}_{p \stackrel{?}{=} p_t} \\
&\quad \cup [\![P, p \cdot 1, \tilde{x}]\!]_{=p_t} \\
[\![\text{unlock}^l\ s; P, p, \tilde{x}]\!]_{=p_t} &= \{\, [\mathsf{state}_p(\tilde{x})] \multimap [\, \mathrm{Unlock}(\mathit{lock}_l, s) \,]\to [\mathsf{state}_{p\cdot 1}(\tilde{x})] \,\}_{p \stackrel{?}{=} p_t} \cup [\![P, p \cdot 1, \tilde{x}]\!]_{=p_t}
\end{aligned}
$$

Fig. 18. Definition of $[\![P, p, \tilde{x}]\!]_{=p_t}$ where $\{\cdot\}_{a \stackrel{?}{=} b} = \{\,\cdot\,\}$ if $a = b$ and $\emptyset$ otherwise.

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. $\exists ri \in_E ginsts([\![P]\!]_{=p}).\ \mathsf{state}_p(\tilde{t}) \in prems(ri)$.

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 $\mathsf{state}_p(\tilde{t})$.

Remark 2. *Note that* $\leftrightarrow_P$ *has the following properties (by the fact that it defines a bijection between multisets).*

- *If* $\mathcal{P}_1 \leftrightarrow_P S_1$ *and* $\mathcal{P}_2 \leftrightarrow_P S_2$ *then* $\mathcal{P}_1 \cup^\# \mathcal{P}_2 \leftrightarrow_P S_1 \cup^\# S_2$.
- *If* $\mathcal{P}_1 \leftrightarrow_P S_1$ *and* $Q \leftrightarrow_P \mathsf{state}_p(\tilde{t})$ *for* $Q \in \mathcal{P}_1$ *and* $\mathsf{state}_p(\tilde{t}) \in S_1$ *(i.e.* Q *and* $\mathsf{state}_p(\tilde{t})$ *are related by the bijection defined by* $\mathcal{P}_1 \leftrightarrow_P S_1$*) then* $\mathcal{P}_1 \setminus^\# \{Q\} \leftrightarrow_P S_1 \setminus^\# \{\mathsf{state}_p(\tilde{t})\}$.

We are now ready to prove the first part of Lemma 1, i. e.,

$$traces^{pi}(P) \subseteq hide(filter(traces^{msr}([\![P]\!])))$$

However, as we proceed by induction, we need to strengthen the induction hypothesis and prove the following, stronger lemma.

Lemma 10. *Let* P *be a well-formed ground process. If*

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

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

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

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. $\mathcal{L}_i =_E \{\, t \mid \exists j \le f(i), u.\ \mathrm{Lock}(u,t) \in_E F_j \wedge \forall j < k \le f(i).\mathrm{Unlock}(u,t) \notin_E F_k \,\}$
6. $[F_1, \dots, F_{n'}] \vDash \alpha$ *where* α *is defined as in Definition 15.*
7. $\exists k.\ f(i-1) < k \le f(i)$ *and* $E_i = F_k$ *and* $\cup_{f(i-1)<j\neq k\le f(i)} F_j \subseteq \mathcal{F}_{res}$

To see that this lemma indeed implies

$$traces^{pi}(P) \subseteq hide(filter(traces^{msr}([\![P]\!])))$$

note that for every trace $[E_1, \dots, E_n] \in traces^{pi}(P)$ there exists $[F_1, \dots F_{n'}] \in traces^{msr}([\![P]\!])$ such that, by Condition 6, $[F_1, \dots F_{n'}] \in filter(traces^{msr}([\![P]\!]))$ and, by Condition 7, $hide([F_1, \dots F_{n'}]) = [E_1, \dots, E_n]$.

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

Base Case. For $n = 0$, we let $f(n) = 1$ and S_1 be the multiset obtained by using the Rule INIT:

$$\emptyset \xrightarrow{\mathsf{Init}} \{\, \mathsf{state}_{[]}() \,\}^{\#}$$

Condition 1, Condition 2, Condition 4, Condition 5, Condition 6 and Condition 7 hold trivially. To show that Condition 3 holds, we have to show that $\mathcal{P}_0 \leftrightarrow_P \{\, \mathsf{state}_{[]}() \,\}^{\#}$. Note that $\mathcal{P}_0 = \{\, P \,\}^{\#}$. We choose the bijection such that $P \leftrightarrow_P \mathsf{state}_{[]}()$. For $\tau = \emptyset$ and $\rho = \emptyset$ we have that $P|_{[]}\tau = P\tau = P\rho$. By Definition 19, $[\![P]\!]_{=[]} = [\![P, [], []]\!]_{=[]}$. We see from Figure 18 that for every P we have that $\mathsf{state}_{[]}() \in \mathit{prems}([\![P, [], []]\!]_{=[]})$. Hence, we conclude that there is a ground instance $ri \in_E \mathit{ginsts}([\![P]\!]_{=[]})$ with $\mathsf{state}_{[]}() \in \mathit{prems}(ri)$.

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

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

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

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

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 $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1}, \sigma_{n-1}, \mathcal{L}_{n-1})$ to $(\mathcal{E}_n, \mathcal{S}_n, \mathcal{P}_n, \sigma_n, \mathcal{L}_n)$. 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 7 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: $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1} = \mathcal{P}' \cup \{0\}, \sigma_{n-1}, \mathcal{L}_{n-1}) \to (\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}', \sigma_{n-1}, \mathcal{L}_{n-1})$. By induction hypothesis we have that $\mathcal{P}_{n-1} \leftrightarrow_P S_{n'}$. Let p and $\tilde{t}$ be such that $0 \leftrightarrow_P \mathsf{state}_p(\tilde{t})$. By Definition 20, there is a $ri \in \mathit{ginsts}([\![P]\!]_{=p})$ such that $\mathsf{state}_p(\tilde{t})$ is part of its premise. By definition of $[\![P]\!]_{=p}$, we can choose $ri = [\mathsf{state}_p(\tilde{t})] -[\ \ ]\to []$. We can extend the previous execution by one step using ri, therefore:

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

with $S_{n'+1} = \{\, S_{f(n-1)} \setminus \{\mathsf{state}_p(\tilde{t})\,\}$. It is left to show that Conditions 1 to 7 hold for n.

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

Condition 3 holds because $\mathcal{P}' = \mathcal{P}_{n-1} \setminus^{\#} \{0\}$, $S_{f(n)} = S_{f(n-1)} \setminus^{\#} \{\, \mathsf{state}_p(\tilde{t}) \,\}^{\#}$, and $0 \leftrightarrow_P \mathsf{state}_p(\tilde{t})$ (see Remark 2).

Case: $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1} = \mathcal{P}' \cup \{Q|R\}, \sigma_{n-1}, \mathcal{L}_{n-1}) \to (\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}' \cup \{Q, R\}, \sigma_{n-1}, \mathcal{L}_{n-1})$. By induction hypothesis we have that $\mathcal{P}_{n-1} \leftrightarrow_P S_{n'}$. Let p and $\tilde{t}$ be such that $Q|R \leftrightarrow_P \mathsf{state}_p(\tilde{t})$. By Definition 20, there is a $ri \in ginsts(\llbracket P \rrbracket_{=p})$ such that $\mathsf{state}_p(\tilde{t})$ is part of its premise. By definition of $\llbracket P \rrbracket_{=p}$, we can choose $ri = [\mathsf{state}_p(\tilde{t})] \multimap [\ ] \to [\mathsf{state}_{p\cdot 1}(\tilde{t}), \mathsf{state}_{p\cdot 2}(\tilde{t})]$. We can extend the previous execution by one step using ri, therefore:

$$\emptyset \xrightarrow{F_1}_{\llbracket P \rrbracket} S_1 \xrightarrow{F_2}_{\llbracket P \rrbracket} \dots \xrightarrow{F_{n'}}_{\llbracket P \rrbracket} S_{n'} \to_{\llbracket P \rrbracket} S_{n'+1} \in exec^{msr}(\llbracket P \rrbracket)$$

with $S_{n'+1} = S_{f(n-1)} \setminus \{\, \mathsf{state}_p(\tilde{t}) \,\}^{\#} \cup \{\, \mathsf{state}_{p\cdot 1}(\tilde{t}), \mathsf{state}_{p\cdot 2}(\tilde{t}) \,\}^{\#}$. It is left to show that Conditions 1 to 7 hold for n.

Condition 1, Condition 2, Condition 4, Condition 5, Condition 6 and Condition 7 hold trivially. We now show that Condition 3 holds.

Condition 3 holds because $\mathcal{P}_n = \mathcal{P}_{n-1} \setminus^{\#} \{Q|R\} \cup^{\#} \{Q, R\}$, $\{Q\} \leftrightarrow_P \{\mathsf{state}_{p\cdot 1}(\tilde{x})\}$ and $\{R\} \leftrightarrow_P \{\mathsf{state}_{p\cdot 2}(\tilde{x})\}$ (by definition of the translation) (see Remark 2).

Case: $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1} = \mathcal{P}' \cup \{!Q\}, \sigma_{n-1}, \mathcal{L}_{n-1}) \to (\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}' \cup \{!Q, Q\}, \sigma_{n-1}, \mathcal{L}_{n-1})$. Let p and $\tilde{t}$ such that $!^i Q \leftrightarrow_P \mathsf{state}_p(\tilde{t})$. By Definition 20, there is a $ri \in ginsts(\llbracket P \rrbracket_{=p})$ such that $\mathsf{state}_p(\tilde{t})$ is part of its premise. By definition of $\llbracket P \rrbracket_{=p}$, we can choose $ri = [\mathsf{state}_p(\tilde{t})] \multimap [\ ] \to [\mathsf{state}_p(\tilde{t}), \mathsf{state}_{p\cdot 1}(\tilde{t})]$. We can extend the previous execution by 1 step using ri, therefore:

$$\emptyset \xrightarrow{F_1}_{\llbracket P \rrbracket} S_1 \xrightarrow{F_2}_{\llbracket P \rrbracket} \dots \xrightarrow{F_{n'}}_{\llbracket P \rrbracket} S_{n'} \xrightarrow{(ri)}_{\llbracket P \rrbracket} S_{n'+1} \in exec^{msr}(\llbracket P \rrbracket)$$

with $S_{n'+1} = S_{f(n)} \cup^{\#} \{\, \mathsf{state}_{p\cdot 1}(\tilde{t}) \,\}^{\#}$. 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 $\llbracket P \rrbracket_{=p}$). Condition 1, Condition 2, Condition 4, Condition 5, Condition 6 and Condition 7 hold trivially.

Case: $(\mathcal{E}_{n-1}, \mathcal{S}_{i_n-1}, \mathcal{P}_{n-1} = \mathcal{P}' \cup \{\, \nu a; Q \,\}, \sigma_{n-1}, \mathcal{L}_{n-1}) \to (\mathcal{E}_{n-1} \cup \{a'\}, \mathcal{S}_{i_n-1}, \mathcal{P}' \cup \{Q\{^{a'}/_a\}\}, \sigma_{n-1}, \mathcal{L}_{n-1})$ for a fresh a'. Let p and $\tilde{t}$ be such that $\{\nu a; Q\} \leftrightarrow_P \mathsf{state}_p(\tilde{t})$. There is a $ri \in ginsts(\llbracket P \rrbracket_{=p})$ such that $\mathsf{state}_p(\tilde{t})$ is part of its premise. By definition of $\llbracket P \rrbracket_{=p}$, there is a $ri \in ginsts(\llbracket P \rrbracket_{=p})$, $ri = [\mathsf{state}_p(\tilde{t}), \mathsf{Fr}(a' : fresh)] \multimap [\ ProtoNonce(a' : fresh)\ ] \to [\mathsf{state}_{p\cdot 1}(\tilde{t}, a' : fresh)]$. Assume there is an $i < n'$ such that $\mathsf{Fr}(a') \in S_i$. If $\mathsf{Fr}(a') \in S_n$, then we can remove the application of the instance of FRESH that added $\mathsf{Fr}(a')$ while still preserving Conditions 1 to 7. If $\mathsf{Fr}(a')$ is consumned at some point, by the definition of $\llbracket P \rrbracket$, the transition where it is consumned is annotated either $ProtoNonce(a')$ or $Lock(a', t)$ for some t. In the last case, we can apply a substitution to the execution that substitutes a by a different fresh name that never appears in $\cup_i \leq n' S_i$. The conditions we have by induction hypothesis hold on this execution, too, since $Lock \in \mathcal{F}_{res}$, and therefore Condition 7 is not affected. The first case implies that $a' \in \mathcal{E}_{n-1}$, contradicting the assumption that a' is fresh with respect to the process execution. Therefore, without loss of generality, the previous execution does not contain an $i < n'$ such that $\mathsf{Fr}(a') \in S_i$, and we can extend the previous execution by two steps using the FRESH rule and ri, therefore:

$$\emptyset \xrightarrow{F_1}_{\llbracket P \rrbracket} S_1 \xrightarrow{F_2}_{\llbracket P \rrbracket} \dots \xrightarrow{F_{n'}}_{\llbracket P \rrbracket} S_{n'} \xrightarrow{(\text{FRESH})}_{\llbracket P \rrbracket} S_{n'+1} \xrightarrow{(ri)}_{\llbracket P \rrbracket} S_{n'+2} \in exec^{msr}(\llbracket P \rrbracket)$$

with $S_{n'+1} = S_{n'} \cup^{\#} \{\, \mathsf{Fr}(a' : fresh) \,\}^{\#}$ and $S_{n'+2} = S_{f(n)} = S_{n'} \cup^{\#} \{\, \mathsf{state}_{p\cdot 1}(\tilde{t}, a' : fresh) \,\}^{\#}$. We define $f(i) := f_p(i)$ for $i < n$ and $f(n) := f(n-1) + 2$. We now show that Condition 3 holds. As

by induction hypothesis $\nu a; Q \leftrightarrow_P \mathsf{state}_{p{\cdot}1}(\tilde{t})$ we also have that $P|_p\sigma = \nu a; Q\rho$ for some σ and ρ. Extending ρ with $\{a' \mapsto a\}$ it is easy to see from definition of $[\![P]\!]_{=p}$ that $\{Q\{^{a'}/_a\}\} \leftrightarrow_P \{\mathsf{state}_{p{\cdot}1}(\tilde{t}, a')\}$. As $\mathcal{P}_n = \mathcal{P}_{n-1} \setminus^{\#} \{\nu a; Q\} \cup^{\#} \{ Q\{^{a'}/_a\} \}^{\#}$, we also immediately obtain that $\mathcal{P}_n \leftrightarrow_P S_{f(n)}$. Since a' is fresh, and therefore $\{ a' \} = \mathcal{E}_n \setminus \mathcal{E}_{n-1}$, and $F_n = \mathit{ProtoNonce}(a')$, Condition 1 holds. Condition 2, Condition 4, Condition 5, Condition 6 and Condition 7 hold trivially.

Case: $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1}, \sigma_{n-1}, \mathcal{L}_{n-1}) \xrightarrow{K(t)} (\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1}, \sigma_{n-1}, \mathcal{L}_{n-1})$. This step requires that $\nu\mathcal{E}_{n-1}.\sigma_{n-1} \vdash t$. From Lemma 8 follows that there is an execution $\emptyset \xrightarrow{F_1} S_1 \xrightarrow{F_2} \ldots \xrightarrow{F_{n'}} S_{n'} \rightarrow^* S \in \mathit{exec}^{msr}_E([\![P]\!])$ such that $!\mathsf{K}(t) \in_E S$ and $S_{n'} \rightarrow^*_R S$ for $R = \{ \text{MDOUT}, \text{MDPUB}, \text{MDFRESH}, \text{MDAPPL} \}$.

From S, we can go one further step using MDIN, since $!\mathsf{K}(t) \in S$:

$$\emptyset \xrightarrow{F_1}_{[\![P]\!]} S_1 \xrightarrow{F_2}_{[\![P]\!]} \ldots \xrightarrow{F_{n'}}_{[\![P]\!]} S_{n'} \rightarrow^*_{R \subset [\![P]\!]} S = S_{n'+s-1} \xrightarrow{K(t)}_{[\![P]\!]} S_{n'+s} \in \mathit{exec}^{msr}([\![P]\!])$$

where $S_{n'+s} = S \cup \{\mathsf{In}(t)\}$.

From the fact that $S_{f(n-1)} \rightarrow^*_R S_{f(n)} = S$, and the induction hypothesis, we can conclude that Condition 7 holds. Condition 3 holds since $\mathcal{P}_n = \mathcal{P}_{n-1}$ and no `state`-facts where neither removed nor added. Condition 1, Condition 2, Condition 4, Condition 5 and Condition 6 hold trivially.

Case: $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1} = \mathcal{P}' \cup \{ \mathrm{out}(t, t'); Q \}, \sigma_{n-1}, \mathcal{L}_{n-1}) \xrightarrow{K(t)} (\mathcal{E}_{n-1}, \mathcal{S}_{n-1},$
$\mathcal{P}' \cup^{\#} \{ Q \}, \sigma_{n-1} \cup \{^{t'}/_x\}, \mathcal{L}_{n-1})$. This step requires that x is fresh and $\nu\mathcal{E}_{n-1}.\sigma \vdash t$. Using Lemma 8, we have that there is an execution $\emptyset \xrightarrow{F_1} S_1 \xrightarrow{F_2} \ldots \xrightarrow{F_{f(n)}} S_{f(n-1)} \rightarrow^* S \in \mathit{exec}^{msr}_E([\![P]\!])$ such that $!\mathsf{K}(t) \in_E S$ and $S_{f(n-1)} \rightarrow^*_R S$ for $R = \{ \text{MDOUT}, \text{MDPUB}, \text{MDFRESH}, \text{MDAPPL} \}$. Let p and $\tilde{t}$ such that $\{\mathrm{out}(t, t'); Q\} \leftrightarrow_P \mathsf{state}_p(\tilde{t})$. By Definition 20, there is a $ri \in \mathit{ginsts}([\![P]\!]_{=p})$ such that $\mathsf{state}_p(\tilde{t})$ is part of its premise. From the definition of $[\![P]\!]_{=p}$, we see that we can choose $ri = [\mathsf{state}_p(\tilde{t}), \mathsf{In}(t)] -[\mathrm{InEvent}(t)]\rightarrow [\mathsf{Out}(t'), \mathsf{state}_{p{\cdot}1}(\tilde{t})]$. To apply this rule, we need the fact $\mathsf{In}(t)$. Since $\nu\mathcal{E}_{n-1}.\sigma \vdash t$, as mentioned before, we can apply Lemma 8. It follows that there is an execution $\emptyset \xrightarrow{F_1} S_1 \xrightarrow{F_2} \ldots \xrightarrow{F_{n'}} S_{n'} \rightarrow^* S \in \mathit{exec}^{msr}_E([\![P]\!])$ such that $!\mathsf{K}(t) \in_E S$ and $S_{n'} \rightarrow^*_R S$ for $R = \{ \text{MDOUT}, \text{MDPUB}, \text{MDFRESH}, \text{MDAPPL} \}$. From S, we can now go two steps further, using MDIN and ri:

$$\begin{aligned} \emptyset \xrightarrow{F_1}_{[\![P]\!]} S_1 \ldots \xrightarrow{F_{n'}}_{[\![P]\!]} S_{n'} \rightarrow^*_{R \subset [\![P]\!]} S = S_{n'+s-2} \\ \xrightarrow{K(t)}_{[\![P]\!]} S_{n'+s-1} \xrightarrow{\mathrm{InEvent}(t)}_{[\![P]\!]} S_{n'+s} \in \mathit{exec}^{msr}([\![P]\!]) \end{aligned}$$

where $S_{n'+s-1} = S \cup^{\#} \{ \mathsf{In}(t) \}^{\#}$ and $S_{f(n)} = S \setminus^{\#} \{ \mathsf{state}_p(\tilde{t}) \} \cup^{\#} \{ \mathsf{Out}(t'), \mathsf{state}_{p{\cdot}1}(\tilde{t}) \}$.

Taking $k = n' + s - 1$ we immediately obtain that Condition 7 holds. Note first that, since $S_{n'} \rightarrow_R S$, $\mathit{set}(S_{n'}) \setminus \{ \mathsf{Fr}(t), \mathsf{Out}(t) | t \in \mathcal{M} \} \subset \mathit{set}(S)$ and $\mathit{set}(S) \setminus \{ !\mathsf{K}(t) | t \in \mathcal{M} \} \subset \mathit{set}(S_{n'})$. Since $\mathcal{P}_n = \mathcal{P}_{n-1} \setminus \{\mathrm{out}(t, t'); Q\} \cup \{Q\}$ and $\{Q\} \leftrightarrow_P \{\mathsf{state}_{p{\cdot}1}(\tilde{t})\}$ (by definition of $[\![P]\!]_{=p}$), we have that $\mathcal{P}_n \leftrightarrow_P S_{f(n)}$, i. e., Condition 3 holds. Condition 4 holds since t' was added to σ_{n-1} and $\mathsf{Out}(t)$ added to $S_{f(n-1)}$. Condition 6 holds since $\mathrm{K}(t)$ appears right before $\mathrm{InEvent}(t)$. Condition 1, Condition 2, and Condition 5 hold trivially.

Case: $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1} = \mathcal{P}' \cup \{\mathrm{in}(t,N);Q\}, \sigma_{n-1}, \mathcal{L}_{n-1}) \to (\mathcal{E}_{n-1}, \mathcal{S}_{n-1},$
$\mathcal{P}' \cup^{\#} \{ Q\theta \}, \sigma_{n-1}, \mathcal{L}_{n-1})$. This step requires that θ is grounding for N and that $\nu\mathcal{E}_{n-1}.\sigma_{n-1} \vdash \langle t, N\theta\rangle$. Using Lemma 8, we have that there is an execution $\emptyset \xrightarrow{F_1} S_1 \xrightarrow{F_2} \ldots \xrightarrow{F_{f(n-1)}} S_{f(n-1)} \to^* S \in exec^{msr}_E([\![P]\!])$ such that $!\mathsf{K}(t) \in_E S$ and $S_{f(n-1)} \to^*_R S$ for $R = \{ \text{MDOUT}, \text{MDPUB}, \text{MDFRESH}, \text{MDAPPL} \}$. The same holds for $N\theta$. We can combine those executions, by removing duplicate instantiations of FRESH, MDFRESH and MDOUT. (This is possible since $!\mathsf{K}$ is persistent.) Let $\emptyset \xrightarrow{F_1} S_1 \xrightarrow{F_2} \ldots \xrightarrow{F_{f(n-1)}} S_{f(n-1)} \to^*_R \overline{S} \in exec^{msr}_E([\![P]\!])$ this combined execution, and $!\mathsf{K}(t), !\mathsf{K}(N\theta) \in_E \overline{S}$.

Let p and $\tilde{t}$ be such that, $\mathrm{in}(t,N);Q \leftrightarrow_P \mathsf{state}_p(\tilde{t})$. By Definition 20 there is a $ri \in ginsts([\![P]\!]_{=p})$ such that $\mathsf{state}_p(\tilde{t})$ is part of its premise. From the definition of $[\![P]\!]_{=p}$ and the fact that θ is grounding for $N\theta$, we have $\mathsf{state}_p(\tilde{t})$ in their premise, namely,

$$ri = [\mathsf{state}_p(\tilde{t}), \mathsf{In}(\langle t, N\theta\rangle)] \; -[\; \mathrm{InEvent}(\langle t, N\theta\rangle) \;]\to [\mathsf{state}_{p\cdot 1}(\tilde{t} \cup (vars(N)\theta)].$$

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

$$\emptyset \xrightarrow{F_1}_{[\![P]\!]} S_1 \ldots \xrightarrow{F_{n'}}_{[\![P]\!]} S_{n'} \to^*_{R \subset [\![P]\!]} \overline{S} = S_{n'+s-3}$$
$$\xrightarrow{(\text{MDAPPL})}_{[\![P]\!]} S_{n'+s-2} \xrightarrow{K(\langle t,N\theta\rangle)}_{[\![P]\!]} S_{n'+s-1} \xrightarrow{\mathrm{InEvent}(\langle t,N\theta\rangle)}_{[\![P]\!]} S_{n'+s} \in exec^{msr}([\![P]\!])$$

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 $!\mathsf{K}(t), !\mathsf{K}(N\theta) \in S$
- $S_{n'+s-2} = S \cup^{\#} \{ !\mathsf{K}(\langle t, N\theta\rangle) \}^{\#}$,
- $S_{n'+s-1} = S \cup^{\#} \{ \mathsf{In}(\langle t, N\theta\rangle) \}^{\#}$,
- $S_{n'+s} = S \setminus^{\#} \{ \mathsf{state}_p(\tilde{t}) \} \cup^{\#} \{ \mathsf{state}_{p\cdot 1}(\tilde{t} \cup (vars(N)\theta)) \}$.

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

We now show that Condition 3 holds. Since by induction hypothesis, $\mathrm{in}(t,N);Q \leftrightarrow_P \mathsf{state}_p(\tilde{t})$, 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 $\mathcal{P}_n = \mathcal{P}_{n-1} \setminus^{\#} \{\mathrm{in}(t,N);Q\} \cup^{\#} \{Q\}$, we have that $\mathcal{P}_n \leftrightarrow_P S_{f(n)}$, i. e., Condition 3 holds. Condition 6 holds since $\mathrm{K}\langle t, N\theta\rangle)$ appears right before $\mathrm{InEvent}\langle t, N\theta\rangle)$. Condition 1, Condition 2, Condition 4, Condition 5 and Condition 6 hold trivially.

Case: $(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{\mathrm{out}(c,m);Q\} \cup \{\mathrm{in}(c',N);R\}, \sigma, \mathcal{L}) \to (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{ Q, R\theta \}, \sigma, \mathcal{L})$. This step requires that θ grounding for N, $t =_E N\theta$ and $c =_E c'$. Let p, p' and $\tilde{t}, \tilde{N}$ such that $\{\mathrm{out}(c,m);P\} \leftrightarrow_P \mathsf{state}_p(\tilde{t})$, $\{\mathrm{in}(c',N);Q\} \leftrightarrow_P \mathsf{state}_{p'}(\tilde{t'})$, and there are $ri \in ginsts([\![P]\!]_{=p})$ and $ri' \in ginsts([\![P]\!]_{=p'})$ such that $\mathsf{state}_p(\tilde{t})$ and $\mathsf{state}_{p'}(\tilde{t'})$ are part of their respective premise. From the definition of $[\![P]\!]_{=p}$ and

the fact that θ is grounding for N, we have:

$$\begin{aligned} ri_1 = \quad & [\mathsf{state}_p(\tilde{t})] \to [\mathsf{Msg}(t, N\theta), \mathsf{state}^{\mathsf{semi}}_{p\cdot 1}(\tilde{t})] \\ ri_2 = \quad & [\mathsf{state}_{p'}(\tilde{t'}), \mathsf{Msg}(t, N\theta)] \to [\mathsf{state}_{p'\cdot 1}(\tilde{t'} \cup (vars(N)\theta)), \mathsf{Ack}(t, N\theta)] \\ ri_3 = \quad & [\mathsf{state}^{\mathsf{semi}}_{p}(\tilde{t}), \mathsf{Ack}(t, N\theta)] \to [\mathsf{state}_{p\cdot 1}(\tilde{t})]. \end{aligned}$$

This allows to extend the previous execution by 3 steps:

$$\emptyset \xrightarrow{F_1}_{[\![P]\!]} S_1 \ldots \xrightarrow{F_{n'}}_{[\![P]\!]} S_{n'} \xrightarrow{(ri_1)}_{[\![P]\!]} S_{n'+s-2} \xrightarrow{(ri_2)}_{[\![P]\!]} S_{n'+s-1} \xrightarrow{(ri_2)}_{[\![P]\!]} S_{n'+s} \in exec^{msr}([\![P]\!])$$

where:

- $S_{n'+s-2} = S_{n'} \setminus^{\#} \{ \mathsf{state}_p(\tilde{t}) \} \cup^{\#} \{ \mathsf{Msg}(t, N\theta), \mathsf{state}^{\mathsf{semi}}_{p\cdot 1}(\tilde{t}) \}^{\#}$,
- $S_{n'+s-1} = S_{n'} \setminus^{\#} \{ \mathsf{state}_p(\tilde{t}), \mathsf{state}_{p'}(\tilde{t'}) \} \cup^{\#} \{ \mathsf{state}^{\mathsf{semi}}_{p\cdot 1}(\tilde{t}), \mathsf{state}_{p'\cdot 1}(\tilde{t'} \cup (vars(N)\theta)), \mathsf{Ack}(t, N\theta) \}^{\#}$,
- $S_{n'+s} = S_{n'} \setminus^{\#} \{ \mathsf{state}_p(\tilde{t}), \mathsf{state}_{p'}(\tilde{t'}) \} \cup^{\#} \{ \mathsf{state}_{p\cdot 1}(\tilde{t}), \mathsf{state}_{p'\cdot 1}(\tilde{t'} \cup (vars(N)\theta)) \}$.

We have that $\mathcal{P}_n = \mathcal{P}_{n-1} \setminus^{\#} \{ \mathrm{out}(c, m); Q,\ \mathrm{in}(c', t'); R \} \cup^{\#} \{ Q, R\theta \}^{\#}$. Exactly as in the two previous cases we have that $Q \leftrightarrow \mathsf{state}_{p\cdot 1}(\tilde{t})$, as well as $R\theta \leftrightarrow \mathsf{state}_{p'\cdot 1}(\tilde{t'})$. Hence we have that, Condition 3 holds. Condition 1, Condition 2, Condition 4, Condition 5, Condition 7 and Condition 6 hold trivially.

Case: $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1} = \mathcal{P}' \cup \{ \mathsf{if}\ pr(t_1, \ldots, t_l)\ \mathsf{then}\ Q\ \mathsf{else}\ Q' \}, \sigma_{n-1}, \mathcal{L}_{n-1}) \to (\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}' \cup \{ Q \}, \sigma_{n-1}, \mathcal{L}_{n-1})$. This step requires that $\sigma_{pr} \left\{ {}^{t_1}/_{x_1}, \ldots, {}^{t_l}/_{x_l} \right\}$ is satisfied. By induction hypothesis we have that $\mathcal{P}_{n-1} \leftrightarrow_P S_{n'}$. Let p and $\tilde{t}$ be such that $\mathsf{if}\ pr(t_1, \ldots, t_l)\ \mathsf{then}\ Q\ \mathsf{else}\ Q' \leftrightarrow_P \mathsf{state}_p(\tilde{t})$. By Definition 20, there is a $ri \in ginsts([\![P]\!]_{=p})$ such that $\mathsf{state}_p(\tilde{t})$ is part of its premise. By definition of $[\![P]\!]_{=p}$, we can choose $ri = [\mathsf{state}_p(\tilde{t})] -[\ \mathrm{Pred}_{pr}(t_1, \ldots, t_l)\ ]\to [\mathsf{state}_{p\cdot 1}(\tilde{t})]$. We can extend the previous execution by one step using ri, therefore:

$$\emptyset \xrightarrow{F_1}_{[\![P]\!]} S_1 \xrightarrow{F_2}_{[\![P]\!]} \ldots \xrightarrow{F_{n'}}_{[\![P]\!]} S_{n'} \xrightarrow{\mathrm{Pred}_{pr}(t_1,\ldots,t_l)}_{[\![P]\!]} S_{n'+1} \in exec^{msr}([\![P]\!])$$

with $S_{n'+1} = \{ S_{n'} \setminus^{\#} \{ \mathsf{state}_p(\tilde{t}) \}^{\#} \cup^{\#} \{ \mathsf{state}_{p\cdot 1}(\tilde{t}) \}^{\#} \}$. It is left to show that Conditions 1 to 7 hold for n. The last step is labelled $F_{f(n)} = \mathrm{Pred}_{pr}(t_1, \ldots, t_l)$. As $\sigma_{pr} \left\{ {}^{t_1}/_{x_1}, \ldots, {}^{t_l}/_{x_l} \right\}$ is satisfied, Condition 6 holds, in particular, α_{pred} is not violated. Since Pred_{pr} is reserved, Condition 7 holds as well.

As before, since we have that $\mathcal{P}_n = \mathcal{P}_{n-1} \setminus^{\#} \{ \mathsf{if}\ pr(t_1, \ldots, t_l)\ \mathsf{then}\ Q\ \mathsf{else}\ Q' \} \cup^{\#} \{Q\}$ and $\{Q\} \leftrightarrow \{\mathsf{state}_{p\cdot 1}(\tilde{t}, a)\}$ (by definition of the translation), we have that $\mathcal{P}_n \leftrightarrow_P S_{f(n)}$, and therefore Condition 3 holds.

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

Case: $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1} = \mathcal{P}' \cup \{ \mathsf{if}\ pr(t_1, \ldots, t_l)\ \mathsf{then}\ Q\ \mathsf{else}\ Q' \}, \sigma_{n-1}, \mathcal{L}_{n-1}) \to (\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}' \cup \{ Q' \}, \sigma_{n-1}, \mathcal{L}_{n-1})$. This step requires that the predicate $\sigma_{pr} \left\{ {}^{t_1}/_{x_1}, \ldots, {}^{t_l}/_{x_l} \right\}$ is not satisfied. This proof step is similar to the previous case, except ri is chosen to be

$$[\mathsf{state}_p(\tilde{t})] -[\ \mathrm{Pred\_not}_{pr}(t_1, \ldots, t_l)\ ]\to [\mathsf{state}_{p\cdot 2}(\tilde{t})].$$

The condition in α_{noteq} holds since $\sigma_{pr} \left\{ {}^{t_1}/_{x_1}, \ldots, {}^{t_l}/_{x_l} \right\}$ is not satisfied, and thus, by definition of the satisfaction relation, $\neg\sigma_{pr} \left\{ {}^{t_1}/_{x_1}, \ldots, {}^{t_l}/_{x_l} \right\}$ is satisfied.

Case: $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1} = \mathcal{P}' \cup \{\, \mathsf{event}(F);Q \,\}, \sigma_{n-1}, \mathcal{L}_{n-1}) \xrightarrow{F} (\mathcal{E}_{n-1}, \mathcal{S}_{n-1},$
$\mathcal{P}' \cup \{\, Q \,\}, \sigma_{n-1}, \mathcal{L}_{n-1})$. By induction hypothesis we have that $\mathcal{P}_{n-1} \leftrightarrow_P S_{n'}$. Let p and $\tilde{t}$ be such that $\mathsf{event}(F);Q \leftrightarrow_P \mathsf{state}_p(\tilde{t})$. By Definition 20, there is a $ri \in ginsts(\llbracket P \rrbracket_{=p})$ such that $\mathsf{state}_p(\tilde{t})$ is part of its premise. By definition of $\llbracket P \rrbracket_{=p}$, we can choose $ri = [\mathsf{state}_p(\tilde{t})] \,-[\, F, \mathrm{Event}() \,]\to\, [\mathsf{state}_{p\cdot 1}(\tilde{t})]$. We can extend the previous execution by one step using ri, therefore:

$$\emptyset \xrightarrow{F_1}_{\llbracket P \rrbracket} S_1 \xrightarrow{F_2}_{\llbracket P \rrbracket} \dots \xrightarrow{F_{n'}}_{\llbracket P \rrbracket} S_{n'} \xrightarrow{F,\mathrm{Event}()}_{\llbracket P \rrbracket} S_{n'+1} \in exec^{msr}(\llbracket P \rrbracket)$$

with $S_{n'+1} = S_{n'}) \setminus^{\#} \{\mathsf{state}_p(\tilde{t})\} \cup^{\#} \{\mathsf{state}_{p\cdot 1}(\tilde{t})\}$. It is left to show that Conditions 1 to 7 hold for n. Condition 3 holds because $\mathcal{P}_n = \mathcal{P}_{n-1} \setminus^{\#} \{\, \mathsf{event}(F);Q \,\} \cup^{\#} \{\, Q \,\}$ and $\{Q\} \leftrightarrow \{\mathsf{state}_{p\cdot 1}(\tilde{t})\}$ (by definition of $\llbracket P \rrbracket_{=p}$). Taking $k = f(n)$ Condition 7 holds. Condition 1, Condition 2, Condition 4, Condition 5 and Condition 6 hold trivially.

Case: $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1} = \mathcal{P}' \cup \{\, \mathsf{insert}\ t, t'; Q \,\}, \sigma_{n-1}, \mathcal{L}_{n-1}) \to (\mathcal{E}_{n-1}, \mathcal{S}_n = \mathcal{S}_{n-1}[t \mapsto t'],$
$\mathcal{P}' \cup \{\, Q \,\}, \sigma_{n-1}, \mathcal{L}_{n-1})$. By induction hypothesis we have that $\mathcal{P}_{n-1} \leftrightarrow_P S_{n'}$. Let p and $\tilde{t}$ be such that $\mathsf{insert}\ t, t'; Q \leftrightarrow_P \mathsf{state}_p(\tilde{t})$. By Definition 20, there is a $ri \in ginsts(\llbracket P \rrbracket_{=p})$ such that $\mathsf{state}_p(\tilde{t})$ is part of its premise. By definition of $\llbracket P \rrbracket_{=p}$, we can choose $ri = [\mathsf{state}_p(\tilde{t})] \,-[\, \mathrm{Insert}(t, t') \,]\to\, [\mathsf{state}_{p\cdot 1}(\tilde{t})]$. We can extend the previous execution by one step using ri, therefore:

$$\emptyset \xrightarrow{F_1}_{\llbracket P \rrbracket} S_1 \xrightarrow{F_2}_{\llbracket P \rrbracket} \dots \xrightarrow{F_{n'}}_{\llbracket P \rrbracket} S_{n'} \xrightarrow{\mathrm{Insert}(t,t')}_{\llbracket P \rrbracket} S_{n'+1} \in exec^{msr}(\llbracket P \rrbracket)$$

with $S_{n'+1} = S_{f(n-1)} \setminus^{\#} \{\, \mathsf{state}_p(\tilde{t}) \,\}^{\#} \cup^{\#} \{\, \mathsf{state}_{p\cdot 1}(\tilde{t}) \,\}^{\#}$. It is left to show that Conditions 1 to 7 hold for n.

This step is labelled $F_{f(n)} = \mathrm{Insert}(t, t')$, hence Condition 7 holds. To see that Condition 2 holds we let $j = f(n)$ for which both conjuncts trivially hold. Since, by induction hypothesis, Condition 6 holds, i.e., $[F_1, \dots F_{n'}] \vDash \alpha$, it holds for this step too. In particular, if $[F_1, \dots F_{n'}] \vDash \alpha_{in}$ and $[F_1, \dots F_{n'}] \vDash \alpha_{notin}$, we also have that $[F_1, \dots F_{n'}, F_{f(n)}] \vDash \alpha_{in}$ and $[F_1, \dots F_{n'}, F_{f(n)}] \vDash \alpha_{notin}$: as the Insert-action was added at the last position of the trace, it appears after any InIn or IsNotSet-action and by the semantics of the logic the formula holds.

Since $\mathcal{P}_n = \mathcal{P}_{n-1} \setminus^{\#} \{\, \mathsf{insert}\ t, t'; Q \,\} \cup^{\#} \{\, Q \,\}$ and $\{Q\} \leftrightarrow \{\mathsf{state}_{p\cdot 1}(\tilde{t})\}$ (by definition of $\llbracket P \rrbracket_{=p}$), we have that Condition 3 holds. Condition 1, Condition 4 and Condition 5 hold trivially.

Case: $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1} = \mathcal{P}' \cup \{\, \mathsf{delete}\ t; Q \,\}, \sigma_{n-1}, \mathcal{L}_{n-1}) \to (\mathcal{E}_{n-1}, \mathcal{S}_n = \mathcal{S}_{n-1}[t \mapsto \bot],$
$\mathcal{P}' \cup \{\, Q \,\}, \sigma_{n-1}, \mathcal{L}_{n-1})$. By induction hypothesis we have that $\mathcal{P}_{n-1} \leftrightarrow_P S_{n'}$. Let p and $\tilde{t}$ be such that $\mathsf{delete}\ t; Q \leftrightarrow_P \mathsf{state}_p(\tilde{t})$. By Definition 20, there is a $ri \in ginsts(\llbracket P \rrbracket_{=p})$ such that $\mathsf{state}_p(\tilde{t})$ is part of its premise. By definition of $\llbracket P \rrbracket_{=p}$, we can choose $ri = [\mathsf{state}_p(\tilde{t})] \,-[\, \mathrm{Delete}(t) \,]\to\, [\mathsf{state}_{p\cdot 1}(\tilde{t})]$. We can extend the previous execution by one step using ri, therefore:

$$\emptyset \xrightarrow{F_1}_{\llbracket P \rrbracket} S_1 \xrightarrow{F_2}_{\llbracket P \rrbracket} \dots \xrightarrow{F_{n'}}_{\llbracket P \rrbracket} S_{n'} \xrightarrow{\mathrm{Delete}(t)}_{\llbracket P \rrbracket} S_{n'+1} \in exec^{msr}(\llbracket P \rrbracket)$$

with $S_{n'+1} = S_{f(n-1)} \setminus^{\#} \{\mathsf{state}_p(\tilde{t})\} \cup^{\#} \{\mathsf{state}_{p\cdot 1}(\tilde{t})\}$. It is left to show that Conditions 1 to 7 hold for n.

This step is labelled $F_{f(n)} = \mathrm{Delete}(t)$, hence Condition 7 holds. Since, by induction hypothesis, Condition 6 holds, i.e., $[F_1, \dots F_{n'}] \vDash \alpha$, it holds for this step too. In particular, if $[F_1, \dots F_{n'}] \vDash \alpha_{in}$ and

$[F_1, \dots F_{n'}] \vDash \alpha_{notin}$, we also have that $[F_1, \dots F_{n'}, F_{f(n)}] \vDash \alpha_{in}$ and $[F_1, \dots F_{n'}, F_{f(n)}] \vDash \alpha_{notin}$: as the Insert-action was added at the last position of the trace, it appears after any InIn or IsNotSet-actions and by the semantics of the logic the formula holds.

We now show that Condition 2 holds. We have that $\mathcal{S}_n = \mathcal{S}_{n-1}[t \mapsto \bot]$ and therefore, for all $t' \neq_E Tt$, $\mathcal{S}_n(x) = \mathcal{S}_{n-1}(x)$. Hence for all such t' we have by induction hypothesis that for some u,

$$\exists j \leq n'.\mathrm{Insert}(t', u) \in F_j \wedge \forall j', u'.j < j' \leq n' \rightarrow \mathrm{Insert}(t', u') \notin_E F_{j'} \wedge \mathrm{Delete}(t') \notin_E F_{j'}$$

As, $F_{n'+1} \neq_E \mathrm{Delete}(x, u)$ and, for all $u' \in \mathcal{M}$, $F_{n'+1} \neq_E \mathrm{Insert}(x, u')$ we also have that

$$\exists j \leq n' + 1.\mathrm{Insert}(t', u) \in F_j \wedge \forall j', u'.j < j' \leq n' + 1 \rightarrow \mathrm{Insert}(t', u') \notin_E F_{j'} \wedge \mathrm{Delete}(t') \notin_E F_{j'}.$$

For $t' =_E t$, the above condition can never be true, because $F_{n'+1} = \mathrm{Delete}(t)$ which allows us to conclude that Condition 2 holds.

Since $\mathcal{P}_n = \mathcal{P}_{n-1} \setminus^{\#} \{ \text{ delete } t; Q \} \cup^{\#} \{ Q \}$ and $\{P\} \leftrightarrow \{\mathsf{state}_{p \cdot 1}(\tilde{t})\}$ (by definition of $[\![P]\!]_{=p}$), we have that Condition 3 holds. Condition 1, Condition 4 and Condition 5 hold trivially.

Case: $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1} = \mathcal{P}' \cup \{ \mathsf{lookup}\ t\ \mathsf{as}\ x\ \mathsf{in}\ Q\ \mathsf{else}\ Q' \}, \sigma_{n-1}, \mathcal{L}_{n-1}) \rightarrow (\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}' \cup \{ Q\{^v/_x\} \}, \sigma_{n-1}, \mathcal{L}_{n-1})$. This step requires that $\mathcal{S}_{n-1}(t') =_E v$ for some $t' =_E t$. By induction hypothesis we have that $\mathcal{P}_{n-1} \leftrightarrow_P S_{n'}$. Let p and $\tilde{t}$ be such that $\mathsf{lookup}\ t\ \mathsf{as}\ v\ \mathsf{in}\ Q\ \mathsf{else}\ Q' \leftrightarrow_P \mathsf{state}_p(\tilde{t})$. By Definition 20, there is a $ri \in \mathit{ginsts}([\![P]\!]_{=p})$ such that $\mathsf{state}_p(\tilde{t})$ is part of its premise. By definition of $[\![P]\!]_{=p}$, we can choose $ri = [\mathsf{state}_p(\tilde{t})] \multimap [\ \mathrm{IsIn}(t, v)\ ] \rightarrow [\mathsf{state}_{p \cdot 1}(\tilde{t}, v)]$. This is possible, since by well-formedness of P, no variable is bound twice. Thus, by definition of the translation, x is not bound by the left-hand side in $[\![P]\!]_{=p}$ (i. e., $x \notin \tilde{x}$), and thus x and can be instantiated to v (or anything else). We can extend the previous execution by one step using ri, therefore:

$$\emptyset \xrightarrow{F_1}_{[\![P]\!]} S_1 \xrightarrow{F_2}_{[\![P]\!]} \dots \xrightarrow{F_{n'}}_{[\![P]\!]} S_{n'} \xrightarrow{\mathrm{IsIn}(t,v)}_{[\![P]\!]} S_{n'+1} \in \mathit{exec}^{msr}([\![P]\!])$$

with $S_{n'+1} = S_{f(n-1)} \setminus^{\#} \{\mathsf{state}_p(\tilde{t})\} \cup^{\#} \{\mathsf{state}_{p \cdot 1}(\tilde{t})\}$. It is left to show that Conditions 1 to 7 hold for n.

This step is labelled $F_{f(n)} = \mathrm{IsIn}(t, v)$, hence Condition 7 holds.

From the induction hypothesis, Condition 2, we have that there is a j such that $\mathrm{Insert}(t, t') \in_E F_j$, $j \leq n'$ and

$$\forall j', u'.\ j < j' \leq n' \rightarrow \mathrm{Insert}(t, u') \notin_E F_{j'} \wedge \mathrm{Delete}(t) \notin_E F_{j'}$$

This can be strengthened, since $F_{f(n)} = \{ \mathrm{IsIn}(t, v) \}$:

$$\forall j', u'.\ j < j' \leq f(n) \rightarrow \mathrm{Insert}(t, u') \notin_E F_{j'} \wedge \mathrm{Delete}(t) \notin_E F_{j'}$$

This allows to conclude that Condition 2 holds. From Condition 2 it also follows that Condition 6, in particular α_{in}, holds.

We now show that Condition 3 holds. By induction hypothesis we have that $\mathsf{lookup}\ t\ \mathsf{as}\ x\ \mathsf{in}\ Q\ \mathsf{else}\ Q' \leftrightarrow_P \mathsf{state}_p(\tilde{t})$, and hence $P|_p\tau = (\mathsf{lookup}\ t\ \mathsf{as}\ x\ \mathsf{in}\ Q\ \mathsf{else}\ Q')\rho$ for some τ and ρ. Therefore, we also have that $P|_{p \cdot 1}\tau = Q\rho$. Thus $(P|_{p \cdot 1}\tau)\{^{v\rho}/_x\}) = (Q\rho)\{^{v\rho}/_x\} = Q\{^v/_x\}\rho$.

Now it is easy to see from the definition of $[\![P]\!]_{=p}$ that $\{Q\{^v/_x\}\} \leftrightarrow_P \{\mathsf{state}_{p\cdot 1}(\tilde{t}, v)\}$. Since $\mathcal{P}_n = \mathcal{P}_{n-1} \setminus^\# \{\, \mathsf{lookup}\ t\ \mathsf{as}\ x\ \mathsf{in}\ Q\ \mathsf{else}\ Q' \,\} \cup^\# \{\, Q\{^v/_x\} \,\}$ we have that $\mathcal{P}_n \leftrightarrow_P S_{f(n)}$, i. e., Condition 3 holds.

Condition 1, Condition 4 and Condition 5 hold trivially.

Case: $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1} = \mathcal{P}' \cup \{\, \mathsf{lookup}\ t\ \mathsf{as}\ x\ \mathsf{in}\ Q\ \mathsf{else}\ Q' \,\}, \sigma_{n-1}, \mathcal{L}_{n-1}) \to (\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}' \cup \{\, Q' \,\}, \sigma_{n-1}, \mathcal{L}_{n-1})$. This step requires that $S(t')$ is undefined for all $t' =_E t$. By induction hypothesis we have that $\mathcal{P}_{n-1} \leftrightarrow_P S_{n'}$. Let p and $\tilde{t}$ be such that $\mathsf{lookup}\ t\ \mathsf{as}\ x\ \mathsf{in}\ Q\ \mathsf{else}\ Q' \leftrightarrow_P \mathsf{state}_p(\tilde{t})$. By Definition 20, there is a $ri \in \mathit{ginsts}([\![P]\!]_{=p})$ such that $\mathsf{state}_p(\tilde{t})$ is part of its premise. By definition of $[\![P]\!]_{=p}$, we can choose $ri = [\mathsf{state}_p(\tilde{t})] \,-[\, \mathrm{IsNotSet}(t) \,]\to [\mathsf{state}_{p\cdot 1}(\tilde{t})]$. We can extend the previous execution by one step using ri, therefore:

$$\emptyset \xrightarrow{F_1}_{[\![P]\!]} S_1 \xrightarrow{F_2}_{[\![P]\!]} \cdots \xrightarrow{F_{n'}}_{[\![P]\!]} S_{n'} \xrightarrow{\mathrm{IsNotSet}(t)}_{[\![P]\!]} S_{n'+1} \in \mathit{exec}^{msr}([\![P]\!])$$

with $S_{n'+1} = S_{f(n-1)} \setminus^\# \mathsf{state}_p(\tilde{t}) \cup^\# \mathsf{state}_{p\cdot 1}(\tilde{t})$. It is left to show that Conditions 1 to 7 hold for n.

This step is labelled $F_{f(n)} = \mathrm{IsNotSet}(t)$, hence Condition 7 holds. Condition 2 also holds trivially and will be used to show Condition 6. Since this step requires that $S(t')$ is undefined for all $t' =_E t$, we have by Condition 2 that

$$\begin{aligned}\forall j \le f(n), u.\ \mathrm{Insert}(t,u) \in_E F_j \\ \to \exists j', u'. j < j' \le f(n) \wedge (\mathrm{Insert}(t,u') \in_E F_{j'} \vee \mathrm{Delete}(t) \in_E F_{j'})\end{aligned}$$

Now suppose that

$$\exists i \le f(n), y. \mathrm{Insert}(t,y) \in_E F_i)$$

As there exists an insert, there is a last insert and hence we also have

$$\exists i \le f(n), y. \mathrm{Insert}(t,y) \in_E F_i \quad \wedge \quad \forall i', y'. i < i' \le f(n) \to \mathrm{Insert}(t,y') \notin_E F_{i'}$$

Applying Condition 2 (cf above) we obtain that

$$\begin{aligned}\exists i \le f(n), y. \mathrm{Insert}(t,y) \in_E F_i \quad \wedge \quad \forall i', y'.\ i < i' \le f(n) \to \mathrm{Insert}(t,y') \notin_E F_{i'} \\ \wedge \quad \exists j', u'.\ i < j' \le f(n) \wedge (\mathrm{Insert}(t,u') \in_E F_{j'} \vee \mathrm{Delete}(t) \in_E F_{j'})\end{aligned}$$

which simplifies to

$$\begin{aligned}\exists i \le f(n), y. \mathrm{Insert}(t,y) \in_E F_i \quad \wedge \quad \forall i', y'.\ i < i' \le f(n) \to \mathrm{Insert}(t',y') \notin F_{i'} \\ \wedge \quad \exists j'.\ i < j' \le f(n) \wedge \mathrm{Delete}(t) \in_E F_{j'}\end{aligned}$$

Now we weaken the statement by dropping the first conjunct and restricting the quantification $\forall i'. i < i' \le f(n)$ to $\forall i'. j' < i' \le f(n)$, since $i < j'$.

$$\exists i \le f(n).\ \exists j'.\ i < j' \le f(n) \wedge \forall i'.\ j' < i' \le f(n) \to \mathrm{Insert}(t',y') \notin F_{i'} \wedge \mathrm{Delete}(t) \in_E F_{j'}$$

We further weaken the statement by weakening the scope of the existential quantification $\exists j'.\ i < j' \leq f(n)$ to $\exists j'.\ j' \leq f(n)$. Afterwards, i is not needed anymore.

$$\exists j'.\ j' \leq f(n) \land \forall i'.\ j' < i' \leq f(n) \rightarrow \mathrm{Insert}(t', y') \notin F_{i'} \land \mathrm{Delete}(t) \in_E F_{j'}$$

This statement was obtained under the hypothesis that $\exists i \leq f(n), y.\mathrm{Insert}(t, y) \in_E F_i)$. Hence we have that

$$\begin{aligned}&\forall i \leq f(n), y.\mathrm{Insert}(t, y) \notin_E F_i\\ \vee&\exists j' \leq f(n).\ \mathrm{Delete}(t) \in_E F_{j'} \land \forall i'.\ j' < i' \leq f(n) \rightarrow \mathrm{Insert}(t', y') \notin F_{i'}\end{aligned}$$

This shows that Condition 6, in particular α_{notin}, holds.

Since $\mathcal{P}_n = \mathcal{P}_{n-1} \setminus^{\#} \{\, \mathsf{lookup}\ t\ \mathsf{as}\ x\ \mathsf{in}\ Q\ \mathsf{else}\ Q' \,\} \cup^{\#} \{\, Q' \,\}$ and $\{Q'\} \leftrightarrow \{\mathsf{state}_{p\cdot 1}(\tilde{t})\}$ (by definition of $[\![P]\!]_{=p}$), we have that Condition 3 holds. Condition 1, Condition 4 and Condition 5 hold trivially.

Case: $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1} = \mathcal{P}' \cup \{\, \mathsf{lock}\ t; Q \,\}, \sigma_{n-1}, \mathcal{L}_{n-1}) \rightarrow (\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}' \cup^{\#} \{\, Q' \,\},$ $\sigma_{n-1}, \mathcal{L}_{n-1} \cup \{\, t \,\})$. This step requires that for all $t' =_E t, t' \notin \mathcal{L}_{n-1}$. Let p and $\tilde{t}$ such that $\mathsf{lock}\ t; Q \leftrightarrow_P \mathsf{state}_p(\tilde{t})$. By Definition 20, there is a $ri \in \mathit{ginsts}([\![P]\!]_{=p})$ such that $\mathsf{state}_p(\tilde{t})$ is part of its premise. By definition of $[\![P]\!]_{=p}$, we can choose $ri = [\mathsf{Fr}(l), \mathsf{state}_p(\tilde{t})] \ -[\ \mathrm{Lock}(l, t)\ ]\rightarrow\ [\mathsf{state}_{p\cdot 1}(\tilde{t}, l)]$ for a fresh name l, that never appeared in a Fr-fact in $\cup_{j \leq f(n-1)} S_j$. We can extend the previous execution by $s = 2$ steps using an instance of FRESH for l and ri:

$$\emptyset \xrightarrow{F_1}_{[\![P]\!]} S_1 \xrightarrow{F_2}_{[\![P]\!]} \cdots \xrightarrow{F_{n'}}_{[\![P]\!]} S_{n'} \rightarrow_{\{\,\mathrm{FRESH}\,\}} S_{n'+s-1} \xrightarrow{\mathrm{Lock}(l,t)}_{[\![P]\!]} S_{n'+s} \in \mathit{exec}^{msr}([\![P]\!])$$

with $S_{n'+s-1} = S_{f(n-1)} \setminus^{\#} \{\, \mathsf{state}_p(\tilde{t}) \,\}^{\#} \cup^{\#} \{\, \mathsf{Fr}(l) \,\}$ and $S_{n'+s} = S_{f(n-1)} \setminus^{\#} \{\, \mathsf{state}_p(\tilde{t}) \,\}^{\#} \cup^{\#}$ $\{\, \mathsf{state}_{p\cdot 1}(\tilde{t}) \,\}^{\#}$. It is left to show that Conditions 1 to 7 hold for n.

The step from $S_{f(n)-1}$ to $S_{f(n)}$ is labelled $F_{f(n)} = \mathrm{Lock}(l, t)$, hence Condition 7 and Condition 2 hold.

$F_{f(n)}$ also preserves Condition 5 for the new set of active locks $\mathcal{L}_{f(n)} = \mathcal{L}_{f(n-1)} \cup \{\, t \,\}$.

In the following we show by contradiction that α_{lock}, and therefore Condition 6 holds. α_{lock} held in the previous step, and $F_{f(n-1)+1}$ is empty, so we assume (by contradiction), that $F_{f(n)} = \mathrm{Lock}(l, t)$ violates α_{lock}. If this was the case, then:

$$\begin{aligned}\exists i < f(n), l_1.\ \mathrm{Lock}(l_1, t) \in_E F_i&\\ \land \forall i < j < f(n).\ &\mathrm{Unlock}(l_1, t) \notin_E F_j\\ &\vee \exists k.\ k \neq j \land \mathrm{Unlock}(\mathrm{l}_1, \mathrm{t}) \in_E F_k\\ &\vee \exists l_2, k.\ \mathrm{Lock}(\mathrm{l}_2, \mathrm{t}) \in_E F_k \land i < k \leq j\\ &\vee \exists l_2, k.\ \mathrm{Unlock}(\mathrm{l}_2, \mathrm{t}) \in_E F_k \land i \leq k < j\end{aligned}$$

We first note that the condition

$$\exists k.\ k \neq j \land \mathrm{Unlock}(\mathrm{l}_1, \mathrm{t}) \in_E F_k$$

is never satisfied if the first condition in the disjuction is unsatisfied and that hence this condition can be removed: if there is j such that $i < j < f(n)$ and $\mathrm{Unlock}(l_1, t) \in_E F_j$, then any k such that $j \neq k$ and

$\mathrm{Unlock}(l_1, t)$ has $\mathsf{Fr}(l_1)$ in the premise (by definition of the translation). Since Fr is linear, and since, by Definition 6, $\mathsf{Fr}(l_1)$ is only added once, this is impossible. Thus said condition can be removed.

Next we observe that, by definition of the translation, $\mathrm{Unlock}(a, b) \in_E F_i$ if and only if $\mathrm{Lock}(a', b) \notin_E F_i$. This allows us to tighten the bounds on the timepoint k in the two last disjuncts:

- we can remove the corner case $\mathrm{Lock}(\mathrm{l}_2, \mathrm{t}) \in_E F_j$, as it implies $\mathrm{Unlock}(\mathrm{l}_1, \mathrm{t}) \notin_E F_j$, and therefore satisfies the first disjunct;
- we can remove the corner case $\mathrm{Unlock}(\mathrm{l}_2, \mathrm{t}) \in_E F_i$, as it contradicts the first conjunct ($\mathrm{Lock}(l_1, t) \in_E F_i$).

These simplifications result in the following formula.

$$\begin{aligned}\exists i < f(n), l_1.\mathrm{Lock}(l_1, t) \in_E F_i \\ \wedge \forall i < j < f(n).\ \mathrm{Unlock}(l_1, t) \notin_E F_j \\ \vee \exists l_2, i < k < j.\ (\mathrm{Lock}(\mathrm{l}_2, \mathrm{t}) \in_E F_k \vee \mathrm{Unlock}(\mathrm{l}_2, \mathrm{t}) \in_E F_k)\end{aligned} \tag{2}$$

Since the semantics of the calculus requires that for all $t' =_E t$, $t' \notin \mathcal{L}_{n-1}$, by induction hypothesis, Condition 5, we have that

$$\begin{aligned}\forall i \leq f(n-1), l_1.\ \mathrm{Lock}(l_1, t) \in_E F_i \rightarrow \\ \exists i < j \leq f(n-1).\ \mathrm{Unlock}(l_1, t) \in_E F_j\end{aligned}$$

Since $F_{f(n-1)+1} = \emptyset$ and $f(n) = f(n-1) + 2$, we have:

$$\begin{aligned}\forall i < f(n), l_1.\mathrm{Lock}(l_1, t) \in_E F_i \rightarrow \\ \exists i < j < f(n).\ \mathrm{Unlock}(l_1, t) \in_E F_j\end{aligned}$$

We also have that

$$\begin{gathered}\exists i < j < f(n).\ \mathrm{Unlock}(l_1, t) \in_E F_j \\ \Leftrightarrow \\ \exists j.\ i < j < f(n) \wedge \mathrm{Unlock}(l_1, t) \in_E F_j \wedge \forall i < k < j.\ \mathrm{Unlock}(l_1, t) \notin_E F_k\end{gathered}$$

To see that this statement holds consider two cases. Either the above statement is not satisfiable, i.e. there is no j such that $\mathrm{Unlock}(l_1, t) \in_E F_j$. Then the second formula is not satisfiable either. Or the above formula is satisfied, i.e., there exists j, such that $\mathrm{Unlock}(l_1, t) \in_E F_j$. Choosing j to be minimal, we immediately have that $\forall i < k < j.\ \mathrm{Unlock}(l_1, t) \notin_E F_k$.

Hence, we obtain

$$\begin{aligned}\forall i < f(n), l_1.\ \mathrm{Lock}(l_1, t) \in_E F_i \rightarrow \\ \exists i < j < f(n).\ \mathrm{Unlock}(l_1, t) \in_E F_j \wedge \forall i < k < j.\ \mathrm{Unlock}(l_1, t) \notin_E F_k\end{aligned}$$

Combining this with (2) we obtain that

$$\exists i < f(n), l_1.\, \mathrm{Lock}(l_1, t) \in_E F_i \wedge \\ \exists i < j < f(n).\, \mathrm{Unlock}(l_1, t) \in_E F_j \\ \wedge \exists l_2, i < k < j.\, (\mathrm{Lock}(\mathrm{l}_2, \mathrm{t}) \in_E F_k \vee (\mathrm{Unlock}(\mathrm{l}_2, \mathrm{t}) \in_E F_k \wedge l_2 \neq_E l_1))$$

Fix $i < f(n)$, j such that $i < j < f(n)$, and l_1 such that $\mathrm{Lock}(l_1, t) \in_E F_i$ and $\mathrm{Unlock}(l_1, t) \in_E F_j$. Then, there are l_2 and k such that $i < k < j$ and either $\mathrm{Lock}(\mathrm{l}_2, \mathrm{t}) \in_E F_k$ or $\mathrm{Unlock}(\mathrm{l}_2, \mathrm{t}) \in_E F_k$, but $l_2 \neq_E l_1$. We proceed by case distinction.
Case 1: there is no unlock in between i and j, i. e., for all m, $i < m < j$, $\mathrm{Unlock}(l', t) \notin F_m$. Then there is a k and l_2 such that $\mathrm{Lock}(\mathrm{l}_2, \mathrm{t}) \in_E F_k$. In this case, α_{lock} is already invalid at the trace produced by the k-prefix of the execution, contradicting the induction hypothesis.
Case 2: there are l' and m, $i < m < j$ such that $\mathrm{Unlock}(l', t) \in F_m$ (see Figure 19).

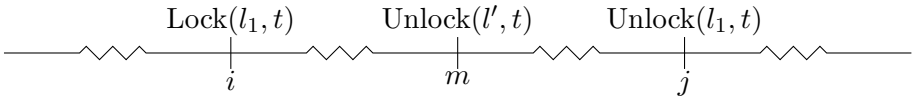

Fig. 19. Visualisation of Case 2.

We first observe that for any l, u, i_1, i_2 , if $\mathrm{Unlock}(\mathrm{l}, \mathrm{u}) \in_E F_{i_1}$ and $\mathrm{Unlock}(\mathrm{l}, \mathrm{u}) \in_E F_{i_2}$, then $i_1 = i_2$. We proceed by contradiction. By definition of $[\![P]\!]$ and well-formedness of P, the steps from $i_1 - 1$ to i_1 and from $i_2 - 1$ to i_2 must be ground instances of rules $[\![P]\!]_{=q}$ and $[\![P]\!]_{=q'}$ such that $P|_q$ and $P|_{q'}$ start with unlock commands that are labelled the same and have the same parameter, since every variable $lock_l$ in $[\![P]\!]$ appears in a Fr-fact in the translation for the corresponding lock command. By definition of $\overline{P}$, this means q and q' have a common prefix q_l that starts with a lock with this label.

Let $q_l \leq q$ denote that q_l is a prefix of q. Since $\overline{P}$ gives $\bot$ if there is a replication or a parallel between q_l and q or q', and since P is well-formed (does not contain $\bot$), we have that every state fact state_r for $q_l \leq r \leq q$ or $q_l \leq r \leq q'$ appearing in $[\![P]\!]$ is a linear fact, since no replication is allowed between q_l and q or q'. This implies that $q' \neq q$. Furthermore, every rule in $\cup_{q_l \leq r \leq q \vee q_l \leq r \leq q'} [\![P]\!]_{=r}$ adds at most one fact state_r and if it adds one fact, it either removes a fact $\mathsf{state}_{r'}$ where $r = r' \cdot 1$ or $r' \cdot 2$, or removes a fact $\mathsf{state}^{\mathsf{semi}}_{r'}$ where $r = r' \cdot 1$, which in turn requires removing $\mathsf{state}_{r'}$ (see translation of out). Therefore, either $q \leq q'$ or $q' \leq q$. But this implies that both have different labels, and since $[\![P]\!]_{=q_l}$ requires $\mathsf{Fr}(l)$, and E distinguishes fresh names, we have a contradiction. (A similiar observation is possible for locks: For any l, u, i_1, i_2 , if $\mathrm{Lock}(\mathrm{l}, \mathrm{u}) \in_E F_{i_1}$ and $\mathrm{Lock}(\mathrm{l}, \mathrm{u}) \in_E F_{i_2}$, then $i_1 = i_2$, since by definition of the translation, the transition from $i_1 - 1$ to i_1 or $i - 2 - 1$ to i_2 removes fact $\mathsf{Fr}(l)$.)

From the first observation we learn that , $l' \neq_E l_1$ for any l' and $m, i < m < j$ such that $\mathrm{Unlock}(l', t) \in F_m$. We now choose the smallest such m. By definition of $[\![P]\!]$, the step from S_{m-1} to S_m must be ground instance of a rule from $[\![P]\!]_{=q}$ for $P|_q$ starting with unlock. Since P is well-formed, there is a q_l such that $P|_{q_l}$ starts with lock, with the same label and parameter as the unlock. As before, since P is well-formed, and therefore there are no replications and parallels between q_l and q, there must be n such that $\mathrm{Lock}(l', t) \in F_n$ and $n < m$. We proceed again by case distinction.
Case 2a: $n < i$ (see Figure 20). By the fact that $m > i$ we have that there is no o such that $n < o < i$ and $\mathrm{Unlock}(l', t) \in_E F_o$ (see first observation). Therefore, the trace produced by the i-prefix of this execution does already not satisfy α_{lock}, i. e., $[F_1, \ldots, F_i] \not\models \alpha_{lock}$.

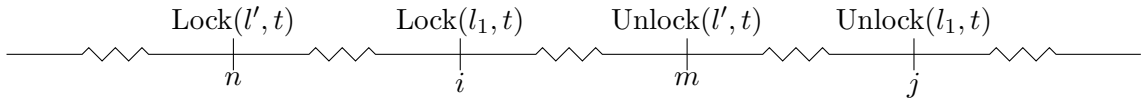

Fig. 20. Visualisation of Case 2a.

Case 2b: $i < n$ (see Figure 21). Again, α_{lock} is not satisfied, i.e., $[F_1, \ldots, F_n] \not\models \alpha_{lock}$, since there is no o such that $i < o < n$ and $\mathrm{Unlock}(l_1, t) \in_E F_o$.

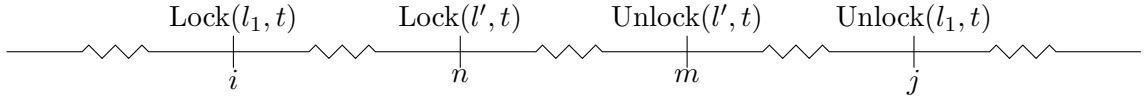

Fig. 21. Visualisation of Case 2b.

Since we could, under the assumption that Condition 1 to Condition 7 hold for $i \leq n'$, reduce every case in which $[F_1, \ldots, F_{n'+1}] \not\models \alpha_{lock}$ to a contradiction, we can conclude that Condition 6 holds for $n' + 1$.

Since $\mathcal{P}_n = \mathcal{P}_{n-1} \setminus^{\#} \{\, \mathsf{lock}\ t; Q \,\} \cup^{\#} \{\, Q \,\}$ and $\{Q\} \leftrightarrow \{\mathsf{state}_{p\cdot 1}(\tilde{t})\}$ (by definition of the translation), we have that Condition 3 holds. Condition 1, and Condition 4 hold trivially.

Case: $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{P}_{n-1} = \mathcal{P}' \cup \{\, \mathsf{unlock}\ t; Q \,\}, \sigma_{n-1}, \mathcal{L}_{n-1}) \to (\mathcal{E}_{n-1}, \mathcal{S}_{n-1},$
$\mathcal{P}' \cup^{\#} \{\, Q' \,\}, \sigma_{n-1}, \mathcal{L}_{n-1} \setminus \{\, t' \colon t' =_E t \,\})$. By induction hypothesis we have that $\mathcal{P}_{n-1} \leftrightarrow_P S_{n'}$. Let p and $\tilde{t}$ be such that $\mathsf{unlock}\ t; Q \leftrightarrow_P \mathsf{state}_p(\tilde{t})$. By Definition 20, there is a $ri \in ginsts(\llbracket P \rrbracket_{=p})$ such that $\mathsf{state}_p(\tilde{t})$ is part of its premise. By definition of $\llbracket P \rrbracket_{=p}$, we can choose $ri = [\mathsf{state}_p(\tilde{t})] -[\ \mathrm{Unlock}(l, t)\ ]\to [\mathsf{state}_{p\cdot 1}(\tilde{t})]$. We can extend the previous execution by one step using ri, therefore:

$$\emptyset \xrightarrow{F_1}_{\llbracket P \rrbracket} S_1 \xrightarrow{F_2}_{\llbracket P \rrbracket} \cdots \xrightarrow{F_{n'}}_{\llbracket P \rrbracket} S_{n'} \xrightarrow{\mathrm{Unlock}(l,t)}_{\llbracket P \rrbracket} S_{n'+1} \in exec^{msr}(\llbracket P \rrbracket)$$

with $S_{n'+1} = S_{f(n-1)} \setminus^{\#} \{\mathsf{state}_p(\tilde{t})\} \cup^{\#} \{\mathsf{state}_{p\cdot 1}(\tilde{t})\}$. It is left to show that Conditions 1 to 7 hold for n.

The step from $S_{f(n-1)}$ to $S_{f(n)}$ is labelled $F_{f(n)} = \mathrm{Unlock}(l, t)$, hence Condition 7 and Condition 2 hold.

In order to show that Condition 5 holds, we perform a case distinction. Assume $t \notin_E \in \mathcal{L}_{n-1}$. Then, $\mathcal{L}_{f(n-1)} = \mathcal{L}_{f(n)}$. In this case, Condition 5 holds by induction hypothesis. In the following, we assume $t \in_E \mathcal{L}_{n-1}$. Thus, there is $j \in n', l'$ such that $\mathrm{Lock}(l', t) \in_E F_j$ and for all k such that $j < k \leq n'$, $\mathrm{Unlock}(l', t) \notin_E F_k$.

Since $P|_p$ is an unlock node and P is well-formed, there is a prefix q of p, such that $P|_q$ is a lock with the same parameter and annotation. By definition of $\overline{P}$, there is no parallel and no replication between q and p. Note that any rule in $\llbracket P \rrbracket$ that produces a state named state_p for a non-empty p is such that it requires a fact with name $\mathsf{state}_{p'}$ for $p = p' \cdot 1$ or $p = p' \cdot 2$ (in case of the translation of out, it might require $\mathsf{state}^{\mathsf{semi}}_{p'}$, which in turn requires $\mathsf{state}_{p'}$). This means that, since $\mathsf{state}_p(\tilde{t}) \in S_{n'}$, there is an i such that $\mathsf{state}_q(\tilde{t'}) \in S_i$ and $\mathsf{state}_q(\tilde{t'}) \notin S_{i-1}$ for $\tilde{t'}$ a prefix to t. This rule is an instance of $\llbracket P \rrbracket_{=q}$ and thus labelled $F_i = \mathrm{Lock}(l, t)$. We proceed by case distinction.

Case 1: $j < i$ (see Figure 22). By induction hypothesis, Condition 6 holds for the trace up to n'. But, $[F_1, \ldots, F_i] \not\models \alpha_{lock}$, since we assumed that for all k such that $j < k \leq n'$, $\mathrm{Unlock}(l', t) \notin_E F_k$.

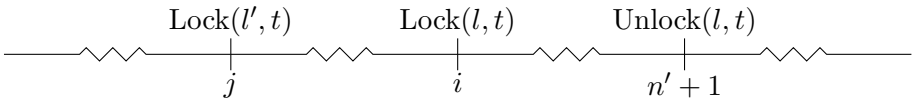

Fig. 22. Visualisation of Case 1.

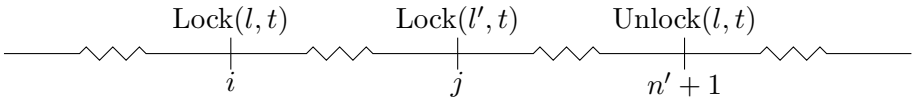

Fig. 23. Visualisation of Case 2.

Case 2: $i < j$ (see Figure 23). As shown in the lock case, any k such that $\mathrm{Unlock}(l,t) \in_E F_k$ is $k = n' + 1$. This contradicts Condition 6 for the trace up to j, since $[F_1, \ldots, F_j] \not\models \alpha_{lock}$, because there is not k such that $i < k < j$ such that $\mathrm{Unlock}(l,t) \in_E F_k$. This concludes the proof that Condition 5 holds for $n+1$.

Condition 6 holds, since none of the axioms, in particular not α_{lock}, become unsatisfied if they were satisfied for the trace up to $f(n-1)$ and an Unlock is added.

Since $\mathcal{P}_n = \mathcal{P}_{n-1} \setminus^{\#} \{ \text{unlock } t; Q \} \cup^{\#} \{ Q \}$ and $\{Q\} \leftrightarrow \{\mathsf{state}_{p\cdot 1}(\tilde{t})\}$ (by definition of the translation), we have that Condition 3 holds. Condition 1, and Condition 4 hold trivially.

□

B.3. Proof that $traces^{pi}(P) \supseteq hide(filter(traces^{msr}([\![P]\!])))$

To prove this direction we actually need to make a detour. We first define the notion of a *normal* msr execution and we next show that any msr execution resulting from a translation of a process has an equivalent normal execution. Finally we show that

$$\{\, tr \in hide(filter(traces^{msr}([\![P]\!]))) \mid tr \text{ is normal} \,\} \subseteq traces^{pi}(P)$$

Definition 21 (normal msr execution)**.** *An 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 *neither contains any fact with the symbol* $\mathsf{state}^{\mathsf{semi}}_p$ *for any* p*, nor any fact with symbol* Ack*.*
3. *if for some* i *and* $t_1, t_2 \in \mathcal{M}$*,* $\mathsf{Ack}(t_1,t_2) \in (S_{i-1} \setminus^{\#} S_i)$*, then there are* p *and* q *such that:*

$$S_{i-3} \rightarrow_{R_1} S_{i-2} \rightarrow_{R_2} S_{i-1} \rightarrow_{R_3} S_i \quad \text{, where:}$$

   - $R_1 = [\mathsf{state}_p(\tilde{x})] \rightarrow [\mathsf{Msg}(t_1,t_2), \mathsf{state}^{\mathsf{semi}}_p(\tilde{x})]$
   - $R_2 = [\mathsf{state}_q(\tilde{y}), \mathsf{Msg}(t_1,t_2)] \rightarrow [\mathsf{state}_{q\cdot 1}(\tilde{y} \cup \tilde{y}'), \mathsf{Ack}(t_1,t_2)]$
   - $R_3 = [\mathsf{state}^{\mathsf{semi}}_p(\tilde{x}), \mathsf{Ack}(t_1,t_2)] \rightarrow [\mathsf{state}_{p\cdot 1}(\tilde{x})]$.

4. $S_{n-1} \xrightarrow{E_n}_{[\![P,[],[]]\!],\mathrm{MDIN},\mathrm{INIT}} S_n$
5. *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), internal communications are always finished (item 2), and not interleaved with any other actions (item 3). Furthermore, the last action is neither the generation of a fresh name, nor a message deduction rule. 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 just before (item 5).

We will now show that any execution has an equivalent normal execution, i. e., an execution that has the same labels, up to reserved facts, and preserves α.

Lemma 11 (Normalisation)**.** *Le 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 $\mathrm{Init}() \in \mathcal{F}_{res}$, 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 $\mathrm{Init}()$ is only added if it was not present before, $[E_1^{(1)}, \dots, E_{n^{(1)}}^{(1)}] \vDash \alpha$, especially α_{init}.
2. For each fact $\mathsf{Ack}(t_1, t_2)$ contained in $S_{n^{(1)}}^{(1)}$, it also contains a fact $\mathsf{state}_p^{\mathsf{semi}}(\tilde{t})$ for some p and $\tilde{t}$ such that there exists a rule of type R_3 that consumes both of them, since $\mathsf{Ack}(t_1, t_2)$ can only be produced by a rule of type R_2 which consumes $\mathsf{Msg}(t_1, t_2)$ which in turn can only be produced along with a fact $\mathsf{state}_p^{\mathsf{semi}}(\tilde{t})$, and by definition of $[\![P]\!]$, there exists a rule in $[\![P]\!]_{=p}$ of form R_3 that consumes $\mathsf{Ack}(t_1, t_2)$ and $\mathsf{state}_p^{\mathsf{semi}}(\tilde{t})$. We append as many applications of rules of type R_3 as there are facts $\mathsf{Ack}(t_1, t_2) \in S_{n^{(1)}}^{(1)}$, and repeat this for all t_1, t_2 such that $\mathsf{Ack}(t_1, t_2) \in S_{n^{(1)}}^{(1)}$. Then, $S_{n^{(1)}}^{(1)} \to_{[\![P]\!]} S_{n'}^{(1)}$ and $S_{n'}^{(1)}$ does not contain Ack-facts anymore.
   If $S_{n'}^{(1)}$ contains a fact $\mathsf{state}_p^{\mathsf{semi}}(\tilde{t})$, we remove the last transition that produced this fact, i. e., for i such that $S_i = S_{i-1} \setminus^{\#} \{\, \mathsf{state}_p(\tilde{t}) \,\}^{\#} \cup^{\#} \{\, \mathsf{Msg}(t_1, t_2), \mathsf{state}_p^{\mathsf{semi}}(\tilde{t}) \,\}^{\#}$, we define

   $$S_j^{(1)\prime} := \begin{cases} S_j^{(1)} & \text{if } j \leq i-1 \\ S_{j+1}^{(1)} \setminus^{\#} \{\, \mathsf{Msg}(t_1, t_2), \mathsf{state}_p^{\mathsf{semi}}(\tilde{t}) \,\}^{\#} \cup^{\#} \{\, \mathsf{state}_p(\tilde{t}) \,\}^{\#} & \text{if } i-1 < j < n' \end{cases}$$

The resulting execution is valid, since $\mathsf{state}_p^{\mathsf{semi}}(\tilde{t}) \in S_{n'}^{(1)}$ and since $\mathsf{Msg}(t_1,t_2) \in S_{n'}^{(1)}$. The latter is the case because if $\mathsf{Msg}(t_1,t_2)$ would be consumed at a later point, say j, $j+1$ would contain $\mathsf{Ack}(t_1,t_2)$, but since $S_{n'-1}^{(1)'}$ does not contain Ack-facts, they can only be consumed by a rule of type R_3, which would have consumed $\mathsf{state}_p^{\mathsf{semi}}(\tilde{t})$. We repeat this procedure for every remaining $\mathsf{state}_p^{\mathsf{semi}}(\tilde{t}) \in S_{n'}^{(1)}$, and call the resulting trace

$$S_0^{(2)} \xrightarrow{E_1^{(2)}}_{[\![P]\!]} \dots \xrightarrow{E_n^{(2)}}_{[\![P]\!]} S_{n^{(2)}}^{(2)}$$

Since no rule added or removed has an action,

$$hide([E_1,\dots,E_n]) = hide([E_1^{(2)},\dots,E_{n^{(2)}}^{(2)}]) \text{ and } [E_1^{(2)},\dots,E_{n^{(2)}}^{(2)}] \vDash \alpha.$$

3. We transform $S_0^{(1)} \xrightarrow{E_1^{(1)}}_{[\![P]\!]} \dots \xrightarrow{E_n^{(1)}}_{[\![P]\!]} S_{n^{(1)}}^{(1)}$ as follows (all equalities are modulo E): Let us call instances of R_1, R_2 or R_3 that appear outside a chain

$$S_{i-3} \rightarrow_{R_1} S_{i-2} \rightarrow_{R_2} S_{i-1} \rightarrow_{R_3} S_i$$

for some i and $t_1, t_2 \in \mathcal{M}$ "unmarked". Do the following for the smallest i that is an unmarked instance of R_3 (we will call the instance of R_3 ri_3 and suppose it is applied from S_{i-1} to S_i): Apply ri_3 after $j < i$ such that S_{j-1} to S_j is the first unmarked instance of R_2, for some q and $\tilde{y}$, i. e., this instance produces a fact $\mathsf{state}_{q\cdot 1}(\tilde{y},\tilde{y}')$ and a fact $\mathsf{Ack}(t_1,t_2)$. Since there is no rule between j and i that might consume $\mathsf{Ack}(t_1,t_2)$ (only rules of form R_3 do, and ri_3 is the first unmarked instance of such a rule) and since ri_3 does not consume $\mathsf{state}_{q\cdot 1}(\tilde{y},\tilde{y}')$, we can move ri_3 between j and $j+1$, adding the conclusions of ri_3 and removing the premises of ri_3 from every $S_{j+1},\dots,S_i$. Note that unmarked instances of R_2 and R_3 are guaranteed to be preceeded by a marked R_1, and therefore only remove facts of form $\mathsf{Ack}(\dots)$ or $\mathsf{Msg}(\dots)$ that have been added in that preceeding step. Since the transition at step j requires a fact $\mathsf{Msg}(t_1,t_2)$, there is an instance of R_1 prior to j, say at $k < j$, since only rules of form R_1 produces facts labelled $\mathsf{Msg}(t_1,t_2)$. Since ri_3 is now applied from S_j to S_{j+1}, we have that an instance ri_1 of a rule of form R_1 that produces $\mathsf{state}_p^{\mathsf{semi}}(\tilde{t})$ must appear before j, i. e., $ri_1 \in ginsts([\![P]\!]_{=p})$. Therefore, it produces a fact $\mathsf{Msg}(t_1,t_2)$ indeed. We choose the largest k that has an unmarked R_1 that produces $\mathsf{Msg}(t_1,t_2)$ and $\mathsf{state}_p^{\mathsf{semi}}(\tilde{t})$ and move it right before j, resulting in the following msr execution:

$$S_t^{(1)'} := \begin{cases} S_t^{(1)} & \text{if } t < k \\ S_{t+1}^{(1)} \cup^{\#} \{\, \mathsf{Msg}(t_1,t_2), \mathsf{state}_p^{\mathsf{semi}}(\tilde{t}) \,\}^{\#} \setminus^{\#} \{\, \mathsf{state}_p(\tilde{t}) \,\}^{\#} & \text{if } k \le t < j-1 \\ S_{(t)}^{(1)} & \text{if } j-1 \le t < j+1 \\ S_{(t-1)}^{(1)} \setminus^{\#} \{\, \mathsf{state}_p^{\mathsf{semi}}(\tilde{t}), \mathsf{Ack}(t_1,t_2) \,\}^{\#} \cup^{\#} \{\, \mathsf{state}_{p\cdot 1}(\tilde{t}) \,\}^{\#} & \text{if } j+1 \le t < i+1 \\ S_t^{(1)} & \text{if } i+1 \le t \end{cases}$$

We apply this procedure until it reaches a fixpoint and call the resulting trace

$$S_0^{(3)} \xrightarrow{E_1^{(3)}}_{[\![P]\!]} \cdots \xrightarrow{E_n^{(3)}}_{[\![P]\!]} S_{n^{(3)}}^{(3)}$$

Since no rule moved during the procedure has an action,

$$hide([E_1, \ldots, E_n]) = hide([E_1^{(3)}, \ldots, E_{n^{(3)}}^{(3)}]) \text{ and } [E_1^{(3)}, \ldots, E_{n^{(3)}}^{(3)}] \vDash \alpha.$$

4. If the last transition is in $\{$ MDOUT, MDPUB, MDFRESH, MDAPPL, FRESH $\}$, we remove it. Repeat until fixpoint is reached and call the resulting trace

$$S_0^{(4)} \xrightarrow{E_1^{(4)}}_{[\![P]\!]} \cdots \xrightarrow{E_n^{(4)}}_{[\![P]\!]} S_{n^{(4)}}^{(4)}$$

Since no rule removed during the procedure has an action,

$$hide([E_1, \ldots, E_n]) = hide([E_1^{(4)}, \ldots, E_{n^{(4)}}^{(4)}]) \text{ and } [E_1^{(4)}, \ldots, E_{n^{(4)}}^{(4)}] \vDash \alpha.$$

5. 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

$$S_0^{(5)} \xrightarrow{E_1^{(5)}}_{[\![P]\!]} \cdots \xrightarrow{E_n^{(5)}}_{[\![P]\!]} S_{n^{(5)}}^{(5)}$$

Since $[E_1^{(4)}, \ldots, E_{n^{(4)}}^{(4)}] \vDash \alpha$, especially α_{inev}, there is no action that is not in $\mathcal{F}_{res}$ between the abovementioned instance of MDIN, therefore, $hide([E_1, \ldots, E_n]) = hide([E_1^{(5)}, \ldots, E_{n^{(5)}}^{(5)}])$ holds. Since α_{inev} is the only part of α that mentions K, and since the tranformation preserved α_{inev}, we have that $[E_1^{(5)}, \ldots, E_{n^{(5)}}^{(5)}] \vDash \alpha$.

□

The next proposition states that normal executions have a kind of prefix closure. Normality is preserved when removing a prefix, except if the prefix ends in the middle of an internal communication or with an action in $\{$ MDOUT, MDPUB, MDFRESH, MDAPPL, FRESH $\}$. In that case removing some additional actions will provide a normal execution.

Proposition 3. *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 then*

1. *if $n \geq 2$ and no* Ack*-fact in $(S_{n-1} \setminus^{\#} S_n)$, then there exists $m < n$ such that $S_m \to_R^* S_{n-1}$ and $\emptyset \xrightarrow{E_1}_{[\![P]\!]} \cdots \xrightarrow{E_m}_{[\![P]\!]} S_m \in exec^{msr}([\![P]\!])$ is normal.*

2. *if for some* $t_1, t_2 \in \mathcal{M}$, $\mathsf{Ack}(t_1, t_2) \in (S_{n-1} \setminus^{\#} S_n)$, *then there exists* $m \leq n-3$ *such that* $S_m \to_R^* S_{n-3}$ *and* $\emptyset \xrightarrow{E_1}_{[\![P]\!]} \cdots \xrightarrow{E_m}_{[\![P]\!]} S_m \in exec^{msr}([\![P]\!])$ *is normal.*

for $R = \{$ MDOUT, MDPUB, MDFRESH, MDAPPL, FRESH $\}$

Proof. If $n \geq 2$ and there is no Ack-fact in $S_{n-1} \setminus S_n$, then we chose the largest $m < n$ such that $S_{m-1} \xrightarrow{E_m}_{[\![P,[],[]]\!],\text{INIT},\text{MDIN}} S_m$, or, if there is an Ack-fact in $S_{n-1} \setminus S_n$, we will chose the largest $m < n-2$ such that $S_{m-1} \xrightarrow{E_m}_{[\![P,[],[]]\!],\text{INIT},\text{MDIN}} S_m$. Such an m exists since $S_0 \xrightarrow{\text{Init}()}_{[\![P]\!]} S_1$ and $\text{INIT} \notin R$.

Such an m always exists, since in case $t_1, t_2 \in \mathcal{M}$, $\mathsf{Ack}(t_1, t_2) \in (S_{n-1} \setminus^{\#} S_n)$, we have that $n \geq 3$, because the execution is normal (item 3 of Definition 21).

Moreover, $S_m \to_R^* S_{n-1}$ in case of no Ack-fact in $S_{n-1} \setminus S_n$ and $S_m \to_R^* S_{n-3}$ if there is an Ack-fact in $S_{n-1} \setminus S_n$, 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 21 is preserved as in both cases $m \geq 1$.
- $S_m \to_R^* S_{n-1}$, respectively $S_{m'} \to_R^* S_{n-3}$, implies item 2 of Definition 21, as none of the rules in $R = \{$ MDOUT, MDPUB, MDFRESH, MDAPPL, FRESH $\}$ remove Ack- or $\mathsf{state}^{\mathsf{semi}}$-facts, and the chain of rules R_1, R_2, R_3 consumes as many as it produces. Thus, if they where in S_m, they would be in S_n, too.
- Items 3 and 5 of Definition 21 hold for all parts of the trace, and therefore also for the prefix of size m.
- Item 4 of Definition 21 holds trivially since $S_m \to_R^* S_{n-1}$, respectively $S_m \to_R^* S_{n-3}$.

□

Definition 22. *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* $\{\mathsf{state}_p(\tilde{t}) \mid \exists p, \tilde{t}.\, \mathsf{state}_p(\tilde{t}) \in^{\#} S\}^{\#}$ *such that whenever* $Q \in^{\#} \mathcal{P}$ *is mapped to* $\mathsf{state}_p(\tilde{t}) \in^{\#} S$*, then:*

1. $\mathsf{state}_p(\tilde{t}) \in_E prems(R)$ *for* $R \in ginsts([\![P]\!]_{=p})$.
2. *Let* θ *be a grounding substitution for* $state_p(\tilde{x}) \in prems([\![P]\!]_{=p})$ *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 $\mathsf{state}_p(\tilde{t})$.

Remark 3. *Note that* $\leftrightsquigarrow_P$ *has the following properties (by the fact that it defines a bijection between multisets).*

- *If* $\mathcal{P}_1 \leftrightsquigarrow_P S_1$ *and* $\mathcal{P}_2 \leftrightsquigarrow_P S_2$ *then* $\mathcal{P}_1 \cup^{\#} \mathcal{P}_2 \leftrightsquigarrow_P S_1 \cup^{\#} S_2$.

– *If* $\mathcal{P}_1 \leftrightsquigarrow_P S_1$ *and* $Q \leftrightsquigarrow_P \mathsf{state}_p(\tilde{t})$ *for* $Q \in \mathcal{P}_1$ *and* $\mathsf{state}_p(\tilde{t}) \in S_1$ *(i.e.* Q *and* $\mathsf{state}_p(\tilde{t})$ *are related by the bijection defined by* $\mathcal{P}_1 \leftrightsquigarrow_P S_1$*) then* $\mathcal{P}_1 \setminus^{\#} \{Q\} \leftrightsquigarrow_P S_1 \setminus^{\#} \{\mathsf{state}_p(\tilde{t})\}$.

Lemma 12. *Le* P *be a well-formed ground process. If*

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

is normal (see Definition 21) and $[E_1, \dots, E_n] \vDash \alpha$ *(see Definition 15), then there are* $(\mathcal{E}_0, \mathcal{S}_0, \mathcal{P}_0, \sigma_0, \mathcal{L}_0), \dots, (\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{P}_{n'}, \sigma_{n'}, \mathcal{L}_{n'})$ *and* $F_1, \dots, F_{n'}$ *such that:*

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

where $(\mathcal{E}_0, \mathcal{S}_0, \mathcal{P}_0, \sigma_0, \mathcal{L}_0) = (\emptyset, \emptyset, \emptyset, \{ P \}, \emptyset, \emptyset)$ *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. $\mathcal{E}_{f(i)} = \{ a \in FN \mid ProtoNonce(a) \in_E \bigcup_{1 \le j \le i} E_j \}$
2. $\forall\, t \in \mathcal{M}.\, \mathcal{S}_{f(i)}(t) = \begin{cases} u & \text{if } \exists j \le i. \mathrm{Insert}(t,u) \in_E E_j \\ & \quad \wedge \forall j', u'. j < j' \le i \to \mathrm{Insert}(t,u') \notin_E E_{j'} \wedge \mathrm{Delete}(t) \notin_E E_{j'} \\ \bot & \text{otherwise} \end{cases}$
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. $\mathcal{L}_{f(i)} =_E \{ t \mid \exists j \le i, u.\, \mathrm{Lock}(u,t) \in_E E_j \wedge \forall j < k \le i. \mathrm{Unlock}(u,t) \notin_E E_k \}$.

Furthermore,

6. $hide([E_1, \dots, E_n]) =_E [F_1, \dots, F_{n'}]$.

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

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

We chose $n' = 0$ and thus

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

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 19, $[\![P]\!]_{=[]} = [\![P, [], []]\!]_{=[]}$. We see from Figure 18 that for every P we have that $\mathsf{state}_{[]}() \in prems(R\theta)$, 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

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

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

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

fulfilling Conditions 1 to 7.

Assume now for the following argument, that there is no fact with the symbol Ack in $S_{n-1} \setminus^{\#} S_n$. This is the case for all cases except for the case where rule instance applied from S_{n-1} to S_n has the form $ri = [\mathsf{state}_p^{\mathsf{semi}}(\tilde{s}), \mathsf{Ack}(t_1, t_2)] \; -[]\rightarrow \; [\mathsf{state}_{p\cdot 1}(\tilde{s})]$. This case will require a similar, but different argument, which we will present when we come to this case.

Since $\emptyset \xrightarrow{E_1}_{[\![P]\!]} \cdots \xrightarrow{E_n}_{[\![P]\!]} S_n \in exec^{msr}([\![P]\!])$ is normal and $n \geq 2$, by Proposition 3, 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 $\emptyset \xrightarrow{E_1}_{[\![P]\!]} \cdots \xrightarrow{E_m}_{[\![P]\!]} S_m \in exec^{msr}([\![P]\!])$ is normal, too. This allows us to apply the induction hypothesis on $\emptyset \xrightarrow{E_1}_{[\![P]\!]} \cdots \xrightarrow{E_m}_{[\![P]\!]} S_m \in exec^{msr}([\![P]\!])$. Hence there is a monotonically increasing function from $\mathbb{N}_m \rightarrow \mathbb{N}_{n'}$ and an execution such that Conditions 1 to 7 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 $(\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{P}_{n'}, \sigma_{n'}, \mathcal{L}_{n'})$ to $(\mathcal{E}_{n'+1}, \mathcal{S}_{n'+1}, \mathcal{P}_{n'+1}, \sigma_{n'+1}, \mathcal{L}_{n'+1})$, 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 $f \colon \mathbb{N}_n \rightarrow \mathbb{N}_{n'+1}$ 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 $\mathsf{!K}$-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 $E_{m+1}, \dots, E_{n-1} = \emptyset$, 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 $\mathsf{state}_p(\tilde{t})$.

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:

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

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 $ri \in ginsts(R)$.

Case: $R = \text{INIT}$ or $R \in \text{MD} \setminus \{\, \text{MDIN} \,\}$. In this case, $\emptyset \xrightarrow{E_1} \ldots \xrightarrow{E_n} S_n$ is not a well-formed msr execution.

Case: $R = \text{MDIN}$. Let $t \in \mathcal{M}$ such that $ri = R\tau = !\mathsf{K}(t) \,-[\, K(t) \,]\!\rightarrow \mathsf{In}(t)$.

From the induction hypothesis, and since $E_{m+1}, \ldots, E_n = \emptyset$, we have that

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

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

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

Let $\tilde{r} = \{\, a \in FN \mid RepNonce(a) \in_E \bigcup_{1 \leq j \leq n} F_j \,\}$. Then, by Lemma 8 and Lemma 9, we have that $\nu\mathcal{E}_{n'}, \tilde{r}.\sigma_{n'} \vdash t$. Therefore, $\nu\mathcal{E}_{n'}.\sigma_{n'} \vdash t$. This allows us to chose the following transition:

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

with $(\mathcal{E}_{n'+1}, \mathcal{S}_{n'+1}, \mathcal{P}_{n'+1}, \sigma_{n'+1}, \mathcal{L}_{n'+1}) = (\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{P}_{n'}, \sigma_{n'}, \mathcal{L}_{n'})$.

Conditions 1 to 7 hold trivially.

Case: $ri = [\mathsf{state}_p(\tilde{t})] \,-[]\!\rightarrow\, []$ (for some p and $\tilde{t}$). By induction hypothesis, we have $\mathcal{P}_{n'} \leftrightsquigarrow_P S_m$, and thus, as previously established, $\mathcal{P}_{n'} \leftrightsquigarrow_P S_{n-1}$. Let $Q \in^{\#} \mathcal{P}_{n'}$ such that $Q \leftrightsquigarrow_P \mathsf{state}_p(\tilde{t})$. Let θ be a grounding substitution for $\mathsf{state}_p(\tilde{x}) \in prems(\llbracket P \rrbracket_{=p})$ 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 22).

From the form of the rule R, and since $Q = P|_p\tau\rho$, we can deduce that $Q = \mathsf{0}$.

We therefore chose the following transition:

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

with $\mathcal{E}_{n'+1} = \mathcal{E}_{n'}$, $\mathcal{S}_{n'+1} = \mathcal{S}_{n'}$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^{\#} \{\, \mathsf{0} \,\}^{\#}$, $\sigma_{n'+1} = \sigma_{n'}$ and $\mathcal{L}_{n'+1} = \mathcal{L}_{n'}$.

We define f as on page 62. Therefore, Conditions 1 to 7 hold for $i < n-1$. It is left to show that Conditions 1 to 7 hold for n.

Condition 3 holds since $Q \leftrightarrow \mathsf{state}_p(\tilde{t})$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^{\#} \{\, \mathsf{0} \,\}^{\#}$ and $S_n = S_{n-1} \setminus^{\#} \{\, \mathsf{state}_p(\tilde{t}) \,\}^{\#}$. Conditions 1, 2, 4 and 6 hold trivially.

Case: $ri = [\mathsf{state}_p(\tilde{t})] \,-[]\!\rightarrow\, [\mathsf{state}_{p\cdot 1}(\tilde{t}), \mathsf{state}_{p\cdot 2}(\tilde{t})]$ (for some p and $\tilde{t}$). By induction hypothesis, we have $\mathcal{P}_{n'} \leftrightsquigarrow_P S_m$, and thus, as previously established, $\mathcal{P}_{n'} \leftrightsquigarrow_P S_{n-1}$. Let $Q \in^{\#} \mathcal{P}_{n'}$ such that $Q \leftrightsquigarrow_P \mathsf{state}_p(\tilde{t})$. Let θ be a grounding substitution for $\mathsf{state}_p(\tilde{x}) \in prems(\llbracket P \rrbracket_{=p})$ 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 22).

From the form of the rule R, and since $Q = P|_p\tau\rho$, we can deduce that $Q = Q_1|Q_2$, for some processes $Q_1 = P|_{p\cdot 1}\tau\rho$ and $Q_2 = P|_{p\cdot 2}\tau\rho$.

We therefore chose the following transition:

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

with $\mathcal{E}_{n'+1} = \mathcal{E}_{n'}$, $\mathcal{S}_{n'+1} = \mathcal{S}_{n'}$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^{\#} \{ Q_1 \mid Q_2 \}^{\#} \cup^{\#} \{ Q_1, Q_2 \}^{\#}$, $\sigma_{n'+1} = \sigma_{n'}$ and $\mathcal{L}_{n'+1} = \mathcal{L}_{n'}$.

We define f as on page 62. 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 $Q_1 \leftrightarrow \mathsf{state}_{p\cdot 1}(\tilde{t})$ and $Q_2 \leftrightarrow \mathsf{state}_{p\cdot 2}(\tilde{t})$. Therefore, and since $Q \leftrightarrow \mathsf{state}_p(\tilde{t})$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^{\#} \{ Q_1 \mid Q_2 \}^{\#} \cup^{\#} \{ Q_1, Q_2 \}^{\#}$, and $S_n = S_{n-1} \setminus^{\#} \{ \mathsf{state}_p \tilde{t} \}^{\#} \cup^{\#} \{ \mathsf{state}_{p\cdot 1}(\tilde{t}), \mathsf{state}_{p\cdot 2}(\tilde{t}) \}^{\#}$, Condition 3 holds.

Conditions 1, 2, 4 and 6 hold trivially.

Case: $ri = [!\mathsf{state}_p(\tilde{t})] \; -[]\rightarrow \; [\mathsf{state}_{p\cdot 1}(\tilde{t})]$ (for some $p, \tilde{t}$). By induction hypothesis, we have $\mathcal{P}_{n'} \leftrightsquigarrow_P S_m$, and thus, as previously established, $\mathcal{P}_{n'} \leftrightsquigarrow_P S_{n-1}$. Let $Q \in^{\#} \mathcal{P}_{n'}$ such that $Q \leftrightsquigarrow_P \mathsf{state}_p(\tilde{t})$. Let θ be a grounding substitution for $\mathsf{state}_p(\tilde{x}) \in \mathit{prems}([\![P]\!]_{=p})$ 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 22).

From the form of the rule R, 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:

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

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 62. 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 and 6 hold trivially.

Case: $ri = [\mathsf{state}_p(\tilde{t}), \mathsf{Fr}(a' \colon \mathit{fresh})] \; -[\; \mathit{ProtoNonce}(a' \colon \mathit{fresh}) \;]\rightarrow \; [\mathsf{state}_{p\cdot 1}(\tilde{t}, a' \colon \mathit{fresh})]$ (for some $p, \tilde{t}$ and $a' \in FN$). By induction hypothesis, we have $\mathcal{P}_{n'} \leftrightsquigarrow_P S_m$, and thus, as previously established, $\mathcal{P}_{n'} \leftrightsquigarrow_P S_{n-1}$. Let $Q \in^{\#} \mathcal{P}_{n'}$ such that $Q \leftrightsquigarrow_P \mathsf{state}_p(\tilde{t})$. Let θ be a grounding substitution for $\mathsf{state}_p(\tilde{x}) \in \mathit{prems}([\![P]\!]_{=p})$ 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 22).

From the form of the rule R, and since $Q = P|_p\tau\rho$, we can deduce that $Q = \nu\, a; Q'$ for a name $a \in FN$ and a process $Q' = P|_{p\cdot 1}\tau\rho$.

By definition of exec^{msr}, the fact $\mathsf{Fr}(a')$ can only be produced once. Since this fact is linear it can only be consumed once. Every rule in $[\![P]\!]$ that produces a label $\mathit{ProtoNonce}(x)$ for some x consumes a fact $\mathsf{Fr}(x)$. Therefore,

$$a' \notin \{ a \in FN \mid \mathit{ProtoNonce}(a) \in_E \bigcup_{1 \le j \le n-1} E_j \}.$$

The induction hypothesis allows us to conclude that $a' \notin \mathcal{E}_{n'}$,i. e., a' is fresh. We therefore chose the following transition:

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

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

We define f as on page 62. 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 $[\![P]\!]$, $state_{p\cdot 1}(\tilde{x}, a) \in prems(R')$ for an $R' \in [\![P]\!]_{=p\cdot 1}$. We can choose $\theta' := \theta[n_a \mapsto a']$ and have $\mathsf{state}_{p\cdot 1}(\tilde{t}, a') = \mathsf{state}_{p\cdot 1}(\tilde{x}, a)\theta'$. Since $Q = P|_p \tau\rho$ for τ and ρ induced by θ, $Q'\{ {}^{a'}/_{a} \} = P|_p \tau'\rho'$ for τ' and ρ' induced by θ', i. e., $\tau' = \tau$ and $\rho' = \rho[a \mapsto a']$. Therefore, $Q'\{ {}^{a'}/_{a} \} \leftrightsquigarrow_P \mathsf{state}_{p\cdot 1}(\tilde{t}, a')$.

Condition 3 holds, since furthermore $\nu\, a'; \mathsf{Q}' \leftrightarrow \mathsf{state}_p(\tilde{t})$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^{\#} \{ \nu\, a'; \mathsf{Q}' \}^{\#} \cup^{\#} \{ Q'\{ {}^{a'}/_{a} \} \}^{\#}$, and $S_n = S_{n-1} \setminus^{\#} \{ \mathsf{Fr}(a), \mathsf{state}_p(\tilde{t}) \}^{\#} \cup^{\#} \mathsf{state}_{p\cdot 1}(\tilde{t}, a\colon fresh)$.

Condition 1, holds since $\mathcal{E}_{n'+1} = \mathcal{E}_{n'} \cup a'$, and $E_n = ProtoNonce(a')$. Condition 6 holds since $ProtoNonce(a) \in \mathcal{F}_{res}$.

Conditions 2 and 4 hold trivially.

Case: $ri = [\mathsf{state}_p(\tilde{t}), \mathsf{In}(t_1)] \; -[\; \mathrm{InEvent}(t_1) \;]\to \; [\mathsf{state}_{p\cdot 1}(\tilde{t}), \mathsf{Out}(t_2)]$ (for some $p, \tilde{t}$ and $t_1, t_2 \in \mathcal{M}$). Since the msr execution is normal, we have that $S_{n-2} \xrightarrow{K(t_1)}_{\text{MDIN}} S_{n-1}$. Since $S_0 \xrightarrow{E_1}_{[\![P]\!]} \dots \xrightarrow{E_n}_{[\![P]\!]} S_n$ is normal, so is $S_0 \xrightarrow{E_1}_{[\![P]\!]} \dots \xrightarrow{E_{n-1}}_{[\![P]\!]} S_{n-1}$, and therefore $S_0 \xrightarrow{E_1}_{[\![P]\!]} \dots \xrightarrow{E_{n-2}}_{[\![P]\!]} S_{n-2}$. Hence there is an $m < n-2$ such $S_0 \xrightarrow{E_1}_{[\![P]\!]} \dots \xrightarrow{E_m}_{[\![P]\!]} S_m$ is a normal trace and $S_m \to_R^* S_{n-1}$ for $R = \{ \text{MDOUT}, \text{MDPUB}, \text{MDFRESH}, \text{MDAPPL}, \text{FRESH} \}$.

By induction hypothesis, we have $\mathcal{P}_{n'} \leftrightsquigarrow_P S_m$, and thus, since $\{ \text{MDOUT}, \text{MDPUB}, \text{MDFRESH}, \text{MDAPPL} \}$and FRESH do not add or remove state-facts, $\mathcal{P}_{n'} \leftrightsquigarrow_P S_{n-2}$. Let $Q \in^{\#} \mathcal{P}_{n'}$ such that $Q \leftrightsquigarrow_P \mathsf{state}_p(\tilde{t})$. Let θ be a grounding substitution for $state(\tilde{x}) \in prems([\![P]\!]_{=p})$ 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 22).

From the form of the rule R, and since $Q = P|_p \tau\rho$, we can deduce that $Q = \mathrm{out}\ (t_1, t_2); Q'$ for a process $Q' = P|_{p\cdot 1} \tau\rho$.

From the induction hypothesis, and since $E_{m+1}, \dots, E_{n-2} = \emptyset$, we have that

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

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

$$\{\, x\sigma_{n'} \mid x \in \mathbf{D}(\sigma_{n'}) \,\}^{\#} =_E \{\, \mathsf{Out}(t) \in \cup_{k \le n-2} S_k \,\}^{\#}. \tag{3}$$

Let $\tilde{r} = \{\, a : fresh \mid RepNonce(a) \in \bigcup_{1 \le j \le n-2} F_j \,\}$. Since $!\mathsf{K}(t_1) \in prems(\text{MDIN}\sigma)$ for $\sigma(x) = t_1$, we have $!\mathsf{K}(t_1) \in_E S_{n-2}$. By Lemma 8 and Lemma 9, we have $\nu\mathcal{E}_{n'}, \tilde{r}.\sigma_{n'} \vdash t$. Therefore, $\nu\mathcal{E}_{n'}.\sigma_{n'} \vdash t$. We chose the following transition:

$$\dots \xrightarrow{F'_n} (\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{P}_{n'}, \sigma_{n'}, \mathcal{L}_{n'}) \xrightarrow{K(t_1)} (\mathcal{E}_{n'+1}, \mathcal{S}_{n'+1}, \mathcal{P}_{n'+1}, \sigma_{n'+1}, \mathcal{L}_{n'+1})$$

with $\mathcal{E}_{n'+1} = \mathcal{E}_{n'}$, $\mathcal{S}_{n'+1} = \mathcal{S}_{n'}$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^{\#} \{\, \mathrm{out}\ (t_1, t_2); Q' \,\}^{\#} \cup^{\#} \{\, Q' \,\}^{\#}$, $\sigma_{n'+1} = \sigma_{n'} \cup \{ {}^{t_2}/_{x} \}$ and $\mathcal{L}_{n'+1} = \mathcal{L}_{n'}$ for a fresh x.

We define f as follows:

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

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

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 $E_{n-1} = K(t_1)$.

Condition 4 holds since $\sigma_{n'+1} = \sigma_{n'} \cup \{ {}^{t_2}/_x \}$, and therefore:

$$\begin{aligned} \{ x\sigma_{n'+1} \mid x \in \mathbf{D}(\sigma_{n'+1}) \}^\# &= \{ x\sigma_{n'} \mid x \in \mathbf{D}(\sigma_{n'}) \}^\# \cup^\# \{ t_2 \}^\# \\ &=_E \{ \mathsf{Out}(t) \in \cup_{k\le n-2} S_k \}^\# \cup^\# \{ t_2 \}^\# \qquad \text{(by (4))} \\ &= \{ \mathsf{Out}(t) \in \cup_{k\le n} S_k \}^\# \end{aligned}$$

By definition of $[\![P]\!]$ and $[\![P]\!]_{=p}$, we have that $Q' \leftrightsquigarrow_P \mathsf{state}_{p\cdot 1}(\tilde{t})$. Therefore, and since we have that $\mathrm{out}\ (t_1,t_2); Q' \leftrightsquigarrow_P \mathsf{state}_p(\tilde{t})$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^\# \{ \mathrm{out}\ (t_1,t_2); Q' \}^\# \cup^\# \{ Q' \}^\#$, and $S_n =_E S_{n-1} \setminus^\# \{ \mathsf{In}(a), \mathsf{state}_p(\tilde{t}) \}^\# \cup^\# \{ \mathsf{state}_{p\cdot 1}(\tilde{t}), \mathsf{Out}(t_2) \}$, Condition 3 holds.

Conditions Condition 1, 2 hold trivially.

Case: $ri = [\mathsf{state}_p(\tilde{t}), \mathsf{In}(\langle t_1,t_2\rangle)] \ -[\ \mathrm{InEvent}(\langle t_1,t_2\rangle)\ ]\to\ [\mathsf{state}_{p\cdot 1}(\tilde{t},\tilde{t}')]$ (for some $p, \tilde{t}, \tilde{t}'$ and $t_1, t_2 \in \mathcal{M}$). Since the msr execution is normal, we have that $S_{n-2} \xrightarrow{K(t_1)}_{\text{MDIN}} S_{n-1}$. Since $S_0 \xrightarrow{E_1}_{[\![P]\!]} \ldots \xrightarrow{E_n}_{[\![P]\!]} S_n$ is normal, so is $S_0 \xrightarrow{E_1}_{[\![P]\!]} \ldots \xrightarrow{E_{n-1}}_{[\![P]\!]} S_{n-1}$, and therefore we have that $S_0 \xrightarrow{E_1}_{[\![P]\!]} \ldots \xrightarrow{E_{n-2}}_{[\![P]\!]} S_{n-2}$. Hence there is an $m < n-2$ such $S_0 \xrightarrow{E_1}_{[\![P]\!]} \ldots \xrightarrow{E_m}_{[\![P]\!]} S_m$ is a normal trace and $S_m \to_R^* S_{n-1}$ for $R = \{ \text{MDOUT}, \text{MDPUB}, \text{MDFRESH}, \text{MDAPPL}, \text{FRESH} \}$.

By induction hypothesis, we have $\mathcal{P}_{n'} \leftrightsquigarrow_P S_m$. Since $\{ \text{MDOUT}, \text{MDPUB}, \text{MDFRESH}, \text{MDAPPL} \}$, FRESH and MDIN do not add or remove state-facts, $\mathcal{P}_{n'} \leftrightsquigarrow_P S_{n-2}$. 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 22). From the form of the rule R, and since $Q = P|_p\tau\rho$, we can deduce that $Q = \mathrm{in}\ (t_1, N); Q'$, for N a term that is not necessarily ground, and a process $Q' = P|_{p\cdot 1}\tau\rho$. Since $ri \in_E ginsts(R)$, we have that there is a substitution τ' such that $N\tau' =_E t_2$.

From the induction hypothesis, and since $E_{m+1},\ldots,E_{n-2} = \emptyset$, we have that

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

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

$$\{ x\sigma_{n'} \mid x \in \mathbf{D}(\sigma_{n'}) \}^\# = \{ \mathsf{Out}(t) \in \cup_{k\le n-2} S_k \}^\#. \tag{4}$$

Let $\tilde{r} = \{\, a : \mathit{fresh} \mid \mathit{RepNonce}(a) \in \bigcup_{1\le j\le n-2} F_j \,\}$. Since $!\mathsf{K}(\langle t_1, t_2\rangle) \in \mathit{prems}(\text{MDIN}\sigma)$ for $\sigma(x) = \langle t_1, t_2\rangle$, we have $!\mathsf{K}(\langle t_1, t_2\rangle)_E \in S_{n-2}$. By Lemma 8 and Lemma 9, we have $\nu\mathcal{E}_{n'}, \tilde{r}.\sigma_{n'} \vdash \langle t_1, t_2\rangle$. Therefore, $\nu\mathcal{E}_{n'}.\sigma_{n'} \vdash \langle t_1, t_2\rangle$. Using DEQ and DAPPL with the function symbols fst and snd, we have $\nu\mathcal{E}_{n'}.\sigma_{n'} \vdash t_1$ and $\nu\mathcal{E}_{n'}.\sigma_{n'} \vdash t_2$. Therefore, we chose the following transition:

$$\cdots \xrightarrow{F'_n} (\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{P}_{n'}, \sigma_{n'}, \mathcal{L}_{n'}) \xrightarrow{K(t_1)} (\mathcal{E}_{n'+1}, \mathcal{S}_{n'+1}, \mathcal{P}_{n'+1}, \sigma_{n'+1}, \mathcal{L}_{n'+1})$$

with $\mathcal{E}_{n'+1} = \mathcal{E}_{n'}$, $\mathcal{S}_{n'+1} = \mathcal{S}_{n'}$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^{\#} \{\, \text{in}\,(t_1, N); Q' \,\}^{\#} \cup^{\#} \{\, Q'\tau' \,\}^{\#}$, $\sigma_{n'+1} = \sigma_{n'}$ and $\mathcal{L}_{n'+1} = \mathcal{L}_{n'}$.

We define f as follows:

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

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

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 $E_{n-1} = K(t_1)$.

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.

Conditions Condition 1, 2 and 4 hold trivially.

Case: $ri = [\mathsf{state}_p^{\mathsf{semi}}(\tilde{s}), \mathsf{Ack}(t_1, t_2)] \,-[]\!\!\rightarrow\, [\mathsf{state}_{p\cdot 1}(\tilde{s})]$ (for some $p, \tilde{t}$ and $t_1, t_2 \in \mathcal{M}$). Since the msr execution is normal, we have that there p,q,$\tilde{x}, \tilde{y}, \tilde{y'}$ such that:

$$S_{n-3} \rightarrow_{R_1} S_{n-2} \rightarrow_{R_2} S_{n-1} \rightarrow_{R_3} S_n \quad \text{, where:}$$

- $R_1 = [\mathsf{state}_p(\tilde{x})] \rightarrow [\mathsf{Msg}(t_1, t_2), \mathsf{state}_p^{\mathsf{semi}}(\tilde{x})]$
- $R_2 = [\mathsf{state}_q(\tilde{y}), \mathsf{Msg}(t_1, t_2)] \rightarrow [\mathsf{state}_{q\cdot 1}(\tilde{y} \cup \tilde{y'}), \mathsf{Ack}(t_1, t_2)]$
- $R_3 = [\mathsf{state}_p^{\mathsf{semi}}(\tilde{x}), \mathsf{Ack}(t_1, t_2)] \rightarrow [\mathsf{state}_{p\cdot 1}(\tilde{x})]$

.

Since in this case, there is a fact with symbol Ack removed from S_{n-1} to S_n, we have to apply a different argument to apply the induction hypothesis.

Since $\emptyset \xrightarrow{E_1}_{[\![P]\!]} \cdots \xrightarrow{E_n}_{[\![P]\!]} S_n \in \mathit{exec}^{msr}([\![P]\!])$ is normal, $n \geq 2$, and $t_1, t_2 \in \mathcal{M}$, $\mathsf{Ack}(t_1, t_2) \in (S_{n-1} \setminus^{\#} S_n)$, there exists $m \leq n-3$ such that $S_m \rightarrow_R^* S_{n-3}$ for $R = \{\, \text{MDOUT}, \text{MDPUB}, \text{MDFRESH}, \text{MDAPPL} \,\} \cup \text{FRESH}$ and $\emptyset \xrightarrow{E_1}_{[\![P]\!]} \cdots \xrightarrow{E_m}_{[\![P]\!]} S_m \in \mathit{exec}^{msr}([\![P]\!])$ is normal. 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 $\mathbb{N}_m \rightarrow \mathbb{N}_{n'}$ and an execution such that Conditions 1 to 7 hold. Let f_p be this function and note that $n' = f_p(m)$.

In the following case distinction, we extend the previous execution by one step from $(\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{P}_{n'}, \sigma_{n'}, \mathcal{L}_{n'})$ to $(\mathcal{E}_{n'+1}, \mathcal{S}_{n'+1}, \mathcal{P}_{n'+1}, \sigma_{n'+1}, \mathcal{L}_{n'+1})$, 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:

$$f(i) := \begin{cases} f_p(i) & \text{if } i \in \mathbb{N}_m \\ n' & \text{if } m < i \leq n-3 \\ n'+1 & \text{if } i = n \end{cases}$$

Since, $S_m \to_R^* S_n$ 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 $i \leq n-3$. Since $E_{m+1}, \ldots, E_{n-1} = \emptyset$, Condition 1, 2, 5 and 6 hold for all $i \leq n-3$.

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 $\mathsf{state}_p(\tilde{t})$. Since $\{\, \text{MDOUT}, \text{MDPUB}, \text{MDFRESH}, \text{MDAPPL} \,\}$ and FRESH do not add or remove state-facts, $\mathcal{P}_{n'} \leftrightsquigarrow_P S_{n-3}$. Let $P \in^{\#} \mathcal{P}_{n'}$ such that $P \leftrightsquigarrow_P \mathsf{state}_p(\tilde{s})$. Let $Q \in^{\#} \mathcal{P}_{n'}$ such that $Q \leftrightsquigarrow_P \mathsf{state}_q(\tilde{t})$.

Let θ' be a grounding substitution for $\mathsf{state}_q(\tilde{y}) \in \mathit{prems}(\llbracket P \rrbracket_{=q})$ such that $\tilde{t} =_E \tilde{y}\theta'$. Then θ' induces a substitution τ' and a bijective renaming ρ' for fresh, but not bound names (in Q) such that $P|_q\tau'\rho' = Q$ (see Definition 22).

From the form of the rules R_1 and R_3, and since $P =_E P|_p\tau\rho$, for τ and ρ induced by the grounding substitution for $\mathsf{state}_p(\tilde{x})$, we can deduce that $P =_E \text{out}\ t_1, t_2;\ P'$ for a process $P' = P|_{p\cdot 1}\tau\rho$. Similarly, from the form of R_2, we can deduce $Q =_E \text{in}\ (t_1, N); Q'$, for N a term that is not necessarily ground, and a process $Q' = P|_{q\cdot 1}\tau'\rho'$. Since $S_{n-2} \to_{R_2} S_{n-1}$, we have that there is a substitution τ^* such that $N\tau'\rho'\tau^* =_E t_2$ and $((\tilde{y} \cup \mathit{vars}(N)) \setminus \tilde{y})\tau^* =_E \tilde{t}'$, where $\tilde{t}'$ such that $\mathsf{state}_{q\cdot 1}(\tilde{t}, \tilde{t}') \in S_{n-1} \setminus^{\#} S_{n-2}$.

Given that $Q =_E \text{in}\ (t_1, N); Q'$ and $P =_E \text{out}\ t_1, t_2;\ P'$, have that $\mathcal{P}_{n'} = \mathcal{P}' \cup^{\#} \{\, \text{out}\ t_1, t_2;\ P', \text{in}\ (t_1', N); Q' \,\}^{\#}$ with $t_1 =_E t_1'$ and $t_2 =_E N\tau^*$. Therefore, we chose the following transition:

$$\cdots \xrightarrow{F'_n} (\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{P}_{n'}, \sigma_{n'}, \mathcal{L}_{n'}) \xrightarrow{K(t_1)} (\mathcal{E}_{n'+1}, \mathcal{S}_{n'+1}, \mathcal{P}_{n'+1}, \sigma_{n'+1}, \mathcal{L}_{n'+1})$$

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

Conditions 1 to 7 hold for $i \leq n-3$. It is left to show that Conditions 1 to 7 hold for n.

As established before, we have τ^* such that $N\tau'\rho'\tau^* =_E t_2$. Let $\mathsf{state}_q(\tilde{t}, \tilde{t}')$ be the state variable added to S_{n-1}. Then, $((\tilde{y} \cup \mathit{vars}(N)) \setminus \tilde{y})\tau^* = \tilde{t}'$. By definition of $\llbracket P \rrbracket_{=q}$, we have that $\mathsf{state}_{q\cdot 1}(\tilde{t}, \tilde{t}') \in \mathit{prems}(\mathit{ginsts}(P_{=p\cdot 1}))$ for a grounding substitution $\theta_{q\cdot 1} = \theta' \cdot \tau^*$. Since τ' and ρ' are induced by θ', $\theta_{q\cdot 1}$ induces $\tau \cdot \tau'$ and the same ρ. We have that $Q'\tau' = (P|_{q\cdot 1}\tau'\rho')\tau* = P|_{q\cdot 1}\tau\tau'\rho$ and therefore $Q'\tau^* \leftrightsquigarrow_P \mathsf{state}_{q\cdot 1}(\tilde{t}, \tilde{t}')$. Similarly, we have $P' \leftrightsquigarrow_P \mathsf{state}_{q\cdot 1}(\tilde{s})$. We conclude that Condition 3 holds.

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

Case: $ri = [\mathsf{state}_p(\tilde{t})] \dashv[\ \text{Pred}_{pr}(t_1, \ldots, t_l)\ ]\to [\mathsf{state}_{p\cdot 1}(\tilde{t})]$ (for some p, $t_1, \ldots, t_l \in \mathcal{M}$ and $\tilde{t}$). By induction hypothesis, we have $\mathcal{P}_{n'} \leftrightsquigarrow_P S_m$, and thus, as previously established, $\mathcal{P}_{n'} \leftrightsquigarrow_P S_{n-1}$. Let $Q \in^{\#} \mathcal{P}_{n'}$ such that $Q \leftrightsquigarrow_P \mathsf{state}_p(\tilde{t})$. Let θ be a grounding substitution for $\mathsf{state}_p(\tilde{x}) \in \mathit{prems}(\llbracket P \rrbracket_{=p})$ 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 22).

From the form of the rule R, and since $Q = P|_p\tau\rho$, we can deduce that $Q = \mathsf{if}\ pr(t_1, \ldots, t_l)\ \mathsf{then}\ Q_1\ \mathsf{else}\ Q_2$ for a process $Q' = P|_{p\cdot 1}\tau\rho$.

Since $[E_1, \ldots, E_n] \vDash \alpha$, and thus $[E_1, \ldots, E_m] \vDash \alpha_{pred}$, $\sigma_{pr} \left\{ {}^{t_1}/_{x_1}, \ldots, {}^{t_l}/_{x_l} \right\}$ is satisfied. We therefore chose the following transition:

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

with $\mathcal{E}_{n'+1} = \mathcal{E}_{n'}$, $\mathcal{S}_{n'+1} = \mathcal{S}_{n'}$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^{\#} \{ \mathsf{if}\ pr(t_1, \ldots, t_l)\ \mathsf{then}\ Q_1 \mathsf{else}\ Q'_2 \}^{\#} \cup^{\#} \{ Q_1 \}^{\#}$, $\sigma_{n'+1} = \sigma_{n'}$ and $\mathcal{L}_{n'+1} = \mathcal{L}_{n'}$.

We define f as on page 62. 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 $Q_1 \leftrightarrow \mathsf{state}_{p \cdot 1}(\tilde{t})$. Therefore, and since

$$\mathsf{if}\ pr(t_1, \ldots, t_l)\ \mathsf{then}\ Q_1\ \mathsf{else}\ Q_2 \leftrightarrow \mathsf{state}_p(\tilde{t})$$

we have that $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^{\#} \{ \mathsf{if}\ pr(t_1, \ldots, t_l)\ \mathsf{then}\ Q_1\ \mathsf{else}\ Q_2 \}^{\#} \cup^{\#} \{ Q_1 \}^{\#}$, and $S_n = S_{n-1} \setminus^{\#} \{ \mathsf{state}_p(\tilde{t}) \}^{\#} \cup^{\#} \{ \mathsf{state}_{p \cdot 1}(\tilde{t}) \}^{\#}$, Condition 3 holds. Conditions 1, 2, 4, 5 and 6 hold trivially.

Case: $ri = [\mathsf{state}_p(\tilde{t})] \ -[\ \mathrm{Pred\_not}_{pr}(t_1, \ldots, t_l)\ ]\rightarrow\ [\mathsf{state}_{p \cdot 1}(\tilde{t})]$ (for some $p, \tilde{t}$ and $t_1, \ldots, t_l \in \mathcal{M}$). In this case, the proof is almost the same as in the previous case, except that the predicate $\neg\sigma_{pr} \left\{ {}^{t_1}/_{x_1}, \ldots, {}^{t_l}/_{x_l} \right\}$ is satisfied, and thus $\sigma_{pr} \left\{ {}^{t_1}/_{x_1}, \ldots, {}^{t_l}/_{x_l} \right\}$ is not satisfied, Q_2 is chosen instead of Q_1 and and $S_n = S_{n-1} \setminus^{\#} \{ \mathsf{state}_p(\tilde{t}) \}^{\#} \cup^{\#} \{ \mathsf{state}_{p \cdot 2}(\tilde{t}) \}^{\#}$.

Case: $ri = [\mathsf{state}_p(\tilde{t})] \ -[\ \mathrm{F}, \mathrm{Event}()\ ]\rightarrow\ [\mathsf{state}_{p \cdot 1}(\tilde{t})]$ (for some $p, \tilde{t}$). This is a special case of the case where $ri = [\mathsf{state}_p(\tilde{t}), l] \ -[\ a\ ]\rightarrow\ [\mathsf{state}_{p \cdot 1}(\tilde{t}), r]$ for $l = r = \emptyset$ and $a = F$.

Case: $ri = [\mathsf{state}_p(\tilde{t})] \ -[\ Insert(t_1, t_2)\ ]\rightarrow\ [\mathsf{state}_{p \cdot 1}(\tilde{t})]$ (for some $p, \tilde{t}$ and $t_1, t_2 \in \mathcal{M}$). By induction hypothesis, we have $\mathcal{P}_{n'} \leftrightsquigarrow_P S_m$, and thus, as previously established, $\mathcal{P}_{n'} \leftrightsquigarrow_P S_{n-1}$. 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 22).

From the form of the rule R, and since $Q = P|_p \tau\rho$, we can deduce that $Q = \mathtt{insert}\ t_1, t_2; Q'$ for a process $Q' = P|_{p \cdot 1} \tau\rho$.

We therefore chose the following transition:

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

with $\mathcal{E}_{n'+1} = \mathcal{E}_{n'}$, $\mathcal{S}_{n'+1} = \mathcal{S}_{n'}[t_1 \mapsto t_2]$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^{\#} \{ \mathtt{insert}\ t_1, t_2; Q' \}^{\#} \cup^{\#} \{ Q' \}^{\#}$, $\sigma_{n'+1} = \sigma_{n'}$ and $\mathcal{L}_{n'+1} = \mathcal{L}_{n'}$.

We define f as on page 62. 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 $Q' \leftrightarrow \mathsf{state}_{p \cdot 1}(\tilde{t})$. Therefore, and since $\mathtt{insert}\ t_1, t_2; Q' \leftrightarrow \mathsf{state}_p(\tilde{t})$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^{\#} \{ \mathtt{insert}\ t_1, t_2; Q' \}^{\#} \cup^{\#} \{ Q' \}^{\#}$, and $S_n = S_{n-1} \setminus^{\#} \{ \mathsf{state}_p(\tilde{t}) \}^{\#} \cup^{\#} \{ \mathsf{state}_{p \cdot 1}(\tilde{t}) \}^{\#}$, Condition 3 holds.

Condition 2 holds, since $E_n = Insert(t_1, t_2)$ is the last element of the trace.

Conditions 1, 4, 5 and 6 hold trivially.

Case: $ri = [\mathsf{state}_p(\tilde{t})] \,-[\, Delete(t_1, t_2) \,]\to\, [\mathsf{state}_{p\cdot 1}(\tilde{t})]$ (for some $p, \tilde{t}$ and $t_1, t_2 \in \mathcal{M}$). By induction hypothesis, we have $\mathcal{P}_{n'} \leftrightsquigarrow_P S_m$, and thus, as previously established, $\mathcal{P}_{n'} \leftrightsquigarrow_P S_{n-1}$. 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 22).

From the form of the rule R, and since $Q = P|_p\tau\rho$, we can deduce that $Q = \mathtt{delete}\ t_1; Q'$ for a process $Q' = P|_{p\cdot 1}\tau\rho$.

We therefore chose the following transition:

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

with $\mathcal{E}_{n'+1} = \mathcal{E}_{n'}$, $\mathcal{S}_{n'+1} = \mathcal{S}_{n'}[t_1 \mapsto t_2]$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^{\#} \{\, \mathtt{delete}\ t_1; Q' \,\}^{\#} \cup^{\#} \{\, Q' \,\}^{\#}$, $\sigma_{n'+1} = \sigma_{n'}$ and $\mathcal{L}_{n'+1} = \mathcal{L}_{n'}$.

We define f as on page 62. 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 $Q' \leftrightarrow \mathsf{state}_{p\cdot 1}(\tilde{t})$. Therefore, and since $\mathtt{delete}\ t_1; Q' \leftrightarrow \mathsf{state}_p(\tilde{t})$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^{\#} \{\, \mathtt{delete}\ t_1; Q' \,\}^{\#} \cup^{\#} \{\, Q' \,\}^{\#}$, and $S_n = S_{n-1} \setminus^{\#} \{\, \mathsf{state}_p(\tilde{t}) \,\}^{\#} \cup^{\#} \{\, \mathsf{state}_{p\cdot 1}(\tilde{t}) \,\}^{\#}$, Condition 3 holds.

Condition 2 holds, since $E_n = Delete(t_1, t_2)$ is the last element of the trace.

Conditions 1, 4, 5 and 6 hold trivially.

Case: $ri = [\mathsf{state}_p(\tilde{t})] \,-[\, IsIn(t_1, t_2) \,]\to\, [\mathsf{state}_{p\cdot 1}(\tilde{t}, t_2)]$ (for some $p, \tilde{t}$ and $t_1, t_2 \in \mathcal{M}$). By induction hypothesis, we have $\mathcal{P}_{n'} \leftrightsquigarrow_P S_m$, and thus, as previously established, $\mathcal{P}_{n'} \leftrightsquigarrow_P S_{n-1}$. 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 22).

From the form of the rule R, and since $Q = P|_p\tau\rho$, we can deduce that $Q = \mathtt{lookup}\ t_1\ \mathtt{as}\ v\ \mathtt{in}\ Q_1\ \mathtt{else}\ Q_2$ for some variable V, and two processes $Q_1 = P|_{p\cdot 1}\tau\rho$ and $Q_2 = P|_{p\cdot 2}\tau\rho$.

Since $[E_1, \ldots, E_n] \vDash \alpha_{in}$, there is an $i < n$ such that $Insert(t_1, t_2) \in_E E_i$ and there is no j such that $i < j < n$ and $Delete(t_1) \in_E E_j$ or and $Insert(t_1, t_2) \in_E TE_j$. Since $E_m, \ldots, E_n = \emptyset$, we know that $i < m$. Hence, by induction hypothesis, $\mathcal{S}_{n'}(t_1) = t_2$. We therefore chose the following transition:

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

with $\mathcal{E}_{n'+1} = \mathcal{E}_{n'}$, $\mathcal{S}_{n'+1} = \mathcal{S}_{n'}$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^{\#} \{\, \mathtt{lookup}\ t_1\ \mathtt{as}\ v\ \mathtt{in}\ Q_1\ \mathtt{else}\ Q_2 \,\}^{\#} \cup^{\#} \{\, Q_1\{\,^{t_2}/_v\,\} \,\}^{\#}$, $\sigma_{n'+1} = \sigma_{n'}$ and $\mathcal{L}_{n'+1} = \mathcal{L}_{n'}$.

We define f as on page 62. 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 $Q_1\{\,^{v}/_{t_2}\,\} \leftrightarrow \mathsf{state}_{p\cdot 1}(\tilde{t}, t_2)$ (for $\tau' = \tau[v \mapsto t_2]$ and $\rho' = \rho$). Therefore, and since $\mathtt{lookup}\ t_1\ \mathtt{as}\ v\ \mathtt{in}\ Q_1\ \mathtt{else}\ Q_2 \leftrightarrow \mathsf{state}_p(\tilde{t})$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^{\#} \{\, \mathtt{lookup}\ t_1\ \mathtt{as}\ v\ \mathtt{in}\ Q_1\ \mathtt{else}\ Q_2 \,\}^{\#} \cup^{\#} \{\, Q' \,\}^{\#}$, and $S_n = \S_{n-1} \setminus^{\#} \{\, \mathsf{state}_p(\tilde{t}) \,\}^{\#} \cup^{\#} \{\, \mathsf{state}_{p\cdot 1}(\tilde{t}, t_2) \,\}^{\#}$, Condition 3 holds.

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

Case: $ri = [\mathsf{state}_p(\tilde{t})] \; -[\; \mathit{IsNotSet}(t_1) \;]\to \; [\mathsf{state}_{p{\cdot}2}(\tilde{t})]$ (for some $p, \tilde{t}$ and $t_1 \in \mathcal{M}$). By induction hypothesis, we have $\mathcal{P}_{n'} \leftrightsquigarrow_P S_m$, and thus, as previously established, $\mathcal{P}_{n'} \leftrightsquigarrow_P S_{n-1}$. Let $Q \in^{\#} \mathcal{P}_{n'}$ such that $Q \leftrightsquigarrow_P \mathsf{state}_p(\tilde{t})$. Let θ be a grounding substitution for $\mathsf{state}_p(\tilde{x}) \in \mathit{prems}([\![P]\!]_{=p})$ 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 22).

From the form of the rule R, and since $Q = P|_p\tau\rho$, we can deduce that $Q = \texttt{lookup}\ t_1\ \texttt{as}\ v\ \texttt{in}\ Q_1\ \texttt{else}\ Q_2$ for a variable v and two processes $Q_1 = P|_{p{\cdot}1}\tau\rho$ and $Q_2 = P|_{p{\cdot}2}\tau\rho$.

Since $[E_1, \ldots, E_n] \vDash \alpha_{notin}$, there is no $i < n$ such that $\mathit{Insert}(t_1, t_2) \in_E E_i$ and there is no j such that $i < j < n$ and $\mathit{Delete}(t_1) \in_E E_j$ or and $\mathit{Insert}(t_1, t_2) \in_E TE_j$. Since $E_m, \ldots, E_n = \emptyset$, we know that holds $j < m$. Hence, by induction hypothesis, $\mathcal{S}_{n'}(t_1)$ is undefined. We therefore chose the following transition:

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

with $\mathcal{E}_{n'+1} = \mathcal{E}_{n'}$, $\mathcal{S}_{n'+1} = \mathcal{S}_{n'}$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^{\#} \{\, \texttt{lookup}\ t_1\ \texttt{as}\ v\ \texttt{in}\ Q_1\ \texttt{else}\ Q_2 \,\}^{\#} \cup^{\#} \{\, Q_2 \,\}^{\#}$, $\sigma_{n'+1} = \sigma_{n'}$ and $\mathcal{L}_{n'+1} = \mathcal{L}_{n'}$.

We define f as on page 62. 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 $Q_2 \leftrightarrow \mathsf{state}_{p{\cdot}2}(\tilde{t})$. Therefore, and since $\texttt{lookup}\ t_1\ \texttt{as}\ v\ \texttt{in}\ Q_1\ \texttt{else}\ Q_2 \leftrightarrow \mathsf{state}_p(\tilde{t})$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^{\#} \{\, \texttt{lookup}\ t_1\ \texttt{as}\ v\ \texttt{in}\ Q_1\ \texttt{else}\ Q_2 \,\}^{\#} \cup^{\#} \{\, Q_2 \,\}^{\#}$, and $S_n = \S_{n-1} \setminus^{\#} \{\, \mathsf{state}_p(\tilde{t}) \,\}^{\#} \cup^{\#} \{\, \mathsf{state}_{p{\cdot}2}(\tilde{t}) \,\}^{\#}$, Condition 3 holds.

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

Case: $ri = [\mathsf{state}_p(\tilde{t}), \mathsf{Fr}(\mathit{lock}_l)] \; -[\; \mathit{Lock}(\mathit{lock}_l, t) \;]\to \; [\mathsf{state}_{p{\cdot}1}(\tilde{t}, \mathit{lock}_l)]$ (for some $p, \tilde{t}$, $\mathit{lock}_l \in \mathit{FN}$ and $t \in \mathcal{M}$). By induction hypothesis, we have $\mathcal{P}_{n'} \leftrightsquigarrow_P S_m$, and thus, as previously established, $\mathcal{P}_{n'} \leftrightsquigarrow_P S_{n-1}$. Let $Q \in^{\#} \mathcal{P}_{n'}$ such that $Q \leftrightsquigarrow_P \mathsf{state}_p(\tilde{t})$. Let θ be a grounding substitution for $\mathsf{state}_p(\tilde{x}) \in \mathit{prems}([\![P]\!]_{=p})$ 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 22).

From the form of the rule R, and since $Q = P|_p\tau\rho$, we can deduce that $Q = \texttt{lock}^l\ t;\ Q'$ for $Q' = P|_{p{\cdot}1}\tau\rho$.

Since $[E_1, \ldots, E_n] \vDash \alpha_{lock}$, for every $i < n$ such that $\mathit{Lock}(l_p, t) \in_E E_i$, there a j such that $i < j < n$ and $\mathit{Unlock}(l_p, t) \in_E E_j$, and in between i and j, there is no lock or unlock, i. e., for all k such that $i < k < j$, and all l_i, $\mathit{Lock}(l_i, t) \notin_E E_k$ and $\mathit{Unlock}(l_i, t) \notin_E E_k$.

Since $E_m, \ldots, E_n = \emptyset$, we know that this holds for $i < m$ and $j < m$ as well. By induction hypothesis, Condition 5, this implies that $t \notin_E \mathcal{L}_{n'}$. We therefore chose the following transition:

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

with $\mathcal{E}_{n'+1} = \mathcal{E}_{n'}$, $\mathcal{S}_{n'+1} = \mathcal{S}_{n'}$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^{\#} \{\, \texttt{lock}^l\ t;\ Q' \,\}^{\#} \cup^{\#} \{\, Q' \,\}^{\#}$, $\sigma_{n'+1} = \sigma_{n'}$ and $\mathcal{L}_{n'+1} = \mathcal{L}_{n'} \cup \{\, t \,\}$.

We define f as on page 62. 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 $Q' \leftrightarrow \mathsf{state}_{p{\cdot}1}(\tilde{t})$. Therefore, and since $\texttt{lock}^l\ t;\ Q' \leftrightarrow \mathsf{state}_p(\tilde{t})$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^{\#} \{\, \texttt{lock}^l\ t;\ Q' \,\}^{\#} \cup^{\#} \{\, Q' \,\}^{\#}$, and $S_n = \S_{n-1} \setminus^{\#} \{\, \mathsf{state}_p(\tilde{t}), \mathsf{Fr}(\mathit{lock}_l) \,\}^{\#} \cup^{\#} \{\, \mathsf{state}_{p{\cdot}1}(\tilde{t}, \mathit{lock}_l) \,\}^{\#}$, Condition 3 holds.

Condition 5 holds since $E_n = \{ \mathrm{Lock}(lock_l, t) \}^\#$ is added to the end of the trace.
Conditions 1, 2, 4 and 6 hold trivially.

Case: $ri = [\mathsf{state}_p(\tilde{t})] \; -[\; Unlock(n_l, t) \;]\to \; [\mathsf{state}_{p\cdot 1}(\tilde{t})]$ (for some $p, \tilde{t}$, $n_l \in FN$ and $t \in \mathcal{M}$). By induction hypothesis, we have $\mathcal{P}_{n'} \leftrightsquigarrow_P S_m$, and thus, as previously established, $\mathcal{P}_{n'} \leftrightsquigarrow_P S_{n-1}$. 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 22).

From the form of the rule R, and since $Q = P|_p\tau\rho$, we can deduce that $Q = \mathtt{unlock}^l\ t;\ Q'$ for $Q' = P|_{p\cdot 1}\tau\rho$.

We therefore chose the following transition:

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

with $\mathcal{E}_{n'+1} = \mathcal{E}_{n'}$, $\mathcal{S}_{n'+1} = \mathcal{S}_{n'}$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^\# \{ \mathtt{unlock}^l\ t;\ Q' \}^\# \cup^\# \{ Q' \}^\#$, $\sigma_{n'+1} = \sigma_{n'}$ and $\mathcal{L}_{n'+1} = \mathcal{L}_{n'} \setminus \{ t \}$.

We define f as on page 62. 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 $Q' \leftrightarrow \mathsf{state}_{p\cdot 1}(\tilde{t})$. Therefore, and since $\mathtt{unlock}^l\ t;\ Q' \leftrightarrow \mathsf{state}_p(\tilde{t})$, $\mathcal{P}_{n'+1} = \mathcal{P}_{n'} \setminus^\# \{ \mathtt{unlock}^l\ t;\ Q' \}^\# \cup^\# \{ Q' \}^\#$, and $S_n = \S_{n-1} \setminus^\# \{ \mathsf{state}_p(\tilde{t}) \}^\# \cup^\# \{ \mathsf{state}_{p\cdot 1}(\tilde{t}) \}^\#$, Condition 3 holds.

We show that Condition 5 holds for $\mathcal{L}_{n'+1} = \mathcal{L}_{n'} \setminus \{ t \}$: For all $t' \neq_E t$, $t' \in_E \mathcal{L}_{n'} \Leftrightarrow t' \in_E \mathcal{L}_{n'+1}$ by induction hypothesis. If $t \notin_E \mathcal{L}_{n'}$, then $\forall j \leq m, u.\mathrm{Lock}(u,t) \in_E E_j \to \exists j < k \leq n.\mathrm{Unlock}(u,t) \in_E E_k$. Since we have $E_m, \ldots, E_{n-1} = \emptyset$ and $E_n = \{ \mathrm{Unlock}(n_l, t) \}^\#$, we can strengthen this to $\forall j \leq n, u.\mathrm{Lock}(u,t) \in_E E_j \to \exists j < k \leq n.\mathrm{Unlock}(u,t) \in_E E_k$, which means that the condition holds in this case. If $t \in_E \mathcal{L}_{n'}$, then $\exists j \leq n, u.\mathrm{Lock}(u,t) \in_E E_j \wedge \forall j < k \leq n.\mathrm{Unlock}(u,t) \notin_E E_k$ and since $E_m, \ldots, E_{n-1} = \emptyset$ and $E_n = \{ \mathrm{Unlock}(n_l, t) \}^\#$, a contradiction to Condition 5 would constitute of j and $u \neq_E n_l$ such that $\mathrm{Lock}(u,t) \in_E E_j$ and $\forall j < k \leq n.\mathrm{Unlock}(u,t) \notin_E E_k$.

We will show that this leads to a contradiction with $[E_1, \ldots, E_n] \vDash \alpha$. Fix j and u. By definition of $[\![P]\!]$ and well-formedness of P, there is a p_l that is a prefix of p such that $P|_{q_l} = \mathtt{lock}^l t; Q''$ for the same annotation l and parameter t. The form of the translation guarantees that if $\mathsf{state}_p(\tilde{t}) \in S_n$, then for some $\tilde{t}'$ there is $i \leq n$ such that $\mathsf{state}_{p'}(\tilde{t}') \in S_i$, if p' is a prefix of p. We therefore have that there is $i < n$ such that $E_i =_E \{ \mathrm{Lock}(n_l, t) \}^\#$. We proceed by case distinction:
Case 1: $j < i$ (see Figure 24). Since $\forall j < k \leq n.\mathrm{Unlock}(u,t) \notin_E E_k$, $[E_1, \ldots, E_n] \not\vDash \alpha_{lock}$.

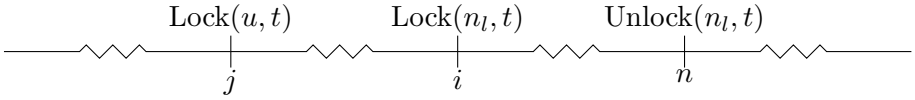

Fig. 24. Visualisation of Case 1.

Case 2: $i < j$ (see Figure 25). By definition of $\overline{P}$, there is no parallel and no replication between p_l and p. Note that any rule in $[\![P]\!]$ that produces a state named state_q for a non-empty q is such that it requires a fact with name $\mathsf{state}_{q'}$ for $q = q' \cdot 1$ or $q = q' \cdot 2$ (in case of the translation of out, it might require $\mathsf{state}^{\mathsf{semi}}_{q'}$, which in turn requires $\mathsf{state}_{q'}$). Therefore, there cannot be a second $k \neq n$ such that

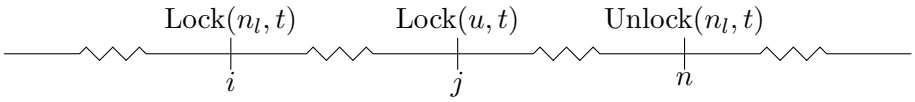

Fig. 25. Visualisation of Case 2.

$\mathrm{Unlock}(n_l, t) \in_E E_k$ (since n_l was added in a Fr-fact in to S_i). This means in particular that there is not k such that $i < k < n$ and $\mathrm{Unlock}(n_l, t) \in_E E_k$. Therefore, $[E_1, \ldots, E_n] \not\models \alpha_{lock}$.

Conditions 1, 2, 4 and 6 hold trivially.

□

B.4. Proof of Lemma 1: putting the pieces together

Lemma 1. *Let P be a well-formed ground process. We have that*

$$traces^{pi}(P) = hide(filter(traces^{msr}([\![P]\!]))).$$

Proof. From Lemma 10, we can conclude that

$$traces^{pi}(P) \subseteq \{\, hide(t) | tr \in traces^{msr}([\![P]\!]) \text{ and } tr \vDash \alpha \,\} = hide(filter(traces^{msr}([\![P]\!]))).$$

From Lemma 11, we have that

$$hide(filter(traces^{msr}([\![P]\!]))) = \{\, tr \in hide(filter(traces^{msr}([\![P]\!]))) \mid tr \text{ is normal} \,\}.$$

From Lemma 12, we can conclude that

$$\{\, tr \in hide(filter(traces^{msr}([\![P]\!]))) \mid tr \text{ is normal} \,\} \subseteq traces^{pi}(P),$$

and hence

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

□