THE UNIVERSITY *of* EDINBURGH

Edinburgh Research Explorer

StatVerif: Verification of stateful processes

Citation for published version:
Arapinis, M, Phillips, J, Ritter, E & Ryan, M 2014, 'StatVerif: Verification of stateful processes', *Journal of Computer Security*, vol. 22, no. 5, pp. 743-821. https://doi.org/10.3233/JCS-140501

Digital Object Identifier (DOI):
10.3233/JCS-140501

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Journal of Computer Security

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

OPEN ACCESS

Download date: 27. Apr. 2024

StatVerif: Verification of Stateful Processes

Myrto Arapinis Joshua Phillips Eike Ritter Mark D. Ryan
{m.d.arapinis, e.ritter, j.phillips, m.d.ryan}@cs.bham.ac.uk
School of Computer Science, University of Birmingham, UK

October 19, 2014

Abstract

We present StatVerif, which is an extension of the ProVerif process calculus with constructs for explicit state, in order to be able to reason about protocols that manipulate global state. Global state is required by protocols used in hardware devices (such as smart cards and the trusted platform module), as well as by protocols involving databases that store persistent information. We provide the operational semantics of StatVerif. We extend the ProVerif compiler to a compiler for StatVerif, which takes processes written in the extended process language and produces Horn clauses. Our compilation is carefully engineered to avoid many false attacks. We prove the correctness of the StatVerif compiler. We illustrate our method on two examples: a small hardware security device and a contract signing protocol. We are able to prove their desired properties automatically.

1 Introduction

Motivation Agents that engage in security protocols necessarily involve a notion of state. For example, a protocol may require an agent A to send a certain message, to receive a response, and then to send another message that depends on the response. In this case, A needs to maintain some state information so that it knows which step of the protocol is the next one. This notion of state is local within a session.

Sometimes, there is also a requirement to maintain longer-term global state. This state is not local to a session: if one session updates the state, then it is updated for other sessions too. Two broad classes of protocols where global state is relevant are:

- Protocols involving security device interfaces. This includes smartcards, stateful RFID chips, the *trusted platform module* (TPM), and secure co-processors such as the IBM 4758. Such devices maintain data including keys and their metadata, including whether keys are loaded, valid, or revoked. They may also have special registers for recording state information, such as monotonic counters or the platform configuration registers of the TPM. They may allow other data to be saved on the device, such as the identity of a stateful RFID tag, that affects its future behaviour.

- Protocols involving databases, such as protocols for RFID tags (where a database holds information about the status of tags), protocols for websites (e.g., where a database holds status about transactions, and browsers hold cookies), and key servers (where a database records the status of keys). It also includes specialised protocols such as fair exchange protocols and contract signing protocols, where a trusted party maintains the status of the exchange in a database.

This notion of global state poses a problem for existing protocol verification techniques, because those techniques often make abstractions that will introduce false attacks when state is considered. We show this in the running example, below. This has been noted before, e.g., by Herzog [1], Mödersheim [2], Guttman [3] and Delaune *et al.* [4]. For example, the ProVerif protocol analyser [5] models an ever-increasing set of derivable facts representing attacker knowledge, and is not able to associate those facts with the states in which they arose. For this reason, the tool typically reports many false attacks.

Related work The AVISPA tool [6] aims to handle mutable state via its OFMC, CL-AtSe and SATMC backends. However, the first two of these require concrete bounds to be given on the number of sessions and fresh nonces. SATMC can avoid this restriction in principle [7], but as we mentioned in [4], SATMC performed poorly in our experiments due to the relatively large message length, a known weakness of SATMC. Also aiming to address global states, Mödersheim [2] has developed a framework that takes global state into account. He introduces a low-level language called AIF, which extends the IF language of AVISPA by adding sets. The framework is based on a strong abstraction that identifies all objects that are in exactly the same sets. This method appears to work well, but the method is tightly coupled with a particular abstraction, and the scope of its applicability is not very clear. The author mentions the restrictions of the low-level and implementation-focused language, and points out the desirability of a high-level language for protocol designers. Guttman has also addressed the problem, by extending the strand space model with a notion of state [3]. However, this extended model does not currently have tool support. In a similar direction to ours, Delaune/Kremer/Ryan/Steel have coded stateful aspects of the TPM directly in Horn clauses [8].

Our approach and contributions We present StatVerif, which is an extension of the ProVerif process language with constructs that allow one to directly model global state. This approach allows us to build on ProVerif's existing successes. More precisely,

- We extend the ProVerif process calculus with explicit state, including assignments, and provide its operational semantics.

- We extend the ProVerif compiler, which takes processes written in the process language, and produces Horn clauses that can be verified by ProVerif. Our translation is carefully engineered to avoid the false attacks mentioned above.

- We prove the correctness of the extended compiler for secrecy properties; that is, we show that attacks are not lost. Therefore, a security proof at the clause level implies a proof at the process calculus level.
- We illustrate our method on two examples: a small hardware security device, and a contract signing protocol. We are able to prove their desired properties automatically.

We only consider secrecy properties, although ProVerif can also prove correspondence and equivalence properties. Full details of our code for the examples are available on the web, along with a ProVerif-based implementation[1].

Running example The following example allows us to explain our results more fully. Consider a hardware device whose behaviour can be configured by the user. The device generates a public key PK_k. A user Alice may use software to encrypt pairs (x, y) of secrets with PK_k, resulting in $\{(x, y)\}_{PK_k}$. Later, she can give the device and a set $\{\{(x_1, y_1)\}_{PK_k}, \ldots, \{(x_n, y_n)\}_{PK_k}\}$ of such ciphertexts to another user Bob. The device allows Bob to configure it as "left" or "right". If Bob chooses to configure it as "left", then after doing so he can use the device to obtain any of the first components x_i of the pairs. If he configures it with "right", then he gets to have the second components y_i. Such a device might, for example, be used to allow a customer to choose between vouchers for a music website, or vouchers for a social networking site, but not both.

We model such a device in our stateful language as the following process:

```
1   new $s$; new $k$;
2     let $PKk = \mathsf{pk}(k)$ in
3     $\mathsf{out}(c, PKk) \mid [s \mapsto \mathsf{init}] \mid$
4     ( ! lock $s$; $\mathsf{in}(c, x)$; read $s$ as $y$;
5         if $y = \mathsf{init}$ then
6           ( if $x = \mathsf{left}$ then $s := \mathsf{left}$; unlock $s$ else
7             if $x = \mathsf{right}$ then $s := \mathsf{right}$; unlock $s$ ) ) |
8     ( ! lock $s$; $\mathsf{in}(c, x)$; read $s$ as $y$; let $z = \mathsf{adec}(k, x)$ in
9         let $xl = \mathsf{projl}(z)$ in
10        let $xr = \mathsf{projr}(z)$ in
11        if $y = \mathsf{left}$ then $\mathsf{out}(c, xl)$; unlock $s$ else
12        if $y = \mathsf{right}$ then $\mathsf{out}(c, xr)$; unlock $s$ )
```

In line 3, we declare a cell s with initial value init. In lines 4–7, we allow the user to provide a value—either left or right—to configure the device; this assigns the value to the cell s. In lines 8–12, we allow the user to provide a ciphertext; in return, the user will receive the left or right component, according to the configuration. Notice that the device, once configured left or right, cannot be configured again.

Details of the constructs including lock will be explained later. We assume the usual equational theory for public key encryption. The desired property is that, given a ciphertext $\{(x, y)\}_{PK_k}$, the attacker cannot obtain the pair (x, y). This property is easily automatically proved using our techniques.

[1] http://markryan.eu/research/StatVerif/

It is interesting to note that it is possible to convert such a process into a semantically equivalent pure ProVerif process. The cell s could be represented by a private channel that stores the configuration value. The subprocesses that read the value s would instead input it from this private channel. However, although the private channel process is semantically equivalent to our process, ProVerif is not able to prove that it satisfies the desired property because, as mentioned, ProVerif's abstractions introduce false attacks. In particular, once init is placed on the private channel, it remains forever available. Therefore, in the private channel model, the device allows itself to be configured and re-configured at will. The user can obtain (x, y) by configuring it first as left and then as right. Our technique does not introduce for states the abstractions that ProVerif uses for private channels.

Outline We give some necessary background about ProVerif and Horn clauses in section II. In section III, we detail the syntax and semantics of our stateful language, and show how it is translated into clauses in section IV. We also prove the correctness of the translation. In section V, we treat the case studies.

2 Background

2.1 ProVerif process language

We start from the ProVerif process language introduced in [9], which we recall in the first half of Figure 1 (up to and including the conditional process). This language is similar to the applied pi calculus [10], and is designed to model security protocols. It allows processes to send terms built over a signature including names and variables. These terms model the messages that are exchanged during a protocol. Cryptographic operations are modelled by reductions such as

$$\begin{array}{rcl}
\mathsf{sdec}(x, \mathsf{senc}(x, y)) & \to & y \\
\mathsf{adec}(x, \mathsf{aenc}(\mathsf{pk}(x), y)) & \to & y \\
\mathsf{check\_getmsg}(\mathsf{pk}(x), \mathsf{sign}(x, y)) & \to & y \\
\mathsf{checkpcs}(ct, \mathsf{pk}(x), \mathsf{pk}(y), \mathsf{pk}(z), \mathsf{pcs}(x, \mathsf{pk}(y), \mathsf{pk}(z), ct)) & \to & \mathsf{ok} \\
\mathsf{convertpcs}(z, \mathsf{pcs}(x, \mathsf{pk}(y), \mathsf{pk}(z), ct)) & \to & \mathsf{sign}(x, ct)
\end{array}$$

In this example, we consider a signature that has the constructors senc, aenc, pk, pcs, sign and ok. The functions sdec, $\mathsf{checkpcs}$, $\mathsf{check\_getmsg}$ and $\mathsf{convertpcs}$ are destructors. The symbols x, y, z are variables. The first three reductions model symmetric and asymmetric encryption and digital signing of messages in the usual way. The last two model *private contract signatures* that are used in our example in section V.

Processes $P, Q, R, \ldots$ are constructed as follows. The process 0 is the empty process which does nothing. In $\mathsf{new}\ \ a; P$, we restrict the name a in P; this can be used to model that a is a fresh random number or key. The process $\mathsf{in}(M, x); P$ models the input of a term on a channel M; the term is then substituted for x in process P. The process $\mathsf{out}(M, N)$ outputs a term N on a channel M. The parallel composition $P \mid Q$ models processes P and Q running concurrently. The conditional if $M = N$ then P else Q behaves as P when M and N are equal modulo the reductions, and behaves as Q otherwise. $!P$ is the replication of P, modelling an unbounded number of copies of the process P. ProVerif can automatically check security properties, while assuming that an arbitrary adversary process is run in parallel.

Example 1. *The following process P models a simple mutual authentication protocol in which a party A engages with another party, say B, by sending to B a signed and encrypted session key k:*

$$\begin{array}{lcl}
P & = & \mathsf{new}\ sk_A; \mathsf{new}\ sk_B; \mathsf{new}\ s; \\
 & & (\mathsf{out}(c, \mathsf{pk}(sk_A)) \mid \mathsf{out}(c, \mathsf{pk}(sk_B)) \mid !P_A \mid !P_B) \\
P_A & = & \mathsf{in}(c, x_{pk}); \mathsf{new}\ k; \\
 & & \mathsf{out}(c, \mathsf{aenc}(x_{pk}, \mathsf{sign}(sk_A, k))); \\
 & & \mathsf{in}(c, z); Q \\
P_B & = & \mathsf{in}(c, y); \text{let } y' = \mathsf{adec}(sk_B, y) \text{ in} \\
 & & \text{let } y_k = \mathsf{check\_getmsg}(\mathsf{pk}(sk_A), y') \text{ in} \\
 & & \mathsf{out}(c, \mathsf{senc}(y_k, s))
\end{array}$$

B responds by sending a secret s encrypted with k. Of course, this protocol is known not to be secure; an attacker can send its own public key to A, and use the session key it receives to start a parallel session with B. Then the attacker will be able to decrypt B's secret.

2.2 Horn clauses

The ProVerif tool works by translating processes written in the process language into clauses of a particular form. Such a clause

$$H_1 \ \wedge\ H_2 \ \wedge\ \ldots\ \wedge\ H_n \ \rightarrow\ C$$

is a conjunction of hypotheses and a conclusion. The hypotheses H_i and the conclusion C are called facts, and are built by applying predicate symbols to terms. ProVerif uses the two predicates $\mathsf{attacker}$ and $\mathsf{message}$. The fact $\mathsf{attacker}(N)$ means that the attacker can learn the value N. The fact $\mathsf{message}(M, N)$ means that the message N is available on the channel M.

In the clause language of ProVerif, terms are formed from variables and names, and by application of function symbols. Names are distinguished syntactically by the fact that they are followed by square brackets $[\ldots]$; function symbols are followed by round brackets $(\ldots)$; and variables are not followed by brackets. To handle the generation of new names by a process, such names in the clause representation are parametrised by the inputs that have occurred before the new name is generated. The new name k in the running example above is generated after the input of xpk; therefore, since there may be different k's for different xpk's, the k becomes parametrised by xpk, and is written $k[xpk]$. The running example process P above is translated into the following clauses:

Clauses corresponding to the process

$$\begin{array}{l}
\mathsf{message}(c[], \mathsf{pk}(skA[])) \\
\mathsf{message}(c[], \mathsf{pk}(skB[])) \\
\mathsf{message}(c[], xpk) \ \rightarrow\ \mathsf{message}(c[], \mathsf{aenc}(xpk, \mathsf{sign}(skA[], k[xpk])))) \\
\mathsf{message}(c[], \mathsf{aenc}(\mathsf{pk}(skB[]), \mathsf{sign}(skA[], y))) \ \rightarrow\ \mathsf{message}(c[], \mathsf{senc}(y, s[]))
\end{array}$$

The first two clauses correspond to the output of the public keys in the main process P. The third one corresponds to the attacker's ability to supply any data xpk to P_A, and in return obtain $\mathsf{aenc}(xpk, \mathsf{sign}(skA[], k[xpk]))$. The last one corresponds to a similar service offered by P_B.

Clauses corresponding to the attacker's ability to apply function symbols These clauses depend only on the equational theory and not on the specific process.

$\mathsf{attacker}(ok())$
$\mathsf{attacker}(v) \;\rightarrow\; \mathsf{attacker}(pk(v))$
$\mathsf{attacker}(v_1) \wedge \mathsf{attacker}(v_2) \;\rightarrow\; \mathsf{attacker}(\mathsf{sign}(v_1, v_2))$
$\mathsf{attacker}(v_1) \wedge \mathsf{attacker}(v_2) \;\rightarrow\; \mathsf{attacker}(\mathsf{senc}(v_1, v_2))$
$\mathsf{attacker}(v_1) \wedge \mathsf{attacker}(v_2) \;\rightarrow\; \mathsf{attacker}(\mathsf{aenc}(v_1, v_2))$
$\mathsf{attacker}(v_1) \wedge \mathsf{attacker}(v_2) \wedge \mathsf{attacker}(v_3) \;\rightarrow\; \mathsf{attacker}(\mathsf{pcs}(v_1, v_2, v_3))$

Clauses corresponding to the term reductions

$\mathsf{attacker}(\mathsf{pk}(x)) \wedge \mathsf{attacker}(\mathsf{sign}(x, y)) \;\rightarrow\; \mathsf{attacker}(y)$
$\mathsf{attacker}(x) \wedge \mathsf{attacker}(\mathsf{aenc}(\mathsf{pk}(x), y)) \;\rightarrow\; \mathsf{attacker}(y)$
$\mathsf{attacker}(x) \wedge \mathsf{attacker}(\mathsf{senc}(x, y)) \;\rightarrow\; \mathsf{attacker}(y)$

Clauses corresponding to the attacker's ability to send and receive messages These clauses are the same for all protocols and all equational theories.

$\mathsf{message}(v_1, v_2) \wedge \mathsf{attacker}(v_1) \;\rightarrow\; \mathsf{attacker}(v_2)$
$\mathsf{attacker}(v_1) \wedge \mathsf{attacker}(v_2) \;\rightarrow\; \mathsf{message}(v_1, v_2)$
$\mathsf{attacker}(c[])$

The first of these three clauses says that if the attacker has a channel name v_1 then he may read messages sent on it. The second one is the dual; he may also send messages on v_1. Lastly, we stipulate that the channel c is public.

Returning to the authentication protocol example, one can check that the fact $\mathsf{attacker}(s[])$ can be derived from the set of clauses. Indeed, this derivation corresponds to a real attack, and the protocol is not secure.

2.3 Translation and correctness

Details of the translation from the process language to clauses may be found in [9]. We do not detail it here, although we extend it to handle states in section IV. The translation has an important correctness property: it does not omit attacks. More precisely, if the process allows the attacker to obtain a secret value, say s, then $\mathsf{attacker}(s)$ can be derived from the clauses that correspond to the process. ProVerif uses a clause resolution strategy that is complete. Therefore, if ProVerif concludes that $\mathsf{attacker}(s)$ is not derivable, it is indeed not derivable. In that case, thanks to the correctness property of the translation, we can conclude that the attacker is indeed not capable of obtaining the secret s from the process.

3 The StatVerif language

We extend the process language of [9] recalled in section II with some constructs to handle global state.

3.1 Syntax and informal semantics

To model global state, StatVerif adds the following new processes:

- $[s \mapsto M]$, which represents a cell s that has the initial value M;
- $\mathsf{read}\ s_1, \ldots, s_n\ \mathsf{as}\ x_1, \ldots, x_n;\ P$, which reads the values stored in cells $s_1, \ldots, s_n$ (calling them $x_1, \ldots, x_n$ respectively), and then continues as P;
- $s_1, \ldots, s_n := M_1, \ldots, M_n;\ P$ which assigns $M_1, \ldots, M_n$ to $s_1, \ldots, s_n$ respectively and then continues as P;
- $\mathsf{lock}\ s_1, \ldots, s_n;\ P$. This process begins a *locked section*; that means that the process takes exclusive access to the state cells $s_1, \ldots, s_n$, and continues as P; and
- $\mathsf{unlock}\ s_1, \ldots, s_n;\ P$, which releases the lock on the state cells $s_1, \ldots, s_n$, continuing as P.

The full syntax of StatVerif is given in Figure 1, subject to an additional restriction: $[s \mapsto M]$ may occur only once for a given cell name s, and may occur only within the scope of new, a parallel and a replication. It may not be in the scope of an input, output, conditional, let, assignment, lock, or unlock.

Note that a process that executes a parallel or a replication after locking one or more cells, but before unlocking them, will block according to the semantics. Such a syntactic construction is therefore not useful.

The purpose of $\mathsf{lock}\ s_1, \ldots, s_n$ and $\mathsf{unlock}\ s_1, \ldots, s_n$ is to allow manipulations of the global state cells $s_1, \ldots, s_n$ to proceed without interference from other concurrent processes. Obviously, such interactions would lead to unwanted results. For example, in our security device, the $\mathsf{lock}\ s$ and $\mathsf{unlock}\ s$ in lines 4, 6 and 7 ensure that the device cannot move from the "left" configuration to the "right" configuration. If we didn't have the $\mathsf{lock}\ s$ and $\mathsf{unlock}\ s$, it would be possible to have the following execution. Consider two parallel sessions of the device. The first inputs left on channel c and reads the state s. Then the second session inputs right on channel c and reads the state s. At this moment both sessions consider the device to be in state init. It would thus be possible for the first session to update s to left and then for the second one to update s to right, *i.e.* the state s goes from init to left and then to right. In other words, without the locked section it is possible to reconfigure the device at will.

The $\mathsf{read}\ s_1, \ldots, s_n\ \mathsf{as}\ x_1, \ldots, x_n$ and $s_1, \ldots, s_n := M_1, \ldots, M_n$ constructs allow multiple cells to be read from or written to atomically. This eliminates the possibility of an older value of one cell being mixed with a newer value of another, and allows the Horn clauses produced by StatVerif to be fewer and simpler, with fewer variables and fewer hypotheses. This, in our experiments, has helped with termination. For example, translating $\mathsf{read}\ s_1\ \mathsf{as}\ x; \mathsf{read}\ s_2\ \mathsf{as}\ y$ may result in the hypotheses $\mathsf{message}((x, vs_1), vc_1, vm_1) \wedge \mathsf{message}((vs_2, y), vc_2, vm_2)$, whereas $\mathsf{read}\ s_1, s_2\ \mathsf{as}\ x, y$ can result in just the hypothesis $\mathsf{message}((x, y), vc, vm)$.

$M, N ::=$	terms
x, y, z	variables
a, b, c, k, s	names
$f(M_1, \ldots, M_n)$	constructor application
$\tilde{M}, \tilde{N} ::=$	tuple of terms
$(M_1, \ldots, M_n)$	
$P, Q ::=$	processes
0	nil
$\mathsf{out}(M, N);\ P$	output
$\mathsf{in}(M, x);\ P$	input
$P \mid Q$	parallel composition
$!P$	replication
$\mathsf{new}\ a;\ P$	restriction
let $x = g(M_1, \ldots, M_n)$ in P else Q	destructor application
if $M = N$ then P else Q	conditional
$[s \mapsto M]$	state
$\mathsf{read}\ s_1, \ldots, s_n\ \mathsf{as}\ x_1, \ldots, x_n;\ P$	read
$s_1, \ldots, s_n := M_1, \ldots, M_n;\ P$	assignment
$\mathsf{lock}\ s_1, \ldots, s_n;\ P$	beginning of locked section
$\mathsf{unlock}\ s_1, \ldots, s_n;\ P$	end of locked section

Figure 1: The StatVerif calculus. The terms and the processes up to and including the conditional are from [9]. The remaining processes are our additions. Some syntax restrictions are mentioned in the text.

We use the usual syntactic notion of subprocess. We sometimes omit the else branch of "if" and "let" processes. If the subprocess if $M = N$ then P occurs in the scope of a $\mathsf{lock}\ s_1, \ldots, s_n$, then it is an abbreviation of if $M = N$ then P else $\mathsf{unlock}\ s_1, \ldots, s_n; 0$, otherwise it is an abbreviation of if $M = N$ then P else 0. Similarly, if the subprocess let $x = g(M_1, \ldots, M_n)$ in P occurs in the scope of a $\mathsf{lock}\ s_1, \ldots, s_n$, then it is an abbreviation of let $x = g(M_1, \ldots, M_n)$ in P else $\mathsf{unlock}\ s_1, \ldots, s_n; 0$, otherwise it is an abbreviation of let $x = g(M_1, \ldots, M_n)$ in P else 0. We also write let $x = f(M_1, \ldots, M_n)$ in P to mean $P\{^{f(M_1, \ldots, M_n)}/_{x}\}$, as an abbreviation for repeated constructor application.

The process $\mathsf{new}\ a; P$ binds a in P, $\mathsf{in}(c, x); P$ binds x in P,
let $x = g(M_1, \ldots, M_n)$ in P else Q binds x in P (but not in Q), and
$\mathsf{read}\ s_1, \ldots, s_n\ \mathsf{as}\ x_1, \ldots, x_n; P$ binds $x_1, \ldots, x_n$ in P. The scope of a, x, x_1, $\ldots x_n$ is P. As usual, we use $\mathrm{bn}(P)$ and $\mathrm{bv}(P)$ to denote the set of bound names and bound variables of P respectively, and $\mathrm{fn}(P)$ and $\mathrm{fv}(P)$ to denote the set of free names and free variables of P respectively.

3.2 Operational semantics

A semantic configuration for StatVerif is a tuple $(\mathcal{E}, \mathcal{S}, \mathcal{P})$, where the environment $\mathcal{E}$ is a finite set of names, $\mathcal{S}$ is a function mapping state cells to their values, $\mathcal{P}$ is a finite multiset of pairs of the form (P, λ) where P is a process and λ is the set of cell names that P has locked for its own exclusive access. In a configuration $(\mathcal{E}, \mathcal{S}, \mathcal{P})$, a cell name appears in at most one of the λs. The environment $\mathcal{E}$ must contain at least the free names of $\mathcal{S}$ and $\mathcal{P}$. The configuration $(\{a_1, \ldots, a_n\}, \mathcal{S}, \{(P_1, \emptyset), \ldots, (P_m, \emptyset)\})$ intuitively corresponds to the process $\mathsf{new}\ a_1, \ldots, a_n; ([s \mapsto \mathcal{S}(s) \mid s \in \mathrm{dom}(\mathcal{S})] \mid P_1 \mid \cdots \mid P_m)$.

The semantics of StatVerif is defined by a reduction relation $\rightarrow$ on semantic configurations, shown in Figure 2. It is an extension of the semantics of [9, Fig. 3]. Notice that it preserves the invariant that at most one of the processes in $\mathcal{P}$ can have a given cell name locked. The cell name s is added to λ by lock, and only one process $(P, \lambda) \in \mathcal{P}$ can satisfy $s \in \mathcal{P}$. If a process has locked a cell, the other running processes cannot use the cell until the corresponding unlock. $s_1, \ldots, s_n := M_1, \ldots, M_n$ and $\mathsf{read}\ s_1, \ldots, s_n\ \mathsf{as}\ x_1, \ldots, x_n$ update and read the store $\mathcal{S}$ in the expected way.

3.3 Definition of secrecy

An adversary A is represented as a process of our calculus. He has some initial knowledge of a finite set of names Init with at least one channel name $\mathit{attch} \in \mathit{Init}$. A is said to be an Init-adversary if A is a closed process and $\mathrm{fn}(A) \subseteq \mathit{Init}$.

Informally, a protocol preserves the secrecy of a message M from Init if, when run in parallel with any Init-adversary A, M cannot be output on a public channel.

Definition 1. *Let P be a closed process, Init a finite set of names such that $\mathit{attch} \in \mathit{Init}$, M a message. P preserves the secrecy of M against Init if for any Init-adversary A, there exists no trace of the form:*
$((\mathit{Init} \cup \mathrm{fn}(P) \cup \mathrm{fn}(M)), \emptyset, \{(P \mid A, \emptyset)\}) \rightarrow^* (\mathcal{E}, \mathcal{S}, \mathcal{Q} \cup \{(\mathsf{out}(\mathit{attch}, M); Q, \lambda)\})$.

$$
\begin{array}{rl}
(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(0, \lambda)\}) & \rightarrow (\mathcal{E}, \mathcal{S}, \mathcal{P}) \\
(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(!P, \emptyset)\}) & \rightarrow (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(!P \mid P, \emptyset)\}) \\
(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(P \mid Q, \emptyset)\}) & \rightarrow (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(P, \emptyset), (Q, \emptyset)\}) \\
(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\mathsf{new}\ a; P, \lambda)\}) & \rightarrow (\mathcal{E} \cup \{a'\}, \mathcal{S}, \mathcal{P} \cup \{(P\{a'/a\}, \lambda)\}) \quad \textbf{if } a' \text{ fresh} \\
(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\mathsf{let}\ x = g(M_1, \ldots, M_n)\ \mathsf{in} & \\
P \text{ else } Q, \lambda)\}) & \rightarrow (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(P\{M'/x\}, \lambda)\}) \\
 & \hfill \textbf{if } g(M_1, \ldots, M_n) \rightarrow M' \\
(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\mathsf{let}\ x = g(M_1, \ldots, M_n)\ \mathsf{in} & \\
P \text{ else } Q, \lambda)\}) & \rightarrow (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(Q, \lambda)\}) \\
 & \hfill \textbf{if } \nexists M',\ g(M_1, \ldots, M_n) \rightarrow M' \\
(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\text{if } M = M & \\
\text{then } P \text{ else } Q, \lambda)\}) & \rightarrow (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(P, \lambda)\}) \\
(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\text{if } M = N & \\
\text{then } P \text{ else } Q, \lambda)\}) & \rightarrow (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(Q, \lambda)\}) \qquad \textbf{if } M \neq N \\
 & \\
(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\mathsf{out}(M, N); P, \lambda_1), & \\
(\mathsf{in}(M, x); Q, \lambda_2)\}) & \rightarrow (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(P, \lambda_1), (Q\{N/x\}, \lambda_2)\}) \\
(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{([s \mapsto M], \emptyset)\}) & \rightarrow (\mathcal{E}, \mathcal{S} \cup \{s \mapsto M\}, \mathcal{P}) \qquad \textbf{if } s \in \mathrm{dom}(\mathcal{E}) \\
 & \hfill \textbf{and } s \notin \mathrm{dom}(\mathcal{S}) \\
(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\mathsf{lock}\ s_1, \ldots, s_n; P, \lambda)\}) & \rightarrow (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(P, \lambda \cup \{s_1, \ldots, s_n\})\}) \\
 & \hfill \textbf{if } \forall (Q, \lambda') \in \mathcal{P}.\ \{s_1, \ldots, s_n\} \cap \lambda' = \emptyset \\
(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\mathsf{unlock}\ s_1, \ldots, s_n; P, \lambda)\}) & \rightarrow (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(P, \lambda \smallsetminus \{s_1, \ldots, s_n\})\}) \\
(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\mathsf{read}\ s_1, \ldots, s_n\ \mathsf{as} & \\
x_1, \ldots, x_n; P, \lambda)\}) & \rightarrow (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(P\{\mathcal{S}(s_1)/x_1, \ldots, \mathcal{S}(s_n)/x_n\}, \lambda)\}) \\
 & \hfill \textbf{if } s_1, \ldots, s_n \in \mathrm{dom}(\mathcal{S}) \\
 & \hfill \textbf{if } \forall (Q, \lambda') \in \mathcal{P}.\ \{s_1, \ldots, s_n\} \cap \lambda' = \emptyset \\
(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(s_1, \ldots, s_n := & \\
M_1, \ldots, M_n; P, \lambda)\}) & \rightarrow (\mathcal{E}, \mathcal{S}[s_i \mapsto M_i \mid 1 \leq i \leq n], \mathcal{P} \cup \{(P, \lambda)\}) \\
 & \hfill \textbf{if } s_1, \ldots, s_n \in \mathrm{dom}(\mathcal{S}) \\
 & \hfill \textbf{if } \forall (Q, \lambda') \in \mathcal{P}.\ \{s_1, \ldots, s_n\} \cap \lambda' = \emptyset
\end{array}
$$

Figure 2: The semantics of StatVerif. $\mathcal{E}$ is a set of names. $\mathcal{S}$ is a function from state cells to their current values. $\mathcal{P}$ is the set of running processes (P, λ), where P is the process itself and λ is the set of cell names to which the process has exclusive access.

Here we consider that M is secret if it is secret in all reachable states. We could have extended this definition to express secrecy relative to a particular state, or to states of a certain form, but for simplicity, and since we don't need to in what follows, we didn't include it here.

4 Translation to Clauses

4.1 The translation

The translation generates clauses from a StatVerif process. In the Horn clauses generated by our translation, each bound name a in P' is represented by the pattern $a[p_1, \ldots, p_n]$ with $p_1, \ldots, p_n$ the set of variables read and input before the generation of a; and terms in general are represented by patterns generated by the following grammar:

p ::=	patterns
x, y, z	variables
$a[p_1, \ldots, p_n]$	name
$f(p_1, \ldots, p_n)$	constructor application

The clauses are built around the predicates $\mathsf{attacker}$ and $\mathsf{message}$ with the following meanings:

- $\mathsf{attacker}(\tilde{M}, N)$ means that there is a reachable state of the process in which the state cells $\tilde{s}$ have the values $\tilde{M}$, and in that state the attacker may know the value N; this binary predicate is also used in [8].
- $\mathsf{message}(\tilde{M}, N, K)$ means that there is a reachable state of the process in which the state cells $\tilde{s}$ have the values $\tilde{M}$, and in that state the value K is available on channel N.

Our translation only applies to StatVerif processes of the form:

$$\mathsf{new}\ \tilde{m}; ([s_1 \mapsto M_1] \mid \cdots \mid [s_n \mapsto M_n] \mid P_0)$$

such that

- P_0 has no $[s \mapsto M]$ in it. (Of course, P_0 may have reads and assignments.)
- each name and variable is bound at most once in P_0; and each name and variable in P_0 is either bound or free but not both.

The tuple $\tilde{m}$ contains cell names and ordinary names. Some of the $s_1, \ldots, s_n$ may be in $\tilde{m}$, and others not.

Note that any process with a bounded number of cell names can be converted into one of the prescribed form. While the restriction to a bounded number of cells may appear to be severe, we will see in section 5 that it is still possible to verify some processes with unbounded numbers of memory cells, by using a correct abstraction: replacing a process having an unbounded number of memory cells with a process having a bounded number of memory cells.

$$\llbracket 0 \rrbracket \rho H \iota \phi \lambda \quad = \quad \emptyset$$

$$\llbracket P \mid Q \rrbracket \rho H \iota \phi \emptyset \quad = \quad \llbracket P \rrbracket \rho H \iota \phi \emptyset \cup \llbracket Q \rrbracket \rho H \iota \phi \emptyset$$

$$\llbracket !P \rrbracket \rho H \iota \phi \emptyset \quad = \quad \llbracket P \rrbracket \rho H \iota \phi \emptyset$$

$$\llbracket \mathsf{new}\ a; P \rrbracket \rho H \iota \phi \lambda \quad = \quad \begin{cases} \llbracket P \rrbracket (\rho \cup \{a \mapsto a[\iota]\}) H \iota \phi \lambda & \textbf{if } a \in \mathrm{bn}(P_0') \\ \llbracket P \rrbracket (\rho \cup \{a \mapsto attn[]\}) H \iota \phi \lambda & \textbf{otherwise} \end{cases}$$

$$\begin{aligned} \llbracket \mathsf{in}(M, x); P \rrbracket \rho H \iota \phi \lambda \quad = \quad & \llbracket P \rrbracket \rho' H' (x :: \iota) \phi' \lambda \\ & \textbf{where } \phi' = \phi\,[k \mapsto vs_k \mid k \notin \lambda] \\ & \textbf{and } H' = H \wedge \mathsf{message}(\phi', \rho(M), x) \\ & \textbf{and } \rho' = \rho \cup \{x \mapsto x\} \cup \{vs_k \mapsto vs_k \mid k \notin \lambda\} \\ & \textbf{and } vs_1, \ldots, vs_n \text{ are fresh variables} \end{aligned}$$

$$\llbracket \mathsf{out}(M, N); P \rrbracket \rho H \iota \phi \lambda \quad = \quad \{H \Rightarrow \mathsf{message}(\phi, \rho(M), \rho(N))\} \cup \llbracket P \rrbracket \rho H \iota \phi \lambda$$

$$\begin{aligned} & \llbracket \mathsf{if}\ M = N\ \mathsf{then} \\ & \qquad P \text{ else } Q \rrbracket \rho H \iota \phi \lambda \quad = \quad \llbracket P \rrbracket (\rho\sigma)(H\sigma)(\iota\sigma)(\phi\sigma)\lambda \cup \llbracket Q \rrbracket \rho H \iota \phi \lambda \\ & \qquad\qquad \textbf{where } \sigma = \mathrm{mgu}(\rho(M), \rho(N)) \end{aligned}$$

$$\begin{aligned} & \llbracket \mathsf{let}\ x = g(M_1, \ldots, M_t)\ \mathsf{in} \\ & \qquad P \text{ else } Q \rrbracket \rho H \iota \phi \lambda \quad = \quad \llbracket Q \rrbracket \rho H \iota \phi \lambda \\ & \bigcup \quad \Big\{ \llbracket P \rrbracket ((\rho\sigma) \cup \rho')(H\sigma)(\iota\sigma)(\phi\sigma)\lambda \ \Big| \\ & \qquad g(p_1, \ldots, p_t) \rightarrow p \in def(g) \\ & \qquad \textbf{and } \{z_1, \ldots, z_m\} = \mathrm{fv}(g(p_1, \ldots, p_t)) \\ & \qquad \textbf{and } z_1', \ldots, z_m' \text{ are fresh variables} \\ & \qquad \textbf{and } \sigma' = \{z_i \mapsto z_i' \mid 1 \leq i \leq m\} \\ & \qquad \textbf{and } \sigma = \mathrm{mgu}(g(M_1\rho, \ldots, M_t\rho), g(p_1\sigma', \ldots, p_t\sigma')) \\ & \qquad \textbf{and } \rho' = \{x \mapsto p\sigma'\sigma\} \cup \{z_i' \mapsto z_i'\sigma \mid 1 \leq i \leq m\} \ \Big\} \end{aligned}$$

$$\begin{array}{rcl}
\llbracket \mathsf{lock}\ s_{i_1},\ldots, & & \\
s_{i_m}; P\rrbracket \rho H \iota \phi \lambda & = & \llbracket P\rrbracket (\rho \cup \{vs_k \mapsto vs_k \mid k \notin \lambda\}) H \iota \phi' (\lambda \cup \{i_1,\ldots,i_m\}) \\
& & \textbf{where } vs_1,\ldots,vs_n \text{ are fresh variables} \\
& & \textbf{and } \phi' = \phi\,[k \mapsto vs_k \mid k \notin \lambda]
\end{array}$$

$$\begin{array}{rcl}
\llbracket \mathsf{unlock}\ s_{i_1},\ldots, & & \\
s_{i_m}; P\rrbracket \rho H \iota \phi \lambda & = & \llbracket P\rrbracket (\rho \cup \{vs_k \mapsto vs_k \mid k \notin \lambda\}) H \iota \phi' (\lambda \setminus \{i_1,\ldots,i_n\}) \\
& & \textbf{where } vs_1,\ldots,vs_n \text{ are fresh variables} \\
& & \textbf{and } \phi' = \phi\,[k \mapsto vs_k \mid k \notin \lambda]
\end{array}$$

$$\begin{array}{rcl}
\llbracket \mathsf{read}\ s_{i_1},\ldots,s_{i_m} & & \\
\mathsf{as}\ x_1,\ldots,x_m; P\rrbracket \rho H \iota \phi \lambda & = & \llbracket P\rrbracket \rho'' H' (x_1 :: \cdots :: x_m :: \iota) \phi' \lambda \\
& & \textbf{where } \rho'' = \rho' \cup \{x_j \mapsto \phi'_{i_j} \mid 1 \leq j \leq m\} \cup \{vc \mapsto vc, vm \mapsto vm\} \\
& & \textbf{and } \rho' = \rho \cup \{vs_k \mapsto vs_k \mid k \notin \lambda\} \\
& & \textbf{and } \phi' = \phi\,[k \mapsto vs_k \mid k \notin \lambda] \\
& & \textbf{and } H' = H \wedge \mathsf{message}(\phi', vc, vm) \\
& & \textbf{and } vs_1,\ldots,vs_n, vc, vm \text{ are fresh variables}
\end{array}$$

$$\begin{array}{rcl}
\llbracket s_{i_1},\ldots,s_{i_m} := & & \\
M_1,\ldots,M_m; P\rrbracket \rho H \iota \phi \lambda & = & \llbracket P\rrbracket (\rho \cup \{vs_k \mapsto vs_k \mid k \notin \lambda\}) H \iota \phi'' \lambda \\
& \cup & \{H \wedge \mathsf{message}(\phi', vc, vm) \Rightarrow \mathsf{message}(\phi'', vc, vm)\} \\
& \cup & \{H \wedge \mathsf{attacker}(\phi', vm) \Rightarrow \mathsf{attacker}(\phi'', vm)\} \\
& & \textbf{where } \phi' = \phi\,[k \mapsto vs_k \mid k \notin \lambda] \\
& & \textbf{and } \phi'' = \phi'\,[i_j \mapsto \rho(M_j) \mid 1 \leq j \leq m] \\
& & \textbf{and } vs_1,\ldots,vs_n, vc, vm \text{ are fresh variables}
\end{array}$$

Figure 3: The rules for translating the stateful process P_0 into clauses. The translation of the StatVerif process new $\tilde{m}; ([s_1 \mapsto M_1] \mid \cdots \mid [s_n \mapsto M_n] \mid P_0)$ is $\llbracket P_0 \rrbracket\ \rho_0$ true $[]\ \phi_0$. (Note that the rule for new a references this P_0 and this n.) In the rules, $k \notin \lambda$ abbreviates $1 \leq k \leq n, k \notin \lambda$.

4.1.1 Clauses corresponding to the protocol

Let $P'_0 = \mathsf{new}\ \tilde{m}; ([s_1 \mapsto M_1] \mid \cdots \mid [s_n \mapsto M_n] \mid P_0)$ be a StatVerif process. Let ρ_0 be the function $\{a \mapsto a[],\ s_i \mapsto s_i[] \mid a \in \mathrm{fn}(P'_0),\ 1 \leq i \leq n\}$ and let $\phi_0 = (\rho_0(M_1), \ldots, \rho_0(M_n))$. The process P'_0 is translated into the following sets of clauses:

- $[\![P_0]\!]\ \rho_0\ \mathsf{true}\ [\,]\ \phi_0\ \emptyset$ where the function $[\![\,\cdot\,]\!]\rho H \iota \phi \lambda$ is given in Figure 3;
- Some other clauses given in the next two subsections.

The rules of Figure 3 generalise the ones given in [9, §5.2.2].

The StatVerif compiler that performs the translation maintains the variables ρ, H, ι, ϕ and λ, which have the following purposes:

- ρ is a function mapping names and variables of the process language to patterns of the clause language.
- H is a conjunction of facts used to accumulate the hypotheses of clauses as they are constructed.
- ι accumulates the set of variables that have been input or read so far by the thread being processed. This set is used to parametrise the Skolem names that represent values created by "new".
- ϕ is a tuple of terms $(M_1, \ldots, M_n)$ representing the last known values of the state cells in the thread under consideration. For a cell that the process being translated has locked, we can be sure that the corresponding element of ϕ represents the current state of the cell, whereas for a cell that the process has not locked, another subprocess running in parallel could have assigned an arbitrary value to the cell, so when constructing hypotheses or clauses, the elements of ϕ corresponding to unlocked cells are discarded and replaced with fresh variables.
- λ is a set of indices indicating which state cells the currently processed thread has locked for its exclusive access.

We explain the rules for the translation given in Figure 3.

- The rules for processing 0, parallel, "new", "let" and "if" are similar to those of [9], with obvious changes for our more general setting.
- The rule for processing $!P$ is simpler than [9], since we don't treat correspondence properties for now.
- For an *input*, we record in ρ and ι the variable that is input, and add a hypothesis to H. As explained above, entries for ϕ corresponding to unlocked cells are replaced with fresh variables.
- An *output* generates a clause that reveals the output on the channel, using the hypotheses accumulated so far.
- For $\mathsf{lock}\ s_{i_1}, \ldots, s_{i_m}$, we initialise the assumed state with variables for the so far unlocked cells (to represent the possibility of a parallel subprocess assigning to them), and we add the cell indices $i_1, \ldots, i_m$ to the set of already locked cell names λ.

- We translate $\mathsf{unlock}\ s_{i_1}, \dots, s_{i_m}$ in the same way as *lock*, except that the cell indices are removed from λ instead of added.
- The assignment $s_{i_1}, \dots, s_{i_m} := M_1, \dots, M_m$ updates the current state ϕ. The indices $i_1, \dots, i_m$ in ϕ are given the values $M_1, \dots, M_m$. The remaining indices are treated as in the *input* case: locked cells retain their values while values of unlocked cells are replaced by fresh variables. Additionally, we generate the "inheritance" clauses that transport possible attacker knowledge and message availability on channels from the state before the assignment to the state after it. In other words, if the attacker can know M before the assignment, he can also know it after the assignment.
- The *read* process assigns to the specified variables the values stored in the cells that are read. As in the *input* case, arbitrary values are assumed for unlocked cells. The hypothesis added to H ensures that the clause is only applicable if the values that were read from cells $s_{i_1}, \dots, s_{i_m}$ correspond to a reachable state.

4.1.2 Clauses corresponding to mutability of public state

If a state cell name s is known to the attacker, then the attacker is able to read and write values from and to the cell. For each $i \in \{1, 2, \dots, n\}$, we have the following clauses for reading:

$$\mathsf{attacker}((x_1, \dots, x_n), s_i[\,]) \ \rightarrow \mathsf{attacker}((x_1, \dots, x_n), x_i)$$

and the following ones for writing:

$$\begin{aligned}&\mathsf{attacker}((x_1, \dots, x_n), s_i[\,]) \ \wedge \ \mathsf{attacker}((x_1, \dots, x_n), y) \\ &\quad \wedge \mathsf{message}((x_1, \dots, x_n), xc, xm) \\ &\quad \rightarrow \ \mathsf{message}((x_1, \dots, x_{i-1}, y, x_{i+1}, \dots, x_n), xc, xm)\end{aligned}$$

$$\begin{aligned}&\mathsf{attacker}((x_1, \dots, x_n), s_i[\,]) \ \wedge \ \mathsf{attacker}((x_1, \dots, x_n), y) \\ &\quad \wedge \mathsf{attacker}((x_1, \dots, x_n), xm) \\ &\qquad \rightarrow \ \mathsf{attacker}((x_1, \dots, x_{i-1}, y, x_{i+1}, \dots, x_n), xm)\end{aligned}$$

4.1.3 Other clauses

Additionally, we have clauses corresponding to the function symbols and the term reductions for the signature at hand. These are the stateful counterparts of the clauses used by ProVerif:
For each constructor f of arity n,

$$\mathsf{attacker}(xs, x_1) \wedge \cdots \wedge \mathsf{attacker}(xs, x_n) \ \rightarrow \ \mathsf{attacker}(xs, f(x_1, \dots, x_n)).$$

For each constructor g, for each rewrite rule $g(M_1, \dots, M_n) \rightarrow M$, let xs be a fresh variable,

$$\mathsf{attacker}(xs, M_1) \wedge \cdots \wedge \mathsf{attacker}(xs, M_n) \ \rightarrow \ \mathsf{attacker}(xs, M)$$

The attacker is also able to read and write on channels that it knows, and the stateful analogues of those clauses are:

$$\begin{aligned}&\mathsf{message}(xs, v_1, v_2) \wedge \mathsf{attacker}(xs, v_1) \ \rightarrow \ \mathsf{attacker}(xs, v_2) \\ &\mathsf{attacker}(xs, v_1) \wedge \mathsf{attacker}(xs, v_2) \ \rightarrow \ \mathsf{message}(xs, v_1, v_2)\end{aligned}$$

Finally, the attacker knows

- all the free names of P_0', *i.e.* we have the clause $\mathsf{attacker}(\rho_0(\phi_0), n[])$ for every $n \in \mathrm{fn}(P)_0'$; and
- a channel *attch* and a name *attn* which he has generated on his own, *i.e.* we have the clauses $\mathsf{attacker}(\rho_0(\phi_0), attch[])$ and $\mathsf{attacker}(\rho_0(\phi_0), attn[])$,

where ρ_0 and ϕ_0 are as defined in section 4.1.1.

4.2 Correctness

Let $P_0' = \mathsf{new}\ \tilde{m}; ([s_1 \mapsto M_1] \mid \cdots \mid [s_n \mapsto M_n] \mid P_0)$ be a closed process and A an $Init$-adversary *s.t.* $attch \in Init$. Without loss of generality, we can assume that the free cell names in A are included in the free cell names of P_0' (i.e., $s_1, \ldots, s_n$), and that the set of bound cell names of A is empty. The reason is that any other cell name of the intruder can be equivalently encoded using channel names as described by Milner.

4.2.1 Instrumented operational semantics

To link the patterns in the generated clauses to the real terms exchanged and manipulated during the execution of P_0', we will consider instrumented semantic configurations $(\mathcal{E}, \mathcal{S}, \mathcal{P})$ where $\mathcal{E}$ will now be a mapping from names to StatVerif patterns, *i.e.* $\mathcal{E}$ records for each name a' the $\mathsf{new}\ a$ in P_0 it is an instance of. This representation of names allows us in particular to associate different instances of a $\mathsf{new}\ a$ with each other when arising from $\mathsf{new}\ a$ in the scope of a replication. $\mathcal{S}$ is as before a function from cell names to terms, and $\mathcal{P}$ is a set of tuples (Q, ι, λ) where we will record in ι the list of $M_1, \ldots, M_n$ that were previously input or read to reach this configuration.

We adapt the semantics to an *instrumented operational semantics* which is defined by a reduction relation on instrumented configurations. Except for the reduction rules for New, Comm, and Read all the other rules of Figure 2 give rise to a corresponding instrumented rule where $\mathcal{E}$ and the ιs are unchanged. And

- The reduction rule for communication becomes the following

$$\begin{aligned}(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\mathsf{out}(M, N); P, \iota_1, \lambda_1), (\mathsf{in}(M, x); Q, \iota_2, \lambda_2)\}) \ \rightarrow \\ (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(P, \iota_1, \lambda_1), (Q\{N/x\}, (N :: \iota_2), \lambda_2))\end{aligned}$$

  which records N in ι_2.

- The reduction rule for read becomes the following

$$\begin{aligned}(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\mathsf{read}\ s_1, \ldots, s_n\ \mathsf{as}\ x_1, \ldots, x_n; P, \iota, \lambda)\}) \ \rightarrow \\ (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(P\{\mathcal{S}(s_1)/x_1, \ldots, \mathcal{S}(s_n)/x_n\}, (\mathcal{S}(s_1) :: \cdots :: \mathcal{S}(s_n) :: \iota), \lambda)\}) \\ \textbf{if } s_1, \ldots, s_n \in \mathrm{dom}(\mathcal{S}) \textbf{ and } \forall (Q, \lambda') \in \mathcal{P}.\ \{s_1, \ldots, s_n\} \cap \lambda' = \emptyset\end{aligned}$$

  which records $\mathcal{S}(s_1), \ldots, \mathcal{S}(s_n)$ in ι_2.

- The reduction rule for name generation is replaced by the two following rules. The first one is for translating processes $\mathsf{new}\ a; P$ coming from the initial honest processes P'_0, *i.e.* $a \in \mathrm{bn}(P'_0)$, and the second one is for translating processes $\mathsf{new}\ a; P$ coming from the initial attacker processes A, *i.e.*. $a \notin \mathrm{bn}(P'_0)$ but $a \in \mathrm{bn}(A)$

  $$(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\mathsf{new}\ a; P, \iota, \lambda)\}) \ \rightarrow$$
  $$(\mathcal{E} \cup \{a' \mapsto a[\mathcal{E}(\iota)]\}, \mathcal{S}, \mathcal{P} \cup \{(P\{a'/a\}, \iota, \lambda)\})$$
  $$\textbf{if } a \in \mathrm{bn}(P'_0) \textbf{ and } a' \text{ fresh}$$

  which records that a' is an instance of $\mathsf{new}\ a$. ι is used to distinguish two instances of $\mathsf{new}\ a$ on the basis of the previous inputs.

  $$(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\mathsf{new}\ a; P, \iota, \lambda)\}) \ \rightarrow$$
  $$(\mathcal{E} \cup \{a' \mapsto attn[]\}, \mathcal{S}, \mathcal{P} \cup \{(P\{a'/a\}, \iota, \lambda)\})$$
  $$\textbf{if } a \notin \mathrm{bn}(P'_0) \textbf{ and } a' \text{ fresh}$$

  which records that a' is an name of the attacker A.

It is easy to see that the instrumented semantics allows exactly the same traces as the original semantics, only adding annotations on the origin of each name.

Proposition 1. *For all traces $(\mathcal{E}_0, \mathcal{S}_0, \mathcal{P}_0) \rightarrow^* (\mathcal{E}_1, \mathcal{S}_1, \mathcal{P}_1)$, there exists an instrumented trace $(\mathcal{E}'_0, \mathcal{S}_0, \mathcal{P}'_0) \rightarrow^* (\mathcal{E}'_1, \mathcal{S}_1, \mathcal{P}'_1)$ such that for all $k \in \{1, 2\}$, $\mathrm{dom}(\mathcal{E}'_k) = \mathcal{E}_k$ and for all $(P, \lambda) \in \mathcal{P}_k$, $(P, \iota, \lambda) \in \mathcal{P}'_k$ for some ι.*

For all instrumented traces $(\mathcal{E}'_0, \mathcal{S}_0, \mathcal{P}'_0) \rightarrow^* (\mathcal{E}'_1, \mathcal{S}_1, \mathcal{P}'_1)$, $(\mathcal{E}_0, \mathcal{S}_0, \mathcal{P}_0) \rightarrow^* (\mathcal{E}_1, \mathcal{S}_1, \mathcal{P}_1)$ is a valid trace such that for all $k \in \{1, 2\}$, $\mathcal{E}_k = \mathrm{dom}(\mathcal{E}'_k)$ and for all $(P, \iota, \lambda) \in \mathcal{P}'_k$, $(P, \lambda) \in \mathcal{P}_k$.

4.2.2 Proof of correctness

Let $P'_0 = \mathsf{new}\ \tilde{m}; ([s_1 \mapsto M_1] \mid \cdots \mid [s_n \mapsto M_n] \mid P_0)$ be a StatVerif process. Let $\mathcal{C}_0$ be the set of clauses generated by StatVerif when applied to P'_0, and $\mathcal{F}_0$ the set of closed facts derivable from $\mathcal{C}_0$. Let $\mathcal{S}_0 = \{s_1 \mapsto M_1, \ldots, s_n \mapsto M_n\}$. Let $\mathcal{E}_0$ be the environment such that

- $\mathrm{fn}(P'_0) \cup \mathrm{cells}(P'_0) \cup \mathrm{fn}(A) = \mathrm{dom}(\mathcal{E}_0)$,
- $\mathcal{E}_0(a) = a[]$ for all $a \in \mathrm{fn}(P'_0) \cup \mathrm{cells}(P'_0) \cup \{attch\}$,
- $\mathcal{E}_0(a) = attn[]$ for all $a \in \mathrm{fn}(A) \smallsetminus \{attch\}$.

Let $\mathcal{S} = \{s_1 \mapsto K_1, \ldots, s_n \mapsto K_n\}$ be a state. $\overline{\mathcal{S}}$ denotes the ordered representation of $\mathcal{S}$, defined as $\overline{\mathcal{S}} = (K_1, \ldots, K_n)$.

We will say that a state $\mathcal{R}$ is a predecessor of the state $\mathcal{S}$, denoted $\mathcal{R} \leq \mathcal{S}$ if:

$$\begin{array}{ll} & \mathsf{attacker}(\overline{\mathcal{R}}, attch[]) \in \mathcal{F}_0 \\ \wedge & \forall M, N\ \mathsf{message}(\overline{\mathcal{R}}, M, N) \in \mathcal{F}_0 \ \Rightarrow\ \mathsf{message}(\overline{\mathcal{S}}, M, N) \in \mathcal{F}_0 \\ \wedge & \forall M\ \mathsf{attacker}(\overline{\mathcal{R}}, M) \in \mathcal{F}_0 \ \Rightarrow\ \mathsf{attacker}(\overline{\mathcal{S}}, M) \in \mathcal{F}_0 \end{array}$$

The proof uses a type system to capture invariants of processes. The type system captures the fact that the clauses generated for P_0 are sound in the sense that for any message M output on a channel N in state $\mathcal{S}$, the corresponding fact $\mathsf{message}(\overline{\mathcal{S}}, \mathcal{E}(M), \mathcal{E}(N))$ is derivable (see typing of the out construct). The rest of the typing rules capture the fact that the type system satisfies subject reduction which in turn ensures that soundness of the clauses is preserved for all executions of the process. The type system is only used for the proof, not the implementation, so notions such as $\mathcal{S}_0 \leq \mathcal{S}$ are never evaluated.

This type system is defined by the rules of Figure 4 (an extended version of the type system of [9, 11]).

A process P is well typed *w.r.t.* the environment $\mathcal{E}$, the state $\mathcal{S}$, the list of StatVerif patterns ι, and the mode λ, if $(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash P$ can be derived from the rules and axiom of Figure 4.

Before proceeding with the proof of our main theorem, we need to establish some properties of our typing system.

Lemma 1 (Typability of A)**.**

$$(\mathcal{E}_0, \mathcal{E}_0(\mathcal{S}_0), [], \emptyset) \vdash A$$

Proof sketch. Let B be a subprocess of A, $\mathcal{E}$ an environment (from names and variables to patterns), $\mathcal{S}$ a state (from cell names to patterns), ι a sequence of patterns, and λ a set of cell indices. We first prove by induction on the depth d of B that, if

(i) $\mathcal{E}_0 \subseteq \mathcal{E}$; and

(ii) $\mathcal{E}(\mathcal{S}_0) \leq \mathcal{S}$; and

(iii) $(\mathrm{bn}(B) \cup \mathrm{bv}(B)) \cap \mathrm{dom}(\mathcal{E}) = \emptyset$; and

(iv) $\forall a \in \mathrm{fn}(B),\ \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0$; and

(v) $\forall x \in \mathrm{fv}(B),\ \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0$; and

(vi) for all $i \in \{1, \ldots, n\}$, $i \in \lambda$ if and only if B is in the scope of a $\mathsf{lock}\ \ldots s_i \ldots$ in A,

then

$$(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash B$$

To conclude the proof of Lemma 1 we then need to show that A, $\mathcal{E}_0$, $\mathcal{E}_0(\mathcal{S}_0)$, [], and $\emptyset$ satisfy conditions (i)- (vi).

(i) By definition $\mathcal{E}_0 \subseteq \mathcal{E}_0$.

(ii) By definition $\mathcal{E}_0(\mathcal{S}_0) \leq \mathcal{E}_0(\mathcal{S}_0)$.

(iii) By hypotheses, $\mathrm{dom}(\mathcal{E}_0) = \mathrm{fn}(P) \cup \mathrm{fn}(A) \cup \mathit{cellP}$ and $(\mathrm{bn}(A) \cup \mathrm{bv}(A)) \cap (\mathrm{fn}(P) \cup \mathrm{fn}(A) \cup \mathit{cellP}) = \emptyset$, thus $(\mathrm{bn}(A) \cup \mathrm{bv}(A)) \cap \mathrm{dom}(\mathcal{E}_0) = \emptyset$.

(iv) By construction, $\forall a \in \mathrm{fn}(A)$

If $a = \mathit{attch}$, then $\mathcal{E}_0(a) = \mathit{attch}[]$, and $\mathsf{attacker}(\overline{\mathcal{E}_0(\mathcal{S}_0)}, \mathit{attch}[]) \in \mathcal{C}_0$ by construction.

If $a \neq \mathit{attch}$, then $\mathcal{E}_0(a) = \mathit{attn}[]$, and $\mathsf{attacker}(\overline{\mathcal{E}_0(\mathcal{S}_0)}, \mathit{attn}[]) \in \mathcal{C}_0$ by construction.

Thus $\forall a \in \mathrm{fn}(A)$ $\mathsf{attacker}(\overline{\mathcal{E}_0(\mathcal{S}_0)}, \mathcal{E}_0(a)) \in \mathcal{F}_0$.

$$\frac{\mathsf{message}(\overline{\mathcal{S}}, \mathcal{E}(M), \mathcal{E}(N)) \in \mathcal{F}_0 \qquad (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash P}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \mathsf{out}(M, N); P}\ \tau_{out}$$

$$\frac{\begin{array}{l} \forall \mathcal{T}\ \forall N\ (\mathcal{S} \leq \mathcal{T}\ \wedge\ \mathcal{T} = \mathcal{T}[j \mapsto \mathcal{S}(j) \mid j \in \lambda] \wedge \\ \qquad\qquad\qquad \mathsf{message}(\overline{\mathcal{T}}, \mathcal{E}(M), N) \in \mathcal{F}_0)\ \Rightarrow \\ \qquad\qquad\qquad\qquad\qquad \mathcal{E} \cup \{x \mapsto N\}, \mathcal{T}, (N :: \iota), \lambda) \vdash P \end{array}}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \mathsf{in}(M, x); P}\ \tau_{in}$$

$$\frac{}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash 0}\ \tau_{nil} \qquad \frac{(\mathcal{E}, \mathcal{S}, \iota, \emptyset) \vdash P \qquad (\mathcal{E}, \mathcal{S}, \iota, \emptyset) \vdash Q}{(\mathcal{E}, \mathcal{S}, \iota, \emptyset) \vdash P \mid Q}\ \tau_{par} \qquad \frac{(\mathcal{E}, \mathcal{S}, \iota, \emptyset) \vdash P}{(\mathcal{E}, \mathcal{S}, \iota, \emptyset) \vdash !P}\ \tau_{repl}$$

$$\frac{(\mathcal{E}(M) = \mathcal{E}(N) \Rightarrow (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash P) \qquad (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \text{if } M = N \text{ then } P \text{ else } Q}\ \tau_{if}$$

$$\frac{a \in \mathrm{bn}(P_0')\ \Rightarrow\ (\mathcal{E} \cup \{a \mapsto a[\iota]\}, \mathcal{S}, \iota, \lambda) \vdash P}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \mathsf{new}\ a; P}\ \tau_{newP}$$

$$\frac{a \notin \mathrm{bn}(P_0')\ \Rightarrow\ (\mathcal{E} \cup \{a \mapsto \mathit{attn}[]\}, \mathcal{S}, \iota, \lambda) \vdash P}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \mathsf{new}\ a; P}\ \tau_{newA}$$

$$\frac{\forall M\ (g(\mathcal{E}(M_1), \ldots, \mathcal{E}(M_n)) \to M)\ \Rightarrow ((\mathcal{E} \cup \{x \mapsto M\}, \mathcal{S}, \iota, \lambda) \vdash P\ \wedge\ (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q)}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \text{let } x = g(M_1, \ldots, M_n) \text{ in } P \text{ else } Q}\ \tau_{let}$$

$$\frac{\begin{array}{l} \forall \mathcal{T}\ (\mathcal{S} \leq \mathcal{T}\ \wedge\ \mathcal{T} = \mathcal{T}[j \mapsto \mathcal{S}(j) \mid j \in \lambda]) \Rightarrow \\ \qquad (\mathcal{E} \cup \{x_k \mapsto \mathcal{T}(j_k) \mid 1 \leq k \leq m\}, \mathcal{T}, (\mathcal{T}(j_1) :: \cdots :: \mathcal{T}(j_m) :: \iota), \lambda) \vdash P \end{array}}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \mathsf{read}\ s_{j_1}, \ldots, s_{j_m}\ \mathsf{as}\ x_1, \ldots, x_m; P}\ \tau_{read}$$

$$\frac{\begin{array}{l} \forall \mathcal{T}\ (\mathcal{S} \leq \mathcal{T}\ \wedge\ \mathcal{T} = \mathcal{T}[j \mapsto \mathcal{S}(j) \mid j \in \lambda]) \Rightarrow \\ \qquad (\mathcal{T} \leq \mathcal{T}[j_k \mapsto \mathcal{E}(M_k) \mid 1 \leq k \leq m] \wedge \\ \qquad\qquad (\mathcal{E}, \mathcal{T}[j_k \mapsto \mathcal{E}(M_k) \mid 1 \leq k \leq m], \iota, \lambda) \vdash P) \end{array}}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash s_{j_1}, \ldots, s_{j_m} := M_1, \ldots, M_m; P}\ \tau_{write}$$

$$\frac{\forall \mathcal{T}\ (\mathcal{S} \leq \mathcal{T}\ \wedge\ \mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda]\ \Rightarrow\ (\mathcal{E}, \mathcal{T}, \iota, \lambda \cup \{j_1, \ldots, j_m\}) \vdash P)}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \mathsf{lock}\ s_{j_1}, \ldots, s_{j_m}; P}\ \tau_{lock}$$

$$\frac{\forall \mathcal{T}\ (\mathcal{S} \leq \mathcal{T}\ \wedge\ \mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda]\ \Rightarrow\ (\mathcal{E}, \mathcal{T}, \iota, \lambda \setminus \{j_1, \ldots, j_m\}) \vdash P)}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \mathsf{unlock}\ s_{j_1}, \ldots, s_{j_m}; P}\ \tau_{unlock}$$

Figure 4: Typing system for correctness proof. Note that the rules τ_{newP} and τ_{newA} refer to the initial honest process P_0'.

(v) A is an $Init$-adversary, so it is a closed process. Thus $\mathrm{fv}(A) = \emptyset$.

(vi) A is by definition under no lock in A, thus by definition $A, \emptyset$ satisfy condition (vi)

We can thus apply the preliminary result we just established to conclude that $(\mathcal{E}_0, \mathcal{E}_0(\mathcal{S}_0), [], \emptyset) \vdash A$. □

Lemma 2 (Typability of P_0)**.**

$$(\mathcal{E}_0, \mathcal{E}_0(\mathcal{S}_0), [], \emptyset) \vdash P_0$$

Proof sketch. Let Q be a subprocess of P_0, σ a substituion from variables to patterns, ρ a mapping from names and variables to patterns, H a conjunction of fact, ι a set of variables, ϕ a tuple of terms, and λ a set of indices. We first prove by induction on the size of Q, that if

(i) ρ binds all the free names and variables of Q, H, ι and ϕ;

(ii) $(\mathrm{bn}(Q) \cup \mathrm{bv}(Q)) \cap \mathrm{dom}(\rho) = \emptyset$;

(iii) σ is a closed substitution;

(iv) $i \in \lambda$ if and only if Q is in the scope of $\mathsf{lock}\ \ldots s_i \ldots$ in P;

(v) $\mathcal{C}_0 \supseteq [\![Q]\!]\rho H \iota \overline{\phi} \lambda$;

(vi) $\forall \mathsf{message}(\xi, M, N) \in H$, $\mathsf{message}(\xi\sigma, M\sigma, N\sigma)$ can be derived from $\mathcal{C}_0$

(vii) $\mathsf{attacker}(\overline{\phi}\sigma, attch[]) \in \mathcal{F}_0$,

then

$$(\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash Q.$$

To conclude the proof of Lemma 2 we then need to show that $\rho = \mathcal{E}_0$, σ *s.t.* $\mathrm{dom}(\sigma) = \emptyset$, $H = \mathsf{true}$, $\iota = []$, $\phi = \mathcal{E}_0(\mathcal{S}_0)$ and $\emptyset$ satisfy conditions (i)-(vii).

(i) Since by hypotheses $\mathrm{fv}(P'_0) = \emptyset$ and $\mathrm{fn}(P'_0) \subseteq \mathrm{dom}(\mathcal{E}_0)$ by construction, ρ binds the free names and variables of P, ι, H and ϕ.

(ii) By definition σ is a closed substitution.

(iii) By construction, $\mathrm{dom}(\mathcal{E}_0) = \mathrm{fn}(P'_0) \cup \mathrm{cells}(P'_0) \cup \{attch\}$, and by hypothesis $\mathrm{bn}(P'_0) \cap \mathrm{fn}(P'_0) = \emptyset$. Thus $(\mathrm{bn}(P_0) \cup \mathrm{bv}(P_0)) \cap \mathrm{dom}(\mathcal{E}_0) = \emptyset$.

(iv) P is not under any lock in P, thus $\emptyset$ satisfies condition (iii).

(v) By definition $\mathcal{C}_0 \supseteq [\![P]\!]\rho H \iota \overline{\phi} \lambda$.

(vi) by definition $H\sigma = \mathsf{true}$, and thus $H\sigma$ can trivially be derived from $\mathcal{C}_0$.

(vii) By construction, $\mathsf{attacker}(\overline{\mathcal{E}_0(\mathcal{S}_0)}, attch[]) \in \mathcal{C}_0$. So in particular, we have that $\mathsf{attacker}(\overline{\phi\sigma}, attch[]) \in \mathcal{F}_0$.

Thus, P, ρ, σ, H, ι, ϕ and $\emptyset$ satisfy the conditions of our induction result according to which $(\mathcal{E}_0, \mathcal{E}_0(\mathcal{S}_0), [], \emptyset) \vdash P$. □

Lemma 3 (Subject reduction)**.** *Let* $(\mathcal{E}, \mathcal{S}, \mathcal{Q}) \to (\mathcal{F}, \mathcal{T}, \mathcal{R})$ *be a valid instrumented transition such that no* $[s \mapsto M]$ *occurs in* $\mathcal{Q}$*, names and variables are bound at most once in* $\mathcal{Q}$*, and* $cells(Q) \subseteq \{s_1, \dots, s_n\}$*. If* $(\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\imath), \lambda) \vdash Q$ *for all* $(Q, \imath, \lambda) \in \mathcal{Q}$*, then* $(\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\jmath), \nu) \vdash R$ *for all* $(R, \jmath, \nu) \in \mathcal{R}$*.*

Proof sketch. The proof is done by case analysis on the rule that fired the transition $(\mathcal{E}, \mathcal{S}, \mathcal{Q}) \to (\mathcal{F}, \mathcal{T}, \mathcal{R})$. □

Theorem 1. *Consider the instrumented trace*

$$tr = (\mathcal{E}_0, \mathcal{S}_0, \{(P_0 \mid A, [], \emptyset)\}) \ \to^* \ (\mathcal{E}, \mathcal{S}, \mathcal{Q} \cup \{(Q, \iota, \lambda)\})$$

If $Q = \mathsf{out}(M, N); Q'$ *then*

$$\mathsf{message}(\overline{\mathcal{E}(\mathcal{S})}, \mathcal{E}(M), \mathcal{E}(N)) \in \mathcal{F}_0 \qquad \textit{and} \qquad \mathsf{attacker}(\overline{\mathcal{E}(\mathcal{S})}, attch[]) \in \mathcal{F}_0.$$

Proof. Consider the instrumented trace

$$\begin{array}{rl}(\mathcal{E}_0, \mathcal{S}_0, \{(P_0 \mid A, [], \emptyset)\}) \ = \ (\mathcal{E}_0, \mathcal{S}_0, \mathcal{Q}_0) \ \to \ (\mathcal{E}_1, \mathcal{S}_1, \mathcal{Q}_1) \ \to \ \dots \ \to & \\ (\mathcal{E}_n, \mathcal{S}_n, \mathcal{Q}_n) \ = \ (\mathcal{E}, \mathcal{S}, \mathcal{Q} \cup \{(Q, \iota, \lambda)\}) & \end{array}$$

We prove by induction on i, that for all $i \in \{0, \dots, n\}$

$$\mathsf{attacker}(\overline{\mathcal{E}_i(\mathcal{S}_i)}, attch[]) \in \mathcal{F}_0 \qquad \text{and} \qquad \forall (R, \imath, \nu) \in \mathcal{Q}_i \ (\mathcal{E}_i, \mathcal{E}_i(\mathcal{S}_i), \mathcal{E}_i(\imath), \nu) \vdash R$$

Base case $(i = 0)$**.** By definition of the StatVerif compiler we have that $\mathsf{attacker}(\overline{\mathcal{E}_0(\mathcal{S}_0)}, attch[]) \in \mathcal{C}_0$ and thus $\mathsf{attacker}(\overline{\mathcal{E}_0(\mathcal{S}_0)}, attch[]) \in \mathcal{F}_0$. Moreover, by Lemma 1 we have $(\mathcal{E}_0, \mathcal{E}_0(\mathcal{S}_0), [], \emptyset) \vdash A$, and by Lemma 2 we have $(\mathcal{E}_0, \mathcal{E}_0(\mathcal{S}_0), [], \emptyset) \vdash P$. Thus, according to the typing rule τ_{par},

$$(\mathcal{E}_0, \mathcal{E}_0(\mathcal{S}_0), [], \emptyset) \vdash A \mid P_0.$$

Inductive case $(i = n)$**.** By inductive hypothesis we know that the last transition satisfies the hypotheses of Lemma 3 according to which we have that $\mathsf{attacker}(\mathcal{E}_n(\mathcal{S}_n), \mathcal{E}_n(attch)) \in \mathcal{F}_0$, and $(\mathcal{E}_n, \mathcal{E}_n(\mathcal{S}_n), \mathcal{E}_n(\imath), \nu) \vdash R$ for all $(R, \imath, \nu) \in \mathcal{Q}$.

This concludes our induction and gives us

$$(\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\iota), \lambda) \vdash Q \quad \text{and} \quad \mathsf{attacker}(\overline{\mathcal{E}(\mathcal{S})}, attch[]) \in \mathcal{F}_0.$$

But then, by rule τ_{out} we know that $\mathsf{message}(\overline{\mathcal{E}(\mathcal{S})}, \mathcal{E}(M), \mathcal{E}(N)) \in \mathcal{F}_0$. □

Corollary 1 (Correctness *w.r.t.* secrecy)**.** *Let* M *a be message. If* P_0 *doesn't preserve the secrecy of* M *against Init, then* $\mathsf{attacker}(\overline{\mathcal{E}(\mathcal{S})}, \mathcal{E}(M)) \in \mathcal{F}_0$ *for some* $\mathcal{E}$ *and some* $\mathcal{S}$*.*

Proof. If P_0 doesn't preserve the secrecy of M against $Init$, then by definition of secrecy and according to Proposition 1, there exists an instrumented trace $tr = (\mathcal{E}_0, \mathcal{S}_0, \{(P_0 \mid A, [], \emptyset)\} \to^* \{(\mathcal{E}, \mathcal{S}, \mathcal{Q} \cup \{(Q, \iota, \lambda)\}$ *s.t.* $Q = \mathsf{out}(attch, M); Q'$. But Theorem 1 then tells us that $\mathsf{attacker}(\overline{\mathcal{E}(\mathcal{S})}, \mathcal{E}(M)) \in \mathcal{F}_0$. □

5 Case studies

To illustrate our method, we describe two case studies in detail. We show the processes in the StatVerif language, and use our rules to translate them to clauses. We have implemented StatVerif[2] on top of ProVerif and have used our tool to automatically verify the security properties of interest.

5.1 A simple security device

5.1.1 Description and process

Consider again the hardware device introduced in section 1. We take the process representing the device, together with the process representing Alice who creates the ciphertexts:

```
let device =
  new $s$;
    out($c, PKk$) | [$s \mapsto$ init] |
    ( ! lock $s$; in($c, x$); read $s$ as $y$;
        if $y$ = init then
          ( if $x$ = left then $s$ := left; unlock $s$ else
            if $x$ = right then $s$ := right; unlock $s$ ) ) |
    ( ! lock $s$; in($c, x$); read $s$ as $y$; let $z$ = adec($k, x$) in
        let $zl$ = projl($z$) in
        let $zr$ = projr($z$) in
        ( if $y$ = left then out($c, zl$); unlock $s$ else
          if $y$ = right then out($c, zr$); unlock $s$ ) )

let user =
  new $sl$; new $sr$; new $r$;
    out($c$, aenc($PKk, r, (sl, sr)$))

let system =new $k$; let $PKk$ = pk($k$) in device | ! user
```

Bob is the attacker. He receives the device and the ciphertexts, and chooses the messages to send to the device. We assume the term reductions:

$$\begin{aligned} \mathsf{adec}(u, \mathsf{aenc}(\mathsf{pk}(u), v, w)) &\to w \\ \mathsf{projl}((u, v)) &\to u \\ \mathsf{projr}((u, v)) &\to v \end{aligned}$$

The query is $\mathsf{query\ attacker}(vs, (sl[], sr[]))$, which asks if there is a reachable state vs in which the attacker may know both secrets sl and sr.

5.1.2 Clauses corresponding to the protocol

We apply the translation described in section 4. We will only show how to compute the clauses corresponding to the system process. In other words we will compute $[\![\text{system}]\!]\rho_0\,\mathsf{true}\,[\,]\phi_0\mathsf{false}$, where $\rho_0 = \{c \mapsto c[], \mathsf{left} \mapsto \mathit{left}[], \mathsf{right} \mapsto \mathit{right}[], \mathsf{init} \mapsto \mathit{init}[]\}$ and $\phi_0 = (\mathit{init}[])$.

[2] http://markryan.eu/research/StatVerif/

The $\mathsf{out}(c, PKk)$ on line 3 is translated to:

$$\mathsf{message}(\mathit{init}[], c[], \mathsf{pk}(k[]))$$

The $s := \mathsf{left}$ on line 6, with $\mathsf{in}(c, x)$ and $\mathsf{read}\ s\ \mathsf{as}\ y$ from line 4, generates:

$$\begin{aligned}
\mathsf{message}(\mathit{init}[], c[], \mathit{left}[])\ \wedge\ \mathsf{message}(\mathit{init}[], yc, ym) \wedge \\
\mathsf{message}(\mathit{init}[], zc, zm) \to \mathsf{message}(\mathit{left}[], zc, zm) \\
\mathsf{message}(\mathit{init}[], c[], \mathit{left}[])\ \wedge\ \mathsf{message}(\mathit{init}[], yc, ym) \wedge \\
\mathsf{attacker}(\mathit{init}[], zm) \to \mathsf{attacker}(\mathit{left}[], zm)
\end{aligned}$$

The $s := \mathsf{right}$ on line 7, with $\mathsf{in}(c, x)$ and $\mathsf{read}\ s\ \mathsf{as}\ y$ from line 4, generates:

$$\begin{aligned}
\mathsf{message}(\mathit{init}[], c[], \mathit{right}[])\ \wedge\ \mathsf{message}(\mathit{init}[], yc, ym) \wedge \\
\mathsf{message}(\mathit{init}[], zc, zm) \to \mathsf{message}(\mathit{right}[], zc, zm) \\
\mathsf{message}(\mathit{init}[], c[], \mathit{right}[])\ \wedge\ \mathsf{message}(\mathit{init}[], yc, ym) \wedge \\
\mathsf{attacker}(\mathit{init}[], zm) \to \mathsf{attacker}(\mathit{right}[], zm)
\end{aligned}$$

The $\mathsf{out}(c, zl)$ on line 11, with lines 8–10, is translated to:

$$\begin{aligned}
\mathsf{message}(\mathit{left}[], c[], \mathsf{aenc}(\mathsf{pk}(k[]), xr, (xsl, xsr))) \wedge \\
\mathsf{message}(\mathit{left}[], yc, ym) \to \mathsf{message}(\mathit{left}[], c[], xsl)
\end{aligned}$$

The $\mathsf{out}(c, zr)$ on line 12, with lines 8–10, is translated to:

$$\begin{aligned}
\mathsf{message}(\mathit{right}[], c[], \mathsf{aenc}(\mathsf{pk}(k[]), xr, (xsl, xsr))) \wedge \\
\mathsf{message}(\mathit{right}[], yc, ym) \to \mathsf{message}(\mathit{right}[], c[], xsr)
\end{aligned}$$

The output on line 15 is translated to:

$$\mathsf{message}(\mathit{init}[], c[], \mathsf{aenc}(\mathsf{pk}(k[]), r[], (sl[], sr[])))$$

5.1.3 Results of the analysis

We ran StatVerif on the StatVerif process corresponding to the hardware device, together with the query given above. StatVerif immediately concluded that the query is not satisfied (i.e., the protocol is secure). We made a few sanity checks, such as modifying the device to allow it to be configured again, and in that case StatVerif reported the valid attack as expected.

5.2 Contract signing protocol

A contract signing protocol allows a set of participants to exchange messages with each other in order to arrive at a state in which each of them has a pre-agreed contract signed by the others. An important property of contract signing protocols is fairness: no participant should be left in the position of having sent another participant his signature on the contract but not having received the others' signatures. To ensure fairness, a trusted party is necessary. Garay and Mackenzie [12] proposed such a protocol which, for efficiency, involves the trusted party only to resolve disputes. This protocol is based on private contract signatures. A private contract signature by A for B on m *w.r.t.* trusted party T acts as a promise by A to B to sign m.

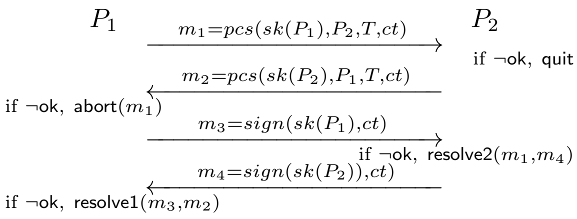
P_1
P_2
m_1=$pcs(sk(P_1)$,P_2,T,$ct)$
if ¬ok, quit
m_2=$pcs(sk(P_2)$,P_1,T,$ct)$
if ¬ok, abort(m_1)
m_3=$sign(sk(P_1)$,$ct)$
if ¬ok, resolve2(m_1,m_4)
m_4=$sign(sk(P_2))$,$ct)$
if ¬ok, resolve1(m_3,m_2)

Figure 5: The GM Main protocol (see [12])

In this section we will show how by applying our techniques to the two-party instance of the Garay and Mackenzie (GM) protocol, we automatically prove that the two-party version of this protocol satisfies fairness. To achieve this result we need no bound on the number of sessions/contracts or agents considered. In comparison, if we model the protocol by a plain ProVerif process using private channels to model the state of the trusted party, and run ProVerif on it, then the tool reports a false attack. It reports the same false attack even if only one contract is considered.

5.2.1 Description and process

The protocol is informally described in Figure 5 and consists of four subprotocols: Main, Abort1, Resolve2 and Resolve1. Usually, contract signers try to achieve the exchange without the help of the trusted party. They first exchange their promises to sign the contract (messages m_1 and m_2), and then exchange their actual signatures of the contract (messages m_3 and m_4). If for some reason they do not succeed in completing their exchange, the signers can ask the trusted party to arbitrate, by asking it either to abort or to resolve:

1. If P_2 doesn't receive P_1's promise, he just quits.
2. If P_1 doesn't receive P_2's promise, he asks the trusted party to abort. He includes his own promise in his request.
3. If P_2 (*resp.* P_1) doesn't receive P_1's (*resp.* P_2's) signature, he asks the trusted party to resolve. He includes his own signature, and P_1's (*resp.* P_2's) promise to sign the contract, in his request.

To deal with these requests, the trusted party records the following information for each contract *ct*:

- *status* - indicating whether it has solved any dispute regarding *ct* in the past. The possible values are *init*, *aborted*, *resolved*1 and *resolved*2.
- *sigs* - the acknowledgement of its decision, if it has made one. As we will now see, this is either its signature on the received abortion request or its signature on the two contracts.

On receipt of a request, the trusted party checks whether it had to solve a dispute on the same contract in the past. If it did ($status \neq init$), it just sends the decision it had taken and stored at that time (*sigs*). If it is the first request it receives, then:

- if it is an abortion request including the promise $m = \mathsf{pcs}(sk(x), y, T, ct)$, it acknowledges the request with the message $\mathsf{sign}(skT, m)$. It then updates the status of ct to *aborted* and stores its decision $\mathsf{sign}(skT, m)$;
- if it is a resolution request including the promise $\mathsf{pcs}(sk(x), y, T, ct)$ and the signature $\mathsf{sign}(sk(y), ct)$, it converts $x's$ promise into a valid signature $\mathsf{sign}(sk(x), ct)$ and replies with the message $\mathsf{sign}(skT, (\mathsf{sign}(sk(x), ct), \mathsf{sign}(sk(y), ct)))$. In other words, it sends to the plaintiff the signature corresponding to the promise. It also stores its reply in *sigs* and updates the *status* of ct to $resolved1$ or $resolved2$, according to which party sent the request.

The following process represents the trusted party:

```
1  let T = new $skT$; (out($c$, pk($skT$)) | ! C)
2  let C = new $status$; new $sigs$; new $ct$;
3            [$status \mapsto$ init] | [$sigs \mapsto$ init] |
4                 out($c$, $ct$); in($c$, $xpk1$); in($c$, $xpk2$);
5                 ( ! Abort1 | ! Resolve2 | ! Resolve1)
```

where Abort1, Resolve2 and Resolve1 are the subprocesses modelling the trusted party's behaviour upon an abortion or resolution request. After having published its public key (line 1), the trusted party can start handling contracts ($!C$). As we just discussed, for each contract it needs to create two new memory cells *status* and *sigs*, both of which it initialises to init (lines 2–3), to record information regarding the particular contract. It can then start replying to requests regarding this contract (line 5). The details of the subprocesses Abort1, Resolve2, and Resolve1 are given in appendix A.1.

As we explained in this section's introduction, it is important that the trusted party is fair to both parties. In other words, we want the following:

- if the participant P_1 has first contacted the trusted party and requested an abortion for contract ct, which was granted, then P_2 cannot obtain P_1's signature from the trusted party (*i.e.* he cannot receive the signature of P_1 on contract ct signed with the trusted party's secret key); and
- if the participant P_1 (*resp.* P_2) has first contacted the trusted party and requested a resolution for contract ct, which was granted, then P_2 (*resp.* P_1) cannot obtain from the trusted party an abortion confirmation (*i.e.* the promise of P_1 (*resp.* P_2) on contract ct signed with the trusted party's secret key).

These two properties can be combined and stated as a secrecy property, and can be formalised as

$$\mathsf{query\ attacker}(xs, (abortC, resolveC))$$

where $abortC = \mathsf{sign}(skT, \mathsf{pcs}(skP_1, \mathsf{pk}(skP_2), \mathsf{pk}(skT), ct))$ is the abortion acknowledgement, and $resolveC = \mathsf{sign}(skT, (\mathsf{sign}(skP_1, ct), \mathsf{sign}(skP_2, ct)))$ is the resolution acknowledgement.

Of course, there are many more properties that one would want a contract signing protocol to satisfy, but we only considered this one for the purpose of illustrating our techniques and showing that they work in non-trivial situations.

5.2.2 From unbounded number of cell names to bounded

Our translation only applies to processes with a bounded number of cell names, *i.e.* with no $[s \mapsto M]$ under a replication. However, in the GM protocol, the trusted party creates two cell names for each contract. So for an unbounded number of contracts it creates an unbounded number of cell names.

To prove that the GM protocol satisfies fairness using our techniques we make the following correct abstraction: the trusted party behaves according to the protocol only for a single contract *ct*. For this witnessing contract it creates the two cells it needs, and to any request regarding *ct* it replies and updates its memory according to the protocol. Thus, fairness of the protocol is proved only for *ct*. To requests concerning any other contract ct' it replies as if it were the first time it received any request regarding ct'.

So the process for the trusted party that we actually verify is the following:

```
let $\mathrm{T}' = \mathsf{new}\ skT;\ (\mathsf{out}(c, \mathsf{pk}(skT)) \mid \mathrm{C} \mid !\ \mathrm{C}')$
let $\mathrm{C}' = \ \mathsf{new}\ ct';\ \mathsf{out}(c, ct');\ \mathsf{in}(c, xpk1);\ \mathsf{in}(c, xpk2);$
              $(\ !\ \mathrm{Abort1}' \mid !\ \mathrm{Resolve2}' \mid !\ \mathrm{Resolve1}')$
```

where C is as we defined it in section 5.2.1 and Abort1′, Resolve2′, Resolve1′ are like Abort1, Resolve2, Resolve1 but with no checks on the status before replying. These subprocesses are given in detail in appendix A.2.

Proposition 2. *Let Init be a finite set of names. If T' satisfies fairness against Init, then T does too.*

Proof sketch. Let $attch \in Init$ and let A be an $Init$-attacker that breaks the fairness of T.

1) In any trace of T, A cannot read or write the trusted party's memory. Indeed, the cell names held by the trusted party are never sent on any channel and are under a restriction. So we can correctly consider A to be a plain process (no cell names occurring in it).

2) Because all the conditions before the trusted party's output are removed in Abort1′, Resolve2′, and Resolve1′, the following holds: for any trace tr of T such that

$$(\mathrm{fn}(T) \cup \mathrm{fn}(A), \emptyset, \{(T \mid A, \mathsf{false})\}) \rightarrow^* \\ (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\mathsf{out}(attch, M); Q, \mathsf{false})\})$$

there exists a trace tr' of T'

$$(\mathrm{fn}(T') \cup \mathrm{fn}(A), \emptyset, \{(T' \mid A, \mathsf{false})\}) \rightarrow^* \\ (\mathcal{E}', \mathcal{S}', \mathcal{P}' \cup \{(\mathsf{out}(attch, M); Q', \mathsf{false})\})$$

Now, since T doesn't preserve fairness against A, there exists a trace

$$(\mathrm{fn}(T) \cup \mathrm{fn}(A), \emptyset, \{(T \mid A, \mathsf{false})\}) \rightarrow^* \\ (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\mathsf{out}(attch, M); Q, \mathsf{false})\})$$

with $M = (abortC, resolveC)$. But then by 2) a trace of $T' \mid A$ breaking fairness also exists. □

5.2.3 Results of the analysis

We ran StatVerif on the StatVerif process corresponding to the two-party instance of the GM contract signing protocol. StatVerif concluded in less than 30 seconds that the query is not satisfied; in other words, that T' satisfies fairness. Thus, according to proposition 2, the two-party instance of the GM protocol satisfies fairness in the general case. The code for this example is available on the web[3].

6 Conclusion

We presented StatVerif, an extension of the ProVerif process calculus with constructs for explicit global state, and detailed the StatVerif compiler that takes processes written in this language and returns a corresponding set of clauses. We proved that the compiler is correct with respect to the operational semantics.

This machinery allows us to naturally write protocols that manipulate state in an intuitive high-level language. The language includes locked sections to allow sequences of state manipulations to be written conveniently and correctly. We demonstrated the language and tool on a couple of case studies. The effectiveness of our approach is further illustrated in some other papers. In [8], the same approach is used to automatically verify a simplified version of key management in Microsoft Bitlocker, and a protocol for making a digital envelope. Both of these protocols rely on the TPM and in particular on reasoning about mutable persistent state. In [13], our StatVerif tool is used to analyse Flicker [14] which also relies on the TPM.

The StatVerif compiler converts processes written in the language to clauses upon which ProVerif can be run. We have engineered the compiler carefully to result in clauses which do not introduce false attacks (as would be the case if one used the natural private-channel encoding of state). Moreover, ProVerif has a good chance to terminate on the translated clauses. Typically, it will do so easily if the state space is finite. For infinite state spaces, some further abstractions are likely to be necessary. We provided the clauses resulting from the translation of the case studies. ProVerif terminates easily on those examples, and we are able to prove their desired properties automatically.

We currently have an implementation of the StatVerif compiler[3]. If appropriate, we would like to contribute it to the ProVerif code-base. We also want to develop some further abstractions that are likely to be necessary in common situations.

Acknowledgements. We gratefully acknowledge financial support from Microsoft Corporation, and from EPSRC via the projects *Verifying Interoperability Requirements in Pervasive Systems* (EP/F033540/1) and *Analysing Security and Privacy Properties* (EP/H005501/1).

References

[1] J. Herzog, "Applying protocol analysis to security device interfaces," *IEEE Security & Privacy Magazine*, vol. 4, no. 4, pp. 84–87, July-Aug 2006.

[2] S. Mödersheim, "Abstraction by set-membership: verifying security protocols and web services with databases," in *Proc. 17th ACM Conference*

[3] `http://markryan.eu/research/StatVerif/`

on Computer and Communications Security (CCS'10). ACM, 2010, pp. 351–360.

[3] J. D. Guttman, "Fair exchange in strand spaces," *Journal of Automated Reasoning*, 2011, to appear.

[4] S. Delaune, S. Kremer, M. D. Ryan, and G. Steel, "A formal analysis of authentication in the TPM," in *Proc. 7th International Workshop on Formal Aspects in Security and Trust (FAST'10)*, Pisa, Italy, 2010.

[5] B. Blanchet, "An efficient cryptographic protocol verifier based on prolog rules," in *Proc. of the 14th IEEE Computer Security Foundations Workshop (CSFW'01)*. Cape Breton, Nova Scotia, Canada: IEEE Computer Society Press, Jun. 2001, pp. 82–96.

[6] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. 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)*, 2005, pp. 281–285.

[7] S. Fröschle and G. Steel, "Analysing PKCS#11 key management APIs with unbounded fresh data," in *Proc. Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security (ARSPA-WITS'09)*, ser. LNCS, P. Degano and L. Viganò, Eds., vol. 5511. York, UK: Springer, 2009, pp. 92–106, to appear.

[8] S. Delaune, S. Kremer, M. D. Ryan, and G. Steel, "Formal analysis of protocols based on TPM state registers," in *Proc. of the 24th IEEE Computer Security Foundations Symposium (CSF'11)*. IEEE Computer Society Press, 2011.

[9] B. Blanchet, "Automatic verification of correspondences for security protocols," *Journal of Computer Security*, vol. 17, no. 4, pp. 363–434, 2009.

[10] M. Abadi and C. Fournet, "Mobile values, new names, and secure communication," in *Proc. 28th Symposium on Principles of Programming Languages (POPL'01)*, H. R. Nielson, Ed. London, UK: ACM Press, 2001, pp. 104–115.

[11] B. Blanchet, "Automatic verification of correspondences for security protocols," *CoRR*, vol. abs/0802.3444, 2008.

[12] J. A. Garay, M. Jakobsson, and P. D. MacKenzie, "Abuse-free optimistic contract signing," in *Proceedings of the 19th Annual Cryptology Conference on Advances in Crypto*, ser. CRYPTO '99, London, UK, 1999, pp. 449–466.

[13] I. Batten, S. Xu, and M. Ryan, "Dynamic measurement and protected execution: model and analysis," in *Proceedings of the 8th International Symposium on Trustworthy Global Computing (TGC 2013)*, 2013.

[14] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and H. Isozaki, "Flicker: An execution infrastructure for tcb minimization," in *Proceedings of the ACM European Conference in Computer Systems (EuroSys)*, Apr. 2008.

A Contract signing

A.1 Contract signing: Trusted party with unbounded memory

In this section we detail the process of our language modelling the GM protocol without any restriction on the number of cell names held by the trusted party T. The overall process T representing the trusted party is followed by definitions of its subprocesses.

```
1  let T = new skT; (out(c, pk(skT)) | ! C)

2  let C =  new status; new sigs; new ct;
3           [status ↦ init] | [sigs ↦ init] |
4                out(c, ct); in(c, xpk1); in(c, xpk2);
5                ( ! Abort1 | ! Resolve2 | ! Resolve1 )
```

Abort1 If P_1 doesn't receive P_2's promise, he requests an abortion from T by sending it a message containing the information about the contract for which he requests the resolution, and of the form:

$$\underbrace{(\underbrace{\mathsf{abort}}_{xcmd},\underbrace{((\overbrace{ct}^{ycontract},\overbrace{(\mathsf{pk}(skP_1),\mathsf{pk}(skP_2))}^{yparties}),\overbrace{\mathsf{sign}(skP_1,(ct,(\mathsf{pk}(skP_1),\mathsf{pk}(skP_2))))}^{ysig})}_{y})}_{x}$$

Upon receipt of such a command (line 7), the trusted party executes the subprocess Abort1 which consists of:

- Extracting from x the parts *xcmd*, *ycontract*, *yparties*, and *ysig* (lines 8, 10–13, 16).
- Checking that it is an abortion request (line 9).
- Checking that it has a record for this contract with these participants (lines 14–15).
- Checking that the third component of x is a signature of the second (lines 17–18).

Once all these checks on the received message have succeeded, the request is handled:

- If the trusted party has already handled an abortion request regarding ct, (*i.e.* $ystatus$ = $\mathsf{aborted}$ at line 20) then it retrieves (line 21) and replies with (line 22) its previous decision regarding this contract.
- Otherwise, if this is the first request regarding ct, (*i.e.* $ystatus$ = init at line 23), it updates the status of ct to $\mathsf{aborted}$ (line 24), and stores (line 24) and sends (line 25) the acknowledgement $\mathsf{sign}(skT, y)$.

```
6  let Abort1 =
7    lock $status, sigs$; in$(c, x)$;
8    let $xcmd$ = projl$(x)$ in
9    if $xcmd$ = abort then
10     let $y$ = projr$(x)$ in
11     let $yl$ = projl$(y)$ in
12     let $ycontract$ = projl$(yl)$ in
13     let $yparties$ = projr$(yl)$ in
14     if $yparties = (xpk1, xpk2)$ then
15       if $ycontract = ct$ then
16         let $ysig$ = projr$(y)$ in
17         let $ym$ = check_getmsg$(xpk1, ysig)$ in
18         if $ym = yl$ then
19           read $status$ as $ystatus$;
20           ( if $ystatus$ = aborted then
21               read $sigs$ as $ysigs$;
22               out$(c, ysigs)$; unlock $ystatus, ysigs$ else
23             if $ystatus$ = init then
24               $ystatus, ysigs$ := aborted, sign$(skT, y)$;
25               out$(c,$ sign$(skT, y))$; unlock $ystatus, ysigs$ )
```

Resolve2 If P_2 doesn't receive P_1's signature, he asks T to resolve by sending it a message containing the information about the contract for which he requests the resolution. This message is of the form:

$$\underbrace{(\underbrace{\mathsf{resolve2}}_{\mathit{xcmd}}, (\underbrace{\mathsf{pcs}(skP_1, \mathsf{pk}(skP2), \mathsf{pk}(skT), ct)}_{\mathit{ypcs1}}, \underbrace{\mathsf{sign}(skP_2, \overbrace{ct}^{\mathit{ycontract}})}_{\mathit{ysig2}}))}_{x}$$

Upon receipt of such a command (line 27), the trusted party executes the subprocess Resolve2 which consists of:

- Extracting from x the parts *xcmd*, *ycontract*, *ypcs1*, and *ysig2* (lines 28 and 30–33).
- Checking that it is a resolution request (line 29).
- Checking that he has a record for this contract (line 34).
- Checking that the received promise (*ypcs1*) and the received signature (*ysig2*) concern the same contract (*ycontract*) (lines 33–36).

Once all these checks on the received message have succeeded, it handles the request:

- If the trusted party has already handled a resolve2 request regarding ct, (*i.e.* *ystatus* = resolved2 at line 38) then it retrieves (line 39) and replies with (line 40) its previous decision regarding this contract.
- Otherwise, if this is the first request regarding ct, (*i.e.* *ystatus* = init at line 41), it updates the status of ct to resolved2, converts the promise *ypcs1* into a valid signature *ysig1* (line 42) and stores (line 43) and sends (line 44) the acknowledgement $\mathsf{sign}(skT, (ysig1, ysig2))$.

```
let Resolve2 =
  lock $status, sigs$; in($c, x$);
  let $xcmd$ = projl($x$) in
  if $xcmd$ = resolve2 then
    let $y$ = projr($x$) in
    let $ypcs1$ = projl($y$) in
    let $ysig2$ = projr($y$) in
    let $ycontract$ = check_getmsg($xpk2, ysig2$) in
    if $ycontract = ct$ then
      let $ycheck$ = checkpcs($ct, xpk1, xpk2$, pk($skT$), $ypcs1$) in
      if $ycheck$ = ok then
        read $status$ as $ystatus$;
        ( if $ystatus$ = resolved2 then
            read $sigs$ as $ysigs$;
            out($c, ysigs$); unlock $status, sigs$ else
          if $ystatus$ = init then
            let $ysig1$ = convertpcs($skT, ypcs1$) in
            $status, sigs$ := resolved2, sign($skT$, ($ysig1, ysig2$));
            out($c$, sign($skT$, ($ysig1, ysig2$))); unlock $status, sigs$ )
```

Resolve1 If P_1 doesn't receive P_2's signature, he asks T to resolve. Upon receipt of such a command, the trusted party executes the subprocess Resolve1 which is analogous to Resolve2 above.

```
let Resolve1 =
  lock $status, sigs$; in($c, x$);
  let $xcmd$ = projl($x$) in
  if $xcmd$ = resolve1 then
    let $y$ = projr($x$) in
    let $ysig1$ = projl($y$) in
    let $ypcs2$ = projr($y$) in
    let $ycontract$ = check_getmsg($xpk1, ysig1$) in
    if $ycontract = ct$ then
      let $ycheck$ = checkpcs($ct, xpk2, xpk1$, pk($skT$), $ypcs2$) in
      if $ycheck$ = ok then
        read $status$ as $ystatus$;
        ( if $ystatus$ = resolved1 then
            read $sigs$ as $ysigs$;
            out($c, ysigs$); unlock $status, sigs$ else
          if $ystatus$ = init then
            let $ysig2$ = convertpcs($skT, ypcs2$) in
            $status, sigs$ := resolved1, sign($skT$, ($ysig1, ysig2$));
            out($c$, sign($skT$, ($ysig1, ysig2$))); unlock $status, sigs$ )
```

A.2 Contract signing: Trusted party with bounded memory

In this section we detail the process of our language model that we actually verified to prove that the 2-party GM protocol satisfies fairness. As we established in Section 5.2.2, T' is a correct abstraction of T *w.r.t.* fairness. Note that in what follows, we took care that the altered process C'' does not handle our witnessing contract ct, which is handled by C'.

```
let $\mathrm{T}' = \mathsf{new}\ skT;\ \mathsf{new}\ status;\ \mathsf{new}\ sigs;\ (\mathsf{out}(c, \mathsf{pk}(skT))\ |\ !\ \mathrm{C}'\ |\ \mathrm{C}'')$

let $\mathrm{C}' =$  $[status \mapsto \mathsf{init}]\ |\ [sigs \mapsto \mathsf{init}]\ |$
           $\mathrm{Abort1}\ \left[{}^{\mathsf{pk}(skA)}/_{xpk1}, {}^{\mathsf{pk}(skB)}/_{xpk2}\right]\ |$
           $\mathrm{Resolve2}\ \left[{}^{\mathsf{pk}(skA)}/_{xpk1}, {}^{\mathsf{pk}(skB)}/_{xpk2}\right]\ |$
           $\mathrm{Resolve1}\ \left[{}^{\mathsf{pk}(skA)}/_{xpk1}, {}^{\mathsf{pk}(skB)}/_{xpk2}\right]$

let $\mathrm{C}'' =$  $\mathsf{new}\ ct';\ \mathsf{out}(c, ct')\ |$
           $!\ \mathrm{Abort1}'\ |\ !\ \mathrm{Resolve2}'\ |\ !\ \mathrm{Resolve1}'$
```

Abort1′ is built from Abort1 just by removing lines 19–24. Because Abort1′ replies to a request without checking the status of the requested contract, it will always reply with the abort acknowledgement.

```
let $\mathrm{Abort1}' =$
  $\mathsf{lock}\ status, sigs;\ \mathsf{in}(c, x);$
  let $xcmd = \mathsf{projl}(x)$ in
  if $xcmd = \mathsf{abort}$ then
    let $y = \mathsf{projr}(x)$ in
    let $yl = \mathsf{projl}(y)$ in
    let $ycontract = \mathsf{projl}(yl)$ in
    let $yparties = \mathsf{projr}(yl)$ in
    if $yparties = (xpk1, xpk2)$ then
      if $ycontract = ct$ then
        let $ysig = \mathsf{projr}(y)$ in
        let $ym = \mathsf{check\_getmsg}(xpk1, ysig)$ in
        if $ym = yl$ then
          $\mathsf{out}(c, \mathsf{sign}(skT, y));\ \mathsf{unlock}\ ystatus, ysigs$
```

Resolve2′ is built from Resolve2 just by removing lines 37–41 and 43. Because Resolve2′ replies to a request without checking the status of the requested contract, it will always reply with the resolve acknowledgement.

```
let $\mathrm{Resolve2}' =$
  $\mathsf{lock}\ status, sigs;\ \mathsf{in}(c, x);$
  let $xcmd = \mathsf{projl}(x)$ in
  if $xcmd = \mathsf{resolve2}$ then
    let $y = \mathsf{projr}(x)$ in
    let $ypcs1 = \mathsf{projl}(y)$ in
    let $ysig2 = \mathsf{projr}(y)$ in
    let $ycontract = \mathsf{check\_getmsg}(xpk2, ysig2)$ in
    if $ycontract = ct$ then
      let $ycheck = \mathsf{checkpcs}(ct, xpk1, xpk2, \mathsf{pk}(skT), ypcs1)$ in
      if $ycheck = \mathsf{ok}$ then
        let $ysig1 = \mathsf{convertpcs}(skT, ypcs1)$ in
        $\mathsf{out}(c, \mathsf{sign}(skT, (ysig1, ysig2)));\ \mathsf{unlock}\ status, sigs)$
```

Resolve1′ is built from Resolve1 just by removing lines 56–60 and 62. Because Resolve1′ replies to a request without checking the status of the requested contract, it will always reply with the resolve acknowledgement.

```
let Resolve1' =
  lock status, sigs; in(c, x);
  let xcmd = projl(x) in
  if xcmd = resolve1 then
    let y = projr(x) in
    let ysig1 = projl(y) in
    let ypcs2 = projr(y) in
    let ycontract = check_getmsg(xpk1, ysig1) in
    if ycontract = ct then
      let ycheck = checkpcs(ct, xpk2, xpk1, pk(skT), ypcs2) in
      if ycheck = ok then
        let ysig2 = convertpcs(skT, ypcs2) in
        out(c, sign(skT, (ysig1, ysig2))); unlock status, sigs
```

B Correctness

Let $P_0' = \mathsf{new}\ \tilde{m}([s_1 \mapsto M_1] \mid \cdots \mid [s_n \mapsto M_n] \mid P_0)$ be a closed process and A an Init-adversary *s.t.* $\mathit{attch} \in \mathit{Init}$. Without loss of generality, we can assume that the free cell names in A are included in the free cell names of P_0', and that the set of bounded cell names of A is empty. The reason is that any other cell name of the intruder can be equivalently encoded using channel names as described by Milner. Moreover, we assume that names and varibales are bound at most once in $P_0' \mid A$, and names and variables are not both bound and free in $P_0' \mid A$.

Let $\mathcal{C}_0$ be the set of clauses generated by StatVerif when applied to P_0', and $\mathcal{F}_0$ the set of closed facts derivable from $\mathcal{C}_0$. Let $\mathcal{S}_0 = \{s_1 \mapsto M_1, \ldots, s_n \mapsto M_n\}$. Let $\mathcal{E}_0$ be the environment such that

- $\mathrm{fn}(P_0') \cup \mathrm{cells}(P_0') \cup \mathrm{fn}(A) = \mathrm{dom}(\mathcal{E}_0)$,
- $\mathcal{E}_0(a) = a[]$ for all $a \in \mathrm{fn}(P_0') \cup \mathrm{cells}(P_0') \cup \{\mathit{attch}\}$,
- $\mathcal{E}_0(a) = \mathit{attn}[]$ for all $a \in \mathrm{fn}(A) \smallsetminus \{\mathit{attch}\}$.

Let $\mathcal{S} = \{s_1 \mapsto K_1, \ldots, s_n \mapsto K_n\}$ be a state. $\overline{\mathcal{S}}$ denotes the ordered representation of $\mathcal{S}$, defined as $\overline{\mathcal{S}} = (K_1, \ldots, K_n)$.

We will say that a state $\mathcal{R}$ is a predecessor of the state $\mathcal{S}$, denoted $\mathcal{R} \leq \mathcal{S}$ if:

$$\begin{array}{ll} & \mathsf{attacker}(\overline{\mathcal{R}}, \mathit{attch}[]) \in \mathcal{F}_0 \\ \wedge & \forall M, N\ \mathsf{message}(\overline{\mathcal{R}}, M, N) \in \mathcal{F}_0 \ \Rightarrow\ \mathsf{message}(\overline{\mathcal{S}}, M, N) \in \mathcal{F}_0 \\ \wedge & \forall M\ \mathsf{attacker}(\overline{\mathcal{R}}, M) \in \mathcal{F}_0 \ \Rightarrow\ \mathsf{attacker}(\overline{\mathcal{S}}, M) \in \mathcal{F}_0 \end{array}$$

B.1 Preliminaries

Lemma 4. *Let M be a term, $\mathcal{S}$ a state (a function from $\{s_1, \ldots, s_n\}$ to patterns), and $\mathcal{E}$ an environment (a function from names and variables to patterns). If*

(i) $\forall a \in \mathrm{fn}(M)$, $\mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0$*; and*

(ii) $\forall x \in \mathrm{fv}(M)$, $\mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0$.

Then

$$\mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(M)) \in \mathcal{F}_0.$$

Proof. We prove this by induction on the depth of M.

Base case ($d = 1$).
$\Rightarrow$ M is a name or a variable
$\overset{\text{Def.}}{\Rightarrow}$ $M \in \text{fn}(M) \cup \text{fv}(M)$
$\overset{\text{Hyp.}}{\Rightarrow}$ $\mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(M)) \in \mathcal{F}_0$

Inductive case ($d > 1$). In this case, $M = f(M_1, \dots, M_n)$ for some constructor f and some terms $M_1, \dots, M_n$. Let $i \in \{1, \dots, n\}$.

(i) $\overset{\text{Def.}}{\Rightarrow}$ $\text{fn}(M_i) \subseteq \text{fn}(M)$
$\overset{\text{Hyp.}}{\Rightarrow}$ $\forall a \in \text{fn}(M_i)$, $\mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0$

(ii) $\overset{\text{Def.}}{\Rightarrow}$ $\text{fv}(M_i) \subseteq \text{fv}(M)$
$\overset{\text{Hyp.}}{\Rightarrow}$ $\forall x \in \text{fv}(M_i)$, $\mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0$

$\overset{\text{I.H.}}{\Rightarrow}$ $\mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(M_i)) \in \mathcal{F}_0$

Thus for all $i \in \{1, \dots, n\}$, $\mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(M_i)) \in \mathcal{F}_0$. Now by definition, $\mathcal{C}_0$ contains the following clause:
$\mathsf{attacker}(xs, xm_1) \wedge \dots \wedge \mathsf{attacker}(xs, xm_n) \rightarrow \mathsf{attacker}(xs, f(xm_1, \dots, xm_n))$
Thus, by resolution we have that

$$\mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(M)) = \mathsf{attacker}(\overline{\mathcal{S}}, f(\mathcal{E}(M_1), \dots, \mathcal{E}(M_n))) \in \mathcal{F}_0. \qquad \square$$

Lemma 5 (Substitution lemma)**.** *Let $\mathcal{E}$ be an environment (a function from names and variables to patterns), x a variable such that $x \notin \text{dom}(\mathcal{E})$, and M a term. Let $\mathcal{E}' = \mathcal{E} \cup \{x \mapsto \mathcal{E}(M)\}$.*

1. *For all N, $\mathcal{E}(N\{M/x\}) = \mathcal{E}'(N)$;*

2. *For all $\mathcal{S}$ (from $\{s_1, \dots, s_n\}$ to patterns), Q, ι, λ such that $\text{bn}(Q) \cap \text{fn}(M) = \emptyset$ and $x \notin \text{bv}(Q)$, if $(\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash Q$ then $(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q\{M/x\}$.*

Proof.
1. We prove the first statement by induction on the depth d of N.

Base case $d = 1$. In that case, N is either a variable or a name.
Let us first suppose that $N \neq x$. Then

$$\mathcal{E}(N\{M/x\}) \quad = \quad \mathcal{E}(N) \quad = \quad \mathcal{E}'(N)$$

Now, if $N = x$, then

$$\mathcal{E}(N\{M/x\}) \quad = \mathcal{E}(x\{M/x\}) \quad = \quad \mathcal{E}(M) \quad = \quad \mathcal{E}'(x) \quad = \quad \mathcal{E}'(N)$$

Inductive case ($d > 1$). In that case, $N = f(N_1, \ldots, N_k)$ for some constructor f and some terms $N_1, \ldots, N_k$, and

$$\begin{aligned}
\mathcal{E}(N\{M/x\}) &\overset{\text{Def.}}{=} \mathcal{E}(f(N_1\{M/x\}, \ldots, N_k\{M/x\})) \\
&\overset{\text{Def.}}{=} f(\mathcal{E}(N_1\{M/x\}), \ldots, \mathcal{E}(N_k\{M/x\})) \\
&\overset{\text{I.H.}}{=} f(\mathcal{E}'(N_1), \ldots, \mathcal{E}'(N_k)) \\
&\overset{\text{Def.}}{=} \mathcal{E}'(f(N_1, \ldots, N_k)) \\
&\overset{\text{Def.}}{=} \mathcal{E}'(N)
\end{aligned}$$

2. We prove the second statement by induction on the depth d of Q.

Base case ($d = 0$). In that case $Q = 0$, thus $Q\{M/x\} = Q = 0$, and according to our typing system

$$\frac{}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash 0\ (= Q\{M/x\})}\ \tau_{nil}$$

Inductive case ($d > 0$). We proceed by case analysis on the structure of Q.

Case $Q = Q_1 \mid Q_2$.

$$\begin{array}{cl}
\overset{\text{Hyp}}{\Rightarrow} & (\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash Q_1 \mid Q_2 \\
& \text{bn}(Q_1 \mid Q_2) \cap \text{fn}(M) = \emptyset \\
& x \notin \text{bv}(Q_1 \mid Q_2) \\
\overset{\tau_{par}, \text{bn}(), \text{bv}()}{\Rightarrow} & \lambda = \emptyset \\
& (\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash Q_1 \\
& (\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash Q_2 \\
& \text{bn}(Q_1) \cap \text{fn}(M) = \emptyset \wedge \text{bn}(Q_2) \cap \text{fn}(M) = \emptyset \\
& x \notin \text{bv}(Q_1) \wedge x \notin \text{bv}(Q_2) \\
\overset{\text{I.H.}}{\Rightarrow} & \lambda = \emptyset \\
& (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q_1\{M/x\} \\
& (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q_2\{M/x\} \\
\overset{\tau_{par}}{\Rightarrow} & (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q_1\{M/x\} \mid Q_2\{M/x\}\ (= Q\{M/x\})
\end{array}$$

Case $Q = !Q'$.

$$\begin{array}{cl}
\overset{\text{Hyp}}{\Rightarrow} & (\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash !Q' \\
& \text{bn}(!Q') \cap \text{fn}(M) = \emptyset \\
& x \notin \text{bv}(!Q') \\
\overset{\tau_{repl}, \text{bn}(), \text{bv}()}{\Rightarrow} & \lambda = \emptyset \\
& (\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash Q' \\
& \text{bn}(Q') \cap \text{fn}(M) = \emptyset \\
& x \notin \text{bv}(Q') \\
\overset{\text{I.H.}}{\Rightarrow} & \lambda = \emptyset \\
& (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q'\{M/x\} \\
\overset{\tau_{repl}}{\Rightarrow} & (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash !Q'\{M/x\}\ (= Q\{M/x\})
\end{array}$$

Case $Q = \text{if } N_1 = N_2 \text{ then } Q_1 \text{ else } Q_2$ with $\mathcal{E}'(N_1) = \mathcal{E}'(N_2)$.

$$
\begin{array}{cl}
\overset{\text{Hyp}}{\Rightarrow} & \mathcal{E}'(N_1) = \mathcal{E}'(N_2) \\
& (\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash \text{if } N_1 = N_2 \text{ then } Q_1 \text{ else } Q_2 \\
& \text{bn}(\text{if } N_1 = N_2 \text{ then } Q_1 \text{ else } Q_2) \cap \text{fn}(M) = \emptyset \\
& x \notin \text{bv}(\text{if } N_1 = N_2 \text{ then } Q_1 \text{ else } Q_2) \\
\overset{\tau_{if},\text{bn}(),\text{bv}()}{\Rightarrow} & \mathcal{E}'(N_1) = \mathcal{E}'(N_2) \\
& (\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash Q_1 \\
& (\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash Q_2 \\
& \text{bn}(Q_1) \cap \text{fn}(M) = \emptyset \ \wedge \ \text{bn}(Q_2) \cap \text{fn}(M) = \emptyset \\
& x \notin \text{bv}(Q_1) \ \wedge \ x \notin \text{bv}(Q_2) \\
\overset{\text{I.H.}}{\Rightarrow} & \mathcal{E}'(N_1) = \mathcal{E}'(N_2) \\
& (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q_1\{M/x\} \\
& (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q_2\{M/x\} \\
\overset{\text{Lem. 5.1}}{\Rightarrow} & \mathcal{E}(N_1\{M/x\}) = \mathcal{E}'(N_1) = \mathcal{E}'(N_2) = \mathcal{E}(N_2\{M/x\}) \\
& (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q_1\{M/x\} \ \wedge \ (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q_2\{M/x\} \\
\overset{\tau_{if}}{\Rightarrow} & (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \text{if } N_1\{M/x\} = N_2\{M/x\} \\
& \qquad \text{then } Q_1\{M/x\} \text{ else } Q_2\{M/x\} \ (= Q\{M/x\})
\end{array}
$$

Case $Q = \text{if } N_1 = N_2 \text{ then } Q_1 \text{ else } Q_2$ **with** $\mathcal{E}'(N_1) \neq \mathcal{E}'(N_2)$**.**

$$
\begin{array}{cl}
\overset{\text{Hyp}}{\Rightarrow} & \mathcal{E}'(N_1) \neq \mathcal{E}'(N_2) \\
& (\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash \text{if } N_1 = N_2 \text{ then } Q_1 \text{ else } Q_2 \\
& \text{bn}(\text{if } N_1 = N_2 \text{ then } Q_1 \text{ else } Q_2) \cap \text{fn}(M) = \emptyset \\
& x \notin \text{bv}(\text{if } N_1 = N_2 \text{ then } Q_1 \text{ else } Q_2) \\
\overset{\tau_{if},\text{bn}(),\text{bv}()}{\Rightarrow} & \mathcal{E}'(N_1) \neq \mathcal{E}'(N_2) \\
& (\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash Q_2 \\
& \text{bn}(Q_2) \cap \text{fn}(M) = \emptyset \\
& x \notin \text{bv}(Q_2) \\
\overset{\text{I.H.}}{\Rightarrow} & \mathcal{E}'(N_1) \neq \mathcal{E}'(N_2) \\
& (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q_2\{M/x\} \\
\overset{\text{Lem. 5.1}}{\Rightarrow} & \mathcal{E}(N_1\{M/x\}) = \mathcal{E}'(N_1) \neq \mathcal{E}'(N_2) = \mathcal{E}(N_2\{M/x\}) \\
& (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q_2\{M/x\} \\
\overset{\tau_{if}}{\Rightarrow} & (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \text{if } N_1\{M/x\} = N_2\{M/x\} \text{ then } Q_1\{M/x\} \\
& \qquad\qquad\qquad\qquad \text{else } Q_2\{M/x\} \ (= Q\{M/x\})
\end{array}
$$

Case $Q = \text{let } y = g(N_1, \ldots, N_k) \text{ in } Q_1 \text{ else } Q_2$**.** Note first that since by hypothesis $x \notin \text{bv}(Q)$ and by definition $y \in \text{bv}(Q)$, then $y \neq x$.

$$
\begin{array}{ll}
\overset{\text{Hyp}}{\Rightarrow} & (\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash \text{let } y = g(N_1, \ldots, N_k) \text{ in } Q_1 \text{ else } Q_2 \\
 & \text{bn}(\text{let } y = g(N_1, \ldots, N_k) \text{ in } Q_1 \text{ else } Q_2) \cap \text{fn}(M) = \emptyset \\
 & x \notin \text{bv}(\text{let } y = g(N_1, \ldots, N_k) \text{ in } Q_1 \text{ else } Q_2) \\
\overset{\tau_{let}, \text{bn}(), \text{bv}()}{\Rightarrow} & \bigwedge_{\{N | g(\mathcal{E}'(N_1), \ldots, \mathcal{E}'(N_k)) \rightarrow N\}} (\mathcal{E}' \cup \{y \mapsto N\}, \mathcal{S}, \iota, \lambda) \vdash Q_1 \\
 & (\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash Q_2 \\
 & \text{bn}(Q_1) \cap \text{fn}(M) = \emptyset \ \wedge \ \text{bn}(Q_2) \cap \text{fn}(M) = \emptyset \\
 & x \notin \text{bv}(Q_1) \ \wedge \ x \notin \text{bv}(Q_2) \\
\overset{\text{I.H.}}{\Rightarrow} & \bigwedge_{\{N | g(\mathcal{E}'(N_1), \ldots, \mathcal{E}'(N_k)) \rightarrow N\}} (\mathcal{E} \cup \{y \mapsto N\}, \mathcal{S}, \iota, \lambda) \vdash Q_1\{M/x\} \\
 & (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q_2\{M/x\} \\
\overset{\text{Lem. 5.1}}{\Rightarrow} & \bigwedge_{\left\{\begin{array}{l} N \mid g(\mathcal{E}(N_1\{M/x\}), \\ \quad \ldots, \mathcal{E}(N_k\{M/x\})) \rightarrow N \end{array}\right\}} (\mathcal{E} \cup \{y \mapsto N\}, \mathcal{S}, \iota, \lambda) \vdash Q_1\{M/x\} \\
 & (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q_2\{M/x\} \\
\overset{\tau_{let}}{\Rightarrow} & (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \text{let } y = g(N_1\{M/x\}, \ldots, N_k\{M/x\}) \text{ in } Q_1\{M/x\} \\
 & \qquad \text{else } Q_2\{M/x\} \ (\overset{y \neq x}{=} Q\{M/x\})
\end{array}
$$

Case $Q = \mathsf{new}\ a; Q'$ **with** $a \in \text{bn}(P_0')$**.**

$$
\begin{array}{ll}
\overset{\text{Hyp}}{\Rightarrow} & (\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash \mathsf{new}\ a; Q' \\
 & \text{bn}(\mathsf{new}\ a; Q') \cap \text{fn}(M) = \emptyset \\
 & x \notin \text{bv}(\mathsf{new}\ a; Q') \\
\overset{\tau_{newP}, \text{bn}(), \text{bv}()}{\Rightarrow} & (\mathcal{E}' \cup \{a \mapsto a[\iota]\}, \mathcal{S}, \iota, \lambda) \vdash Q' \\
 & \text{bn}(Q') \cap \text{fn}(M) = \emptyset \\
 & x \notin \text{bv}(Q') \\
\overset{\text{I.H.}}{\Rightarrow} & (\mathcal{E} \cup \{a \mapsto a[\iota]\}, \mathcal{S}, \iota, \lambda) \vdash Q'\{M/x\} \\
\overset{\tau_{newP}}{\Rightarrow} & (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \mathsf{new}\ a; Q'\{M/x\} \ (= Q\{M/x\})
\end{array}
$$

Case $Q = \mathsf{new}\ a; Q'$ **with** $a \notin \text{bn}(P_0')$**.**

$$
\begin{array}{ll}
\overset{\text{Hyp}}{\Rightarrow} & (\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash \mathsf{new}\ a; Q' \\
 & \text{bn}(\mathsf{new}\ a; Q') \cap \text{fn}(M) = \emptyset \\
 & x \notin \text{bv}(\mathsf{new}\ a; Q') \\
\overset{\tau_{newA}, \text{bn}(), \text{bv}()}{\Rightarrow} & (\mathcal{E}' \cup \{a \mapsto attn[]\}, \mathcal{S}, \iota, \lambda) \vdash Q' \\
 & \text{bn}(Q') \cap \text{fn}(M) = \emptyset \\
 & x \notin \text{bv}(Q') \\
\overset{\text{I.H.}}{\Rightarrow} & (\mathcal{E} \cup \{a \mapsto attn[]\}, \mathcal{S}, \iota, \lambda) \vdash Q'\{M/x\} \\
\overset{\tau_{newA}}{\Rightarrow} & (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \mathsf{new}\ a; Q'\{M/x\} \ (= Q\{M/x\})
\end{array}
$$

Case $Q = \mathsf{out}(N_1, N_2); Q'$**.**

$$\begin{array}{rl}
\overset{\text{Hyp}}{\Rightarrow} & (\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash \mathsf{out}(N_1, N_2); Q' \\
& \mathrm{bn}(\mathsf{out}(N_1, N_2); Q') \cap \mathrm{fn}(M) = \emptyset \\
& x \notin \mathrm{bv}(\mathsf{out}(N_1, N_2); Q') \\
\overset{\tau_{out}, \mathrm{bn}(), \mathrm{bv}()}{\Rightarrow} & \mathsf{message}(\overline{\mathcal{S}}, \mathcal{E}'(N_1), \mathcal{E}'(N_2)) \in \mathcal{F}_0 \ \wedge \ (\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash Q' \\
& \mathrm{bn}(Q') \cap \mathrm{fn}(M) = \emptyset \\
& x \notin \mathrm{bv}(Q') \\
\overset{\text{I.H.}}{\Rightarrow} & \mathsf{message}(\overline{\mathcal{S}}, \mathcal{E}'(N_1), \mathcal{E}'(N_2)) \in \mathcal{F}_0 \ \wedge \ (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q'\{M/x\} \\
\overset{\text{Lem. 5.1}}{\Rightarrow} & \mathsf{message}(\overline{\mathcal{S}}, \mathcal{E}(N_1\{M/x\}), \mathcal{E}(N_2\{M/x\})) \in \mathcal{F}_0 \\
& \hfill \wedge \ (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q'\{M/x\} \\
\overset{\tau_{out}}{\Rightarrow} & (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \mathsf{out}(N_1\{M/x\}, N_2\{M/x\}); Q'\{M/x\} \ (= Q\{M/x\})
\end{array}$$

Case $Q = \mathsf{in}(N, y); Q'$**.** Note first that by hypothesis $x \notin \mathrm{bv}(Q)$ and by definition $y \in \mathrm{bv}(Q)$, thus $y \neq x$.

$$\begin{array}{rl}
\overset{\text{Hyp}}{\Rightarrow} & (\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash \mathsf{in}(N, y); Q' \\
& \mathrm{bn}(\mathsf{in}(N, y); Q') \cap \mathrm{fn}(M) = \emptyset \\
& x \notin \mathrm{bv}(\mathsf{in}(N, y); Q') \\
\overset{\tau_{in}, \mathrm{bn}(), \mathrm{bv}()}{\Rightarrow} & \mathrm{bn}(Q') \cap \mathrm{fn}(M) = \emptyset \\
& x \notin \mathrm{bv}(Q') \\
& \displaystyle \bigwedge_{\left\{ N' \,\middle|\, \begin{array}{l} \exists \mathcal{T}.\, \mathcal{S} \leq \mathcal{T} \\ \mathcal{T} = \mathcal{T}[j \mapsto \mathcal{S}(j) \mid j \in \lambda] \\ \mathsf{message}(\overline{\mathcal{T}}, \mathcal{E}'(N), N') \in \mathcal{F}_0 \end{array} \right\}} (\mathcal{E}' \cup \{y \mapsto N'\}, \mathcal{T}, (N' :: \iota), \lambda) \vdash Q' \\
\overset{\text{I.H.}}{\Rightarrow} & \displaystyle \bigwedge_{\left\{ N' \,\middle|\, \begin{array}{l} \exists \mathcal{T}.\, \mathcal{S} \leq \mathcal{T} \\ \mathcal{T} = \mathcal{T}[j \mapsto \mathcal{S}(j) \mid j \in \lambda] \\ \mathsf{message}(\overline{\mathcal{T}}, \mathcal{E}'(N), N') \in \mathcal{F}_0 \end{array} \right\}} (\mathcal{E} \cup \{y \mapsto N'\}, \mathcal{T}, (N' :: \iota), \lambda) \vdash Q'\{M/x\} \\
\overset{\text{Lem. 5.1}}{\Rightarrow} & \displaystyle \bigwedge_{\left\{ N' \,\middle|\, \begin{array}{l} \exists \mathcal{T}.\, \mathcal{S} \leq \mathcal{T} \\ \mathcal{T} = \mathcal{T}[j \mapsto \mathcal{S}(j) \mid j \in \lambda] \\ \mathsf{message}(\overline{\mathcal{T}}, \mathcal{E}(N\{M/x\}), N') \in \mathcal{F}_0 \end{array} \right\}} (\mathcal{E} \cup \{y \mapsto N'\}, \mathcal{T}, (N' :: \iota), \lambda) \vdash Q'\{M/x\} \\
\overset{\tau_{in}}{\Rightarrow} & (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \mathsf{in}(N\{M/x\}, y); Q'\{M/x\} \ (\overset{y \neq x}{=} Q\{M/x\})
\end{array}$$

Case $Q = \mathsf{lock}\ s_{j_1}, \ldots, s_{j_m}; Q'$**.**

$$\overset{\text{Hyp}}{\Rightarrow} \quad \begin{array}{l} (\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash \mathsf{lock}\ s_{j_1}, \ldots, s_{j_m}; Q' \\ \mathrm{bn}(\mathsf{lock}\ s_{j_1}, \ldots, s_{j_m}; Q') \cap \mathrm{fn}(M) = \emptyset \\ x \notin \mathrm{bv}(\mathsf{lock}\ s_{j_1}, \ldots, s_{j_m}; Q') \end{array}$$

$$\overset{\tau_{lock}, \mathrm{bn}(), \mathrm{bv}()}{\Rightarrow} \quad \begin{array}{l} \displaystyle\bigwedge_{\{\mathcal{T} \mid \substack{\mathcal{S} \le \mathcal{T} \\ \mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda]\}}} (\mathcal{E}', \mathcal{T}, \iota, \lambda \cup \{j_1, \ldots, j_m\}) \vdash Q' \\ \mathrm{bn}(Q') \cap \mathrm{fn}(M) = \emptyset \\ x \notin \mathrm{bv}(Q') \end{array}$$

$$\overset{\text{I.H.}}{\Rightarrow} \quad \bigwedge_{\{\mathcal{T} \mid \substack{\mathcal{S} \le \mathcal{T} \\ \mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda]\}}} (\mathcal{E}, \mathcal{T}, \iota, \lambda \cup \{j_1, \ldots, j_m\}) \vdash Q'\{M/x\}$$

$$\overset{\tau_{lock}}{\Rightarrow} \quad (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \mathsf{lock}\ s_{j_1}, \ldots, s_{j_m}; Q'\{M/x\}\ (= Q\{M/x\})$$

Case $Q = \mathsf{unlock}\ s_{j_1}, \ldots, s_{j_m}; Q'$**.**

$$\overset{\text{Hyp}}{\Rightarrow} \quad \begin{array}{l} (\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash \mathsf{unlock}\ s_{j_1}, \ldots, s_{j_m}; Q' \\ \mathrm{bn}(\mathsf{unlock}\ s_{j_1}, \ldots, s_{j_m}; Q') \cap \mathrm{fn}(M) = \emptyset \\ x \notin \mathrm{bv}(\mathsf{unlock}\ s_{j_1}, \ldots, s_{j_m}; Q') \end{array}$$

$$\overset{\tau_{lock}, \mathrm{bn}(), \mathrm{bv}()}{\Rightarrow} \quad \begin{array}{l} \displaystyle\bigwedge_{\{\mathcal{T} \mid \substack{\mathcal{S} \le \mathcal{T} \\ \mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda]\}}} (\mathcal{E}', \mathcal{T}, \iota, \lambda \setminus \{j_1, \ldots, j_m\}) \vdash Q' \\ \mathrm{bn}(Q') \cap \mathrm{fn}(M) = \emptyset \\ x \notin \mathrm{bv}(Q') \end{array}$$

$$\overset{\text{I.H.}}{\Rightarrow} \quad \bigwedge_{\{\mathcal{T} \mid \substack{\mathcal{S} \le \mathcal{T} \\ \mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda]\}}} (\mathcal{E}, \mathcal{T}, \iota, \lambda \setminus \{j_1, \ldots, j_m\}) \vdash Q'\{M/x\}$$

$$\overset{\tau_{lock}}{\Rightarrow} \quad (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \mathsf{unlock}\ s_{j_1}, \ldots, s_{j_m}; Q'\{M/x\}\ (= Q\{M/x\})$$

Case $Q = s_{j_1}, \ldots, s_{j_m} := N_1, \ldots, N_m; Q'$**.**

$$\overset{\text{Hyp}}{\Rightarrow} \quad \begin{array}{l} (\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash s_{j_1}, \ldots, s_{j_m} := N_1, \ldots, N_m; Q' \\ \mathrm{bn}(s_{j_1}, \ldots, s_{j_m} := N_1, \ldots, N_m; Q') \cap \mathrm{fn}(M) = \emptyset \\ x \notin \mathrm{bv}(s_{j_1}, \ldots, s_{j_m} := N_1, \ldots, N_m; Q') \end{array}$$

$$\overset{\tau_{write}, \mathrm{bn}(), \mathrm{bv}()}{\Rightarrow} \quad \begin{array}{l} \displaystyle\bigwedge_{\{\mathcal{T} \mid \substack{\mathcal{S} \le \mathcal{T} \\ \mathcal{T} = \mathcal{T}[j \mapsto \mathcal{S}(j) \mid j \in \lambda]\}}} \begin{array}{r} \mathcal{T} \le \mathcal{T}[j_k \mapsto \mathcal{E}'(N_k) \mid 1 \le k \le m] \wedge \\ (\mathcal{E}', \mathcal{T}[j_k \mapsto \mathcal{E}'(N_k) \mid 1 \le k \le m], \iota, \lambda) \vdash Q' \end{array} \\ \mathrm{bn}(Q') \cap \mathrm{fn}(M) = \emptyset \\ x \notin \mathrm{bv}(Q') \end{array}$$

$$\overset{\text{I.H.}}{\Rightarrow} \quad \bigwedge_{\{\mathcal{T} \mid \substack{\mathcal{S} \le \mathcal{T} \\ \mathcal{T} = \mathcal{T}[j \mapsto \mathcal{S}(j) \mid j \in \lambda]\}}} \begin{array}{r} \mathcal{T} \le \mathcal{T}[_{j_k} \mapsto \mathcal{E}'(N_k) \mid 1 \le k \le m] \wedge \\ (\mathcal{E}, \mathcal{T}[_{j_k} \mapsto \mathcal{E}'(N_k) \mid 1 \le k \le m], \\ \iota, \lambda) \vdash Q'\{M/x\} \end{array}$$

$$\overset{\text{Lem. 5.1}}{\Rightarrow} \quad \bigwedge_{\{\mathcal{T} \mid \substack{\mathcal{S} \le \mathcal{T} \\ \mathcal{T} = \mathcal{T}[j \mapsto \mathcal{S}(j) \mid j \in \lambda]\}}} \begin{array}{r} \mathcal{T} \le \mathcal{T}[j_k \mapsto \mathcal{E}(N_k\{M/x\}) \mid 1 \le k \le m] \wedge \\ (\mathcal{E}, \mathcal{T}[j_k \mapsto \mathcal{E}(N_k\{M/x\}) \mid \\ 1 \le k \le m], \iota, \lambda) \vdash Q'\{M/x\} \end{array}$$

$$\overset{\tau_{write}}{\Rightarrow} \quad \begin{array}{r} (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash s_{j_1}, \ldots, s_{j_m} := N_1\{M/x\}, \ldots, N_m\{M/x\}; \\ Q'\{M/x\}\ (\overset{s_{j_1} \neq x, \ldots, s_{j_m} \neq x}{=} Q\{M/x\}) \end{array}$$

Case $Q = \mathsf{read}\ s_{j_1}, \ldots, s_{j_m}\ \mathsf{as}\ y_1, \ldots, y_m; Q'$**.** Note first that by hypothesis $x \notin \mathrm{bv}(Q)$ and by definition $y_1, \ldots, y_m \in \mathrm{bv}(Q)$, thus $x \notin \{y_1, \ldots, y_m\}$

$$\overset{\text{Hyp}}{\Rightarrow} \quad \begin{array}{l} (\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash \mathsf{read}\ s_{j_1}, \ldots, s_{j_m}\ \mathsf{as}\ y_1, \ldots, y_m; Q' \\ \mathrm{bn}(\mathsf{read}\ s_{j_1}, \ldots, s_{j_m}\ \mathsf{as}\ y_1, \ldots, y_m; Q') \cap \mathrm{fn}(M) = \emptyset \\ x \notin \mathrm{bv}(\mathsf{read}\ s_{j_1}, \ldots, s_{j_m}\ \mathsf{as}\ y_1, \ldots, y_m; Q') \end{array}$$

$$\overset{\tau_{read}, \mathrm{bn}(), \mathrm{bv}()}{\Rightarrow} \quad \begin{array}{l} \bigwedge_{\substack{\{\mathcal{T} \mid \ \mathcal{S} \leq \mathcal{T} \\ \mathcal{T} = \mathcal{T}[j \mapsto \mathcal{S}(j) \mid j \in \lambda]\}}} \begin{array}{l} (\mathcal{E}' \cup \{y_k \mapsto \mathcal{T}(j_k) \mid 1 \leq k \leq m\}, \mathcal{T}, \\ \quad (\mathcal{T}(j_1) :: \ldots \mathcal{T}(j_m) :: \iota), \lambda) \vdash Q' \end{array} \\ \mathrm{bn}(Q') \cap \mathrm{fn}(M) = \emptyset \\ x \notin \mathrm{bv}(Q') \end{array}$$

$$\overset{\text{I.H.}}{\Rightarrow} \quad \bigwedge_{\substack{\{\mathcal{T} \mid \ \mathcal{S} \leq \mathcal{T} \\ \mathcal{T} = \mathcal{T}[j \mapsto \mathcal{S}(j) \mid j \in \lambda]\}}} \begin{array}{l} (\mathcal{E} \cup \{y_k \mapsto \mathcal{T}(j_k) \mid 1 \leq k \leq m\}, \mathcal{T}, \\ \quad (\mathcal{T}(j_1) :: \ldots \mathcal{T}(j_m) :: \iota), \lambda) \vdash Q'\{M/x\} \end{array}$$

$$\overset{\tau_{read}}{\Rightarrow} \quad \begin{array}{l} (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \mathsf{read}\ s_{j_1}, \ldots, s_{j_m}\ \mathsf{as}\ y_1, \ldots, y_m; \\ \qquad Q'\{M/x\}\ (\overset{x \notin \{s_{j_1}, y_1, \ldots, s_{j_m}, y_m\}}{=} Q\{M/x\}) \quad \square \end{array}$$

Lemma 6 (Type propagation)**.** *Let Q, $\mathcal{E}$ (from names and variables to patterns), $\mathcal{S}$ (from cell names to patterns), $\mathcal{T}$ (from cell names to patterns), ι, and λ such that $\mathcal{S} \leq \mathcal{T}$ and $\mathcal{T} = \mathcal{T}[j \mapsto \mathcal{S}(j) \mid j \in \lambda]$*

$$(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q \ \Rightarrow\ (\mathcal{E}, \mathcal{T}, \iota, \lambda) \vdash Q.$$

Proof. We prove this by induction on the depth d of the proof of $(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q$.

Case $d = 0$**.** In this case, $Q = 0$, and according to our type rule τ_{nil}, $(\mathcal{E}, \mathcal{T}, \iota, \lambda) \vdash Q$.

Case $d > 0$**.** We proceed by case analysis on the structure of Q.

Case $Q = Q_1 \mid Q_2$**.**

$$\begin{array}{ll} \overset{\text{Hyp.}}{\Rightarrow} & (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q_1 \mid Q_2 \\ \overset{\tau_{par}}{\Rightarrow} & (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q_1 \ \wedge\ (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q_2 \ \wedge\ \lambda = \emptyset \\ \overset{\text{I.H.}}{\Rightarrow} & (\mathcal{E}, \mathcal{T}, \iota, \lambda) \vdash Q_1 \ \wedge\ (\mathcal{E}, \mathcal{T}, \iota, \lambda) \vdash Q_2 \ \wedge\ \lambda = \emptyset \\ \overset{\tau_{par}}{\Rightarrow} & (\mathcal{E}, \mathcal{T}, \iota, \lambda) \vdash Q_1 \mid Q_2\ (= Q) \end{array}$$

Case $Q = !Q'$**.**

$$\begin{array}{ll} \overset{\text{Hyp.}}{\Rightarrow} & (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash !Q' \\ \overset{\tau_{repl}}{\Rightarrow} & (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q' \ \wedge\ \lambda = \emptyset \\ \overset{\text{I.H.}}{\Rightarrow} & (\mathcal{E}, \mathcal{T}, \iota, \lambda) \vdash Q' \ \wedge\ \lambda = \emptyset \\ \overset{\tau_{repl}}{\Rightarrow} & (\mathcal{E}, \mathcal{T}, \iota, \lambda) \vdash !Q'\ (= Q) \end{array}$$

Case $Q = \text{if } M = N \text{ then } Q_1 \text{ else } Q_2$ **with** $\mathcal{E}(M) = \mathcal{E}(N)$**.**

$$\begin{array}{ll} \overset{\text{Hyp.}}{\Rightarrow} & (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \text{if } M = N \text{ then } Q_1 \text{ else } Q_2 \ \wedge\ \mathcal{E}(M) = \mathcal{E}(N) \\ \overset{\tau_{if}}{\Rightarrow} & (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q_1 \ \wedge\ (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q_2 \ \wedge\ \mathcal{E}(M) = \mathcal{E}(N) \\ \overset{\text{I.H.}}{\Rightarrow} & (\mathcal{E}, \mathcal{T}, \iota, \lambda) \vdash Q_1 \ \wedge\ (\mathcal{E}, \mathcal{T}, \iota, \lambda) \vdash Q_2 \ \wedge\ \mathcal{E}(M) = \mathcal{E}(N) \\ \overset{\tau_{if}}{\Rightarrow} & (\mathcal{E}, \mathcal{T}, \iota, \lambda) \vdash \text{if } M = N \text{ then } Q_1 \text{ else } Q_2\ (= Q) \end{array}$$

Case $Q = \text{if } M = N \text{ then } Q_1 \text{ else } Q_2$ **with** $\mathcal{E}(M) \neq \mathcal{E}(N)$**.**

$$\begin{aligned}
&\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \text{if } M = N \text{ then } Q_1 \text{ else } Q_2 \ \wedge\ \mathcal{E}(M) \neq \mathcal{E}(N) \\
&\overset{\tau_{if}}{\Rightarrow} && (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q_2 \ \wedge\ \mathcal{E}(M) \neq \mathcal{E}(N) \\
&\overset{\text{I.H.}}{\Rightarrow} && (\mathcal{E}, \mathcal{T}, \iota, \lambda) \vdash Q_2 \ \wedge\ \mathcal{E}(M) \neq \mathcal{E}(N) \\
&\overset{\tau_{if}}{\Rightarrow} && (\mathcal{E}, \mathcal{T}, \iota, \lambda) \vdash \text{if } M = N \text{ then } Q_1 \text{ else } Q_2 \ (= Q)
\end{aligned}$$

Case $Q = \text{let } x = g(M_1, \ldots, M_n) \text{ in } Q_1 \text{ else } Q_2$**.**

$$\begin{aligned}
&\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \text{let } x = g(M_1, \ldots, M_n) \text{ in } Q_1 \text{ else } Q_2 \\
&\overset{\tau_{let}}{\Rightarrow} && \bigwedge_{\{M | g(\mathcal{E}(M_1), \ldots, \mathcal{E}(M_1)) \to M\}} (\mathcal{E} \cup \{x \mapsto M\}, \mathcal{S}, \iota, \lambda) \vdash Q_1 \ \wedge\ (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q_2 \\
&\overset{\text{I.H.}}{\Rightarrow} && \bigwedge_{\{M | g(\mathcal{E}(M_1), \ldots, \mathcal{E}(M_1)) \to M\}} (\mathcal{E} \cup \{x \mapsto M\}, \mathcal{T}, \iota, \lambda) \vdash Q_1 \ \wedge\ (\mathcal{E}, \mathcal{T}, \iota, \lambda) \vdash Q_2 \\
&\overset{\tau_{let}}{\Rightarrow} && (\mathcal{E}, \mathcal{T}, \iota, \lambda) \vdash \text{let } x = g(M_1, \ldots, M_n) \text{ in } Q_1 \text{ else } Q_2 \ (= Q)
\end{aligned}$$

Case $Q = \mathsf{new}\ a; Q'$ **with** $a \in \text{bn}(P'_0)$**.**

$$\begin{aligned}
&\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \mathsf{new}\ a; Q' \\
&\overset{\tau_{newP}}{\Rightarrow} && (\mathcal{E} \cup \{a \mapsto a[\iota]\}, \mathcal{S}, \iota, \lambda) \vdash Q' \\
&\overset{\text{I.H.}}{\Rightarrow} && (\mathcal{E} \cup \{a \mapsto a[\iota]\}, \mathcal{T}, \iota, \lambda) \vdash Q' \\
&\overset{\tau_{newP}}{\Rightarrow} && (\mathcal{E}, \mathcal{T}, \iota, \lambda) \vdash \mathsf{new}\ a; Q' \ (= Q)
\end{aligned}$$

Case $Q = \mathsf{new}\ a; Q'$ **with** $a \notin \text{bn}(P'_0)$**.**

$$\begin{aligned}
&\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \mathsf{new}\ a; Q' \\
&\overset{\tau_{newA}}{\Rightarrow} && (\mathcal{E} \cup \{a \mapsto attn[]\}, \mathcal{S}, \iota, \lambda) \vdash Q' \\
&\overset{\text{I.H.}}{\Rightarrow} && (\mathcal{E} \cup \{a \mapsto attn[]\}, \mathcal{T}, \iota, \lambda) \vdash Q' \\
&\overset{\tau_{newA}}{\Rightarrow} && (\mathcal{E}, \mathcal{T}, \iota, \lambda) \vdash \mathsf{new}\ a; Q' \ (= Q)
\end{aligned}$$

Case $Q = \mathsf{out}(M, N); Q'$**.**

$$\begin{aligned}
&\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \mathsf{out}(M, N); Q' \\
&\overset{\tau_{out}}{\Rightarrow} && \mathsf{message}(\overline{\mathcal{S}}, \mathcal{E}(M), \mathcal{E}(N)) \in \mathcal{F}_0 \wedge (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash Q' \\
&\overset{\text{I.H.}}{\Rightarrow} && \mathsf{message}(\overline{\mathcal{S}}, \mathcal{E}(M), \mathcal{E}(N)) \in \mathcal{F}_0 \wedge (\mathcal{E}, \mathcal{T}, \iota, \lambda) \vdash Q' \\
&\overset{\mathcal{S} \leq \mathcal{T}}{\Rightarrow} && \mathsf{message}(\overline{\mathcal{T}}, \mathcal{E}(M), \mathcal{E}(N)) \in \mathcal{F}_0 \wedge (\mathcal{E}, \mathcal{T}, \iota, \lambda) \vdash Q' \\
&\overset{\tau_{out}}{\Rightarrow} && (\mathcal{E}, \mathcal{T}, \iota, \lambda) \vdash \mathsf{out}(M, N); Q' \ (= Q)
\end{aligned}$$

Case $Q = \mathsf{in}(M, x); Q'$**.**

$$\begin{aligned}
&\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \mathsf{in}(M, x); Q' \\
&\overset{\tau_{in}}{\Rightarrow} && \forall \mathcal{U} \forall N\ (\mathcal{S} \leq \mathcal{U} \ \wedge\ \mathcal{U} = \mathcal{U}[j \mapsto \mathcal{S}(j) \mid j \in \lambda] \wedge \\
& && \quad \mathsf{message}(\overline{\mathcal{U}}, \mathcal{E}(M), N) \in \mathcal{F}_0) \ \Rightarrow \\
& && \qquad (\mathcal{E} \cup \{x \mapsto N\}, \mathcal{U}, N :: \iota, \lambda) \vdash Q' \\
&\overset{\substack{\text{transitivity of } \leq \ \wedge \\ \mathcal{T} = \mathcal{T}[j \mapsto \mathcal{S}(j) \mid j \in \lambda]}}{\Rightarrow} && \forall \mathcal{U} \forall N\ (\mathcal{T} \leq \mathcal{U} \ \wedge\ \mathcal{U} = \mathcal{U}[j \mapsto \mathcal{S}(j) \mid j \in \lambda] \wedge \\
& && \quad \mathsf{message}(\overline{\mathcal{U}}, \mathcal{E}(M), N) \in \mathcal{F}_0) \ \Rightarrow \\
& && \qquad (\mathcal{E} \cup \{x \mapsto N\}, \mathcal{U}, N :: \iota, \lambda) \vdash Q' \\
&\overset{\mathcal{T} = \mathcal{T}[j \mapsto \mathcal{S}(j) | j \in \lambda]}{\Rightarrow} && \forall \mathcal{U} \forall N\ (\mathcal{T} \leq \mathcal{U} \ \wedge\ \mathcal{U} = \mathcal{U}[j \mapsto \mathcal{T}(j) \mid j \in \lambda] \wedge \\
& && \quad \mathsf{message}(\overline{\mathcal{U}}, \mathcal{E}(M), N) \in \mathcal{F}_0) \ \Rightarrow \\
& && \qquad (\mathcal{E} \cup \{x \mapsto N\}, \mathcal{U}, N :: \iota, \lambda) \vdash Q' \\
&\overset{\tau_{in}}{\Rightarrow} && (\mathcal{E}, \mathcal{T}, \iota, \lambda) \vdash \mathsf{in}(M, x); Q' \ (= Q)
\end{aligned}$$

Case $Q = \mathsf{lock}\ s_{j_1},\ldots,s_{j_m};Q'$**.**

$$\begin{array}{rl}
\overset{\text{Hyp.}}{\Rightarrow} & (\mathcal{E},\mathcal{S},\iota,\lambda) \vdash \mathsf{lock}\ s_{j_1},\ldots,s_{j_m};Q' \\
\overset{\tau_{lock}}{\Rightarrow} & \forall \mathcal{U}\ (\mathcal{S} \le \mathcal{U}\ \wedge\ \mathcal{U} = \mathcal{U}[k \mapsto \mathcal{S}(k) \mid k \in \lambda])\ \Rightarrow \\
& \qquad\qquad (\mathcal{E},\mathcal{U},\iota,\lambda \cup \{j_1,\ldots,j_m\}) \vdash Q' \\
\overset{\substack{\text{transitivity of } \le\ \wedge \\ \mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda]}}{\Rightarrow} & \forall \mathcal{U}\ (\mathcal{T} \le \mathcal{U}\ \wedge\ \mathcal{U} = \mathcal{U}[k \mapsto \mathcal{S}(k) \mid k \in \lambda])\ \Rightarrow \\
& \qquad\qquad (\mathcal{E},\mathcal{U},\iota,\lambda \cup \{j_1,\ldots,j_m\}) \vdash Q' \\
\overset{\mathcal{T}=\mathcal{T}[k\mapsto\mathcal{S}(k)|k\in\lambda]}{\Rightarrow} & \forall \mathcal{U}\ (\mathcal{T} \le \mathcal{U}\ \wedge\ \mathcal{U} = \mathcal{U}[k \mapsto \mathcal{T}(k) \mid k \in \lambda])\ \Rightarrow \\
& \qquad\qquad (\mathcal{E},\mathcal{U},\iota,\lambda \cup \{j_1,\ldots,j_m\}) \vdash Q' \\
\overset{\tau_{lock}}{\Rightarrow} & (\mathcal{E},\mathcal{T},\iota,\lambda) \vdash \mathsf{lock}\ s_{j_1},\ldots,s_{j_m};Q'\ (= Q)
\end{array}$$

Case $Q = \mathsf{unlock}\ s_{j_1},\ldots,s_{j_m};Q'$**.**

$$\begin{array}{rl}
\overset{\text{Hyp.}}{\Rightarrow} & (\mathcal{E},\mathcal{S},\iota,\lambda) \vdash \mathsf{unlock}\ s_{j_1},\ldots,s_{j_m};Q' \\
\overset{\tau_{unlock}}{\Rightarrow} & \forall \mathcal{U}\ (\mathcal{S} \le \mathcal{U}\ \wedge\ \mathcal{U} = \mathcal{U}[k \mapsto \mathcal{S}(k) \mid k \in \lambda])\ \Rightarrow \\
& \qquad\qquad (\mathcal{E},\mathcal{U},\iota,\lambda \smallsetminus \{j_1,\ldots,j_m\}) \vdash Q' \\
\overset{\substack{\text{transitivity of } \le\ \wedge \\ \mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda]}}{\Rightarrow} & \forall \mathcal{U}\ (\mathcal{T} \le \mathcal{U}\ \wedge\ \mathcal{U} = \mathcal{U}[k \mapsto \mathcal{S}(k) \mid k \in \lambda])\ \Rightarrow \\
& \qquad\qquad (\mathcal{E},\mathcal{U},\iota,\lambda \smallsetminus \{j_1,\ldots,j_m\}) \vdash Q' \\
\overset{\mathcal{T}=\mathcal{T}[k\mapsto\mathcal{S}(k)|k\in\lambda]}{\Rightarrow} & \forall \mathcal{U}\ (\mathcal{T} \le \mathcal{U}\ \wedge\ \mathcal{U} = \mathcal{U}[k \mapsto \mathcal{T}(k) \mid k \in \lambda])\ \Rightarrow \\
& \qquad\qquad (\mathcal{E},\mathcal{U},\iota,\lambda \smallsetminus \{j_1,\ldots,j_m\}) \vdash Q' \\
\overset{\tau_{unlock}}{\Rightarrow} & (\mathcal{E},\mathcal{T},\iota,\lambda) \vdash \mathsf{unlock}\ s_{j_1},\ldots,s_{j_m};Q'\ (= Q)
\end{array}$$

Case $Q = s_{j_1},\ldots,s_{j_m} := M_1,\ldots,M_m;Q'$**.**

$$\begin{array}{rl}
\overset{\text{Hyp.}}{\Rightarrow} & (\mathcal{E},\mathcal{S},\iota,\lambda) \vdash s_{j_1},\ldots,s_{j_m} := M_1,\ldots,M_m;Q' \\
& \forall \mathcal{U}\ (\mathcal{S} \le \mathcal{U}\ \wedge\ \mathcal{U} = \mathcal{U}[k \mapsto \mathcal{S}(k) \mid k \in \lambda])\ \Rightarrow \\
\overset{\tau_{write}}{\Rightarrow} & \quad (\mathcal{U} \le \mathcal{U}[j_k \mapsto \mathcal{E}(M_k) \mid 1 \le k \le m]\ \wedge \\
& \qquad (\mathcal{E},\mathcal{U}[j_k \mapsto \mathcal{E}(M_k) \mid 1 \le k \le m],\iota,\lambda) \vdash Q') \\
\overset{\substack{\text{transitivity of } \le\ \wedge \\ \mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda]}}{\Rightarrow} & \forall \mathcal{U}\ (\mathcal{T} \le \mathcal{U}\ \wedge\ \mathcal{U} = \mathcal{U}[k \mapsto \mathcal{S}(k) \mid k \in \lambda])\ \Rightarrow \\
& \quad (\mathcal{U} \le \mathcal{U}[j_k \mapsto \mathcal{E}(M_k) \mid 1 \le k \le m]\ \wedge \\
& \qquad (\mathcal{E},\mathcal{U}[j_k \mapsto \mathcal{E}(M_k) \mid 1 \le k \le m],\iota,\lambda) \vdash Q') \\
& \forall \mathcal{U}\ (\mathcal{T} \le \mathcal{U}\ \wedge\ \mathcal{U} = \mathcal{U}[k \mapsto \mathcal{T}(k) \mid k \in \lambda])\ \Rightarrow \\
\overset{\mathcal{T}=\mathcal{T}[k\mapsto\mathcal{S}(k)|k\in\lambda]}{\Rightarrow} & \quad (\mathcal{U} \le \mathcal{U}[j_k \mapsto \mathcal{E}(M_k) \mid 1 \le k \le m]\ \wedge \\
& \qquad (\mathcal{E},\mathcal{U}[j_k \mapsto \mathcal{E}(M_k) \mid 1 \le k \le m],\iota,\lambda) \vdash Q') \\
\overset{\tau_{write}}{\Rightarrow} & (\mathcal{E},\mathcal{T},\iota,\lambda) \vdash s_{j_1},\ldots,s_{j_m} := M_1,\ldots,M_m;Q'\ (= Q)
\end{array}$$

Case $Q = \mathsf{read}\ s_{j_1},\ldots,s_{j_m}\ \mathsf{as}\ x_1,\ldots,x_m;Q'$**.**

$$
\begin{aligned}
&\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \mathsf{read}\ s_{j_1}, \ldots, s_{j_m}\ \mathsf{as}\ x_1, \ldots, x_m; Q' \\
&\overset{\tau_{read}}{\Rightarrow} && \forall \mathcal{U}\ (\mathcal{S} \leq \mathcal{U}\ \wedge\ \mathcal{U} = \mathcal{U}[k \mapsto \mathcal{S}(k) \mid k \in \lambda])\ \Rightarrow \\
& && (\mathcal{E} \cup \{x_k \mapsto \mathcal{U}(j_k) \mid 1 \leq k \leq m\}, \mathcal{U}, \\
& && \qquad\qquad (\mathcal{U}(j_1) :: \cdots :: \mathcal{U}(j_m) :: \iota), \lambda) \vdash Q' \\
&\overset{\substack{\text{transitivity of } \leq\ \wedge \\ \mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda]}}{\Rightarrow} && \forall \mathcal{U}\ (\mathcal{T} \leq \mathcal{U}\ \wedge\ \mathcal{U} = \mathcal{U}[k \mapsto \mathcal{S}(k) \mid k \in \lambda])\ \Rightarrow \\
& && (\mathcal{E} \cup \{x_k \mapsto \mathcal{U}(j_k) \mid 1 \leq k \leq m\}, \mathcal{U}, \\
& && \qquad\qquad (\mathcal{U}(j_1) :: \cdots :: \mathcal{U}(j_m) :: \iota), \lambda) \vdash Q' \\
&\overset{\mathcal{T}=\mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda]}{\Rightarrow} && \forall \mathcal{U}\ (\mathcal{T} \leq \mathcal{U}\ \wedge\ \mathcal{U} = \mathcal{U}[k \mapsto \mathcal{T}(k) \mid k \in \lambda])\ \Rightarrow \\
& && (\mathcal{E} \cup \{x_k \mapsto \mathcal{U}(j_k) \mid 1 \leq k \leq m\}, \mathcal{U}, \\
& && \qquad\qquad (\mathcal{U}(j_1) :: \cdots :: \mathcal{U}(j_m) :: \iota), \lambda) \vdash Q' \\
&\overset{\tau_{read}}{\Rightarrow} && (\mathcal{E}, \mathcal{T}, \iota, \lambda) \vdash \mathsf{read}\ s_{j_1}, \ldots, s_{j_m}\ \mathsf{as} \\
& && \qquad x_1, \ldots, x_m; Q'\ (= Q) \quad \square
\end{aligned}
$$

B.2 Proof of Lemma 1: Typability of A

Lemma 1 (Typability of A)**.**

$$(\mathcal{E}_0, \mathcal{E}_0(\mathcal{S}_0), [], \emptyset) \vdash A$$

Proof. Let B be a subprocess of A, $\mathcal{E}$ an environment (from names and variables to patterns), $\mathcal{S}$ a state (from cell names to patterns), ι a sequence of patterns, and λ a set of cell indices. We first prove by induction on the depth d of B that, if

(i) $\mathcal{E}_0 \subseteq \mathcal{E}$; and

(ii) $\mathcal{E}(\mathcal{S}_0) \leq \mathcal{S}$; and

(iii) $(\mathrm{bn}(B) \cup \mathrm{bv}(B)) \cap \mathrm{dom}(\mathcal{E}) = \emptyset$; and

(iv) $\forall a \in \mathrm{fn}(B),\ \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0$; and

(v) $\forall x \in \mathrm{fv}(B),\ \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0$; and

(vi) for all $i \in \{1, \ldots, n\}$, $i \in \lambda$ if and only if B is in the scope of a $\mathsf{lock}\ \ldots s_i \ldots$ in A,

then

$$(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash B$$

Base case ($d = 0$)**.** In that case $B = 0$ and according to our typing system

$$\frac{}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash 0\ (= B)}\ \tau_{nil}$$

Inductive case ($d > 0$)**.** We proceed by case analysis on the structure of B.

Case $B = B_1 \mid B_2$**.** First note that no parallel composition can occur in the scope of a lock, so $\lambda = \emptyset$.

(i) By hypothesis, $\mathcal{E}_0 \subseteq \mathcal{E}$.

(ii) By hypothesis, $\mathcal{E}(\mathcal{S}_0) \leq \mathcal{S}$.

(iii) $\overset{\text{Def.}}{\Rightarrow}$ $\text{bn}(B_1) \cup \text{bv}(B_1) \subseteq \text{bn}(B) \cup \text{bv}(B)$
$\text{bn}(B_2) \cup \text{bv}(B_2) \subseteq \text{bn}(B) \cup \text{bv}(B)$
$\overset{\text{Hyp.}}{\Rightarrow}$ $(\text{bn}(B_1) \cup \text{bv}(B_1)) \cap \text{dom}(\mathcal{E}) \subseteq (\text{bn}(B) \cup \text{bv}(B)) \cap \text{dom}(\mathcal{E}) = \emptyset$
$(\text{bn}(B_2) \cup \text{bv}(B_2)) \cap \text{dom}(\mathcal{E}) \subseteq (\text{bn}(B) \cup \text{bv}(B)) \cap \text{dom}(\mathcal{E}) = \emptyset$

(iv) $\overset{\text{Def.}}{\Rightarrow}$ $\text{fn}(B_1) \subseteq \text{fn}(B)$ $\overset{\text{Hyp.}}{\Rightarrow}$ $\forall a \in \text{fn}(B_1).\ \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0$
$\overset{\text{Def.}}{\Rightarrow}$ $\text{fn}(B_2) \subseteq \text{fn}(B)$ $\overset{\text{Hyp.}}{\Rightarrow}$ $\forall a \in \text{fn}(B_2).\ \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0$

(v) $\overset{\text{Def.}}{\Rightarrow}$ $\text{fv}(B_1) \subseteq \text{fv}(B)$ $\overset{\text{Hyp.}}{\Rightarrow}$ $\forall x \in \text{fv}(B_1).\ \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0$
$\overset{\text{Def.}}{\Rightarrow}$ $\text{fv}(B_2) \subseteq \text{fv}(B)$ $\overset{\text{Hyp.}}{\Rightarrow}$ $\forall x \in \text{fv}(B_2).\ \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0$

(vi) B is in the scope of a $\mathsf{lock}\ \ldots s_i \ldots$ if and only if B_1 and B_2 are too, so B_1, λ and B_2, λ satisfy condition (vi) by hypothesis.

Thus $(B1, \mathcal{E}, \mathcal{S}, \iota, \lambda)$ and $(B2, \mathcal{E}, \mathcal{S}, \iota, \lambda)$ satisfy conditions (i)- (vi), so we can apply our inductive hypothesis

$$(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash B_1 \qquad\qquad (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash B_2$$

and, according to our typing system

$$\frac{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash B_1 \qquad (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash B_2}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash B_1 \mid B_2\ (= B)}\ \tau_{par}$$

Case $B =!B'$**.** First note that no replication can occur in the scope of a lock, so $\lambda = \emptyset$

(i) By hypothesis, $\mathcal{E}_0 \subseteq \mathcal{E}$.

(ii) By hypothesis, $\mathcal{E}(\mathcal{S}_0) \leq \mathcal{S}$.

(iii) $\overset{\text{Def.}}{\Rightarrow}$ $\text{bn}(B') \cup \text{bv}(B') = \text{bn}(B) \cup \text{bv}(B)$
$\overset{\text{Hyp.}}{\Rightarrow}$ $(\text{bn}(B') \cup \text{bv}(B')) \cap \text{dom}(\mathcal{E}) = (\text{bn}(B) \cup \text{bv}(B)) \cap \text{dom}(\mathcal{E}) = \emptyset$

(iv) $\overset{\text{Def.}}{\Rightarrow}$ $\text{fn}(B') = \text{fn}(B)$ $\overset{\text{Hyp.}}{\Rightarrow}$ $\forall a \in \text{fn}(B').\ \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0$

(v) $\overset{\text{Def.}}{\Rightarrow}$ $\text{fv}(B') = \text{fv}(B)$ $\overset{\text{Hyp.}}{\Rightarrow}$ $\forall x \in \text{fv}(B').\ \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0$

(vi) B is in the scope of a $\mathsf{lock}\ \ldots s_i \ldots$ if and only if B' is too, so B', λ satisfy condition (vi) by hypothesis.

Thus $(B', \mathcal{E}, \mathcal{S}, \iota, \lambda)$ satisfy conditions (i)- (vi), so we can apply our inductive hypothesis

$$(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash B'$$

and, according to our typing system

$$\frac{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash B'}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash !B'\ (= B)}\ \tau_{repl}$$

Case $B = \text{if } M = N \text{ then } B_1 \text{ else } B_2$ **with** $\mathcal{E}(M) = \mathcal{E}(N)$**.**

(i) By hypothesis, $\mathcal{E}_0 \subseteq \mathcal{E}$.

(ii) By hypothesis, $\mathcal{E}(\mathcal{S}_0) \leq \mathcal{S}$.

(iii)
$$\begin{array}{ll} \overset{\text{Def.}}{\Rightarrow} & \mathrm{bn}(B_1) \cup \mathrm{bv}(B_1) \subseteq \mathrm{bn}(B) \cup \mathrm{bv}(B) \\ & \mathrm{bn}(B_2) \cup \mathrm{bv}(B_2) \subseteq \mathrm{bn}(B) \cup \mathrm{bv}(B) \\ \overset{\text{Hyp.}}{\Rightarrow} & (\mathrm{bn}(B_1) \cup \mathrm{bv}(B_1)) \cap \mathrm{dom}(\mathcal{E}) \subseteq (\mathrm{bn}(B) \cup \mathrm{bv}(B)) \cap \mathrm{dom}(\mathcal{E}) = \emptyset \\ & (\mathrm{bn}(B_2) \cup \mathrm{bv}(B_2)) \cap \mathrm{dom}(\mathcal{E}) \subseteq (\mathrm{bn}(B) \cup \mathrm{bv}(B)) \cap \mathrm{dom}(\mathcal{E}) = \emptyset \end{array}$$

(iv) $\overset{\text{Def.}}{\Rightarrow} \mathrm{fn}(B_1) \subseteq \mathrm{fn}(B) \overset{\text{Hyp.}}{\Rightarrow} \forall a \in \mathrm{fn}(B_1).\ \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0$
$\overset{\text{Def.}}{\Rightarrow} \mathrm{fn}(B_2) \subseteq \mathrm{fn}(B) \overset{\text{Hyp.}}{\Rightarrow} \forall a \in \mathrm{fn}(B_2).\ \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0$

(v) $\overset{\text{Def.}}{\Rightarrow} \mathrm{fv}(B_1) \subseteq \mathrm{fv}(B) \overset{\text{Hyp.}}{\Rightarrow} \forall x \in \mathrm{fv}(B_1).\ \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0$
$\overset{\text{Def.}}{\Rightarrow} \mathrm{fv}(B_2) \subseteq \mathrm{fv}(B) \overset{\text{Hyp.}}{\Rightarrow} \forall x \in \mathrm{fv}(B_2).\ \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0$

(vi) B is in the scope of a $\mathsf{lock}\ \dots s_i \dots$ if and only if B_1 and B_2 are too, so B_1, λ and B_2, λ satisfy condition (vi) by hypothesis.

Thus $(B_1, \mathcal{E}, \mathcal{S}, \iota, \lambda)$ and $(B_2, \mathcal{E}, \mathcal{S}, \iota, \lambda)$ satisfy conditions (i)- (vi), so we can apply our inductive hypothesis

$$(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash B_1 \qquad (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash B_2$$

and, according to our typing system

$$\frac{\mathcal{E}(M) = \mathcal{E}(N) \qquad (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash B_1 \qquad (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash B_2}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \text{if } M = N \text{ then } B_1 \text{ else } B_2\ (= B)}\ \tau_{if}$$

Case $B = \text{if } M = N \text{ then } B_1 \text{ else } B_2$ **with** $\mathcal{E}(M) \neq \mathcal{E}(N)$**.**

(i) By hypothesis, $\mathcal{E}_0 \subseteq \mathcal{E}$.

(ii) By hypothesis, $\mathcal{E}(\mathcal{S}_0) \leq \mathcal{S}$.

(iii)
$$\begin{array}{ll} \overset{\text{Def.}}{\Rightarrow} & \mathrm{bn}(B_2) \cup \mathrm{bv}(B_2) \subseteq \mathrm{bn}(B) \cup \mathrm{bv}(B) \\ \overset{\text{Hyp.}}{\Rightarrow} & (\mathrm{bn}(B_2) \cup \mathrm{bv}(B_2)) \cap \mathrm{dom}(\mathcal{E}) \subseteq (\mathrm{bn}(B) \cup \mathrm{bv}(B)) \cap \mathrm{dom}(\mathcal{E}) = \emptyset \end{array}$$

(iv) $\overset{\text{Def.}}{\Rightarrow} \mathrm{fn}(B_2) \subseteq \mathrm{fn}(B) \overset{\text{Hyp.}}{\Rightarrow} \forall a \in \mathrm{fn}(B_2).\ \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0$

(v) $\overset{\text{Def.}}{\Rightarrow} \mathrm{fv}(B_2) \subseteq \mathrm{fv}(B) \overset{\text{Hyp.}}{\Rightarrow} \forall x \in \mathrm{fv}(B_2).\ \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0$

(vi) B is in the scope of a $\mathsf{lock}\ \dots s_i \dots$ if and only if B_2 is too, so B_2, λ satisfy condition (vi) by hypothesis.

Thus $(B_2, \mathcal{E}, \mathcal{S}, \iota, \lambda)$ satisfy conditions (i)- (vi), so we can apply our inductive hypothesis

$$(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash B_2$$

and, according to our typing system

$$\frac{\mathcal{E}(M) \neq \mathcal{E}(N) \qquad (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash B_2}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \text{if } M = N \text{ then } B_1 \text{ else } B_2\ (= B)}\ \tau_{if}$$

Case $B = \mathsf{new}\ a; B'$**.** By hypothesis on A, $\mathrm{bn}(A) \cap \mathrm{bn}(P_0') = \emptyset$, thus $a \notin \mathrm{bn}(P_0')$. Let $\mathcal{E}' = \mathcal{E} \cup \{a \mapsto attn[]\}$.

(i) By hypothesis, $\mathcal{E}_0 \subseteq \mathcal{E} \subseteq \mathcal{E} \cup \{a \mapsto attn[]\} = \mathcal{E}'$. Moreover, by hypothesis $\mathrm{bn}(B) \cap \mathrm{dom}(\mathcal{E}) = \emptyset$, thus $a \notin \mathrm{dom}(\mathcal{E})$, thus, $\mathcal{E}'$ is an environment, *i.e.* a function from names and variables to patterns.

(ii) By hypothesis, $\mathcal{E}(\mathcal{S}_0) \leq \mathcal{S}$.

(iii) $$\begin{aligned} &\overset{\text{Def.}}{\Rightarrow} && \mathrm{bn}(B') \cup \mathrm{bv}(B') \subseteq \mathrm{bn}(B) \cup \mathrm{bv}(B) \\ &\overset{\text{Hyp.}}{\Rightarrow} && (\mathrm{bn}(B') \cup \mathrm{bv}(B')) \cap \mathrm{dom}(\mathcal{E}) \subseteq (\mathrm{bn}(B) \cup \mathrm{bv}(B)) \cap \mathrm{dom}(\mathcal{E}) = \emptyset \end{aligned}$$

(iv) Let $b \in \mathrm{fn}(B')$. Then either $b \neq a$ or $b = a$.

Case $b \neq a$**.** $$\begin{aligned} &\overset{\text{Def.}}{\Rightarrow} && b \in \mathrm{fn}(B) \\ &\overset{\text{Hyp.}}{\Rightarrow} && \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(b)) \in \mathcal{F}_0 \\ &\overset{\mathcal{E}'(b)=\mathcal{E}(b)}{\Rightarrow} && \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}'(b)) \in \mathcal{F}_0 \end{aligned}$$

Case $b = a$**.** $$\begin{aligned} &\overset{\text{Def.}}{\Rightarrow} && \mathsf{attacker}(\overline{\mathcal{E}(\mathcal{S}_0)}, attn[]) \in \mathcal{C}_0 \\ &\overset{\text{Def.}}{\Rightarrow} && \mathsf{attacker}(\overline{\mathcal{E}(\mathcal{S}_0)}, attn[]) \in \mathcal{F}_0 \\ &\overset{\mathcal{E}(\mathcal{S}_0)\leq\mathcal{S}}{\Rightarrow} && \mathsf{attacker}(\overline{\mathcal{S}}, attn[]) \in \mathcal{F}_0 \\ &\overset{\mathcal{E}'(b)=attn[]}{\Rightarrow} && \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}'(b)) \in \mathcal{F}_0 \end{aligned}$$

(v) Let $x \in \mathrm{fv}(B')$. $$\begin{aligned} &\overset{\text{Def.}}{\Rightarrow} && x \in \mathrm{fv}(B) \\ &\overset{\text{Hyp.}}{\Rightarrow} && \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0 \\ &\overset{\mathcal{E}'(x)=\mathcal{E}(x)}{\Rightarrow} && \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}'(x)) \in \mathcal{F}_0 \end{aligned}$$

(vi) B is in the scope of a $\mathsf{lock}\ \dots s_i \dots$ if and only if B' is too, so B', λ satisfy condition (vi) by hypothesis.

Thus $(B', \mathcal{E}, \mathcal{S}, \iota, \lambda)$ satisfy conditions (i)- (vi), so we can apply our inductive hypothesis

$$(\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash B'$$

and according to our typing system

$$\frac{a \notin \mathrm{bn}(P_0') \ \Rightarrow\ (\mathcal{E} \cup \{a \mapsto attn[]\}, \mathcal{S}, \iota, \lambda) \vdash B'}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \mathsf{new}\ a; B'}\ \tau_{newA}$$

Case $B = \text{let } x = g(M_1, \dots, M_k) \text{ in } B_1 \text{ else } B_2$**.** Let M be a term such that $g(\mathcal{E}(M_1), \dots \mathcal{E}(M_k)) \to M$. Let $\mathcal{E}' = \mathcal{E} \cup \{x \mapsto M\}$.

(i) By hypothesis, $\mathcal{E}_0 \subseteq \mathcal{E}$.

By hypothesis, $\mathcal{E}_0 \subseteq \mathcal{E} \subseteq \mathcal{E} \cup \{x \mapsto M\} = \mathcal{E}'$.

(ii) By hypothesis, $\mathcal{E}(\mathcal{S}_0) \leq \mathcal{S}$.

(iii) $\overset{\text{Def.}}{\Rightarrow}$ $\mathrm{bn}(B_1) \cup \mathrm{bv}(B_1) \subset \mathrm{bn}(B) \cup \mathrm{bv}(B)$
$\mathrm{bn}(B_2) \cup \mathrm{bv}(B_2) \subseteq \mathrm{bn}(B) \cup \mathrm{bv}(B)$
$\overset{\text{Hyp.}}{\Rightarrow}$ $(\mathrm{bn}(B_1) \cup \mathrm{bv}(B_1)) \cap \mathrm{dom}(\mathcal{E}) \subset (\mathrm{bn}(B) \cup \mathrm{bv}(B)) \cap \mathrm{dom}(\mathcal{E}) = \emptyset$
$(\mathrm{bn}(B_2) \cup \mathrm{bv}(B_2)) \cap \mathrm{dom}(\mathcal{E}) \subseteq (\mathrm{bn}(B) \cup \mathrm{bv}(B)) \cap \mathrm{dom}(\mathcal{E}) = \emptyset$

(iv) $\overset{\text{Def.}}{\Rightarrow}$ $\mathrm{fn}(B_1) \subseteq \mathrm{fn}(B)$ $\overset{\text{Hyp.}}{\Rightarrow}$ $\forall a \in \mathrm{fn}(B_1).\ \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0$
$\overset{\text{Def.}}{\Rightarrow}$ $\mathrm{fn}(B_2) \subseteq \mathrm{fn}(B)$ $\overset{\text{Hyp.}}{\Rightarrow}$ $\forall a \in \mathrm{fn}(B_2).\ \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0$

(v) $\overset{\text{Def.}}{\Rightarrow}$ $\mathrm{fv}(B_1) \subseteq \mathrm{fv}(B) \cup \{x\}$. Let $y \in \mathrm{fv}(B_1)$.

Case $y \in \mathrm{fv}(B)$**.** $\overset{\text{Hyp.}}{\Rightarrow}$ $\mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(y)) \in \mathcal{F}_0$
$\overset{\mathcal{E}(y)=\mathcal{E}'(y)}{\Rightarrow}$ $\mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}'(y)) \in \mathcal{F}_0$
$\overset{\mathcal{S}\leq\mathcal{T}}{\Rightarrow}$ $\mathsf{attacker}(\overline{\mathcal{T}}, \mathcal{E}'(y)) \in \mathcal{F}_0$

Case $y = x$**.** In that case, by construction $\mathcal{C} = \mathsf{att}(xs, N_1) \wedge \ldots \mathsf{attacker}(xs, N_k) \Rightarrow \mathsf{attacker}(xs, N) \in \mathcal{C}_0$ for some $N_1, \ldots, N_k$ and $\mathcal{E}(M_1) = N_1\sigma$, …, $\mathcal{E}(M_k) = N_k\sigma$, $M = N\sigma$ for some σ. Now,

$\overset{\text{Hyp.}}{\Rightarrow}$ $\forall i \in \{1, \ldots, k\} \forall u \in \mathrm{fn}(M_) \cup \mathrm{fv}(M_i)\ \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(u))$
$\overset{\text{Lem. 4}}{\Rightarrow}$ $\forall i \in \{1, \ldots, k\}\ \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(M_i)) \in \mathcal{F}_0$
$\overset{\mathcal{S}\leq\mathcal{T}}{\Rightarrow}$ $\forall i \in \{1, \ldots, k\}\ \mathsf{attacker}(\overline{\mathcal{T}}, \mathcal{E}(M_i)) \in \mathcal{F}_0$
$\overset{\sigma\ \wedge\ \mathcal{C}\in\mathcal{C}_0}{\Rightarrow}$ $\mathsf{attacker}(\overline{\mathcal{T}}, M) \in \mathcal{F}_0$
$\overset{\mathcal{E}'(y)=M}{\Rightarrow}$ $\mathsf{attacker}(\overline{\mathcal{T}}, \mathcal{E}'(y)) \in \mathcal{F}_0$

$\overset{\text{Def.}}{\Rightarrow}$ $\mathrm{fv}(B_2) \subseteq \mathrm{fv}(B)$ $\overset{\text{Hyp.}}{\Rightarrow}$ $\forall x \in \mathrm{fv}(B_2).\ \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0$

(vi) B is in the scope of a $\mathsf{lock}\ \ldots s_i \ldots$ if and only if B_1 and B_2 are too, so B_1, λ and B_2, λ satisfy condition (vi) by hypothesis.

Thus $(B_1, \mathcal{E}', \mathcal{S}, \iota, \lambda)$ and $(B_2, \mathcal{E}, \mathcal{S}, \iota, \lambda)$ satisfy conditions (i)- (vi), so we can apply our inductive hypothesis

$$(\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash B_1 \qquad (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash B_2$$

and, according to our typing system

$$\frac{\forall M\ (g(\mathcal{E}(M_1), \ldots, \mathcal{E}(M_k)) \to M)\ \Rightarrow ((\mathcal{E} \cup \{x \mapsto M\}, \mathcal{S}, \iota, \lambda) \vdash B_1 \wedge (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash B_2)}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \text{let } x = g(M_1, \ldots, M_k) \text{ in } B_1 \text{ else } B_2\ (= B)}\ \tau_{let}$$

Case $B = \mathsf{out}(M, N); B'$**.**

(i) By hypothesis, $\mathcal{E}_0 \subseteq \mathcal{E}$.

(ii) By hypothesis, $\mathcal{E}(\mathcal{S}_0) \leq \mathcal{S}$.

(iii) $\overset{\text{Def.}}{\Rightarrow}$ $\mathrm{bn}(B') \cup \mathrm{bv}(B') = \mathrm{bn}(B) \cup \mathrm{bv}(B)$
$\overset{\text{Hyp.}}{\Rightarrow}$ $(\mathrm{bn}(B') \cup \mathrm{bv}(B')) \cap \mathrm{dom}(\mathcal{E}) = (\mathrm{bn}(B) \cup \mathrm{bv}(B)) \cap \mathrm{dom}(\mathcal{E}) = \emptyset$

(iv) $\overset{\text{Def.}}{\Rightarrow}$ $\mathrm{fn}(B') \subseteq \mathrm{fn}(B)$ $\overset{\text{Hyp.}}{\Rightarrow}$ $\forall a \in \mathrm{fn}(B').\ \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0$

(v) $\overset{\text{Def.}}{\Rightarrow}$ $\text{fv}(B') \subseteq \text{fv}(B)$ $\overset{\text{Hyp.}}{\Rightarrow}$ $\forall x \in \text{fv}(B').\ \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0$

(vi) B is in the scope of a $\mathsf{lock}\ \ldots s_i \ldots$ if and only if B' is too, so B', λ satisfy condition (vi) by hypothesis.

Thus $(B', \mathcal{E}, \mathcal{S}, \iota, \lambda)$ satisfy conditions (i)- (vi), so we can apply our inductive hypothesis

$$(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash B'$$

Moreover, by hypothesis we have that

- for all $a \in \text{fn}(M) \cup \text{fn}(N)$, $\mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0$, because $\text{fn}(M) \cup \text{fn}(N) \subseteq \text{fn}(B)$; and
- for all $x \in \text{fv}(M) \cup \text{fv}(N)$, $\mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0$, because $\text{fv}(M) \cup \text{fv}(N) \subseteq \text{fv}(B)$.

Thus according to lemma 4 it is the case that

$$\mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(M)) \in \mathcal{F}_0 \qquad \text{and} \qquad \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(N)) \in \mathcal{F}_0$$

And because

$$\mathsf{attacker}(xs, xc) \wedge \mathsf{attacker}(xs, xm) \Rightarrow \mathsf{message}(xs, xc, xm) \in \mathcal{C}_0$$

we have by resolution that

$$\mathsf{message}(\overline{\mathcal{S}}, \mathcal{E}(M), \mathcal{E}(N)) \in \mathcal{F}_0.$$

Thus, according to our typing system

$$\frac{\mathsf{message}(\overline{\mathcal{S}}, \mathcal{E}(M), \mathcal{E}(N)) \in \mathcal{F}_0 \qquad (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash B'}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \mathsf{out}(M, N); B'\ (= B)}\ \mathcal{T}_{out}$$

Case $B = \mathsf{in}(M, x); B'$. Let $\mathcal{T}$ be a state such that $\mathcal{S} \leq \mathcal{T}$ and $\mathcal{T} = \mathcal{T}[j \mapsto \mathcal{S}(j) \mid j \in \lambda]$, N a term such that $\mathsf{message}(\overline{\mathcal{T}}, \mathcal{E}(M), N) \in \mathcal{F}_0$, $\mathcal{E}' = \mathcal{E} \cup \{x \mapsto N\}$, and $\iota' = N :: \iota$.

(i) By hypothesis, $\mathcal{E}_0 \subseteq \mathcal{E} \subseteq \mathcal{E} \cup \{x \mapsto N\} = \mathcal{E}'$. Moreover, since $\text{bv}(B) \cap \text{dom}(\mathcal{E}) = \emptyset$, $x \notin \text{dom}(\mathcal{E})$. Thus $\mathcal{E}'$ is indeed an environment, *i.e.* a function from variables and names to patterns.

(ii)
$$\begin{aligned} &\overset{\text{Hyp.}}{\Rightarrow} && \mathcal{E}(\mathcal{S}_0) \leq \mathcal{S} \\ &\overset{\text{Transitivity of} \leq}{\Rightarrow} && \mathcal{E}(\mathcal{S}_0) \leq \mathcal{T} \end{aligned}$$

(iii)
$$\begin{aligned} &\overset{\text{Def.}}{\Rightarrow} && \text{bn}(B') \cup \text{bv}(B') \subset \text{bn}(B) \cup \text{bv}(B) \\ &\overset{\text{Hyp.}}{\Rightarrow} && (\text{bn}(B') \cup \text{bv}(B')) \cap \text{dom}(\mathcal{E}) \subset (\text{bn}(B) \cup \text{bv}(B)) \cap \text{dom}(\mathcal{E}) = \emptyset \end{aligned}$$

(iv) Let $a \in \text{fn}(B')$.
$$\begin{aligned} &\overset{\text{Def.}}{\Rightarrow} && a \in \text{fn}(B) \\ &\overset{\text{Hyp.}}{\Rightarrow} && \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0 \\ &\overset{\mathcal{E}(a) = \mathcal{E}'(a)}{\Rightarrow} && \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}'(a)) \in \mathcal{F}_0 \\ &\overset{\mathcal{S} \leq \mathcal{T}}{\Rightarrow} && \mathsf{attacker}(\overline{\mathcal{T}}, \mathcal{E}'(a)) \in \mathcal{F}_0 \end{aligned}$$

(v) Let $y \in \mathrm{fv}(B')$. Then either $y \in \mathrm{fv}(B)$ or $y = x$.

Case $y \in \mathrm{fv}(B)$**.**

$$\begin{aligned}
&\overset{\text{Hyp.}}{\Rightarrow} && \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(y)) \in \mathcal{F}_0\\
&\overset{\mathcal{E}(y)=\mathcal{E}'(y)}{\Rightarrow} && \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}'(y)) \in \mathcal{F}_0\\
&\overset{\mathcal{S}\leq\mathcal{T}}{\Rightarrow} && \mathsf{attacker}(\overline{\mathcal{T}}, \mathcal{E}'(y)) \in \mathcal{F}_0
\end{aligned}$$

Case $y = x$**.**

$$\begin{aligned}
&\overset{\text{Hyp.}}{\Rightarrow} && \forall u \in \mathrm{fn}(M) \cup \mathrm{fv}(M)\ \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(u))\\
&\overset{\text{Lem. 4}}{\Rightarrow} && \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(M)) \in \mathcal{F}_0\\
&\overset{\mathcal{S}\leq\mathcal{T}}{\Rightarrow} && \mathsf{attacker}(\overline{\mathcal{T}}, \mathcal{E}(M)) \in \mathcal{F}_0\\
&\overset{\substack{\mathsf{message}(xs,xc,xm)\wedge\\ \mathsf{attacker}(xs,xc)\Rightarrow\mathsf{attacker}(xs,xm)\in\mathcal{C}_0}}{\Rightarrow} && \mathsf{attacker}(\overline{\mathcal{T}}, N) \in \mathcal{F}_0\\
&\overset{\mathcal{E}'(y)=N}{\Rightarrow} && \mathsf{attacker}(\overline{\mathcal{T}}, \mathcal{E}'(y)) \in \mathcal{F}_0
\end{aligned}$$

(vi) B is in the scope of a $\mathsf{lock}\ \ldots s_i \ldots$ if and only if B' is too, so B', λ satisfy condition (vi) by hypothesis.

Thus $(B', \mathcal{E}', \mathcal{S}, \iota', \lambda)$ satisfy conditions (i)- (vi), so we can apply our inductive hypothesis

$$(\mathcal{E}', \mathcal{T}, \iota', \lambda) \vdash B'$$

Finally, according to our typing system

$$\frac{\begin{array}{l}\forall \mathcal{T}\ \forall N\ (\mathcal{S} \leq \mathcal{T}\ \wedge\ \mathcal{T} = \mathcal{T}[j \mapsto \mathcal{S}(j) \mid j \in \lambda]\ \wedge\\ \qquad \mathsf{message}(\overline{\mathcal{T}}, \mathcal{E}(M), N) \in \mathcal{F}_0)\ \Rightarrow\ (\mathcal{E} \cup \{x \mapsto N\}, \mathcal{T}, (N :: \iota), \lambda) \vdash B'\end{array}}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \mathsf{in}(M, x); B'\ (= B)}\ \tau_{in}$$

Case $B = [s \mapsto M]$**.** This case cannot occur because by hypothesis no $[s \mapsto M]$ occurs in A.

Case $B = \mathsf{lock}\ s_{j_1}, \ldots, s_{j_m}; B'$**.** Let $\mathcal{T}$ be a state such that $\mathcal{S} \leq \mathcal{T}$ and $\mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda]$. Let $\lambda' = \lambda \cup \{j_1, \ldots, j_m\}$.

(i) By hypothesis, $\mathcal{E}_0 \subseteq \mathcal{E}$.

(ii) By hypothesis, $\mathcal{E}(\mathcal{S}_0) \leq \mathcal{S}$.

(iii)

$$\begin{aligned}
&\overset{\text{Def.}}{\Rightarrow} && \mathrm{bn}(B') \cup \mathrm{bv}(B') = \mathrm{bn}(B) \cup \mathrm{bv}(B)\\
&\overset{\text{Hyp.}}{\Rightarrow} && (\mathrm{bn}(B') \cup \mathrm{bv}(B')) \cap \mathrm{dom}(\mathcal{E}) = (\mathrm{bn}(B) \cup \mathrm{bv}(B)) \cap \mathrm{dom}(\mathcal{E}) = \emptyset
\end{aligned}$$

(iv) Let $a \in \mathrm{fn}(B')$.

$$\begin{aligned}
&\overset{\text{Def.}}{\Rightarrow} && a \in \mathrm{fn}(B)\\
&\overset{\text{Hyp.}}{\Rightarrow} && \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0\\
&\overset{\mathcal{S}\leq\mathcal{T}}{\Rightarrow} && \mathsf{attacker}(\overline{\mathcal{T}}, \mathcal{E}(a)) \in \mathcal{F}_0
\end{aligned}$$

(v) Let $x \in \mathrm{fv}(B')$.

$$\begin{aligned}
&\overset{\text{Def.}}{\Rightarrow} && x \in \mathrm{fv}(B)\\
&\overset{\text{Hyp.}}{\Rightarrow} && \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0\\
&\overset{\mathcal{S}\leq\mathcal{T}}{\Rightarrow} && \mathsf{attacker}(\overline{\mathcal{T}}, \mathcal{E}(x)) \in \mathcal{F}_0
\end{aligned}$$

(vi) Because if B is in the scope of a $\mathsf{lock}\ \ldots s_i \ldots$ so is B', and because if B' is in the scope of a $\mathsf{lock}\ \ldots s_i \ldots$ with $i \notin \{j_1, \ldots, j_m\}$ so is B', B', λ' satisfy condition (vi) by hypothesis.

Thus, B', $\mathcal{E}$, $\mathcal{T}$, ι, and λ' thus satisfy conditions (i)-(vi), so we can apply our inductive hypothesis

$$(\mathcal{E}, \mathcal{T}, \iota, \lambda') \vdash B'.$$

Finally, according to our typing system

$$\frac{\forall \mathcal{T}\ (\mathcal{S} \leq \mathcal{T}\ \wedge\ \mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda]\ \Rightarrow\ (\mathcal{E}, \mathcal{T}, \iota, \lambda \cup \{j_1, \dots, j_m\}) \vdash B')}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \mathsf{lock}\ s_{j_1}, \dots, s_{j_m}; B'\ (= B)}\ \tau_{lock}$$

Case $B = \mathsf{unlock}\ s_{j_1}, \dots, s_{j_m}; B'$**.** Let $\mathcal{T}$ be a state such that $\mathcal{S} \leq \mathcal{T}$ and $\mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda]$. Let $\lambda' = \lambda \smallsetminus \{j_1, \dots, j_m\}$.

(i) By hypothesis, $\mathcal{E}_0 \subseteq \mathcal{E}$.

(ii) By hypothesis, $\mathcal{E}(\mathcal{S}_0) \leq \mathcal{S}$.

(iii)
$$\begin{array}{ll} \overset{\text{Def.}}{\Rightarrow} & \mathrm{bn}(B') \cup \mathrm{bv}(B') = \mathrm{bn}(B) \cup \mathrm{bv}(B) \\ \overset{\text{Hyp.}}{\Rightarrow} & (\mathrm{bn}(B') \cup \mathrm{bv}(B')) \cap \mathrm{dom}(\mathcal{E}) = (\mathrm{bn}(B) \cup \mathrm{bv}(B)) \cap \mathrm{dom}(\mathcal{E}) = \emptyset \end{array}$$

(iv) Let $a \in \mathrm{fn}(B')$.
$$\begin{array}{ll} \overset{\text{Def.}}{\Rightarrow} & a \in \mathrm{fn}(B) \\ \overset{\text{Hyp.}}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0 \\ \overset{\mathcal{S} \leq \mathcal{T}}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{T}}, \mathcal{E}(a)) \in \mathcal{F}_0 \end{array}$$

(v) Let $x \in \mathrm{fv}(B')$.
$$\begin{array}{ll} \overset{\text{Def.}}{\Rightarrow} & x \in \mathrm{fv}(B) \\ \overset{\text{Hyp.}}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0 \\ \overset{\mathcal{S} \leq \mathcal{T}}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{T}}, \mathcal{E}(x)) \in \mathcal{F}_0 \end{array}$$

(vi) Because if B' is in the scope of a $\mathsf{lock}\ \dots s_i \dots$ so is B, and because if B is in the scope of a $\mathsf{lock}\ \dots s_i \dots$ with $i \notin \{j_1, \dots, j_m\}$ so is B', B', λ' satisfy condition (vi) by hypothesis.

Thus, B', $\mathcal{E}$, $\mathcal{T}$, ι, and λ' satisfy conditions (i)-(vi), so we can apply our inductive hypothesis

$$(\mathcal{E}, \mathcal{T}, \iota, \lambda') \vdash B'.$$

Finally, according to our typing system

$$\frac{\forall \mathcal{T}\ (\mathcal{S} \leq \mathcal{T}\ \wedge\ \mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda]\ \Rightarrow\ (\mathcal{E}, \mathcal{T}, \iota, \lambda \smallsetminus \{j_1, \dots, j_m\}) \vdash B')}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \mathsf{unlock}\ s_{j_1}, \dots, s_{j_m}; B'\ (= B)}\ \tau_{unlock}$$

Case $B = \mathsf{read}\ s_{j_1}, \dots, s_{j_m}\ \mathsf{as}\ x_1, \dots, x_k; B'$**.** Let $\mathcal{T}$ such that $\mathcal{S} \leq \mathcal{T}$ and $\mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda]$, $\mathcal{E}' = \mathcal{E} \cup \{x_k \mapsto \mathcal{T}(j_k) \mid 1 \leq k \leq m\}$. Let $\iota' = \mathcal{T}(j_1) :: \cdots :: \mathcal{T}(j_m) :: \iota$.

(i) By hypothesis $\mathcal{E}_0 \subseteq \mathcal{E} \subseteq \mathcal{E} \cup \{x_k \mapsto \mathcal{T}(s_{j_k}) \mid 1 \leq k \leq m\} = \mathcal{E}'$. Moreover, since $\mathrm{bv}(B) \cap \mathrm{dom}(\mathcal{E}) = \emptyset$, $x \notin \mathrm{dom}(\mathcal{E})$. Thus $\mathcal{E}'$ is indeed an environment, *i.e.* a function from variables and names to patterns.

(ii)
$$\begin{array}{ll} \overset{\text{Hyp.}}{\Rightarrow} & \mathcal{E}(\mathcal{S}_0) \leq \mathcal{S} \\ \overset{\text{Transitivity of } \leq}{\Rightarrow} & \mathcal{E}(\mathcal{S}_0) \leq \mathcal{T} \end{array}$$

(iii)
$$\begin{array}{ll} \overset{\text{Def.}}{\Rightarrow} & \mathrm{bn}(B') \cup \mathrm{bv}(B') \subset \mathrm{bn}(B) \cup \mathrm{bv}(B) \\ \overset{\text{Hyp.}}{\Rightarrow} & (\mathrm{bn}(B') \cup \mathrm{bv}(B')) \cap \mathrm{dom}(\mathcal{E}) \subset (\mathrm{bn}(B) \cup \mathrm{bv}(B)) \cap \mathrm{dom}(\mathcal{E}) = \emptyset \end{array}$$

(iv) Let $a \in \mathrm{fn}(B')$.
$$\begin{array}{ll} \overset{\text{Def.}}{\Rightarrow} & a \in \mathrm{fn}(B) \\ \overset{\text{Hyp.}}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0 \\ \overset{\mathcal{E}(a)=\mathcal{E}'(a)}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}'(a)) \in \mathcal{F}_0 \\ \overset{\mathcal{S}\leq\mathcal{T}}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{T}}, \mathcal{E}'(a)) \in \mathcal{F}_0 \end{array}$$

(v) Let $y \in \mathrm{fv}(B')$. Then either $y \in \mathrm{fv}(B)$ or $y \in \{x_1, \dots, x_m\}$.

Case $y \in \mathrm{fv}(B)$**.**
$$\begin{array}{ll} \overset{\text{Hyp.}}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(y)) \in \mathcal{F}_0 \\ \overset{\mathcal{E}(y)=\mathcal{E}'(y)}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}'(y)) \in \mathcal{F}_0 \\ \overset{\mathcal{S}\leq\mathcal{T}}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{T}}, \mathcal{E}'(y)) \in \mathcal{F}_0 \end{array}$$

Case $y = x_k$ **for some** $k \in \{1, \dots, m\}$**.** By hypothesis $\mathit{cells}(A) \subseteq \mathrm{fn}(P)$. Thus by definition of $\mathcal{E}_0$, $\mathit{cells}(A) \subseteq \mathrm{dom}(\mathcal{E}_0)$ and by construction of $\mathcal{C}_0$, for all $i \in \{1 \dots, n\}$

$$\begin{array}{r} \mathcal{C}_i = \mathsf{message}((xs_1, \dots, xs_n), xc, xm) \wedge \\ \mathsf{attacker}((xs_1, \dots, xs_n), \mathcal{E}_0(s_i)) \Rightarrow \\ \mathsf{attacker}((xs_1, \dots, xs_n), xs_i) \in \mathcal{C}_0 \end{array}$$

$$\begin{array}{ll} \overset{\text{Def.}}{\Rightarrow} & s_{j_1}, \dots, s_{j_m} \in \mathrm{fn}(B) \\ \overset{\text{Hyp.}}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(s_{j_k})) \in \mathcal{F}_0 \\ \overset{\mathcal{E}_0(s_{j_k})=\mathcal{E}(s_{j_k})}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}_0(s_{j_k})) \in \mathcal{F}_0 \\ \overset{\mathcal{S}\leq\mathcal{T}}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{T}}, \mathcal{E}_0(s_{j_k})) \in \mathcal{F}_0 \\ \overset{\mathcal{C}_{j_k}\in\mathcal{C}_0}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{T}}, \overline{\mathcal{T}}_{j_k}) \in \mathcal{F}_0 \\ \overset{\overline{\mathcal{T}}_{j_k}=\mathcal{T}(j_k)}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{T}}, \mathcal{T}(j_k)) \in \mathcal{F}_0 \\ \overset{\mathcal{E}'(y)=\mathcal{T}(j_k)}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{T}}, \mathcal{E}'(y)) \in \mathcal{F}_0 \end{array}$$

(vi) B is in the scope of a $\mathsf{lock}\ \dots s_i \dots$ if and only if B' is too, so B', λ satisfy condition (vi) by hypothesis.

Thus $(B', \mathcal{E}', \mathcal{T}, \iota', \lambda)$ satisfy conditions (i)- (vi), so we can apply our inductive hypothesis

$$(\mathcal{E}', \mathcal{T}, \iota', \lambda) \vdash B'$$

and thus, according to our typing system

$$\dfrac{\begin{array}{l} \forall \mathcal{T}\ (\mathcal{S} \leq \mathcal{T}\ \wedge\ \mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda])\ \Rightarrow \\ \qquad (\mathcal{E} \cup \{x_{j_k} \mapsto \mathcal{T}(_{j_k}) \mid 1 \leq k \leq m\}, \mathcal{T}, (\mathcal{T}(j_1) :: \cdots :: \mathcal{T}(j_m) :: \iota), \lambda) \vdash B' \end{array}}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \mathsf{read}\ s_{j_1}, \dots, s_{j_m}\ \mathsf{as}\ x_1, \dots, x_m; B'\ (= B)}\ \tau_{inT}$$

Case $B = s_{j_1}, \dots, s_{j_m} := M_1, \dots, M_m; B'$**.** Let $\mathcal{T}$ such that $\mathcal{S} \leq \mathcal{T}$ and $\mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda]$. Let $\mathcal{T}' = \mathcal{T}[j_k \mapsto \mathcal{E}(M_k) \mid 1 \leq k \leq m]$. We first show that $\mathcal{T} \leq \mathcal{T}'$. By hypothesis $\mathit{cells}(A) \subseteq \mathrm{fn}(P)$. Thus by construction of $\mathcal{E}_0$, $\mathit{cells}(A) \subseteq \mathrm{dom}(\mathcal{E}_0)$, and by construction of $\mathcal{C}_0$ for all $i \in \{1, \dots, n\}$

$$\begin{aligned}
\mathcal{C}_{i_1} = {} & \mathsf{message}((xs_1, \dots, xs_n), xc, xm) \wedge \mathsf{attacker}((xs_1, \dots, xs_n), \mathcal{E}_0(s_i)) \wedge \\
& \mathsf{attacker}((xs_1, \dots, xs_n), ys_i) \Rightarrow \mathsf{message}((xs_1, \dots, ys_i, \dots, xs_n), xc, xm) \in \mathcal{C}_0 \\
\mathcal{C}_{i_2} = {} & \mathsf{attacker}((xs_1, \dots, xs_n), xm) \wedge \mathsf{attacker}((xs_1, \dots, xs_n), \mathcal{E}_0(s_i)) \wedge \\
& \mathsf{attacker}((xs_1, \dots, xs_n), ys_i) \Rightarrow \mathsf{attacker}((xs_1, \dots, ys_i, \dots, xs_n), xm) \in \mathcal{C}_0
\end{aligned}$$

$$\begin{array}{ll|ll}
\overset{\text{Def.}}{\Rightarrow} & s_{j_1}, \dots, s_{j_m} \in \mathrm{fn}(B) & \overset{\text{Hyp.}}{\Rightarrow} & \bigwedge_{1 \leq k \leq m} \forall u \in \mathrm{fn}(M_k) \cup \mathrm{fv}(M_k).\ \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(u)) \in \mathcal{F}_0 \\
\overset{\text{Hyp.}}{\Rightarrow} & \bigwedge_{1 \leq k \leq m} \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(s_{j_k})) \in \mathcal{F}_0 & \overset{\text{Lem. 4}}{\Rightarrow} & \bigwedge_{1 \leq k \leq m} \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(M_k)) \in \mathcal{F}_0 \\
\overset{\mathcal{S} \leq \mathcal{T}}{\Rightarrow} & \bigwedge_{1 \leq k \leq m} \mathsf{attacker}(\overline{\mathcal{T}}, \mathcal{E}(s_{j_k})) \in \mathcal{F}_0 & \overset{\mathcal{S} \leq \mathcal{T}}{\Rightarrow} & \bigwedge_{1 \leq k \leq m} \mathsf{attacker}(\overline{\mathcal{T}}, \mathcal{E}(M_k)) \in \mathcal{F}_0 \\
\overset{\mathcal{E}_0(s_{j_k}) = \mathcal{E}(s_{j_k})}{\Rightarrow} & \bigwedge_{1 \leq k \leq m} \mathsf{attacker}(\overline{\mathcal{T}}, \mathcal{E}_0(s_{j_k})) \in \mathcal{F}_0 & &
\end{array}$$

Moreover,

$$\begin{array}{ll}
\overset{\text{Def.}}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{E}(\mathcal{S}_0)}, \mathit{attch}[]) \in \mathcal{C}_0 \\
\overset{\text{Def.}}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{E}(\mathcal{S}_0)}, \mathit{attch}[]) \in \mathcal{F}_0 \\
\overset{\mathcal{E}(\mathcal{S}_0) \leq \mathcal{S}}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{S}}, \mathit{attch}[]) \in \mathcal{F}_0 \\
\overset{\mathcal{S} \leq \mathcal{T}}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{T}}, \mathit{attch}[]) \in \mathcal{F}_0
\end{array}$$

Combining all this we can infer the following

$$\begin{array}{ll}
\overset{\mathcal{C}_{i_1}, \mathcal{C}_{i_2} \in \mathcal{C}_0 \text{ and } \mathcal{S} \leq \mathcal{T}}{\Rightarrow} & \forall K, L\ \mathsf{message}(\overline{\mathcal{T}}, K, L) \in \mathcal{F}_0 \ \Rightarrow\ \mathsf{message}(\overline{\mathcal{T}'}, K, L) \in \mathcal{F}_0 \\
& \forall K\ \mathsf{attacker}(\overline{\mathcal{T}}, K) \in \mathcal{F}_0 \ \Rightarrow\ \mathsf{attacker}(\overline{\mathcal{T}'}, K) \in \mathcal{F}_0 \\
& \mathsf{attacker}(\overline{\mathcal{T}}, \mathit{attch}[]) \in \mathcal{F}_0 \\
\overset{\text{Def.}}{\Rightarrow} & \mathcal{T} \leq \mathcal{T}'
\end{array}$$

(i) By hypothesis $\mathcal{E}_0 \subseteq \mathcal{E}$.

(ii)
$$\begin{array}{ll}
\overset{\text{Hyp.}}{\Rightarrow} & \mathcal{E}(\mathcal{S}_0) \leq \mathcal{S} \\
\overset{\text{Transitivity of } \leq}{\Rightarrow} & \mathcal{E}(\mathcal{S}_0) \leq \mathcal{T} \\
\overset{\text{Transitivity of } \leq}{\Rightarrow} & \mathcal{E}(\mathcal{S}_0) \leq \mathcal{T}'
\end{array}$$

(iii)
$$\begin{array}{ll}
\overset{\text{Def.}}{\Rightarrow} & \mathrm{bn}(B') \cup \mathrm{bv}(B') = \mathrm{bn}(B) \cup \mathrm{bv}(B) \\
\overset{\text{Hyp.}}{\Rightarrow} & (\mathrm{bn}(B') \cup \mathrm{bv}(B')) \cap \mathrm{dom}(\mathcal{E}) = (\mathrm{bn}(B) \cup \mathrm{bv}(B)) \cap \mathrm{dom}(\mathcal{E}) = \emptyset
\end{array}$$

(iv) Let $a \in \mathrm{fn}(B')$.
$$\begin{array}{ll}
\overset{\text{Def.}}{\Rightarrow} & a \in \mathrm{fn}(B) \\
\overset{\text{Hyp.}}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0 \\
\overset{\mathcal{S} \leq \mathcal{T}}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{T}}, \mathcal{E}(a)) \in \mathcal{F}_0 \\
\overset{\mathcal{T} \leq \mathcal{T}'}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{T}'}, \mathcal{E}(a)) \in \mathcal{F}_0
\end{array}$$

(v) Let $y \in \mathrm{fv}(B')$.
$$\begin{array}{ll}
\overset{\text{Def.}}{\Rightarrow} & y \in \mathrm{fv}(B) \\
\overset{\text{Hyp.}}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{S}}, \mathcal{E}(y)) \in \mathcal{F}_0 \\
\overset{\mathcal{S} \leq \mathcal{T}}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{T}}, \mathcal{E}(y)) \in \mathcal{F}_0 \\
\overset{\mathcal{T} \leq \mathcal{T}'}{\Rightarrow} & \mathsf{attacker}(\overline{\mathcal{T}'}, \mathcal{E}(y)) \in \mathcal{F}_0
\end{array}$$

(vi) B is in the scope of a $\mathsf{lock}\ \ldots s_i \ldots$ if and only if B' is too, so B', λ satisfy condition (vi) by hypothesis.

Thus $(B', \mathcal{E}, \mathcal{T}', \iota, \lambda)$ satisfy conditions (i)- (vi), so we can apply our inductive hypothesis

$$(\mathcal{E}, \mathcal{T}', \iota, \lambda) \vdash B'$$

and thus, according to our typing system

$$\frac{\begin{array}{r} \forall \mathcal{T}\ (\mathcal{S} \leq \mathcal{T}\ \wedge\ \mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda])\ \Rightarrow \\ (\mathcal{T} \leq \mathcal{T}[j_k \mapsto \mathcal{E}(M_k) \mid 1 \leq k \leq m] \wedge \\ (\mathcal{E}, \mathcal{T}[j_k \mapsto \mathcal{E}(M_k) \mid 1 \leq k \leq m], \iota, \lambda) \vdash B') \end{array}}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash s_{j_1}, \ldots, s_{j_m} := M_1, \ldots, M_m; B'\ (= B)}\ \tau_{write}$$

To conclude the proof of Lemma 1 we then need to show that A, $\mathcal{E}_0$, $\mathcal{E}_0(\mathcal{S}_0)$, $[]$, and $\emptyset$ satisfy conditions (i)- (vi).

(i) By definition $\mathcal{E}_0 \subseteq \mathcal{E}_0$.

(ii) By definition $\mathcal{E}_0(\mathcal{S}_0) \leq \mathcal{E}_0(\mathcal{S}_0)$.

(iii) By hypotheses, $\mathrm{dom}(\mathcal{E}_0) = \mathrm{fn}(P) \cup \mathrm{fn}(A) \cup \mathit{cellP}$ and $(\mathrm{bn}(A) \cup \mathrm{bv}(A)) \cap (\mathrm{fn}(P) \cup \mathrm{fn}(A) \cup \mathit{cellP}) = \emptyset$, thus $(\mathrm{bn}(A) \cup \mathrm{bv}(A)) \cap \mathrm{dom}(\mathcal{E}_0) = \emptyset$.

(iv) By construction, $\forall a \in \mathrm{fn}(A)$

If $a = \mathit{attch}$, then $\mathcal{E}_0(a) = \mathit{attch}[]$, and $\mathsf{attacker}(\overline{\mathcal{E}_0(\mathcal{S}_0)}, \mathit{attch}[]) \in \mathcal{C}_0$ by construction.

If $a \neq \mathit{attch}$, then $\mathcal{E}_0(a) = \mathit{attn}[]$, and $\mathsf{attacker}(\overline{\mathcal{E}_0(\mathcal{S}_0)}, \mathit{attn}[]) \in \mathcal{C}_0$ by construction.

Thus $\forall a \in \mathrm{fn}(A)\ \mathsf{attacker}(\overline{\mathcal{E}_0(\mathcal{S}_0)}, \mathcal{E}_0(a)) \in \mathcal{F}_0$.

(v) A is an Init-adversary, so it is a closed process. Thus $\mathrm{fv}(A) = \emptyset$.

(vi) A is by definition under no lock in A, thus by definition $A, \emptyset$ satisfy condition (vi)

We can thus apply the preliminary result we just established to conclude that $(\mathcal{E}_0, \mathcal{E}_0(\mathcal{S}_0), [], \emptyset) \vdash A$. □

B.3 Proof of Lemma 2: Typability of P_0

Lemma 2 (Typability of P_0)**.**

$$(\mathcal{E}_0, \mathcal{E}_0(\mathcal{S}_0), [], \emptyset) \vdash P_0$$

Proof. Let Q be a subprocess of P_0 and σ, ρ, H, ι, ϕ, and λ. We first prove by induction on the size of Q, that if

(i) ρ binds all the free names and variables of Q, H, ι and ϕ;

(ii) $(\mathrm{bn}(Q) \cup \mathrm{bv}(Q)) \cap \mathrm{dom}(\rho) = \emptyset$;

(iii) σ is a closed substitution;

(iv) $i \in \lambda$ if and only if Q is in the scope of $\mathsf{lock}\ \dots s_i \dots$ in P_0;

(v) $\mathcal{C}_0 \supseteq [\![Q]\!]\rho H \iota \overline{\phi} \lambda$;

(vi) $\forall \mathsf{message}(\xi, M, N) \in H$, $\mathsf{message}(\xi\sigma, M\sigma, N\sigma)$ can be derived from $\mathcal{C}_0$

(vii) $\mathsf{attacker}(\overline{\phi}\sigma, attch[]) \in \mathcal{F}_0$,

then

$$(\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash Q.$$

Base case ($|Q| = 0$). In that case $Q = 0$, and thus according to the rule τ_{nil} of our type system

$$\frac{}{(\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash 0\ (= Q)}\ \tau_{nil}$$

Inductive case ($|Q| > 0$). We proceed by case analysis on the structure of Q.

Case $Q = Q_1 \mid Q_2$. In that case, $\lambda = \emptyset$ because no parallel composition can occur under a lock. We will show that $(Q_1, \sigma, \rho, H, \iota, \phi, \lambda)$ and $(Q_2, \sigma, \rho, H, \iota, \phi, \lambda)$ satisfy conditions (i)-(vii)

(i) By definition, $\mathrm{fv}(Q_1) \cup \mathrm{fv}(Q_2) = \mathrm{fv}(Q)$ and $\mathrm{fn}(Q_1) \cup \mathrm{fn}(Q_2) = \mathrm{fn}(Q)$. Thus if ρ binds the free names and variables of Q, it also binds the free names and variables of Q_1 and Q_2.

(ii)
$$\begin{array}{ll} \overset{\text{Def.}}{\Rightarrow} & \mathrm{bn}(Q_1) \cup \mathrm{bv}(Q_1) \subseteq \mathrm{bn}(Q) \cup \mathrm{bv}(Q) \\ & \mathrm{bn}(Q_2) \cup \mathrm{bv}(Q_2) \subseteq \mathrm{bn}(Q) \cup \mathrm{bv}(Q) \\ \overset{\text{Hyp.}}{\Rightarrow} & (\mathrm{bn}(Q_1) \cup \mathrm{bv}(Q_1)) \cap \mathrm{dom}(\rho) \subseteq (\mathrm{bn}(Q) \cup \mathrm{bv}(Q)) \cap \mathrm{dom}(\rho) = \emptyset \\ & (\mathrm{bn}(Q_2) \cup \mathrm{bv}(Q_2)) \cap \mathrm{dom}(\rho) \subseteq (\mathrm{bn}(Q) \cup \mathrm{bv}(Q)) \cap \mathrm{dom}(\rho) = \emptyset \end{array}$$

(iii) By hypothesis σ is a closed substitution.

(iv) By definition since Q is under a $\mathsf{lock}\ \dots s_i \dots$ in P_0 if and only if Q_1 and Q_2 are also in the scope of a $\mathsf{lock}\ \dots s_i \dots$ in P_0, thus condition (iii) is satisfied by hypothesis.

(v)
$$\begin{array}{lll} \mathcal{C}_0 & \overset{\text{Hyp.}}{\supseteq} & [\![Q_1 \mid Q_2]\!]\rho H \iota \overline{\phi} \lambda \\ & \overset{\text{Def.}}{=} & [\![Q_1]\!]\rho H \iota \overline{\phi} \lambda \cup [\![Q_2]\!]\rho H \iota \overline{\phi} \lambda \end{array}$$

(vi) Let $\mathsf{message}(\xi, K, L) \in H$. By hypothesis, we know that $\mathsf{message}(\xi\sigma, K\sigma, L\sigma)$ is derivable from $\mathcal{C}_0$.

(vii) By hypothesis $\mathsf{attacker}(\overline{\phi}\sigma, attch[]) \in \mathcal{F}_0$

We can thus apply our induction hypothesis to infer that

$$(\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash Q_1 \qquad \text{and} \qquad (\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash Q_2$$

But then according to our type system

$$\frac{(\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash Q_1 \qquad (\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash Q_2}{(\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash Q_1 \mid Q_2\ (= Q)}\ \tau_{par}$$

Case $Q = !Q'$. In that case, $\lambda = \emptyset$ because no replication can occur under a lock. We will show that $(Q', \sigma, \rho, H, \iota, \phi, \lambda)$ satisfy conditions (i)-(vii)

(i) By definition, $\mathrm{fv}(Q') = \mathrm{fv}(Q)$ and $\mathrm{fn}(Q') = \mathrm{fn}(Q)$. Thus if ρ binds the free names and variables of Q, it also binds the free names and variables of Q'.

(ii)
$$\begin{aligned} &\overset{\text{Def.}}{\Rightarrow} && \mathrm{bn}(Q') \cup \mathrm{bv}(Q') = \mathrm{bn}(Q) \cup \mathrm{bv}(Q) \\ &\overset{\text{Hyp.}}{\Rightarrow} && (\mathrm{bn}(Q') \cup \mathrm{bv}(Q')) \cap \mathrm{dom}(\rho) = (\mathrm{bn}(Q) \cup \mathrm{bv}(Q)) \cap \mathrm{dom}(\rho) = \emptyset \end{aligned}$$

(iii) By hypothesis σ is a closed substitution.

(iv) By definition Q is under a $\mathsf{lock}\ \ldots s_i \ldots$ in P_0 if and only if Q' is also under a $\mathsf{lock}\ \ldots s_i \ldots$ in P_0, thus condition (iii) is satisfied by hypothesis.

(v)
$$\begin{aligned} \mathcal{C}_0 &\overset{\text{Hyp.}}{\supseteq} && \llbracket !Q' \rrbracket \rho H \iota \overline{\phi} \lambda \\ &\overset{\text{Def.}}{=} && \llbracket Q' \rrbracket \rho H \iota \overline{\phi} \lambda \end{aligned}$$

(vi) Let $\mathsf{message}(\xi, K, L) \in H$. By hypothesis, we know that $\mathsf{message}(\xi\sigma, K\sigma, L\sigma)$ is derivable from $\mathcal{C}_0$.

(vii) By hypothesis $\mathsf{attacker}(\overline{\phi}\sigma, attch[]) \in \mathcal{F}_0$

We can thus apply our induction hypothesis to infer that

$$(\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash Q'$$

But then according to our type system

$$\frac{(\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash Q'}{(\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash !Q' \ (= Q)}\ \tau_{repl}$$

Case $Q = \text{if } M = N \text{ then } Q_1 \text{ else } Q_2$ with $M\rho\sigma = N\rho\sigma$. Let $\theta = \mathrm{mgu}(\rho(M), \rho(N))$. Since $M\rho\sigma = N\rho\sigma$, by definition of a most general unifier, there exists σ' *s.t.* $\sigma = \theta\sigma'$. Let $\rho' = \rho\theta$, $H' = H\theta$, $\iota' = \iota\theta$, $\phi' = \phi\theta$, and $\lambda' = \lambda$. We show that $(Q_1, \sigma', \rho', H', \iota', \phi', \lambda')$ and $(Q_2, \sigma, \rho, H, \iota, \phi, \lambda)$ satisfy conditions (i)-(vii).

(i)
$$\begin{aligned} &\overset{\text{Def.}}{\Rightarrow} && \mathrm{fn}(Q_1) \cup \mathrm{fv}(Q_1) \cup \mathrm{fn}(\iota') \cup \mathrm{fv}(\iota') \cup \mathrm{fn}(H') \cup \mathrm{fv}(H') \cup \mathrm{fn}(\phi') \cup \mathrm{fv}(\phi') \subseteq \\ & && \mathrm{fn}(Q) \cup \mathrm{fv}(Q) \cup \mathrm{fn}(\iota\theta) \cup \mathrm{fv}(\iota\theta) \cup \mathrm{fn}(H\theta) \cup \mathrm{fv}(H\theta) \cup \mathrm{fn}(\phi\theta) \cup \mathrm{fv}(\phi\theta) \\ &\overset{\text{Def.}}{\Rightarrow} && \mathrm{fn}(Q_1) \cup \mathrm{fv}(Q_1) \cup \mathrm{fn}(\iota') \cup \mathrm{fv}(\iota') \cup \mathrm{fn}(H') \cup \mathrm{fv}(H') \cup \mathrm{fn}(\phi') \cup \mathrm{fv}(\phi') \subseteq \\ & && \mathrm{fn}(Q) \cup \mathrm{fv}(Q) \cup \mathrm{fn}(\iota) \cup \mathrm{fv}(\iota) \cup \mathrm{fn}(H) \cup \mathrm{fv}(H) \cup \mathrm{fn}(\phi) \cup \mathrm{fv}(\phi) \cup \\ & && \mathrm{fn}(\{M, N\}) \cup \mathrm{fv}(\{M, N\}) \\ &\overset{\text{Def.}}{\Rightarrow} && \mathrm{fn}(Q_1) \cup \mathrm{fv}(Q_1) \cup \mathrm{fn}(\iota') \cup \mathrm{fv}(\iota') \cup \mathrm{fn}(H') \cup \mathrm{fv}(H') \cup \mathrm{fn}(\phi') \cup \mathrm{fv}(\phi') \subseteq \\ & && \mathrm{fn}(Q) \cup \mathrm{fv}(Q) \cup \mathrm{fn}(\iota) \cup \mathrm{fv}(\iota) \cup \mathrm{fn}(H) \cup \mathrm{fv}(H) \cup \mathrm{fn}(\phi) \cup \mathrm{fv}(\phi) \\ &\overset{\text{Hyp.}}{\Rightarrow} && \mathrm{fn}(Q_1) \cup \mathrm{fv}(Q_1) \subseteq \mathrm{dom}(\rho) \\ &\overset{\mathrm{dom}(\rho)=\mathrm{dom}(\rho')}{\Rightarrow} && \mathrm{fn}(Q_1) \cup \mathrm{fv}(Q_1) \subseteq \mathrm{dom}(\rho') \\ &\Rightarrow && \rho' \text{ binds the free names and variables of } Q_1, \iota', H', \phi' \end{aligned}$$

$$\begin{aligned}
&\overset{\text{Def.}}{\Rightarrow} && \text{fn}(Q_2) \cup \text{fv}(Q_2) \subseteq \text{fn}(Q) \cup \text{fv}(Q) \\
&\overset{\text{Hyp.}}{\Rightarrow} && \text{fn}(Q_2) \cup \text{fv}(Q_2) \subseteq \text{dom}(\rho) \\
&\Rightarrow && \rho \text{ binds the free names and variables of } Q_2, \iota, H, \phi
\end{aligned}$$

(ii)

$$\begin{aligned}
&\overset{\text{Def.}}{\Rightarrow} && \text{bn}(Q_1) \cup \text{bv}(Q_1) \subseteq \text{bn}(Q) \cup \text{bv}(Q) \\
&\overset{\text{Def.}}{\Rightarrow} && (\text{bn}(Q_1) \cup \text{bv}(Q_1)) \cap \text{dom}(\rho) \subseteq (\text{bn}(Q) \cup \text{bv}(Q)) \cap \text{dom}(\rho) = \emptyset \\
&\overset{\text{dom}(\rho)=\text{dom}(\rho')}{\Rightarrow} && (\text{bn}(Q_1) \cup \text{bv}(Q_1)) \cap \text{dom}(\rho') = \emptyset
\end{aligned}$$

$$\begin{aligned}
&\overset{\text{Def.}}{\Rightarrow} && \text{bn}(Q_2) \cup \text{bv}(Q_2) \subseteq \text{bn}(Q) \cup \text{bv}(Q) \\
&\overset{\text{Def.}}{\Rightarrow} && (\text{bn}(Q_2) \cup \text{bv}(Q_2)) \cap \text{dom}(\rho) \subseteq (\text{bn}(Q) \cup \text{bv}(Q)) \cap \text{dom}(\rho) = \emptyset
\end{aligned}$$

(iii)

$$\begin{aligned}
&\overset{\text{Hyp.}}{\Rightarrow} && \sigma \text{ is closed} \\
&\overset{\sigma=\theta\sigma'}{\Rightarrow} && \sigma' \text{ is closed}
\end{aligned}$$

(iv) By definition Q is under a $\mathsf{lock}\ \ldots s_i \ldots$ in P_0 if and only if Q_1 and Q_2 are under a $\mathsf{lock}\ \ldots s_i \ldots$, so condition (iii) is satisfied by hypothesis because $\lambda' = \lambda$.

(v)

$$\begin{aligned}
\mathcal{C}_0 &\overset{\text{Hyp.}}{\supseteq} [\![\text{if } M = N \text{ then } Q_1 \text{ else } Q_2]\!]\rho H \iota \overline{\phi} \lambda \\
&\overset{\text{Def.}}{=} [\![Q_1]\!](\rho\theta)(H\theta)(\iota\theta)(\overline{\phi}\theta)\lambda \cup [\![Q_2]\!]\rho H \iota \overline{\phi} \lambda \\
&\overset{\text{Def.}}{=} [\![Q_1]\!]\rho' H' \iota' \overline{\phi'} \lambda' \cup [\![Q_2]\!]\rho H \iota \overline{\phi} \lambda
\end{aligned}$$

(vi) Let $\mathsf{message}(\xi, K, L) \in H'$.

$$\begin{aligned}
&\overset{H'=H\theta}{\Rightarrow} && \mathsf{message}(\xi, K, L) \in H\theta \\
&\overset{\text{Def.}}{\Rightarrow} && \mathsf{message}(\xi', K', L') \in H \ \wedge\ \xi = \xi'\theta \ \wedge\ K = K'\theta \ \wedge\ L = L'\theta \\
&\overset{\text{Hyp.}}{\Rightarrow} && \mathsf{message}(\xi'\sigma, K'\sigma, L'\sigma) \text{ is derivable from } \mathcal{C}_0 \\
&\overset{\sigma=\theta\sigma'}{\Rightarrow} && \mathsf{message}(\xi'\theta\sigma', K'\theta\sigma', L'\theta\sigma') \text{ is derivable from } \mathcal{C}_0 \\
&\overset{\xi=\xi'\ K=K'\theta\ L=L'\theta}{\Rightarrow} && \mathsf{message}(\xi\sigma', K\sigma', L\sigma') \text{ is derivable from } \mathcal{C}_0
\end{aligned}$$

Let $\mathsf{message}(\xi, K, L) \in H$. By hypothesis, $\mathsf{message}(\xi\sigma, K\sigma, L\sigma)$ is derivable from $\mathcal{C}_0$.

(vii)

$$\begin{array}{ll|ll}
\overset{\text{Hyp.}}{\Rightarrow} & \mathsf{attacker}(\overline{\phi}\sigma, attch[]) & \overset{\text{Hyp.}}{\Rightarrow} & \mathsf{attacker}(\overline{\phi}\sigma, attch[]) \\
\overset{\sigma=\theta\sigma'}{\Rightarrow} & \mathsf{attacker}(\overline{\phi}\theta\sigma', attch[]) & & \\
\overset{\phi'=\phi\theta}{\Rightarrow} & \mathsf{attacker}(\overline{\phi'}\sigma', attch[]) & &
\end{array}$$

We can now apply our induction hypothesis to infer that

$$\begin{array}{llll}
\overset{\text{I.H.}}{\Rightarrow} & (\rho'\sigma', \phi'\sigma', \iota'\sigma', \lambda') \vdash Q_1 & \overset{\text{I.H.}}{\Rightarrow} & (\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash Q_2 \\
\overset{\rho'=\rho\theta\ \phi'=\phi\theta\ \iota'=\iota\theta\ \lambda'=\lambda}{\Rightarrow} & (\rho\theta\sigma', \phi\theta\sigma', \iota\theta\sigma', \lambda) \vdash Q_1 & & \\
\overset{\sigma=\theta\sigma'}{\Rightarrow} & (\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash Q_1 & &
\end{array}$$

and thus according to our typing system

$$\frac{M\rho\sigma = N\rho\sigma \qquad (\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash Q_1 \qquad (\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash Q_2}{(\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash \text{if } M = N \text{ then } Q_1 \text{ else } Q_2}\ \tau_{if}$$

Case $Q = \text{if } M = N \text{ then } Q_1 \text{ else } Q_2$ **with** $M\rho\sigma \neq N\rho\sigma$**.** We show that $(Q_2, \sigma, \rho, H, \iota, \phi, \lambda)$ satisfy conditions (i)–(vii).

(i) $$\begin{aligned} &\overset{\text{Def.}}{\Rightarrow} && \text{fn}(Q_2) \cup \text{fv}(Q_2) \subseteq \text{fn}(Q) \cup \text{fv}(Q) \\ &\overset{\text{Hyp.}}{\Rightarrow} && \text{fn}(Q_2) \cup \text{fv}(Q_2) \subseteq \text{dom}(\rho) \\ &\Rightarrow && \rho \text{ binds the free names and variables of } Q_2, \iota, H, \phi \end{aligned}$$

(ii) $$\begin{aligned} &\overset{\text{Def.}}{\Rightarrow} && \text{bn}(Q_2) \cup \text{bv}(Q_2) \subseteq \text{bn}(Q) \cup \text{bv}(Q) \\ &\overset{\text{Def.}}{\Rightarrow} && (\text{bn}(Q_2) \cup \text{bv}(Q_2)) \cap \text{dom}(\rho) \subseteq (\text{bn}(Q) \cup \text{bv}(Q)) \cap \text{dom}(\rho) = \emptyset \end{aligned}$$

(iii) By hypothesis, σ is closed.

(iv) By definition Q is under a $\mathsf{lock}\ \ldots s_i \ldots$ in P_0 if and only if Q_2 are under a $\mathsf{lock}\ \ldots s_i \ldots$, so condition (iii) is satisfied by hypothesis.

(v) $$\begin{aligned} \mathcal{C}_0 &\overset{\text{Hyp.}}{\supseteq} && [\![\text{if } M = N \text{ then } Q_1 \text{ else } Q_2]\!]\rho H \iota \overline{\phi} \lambda \\ &\overset{\text{Def.}}{\supseteq} && [\![Q_2]\!]\rho H \iota \overline{\phi} \lambda \end{aligned}$$

(vi) Let $\mathsf{message}(\xi, K, L) \in H$. By hypothesis, $\mathsf{message}(\xi\sigma, K\sigma, L\sigma)$ is derivable from $\mathcal{C}_0$.

(vii) $\overset{\text{Hyp.}}{\Rightarrow}\ \mathsf{attacker}(\overline{\phi}\sigma, attch[])$

We can now apply our induction hypothesis to infer that

$$(\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash Q_2$$

and thus according to our typing system

$$\frac{M\rho\sigma \neq N\rho\sigma \qquad\qquad (\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash Q_2}{(\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash \text{if } M = N \text{ then } Q_1 \text{ else } Q_2}\ \tau_{if}$$

Case $Q = \text{let } x = g(M_1, \ldots, M_k) \text{ in } Q_1 \text{ else } Q_2$**.** Let M be a pattern such that $g(\rho\sigma(M_1), \ldots, \rho\sigma(M_k)) \to M$ using $g(p_1, \ldots, p_k) \to p \in def(g)$ with $\text{fv}(\{p_1, \ldots, p_k, p\}) = \{x_1, \ldots, x_\ell\}$. Let $\sigma' = \{x_i \mapsto y_i \mid 1 \leq i \leq \ell\}$ with $y_1, \ldots, y_\ell$ fresh, and $\theta = \text{mgu}(g(\rho(M_1), \ldots, \rho(M_k)), g(p_1\sigma', \ldots, p_k\sigma'))$. By definition of a most general unifier, there exists σ'' *s.t.* $\sigma = \theta\sigma''$ and $M = p\sigma'\theta$. Let $\rho' = \rho\theta \cup \{x \mapsto p\sigma'\theta\} \cup \{y_i \mapsto y_i \mid 1 \leq i \leq \ell\}$, $H' = H\theta$, $\iota' = \iota\theta$, $\phi' = \phi\theta$, and $\lambda' = \lambda$. We show that $(Q_1, \sigma'', \rho', H', \iota', \phi', \lambda')$ and $(Q_2, \sigma, \rho, H, \iota, \phi, \lambda)$ satisfy conditions (i)-(vii).

(i)

$$\begin{array}{rl}
\overset{\text{Def.}}{\Rightarrow} & \mathrm{fn}(Q_1) \cup \mathrm{fv}(Q_1) \cup \mathrm{fn}(\iota') \cup \mathrm{fv}(\iota') \cup \mathrm{fn}(H') \cup \mathrm{fv}(H') \cup \mathrm{fn}(\phi') \cup \mathrm{fv}(\phi') \subseteq \\
& \quad \mathrm{fn}(Q) \cup \mathrm{fv}(Q) \cup \{x\} \cup \mathrm{fn}(\iota\theta) \cup \mathrm{fv}(\iota\theta) \cup \mathrm{fn}(H\theta) \cup \mathrm{fv}(H\theta) \cup \mathrm{fn}(\phi\theta) \cup \mathrm{fv}(\phi\theta) \\
\overset{\text{Def.}}{\Rightarrow} & \mathrm{fn}(Q_1) \cup \mathrm{fv}(Q_1) \cup \mathrm{fn}(\iota') \cup \mathrm{fv}(\iota') \cup \mathrm{fn}(H') \cup \mathrm{fv}(H') \cup \mathrm{fn}(\phi') \cup \mathrm{fv}(\phi') \subseteq \\
& \quad \mathrm{fn}(Q) \cup \mathrm{fv}(Q) \cup \{x\} \cup \mathrm{fn}(\iota) \cup \mathrm{fv}(\iota) \cup \mathrm{fn}(H) \cup \mathrm{fv}(H) \cup \\
& \qquad \mathrm{fn}(\phi) \cup \mathrm{fv}(\phi) \cup \mathrm{fn}(\{M_1, \ldots, M_k\}) \cup \mathrm{fv}(\{M_1, \ldots, M_k\}) \cup \{y_i \mid 1 \leq i \leq \ell\} \\
\overset{\text{Def.}}{\Rightarrow} & \mathrm{fn}(Q_1) \cup \mathrm{fv}(Q_1) \cup \mathrm{fn}(\iota') \cup \mathrm{fv}(\iota') \cup \mathrm{fn}(H') \cup \mathrm{fv}(H') \cup \mathrm{fn}(\phi') \cup \mathrm{fv}(\phi') \subseteq \\
& \quad \mathrm{fn}(Q) \cup \mathrm{fv}(Q) \cup \mathrm{fn}(\iota) \cup \mathrm{fv}(\iota) \cup \mathrm{fn}(H) \cup \mathrm{fv}(H) \cup \mathrm{fn}(\phi) \cup \mathrm{fv}(\phi) \\
\overset{\text{Hyp.}}{\Rightarrow} & \mathrm{fn}(Q_1) \cup \mathrm{fv}(Q_1) \subseteq \mathrm{dom}(\rho) \cup \{x\} \cup \{y_i \mid 1 \leq i \leq \ell\} \\
\overset{\mathrm{dom}(\rho') = \mathrm{dom}(\rho) \cup \{x\} \cup \{y_i \mid 1 \leq i \leq \ell\}}{\Rightarrow} & \mathrm{fn}(Q_1) \cup \mathrm{fv}(Q_1) \subseteq \mathrm{dom}(\rho') \\
\Rightarrow & \rho' \text{ binds the free names and variables of } Q_1, \iota', H', \phi'
\end{array}$$

$$\begin{array}{rl}
\overset{\text{Def.}}{\Rightarrow} & \mathrm{fn}(Q_2) \cup \mathrm{fv}(Q_2) \subseteq \mathrm{fn}(Q) \cup \mathrm{fv}(Q) \\
\overset{\text{Hyp.}}{\Rightarrow} & \mathrm{fn}(Q_2) \cup \mathrm{fv}(Q_2) \subseteq \mathrm{dom}(\rho) \\
\Rightarrow & \rho \text{ binds the free names and variables of } Q_2, \iota, H, \phi
\end{array}$$

(ii)

$$\begin{array}{rl}
\overset{\text{Def.}}{\Rightarrow} & \mathrm{bn}(Q_1) \cup \mathrm{bv}(Q_1) \subseteq \mathrm{bn}(Q) \cup \mathrm{bv}(Q) \\
\overset{\text{Def.}}{\Rightarrow} & (\mathrm{bn}(Q_1) \cup \mathrm{bv}(Q_1)) \cap \mathrm{dom}(\rho) \subseteq (\mathrm{bn}(Q) \cup \mathrm{bv}(Q)) \cap \mathrm{dom}(\rho) = \emptyset \\
\overset{\substack{x \notin \mathrm{bv}(Q_1) \\ y_1, \ldots, y_\ell \text{ fresh}}}{\Rightarrow} & (\mathrm{bn}(Q_1) \cup \mathrm{bv}(Q_1)) \cap \mathrm{dom}(\rho') = \emptyset
\end{array}$$

$$\begin{array}{rl}
\overset{\text{Def.}}{\Rightarrow} & \mathrm{bn}(Q_2) \cup \mathrm{bv}(Q_2) \subseteq \mathrm{bn}(Q) \cup \mathrm{bv}(Q) \\
\overset{\text{Def.}}{\Rightarrow} & (\mathrm{bn}(Q_2) \cup \mathrm{bv}(Q_2)) \cap \mathrm{dom}(\rho) \subseteq (\mathrm{bn}(Q) \cup \mathrm{bv}(Q)) \cap \mathrm{dom}(\rho) = \emptyset
\end{array}$$

(iii)

$$\begin{array}{rl}
\overset{\text{Hyp.}}{\Rightarrow} & \sigma \text{ is closed} \\
\overset{\sigma = \theta\sigma''}{\Rightarrow} & \sigma'' \text{ is closed}
\end{array}$$

(iv) By definition Q is under a $\mathsf{lock}\ \ldots s_i \ldots$ in P_0 if and only if Q_1 and Q_2 are under a $\mathsf{lock}\ \ldots s_i \ldots$, so condition (iii) is satisfied by hypothesis because $\lambda' = \lambda$.

(v)

$$\begin{array}{rcl}
\mathcal{C}_0 & \overset{\text{Hyp.}}{\supseteq} & [\![\text{let } x = g(M_1, \ldots, M_k) \text{ in } Q_1 \text{ else } Q_2]\!]\rho H \iota \overline{\phi} \lambda \\
& \overset{\text{Def.}}{=} & [\![Q_1]\!]\rho' H' \iota' \overline{\phi'} \lambda' \cup [\![Q_2]\!]\rho H \iota \overline{\phi} \lambda
\end{array}$$

(vi) Let $\mathsf{message}(\xi, K, L) \in H'$.

$$\begin{array}{rl}
\overset{H' = H\theta}{\Rightarrow} & \mathsf{message}(\xi, K, L) \in H\theta \\
\overset{\text{Def.}}{\Rightarrow} & \mathsf{message}(\xi', K', L') \in H \ \wedge \ \xi = \xi'\theta \ \wedge \ K = K'\theta \ \wedge \ L = L'\theta \\
\overset{\text{Hyp.}}{\Rightarrow} & \mathsf{message}(\xi'\sigma, K'\sigma, L'\sigma) \text{ is derivable from } \mathcal{C}_0 \\
\overset{\sigma = \theta\sigma''}{\Rightarrow} & \mathsf{message}(\xi'\theta\sigma'', K'\theta\sigma'', L'\theta\sigma'') \text{ is derivable from } \mathcal{C}_0 \\
\overset{\xi = \xi' \ \ K = K'\theta \ \ L = L'\theta}{\Rightarrow} & \mathsf{message}(\xi\sigma'', K\sigma'', L\sigma'') \text{ is derivable from } \mathcal{C}_0
\end{array}$$

Let $\mathsf{message}(\xi, K, L) \in H$. By hypothesis, $\mathsf{message}(\xi\sigma, K\sigma, L\sigma)$ is derivable from $\mathcal{C}_0$.

(vii)

$$\begin{array}{rl|rl}
\overset{\text{Hyp.}}{\Rightarrow} & \mathsf{attacker}(\overline{\phi}\sigma, attch[]) & \overset{\text{Hyp.}}{\Rightarrow} & \mathsf{attacker}(\overline{\phi}\sigma, attch[]) \\
\overset{\sigma = \theta\sigma''}{\Rightarrow} & \mathsf{attacker}(\overline{\phi}\theta\sigma'', attch[]) & & \\
\overset{\phi' = \phi\theta}{\Rightarrow} & \mathsf{attacker}(\overline{\phi'}\sigma'', attch[]) & &
\end{array}$$

We can now apply our induction hypothesis to infer that

$$\begin{array}{ll}
\overset{\text{I.H.}}{\Rightarrow} & (\rho'\sigma'', \phi'\sigma'', \iota'\sigma'', \lambda') \vdash Q_1 \qquad\qquad \overset{\text{I.H.}}{\Rightarrow} \quad (\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash Q_2 \\
\overset{Def.}{\Rightarrow} & (\rho\theta \cup \{x \mapsto p\sigma'\theta\} \cup \{y_i \mapsto y_i \mid 1 \leq i \leq \ell\}, \phi'\sigma'', \iota'\sigma'', \lambda') \vdash Q_1 \\
\overset{\substack{y_1, \ldots, y_\ell \text{ fresh} \\ \text{Lem. 5.2}}}{\Rightarrow} & (\rho\theta \cup \{x \mapsto p\sigma'\theta\}, \phi'\sigma'', \iota'\sigma'', \lambda') \vdash Q_1 \\
\overset{M=p\sigma'\theta}{\Rightarrow} & (\rho\theta\sigma'' \cup \{x \mapsto M\}, \phi'\sigma'', \iota'\sigma'', \lambda') \vdash Q_1 \\
\overset{\phi'=\phi\theta \;\; \iota'=\iota\theta \;\; \lambda'=\lambda}{\Rightarrow} & (\rho\theta\sigma'' \cup \{x \mapsto M\}, \phi\theta\sigma'', \iota\theta\sigma'', \lambda) \vdash Q_1 \\
\overset{\sigma=\theta\sigma''}{\Rightarrow} & (\rho\sigma \cup \{x \mapsto M\}, \phi\sigma, \iota\sigma, \lambda) \vdash Q_1
\end{array}$$

and thus according to our typing system

$$\frac{\forall M \;\, (g(\rho\sigma(M_1), \ldots, \rho\sigma(M_k)) \rightarrow M) \Rightarrow (\rho\sigma \cup \{x \mapsto M\}, \phi\sigma, \iota\sigma, \lambda) \vdash Q_1 \;\, (\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash Q_2}{(\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash \text{let } x = g(M_1, \ldots, M_k) \text{ in } Q_1 \text{ else } Q_2}\, \tau_{if}$$

Case $Q = \mathsf{new}\ a; Q'$**.** Note that Q being a subprocess of P_0 implies that $a \in \text{bn}(P_0')$. Let $\rho' = \rho \cup \{a \mapsto a[\iota]\}$. We first show that $(Q', \sigma, \rho', H, \iota, \phi, \lambda)$ satisfy conditions (i)-(vii)

(i) $$\begin{array}{ll}
\overset{\text{Def.}}{\Rightarrow} & \text{fv}(Q') \cup \text{fn}(Q') \cup \text{fn}(\iota) \cup \text{fv}(\iota) \cup \text{fn}(H) \cup \text{fv}(H) \cup \text{fn}(\phi) \cup \text{fv}(\phi) = \\
& \quad \text{fv}(Q) \cup \text{fn}(Q) \cup \{a\} \cup \text{fn}(\iota) \cup \text{fv}(\iota) \cup \text{fn}(H) \cup \text{fv}(H) \cup \text{fn}(\phi) \cup \text{fv}(\phi) \\
\overset{\text{Hyp.}}{\Rightarrow} & \text{fv}(Q') \cup \text{fn}(Q') \cup \text{fn}(\iota) \cup \text{fv}(\iota) \cup \text{fn}(H) \cup \text{fv}(H) \cup \text{fn}(\phi) \cup \text{fv}(\phi) \subseteq \\
& \quad \text{dom}(\rho) \cup \{a\} \\
\overset{\text{dom}(\rho')=\text{dom}(\rho)\cup\{a\}}{\Rightarrow} & \text{fv}(Q') \cup \text{fn}(Q') \cup \text{fn}(\iota) \cup \text{fv}(\iota) \cup \text{fn}(H) \cup \text{fv}(H) \cup \text{fn}(\phi) \cup \text{fv}(\phi) \subseteq \text{dom}(\rho')
\end{array}$$

(ii) $$\begin{array}{ll}
\overset{\text{Def.}}{\Rightarrow} & \text{bn}(Q') \cup \text{bv}(Q') \subset \text{bn}(Q) \cup \text{bv}(Q) \\
\overset{\text{Hyp.}}{\Rightarrow} & (\text{bn}(Q') \cup \text{bv}(Q')) \cap \text{dom}(\rho) \subset (\text{bn}(Q) \cup \text{bv}(Q)) \cap \text{dom}(\rho) = \emptyset \\
\overset{a\notin\text{bn}(Q')}{\Rightarrow} & (\text{bn}(Q') \cup \text{bv}(Q')) \cap \text{dom}(\rho') = \emptyset
\end{array}$$

(iii) By hypothesis, σ is a closed substitution.

(iv) By definition Q is under a $\mathsf{lock}\ \ldots s_i \ldots$ in P_0 if and only if Q' is under a $\mathsf{lock}\ \ldots s_i \ldots$, so condition (iii) is satisfied by hypothesis.

(v) $$\begin{array}{lll}
\mathcal{C}_0 & \overset{\text{Hyp.}}{\supseteq} & [\![\mathsf{new}\ a; Q']\!]\rho H \iota \phi \lambda \\
& \overset{\text{Def.}}{=} & [\![Q']\!](\rho \cup \{a \mapsto a[\iota]\}) H \iota \phi \lambda \\
& \overset{\rho'=\rho\cup\{a\mapsto a[\iota]\}}{=} & [\![Q']\!]\rho' H \iota \phi \lambda
\end{array}$$

(vi) Let $\mathsf{message}(\xi, K, L) \in H$. By hypothesis $\mathsf{message}(\xi\sigma, K\sigma, L\sigma) \in H$ is derivable from $\mathcal{C}_0$.

(vii) By hypothesis $\mathsf{attacker}(\overline{\phi}\sigma, attch[]) \in \mathcal{F}_0$.

We can thus apply our induction hypothesis to infer that

$$\begin{array}{ll}
\overset{\text{I.H.}}{\Rightarrow} & ((\rho'\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash Q' \\
\overset{\rho'=\rho\cup\{a\mapsto a[\iota]\}}{\Rightarrow} & ((\rho \cup \{a \mapsto a[\iota]\})\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash Q' \\
\overset{a\notin\text{dom}(\sigma)}{\Rightarrow} & (\rho\sigma \cup \{a \mapsto a[\iota]\}, \phi\sigma, \iota\sigma, \lambda) \vdash Q'
\end{array}$$

But then according to our typing system

$$\frac{a \in \mathrm{bn}(P_0') \qquad\qquad (\rho\sigma \cup \{a \mapsto a[\iota]\}, \phi\sigma, \iota\sigma, \lambda) \vdash Q'}{(\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash \mathsf{new}\ a; Q'\ (= Q)}\ \tau_{newP}$$

Case $Q = \mathsf{out}(K, L); Q'$**.** We first prove that $(Q', \sigma, \rho, H, \iota, \phi, \lambda)$ satisfy conditions (i)-(vii).

(i) By definition $\mathrm{fn}(Q') \subseteq \mathrm{fn}(Q)$ and $\mathrm{fv}(Q') \subseteq \mathrm{fv}(Q)$. Thus since by hypothesis ρ binds the free names and variables of Q, it also binds the free names and variables of Q'.

(ii) $\overset{\text{Def.}}{\Rightarrow}\quad \mathrm{bn}(Q') \cup \mathrm{bv}(Q') \subseteq \mathrm{bn}(Q) \cup \mathrm{bv}(Q)$
$\overset{\text{Hyp.}}{\Rightarrow}\quad (\mathrm{bn}(Q') \cup \mathrm{bv}(Q')) \cap \mathrm{dom}(\rho) \subseteq (\mathrm{bn}(Q) \cup \mathrm{bv}(Q)) \cap \mathrm{dom}(\rho) = \emptyset$

(iii) By hypothesis, σ is a closed substitution.

(iv) By definition Q is under a $\mathsf{lock}\ \ldots s_i \ldots$ in P_0 if and only if Q' is under a $\mathsf{lock}\ \ldots s_i \ldots$ in P_0. Thus Q' and λ satisfy (iii) because by hypothesis Q' and λ satisfy it.

(v) $\mathcal{C}_0 \overset{\text{Hyp.}}{\supseteq} [\![\mathsf{out}(K, L); Q']\!]\rho H \iota \overline{\phi} \lambda$
$\overset{\text{Def.}}{\supseteq} [\![Q']\!]\rho H \iota \overline{\phi} \lambda$

(vi) Let $\mathsf{message}(\xi, K, L) \in H$. By hypothesis, we know that $\mathsf{message}(\xi\sigma, K\sigma, L\sigma)$ is derivable from $\mathcal{C}_0$.

(vii) By hypothesis, $\mathsf{attacker}(\overline{\phi}\sigma, \mathit{attch}[]) \in \mathcal{F}_0$.

We can thus apply our induction hypothesis to infer that $(\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash Q'$. Moreover, by definition of our translation

$$[\![Q]\!]\rho H \iota \overline{\phi} \lambda \;=\; \{H \Rightarrow \mathsf{message}(\overline{\phi}, \rho(K), \rho(L))\} \;\cup\; [\![Q']\!]\rho H \iota \overline{\phi} \lambda.$$

with H and σ satisfying condition (vi), *i.e.* $H\sigma$ is derivable from $\mathcal{C}_0$. So by resolution we can thus derive

$$\mathsf{message}(\phi\sigma, \rho(K)\sigma, \rho(L)\sigma) \overset{\mathrm{fn}(Q) \subseteq \mathrm{dom}(\rho)}{=} \mathsf{message}(\phi\sigma, (\rho\sigma)(K), (\rho\sigma)(L))$$

Finally, according to our typing system

$$\frac{\mathsf{message}(\overline{\phi}\sigma, (\rho\sigma)(K), (\rho\sigma)(L)) \in \mathcal{F}_0 \qquad\qquad (\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash Q'}{(\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash \mathsf{out}(K, L); Q'\ (= Q)}\ \tau_{out}$$

Case $Q = \mathsf{in}(K, x); Q'$**.** Let ψ be a state such that $\phi\sigma \leq \psi$ and $\psi = \psi[j \mapsto \phi\sigma(j) \mid j \in \lambda]$. Let L be a pattern such that $\mathsf{message}(\overline{\psi}, (\rho\sigma)(K), L) \in \mathcal{F}_0$. Let $\rho' = \rho \cup \{x \mapsto x\} \cup \{vs_j \mapsto vs_j \mid j \notin \lambda\}$, $\sigma' = \sigma \cup \{x \mapsto L\} \cup \{vs_j \mapsto \psi(j) \mid j \notin \lambda\}$, $\iota' = x :: \iota$, $\phi' = \phi[j \mapsto vs_j \mid j \notin \lambda]$, $H' = H \wedge \mathsf{message}(\overline{\phi'}, \rho(K), x)$, and $\lambda' = \lambda$, for $vs_1, \ldots, vs_n$ fresh. We show that $(Q', \sigma', \rho', H', \iota', \phi', \lambda')$ satisfy conditions (i)-(vii).

(i) $\overset{\text{Def.}}{\Rightarrow}$ $\mathrm{fv}(Q') \cup \mathrm{fn}(Q') \cup \mathrm{fv}(\iota') \cup \mathrm{fn}(\iota') \cup \mathrm{fv}(H') \cup \mathrm{fn}(H') \cup \mathrm{fv}(\phi') \cup \mathrm{fn}(\phi') \subseteq$
$\mathrm{fv}(Q) \cup \{x\} \cup \mathrm{fn}(Q) \cup \mathrm{fv}(\iota) \cup \mathrm{fn}(\iota) \cup \mathrm{fv}(H) \cup \mathrm{fn}(H) \cup$
$\mathrm{fv}(\phi) \cup \mathrm{fn}(\phi) \cup \{vs_j \mid j \notin \lambda\}$

$\overset{\text{Hyp.}}{\Rightarrow}$ $\mathrm{fv}(Q') \cup \mathrm{fn}(Q') \cup \mathrm{fv}(\iota') \cup \mathrm{fn}(\iota') \cup \mathrm{fv}(H') \cup \mathrm{fn}(H') \cup \mathrm{fv}(\phi') \cup \mathrm{fn}(\phi') \subseteq$
$\mathrm{dom}(\rho) \cup \{x\} \cup \{vs_j \mid j \notin \lambda\}$

$\overset{\mathrm{dom}(\rho') \supseteq \mathrm{dom}(\rho) \cup \{x\} \cup \{vs_j \mid j \notin \lambda\}}{\Rightarrow}$ $\mathrm{fv}(Q') \cup \mathrm{fn}(Q') \cup \mathrm{fv}(\iota') \cup \mathrm{fn}(\iota') \cup \mathrm{fv}(H') \cup \mathrm{fn}(H') \cup$
$\mathrm{fv}(\phi') \cup \mathrm{fn}(\phi') \subseteq \mathrm{dom}(\rho')$

$\Rightarrow$ ρ' binds the free names and variables of Q', ι', H', ϕ'

(ii) $\overset{\text{Def.}}{\Rightarrow}$ $\mathrm{bn}(Q') \cup \mathrm{bv}(Q') \subseteq \mathrm{bn}(Q) \cup \mathrm{bv}(Q)$

$\overset{\text{Hyp.}}{\Rightarrow}$ $(\mathrm{bn}(Q') \cup \mathrm{bv}(Q')) \cap \mathrm{dom}(\rho) \subseteq (\mathrm{bn}(Q) \cup \mathrm{bv}(Q)) \cap \mathrm{dom}(\rho) = \emptyset$

$\overset{\substack{x \notin \mathrm{bv}(Q') \\ vs_1, \ldots, vs_n \text{ fresh}}}{\Rightarrow}$ $(\mathrm{bn}(Q') \cup \mathrm{bv}(Q')) \cap \mathrm{dom}(\rho') = \emptyset$

(iii) $\overset{\text{Hyp.}}{\Rightarrow}$ $L,\ \phi(1),\ \ldots,\ \psi(n)$ are ground

$\overset{\text{Def.}}{\Rightarrow}$ $\{x \mapsto L\} \cup \{vs_j \mapsto \psi(j) \mid j \notin \lambda\}$ is closed

$\overset{\sigma \text{ closed}}{\Rightarrow}$ $\sigma \cup \{x \mapsto L\} \cup \{vs_j \mapsto \psi(j) \mid \lambda\}$ is closed

$\overset{\sigma' = \sigma \cup \{x \mapsto L\} \cup \{vs_j \mapsto \psi(j) \mid j \notin \lambda\}}{\Rightarrow}$ σ' is closed

(iv) By definition Q is under a $\mathsf{lock}\ \ldots s_i \ldots$ in P_0 if and only if Q' is under a $\mathsf{lock}\ \ldots s_i \ldots$, so condition (iii) is satisfied by hypothesis.

(v) $\mathcal{C}_0 \overset{\text{Hyp.}}{\supseteq} \llbracket \mathsf{in}(K, x); Q' \rrbracket \rho H \iota \overline{\phi} \lambda$
$\overset{\text{Hyp.}}{=} \llbracket Q' \rrbracket (\rho \cup \{x \mapsto x\} \cup \{vs_j \mapsto vs_j \mid j \notin \lambda\})(H \wedge$
$\mathsf{message}(\overline{\phi'}, \rho(K), x))(x :: \iota)\overline{\phi'}\lambda$
$\overset{\text{Def.}}{=} \llbracket Q' \rrbracket \rho' H' \iota' \overline{\phi'} \lambda'$

(vi) Let $\mathsf{message}(\xi', K', L') \in H'$. Then either $\mathsf{message}(\xi', K', L') \in H$ or $\mathsf{message}(\xi', K', L') = \mathsf{message}(\overline{\phi'}, \rho(K), x)$.

If $\mathsf{message}(\xi', K', L') \in H$**.**

$\overset{\text{Hyp.}}{\Rightarrow}$ $\mathsf{message}(\xi'\sigma, K'\sigma, L'\sigma)$ is derivable from $\mathcal{C}_0$

$\overset{\substack{x \notin \mathrm{fn}(\xi') \cup \mathrm{fn}(K') \cup \mathrm{fn}(L') \\ vs_1, \ldots, vs_n \text{ fresh}}}{\Rightarrow}$ $\mathsf{message}(\xi'\sigma', K'\sigma', L'\sigma')$ is derivable from $\mathcal{C}_0$

If $\mathsf{message}(\xi', K', L') = \mathsf{message}(\overline{\phi'}, \rho(K), x)$**.**

$\overset{\text{Hyp.}}{\Rightarrow}$ $\mathsf{message}(\overline{\psi}, \rho(K)\sigma, L)$ is derivable from $\mathcal{C}_0$

$\overset{\substack{x \notin \mathrm{fn}(\rho(K)) \\ vs_1, \ldots, vs_n \text{ fresh}}}{\Rightarrow}$ $\mathsf{message}(\overline{\psi}, \rho(K)\sigma', L)$ is derivable from $\mathcal{C}_0$

$\overset{\sigma'(x) = L}{\Rightarrow}$ $\mathsf{message}(\overline{\psi}, \rho(K)\sigma', x\sigma')$ is derivable from $\mathcal{C}_0$

$\overset{\psi = \phi'\sigma'}{\Rightarrow}$ $\mathsf{message}(\overline{\phi'}\sigma', \rho(K)\sigma', x\sigma')$ is derivable from $\mathcal{C}_0$

$\overset{\mathsf{message}(\xi', K', L') = \mathsf{message}(\overline{\phi'}, \rho(K), x)}{\Rightarrow}$ $\mathsf{message}(\xi'\sigma', K'\sigma', L'\sigma')$ is derivable from $\mathcal{C}_0$

(vii) $\overset{\text{Hyp.}}{\Rightarrow}$ $\mathsf{attacker}(\overline{\phi}\sigma, attch[]) \in \mathcal{F}_0$

$\overset{\phi\sigma \leq \psi}{\Rightarrow}$ $\mathsf{attacker}(\overline{\psi}, attch[]) \in \mathcal{F}_0$

$\overset{\phi'\sigma'}{\Rightarrow}$ $\mathsf{attacker}(\overline{\phi'}\sigma', attch[]) \in \mathcal{F}_0$

We can thus apply our induction hypothesis to infer that

$$\begin{array}{rl} \overset{\text{I.H.}}{\Rightarrow} & (\rho'\sigma', \phi'\sigma', \iota'\sigma', \lambda') \vdash Q' \\ \overset{\psi=\phi'\sigma'}{\Rightarrow} & (\rho'\sigma', \psi, \iota'\sigma', \lambda') \vdash Q' \\ \overset{\rho'\sigma'=\rho\sigma'\cup\{x\mapsto L\}}{\Rightarrow} & (\rho\sigma' \cup \{x \mapsto L\}, \psi, \iota'\sigma', \lambda') \vdash Q' \\ \overset{x\notin\rho}{\Rightarrow} & (\rho\sigma \cup \{x \mapsto L\}, \psi, \iota'\sigma', \lambda') \vdash Q' \\ \overset{\iota'\sigma'=L::\iota\sigma\wedge\lambda'=\lambda}{\Rightarrow} & (\rho\sigma \cup \{x \mapsto L\}, \psi, L :: \iota\sigma, \lambda) \vdash Q' \end{array}$$

Thus according to our typing system

$$\frac{\begin{array}{r}\forall\psi\forall L\ (\phi\sigma \leq \psi \wedge \psi = \psi[j \mapsto \phi\sigma(j) \mid j \in \lambda] \wedge \mathsf{message}(\overline{\psi}, (\rho\sigma)(K), L) \in \mathcal{F}_0 \ \Rightarrow \\ (\rho\sigma \cup \{x \mapsto L\}, \psi, L :: \iota\sigma, \lambda) \vdash Q')\end{array}}{(\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash \mathsf{in}(K, x); Q'\ (= Q)}\ \tau_{in}$$

Case $Q = \mathsf{lock}\ s_{j_1}, \ldots, s_{j_m}; Q'$ **.** Let ψ such that $\phi\sigma \leq \psi$ and $\psi = \psi[j \mapsto \phi\sigma(j) \mid j \in \lambda]$. Let $\rho' = \rho \cup \{vs_j \mapsto vs_j \mid j \notin \lambda\}$, $\sigma' = \sigma \cup \{vs_j \mapsto \psi(j) \mid j \notin \lambda\}$, $\phi' = \phi[j \mapsto vs_j \mid j \notin \lambda]$, $H' = H$, $\iota' = \iota$, $\lambda' = \lambda \cup \{j_1, \ldots, j_m\}$, for some $vs_1, \ldots, vs_n$ fresh. We will show that $(Q', \sigma', \rho', H', \iota', \phi', \lambda')$ satisfy conditions (i)-(vii).

$$\begin{array}{lrl} \textit{(i)} & \overset{\text{Def.}}{\Rightarrow} & \mathrm{fn}(Q') \cup \mathrm{fv}(Q') \cup \mathrm{fn}(\iota') \cup \mathrm{fv}(\iota') \cup \mathrm{fn}(H') \cup \mathrm{fv}(H') \cup \mathrm{fn}(\phi') \cup \\ & & \mathrm{fv}(\phi') \subseteq \mathrm{fn}(Q) \cup \mathrm{fv}(Q) \cup \mathrm{fn}(\iota) \cup \mathrm{fv}(\iota) \cup \mathrm{fn}(H) \cup \mathrm{fv}(H) \cup \\ & & \mathrm{fn}(\phi) \cup \mathrm{fv}(\phi) \cup \{vs_j \mid j \notin \lambda\} \\ & \overset{\text{Hyp.}}{\Rightarrow} & \mathrm{fn}(Q') \cup \mathrm{fv}(Q') \cup \mathrm{fn}(\iota') \cup \mathrm{fv}(\iota') \cup \mathrm{fn}(H') \cup \mathrm{fv}(H') \cup \\ & & \mathrm{fn}(\phi') \cup \mathrm{fv}(\phi') \subseteq \mathrm{dom}(\rho) \cup \{vs_j \mid j \notin \lambda\} \\ & \overset{\rho'=\mathrm{dom}(\rho)\cup\{vs_j\mapsto vs_j|j\notin\lambda\}}{\Rightarrow} & \mathrm{fn}(Q') \cup \mathrm{fv}(Q') \subseteq \mathrm{dom}(\rho') \end{array}$$

$$\begin{array}{lrl} \textit{(ii)} & \overset{\text{Def.}}{\Rightarrow} & \mathrm{bn}(Q') \cup \mathrm{bv}(Q') = \mathrm{bn}(Q) \cup \mathrm{bv}(Q) \\ & \overset{\text{Hyp.}}{\Rightarrow} & (\mathrm{bn}(Q') \cup \mathrm{bv}(Q')) \cap \mathrm{dom}(\rho) = (\mathrm{bn}(Q) \cup \mathrm{bv}(Q)) \cap \mathrm{dom}(\rho) = \emptyset \\ & \overset{vs_1,\ldots,vs_n\notin\mathrm{bv}(Q')}{\Rightarrow} & (\mathrm{bn}(Q') \cup \mathrm{bv}(Q')) \cap \mathrm{dom}(\rho') = \emptyset \end{array}$$

$$\begin{array}{lrl} \textit{(iii)} & \overset{\text{Hyp.}}{\Rightarrow} & \psi(1),\ \ldots,\ \psi(n) \text{ are ground} \\ & \overset{\text{Def.}}{\Rightarrow} & \{vs_j \mapsto \psi(j) \mid j \notin \lambda\} \text{ is closed} \\ & \overset{\sigma\ \text{closed}}{\Rightarrow} & \sigma \cup \{vs_j \mapsto \psi(j) \mid j \notin \lambda\} \text{ is closed} \\ & \overset{\sigma'=\sigma\cup\{vs_j\mapsto\psi(j)|j\notin\lambda\}}{\Rightarrow} & \sigma' \text{ is closed} \end{array}$$

(iv) By definition Q' is under a $\mathsf{lock}\ \ldots s \ldots$ if and only if either $s \in \{s_{j_1}, \ldots, s_{j_m}\}$ or Q is under a $\mathsf{lock}\ \ldots s \ldots$, so condition (iii) is satisfied by construction of λ'.

$$\begin{array}{llcl} \textit{(v)} & \mathcal{C}_0 & \overset{\text{Hyp.}}{\supseteq} & [\![\mathsf{lock}\ s_{j_1}, \ldots, s_{j_m}; Q']\!]\rho H \iota \overline{\phi} \lambda \\ & & \overset{\text{Def.}}{=} & [\![Q']\!](\rho \cup \{vs_j \mapsto vs_j \mid j \notin \lambda\}) H \iota \\ & & & \quad (\overline{\phi}[j \mapsto vs_j \mid j \notin \lambda = \mathsf{false}])(\lambda \cup \{j_1, \ldots, j_m\}) \\ & & \overset{\text{Def.}}{=} & [\![Q']\!]\rho' H' \iota' \overline{\phi'} \lambda' \end{array}$$

(vi) Let $\mathsf{message}(\xi, K, L) \in H$.

$$\begin{aligned}
&\overset{\text{Hyp.}}{\Rightarrow} && \mathsf{message}(\xi\sigma, K\sigma, L\sigma) \text{ is derivable from } \mathcal{C}_0\\
&\overset{vs_1,\ldots,vs_n \notin \text{fv}(\xi)\cup\text{fv}(K)\cup\text{fv}(L)}{\Rightarrow} && \mathsf{message}(\xi\sigma', K\sigma', L\sigma') \text{ is derivable from } \mathcal{C}_0
\end{aligned}$$

(vii)
$$\begin{aligned}
&\overset{\text{Hyp.}}{\Rightarrow} && \mathsf{attacker}(\overline{\phi}\sigma, attch[]) \in \mathcal{F}_0\\
&\overset{\phi\sigma\leq\psi}{\Rightarrow} && \mathsf{attacker}(\overline{\psi}, attch[]) \in \mathcal{F}_0\\
&\overset{\psi=\phi'\sigma'}{\Rightarrow} && \mathsf{attacker}(\overline{\phi'}\sigma', attch[]) \in \mathcal{F}_0
\end{aligned}$$

We can thus apply our induction hypothesis to infer that

$$\begin{aligned}
&\overset{\text{I.H}}{\Rightarrow} && (\rho'\sigma', \phi'\sigma', \iota'\sigma', \lambda') \vdash Q'\\
&\overset{\text{Def.}}{\Rightarrow} && (\rho\sigma \cup \{vs_j \mapsto \psi(j) \mid j \notin \lambda\}, \psi, \iota\sigma, \lambda \cup \{j_1, \ldots, j_m\}) \vdash Q'\\
&\overset{\text{Lem. 5}}{\Rightarrow} && (\rho\sigma, \psi, \iota\sigma, \lambda \cup \{j_1, \ldots, j_m\}) \vdash Q'\{vs_j \mapsto \psi(j) \mid j \notin \lambda\}\\
&\overset{vs_1,\ldots,vs_n \notin \text{fv}(Q')}{\Rightarrow} && (\rho\sigma, \psi, \iota\sigma, \lambda \cup \{j_1, \ldots, j_m\}) \vdash Q'
\end{aligned}$$

But then according to our typing system

$$\frac{\forall\psi\ (\phi\sigma \leq \psi \wedge \psi = \psi[j \mapsto \phi\sigma(j) \mid j \in \lambda])\ \Rightarrow\ (\rho\sigma, \psi, \iota\sigma, \lambda \cup \{j_1, \ldots, j_m\}) \vdash Q'}{(\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash \mathsf{lock}\ s_{j_1}, \ldots, s_{j_m}; Q'\ (= Q)}\ \tau_{\mathsf{lock}}$$

Case $Q = \mathsf{unlock}\ s_{j_1}, \ldots, s_{j_m}; Q'$**.** Let ψ such that $\phi\sigma \leq \psi$ and $\psi = \psi[j \mapsto \phi\sigma(j) \mid j \in \lambda]$. Let $\rho' = \rho \cup \{vs_j \mapsto vs_j \mid j \notin \lambda\}$, $\sigma' = \sigma \cup \{vs_j \mapsto \psi(j) \mid j \notin \lambda\}$, $\phi' = \phi[j \mapsto vs_j \mid j \notin \lambda]$, $H' = H$, $\iota' = \iota$, $\lambda' = \lambda \setminus \{j_1, \ldots, j_m\}$, for some $vs_1, \ldots, vs_n$ fresh. We will show that $(Q', \sigma', \rho', H', \iota', \phi', \lambda')$ satisfy conditions (i)-(vii).

(i)
$$\begin{aligned}
&\overset{\text{Def.}}{\Rightarrow} && \text{fn}(Q') \cup \text{fv}(Q') \cup \text{fn}(\iota') \cup \text{fv}(\iota') \cup \text{fn}(H') \cup \text{fv}(H') \cup\\
& && \quad \text{fn}(\phi') \cup \text{fv}(\phi') \subseteq \text{fn}(Q) \cup \text{fv}(Q) \cup \text{fn}(\iota) \cup \text{fv}(\iota) \cup \text{fn}(H) \cup \text{fv}(H) \cup\\
& && \quad\quad \text{fn}(\phi) \cup \text{fv}(\phi) \cup \{vs_j \mid j \notin \lambda\}\\
&\overset{\text{Hyp.}}{\Rightarrow} && \text{fn}(Q') \cup \text{fv}(Q') \cup \text{fn}(\iota') \cup \text{fv}(\iota') \cup \text{fn}(H') \cup \text{fv}(H') \cup\\
& && \quad \text{fn}(\phi') \cup \text{fv}(\phi') \subseteq \text{dom}(\rho) \cup \{vs_j \mid j \notin \lambda\}\\
&\overset{\rho'=\text{dom}(\rho)\cup\{vs_j\mapsto vs_j \mid j\notin\lambda\}}{\Rightarrow} && \text{fn}(Q') \cup \text{fv}(Q') \subseteq \text{dom}(\rho')
\end{aligned}$$

(ii)
$$\begin{aligned}
&\overset{\text{Def.}}{\Rightarrow} && \text{bn}(Q') \cup \text{bv}(Q') = \text{bn}(Q) \cup \text{bv}(Q)\\
&\overset{\text{Hyp.}}{\Rightarrow} && (\text{bn}(Q') \cup \text{bv}(Q')) \cap \text{dom}(\rho) = (\text{bn}(Q) \cup \text{bv}(Q)) \cap \text{dom}(\rho) = \emptyset\\
&\overset{vs_1,\ldots,vs_n \notin \text{bv}(Q')}{\Rightarrow} && (\text{bn}(Q') \cup \text{bv}(Q')) \cap \text{dom}(\rho') = \emptyset
\end{aligned}$$

(iii)
$$\begin{aligned}
&\overset{\text{Hyp.}}{\Rightarrow} && \psi(1),\ \ldots,\ \psi(n) \text{ are ground}\\
&\overset{\text{Def.}}{\Rightarrow} && \{vs_j \mapsto \psi(j) \mid j \notin \lambda\} \text{ is closed}\\
&\overset{\sigma\text{ closed}}{\Rightarrow} && \sigma \cup \{vs_j \mapsto \psi(j) \mid j \notin \lambda\} \text{ is closed}\\
&\overset{\sigma'=\sigma\cup\{vs_j\mapsto\psi(j) \mid k\notin\lambda\}}{\Rightarrow} && \sigma' \text{ is closed}
\end{aligned}$$

(iv) By definition Q' is not under a $\mathsf{lock}\ \ldots s \ldots$ if and only if either $s \in \{s_{j_1}, \ldots, s_{j_m}\}$ or Q is not under a $\mathsf{lock}\ \ldots s \ldots$, so condition (iii) is satisfied by construction of λ'.

$$\begin{aligned}
\textit{(v)}\quad \mathcal{C}_0 &\overset{\text{Hyp.}}{\supseteq} [\![\mathsf{unlock}\ s_{j_1},\dots,s_{j_m};Q']\!]\rho H\iota\overline{\phi}\lambda\\
&\overset{\text{Def.}}{=} [\![Q']\!](\rho\cup\{vs_j\mapsto vs_j \mid j\notin\lambda\})H\iota\\
&\qquad(\overline{\phi}[j\mapsto vs_j\mid j\notin\lambda])(\lambda\smallsetminus\{j_1,\dots,j_m\})\\
&\overset{\text{Def.}}{=} [\![Q']\!]\rho' H'\iota'\overline{\phi'}\lambda'
\end{aligned}$$

(vi) Let $\mathsf{message}(\xi,K,L)\in H$.

$$\begin{aligned}
&\overset{\text{Hyp.}}{\Rightarrow} && \mathsf{message}(\xi\sigma,K\sigma,L\sigma) \text{ is derivable from } \mathcal{C}_0\\
&\overset{vs_1,\dots,vs_n\notin \mathrm{fv}(\xi)\cup\mathrm{fv}(K)\cup\mathrm{fv}(L)}{\Rightarrow} && \mathsf{message}(\xi\sigma',K\sigma',L\sigma') \text{ is derivable from } \mathcal{C}_0
\end{aligned}$$

$$\begin{aligned}
\textit{(vii)}\quad &\overset{\text{Hyp.}}{\Rightarrow} && \mathsf{attacker}(\overline{\phi}\sigma,attch[])\in\mathcal{F}_0\\
&\overset{\phi\sigma\leq\psi}{\Rightarrow} && \mathsf{attacker}(\overline{\psi},attch[])\in\mathcal{F}_0\\
&\overset{\psi=\phi'\sigma'}{\Rightarrow} && \mathsf{attacker}(\overline{\phi'}\sigma',attch[])\in\mathcal{F}_0
\end{aligned}$$

We can thus apply our induction hypothesis to infer that

$$\begin{aligned}
&\overset{\text{I.H}}{\Rightarrow} && (\rho'\sigma',\phi'\sigma',\iota'\sigma',\lambda')\vdash Q'\\
&\overset{\text{Def.}}{\Rightarrow} && (\rho\sigma\cup\{vs_j\mapsto\psi()j\mid j\notin\lambda\},\psi,\iota\sigma,\lambda\smallsetminus\{j_1,\dots,j_m\})\vdash Q'\\
&\overset{\text{Lem. 5}}{\Rightarrow} && (\rho\sigma,\psi,\iota\sigma,\lambda\smallsetminus\{j_1,\dots,j_m\})\vdash Q'\{vs_j\mapsto\psi(j)\mid j\notin\lambda\}\\
&\overset{vs_1,\dots,vs_n\notin\mathrm{fv}(Q')}{\Rightarrow} && (\rho\sigma,\psi,\iota\sigma,\lambda\smallsetminus\{j_1,\dots,j_m\})\vdash Q'
\end{aligned}$$

But then according to our typing system

$$\frac{\forall\psi\ (\phi\sigma\leq\psi\wedge\psi=\psi[j\mapsto\phi\sigma(j)\mid j\in\lambda])\ \Rightarrow\ (\rho\sigma,\psi,\iota\sigma,\lambda\smallsetminus\{j_1,\dots,j_m\})\vdash Q'}{(\rho\sigma,\phi\sigma,\iota\sigma,\lambda)\vdash\mathsf{unlock}\ s_{j_1},\dots,s_{j_m};Q'\ (=Q)}\,\tau_{\mathsf{unlock}}$$

Case $\mathbf{Q} =$ $s_{j_1},\dots,s_{j_m} := K_1,\dots,K_m;Q'$ Let ψ such that $\phi\sigma\leq\psi$ and $\psi=\psi[j\mapsto\phi\sigma(j)\mid j\in\lambda]$. Let $\rho'=\rho\cup\{vs_j\mapsto vs_j\mid j\notin\lambda\}$, $\sigma'=\sigma\cup\{vs_j\mapsto\psi(j)\mid j\notin\lambda\}$, $\phi'=\phi[j\mapsto vs_j\mid j\notin\lambda]$, $H'=H$, $\iota'=\iota$, $\lambda'=\lambda$, and $\phi''=\phi'[j_k\mapsto\rho(M_k)\mid 1\leq k\leq m]$, for some $vs_1,\dots,vs_n$ fresh. We will show that $(Q',\sigma',\rho',H',\iota',\phi'',\lambda')$ satisfy conditions (i)-(vii). But first, we will show that $\phi'\sigma'=\psi\leq\psi[j_k\mapsto\rho\sigma(M_k)\mid 1\leq k\leq m]=\phi''\sigma'$. By definition of our translation

$$\begin{aligned}
&[\![s_{j_1},\dots,s_{j_m}:=K_1,\dots,K_m;Q']\!]\rho H\phi\iota\lambda=\\
&\qquad[\![Q']\!](\rho\cup\{vs_j\mapsto vs_j\mid j\notin\lambda\}H\iota\phi''\lambda\\
&\qquad\cup\{H\wedge\mathsf{message}(\overline{\phi'},wc,wm)\rightarrow\mathsf{message}\{\overline{\phi''},wc,wm\}\} && (=\mathcal{C}_1)\\
&\qquad\cup\{H\wedge\mathsf{attacker}(\overline{\phi'},wm)\rightarrow\mathsf{message}\{\overline{\phi''},wm\}\} && (=\mathcal{C}_2)
\end{aligned}$$

for some wc,wm fresh.

- $$\begin{aligned}
&\overset{\text{Hyp}}{\Rightarrow} && \mathsf{attacker}(\overline{\phi\sigma},attch[])\in\mathcal{F}_0\\
&\overset{\phi\sigma\leq\psi}{\Rightarrow} && \mathsf{attacker}(\overline{\psi},attch[])\in\mathcal{F}_0
\end{aligned}$$
- Let $\mathsf{message}(\overline{\psi},K',L')\in\mathcal{F}_0$.

$$\begin{aligned}
&\overset{\text{Hyp.}}{\Rightarrow} && H\sigma \text{ derivable from } \mathcal{C}_0\wedge\mathsf{message}(\overline{\psi},K',L')\in\mathcal{F}_0\\
&\overset{vs_1,\dots,vs_n\text{ fresh}}{\Rightarrow} && H\sigma' \text{ derivable from } \mathcal{C}_0\wedge\mathsf{message}(\overline{\psi},K',L')\in\mathcal{F}_0\\
&\overset{\psi=\phi'\sigma'}{\Rightarrow} && H\sigma' \text{ derivable from } \mathcal{C}_0\wedge\mathsf{message}(\overline{\phi'}\sigma',K',L')\in\mathcal{F}_0\\
&\overset{\mathcal{C}_1\in\mathcal{C}_0}{\Rightarrow} && \mathsf{message}(\overline{\phi''}\sigma',K',L')\in\mathcal{F}_0\\
&\overset{\psi[j_k\mapsto(\rho\sigma)(M_K)|1\leq k\leq m]=\phi''\sigma'}{\Rightarrow} && \mathsf{message}(\overline{\psi}[j_k\mapsto(\rho\sigma)(M_K)\mid 1\leq k\leq m],K',L')\in\mathcal{F}_0
\end{aligned}$$

- Let $\mathsf{attacker}(\overline{\psi}, K') \in \mathcal{F}_0$.

$$\begin{aligned}
&\overset{\text{Hyp.}}{\Rightarrow} && H\sigma \text{ derivable from } \mathcal{C}_0 \wedge \mathsf{attacker}(\overline{\psi}, K') \in \mathcal{F}_0 \\
&\overset{vs_1,\ldots,vs_n \text{ fresh}}{\Rightarrow} && H\sigma' \text{ derivable from } \mathcal{C}_0 \wedge \mathsf{attacker}(\overline{\psi}, K') \in \mathcal{F}_0 \\
&\overset{\psi=\phi'\sigma'}{\Rightarrow} && H\sigma' \text{ derivable from } \mathcal{C}_0 \wedge \mathsf{attacker}(\overline{\phi'}\sigma', K') \in \mathcal{F}_0 \\
&\overset{\mathcal{C}_2\in\mathcal{C}_0}{\Rightarrow} && \mathsf{attacker}(\overline{\phi''}\sigma', K') \in \mathcal{F}_0 \\
&\overset{\psi[j_k\mapsto(\rho\sigma)(M_K)|1\leq k\leq m]=\phi''\sigma'}{\Rightarrow} && \mathsf{attacker}(\overline{\psi}[j_k \mapsto (\rho\sigma)(M_K) \mid 1 \leq k \leq m], K') \in \mathcal{F}_0
\end{aligned}$$

$$\Rightarrow \ \phi'\sigma' = \psi \leq \psi[j_k \mapsto \rho\sigma(M_k) \mid 1 \leq k \leq m] = \phi''\sigma'.$$

We will now show that $(Q', \sigma', \rho', H', \iota, \phi'', \lambda')$ satisfy conditions (i)-(vii).

(i)

$$\begin{aligned}
&\overset{\text{Def.}}{\Rightarrow} && \mathrm{fv}(Q') \cup \mathrm{fn}(Q') \cup \mathrm{fv}(\iota') \cup \mathrm{fn}(\iota') \cup \mathrm{fv}(H') \cup \mathrm{fn}(H') \cup \\
& && \quad \mathrm{fv}(\phi') \cup \mathrm{fn}(\phi') \subseteq \mathrm{fv}(Q) \cup \mathrm{fn}(Q) \cup \mathrm{fv}(\iota) \cup \mathrm{fn}(\iota) \cup \mathrm{fv}(H) \cup \mathrm{fn}(H) \cup \\
& && \quad \mathrm{fv}(\phi) \cup \mathrm{fn}(\phi) \cup \{vs_j \mid j \notin \lambda\} \\
&\overset{\text{Hyp.}}{\Rightarrow} && \mathrm{fv}(Q') \cup \mathrm{fn}(Q') \cup \mathrm{fv}(\iota') \cup \mathrm{fn}(\iota') \cup \mathrm{fv}(H') \cup \mathrm{fn}(H') \cup \mathrm{fv}(\phi') \cup \\
& && \quad \mathrm{fn}(\phi') \subseteq \mathrm{dom}(\rho) \cup \{vs_j \mid j \notin \lambda\} \\
&\overset{\rho'=\rho\cup\cup\{vs_j\mapsto vs_j|j\notin\lambda\}}{\Rightarrow} && \mathrm{fv}(Q') \cup \mathrm{fn}(Q') \subseteq \mathrm{dom}(\rho')
\end{aligned}$$

(ii)

$$\begin{aligned}
&\overset{\text{Def.}}{\Rightarrow} && \mathrm{bn}(Q') \cup \mathrm{bv}(Q') = \mathrm{bn}(Q) \cup \mathrm{bv}(Q) \\
&\overset{\text{Hyp.}}{\Rightarrow} && (\mathrm{bn}(Q') \cup \mathrm{bv}(Q')) \cap \mathrm{dom}(\rho) = (\mathrm{bn}(Q) \cup \mathrm{bv}(Q)) \cap \mathrm{dom}(\rho) = \emptyset \\
&\overset{vs_1,\ldots,vs_m \text{ fresh}}{\Rightarrow} && (\mathrm{bn}(Q') \cup \mathrm{bv}(Q')) \cap \mathrm{dom}(\rho') = \emptyset
\end{aligned}$$

(iii)

$$\begin{aligned}
&\overset{\text{Hyp.}}{\Rightarrow} && \psi(1),\ \ldots,\ \psi(n) \text{ are ground} \\
&\overset{\text{Def.}}{\Rightarrow} && \{vs_j \mapsto \psi(j) \mid j \notin \lambda\} \text{ is closed} \\
&\overset{\sigma \text{ closed}}{\Rightarrow} && \sigma \cup \{vs_j \mapsto \psi(j) \mid j \notin \lambda\} \text{ is closed} \\
&\overset{\sigma'=\sigma\cup\{vs_j\mapsto\psi(j)|j\notin\lambda\}}{\Rightarrow} && \sigma' \text{ is closed}
\end{aligned}$$

(iv) By definition Q is under a $\mathsf{lock}\ \ldots s_i \ldots$ in P_0 if and only if Q' is under a $\mathsf{lock}\ \ldots s_i \ldots$, so condition (iii) is satisfied by hypothesis.

(v)

$$\begin{aligned}
\mathcal{C}_0 &\overset{\text{Hyp.}}{\supseteq} && [\![s_{j_1}, \ldots s_{j_m} := K_1, \ldots, K_m; Q']\!] \rho H \iota \overline{\phi} \lambda \\
&\overset{\text{Def.}}{\supseteq} && [\![Q']\!] \rho' H' \iota' \overline{\phi''} \lambda'
\end{aligned}$$

(vi) Let $\mathsf{message}(\xi, K', L') \in H' = H$. By hypothesis $\mathsf{message}(\xi\sigma, K'\sigma, L'\sigma)$ is derivable from $\mathcal{C}_0$, and because $v_1, \ldots, vs_n$ are fresh, *i.e.* not $\mathrm{infn}()\xi \cup \mathrm{fn}(K') \cup \mathrm{fn}(L')$, then $\mathsf{message}(\xi\sigma', K'\sigma', L'\sigma') = \mathsf{message}(\xi\sigma, K'\sigma, L'\sigma)$ is derivable from $\mathcal{C}_0$.

(vii)

$$\begin{aligned}
&\overset{\text{Hyp.}}{\Rightarrow} && \mathsf{attacker}(\overline{\phi}\sigma, attch[]) \in \mathcal{F}_0 \\
&\overset{\phi\sigma\leq\psi}{\Rightarrow} && \mathsf{attacker}(\overline{\phi'}\sigma, attch[]) \in \mathcal{F}_0 \\
&\overset{\psi\leq\psi'}{\Rightarrow} && \mathsf{attacker}(\overline{\phi'}\sigma, attch[]) \in \mathcal{F}_0 \\
&\overset{\psi'=\phi''\sigma'}{\Rightarrow} && \mathsf{attacker}(\overline{\phi''}\sigma', attch[]) \in \mathcal{F}_0
\end{aligned}$$

We can thus apply our inductive hypothesis to infer that

$$\begin{array}{cl}
\overset{\text{I.H}}{\Rightarrow} & (\rho'\sigma', \phi''\sigma', \iota'\sigma', \lambda') \vdash Q' \\
\overset{\substack{\rho'\sigma'=\rho\sigma\cup\{vs_j\mapsto vs_j|j\not\in\lambda\} \\ \phi''\sigma'=\psi[j_k\mapsto(\rho\sigma)(K_k)|1\leq k\leq m]}}{\Rightarrow} & (\rho\sigma \cup \{vs_j \mapsto vs_j \mid j \not\in \lambda\}, \\
& \quad \psi[j_k \mapsto (\rho\sigma)(K_k) \mid 1 \leq k \leq m], \iota'\sigma', \lambda') \vdash Q' \\
\overset{\text{Lem. 5}}{\Rightarrow} & (\rho\sigma, \psi[j_k \mapsto (\rho\sigma)(K_k) \mid 1 \leq k \leq m], \iota'\sigma', \lambda') \vdash \\
& \quad Q'\{vs_j \mapsto vs_j \mid j \not\in \lambda\} \\
\overset{vs_1,\ldots,vs_m\not\in\text{fv}(Q')\subseteq\text{dom}(\rho)}{\Rightarrow} & (\rho\sigma, \psi[j_k \mapsto (\rho\sigma)(K_k) \mid 1 \leq k \leq m], \iota'\sigma', \lambda') \vdash Q' \\
\overset{\iota'\sigma'=\iota\sigma\ \ \lambda'=\lambda}{\Rightarrow} & (\rho\sigma, \psi[j_k \mapsto (\rho\sigma)(K_k) \mid 1 \leq k \leq m], \iota\sigma, \lambda) \vdash Q'
\end{array}$$

But then according to our typing system

$$\frac{\begin{array}{c}\forall\psi\ (\phi\sigma \leq \psi\ \wedge\ \psi = \psi[j \mapsto \phi\sigma(j) \mid j \in \lambda]) \Rightarrow \\ (\psi \leq \psi[j_k \mapsto (\rho\sigma)(K_k) \mid 1 \leq k \leq m] \wedge \\ (\rho\sigma, \psi[j_k \mapsto (\rho\sigma)(K_k) \mid 1 \leq k \leq m], \iota\sigma, \lambda) \vdash Q')\end{array}}{(\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash s_{j_1}, \ldots, s_{j_m} := K_1, \ldots, K_m; Q'\ (= Q)}\ \tau_{write}$$

Case Q = $\mathsf{read}\ s_{j_1}, \ldots, s_{j_m}\ \mathsf{as}\ x_1, \ldots, x_m; Q'$**.** Let $\psi\}$ be a state such that $\phi\sigma \leq \psi$ and $\psi = \psi[j \mapsto \phi\sigma(j) \mid j \in \lambda]$. Let $\rho' = \rho \cup \{vc \mapsto vc, vm \mapsto vm\} \cup \{x_k \mapsto \phi'(j_k) \mid 1 \leq k \leq m\} \cup \{vs_j \mapsto vs_j \mid j \not\in \lambda\}$, $\sigma' = \sigma \cup \{vs_j \mapsto \psi(j) \mid j \not\in \lambda\} \cup \{vc \mapsto attch[], vm \mapsto attch[]\}$, $\iota' = x_1 :: \cdots :: x_m :: \iota$, $\phi' = \phi[j \mapsto vs_j \mid j \not\in \lambda]$, $H' = H \wedge \mathsf{message}(\overline{\phi'}, vc, vm)$, and $\lambda' = \lambda$, for $vc, vm, vs_1, \ldots, vs_n$ fresh. We show that $(Q', \sigma', \rho', H', \iota', \phi', \lambda')$ satisfy conditions (i)-(vii).

$$\begin{array}{lcl}
\textit{(i)} & \overset{\text{Def.}}{\Rightarrow} & \text{fv}(Q') \cup \text{fn}(Q') \cup \text{fv}(\iota') \cup \text{fn}(\iota') \cup \text{fv}(H') \cup \text{fn}(H') \cup \\
& & \quad \text{fv}(\phi') \cup \text{fn}(\phi') \subseteq \text{fv}(Q) \cup \{x_1, \ldots, x_m\} \cup \text{fn}(Q) \cup \text{fv}(\iota) \cup \\
& & \quad\quad \text{fn}(\iota) \cup \text{fv}(H) \cup \text{fn}(H) \cup \{vc, vm\} \cup \text{fv}(\phi) \cup \text{fn}(\phi)\{vs_j \mid j \not\in \lambda\} \\
& \overset{\text{Hyp.}}{\Rightarrow} & \text{fv}(Q') \cup \text{fn}(Q') \cup \text{fv}(\iota') \cup \text{fn}(\iota') \cup \text{fv}(H') \cup \\
& & \quad \text{fn}(H') \cup \text{fv}(\phi') \cup \text{fn}(\phi') \subseteq \text{dom}(\rho) \cup \{x_1, \ldots, x_m\} \cup \{vc, vm\} \\
& \overset{\text{dom}(\rho')=\text{dom}(\rho)\cup\{x_1,\ldots,x_m\}\cup\{vs_j|j\not\in\lambda\}}{\Rightarrow} & \text{fv}(Q') \cup \text{fn}(Q') \cup \text{fv}(\iota') \cup \text{fn}(\iota') \cup \text{fv}(H') \cup \text{fn}(H') \cup \\
& & \qquad\qquad\qquad \text{fv}(\phi') \cup \text{fn}(\phi') \subseteq \text{dom}(\rho')
\end{array}$$

$$\begin{array}{lcl}
\textit{(ii)} & \overset{\text{Def.}}{\Rightarrow} & \text{bn}(Q') \cup \text{bv}(Q') \subseteq \text{bn}(Q) \cup \text{bv}(Q) \\
& \overset{\text{Hyp.}}{\Rightarrow} & (\text{bn}(Q') \cup \text{bv}(Q')) \cap \text{dom}(\rho) \subseteq (\text{bn}(Q) \cup \text{bv}(Q)) \cap \text{dom}(\rho) = \emptyset \\
& \overset{\substack{x_1, \ldots, x_m \not\in \text{bv}(Q') \\ vc, vm, vs_1, \ldots, vs_n \text{ fresh}}}{\Rightarrow} & (\text{bn}(Q') \cup \text{bv}(Q')) \cap \text{dom}(\rho') = \emptyset
\end{array}$$

$$\begin{array}{lcl}
\textit{(iii)} & \overset{\text{Hyp.}}{\Rightarrow} & L,\ \psi(1),\ \ldots,\ \psi(n), attch[] \text{ are ground} \\
& \overset{\text{Def.}}{\Rightarrow} & \{vs_j \mapsto \psi(j) \mid j \not\in \lambda\} \cup \{vc \mapsto attch[], vm \mapsto attch[]\} \text{ is closed} \\
& \overset{\sigma\text{ closed}}{\Rightarrow} & \sigma \cup \{vs_j \mapsto \psi(j) \mid j \not\in \lambda\} \cup \{vc \mapsto attch[], vm \mapsto attch[]\} \text{ is closed} \\
& \overset{\substack{\sigma' = \sigma \cup \{vs_j \mapsto \psi(j) \mid j \not\in \lambda\} \cup \\ \{vc \mapsto attch[], vm \mapsto attch[]\}}}{\Rightarrow} & \sigma' \text{ is closed}
\end{array}$$

(iv) By definition Q is under a $\mathsf{lock}\ \ldots s_i \ldots$ in P_0 if and only if Q' is under a $\mathsf{lock}\ \ldots s_i \ldots$, so condition (iii) is satisfied by hypothesis.

$$\begin{aligned}
\textit{(v)}\quad \mathcal{C}_0 &\overset{\text{Hyp.}}{\supseteq} \llbracket \mathsf{read}\ s_{j_1}, \ldots, s_{j_m}\ \mathsf{as}\ x_1, \ldots, x_m; Q' \rrbracket \rho H \iota \overline{\phi} \lambda \\
&\overset{\text{Hyp.}}{=} \llbracket Q' \rrbracket (\rho \cup \{x_j \mapsto \phi'(j_k) \mid 1 \leq k \leq m\} \cup \{vs_j \mapsto vs_j \mid j \notin \lambda\}) \\
&\qquad\qquad (H \wedge \mathsf{message}(\overline{\phi'}, vc, vm))(x_1 :: \cdots :: x_m :: \iota)\overline{\phi'}\lambda \\
&\overset{\text{Def.}}{=} \llbracket Q' \rrbracket \rho' H' \iota' \overline{\phi'} \lambda'
\end{aligned}$$

(vi) Let $\mathsf{message}(\xi', K', L') \in H'$. Then either $\mathsf{message}(\xi', K', L') \in H$ or $\mathsf{message}(\xi', K', L') = \mathsf{message}(\overline{\phi'}, vc, vm)$.

If $\mathsf{message}(\xi', K', L') \in H$**.**

$$\begin{aligned}
&\overset{\text{Hyp.}}{\Rightarrow} && \mathsf{message}(\xi'\sigma, K'\sigma, L'\sigma) \text{ is derivable from } \mathcal{C}_0 \\
&\overset{\substack{x_1, \ldots, x_m \notin \mathrm{fn}(\xi') \cup \mathrm{fn}(K') \cup \mathrm{fn}(L') \\ vc, vm, vs_1, \ldots, vs_n \text{ fresh}}}{\Rightarrow} && \mathsf{message}(\xi'\sigma', K'\sigma', L'\sigma') \text{ is derivable from } \mathcal{C}_0
\end{aligned}$$

If $\mathsf{message}(\xi', K', L') = \mathsf{message}(\overline{\phi'}, vc, vm)$**.**

$$\begin{aligned}
&\overset{\text{Hyp.}}{\Rightarrow} && \mathsf{attacker}(\overline{\phi}\sigma, attch[]) \text{ is derivable from } \mathcal{C}_0 \\
&\overset{\text{Def. of } \mathcal{C}_0}{\Rightarrow} && \mathsf{message}(\overline{\phi}\sigma, attch[], attch[]) \text{ is derivable from } \mathcal{C}_0 \\
&\overset{\phi\sigma \leq \psi}{\Rightarrow} && \mathsf{message}(\overline{\psi}, attch[], attch[]) \text{ is derivable from } \mathcal{C}_0 \\
&\overset{\psi = \phi'\sigma'}{\Rightarrow} && \mathsf{message}(\overline{\phi'}\sigma', attch[], attch[]) \text{ is derivable from } \mathcal{C}_0 \\
&\overset{\sigma'(vc) = attch[] \wedge \sigma'(vm) = attch[]}{\Rightarrow} && \mathsf{message}(\overline{\psi}, vc\sigma', vm\sigma') \text{ is derivable from } \mathcal{C}_0 \\
&\overset{\mathsf{message}(\xi', K', L') = \mathsf{message}(\overline{\phi'}, vc, vm)}{\Rightarrow} && \mathsf{message}(\xi'\sigma', K'\sigma', L'\sigma') \text{ is derivable from } \mathcal{C}_0
\end{aligned}$$

$$\begin{aligned}
\textit{(vii)}\quad &\overset{\text{Hyp.}}{\Rightarrow} && \mathsf{attacker}(\overline{\phi}\sigma, attch[]) \in \mathcal{F}_0 \\
&\overset{\phi\sigma \leq \psi}{\Rightarrow} && \mathsf{attacker}(\overline{\psi}, attch[]) \in \mathcal{F}_0 \\
&\overset{\psi = \phi'\sigma'}{\Rightarrow} && \mathsf{attacker}(\overline{\phi'}\sigma', attch[]) \in \mathcal{F}_0
\end{aligned}$$

We can thus apply our induction hypothesis to infer that

$$\begin{aligned}
&\overset{\text{I.H.}}{\Rightarrow} && (\rho'\sigma', \phi'\sigma', \iota'\sigma', \lambda') \vdash Q' \\
&\overset{\substack{\{x_1, \ldots, x_m\} \cap \mathrm{dom}(\rho) = \emptyset \\ vc, vm, vs_1, \ldots, vs_n \text{ fresh} \\ \text{Lem. 5.2}}}{\Rightarrow} && (\rho\sigma \cup \{x_k \mapsto \phi'\sigma'(j_k) \mid 1 \leq k \leq m\}, \phi'\sigma', \iota'\sigma', \lambda') \vdash Q' \\
&\overset{\psi = \phi'\sigma'}{\Rightarrow} && (\rho\sigma \cup \{x_k \mapsto \psi(j_k) \mid 1 \leq k \leq m\}, \psi, \iota'\sigma', \lambda') \vdash Q' \\
&\overset{\iota'\sigma' = \psi(j_1) :: \cdots :: \psi(j_m) :: (\iota\sigma)\ \ \lambda' = \lambda}{\Rightarrow} && (\rho\sigma \cup \{x_k \mapsto \psi(j_k) \mid 1 \leq k \leq m\}, \\
&&& \quad \psi, \psi(j_1) :: \cdots :: \psi(j_m) :: (\iota\sigma), \lambda) \vdash Q'
\end{aligned}$$

Thus according to our typing system

$$\frac{\begin{array}{l}\forall \psi\ (\phi\sigma \leq \psi\ \wedge\ \psi = \psi[j \mapsto \phi\sigma(j) \mid j \in \lambda]) \Rightarrow \\ \qquad (\rho\sigma \cup \{x_k \mapsto \psi(j_k) \mid 1 \leq k \leq m\}, \psi, (\psi(j_1) :: \cdots :: \psi(j_m) :: (\iota\sigma)), \lambda) \vdash Q\end{array}}{(\rho\sigma, \phi\sigma, \iota, \lambda) \vdash \mathsf{read}\ s_{j_1}, \ldots, s_{j_m}\ \mathsf{as}\ x_1, \ldots, x_m; Q}\ \tau_{read}$$

To conclude the proof of Lemma 2 we then need to show that $\rho = \mathcal{E}_0$, σ *s.t.* $\mathrm{dom}(\sigma) = \emptyset$, $H = \mathsf{true}$, $\iota = []$, $\phi = \mathcal{E}_0(\mathcal{S}_0)$ and $\emptyset$ satisfy conditions (i)-(vii).

(i) Since by hypotheses $\mathrm{fv}(P_0') = \emptyset$ and $\mathrm{fn}(P_0') \subseteq \mathrm{dom}(\mathcal{E}_0)$ by construction, ρ binds the free names and variables of P_0, ι, H and ϕ.

(ii) By construction, $\mathrm{dom}(\mathcal{E}_0) = \mathrm{fn}(P_0') \cup \mathrm{cells}(P_0') \cup \{attch\}$, and by hypothesis $\mathrm{bn}(P_0') \cap \mathrm{fn}(P_0') = \emptyset$. Thus $(\mathrm{bn}(P_0) \cup \mathrm{bv}(P_0)) \cap \mathrm{dom}(\mathcal{E}_0) = \emptyset$.

(iii) By definition σ is a closed substitution.

(iv) P_0 is not under any lock in P_0, thus $\emptyset$ satisfies condition (iii).

(v) By definition $\mathcal{C}_0 \supseteq [\![P_0]\!]\rho H \iota \overline{\phi} \lambda$.

(vi) by definition $H\sigma = \mathsf{true}$, and thus $H\sigma$ can trivially be derived from $\mathcal{C}_0$.

(vii) By construction, $\mathsf{attacker}(\overline{\mathcal{E}_0(\mathcal{S}_0)}, attch[]) \in \mathcal{C}_0$. So in particular, we have that $\mathsf{attacker}(\overline{\phi\sigma}, attch[]) \in \mathcal{F}_0$.

Thus, P_0, ρ, σ, H, ι, ϕ and $\emptyset$ satisfy the conditions of our induction result according to which $(\mathcal{E}_0, \mathcal{E}_0(\mathcal{S}_0), [], \emptyset) \vdash P_0$.

□

B.4 Proof of Lemma 3: Subject Reduction

Lemma 3 (Subject reduction)**.** *Let $(\mathcal{E}, \mathcal{S}, \mathcal{Q}) \to (\mathcal{F}, \mathcal{T}, \mathcal{R})$ be a valid instrumented transition such that no $[s \mapsto M]$ occurs in $\mathcal{Q}$, names and variables are bound at most once in $\mathcal{Q}$, and $cells(Q) \subseteq \{s_1, \ldots, s_n\}$. If $(\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\imath), \lambda) \vdash Q$ for all $(Q, \imath, \lambda) \in \mathcal{Q}$, then $(\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\jmath), \nu) \vdash R$ for all $(R, \jmath, \nu) \in \mathcal{R}$.*

Proof. We prove this by case analysis on the rule R that fired the transition $(\mathcal{E}, \mathcal{S}, \mathcal{Q}) \to (\mathcal{F}, \mathcal{T}, \mathcal{R})$.

Case R = Red Nil. In this case, $\mathcal{F} = \mathcal{E}$, $\mathcal{T} = \mathcal{S}$, $\mathcal{Q} = \mathcal{Q}' \cup \{(0, \imath, \lambda)\}$, and $\mathcal{R} = \mathcal{Q}'$. Let $(R, \jmath, \nu) \in \mathcal{R}$.

$$\begin{aligned} &\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\jmath), \nu) \vdash R \\ &\overset{\mathcal{F}=\mathcal{E}\ \mathcal{T}=\mathcal{S}}{\Rightarrow} && (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\jmath), \nu) \vdash R \end{aligned}$$

Case R = Red Repl. In this case, $\mathcal{F} = \mathcal{E}$, $\mathcal{T} = \mathcal{S}$, $\mathcal{Q} = \mathcal{Q}' \cup \{(!Q, \imath, \lambda)\}$, and $\mathcal{R} = \mathcal{Q} \cup \{(Q, \imath, \lambda)\}$. Let $(R, \jmath, \nu) \in \mathcal{R}$, then either $(R, \jmath, \nu) \in \mathcal{Q}'$ or $(R, \jmath, \nu) = (Q, \imath, \lambda)$.

If $(R, \jmath, \nu) \in \mathcal{Q}$**.**

$$\begin{aligned} &\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\jmath), \nu) \vdash R \\ &\overset{\mathcal{F}=\mathcal{E}\ \mathcal{T}=\mathcal{S}}{\Rightarrow} && (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\jmath), \nu) \vdash R \end{aligned}$$

If $(Q, \jmath, \nu) = (R, \imath, \lambda)$**.**

$$\begin{aligned} &\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\imath), \lambda) \vdash !Q \\ &\overset{\mathcal{T}_{repl}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\imath), \lambda) \vdash Q \\ &\overset{\mathcal{F}=\mathcal{E}\ \ \mathcal{T}=\mathcal{S}\ \ \jmath=\imath\ \ \nu=\lambda\ \ R=Q}{\Rightarrow} && (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\jmath), \nu) \vdash R \end{aligned}$$

Case R = Red Par. In this case, $\mathcal{F} = \mathcal{E}$, $\mathcal{T} = \mathcal{S}$, $\mathcal{Q} = \mathcal{Q}' \cup \{(Q_1 \mid Q_2, \imath, \lambda)\}$, and $\mathcal{R} = \mathcal{Q}' \cup \{(Q_1, \imath, \lambda), (Q_2, \imath, \lambda)\}$. Let $(R, \jmath, \nu) \in \mathcal{R}$, then either $(R, \jmath, \nu) \in \mathcal{Q}'$ or $(R, \jmath, \nu) \in \{(Q_1, \imath, \lambda), (Q_2, \imath, \lambda)\}$.

If $(R, \jmath, \nu) \in \mathcal{Q}'$**.**

$$\begin{aligned} &\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\jmath), \nu) \vdash R \\ &\overset{\mathcal{F}=\mathcal{E}\ \mathcal{T}=\mathcal{S}}{\Rightarrow} && (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\jmath), \nu) \vdash R \end{aligned}$$

If $(R,\jmath,\nu) = (Q_1,\imath,\lambda)$**.**

$$\begin{array}{rl}
\overset{\text{Hyp.}}{\Rightarrow} & (\mathcal{E},\mathcal{E}(\mathcal{S}),\mathcal{E}(\imath),\lambda) \vdash Q_1 \mid Q_2 \\
\overset{\tau_{par}}{\Rightarrow} & (\mathcal{E},\mathcal{E}(\mathcal{S}),\mathcal{E}(\imath),\lambda) \vdash Q_1 \\
\overset{\mathcal{F}=\mathcal{E}\ \mathcal{T}=\mathcal{S}\ \jmath=\imath\ \nu=\lambda\ R=Q_1}{\Rightarrow} & (\mathcal{F},\mathcal{F}(\mathcal{T}),\mathcal{F}(\jmath),\nu) \vdash R
\end{array}$$

If $(R,\jmath,\nu) = (Q_2,\imath,\lambda)$**.**

$$\begin{array}{rl}
\overset{\text{Hyp.}}{\Rightarrow} & (\mathcal{E},\mathcal{E}(\mathcal{S}),\mathcal{E}(\imath),\lambda) \vdash Q_1 \mid Q_2 \\
\overset{\tau_{par}}{\Rightarrow} & (\mathcal{E},\mathcal{E}(\mathcal{S}),\mathcal{E}(\imath),\lambda) \vdash Q_2 \\
\overset{\mathcal{F}=\mathcal{E}\ \mathcal{T}=\mathcal{S}\ \jmath=\imath\ \nu=\lambda\ R=Q_2}{\Rightarrow} & (\mathcal{F},\mathcal{F}(\mathcal{T}),\mathcal{F}(\jmath),\nu) \vdash R
\end{array}$$

Case R = Red New 1. In this case, $\mathcal{F} = \mathcal{E} \cup \{a' \mapsto a[\mathcal{E}(\imath)]\}$ where a' is fresh, $a \in \mathrm{bn}(P_0')$, $\mathcal{T} = \mathcal{S}$, $\mathcal{Q} = \mathcal{Q}' \cup \{(\mathsf{new}\ a; Q, \imath, \lambda)\}$, and $\mathcal{R} = \mathcal{Q}' \cup \{(Q\{a'/a\},\imath,\lambda)\}$. Let $(R,\jmath,\nu) \in \mathcal{R}$, then either $(R,\jmath,\nu) \in \mathcal{Q}'$ or $(R,\jmath,\nu) = (Q\{a'/a\},\imath,\lambda)$.

If $(R,\jmath,\nu) \in \mathcal{Q}'$**.**

$$\begin{array}{rl}
\overset{\text{Hyp.}}{\Rightarrow} & (\mathcal{E},\mathcal{E}(\mathcal{S}),\mathcal{E}(\jmath),\nu) \vdash R \\
\overset{a'\text{ fresh}}{\Rightarrow} & (\mathcal{E},\mathcal{F}(\mathcal{S}),\mathcal{F}(\jmath),\nu) \vdash R \\
\overset{a\notin\mathrm{fn}(R)}{\Rightarrow} & (\mathcal{E},\mathcal{F}(\mathcal{S}),\mathcal{F}(\jmath),\nu) \vdash R\{a'/a\} \\
\overset{\text{Lem. 5.2}}{\Rightarrow} & (\mathcal{F},\mathcal{F}(\mathcal{S}),\mathcal{F}(\jmath),\nu) \vdash R
\end{array}$$

If $(R,\jmath,\nu) = (Q\{a'/a\},\imath,\lambda)$**.**

$$\begin{array}{rl}
\overset{\text{Hyp.}}{\Rightarrow} & (\mathcal{E},\mathcal{E}(\mathcal{S}),\mathcal{E}(\imath),\lambda) \vdash \mathsf{new}\ a; Q' \\
\overset{\tau_{newP}}{\Rightarrow} & (\mathcal{E} \cup \{a \mapsto a[\mathcal{E}(\imath)]\},\mathcal{E}(\mathcal{S}),\mathcal{E}(\imath),\lambda) \vdash Q' \\
\overset{a'\text{ fresh}}{\Rightarrow} & (\mathcal{E} \cup \{a \mapsto a[\mathcal{E}(\imath)]\},\mathcal{F}(\mathcal{S}),\mathcal{F}(\imath),\lambda) \vdash Q' \\
\overset{a'\text{ fresh, }\alpha\text{-renaming}}{\Rightarrow} & (\mathcal{E} \cup \{a' \mapsto a[\mathcal{E}(\imath)]\},\mathcal{F}(\mathcal{S}),\mathcal{F}(\imath),\lambda) \vdash Q'\{a'/a\} \\
\overset{\mathcal{F}=\mathcal{E}\cup\{a'\mapsto a[\mathcal{E}(\imath)]\}}{\Rightarrow} & (\mathcal{F},\mathcal{F}(\mathcal{S}),\mathcal{F}(\imath),\lambda) \vdash Q'\{a'/a\} \\
\overset{\mathcal{T}=\mathcal{S}\ \jmath=\imath\ \nu=\lambda\ R=Q'\{a'/a\}}{\Rightarrow} & (\mathcal{F},\mathcal{F}(\mathcal{T}),\mathcal{F}(\jmath),\nu) \vdash Q'\{a'/a\}
\end{array}$$

Case R = Red New 2. In this case, $\mathcal{F} = \mathcal{E} \cup \{a' \mapsto attn[]\}$ where a' fresh, $a \notin \mathrm{bn}(P_0')$, $\mathcal{T} = \mathcal{S}$, $\mathcal{Q} = \mathcal{Q}' \cup \{(\mathsf{new}\ a; Q,\imath,\lambda)\}$, and $\mathcal{R} = \mathcal{Q}' \cup \{(Q\{a'/a\},\imath,\lambda)\}$. Let $(R,\jmath,\nu) \in \mathcal{R}$, then either $(R,\jmath,\nu) \in \mathcal{Q}'$ or $(R,\jmath,\nu) = (Q\{a'/a\},\imath,\lambda)$.

If $(R,\jmath,\nu) \in \mathcal{Q}'$**.**

$$\begin{array}{rl}
\overset{\text{Hyp.}}{\Rightarrow} & (\mathcal{E},\mathcal{E}(\mathcal{S}),\mathcal{E}(\jmath),\nu) \vdash R \\
\overset{a'\text{ fresh}}{\Rightarrow} & (\mathcal{E},\mathcal{F}(\mathcal{S}),\mathcal{F}(\jmath),\nu) \vdash R \\
\overset{a\notin\mathrm{fn}(R)}{\Rightarrow} & (\mathcal{E},\mathcal{F}(\mathcal{S}),\mathcal{F}(\jmath),\nu) \vdash R\{a'/a\} \\
\overset{\text{Lem. 5.2}}{\Rightarrow} & (\mathcal{F},\mathcal{F}(\mathcal{S}),\mathcal{F}(\jmath),\nu) \vdash R
\end{array}$$

If $(R,\jmath,\nu) = (Q\{a'/a\},\imath,\lambda)$**.**

$$\begin{array}{rl}
\overset{\text{Hyp.}}{\Rightarrow} & (\mathcal{E},\mathcal{E}(\mathcal{S}),\mathcal{E}(\imath),\lambda) \vdash \mathsf{new}\ a; Q' \\
\overset{\tau_{newP}}{\Rightarrow} & (\mathcal{E} \cup \{a \mapsto attn[]\},\mathcal{E}(\mathcal{S}),\mathcal{E}(\imath),\lambda) \vdash Q' \\
\overset{a'\text{ fresh}}{\Rightarrow} & (\mathcal{E} \cup \{a \mapsto attn[]\},\mathcal{F}(\mathcal{S}),\mathcal{F}(\imath),\lambda) \vdash Q' \\
\overset{a'\text{ fresh, }\alpha\text{-renaming}}{\Rightarrow} & (\mathcal{E} \cup \{a' \mapsto attn[]\},\mathcal{F}(\mathcal{S}),\mathcal{F}(\imath),\lambda) \vdash Q'\{a'/a\} \\
\overset{\mathcal{F}=\mathcal{E}\cup\{a'\mapsto attn[]\}}{\Rightarrow} & (\mathcal{F},\mathcal{F}(\mathcal{S}),\mathcal{F}(\imath),\lambda) \vdash Q'\{a'/a\} \\
\overset{\mathcal{T}=\mathcal{S}\ \jmath=\imath\ \nu=\lambda\ R=Q'\{a'/a\}}{\Rightarrow} & (\mathcal{F},\mathcal{F}(\mathcal{T}),\mathcal{F}(\jmath),\nu) \vdash Q'\{a'/a\}
\end{array}$$

Case R = Red Destr 1. In this case, $\mathcal{F} = \mathcal{E}$, $\mathcal{T} = \mathcal{S}$, $g(M_1,\ldots,M_n) \to M$, $\mathcal{Q} = \mathcal{Q}' \cup \{(\text{let } x = g(M_1,\ldots,M_n) \text{ in } Q_1 \text{ else } Q_2,\imath,\lambda)\}$, and $\mathcal{R} = \mathcal{Q}' \cup$

$\{(Q_1\{M/x\}, \imath, \lambda)\}$. Let $(R, \jmath, \nu) \in \mathcal{R}$, then either $(R, \jmath, \nu) \in \mathcal{Q}'$ or $(R, \jmath, \nu) = (Q_1\{M/x\}, \imath, \lambda)$.

If $(R, \jmath, \nu) \in \mathcal{Q}'$. First note that $g(M_1, \ldots, M_n) \to M$, implies that $g(\mathcal{E}(M_1), \ldots, \mathcal{E}(M_k)) \to \mathcal{E}(M)$ because the M_is only contain variables and constructors.

$$\begin{aligned} &\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\jmath), \nu) \vdash R \\ &\overset{\mathcal{F}=\mathcal{E}\ \mathcal{T}=\mathcal{S}}{\Rightarrow} && (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\jmath), \nu) \vdash R \end{aligned}$$

If $(R, \jmath, \nu) = (Q_1\{M/x\}, \imath, \lambda)$. Note first that $g(\mathcal{E}(M_1), \ldots, \mathcal{E}(M_n)) \to \mathcal{E}(M)$ because the M_is only contain variables and constructors.

$$\begin{aligned} &\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\imath), \lambda) \vdash \text{let } x = g(M_1, \ldots, M_n) \text{ in } Q_1 \text{ else } Q_2 \\ &\overset{\mathcal{T}_{let}}{\Rightarrow} && (\mathcal{E} \cup \{x \mapsto \mathcal{E}(M)\}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\imath), \lambda) \vdash Q_1 \\ &\overset{\text{Lem. 5.2}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\imath), \lambda) \vdash Q_1\{M/x\} \\ &\overset{\mathcal{F}=\mathcal{E}\ \mathcal{T}=\mathcal{S}\ \jmath=\imath\ \nu=\lambda\ R=Q_1\{M/x\}}{\Rightarrow} && (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\jmath), \nu) \vdash R \end{aligned}$$

Case R = Red Destr 2. In this case, $\mathcal{F} = \mathcal{E}$, $\mathcal{T} = \mathcal{S}$, for all M, $g(M_1, \ldots, M_n) \not\to M$, $\mathcal{Q} = \mathcal{Q}' \cup \{(\text{let } x = g(M_1, \ldots, M_n) \text{ in } Q_1 \text{ else } Q_2, \imath, \lambda)\}$, and $\mathcal{R} = \mathcal{Q}' \cup \{(Q_2, \imath, \lambda)\}$. Let $(R, \jmath, \nu) \in \mathcal{R}$, then either $(R, \jmath, \nu) \in \mathcal{Q}'$ or $(R, \jmath, \nu) = (Q_2, \imath, \lambda)$.

If $(R, \jmath, \nu) \in \mathcal{Q}'$.

$$\begin{aligned} &\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\jmath), \nu) \vdash R \\ &\overset{\mathcal{F}=\mathcal{E}\ \mathcal{T}=\mathcal{S}}{\Rightarrow} && (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\jmath), \nu) \vdash R \end{aligned}$$

If $(R, \jmath, \nu) = (Q_2, \imath, \lambda)$.

$$\begin{aligned} &\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\imath), \lambda) \vdash \text{let } x = g(M_1, \ldots, M_n) \text{ in } Q_1 \text{ else } Q_2 \\ &\overset{\mathcal{T}_{let}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\imath), \lambda) \vdash Q_2 \\ &\overset{\mathcal{F}=\mathcal{E}\ \mathcal{T}=\mathcal{S}\ \jmath=\imath\ \nu=\lambda\ R=Q_2}{\Rightarrow} && (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\jmath), \nu) \vdash R \end{aligned}$$

Case R = Red Cond 1. In this case, $\mathcal{F} = \mathcal{E}$, $\mathcal{T} = \mathcal{S}$, $\mathcal{Q} = \mathcal{Q}' \cup \{(\text{if } M = M \text{ then } Q_1 \text{ else } Q_2, \imath, \lambda)\}$, and $\mathcal{R} = \mathcal{Q}' \cup \{(Q_1, \imath, \lambda)\}$. Let $(R, \jmath, \nu) \in \mathcal{R}$, then either $(R, \jmath, \nu) \in \mathcal{Q}'$ or $(R, \jmath, \nu) = (Q_1, \imath, \lambda)$.

If $(R, \jmath, \nu) \in \mathcal{Q}'$.

$$\begin{aligned} &\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\jmath), \nu) \vdash R \\ &\overset{\mathcal{F}=\mathcal{E}\ \mathcal{T}=\mathcal{S}}{\Rightarrow} && (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\jmath), \nu) \vdash R \end{aligned}$$

If $(R, \jmath, \nu) = (Q_1, \imath, \lambda)$.

$$\begin{aligned} &\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\imath), \lambda) \vdash \text{if } M = M \text{ then } Q_1 \text{ else } Q_2 \\ &\overset{\mathcal{T}_{if}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\imath), \lambda) \vdash Q_1 \\ &\overset{\mathcal{F}=\mathcal{E}\ \mathcal{T}=\mathcal{S}\ \jmath=\imath\ \nu=\lambda\ R=Q_1}{\Rightarrow} && (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\jmath), \nu) \vdash R \end{aligned}$$

Case R = Red Cond 2. In this case, $\mathcal{F} = \mathcal{E}$, $\mathcal{T} = \mathcal{S}$, $\mathcal{Q} = \mathcal{Q}' \cup \{(\text{if } M = N \text{ then } Q_1 \text{ else } Q_2, \imath, \lambda)\}$, and $\mathcal{R} = \mathcal{Q}' \cup \{(Q_2, \imath, \lambda)\}$. Let $(R, \jmath, \nu) \in \mathcal{R}$, then either $(R, \jmath, \nu) \in \mathcal{Q}'$ or $(R, \jmath, \nu) = (Q_2, \imath, \lambda)$.

If $(R, \jmath, \nu) \in \mathcal{Q}'$.

$$\begin{aligned} &\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\jmath), \nu) \vdash R \\ &\overset{\mathcal{F}=\mathcal{E}\ \mathcal{T}=\mathcal{S}}{\Rightarrow} && (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\jmath), \nu) \vdash R \end{aligned}$$

If $(R, \jmath, \nu) = (Q_2, \imath, \lambda)$.

$$\begin{aligned} &\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\imath), \lambda) \vdash \text{if } M = N \text{ then } Q_1 \text{ else } Q_2 \\ &\overset{\mathcal{T}_{if}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\imath), \lambda) \vdash Q_2 \\ &\overset{\mathcal{F}=\mathcal{E}\ \mathcal{T}=\mathcal{S}\ \jmath=\imath\ \nu=\lambda\ R=Q_2}{\Rightarrow} && (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\jmath), \nu) \vdash R \end{aligned}$$

Case R = Red I/O. In this case, $\mathcal{F} = \mathcal{E}$, $\mathcal{T} = \mathcal{S}$, $\mathcal{Q} = \mathcal{Q}' \cup \{(\mathsf{out}(M, N); Q_1, \imath_1, \lambda_1), (\mathsf{in}(M, x); Q_2, \imath_2, \lambda_2)\}$, and

$\mathcal{R} = \mathcal{Q}' \cup \{(Q_1, \imath_1, \lambda_1), (Q_2\{N/x\}, (N :: \imath_2), \lambda_2)\}$.
Let $(R, \jmath, \nu) \in \mathcal{R}$, then either $(R, \jmath, \nu) \in \mathcal{Q}'$ or
$(R, \jmath, \nu) \in \{(Q_1, \imath_1, \lambda_1), (Q_2\{N/x\}, (N :: \imath_2), \lambda_2)\}$.

If $(R, \jmath, \nu) \in \mathcal{Q}'$.

$$\begin{aligned} &\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\jmath), \nu) \vdash R \\ &\overset{\mathcal{F}=\mathcal{E}\ \mathcal{T}=\mathcal{S}}{\Rightarrow} && (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\jmath), \nu) \vdash R \end{aligned}$$

If $(R, \jmath, \nu) = (Q_1, \imath_1, \lambda_1)$.

$$\begin{aligned} &\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\imath_1), \lambda_1) \vdash \mathsf{out}(M, N); Q_1 \\ &\overset{\tau_{out}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\imath_1), \lambda_1) \vdash Q_1 \\ &\overset{\mathcal{F}=\mathcal{E}\ \mathcal{T}=\mathcal{S}\ \jmath=\imath_1\ \nu=\lambda_1\ R=Q_1}{\Rightarrow} && (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\jmath), \nu) \vdash R \end{aligned}$$

If $(R, \jmath, \nu) = (Q_2\{N/x\}, (N :: \imath_2), \lambda_2)$. First note that, by hypothesis we have that $(\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\imath_1), \lambda_1) \vdash \mathsf{out}(M, N); Q_1$, but then according to the typing rule τ_{out}, $\mathsf{message}(\overline{\mathcal{E}(\mathcal{S})}, \mathcal{E}(M), \mathcal{E}(N)) \in \mathcal{F}_0$. Moreover

$$\begin{aligned} &\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\imath_2), \lambda_2) \vdash \mathsf{in}(M, x); Q_2 \\ &\overset{\tau_{in}}{\Rightarrow} && (\mathcal{E} \cup \{x \mapsto \mathcal{E}(N)\}, \mathcal{E}(\mathcal{S}), (\mathcal{E}(N) :: \mathcal{E}(\imath_2)), \lambda_2) \vdash Q_2 \\ &\overset{\text{Lem. 5.2}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), (\mathcal{E}(N) :: \mathcal{E}(\imath_2)), \lambda_2) \vdash P_2\{N/x\} \\ &\overset{\mathcal{E}(N)::\mathcal{E}(\imath_2)=\mathcal{E}(N::\imath_2)}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(N :: \imath_2), \lambda_2) \vdash Q_2\{N/x\} \\ &\overset{\mathcal{F}=\mathcal{E}\ \mathcal{T}=\mathcal{S}\ \jmath=N::\imath_2\ \nu=\lambda_2\ R=Q_2\{N/x\}}{\Rightarrow} && (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\jmath), \nu) \vdash R \end{aligned}$$

Case R = Red Init State. This case cannot occur because by hypothesis no $[s \mapsto M]$ occurs in $\mathcal{Q}$.

Case R = Red Lock. In this case,
$\mathcal{F} = \mathcal{E}$, $\mathcal{T} = \mathcal{S}$, $\mathcal{Q} = \mathcal{Q}' \cup \{(\mathsf{lock}\ s_{j_1}, \ldots, s_{j_m}; Q, \imath, \lambda)\}$, and $\mathcal{R} = \mathcal{Q}' \cup \{(Q, \imath, \lambda \cup \{j_1, \ldots, j_m\})\}$. Let $(R, \jmath, \nu) \in \mathcal{R}$, then either $(R, \jmath, \nu) \in \mathcal{Q}'$ or $(R, \jmath, \nu) = \{(Q, \imath, \lambda \cup \{j_1, \ldots, j_m\})\}$.

If $(R, \jmath, \nu) \in \mathcal{Q}'$.

$$\begin{aligned} &\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\jmath), \nu) \vdash R \\ &\overset{\mathcal{F}=\mathcal{E}\ \mathcal{T}=\mathcal{S}}{\Rightarrow} && (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\jmath), \nu) \vdash R \end{aligned}$$

If $(R, \jmath, \nu) = (Q, \imath, \lambda \cup \{j_1, \ldots, j_m\})$.

$$\begin{aligned} &\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\imath), \lambda) \vdash \mathsf{lock}\ s_{j_1}, \ldots, s_{j_m}; Q \\ &\overset{\tau_{lock}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\imath), \lambda \cup \{s_{j_1}, \ldots, s_{j_m}\}) \vdash Q \\ &\overset{\mathcal{F}=\mathcal{E}\ \mathcal{T}=\mathcal{S}\ \jmath=\imath\ \nu=\lambda\cup\{s_{j_1},\ldots,s_{j_m}\})\ R=Q}{\Rightarrow} && (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\jmath), \nu) \vdash R \end{aligned}$$

Case R = Red Unlock. In this case, $\mathcal{F} = \mathcal{E}$, $\mathcal{T} = \mathcal{S}$, $\mathcal{Q} = \mathcal{Q}' \cup \{(\mathsf{unlock}\ s_{j_1}, \ldots, s_{j_m}; Q, \imath, \lambda)\}$, and $\mathcal{R} = \mathcal{Q}' \cup \{(Q, \imath, \lambda \setminus \{s_{j_1}, \ldots, s_{j_m}\}))\}$. Let $(R, \jmath, \nu) \in \mathcal{R}$, then either $(R, \jmath, \nu) \in \mathcal{Q}'$ or $(R, \jmath, \nu) = (Q, \imath, \lambda \setminus \{s_{j_1}, \ldots, s_{j_m}\})$.

If $(R, \jmath, \nu) \in \mathcal{Q}'$.

$$\begin{aligned} &\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\jmath), \nu) \vdash R \\ &\overset{\mathcal{F}=\mathcal{E}\ \mathcal{T}=\mathcal{S}}{\Rightarrow} && (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\jmath), \nu) \vdash R \end{aligned}$$

If $(R, \jmath, \nu) = (Q, \imath, \lambda \setminus \{s_{j_1}, \ldots, s_{j_m}\})$.

$$\begin{aligned} &\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\imath), \lambda) \vdash \mathsf{unlock}\ s_{j_1}, \ldots, s_{j_m}; Q \\ &\overset{\tau_{unlock}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\imath), \lambda \setminus \{s_{j_1}, \ldots, s_{j_m}\}) \vdash Q \\ &\overset{\mathcal{F}=\mathcal{E}\ \mathcal{T}=\mathcal{S}\ \jmath=\imath\ \nu=\lambda\setminus\{s_{j_1},\ldots,s_{j_m}\}\ R=Q}{\Rightarrow} && (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\jmath), \nu) \vdash R \end{aligned}$$

Case R = Red Read. In this case, $\mathcal{F} = \mathcal{E}$, $\mathcal{T} = \mathcal{S}$,
$\mathcal{Q} = \mathcal{Q}' \cup \{(\mathsf{read}\ s_{j_1}, \ldots, s_{j_m}\ \mathsf{as}\ x_1, \ldots, x_m; Q, \imath, \lambda)\}$, and

$\mathcal{R} = \mathcal{Q}' \cup \{(Q\{\mathcal{S}(j_1)/x_1, \dots, \mathcal{S}(j_m)/x_m\}, (\mathcal{S}(j_1) :: \cdots :: \mathcal{S}(j_m) :: \imath), \lambda)\}$.
Let $(R, \jmath, \nu) \in \mathcal{R}$, then either $(R, \jmath, \nu) \in \mathcal{Q}'$ or
$(R, \jmath, \nu) = (Q\{\mathcal{S}(j_1)/x_1, \dots, \mathcal{S}(j_m)/x_m\}, (\mathcal{S}(j_1) :: \cdots :: \mathcal{S}(j_m) :: \imath), \lambda)$.

If $(R, \jmath, \nu) \in \mathcal{Q}'$**.**

$$\begin{aligned} &\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\jmath), \nu) \vdash R \\ &\overset{\mathcal{F}=\mathcal{E}\ \mathcal{T}=\mathcal{S}}{\Rightarrow} && (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\jmath), \nu) \vdash R \end{aligned}$$

If $(R, \jmath, \nu) = (Q\{\mathcal{S}(s_{j_1})/x_1, \dots, \mathcal{S}(s_{j_m})/x_m\}, (\mathcal{S}(j_1) :: \cdots :: \mathcal{S}(j_m) :: \imath), \lambda)$**.**

$$\begin{aligned} &\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\imath), \lambda) \vdash \mathsf{read}\ s_{j_1}, \dots, s_{s_m}\ \mathsf{as}\ x_1, \dots, x_m; Q \\ &\overset{\tau_{read}}{\Rightarrow} && (\mathcal{E} \cup \{x_k \mapsto \mathcal{E}(\mathcal{S}(j_k)) \mid 1 \leq k \leq m\}, \\ & && \qquad \mathcal{E}(\mathcal{S}), (\mathcal{E}(\mathcal{S}(j_1)) :: \cdots :: \mathcal{E}(\mathcal{S}(j_m)) :: \mathcal{E}(\imath)), \lambda) \vdash Q \\ &\overset{\text{Lem. 5.2}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), (\mathcal{E}(\mathcal{S}(j_1)) :: \cdots :: \mathcal{E}(\mathcal{S}(j_m)) :: \mathcal{E}(\imath)), \lambda) \vdash \\ & && \qquad Q\{\mathcal{S}(j_k)/x_k \mid 1 \leq k \leq m\} \\ &\overset{\substack{\mathcal{E}(\mathcal{S}(j_1))::\cdots::\mathcal{E}(\mathcal{S}(j_m))::\mathcal{E}(\imath)= \\ \mathcal{E}(\mathcal{S}(j_1)::\cdots::\mathcal{S}(j_m)::\iota)}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), (\mathcal{E}(\mathcal{S}(j_1) :: \cdots :: \mathcal{S}(j_m) :: \iota)), \lambda) \vdash \\ & && \qquad Q\{\mathcal{S}(j_k)/x_k \mid 1 \leq k \leq m\} \\ &\overset{\substack{\mathcal{F}=\mathcal{E}\ \mathcal{T}=\mathcal{S}\ \jmath=\mathcal{S}(j_1)::\cdots::\mathcal{S}(j_m)::\iota\wedge \\ \nu=\lambda\wedge R=Q\{\mathcal{S}(s_k)/x\}}}{\Rightarrow} && (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\jmath), \nu) \vdash R \end{aligned}$$

Case R = Red Write. In this case, $\mathcal{F} = \mathcal{E}$, $\mathcal{T} = \mathcal{S}[j_k \mapsto M_k \mid 1 \leq k \leq m]$, $\mathcal{Q} = \mathcal{Q}' \cup \{(s_{j_1}, \dots, s_{j_m} := M_1, \dots, M_m; Q, \imath, \lambda)\}$, and $\mathcal{R} = \mathcal{Q}' \cup \{(Q, \imath, \lambda)\}$. Let $(R, \jmath, \nu) \in \mathcal{R}$, then either $(R, \jmath, \nu) \in \mathcal{Q}'$ or $(R, \jmath, \nu) = (Q, \imath, \lambda)$. Let us first note that by hypothesis having $(s_{j_1}, \dots, s_{j_m} := M_1, \dots, M_m; Q, \imath, \lambda) \in \mathcal{Q}$ implies $(\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\imath), \lambda) \vdash s_{j_1}, \dots, s_{j_m} := M_1, \dots, M_m; Q$. And thus because $\mathcal{E}(\mathcal{S}) \leq \mathcal{E}(\mathcal{S})$ according to the typing rule τ_{write} it is the case that $\mathcal{E}(\mathcal{S}) \leq \mathcal{E}(\mathcal{T})$.

If $(R, \jmath, \nu) \in \mathcal{Q}'$**.**

$$\begin{aligned} &\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\jmath), \nu) \vdash R \\ &\overset{\text{Lem. 6}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{T}), \mathcal{E}(\jmath), \nu) \vdash R \\ &\overset{\mathcal{F}=\mathcal{E}}{\Rightarrow} && (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\jmath), \nu) \vdash R \end{aligned}$$

If $(R, \jmath, \nu) = (Q, \imath, \lambda)$**.**

$$\begin{aligned} &\overset{\text{Hyp.}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\imath), \lambda) \vdash s_{j_1}, \dots, s_{j_m} := M_1, \dots, M_m; Q \\ &\overset{\tau_{write}}{\Rightarrow} && (\mathcal{E}, \mathcal{E}(\mathcal{T}), \mathcal{E}(\imath), \lambda) \vdash Q \\ &\overset{\mathcal{F}=\mathcal{E}\ \jmath=\imath\ \nu=\lambda\ R=Q}{\Rightarrow} && (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\jmath), \nu) \vdash R \end{aligned}$$

□

We would like to thank the two reviewers for their detailed comments that have greatly contributed to improving our paper.

------ Review 1 ------

>>> The goal of this paper is to explain how to adapt ProVerif so that
>>> it can establish properties of protocols that are constrained by long
>>> term state. The current paper treats secrecy properties, though not
>>> correspondence properties, as would be appropriate to stress in the
>>> introduction.
>>> The strategy of work is adapted to the underlying form of ProVerif.
>>> ProVerif consists of two main parts, one of which translates (or
>>> abstracts) process algebra expressions to a clausal form; the second
>>> executes a resolution algorithm on these clauses. In StatVerif, a new
>>> translation takes as input an extended process algebra with some special
>>> constructs for reading, writing, and locking locations in a bounded
>>> store. The output language consists of clauses using a different pair of
>>> predicates from ProVerif; they are obtained from the ProVerif predicates
>>> by adding an argument position to each. The argument refers to the step
>>> at which the remainder of the assertion holds. The paper describes
>>> the translation and the semantics of the input and output languages, and
>>> proves that the translation is sound. It includes two examples. One is a
>>> simple hardware security module; the other is a fair exchange protocol.
>>> The contribution is substantial; the exposition and examples are
>>> effective; and the proof is plausible (though I have not checked many
>>> details). I recommend it for acceptance with minor revisions, which I
>>> will describe below.

>>> page number:

>>> 2. Mention secrecy rather than correspondence in this paper.
RESPONSE: We have now corrected this.

>>> 3. After code example, the text mentions "allow the user to
>>> provide" and "in return, the user will receive". The first "user" is
>>> Alice, and the second is Bob.
RESPONSE: We have now corrected this.

>>> 8. Last para of 3.1: Reword; sentence structure came out unclear.
RESPONSE: We have now corrected this.

>>> 9. Fig 2, read and assign clauses. Should the \cup in the LHS
>>> really be *disjoint* union?
RESPONSE: We consider multisets of processes, so the \cup symbol corresponds to the multiset union. This is true for all the clauses and not only the read and assign ones.

>>> 10. 3.3 Header: Ligature lost in my printout: "De nition of secrecy"
RESPONSE: We do not see the ligature problem mentioned.

>>> 10. 4.1, line 2: "each bounded name" should be bound name. (Also occurs passim.)
RESPONSE: We have now corrected this.

>>> 11. Fig 11. Caption below page no. It's better to split this into
>>> parts anyway, e.g. the part through if, and the part starting with lock.
RESPONSE: We have now corrected this.

>>>Actually, maybe also interchange if clause with let (matching),
>>> since the latter involves the conditional idea as well as the binding
>>> manipulations. It could be more readable with if first.
RESPONSE: We have now corrected this.

>>>attn is not yet explained.
RESPONSE: In Fig 2, we introduce the translation. The explanations
(including for attn) are given right after in sections 4.1.1, 4.1.2, and
4.1.3.

>>>The constant "fresh" is curious. You use it to indicate that all of
>>> these slots may have changed unpredictably since phi was recorded. But
>>> of course all of the "fresh"es are the same. And that's the worst case,
>>> since those values are maximally available and predictable for the
>>> adversary. You should comment on this in the text. In fact, you might
>>> consider, instead of the name fresh", calling this constant "stale".
RESPONSE: We have now corrected this. Please see Figure 3.

>>>Comment about the if clause etc: Since these clauses take the union
>>> of what we get from both branches, it would be the same if there were an
>>> implicit replication.
RESPONSE: Everything is the same with or without a replication, because
we do nothing when translating a replication. We handle conditionals
exactly like ProVerif does.

>>> 12. "to correctly abstract processes" should be "to correctly
>>> abstract *some* processes". Actually, reword to separate this sentence
>>> into several.
RESPONSE: We hope this paragraph is clearer now.

>>>Last bullet point: Expand on absence of correspondence properties.
>>> Comment about what might be needed to make them work.
RESPONSE: Our techniques do not directly apply to handle injective
correspondence properties. We haven't yet understood what would be
needed to make them work.

>>> 13. Assignment clause. Please expand.
RESPONSE: We have now corrected this.

>>>read clause: Should "arbitrary states" be arbitrary *values*?
RESPONSE: We have now corrected this.

>>> 14. 4.2. Intro. Please be more explicit about the syntactic setup.
>>> For instance, if it's closed, can it involve attch? The free cell names
>>> are just s_1...s_n, right?
RESPONSE: We have now corrected this.

>>>4.2.1. "to identify different instances". Strangely, "identify" can
>>> have opposite meanings. To identify variables means to equate them.
>>> However, to identify people means to distinguish them from all others.
>>> You appear to mean the latter. Clarify.
RESPONSE: We have now corrected this.

>>> 15. "new" clause: Clarify free and bound names.
RESPONSE: Our translation is parameterised by the initial honest process
P'_0 given to StatVerif. But we haven't made it explicit (with a
[[.]]_{P'_0} notation for example) for readability reasons. We have now
made this more clear at the beginning of Section 4.1 and in the caption
of Figure 3.

>>> 17. Fig 4. Strange type system, since premises \forall T . S\le T
>>> are not syntactic, and are presumably very hard to evaluate; certainly
>>> these rules give you no way to deduce formulas of this form. Comment.
RESPONSE: The typing system is just used in the proof but StatVerif does
not actually type check P'_0 against this type system. So this is why it
doesn't need to give us a way to deduce formulas. It is just part of the
proof technique which is adapted from the original ProVerif proof
technique for correctness. We have added explanations page 17.

>>>In the let rule, is M assumed to be normal in some sense? What does
>>> this rule mean if the rewrite rules are not convergent?
RESPONSE: The typing rule for the let construct requires that for all
the possible reductions of g(M_1, \dots, M_n), the process Q be

```
well-typed in the corresponding environment. Thus there is no
convergence requirement on the rewriting system.


>>> 23. "These three properties": Two?
RESPONSE: We have now corrected this.


------ Review 2 ------

>>> This paper presents StatVerif, a ProVerif extension to verify
>>> stateful protocols. The motivation for this work is the incapability of
>>> ProVerif to deal with several classes of stateful protocols, i.e.,
>>> protocols that maintain a state across sessions. Although these
>>> protocols can be modeled in applied pi-calculus using private channels,
>>> the abstraction which ProVerif builds on makes messages permanently
>>> available on these channels, introducing a number of false positives.
>>> The idea of this work is to extend the calculus with constructs to
>>> explicitly reason about state and to extend the Horn clauses used in the
>>> analysis with an argument tracking the current state. Such an
>>> abstraction is more precise and the authors show the effectiveness of
>>> their approach by analysing a simple protocol for cryptographic devices
>>> and a contract signing protocol. The contribution of this work is
>>> certainly interesting and useful. Furthermore, the presentation is
>>> overall clear and the technical content carefully explained. It catches
>>> the eye though that the body of the paper is largely identical with the
>>> paper published at CSF. Clearly the paper contains a huge amount of
>>> additional material in the appendix, and I would personally appreciate
>>> if the authors tried to integrate some of that material in the body of
>>> the paper.  RESPONSE: We agree with the reviewer that it's good if the
>>> body of the paper is self-contained. In this case, however, we think we
>>> have the right balance. Appendix A contains the complete StatVerif code
>>> of our two examples. We believe that integrating this to the body of the
>>> paper would break the flow and greatly harm the readability of our
>>> paper. We do illustrate the StatVerif constructs with parts of the code
>>> corresponding to the security device example in the body of the paper as
>>> we introduced the different notions and constructs. Appendix B contains
>>> the details of the proof of correctness of our translation. This are
>>> mainly easy inductions so we do not feel that detailing them in the body
>>> of the paper would help the understanding of the reader. On the contrary
>>> we fear that it would harm readability. The other extra contribution
>>> with respect to the conference paper is the implementation of StatVerif
>>> as an extension of ProVerif which is available online
>>> [http://markryan.eu/research/statverif/].


>>> Detailed comments:

>>> p.7: the last four primitives in Figure 1 operate on multiple
>>> cells as opposed to a single one. From a semantic point of view, this
>>> does not seem to be necessary. Does it simplify the analysis?
RESPONSE: It does by removing the superfluous intermediate states. We
have added a paragraph to explain this.


>>> p.8: you did not formalize the notion of scope.
RESPONSE: We have now corrected this.


>>> p.8: you say that the state (initialization) construct may occur
>>> within the scope of a replication. Initializing several times the same
>>> cell does not make sense and, indeed, is forbidden by the semantics.
RESPONSE: \new s; !([s\mapsto init] | P) doesn't make any sense. But
!\new s; ([s\mapsto init] | P) does. In particular, our semantics
doesn't allow more than one execution of the initialisation of cell s in
the first example.


>>> p.8: I did not fully understand the abbreviations introduced
>>> before the list of binders (i.e., the ones omitting the unlock
>>> construct). What happens if there are nested if or nested let
>>> constructs? There would be multiple unlock constructs, right? Does it
>>> make sense?
RESPONSE: It does make sense because only one else branch is ever taken.
The following example shows how that would work:
    lock s; if M = N then if M' = N' then P
    abbreviates
    lock s; if M = N then if M' = N' then P else unlock s; 0 else unlock s; 0
    which makes sense


>>> p.9: The notation \tilde{M} has not been introduced. If it stands
>>> for a sequence of arguments, then the attacker predicate is not binary.
RESPONSE: We have now corrected this.


>>> p.11: I did not understand the translation rule for the
>>> restriction. What is P?
RESPONSE: Our translation is parameterised by the initial honest process
P'_0 given to StatVerif. But we haven't made it explicit (with a
[[.]]_{P'_0} notation for example) for readability reasons. We have now
made this more clear at the beginning of Section 4.1 and in the caption
of Figure 3.


>>> p.11: You did not introduce "fresh". What is it formally? Is it
>>> always the same constant or a fresh one?
RESPONSE: As for reviewer 1 see the Notation introduced page 13.
```

```
>>> p.11: You should explain the type system in more detail and
>>> explain which properties of a process it captures.
RESPONSE: We have added explanations in section 4.2.2.


>>> p.16: You did not introduce the notion of subprocess.
RESPONSE: We have now corrected this.


>>> p.18: I could not parse the first sentence in the proof of Lemma 2.
RESPONSE: We have now corrected this.


>>> p.19: you say that you use ProVerif to analyze the case studies,
>>> but ProVerif does not deal with the Horn clauses produced by the
>>> StatVerif compiler. You should be more precise on this point. Did you
>>> modify the ProVerif resolution algorithm? If so, you should explain how
>>> and argue why it is sound.
RESPONSE: ProVerif does handle Horn clauses produced by the StatVerif
compiler. Indeed, one can either input a ProVerif process to ProVerif or
a set of arbitrary Horn Clauses, so we didn’t need to modify the
resolution algorithm, only the translation.


>>> p.25: "we are currently implementing the StatVerif compiler"...you
>>> seem to have already an implementation.
RESPONSE: Indeed we have implemented StatVerif on top of ProVerif and it
is available online [http://markryan.eu/research/statverif/]. We have
now made this clear in the paper.


>>> Besides the two simple examples borrowed from the conference
>>> submission, I would have expected a more comprehensive example in the
>>> journal version. For instance, it would be interesting to analyse a
>>> real-life cryptographic protocol suite (e.g., a security API for trusted
>>> hardware) to demonstrate how the analysis scales to larger protocols.
RESPONSE: There are other examples in the literature [8,13] that further
demonstrate the applicability of the StatVerif approach. We have added a
few words on this in our conclusion. However we do not feel that we
should include these examples in this paper because in [8,13] it is not
enough to use StatVerif to automatically analyse the considered
protocols. So extra abstractions are made before using StatVerif. So
including these examples would require us to introduce a lot more than
just the protocols and their models. In particular we would need to
introduce the notion of k-stability.
```