

# 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

November 28, 2013

## 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<sup>1</sup>.

**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  $\{(x_1, y_1)\}_{PK_k}, \dots, \{(x_n, y_n)\}_{PK_k}$  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 = pk(k) in
3   out(c, PKk) | [s ↦ init] |
4   ( ! lock s; in(c, x); read s as y;
5     if y = init then
6       ( if x = left then s := left; unlock s else
7         if x = right then s := right; unlock s ) ) |
8   ( ! lock s; in(c, x); read s as y; let z = adec(k, x) in
9     let xl = projl(z) in
10    let xr = projr(z) in
11    if y = left then out(c, xl); unlock s else
12    if y = right then 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.

<sup>1</sup><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

$$\begin{aligned}
 \text{sdec}(x, \text{senc}(x, y)) &\rightarrow y \\
 \text{adec}(x, \text{aenc}(\text{pk}(x), y)) &\rightarrow y \\
 \text{check\_getmsg}(\text{pk}(x), \text{sign}(x, y)) &\rightarrow y \\
 \text{checkpcs}(ct, \text{pk}(x), \text{pk}(y), \text{pk}(z), \text{pcs}(x, \text{pk}(y), \text{pk}(z), ct)) &\rightarrow \text{ok} \\
 \text{convertpcs}(z, \text{pcs}(x, \text{pk}(y), \text{pk}(z), ct)) &\rightarrow \text{sign}(x, ct)
 \end{aligned}$$

In this example, we consider a signature that has the constructors `senc`, `aenc`, `pk`, `pcs`, `sign` and `ok`. The functions `sdec`, `checkpcs`, `check_getmsg` and `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  $P, Q, R, \dots$  are constructed as follows. The process  $0$  is the empty process which does nothing. In `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 `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 `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  $k$ :*

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

$B$  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

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

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 `attacker` and `message`. The fact `attacker( $N$ )` means that the attacker can learn the value  $N$ . The fact `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  $[\dots]$ ; function symbols are followed by round brackets  $(\dots)$ ; 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

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

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  $\text{aenc}(xpk, \text{sign}(skA[], k[xpk]))$ . 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.

$$\begin{aligned} & \text{attacker}(ok()) \\ & \text{attacker}(v) \rightarrow \text{attacker}(pk(v)) \\ & \text{attacker}(v_1) \wedge \text{attacker}(v_2) \rightarrow \text{attacker}(\text{sign}(v_1, v_2)) \\ & \text{attacker}(v_1) \wedge \text{attacker}(v_2) \rightarrow \text{attacker}(\text{senc}(v_1, v_2)) \\ & \text{attacker}(v_1) \wedge \text{attacker}(v_2) \rightarrow \text{attacker}(\text{aenc}(v_1, v_2)) \\ & \text{attacker}(v_1) \wedge \text{attacker}(v_2) \wedge \text{attacker}(v_3) \rightarrow \text{attacker}(\text{pcs}(v_1, v_2, v_3)) \end{aligned}$$

**Clauses corresponding to the term reductions**

$$\begin{aligned} & \text{attacker}(\text{pk}(x)) \wedge \text{attacker}(\text{sign}(x, y)) \rightarrow \text{attacker}(y) \\ & \text{attacker}(x) \wedge \text{attacker}(\text{aenc}(\text{pk}(x), y)) \rightarrow \text{attacker}(y) \\ & \text{attacker}(x) \wedge \text{attacker}(\text{senc}(x, y)) \rightarrow \text{attacker}(y) \end{aligned}$$

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

$$\begin{aligned} & \text{message}(v_1, v_2) \wedge \text{attacker}(v_1) \rightarrow \text{attacker}(v_2) \\ & \text{attacker}(v_1) \wedge \text{attacker}(v_2) \rightarrow \text{message}(v_1, v_2) \\ & \text{attacker}(c[]) \end{aligned}$$

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  $\text{attacker}(s[])$  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  $\text{attacker}(s)$  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  $\text{attacker}(s)$  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  $M$ ;
- $\text{read } s_1, \dots, s_n \text{ as } x_1, \dots, x_n; P$ , which reads the values stored in cells  $s_1, \dots, s_n$  (calling them  $x_1, \dots, x_n$  respectively), and then continues as  $P$ ;
- $s_1, \dots, s_n := M_1, \dots, M_n; P$  which assigns  $M_1, \dots, M_n$  to  $s_1, \dots, s_n$  respectively and then continues as  $P$ ;
- $\text{lock } s_1, \dots, s_n; P$ . This process begins a *locked section*; that means that the process takes exclusive access to the state cells  $s_1, \dots, s_n$ , and continues as  $P$ ; and
- $\text{unlock } s_1, \dots, s_n; P$ , which releases the lock on the state cells  $s_1, \dots, s_n$ , 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 **lock**  $s_1, \dots, s_n$  and **unlock**  $s_1, \dots, s_n$  is to allow manipulations of the global state cells  $s_1, \dots, s_n$  to proceed without interference from other concurrent processes. Obviously, such interactions would lead to unwanted results. For example, in our security device, the **lock**  $s$  and **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 **lock**  $s$  and **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 **read**  $s_1, \dots, s_n \text{ as } x_1, \dots, x_n$  and  $s_1, \dots, s_n := M_1, \dots, M_n$  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  $\text{read } s_1 \text{ as } x; \text{read } s_2 \text{ as } y$  may result in the hypotheses  $\text{message}((x, vs_1), vc_1, vm_1) \wedge \text{message}((vs_2, y), vc_2, vm_2)$ , whereas  $\text{read } s_1, s_2 \text{ as } x, y$  can result in just the hypothesis  $\text{message}((x, y), vc, vm)$ .

$M, N ::=$	terms
$x, y, z$	variables
$a, b, c, k, s$	names
$f(M_1, \dots, M_n)$	constructor application
$\tilde{M}, \tilde{N} ::=$	tuple of terms
$(M_1, \dots, M_n)$	
$P, Q ::=$	processes
$0$	nil
$\text{out}(M, N); P$	output
$\text{in}(M, x); P$	input
$P \mid Q$	parallel composition
$!P$	replication
$\text{new } a; P$	restriction
$\text{let } x = g(M_1, \dots, M_n)$	
$\quad \text{in } P \text{ else } Q$	destructor application
$\text{if } M = N \text{ then } P \text{ else } Q$	conditional
$[s \mapsto M]$	state
$\text{read } s_1, \dots, s_n \text{ as } x_1, \dots, x_n; P$	read
$s_1, \dots, s_n := M_1, \dots, M_n; P$	assignment
$\text{lock } s_1, \dots, s_n; P$	beginning of locked section
$\text{unlock } s_1, \dots, s_n; P$	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  $\text{if } M = N \text{ then } P$  occurs in the scope of a **lock**  $s_1, \dots, s_n$ , then it is an abbreviation of  $\text{if } M = N \text{ then } P \text{ else unlock } s_1, \dots, s_n; 0$ , otherwise it is an abbreviation of  $\text{if } M = N \text{ then } P \text{ else } 0$ . Similarly, if the subprocess  $\text{let } x = g(M_1, \dots, M_n) \text{ in } P$  occurs in the scope of a **lock**  $s_1, \dots, s_n$ , then it is an abbreviation of  $\text{let } x = g(M_1, \dots, M_n) \text{ in } P \text{ else unlock } s_1, \dots, s_n; 0$ , otherwise it is an abbreviation of  $\text{let } x = g(M_1, \dots, M_n) \text{ in } P \text{ else } 0$ . We also write  $\text{let } x = f(M_1, \dots, M_n) \text{ in } P$  to mean  $P\{f(M_1, \dots, M_n)/x\}$ , as an abbreviation for repeated constructor application.

The process **new**  $a; P$  binds  $a$  in  $P$ ,  $\text{in}(c, x); P$  binds  $x$  in  $P$ ,  $\text{let } x = g(M_1, \dots, M_n) \text{ in } P \text{ else } Q$  binds  $x$  in  $P$  (but not in  $Q$ ), and **read**  $s_1, \dots, s_n \text{ as } x_1, \dots, x_n; P$  binds  $x_1, \dots, x_n$  in  $P$ . The scope of  $a, x, x_1, \dots, x_n$  is  $P$ . As usual, we use  $\text{bn}(P)$  and  $\text{bv}(P)$  to denote the set of bound names and bound variables of  $P$  respectively, and  $\text{fn}(P)$  and  $\text{fv}(P)$  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  $(P, \lambda)$  where  $P$  is a process and  $\lambda$  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  $\lambda$ s. The environment  $\mathcal{E}$  must contain at least the free names of  $\mathcal{S}$  and  $\mathcal{P}$ . The configuration  $(\{a_1, \dots, a_n\}, \mathcal{S}, \{(P_1, \emptyset), \dots, (P_m, \emptyset)\})$  intuitively corresponds to the process  $\text{new } a_1, \dots, a_n; ([s \mapsto \mathcal{S}(s) \mid s \in \text{dom}(\mathcal{S})] \mid P_1 \mid \dots \mid P_m)$ .

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  $\lambda$  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**.  $s_1, \dots, s_n := M_1, \dots, M_n$  and **read**  $s_1, \dots, s_n \text{ as } x_1, \dots, x_n$  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  $\text{Init}$  with at least one channel name  $\text{attach} \in \text{Init}$ .  $A$  is said to be an *Init*-adversary if  $A$  is a closed process and  $\text{fn}(A) \subseteq \text{Init}$ .

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,  $\text{Init}$  a finite set of names such that  $\text{attach} \in \text{Init}$ ,  $M$  a message.  $P$  preserves the secrecy of  $M$  against  $\text{Init}$  if for any *Init*-adversary  $A$ , there exists no trace of the form:*

$$((\text{Init} \cup \text{fn}(P) \cup \text{fn}(M)), \emptyset, \{(P \mid A, \emptyset)\}) \rightarrow^* (\mathcal{E}, \mathcal{S}, \mathcal{Q} \cup \{(\text{out}(\text{attach}, M); Q, \lambda)\}).$$

$$\begin{aligned}
& (\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 \{(\text{new } a; P, \lambda)\}) \rightarrow (\mathcal{E} \cup \{a'\}, \mathcal{S}, \mathcal{P} \cup \{(P\{a'/a\}, \lambda)\}) \quad \text{if } a' \text{ fresh} \\
& (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\text{let } x = g(M_1, \dots, M_n) \text{ in} \\
& \quad P \text{ else } Q, \lambda)\}) \rightarrow (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(P\{M'/x\}, \lambda)\}) \\
& \quad \quad \quad \text{if } g(M_1, \dots, M_n) \rightarrow M' \\
& (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\text{let } x = g(M_1, \dots, M_n) \text{ in} \\
& \quad P \text{ else } Q, \lambda)\}) \rightarrow (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(Q, \lambda)\}) \\
& \quad \quad \quad \text{if } \nexists M', g(M_1, \dots, M_n) \rightarrow M' \\
& (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\text{if } M = M \\
& \quad \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 \\
& \quad \text{then } P \text{ else } Q, \lambda)\}) \rightarrow (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(Q, \lambda)\}) \quad \text{if } M \neq N \\
& (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\text{out}(M, N); P, \lambda_1), \\
& \quad (\text{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}) \quad \text{if } s \in \text{dom}(\mathcal{E}) \\
& \quad \quad \quad \text{and } s \notin \text{dom}(\mathcal{S}) \\
& (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\text{lock } s_1, \dots, s_n; P, \lambda)\}) \rightarrow (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(P, \lambda \cup \{s_1, \dots, s_n\})\}) \\
& \quad \quad \quad \text{if } \forall (Q, \lambda') \in \mathcal{P}. \{s_1, \dots, s_n\} \cap \lambda' = \emptyset \\
& (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\text{unlock } s_1, \dots, s_n; P, \lambda)\}) \rightarrow (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(P, \lambda \setminus \{s_1, \dots, s_n\})\}) \\
& (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\text{read } s_1, \dots, s_n \text{ as} \\
& \quad x_1, \dots, x_n; P, \lambda)\}) \rightarrow (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(P\{\mathcal{S}(s_1)/x_1, \dots, \mathcal{S}(s_n)/x_n\}, \lambda)\}) \\
& \quad \quad \quad \text{if } s_1, \dots, s_n \in \text{dom}(\mathcal{S}) \\
& \quad \quad \quad \text{if } \forall (Q, \lambda') \in \mathcal{P}. \{s_1, \dots, s_n\} \cap \lambda' = \emptyset \\
& (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(s_1, \dots, s_n := \\
& \quad M_1, \dots, 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)\}) \\
& \quad \quad \quad \text{if } s_1, \dots, s_n \in \text{dom}(\mathcal{S}) \\
& \quad \quad \quad \text{if } \forall (Q, \lambda') \in \mathcal{P}. \{s_1, \dots, s_n\} \cap \lambda' = \emptyset
\end{aligned}$$

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  $(P, \lambda)$ , where  $P$  is the process itself and  $\lambda$  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  $a[p_1, \dots, p_n]$  with  $p_1, \dots, p_n$  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:

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

The clauses are built around the predicates **attacker** and **message** with the following meanings:

- **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].
- **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:

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

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  $s_1, \dots, s_n$  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.

$$\begin{aligned}
\llbracket 0 \rrbracket \rho H \iota \phi \lambda &= \emptyset \\
\llbracket P \mid Q \rrbracket \rho H \iota \phi \emptyset &= \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 &= \llbracket P \rrbracket \rho H \iota \phi \emptyset \\
\llbracket \text{new } a; P \rrbracket \rho H \iota \phi \lambda &= \begin{cases} \llbracket P \rrbracket (\rho \cup \{a \mapsto a[\iota]\}) H \iota \phi \lambda & \text{if } a \in \text{bn}(P_0) \\ \llbracket P \rrbracket (\rho \cup \{a \mapsto \text{attn}[\ ]\}) H \iota \phi \lambda & \text{otherwise} \end{cases} \\
\llbracket \text{in}(M, x); P \rrbracket \rho H \iota \phi \lambda &= \llbracket P \rrbracket \rho' H'(x :: \iota) \phi' \lambda \\
&\quad \text{where } \phi' = \phi [k \mapsto vs_k \mid k \notin \lambda] \\
&\quad \text{and } H' = H \wedge \text{message}(\phi', \rho(M), x) \\
&\quad \text{and } \rho' = \rho \cup \{x \mapsto x\} \cup \{vs_k \mapsto vs_k \mid k \notin \lambda\} \\
&\quad \text{and } vs_1, \dots, vs_n \text{ are fresh variables} \\
\llbracket \text{out}(M, N); P \rrbracket \rho H \iota \phi \lambda &= \{H \Rightarrow \text{message}(\phi, \rho(M), \rho(N))\} \cup \llbracket P \rrbracket \rho H \iota \phi \lambda \\
\llbracket \text{if } M = N \text{ then} \\
\quad P \text{ else } Q \rrbracket \rho H \iota \phi \lambda &= \llbracket P \rrbracket (\rho \sigma) (H \sigma) (\iota \sigma) (\phi \sigma) \lambda \cup \llbracket Q \rrbracket \rho H \iota \phi \lambda \\
&\quad \text{where } \sigma = \text{mgu}(\rho(M), \rho(N)) \\
\llbracket \text{let } x = g(M_1, \dots, M_t) \text{ in} \\
\quad P \text{ else } Q \rrbracket \rho H \iota \phi \lambda &= \llbracket Q \rrbracket \rho H \iota \phi \lambda \\
&\cup \left\{ \llbracket P \rrbracket ((\rho \sigma) \cup \rho') (H \sigma) (\iota \sigma) (\phi \sigma) \lambda \mid \right. \\
&\quad \left. \begin{aligned} &g(p_1, \dots, p_t) \mapsto p \in \text{def}(g) \\ &\text{and } \{z_1, \dots, z_m\} = \text{fv}(g(p_1, \dots, p_t)) \\ &\text{and } z'_1, \dots, z'_m \text{ are fresh variables} \\ &\text{and } \sigma' = \{z_i \mapsto z'_i \mid 1 \leq i \leq m\} \\ &\text{and } \sigma = \text{mgu}(g(M_1 \rho, \dots, M_t \rho), g(p_1 \sigma', \dots, p_t \sigma')) \\ &\text{and } \rho' = \{x \mapsto p \sigma' \sigma\} \cup \{z'_i \mapsto z'_i \sigma \mid 1 \leq i \leq m\} \end{aligned} \right\}
\end{aligned}$$

$$\begin{aligned}
\llbracket \text{lock } s_{i_1}, \dots, \\
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, \dots, i_m\}) \\
&\quad \text{where } vs_1, \dots, vs_n \text{ are fresh variables} \\
&\quad \text{and } \phi' = \phi [k \mapsto vs_k \mid k \notin \lambda] \\
\llbracket \text{unlock } s_{i_1}, \dots, \\
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, \dots, i_m\}) \\
&\quad \text{where } vs_1, \dots, vs_n \text{ are fresh variables} \\
&\quad \text{and } \phi' = \phi [k \mapsto vs_k \mid k \notin \lambda] \\
\llbracket \text{read } s_{i_1}, \dots, s_{i_m} \\
\text{as } x_1, \dots, x_m; P \rrbracket \rho H \iota \phi \lambda &= \llbracket P \rrbracket \rho'' H' (x_1 :: \dots :: x_m :: \iota) \phi' \lambda \\
&\quad \text{where } \rho'' = \rho' \cup \{x_j \mapsto \phi'_{i_j} \mid 1 \leq j \leq m\} \cup \{vc \mapsto vc, vm \mapsto vm\} \\
&\quad \quad \text{and } \rho' = \rho \cup \{vs_k \mapsto vs_k \mid k \notin \lambda\} \\
&\quad \quad \text{and } \phi' = \phi [k \mapsto vs_k \mid k \notin \lambda] \\
&\quad \quad \text{and } H' = H \wedge \text{message}(\phi', vc, vm) \\
&\quad \text{and } vs_1, \dots, vs_n, vc, vm \text{ are fresh variables} \\
\llbracket s_{i_1}, \dots, s_{i_m} := \\
M_1, \dots, 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 \\
&\quad \cup \{H \wedge \text{message}(\phi', vc, vm) \Rightarrow \text{message}(\phi'', vc, vm)\} \\
&\quad \cup \{H \wedge \text{attacker}(\phi', vm) \Rightarrow \text{attacker}(\phi'', vm)\} \\
&\quad \quad \text{where } \phi' = \phi [k \mapsto vs_k \mid k \notin \lambda] \\
&\quad \quad \text{and } \phi'' = \phi' [i_j \mapsto \rho(M_j) \mid 1 \leq j \leq m] \\
&\quad \text{and } vs_1, \dots, vs_n, vc, vm \text{ are fresh variables}
\end{aligned}$$

Figure 3: The rules for translating the stateful process  $P_0$  into clauses. The translation of the StatVerif process  $\text{new } \tilde{n}; ([s_1 \mapsto M_1] \mid \dots \mid [s_n \mapsto M_n] \mid P_0)$  is  $\llbracket P_0 \rrbracket \rho_0 \text{ true } \square \phi_0$ . (Note that the rule for  $\text{new } a$  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  $P'_0 = \text{new } \tilde{m}; ([s_1 \mapsto M_1] \mid \dots \mid [s_n \mapsto M_n] \mid P_0)$  be a StatVerif process. Let  $\rho_0$  be the function  $\{a \mapsto a[], s_i \mapsto s_i[] \mid a \in \text{fn}(P'_0), 1 \leq i \leq n\}$  and let  $\phi_0 = (\rho_0(M_1), \dots, \rho_0(M_n))$ . The process  $P'_0$  is translated into the following sets of clauses:

- $\llbracket P_0 \rrbracket \rho_0 \text{ true } [] \phi_0 \emptyset$  where the function  $\llbracket \cdot \rrbracket \rho H \iota \phi \lambda$  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  $\rho, H, \iota, \phi$  and  $\lambda$ , which have the following purposes:

- $\rho$  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.
- $\iota$  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”.
- $\phi$  is a tuple of terms  $(M_1, \dots, M_n)$  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  $\phi$  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  $\phi$  corresponding to unlocked cells are discarded and replaced with fresh variables.
- $\lambda$  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  $\rho$  and  $\iota$  the variable that is input, and add a hypothesis to  $H$ . As explained above, entries for  $\phi$  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 *lock*  $s_{i_1}, \dots, s_{i_m}$ , 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  $i_1, \dots, i_m$  to the set of already locked cell names  $\lambda$ .

- We translate `unlock`  $s_{i_1}, \dots, s_{i_m}$  in the same way as `lock`, except that the cell indices are removed from  $\lambda$  instead of added.
- The assignment  $s_{i_1}, \dots, s_{i_m} := M_1, \dots, M_m$  updates the current state  $\phi$ . The indices  $i_1, \dots, i_m$  in  $\phi$  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:

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

and the following ones for writing:

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

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

#### 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  $n$ ,

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

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

$$\text{attacker}(xs, M_1) \wedge \dots \wedge \text{attacker}(xs, M_n) \rightarrow \text{attacker}(xs, M)$$

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

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

Finally, the attacker knows

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

where  $\rho_0$  and  $\phi_0$  are as defined in section 4.1.1.

## 4.2 Correctness

Let  $P'_0 = \mathbf{new} \tilde{m}; ([s_1 \mapsto M_1] \mid \cdots \mid [s_n \mapsto M_n] \mid P_0)$  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  $P'_0$  (*i.e.*,  $s_1, \dots, s_n$ ), 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 new  $a$  in  $P_0$  it is an instance of. This representation of names allows us in particular to associate different instances of a new  $a$  with each other when arising from 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  $(Q, \iota, \lambda)$  where we will record in  $\iota$  the list of  $M_1, \dots, M_n$  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  $\iota$ s are unchanged. And

- The reduction rule for communication becomes the following

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

which records  $N$  in  $\iota_2$ .

- The reduction rule for read becomes the following

$$(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\mathbf{read} s_1, \dots, s_n \text{ as } x_1, \dots, x_n; P, \iota, \lambda)\}) \rightarrow (\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(P\{\mathcal{S}(s_1)/x_1, \dots, \mathcal{S}(s_n)/x_n\}, (\mathcal{S}(s_1) :: \dots :: \mathcal{S}(s_n) :: \iota), \lambda)\})$$

**if**  $s_1, \dots, s_n \in \text{dom}(\mathcal{S})$  **and**  $\forall (Q, \lambda') \in \mathcal{P}. \{s_1, \dots, s_n\} \cap \lambda' = \emptyset$

which records  $\mathcal{S}(s_1), \dots, \mathcal{S}(s_n)$  in  $\iota_2$ .



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

$$(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\text{new } a; P, \iota, \lambda)\}) \rightarrow (\mathcal{E} \cup \{a' \mapsto a[\mathcal{E}(\iota)]\}, \mathcal{S}, \mathcal{P} \cup \{(P\{a'/a\}, \iota, \lambda)\})$$

**if**  $a \in \text{bn}(P'_0)$  **and**  $a'$  fresh

which records that  $a'$  is an instance of **new**  $a$ .  $\iota$  is used to distinguish two instances of **new**  $a$  on the basis of the previous inputs.

$$(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\text{new } a; P, \iota, \lambda)\}) \rightarrow (\mathcal{E} \cup \{a' \mapsto \text{attn}[]\}, \mathcal{S}, \mathcal{P} \cup \{(P\{a'/a\}, \iota, \lambda)\})$$

**if**  $a \notin \text{bn}(P'_0)$  **and**  $a'$  fresh

which records that  $a'$  is a 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  $k \in \{1, 2\}$ ,  $\text{dom}(\mathcal{E}'_k) = \mathcal{E}_k$  and for all  $(P, \lambda) \in \mathcal{P}_k$ ,  $(P, \iota, \lambda) \in \mathcal{P}'_k$  for some  $\iota$ .*

*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  $k \in \{1, 2\}$ ,  $\mathcal{E}_k = \text{dom}(\mathcal{E}'_k)$  and for all  $(P, \iota, \lambda) \in \mathcal{P}'_k$ ,  $(P, \lambda) \in \mathcal{P}_k$ .*

#### 4.2.2 Proof of correctness

Let  $P'_0 = \text{new } \tilde{m}; ([s_1 \mapsto M_1] \mid \dots \mid [s_n \mapsto M_n] \mid P_0)$  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  $\mathcal{S}_0 = \{s_1 \mapsto M_1, \dots, s_n \mapsto M_n\}$ . Let  $\mathcal{E}_0$  be the environment such that

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

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

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

$$\begin{aligned} & \text{attacker}(\overline{\mathcal{R}}, \text{attach}[]) \in \mathcal{F}_0 \\ \wedge \quad & \forall M, N \text{ message}(\overline{\mathcal{R}}, M, N) \in \mathcal{F}_0 \Rightarrow \text{message}(\overline{\mathcal{S}}, M, N) \in \mathcal{F}_0 \\ \wedge \quad & \forall M \text{ attacker}(\overline{\mathcal{R}}, M) \in \mathcal{F}_0 \Rightarrow \text{attacker}(\overline{\mathcal{S}}, M) \in \mathcal{F}_0 \end{aligned}$$

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  $\text{message}(\overline{\mathcal{S}}, \mathcal{E}(M), \mathcal{E}(N))$  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  $\iota$ , and the mode  $\lambda$ , 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),  $\iota$  a sequence of patterns, and  $\lambda$  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)  $(\text{bn}(B) \cup \text{bv}(B)) \cap \text{dom}(\mathcal{E}) = \emptyset$ ; and
- (iv)  $\forall a \in \text{fn}(B), \text{attacker}(\overline{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0$ ; and
- (v)  $\forall x \in \text{fv}(B), \text{attacker}(\overline{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0$ ; and
- (vi) for all  $i \in \{1, \dots, n\}$ ,  $i \in \lambda$  if and only if  $B$  is in the scope of a lock  $\dots s_i \dots$  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,  $\text{dom}(\mathcal{E}_0) = \text{fn}(P) \cup \text{fn}(A) \cup \text{cell}P$  and  $(\text{bn}(A) \cup \text{bv}(A)) \cap (\text{fn}(P) \cup \text{fn}(A) \cup \text{cell}P) = \emptyset$ , thus  $(\text{bn}(A) \cup \text{bv}(A)) \cap \text{dom}(\mathcal{E}_0) = \emptyset$ .
- (iv) By construction,  $\forall a \in \text{fn}(A)$   
 If  $a = \text{attch}$ , then  $\mathcal{E}_0(a) = \text{attch}[\ ]$ , and  $\text{attacker}(\overline{\mathcal{E}_0(\mathcal{S}_0)}, \text{attch}[\ ]) \in \mathcal{C}_0$  by construction.  
 If  $a \neq \text{attch}$ , then  $\mathcal{E}_0(a) = \text{attn}[\ ]$ , and  $\text{attacker}(\overline{\mathcal{E}_0(\mathcal{S}_0)}, \text{attn}[\ ]) \in \mathcal{C}_0$  by construction.  
 Thus  $\forall a \in \text{fn}(A) \text{attacker}(\overline{\mathcal{E}_0(\mathcal{S}_0)}, \mathcal{E}_0(a)) \in \mathcal{F}_0$ .

$$\begin{array}{c}
\frac{\text{message}(\bar{\mathcal{S}}, \mathcal{E}(M), \mathcal{E}(N)) \in \mathcal{F}_0 \quad (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash P}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \text{out}(M, N); P} \tau_{out} \\
\\
\frac{\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 \text{message}(\bar{\mathcal{T}}, \mathcal{E}(M), N) \in \mathcal{F}_0) \Rightarrow \mathcal{E} \cup \{x \mapsto N\}, \mathcal{T}, (N :: \iota), \lambda) \vdash P}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \text{in}(M, x); P} \tau_{in} \\
\\
\frac{}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash 0} \tau_{nil} \quad \frac{(\mathcal{E}, \mathcal{S}, \iota, \emptyset) \vdash P \quad (\mathcal{E}, \mathcal{S}, \iota, \emptyset) \vdash Q}{(\mathcal{E}, \mathcal{S}, \iota, \emptyset) \vdash P \mid Q} \tau_{par} \quad \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) \quad (\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 \text{bn}(P'_0) \Rightarrow (\mathcal{E} \cup \{a \mapsto a[l]\}, \mathcal{S}, \iota, \lambda) \vdash P}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \text{new } a; P} \tau_{newP} \\
\\
\frac{a \notin \text{bn}(P'_0) \Rightarrow (\mathcal{E} \cup \{a \mapsto \text{attn}[\ ]\}, \mathcal{S}, \iota, \lambda) \vdash P}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \text{new } a; P} \tau_{newA} \\
\\
\frac{\forall M (g(\mathcal{E}(M_1), \dots, \mathcal{E}(M_n)) \rightarrow 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, \dots, M_n) \text{ in } P \text{ else } Q} \tau_{let} \\
\\
\frac{\forall \mathcal{T} (\mathcal{S} \leq \mathcal{T} \wedge \mathcal{T} = \mathcal{T}[j \mapsto \mathcal{S}(j) \mid j \in \lambda]) \Rightarrow (\mathcal{E} \cup \{x_k \mapsto \mathcal{T}(j_k) \mid 1 \leq k \leq m\}, \mathcal{T}, (\mathcal{T}(j_1) :: \dots :: \mathcal{T}(j_m) :: \iota), \lambda) \vdash P}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \text{read } s_{j_1}, \dots, s_{j_m} \text{ as } x_1, \dots, x_m; P} \tau_{read} \\
\\
\frac{\forall \mathcal{T} (\mathcal{S} \leq \mathcal{T} \wedge \mathcal{T} = \mathcal{T}[j \mapsto \mathcal{S}(j) \mid j \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 P)}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash s_{j_1}, \dots, s_{j_m} := M_1, \dots, 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, \dots, j_m\}) \vdash P)}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \text{lock } s_{j_1}, \dots, 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, \dots, j_m\}) \vdash P)}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \text{unlock } s_{j_1}, \dots, s_{j_m}; P} \tau_{unlock}
\end{array}$$

Figure 4: Typing system for correctness proof. Note that the rules  $\tau_{newP}$  and  $\tau_{newA}$  refer to the initial honest process  $P'_0$ .

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

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

**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  $P_0$ ,  $\sigma$  a substitution from variables to patterns,  $\rho$  a mapping from names and variables to patterns,  $H$  a conjunction of fact,  $\iota$  a set of variables,  $\phi$  a tuple of terms, and  $\lambda$  a set of indices. We first prove by induction on the size of  $Q$ , that if

(i)  $\rho$  binds all the free names and variables of  $Q$ ,  $H$ ,  $\iota$  and  $\phi$ ;

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

(iii)  $\sigma$  is a closed substitution;

(iv)  $i \in \lambda$  if and only if  $Q$  is in the scope of `lock ... si ...` in  $P$ ;

(v)  $\mathcal{C}_0 \supseteq \llbracket Q \rrbracket \rho H \iota \bar{\phi} \lambda$ ;

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

(vii)  $\text{attacker}(\bar{\phi}\sigma, \text{attach}[]) \in \mathcal{F}_0$ ,

then

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

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

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

(ii) By definition  $\sigma$  is a closed substitution.

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

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

(v) By definition  $\mathcal{C}_0 \supseteq \llbracket P \rrbracket \rho H \iota \bar{\phi} \lambda$ .

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

(vii) By construction,  $\text{attacker}(\bar{\phi}\sigma, \text{attach}[]) \in \mathcal{C}_0$ . So in particular, we have that  $\text{attacker}(\bar{\phi}\sigma, \text{attach}[]) \in \mathcal{F}_0$ .

Thus,  $P$ ,  $\rho$ ,  $\sigma$ ,  $H$ ,  $\iota$ ,  $\phi$  and  $\emptyset$  satisfy the conditions of our induction result according to which  $(\mathcal{E}_0, \mathcal{E}_0(\mathcal{S}_0), [], \emptyset) \vdash P$ .  $\square$

**Lemma 3** (Subject reduction). *Let  $(\mathcal{E}, \mathcal{S}, \mathcal{Q}) \rightarrow (\mathcal{F}, \mathcal{T}, \mathcal{R})$  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  $\text{cells}(\mathcal{Q}) \subseteq \{s_1, \dots, s_n\}$ . If  $(\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\iota), \lambda) \vdash Q$  for all  $(Q, \iota, \lambda) \in \mathcal{Q}$ , then  $(\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(\mathcal{J}), \nu) \vdash R$  for all  $(R, \mathcal{J}, \nu) \in \mathcal{R}$ .*

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

**Theorem 1.** *Consider the instrumented trace*

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

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

$$\text{message}(\overline{\mathcal{E}(\mathcal{S})}, \mathcal{E}(M), \mathcal{E}(N)) \in \mathcal{F}_0 \quad \text{and} \quad \text{attacker}(\overline{\mathcal{E}(\mathcal{S})}, \text{attach}[]) \in \mathcal{F}_0.$$

*Proof.* Consider the instrumented trace

$$(\mathcal{E}_0, \mathcal{S}_0, \{(P_0 \mid A, [], \emptyset)\}) = (\mathcal{E}_0, \mathcal{S}_0, \mathcal{Q}_0) \rightarrow (\mathcal{E}_1, \mathcal{S}_1, \mathcal{Q}_1) \rightarrow \dots \rightarrow (\mathcal{E}_n, \mathcal{S}_n, \mathcal{Q}_n) = (\mathcal{E}, \mathcal{S}, \mathcal{Q} \cup \{(Q, \iota, \lambda)\})$$

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

$$\text{attacker}(\overline{\mathcal{E}_i(\mathcal{S}_i)}, \text{attach}[]) \in \mathcal{F}_0 \quad \text{and} \quad \forall (R, \iota, \nu) \in \mathcal{Q}_i \ (\mathcal{E}_i, \mathcal{E}_i(\mathcal{S}_i), \mathcal{E}_i(\iota), \nu) \vdash R$$

**Base case** ( $i = 0$ ). By definition of the StatVerif compiler we have that  $\text{attacker}(\overline{\mathcal{E}_0(\mathcal{S}_0)}, \text{attach}[]) \in \mathcal{C}_0$  and thus  $\text{attacker}(\overline{\mathcal{E}_0(\mathcal{S}_0)}, \text{attach}[]) \in \mathcal{F}_0$ . 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  $\tau_{par}$ ,

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

**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  $\text{attacker}(\mathcal{E}_n(\mathcal{S}_n), \mathcal{E}_n(\text{attach})) \in \mathcal{F}_0$ , and  $(\mathcal{E}_n, \mathcal{E}_n(\mathcal{S}_n), \mathcal{E}_n(\iota), \nu) \vdash R$  for all  $(R, \iota, \nu) \in \mathcal{Q}$ .

This concludes our induction and gives us

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

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

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

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

## 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<sup>2</sup> 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:

```

1  let device =
2    new s;
3    out(c, PKk) | [s ↦ init] |
4    ( ! lock s; in(c, x); read s as y;
5      if y = init then
6        ( if x = left then s := left; unlock s else
7          if x = right then s := right; unlock s ) ) |
8    ( ! lock s; in(c, x); read s as y; let z = adec(k, x) in
9      let zl = projl(z) in
10     let zr = projr(z) in
11     ( if y = left then out(c, zl); unlock s else
12       if y = right then out(c, zr); unlock s ) )
13
14 let user =
15   new sl; new sr; new r;
16   out(c, aenc(PKk, r, (sl, sr)))
17
18 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:

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

The query is `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  $\llbracket \text{system} \rrbracket \rho_0 \text{ true} \llbracket \phi_0 \rrbracket \text{false}$ , where  $\rho_0 = \{c \mapsto c[], \text{left} \mapsto \text{left}[], \text{right} \mapsto \text{right}[], \text{init} \mapsto \text{init}[]\}$  and  $\phi_0 = (\text{init}[])$ .

<sup>2</sup><http://markryan.eu/research/StatVerif/>

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

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

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

$$\begin{aligned} & \text{message}(\text{init}[], c[], \text{left}[]) \wedge \text{message}(\text{init}[], yc, ym) \wedge \\ & \quad \text{message}(\text{init}[], zc, zm) \rightarrow \text{message}(\text{left}[], zc, zm) \\ & \text{message}(\text{init}[], c[], \text{left}[]) \wedge \text{message}(\text{init}[], yc, ym) \wedge \\ & \quad \text{attacker}(\text{init}[], zm) \rightarrow \text{attacker}(\text{left}[], zm) \end{aligned}$$

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

$$\begin{aligned} & \text{message}(\text{init}[], c[], \text{right}[]) \wedge \text{message}(\text{init}[], yc, ym) \wedge \\ & \quad \text{message}(\text{init}[], zc, zm) \rightarrow \text{message}(\text{right}[], zc, zm) \\ & \text{message}(\text{init}[], c[], \text{right}[]) \wedge \text{message}(\text{init}[], yc, ym) \wedge \\ & \quad \text{attacker}(\text{init}[], zm) \rightarrow \text{attacker}(\text{right}[], zm) \end{aligned}$$

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

$$\begin{aligned} & \text{message}(\text{left}[], c[], \text{aenc}(\text{pk}(k[]), xr, (xsl, xsr))) \wedge \\ & \quad \text{message}(\text{left}[], yc, ym) \rightarrow \text{message}(\text{left}[], c[], xsl) \end{aligned}$$

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

$$\begin{aligned} & \text{message}(\text{right}[], c[], \text{aenc}(\text{pk}(k[]), xr, (xsl, xsr))) \wedge \\ & \quad \text{message}(\text{right}[], yc, ym) \rightarrow \text{message}(\text{right}[], c[], xsr) \end{aligned}$$

The output on line 15 is translated to:

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

### 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  $m$ .

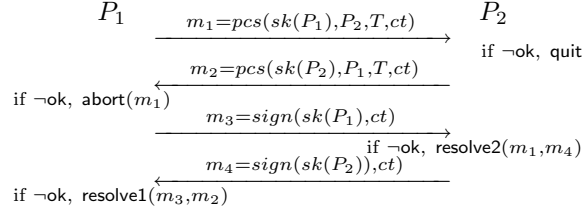


Figure 5: The GM Main protocol (see [12])

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*, *resolved1* and *resolved2*.
- *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  $m = \text{pcs}(sk(x), y, T, ct)$ , it acknowledges the request with the message  $\text{sign}(skT, m)$ . It then updates the status of *ct* to *aborted* and stores its decision  $\text{sign}(skT, m)$ ;
- if it is a resolution request including the promise  $\text{pcs}(sk(x), y, T, ct)$  and the signature  $\text{sign}(sk(y), ct)$ , it converts *x*'s promise into a valid signature  $\text{sign}(sk(x), ct)$  and replies with the message  $\text{sign}(skT, (\text{sign}(sk(x), ct), \text{sign}(sk(y), ct)))$ . 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 ↦ init] | [sigs ↦ 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

$$\text{query attacker}(xs, (\text{abort}C, \text{resolve}C))$$

where  $\text{abort}C = \text{sign}(skT, \text{pcs}(skP_1, \text{pk}(skP_2), \text{pk}(skT), ct))$  is the abortion acknowledgement, and  $\text{resolve}C = \text{sign}(skT, (\text{sign}(skP_1, ct), \text{sign}(skP_2, ct)))$  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:

$$\begin{aligned} &^1 \text{ let } T' = \text{new } skT; (\text{out}(c, \text{pk}(skT)) \mid C \mid ! C') \\ &^2 \text{ let } C' = \text{new } ct'; \text{out}(c, ct'); \text{in}(c, xpk1); \text{in}(c, xpk2); \\ &^3 \quad \quad \quad (! \text{Abort1}' \mid ! \text{Resolve2}' \mid ! \text{Resolve1}') \end{aligned}$$

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

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

*Proof sketch.* Let  $\text{attach} \in \text{Init}$  and let  $A$  be an  $\text{Init}$ -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  $\text{Abort1}'$ ,  $\text{Resolve2}'$ , and  $\text{Resolve1}'$ , the following holds: for any trace  $tr$  of  $T$  such that

$$\begin{aligned} &(\text{fn}(T) \cup \text{fn}(A), \emptyset, \{(T \mid A, \text{false})\}) \rightarrow^* \\ &(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\text{out}(\text{attach}, M); Q, \text{false})\}) \end{aligned}$$

there exists a trace  $tr'$  of  $T'$

$$\begin{aligned} &(\text{fn}(T') \cup \text{fn}(A), \emptyset, \{(T' \mid A, \text{false})\}) \rightarrow^* \\ &(\mathcal{E}', \mathcal{S}', \mathcal{P}' \cup \{(\text{out}(\text{attach}, M); Q', \text{false})\}) \end{aligned}$$

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

$$\begin{aligned} &(\text{fn}(T) \cup \text{fn}(A), \emptyset, \{(T \mid A, \text{false})\}) \rightarrow^* \\ &(\mathcal{E}, \mathcal{S}, \mathcal{P} \cup \{(\text{out}(\text{attach}, M); Q, \text{false})\}) \end{aligned}$$

with  $M = (\text{abort}C, \text{resolve}C)$ . But then by 2) a trace of  $T' \mid A$  breaking fairness also exists.  $\square$

### 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<sup>3</sup>.

## 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<sup>3</sup>. 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*

---

<sup>3</sup><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     [ $status \mapsto \text{init}$ ] | [ $sigs \mapsto \text{init}$ ] |
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:

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

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.*  $ystatus = \text{aborted}$  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.*  $ystatus = \text{init}$  at line 23), it updates the status of  $ct$  to **aborted** (line 24), and stores (line 24) and sends (line 25) the acknowledgement  $\text{sign}(skT, y)$ .

```

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 ysig;
22            out(c, ysig); unlock ystatus, ysig else
23            if ystatus = init then
24              ystatus, ysig := aborted, sign(skT, y);
25              out(c, sign(skT, y)); unlock ystatus, ysig )

```

**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:

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

Upon receipt of such a command (line 27), the trusted party executes the sub-process 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.*  $ystatus = \text{resolved2}$  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.*  $ystatus = \text{init}$  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  $\text{sign}(skT, (ysig1, ysig2))$ .

```

26 let Resolve2 =
27   lock status, sigs; in(c, x);
28   let xcmd = projl(x) in
29   if xcmd = resolve2 then
30     let y = projr(x) in
31     let ypcs1 = projl(y) in
32     let ysig2 = projr(y) in
33     let ycontract = check_getmsg(xpk2, ysig2) in
34     if ycontract = ct then
35       let ycheck = checkpcs(ct, xpk1, xpk2, pk(skT), ypcs1) in
36       if ycheck = ok then
37         read status as ystatus;
38         ( if ystatus = resolved2 then
39           read sigs as ysigs;
40           out(c, ysigs); unlock status, sigs else
41         if ystatus = init then
42           let ysig1 = convertpcs(skT, ypcs1) in
43           status, sigs := resolved2, sign(skT, (ysig1, ysig2));
44           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.

```

45 let Resolve1 =
46   lock status, sigs; in(c, x);
47   let xcmd = projl(x) in
48   if xcmd = resolve1 then
49     let y = projr(x) in
50     let ysig1 = projl(y) in
51     let ypcs2 = projr(y) in
52     let ycontract = check_getmsg(xpk1, ysig1) in
53     if ycontract = ct then
54       let ycheck = checkpcs(ct, xpk2, xpk1, pk(skT), ypcs2) in
55       if ycheck = ok then
56         read status as ystatus;
57         ( if ystatus = resolved1 then
58           read sigs as ysigs;
59           out(c, ysigs); unlock status, sigs else
60         if ystatus = init then
61           let ysig2 = convertpcs(skT, ypcs2) in
62           status, sigs := resolved1, sign(skT, (ysig1, ysig2));
63           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  $C'$ .

```

1 let T' = new skT; new status; new sigs; (out(c, pk(skT)) | ! C' | C'')
2 let C' = [status ↦ init] | [sigs ↦ init] |
3         Abort1 [pk(skA)/xpk1, pk(skB)/xpk2] |
4         Resolve2 [pk(skA)/xpk1, pk(skB)/xpk2] |
5         Resolve1 [pk(skA)/xpk1, pk(skB)/xpk2]
6 let C'' = new ct'; out(c, ct') |
7          ! Abort1' | ! Resolve2' | ! 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.

```

8 let Abort1' =
9   lock status, sigs; in(c, x);
10  let xcmd = projl(x) in
11  if xcmd = abort then
12    let y = projr(x) in
13    let yl = projl(y) in
14    let ycontract = projl(yl) in
15    let yparties = projr(yl) in
16    if yparties = (xpk1, xpk2) then
17      if ycontract = ct then
18        let ysig = projr(y) in
19        let ym = check_getmsg(xpk1, ysig) in
20        if ym = yl then
21          out(c, sign(skT, y)); unlock ystatus, ysig

```

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.

```

22 let Resolve2' =
23   lock status, sigs; in(c, x);
24   let xcmd = projl(x) in
25   if xcmd = resolve2 then
26     let y = projr(x) in
27     let ypcs1 = projl(y) in
28     let ysig2 = projr(y) in
29     let ycontract = check_getmsg(xpk2, ysig2) in
30     if ycontract = ct then
31       let ycheck = checkpcs(ct, xpk1, xpk2, pk(skT), ypcs1) in
32       if ycheck = ok then
33         let ysig1 = convertpcs(skT, ypcs1) in
34         out(c, sign(skT, (ysig1, ysig2))); 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.



```

35 let Resolve1' =
36   lock status, sigs; in(c, x);
37   let xcmd = projl(x) in
38   if xcmd = resolve1 then
39     let y = projr(x) in
40     let ysig1 = projl(y) in
41     let ypcs2 = projr(y) in
42     let ycontract = check_getmsg(xpk1, ysig1) in
43     if ycontract = ct then
44       let ycheck = checkpcs(ct, xpk2, xpk1, pk(skT), ypcs2) in
45       if ycheck = ok then
46         let ysig2 = convertpcs(skT, ypcs2) in
47         out(c, sign(skT, (ysig1, ysig2))); unlock status, sigs

```

## B Correctness

Let  $P'_0 = \text{new } \tilde{m}([s_1 \mapsto M_1] \mid \cdots \mid [s_n \mapsto M_n] \mid P_0)$  be a closed process and  $A$  an *Init*-adversary *s.t.*  $\text{attach} \in \text{Init}$ . 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 variables 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  $\mathcal{S}_0 = \{s_1 \mapsto M_1, \dots, s_n \mapsto M_n\}$ . Let  $\mathcal{E}_0$  be the environment such that

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

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

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

$$\begin{aligned}
& \text{attacker}(\overline{\mathcal{R}}, \text{attach}[]) \in \mathcal{F}_0 \\
& \wedge \forall M, N \text{ message}(\overline{\mathcal{R}}, M, N) \in \mathcal{F}_0 \Rightarrow \text{message}(\overline{\mathcal{S}}, M, N) \in \mathcal{F}_0 \\
& \wedge \forall M \text{ attacker}(\overline{\mathcal{R}}, M) \in \mathcal{F}_0 \Rightarrow \text{attacker}(\overline{\mathcal{S}}, M) \in \mathcal{F}_0
\end{aligned}$$

### B.1 Preliminaries

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

- (i)  $\forall a \in \text{fn}(M), \text{attacker}(\overline{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0$ ; and
- (ii)  $\forall x \in \text{fv}(M), \text{attacker}(\overline{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0$ .

Then

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

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

**Base case** ( $d = 1$ ).

$\Rightarrow$   $M$  is a name or a variable

$\stackrel{\text{Def.}}{\Rightarrow} M \in \text{fn}(M) \cup \text{fv}(M)$

$\stackrel{\text{Hyp.}}{\Rightarrow} \text{attacker}(\overline{\mathcal{S}}, \mathcal{E}(M)) \in \mathcal{F}_0$

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

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

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

Thus for all  $i \in \{1, \dots, n\}$ ,  $\text{attacker}(\overline{\mathcal{S}}, \mathcal{E}(M_i)) \in \mathcal{F}_0$ . Now by definition,  $\mathcal{C}_0$  contains the following clause:

$$\text{attacker}(xs, xm_1) \wedge \dots \wedge \text{attacker}(xs, xm_n) \rightarrow \text{attacker}(xs, f(xm_1, \dots, xm_n))$$

Thus, by resolution we have that

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

**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),  $Q, \iota, \lambda$  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

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

Now, if  $N = x$ , then

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\begin{array}{l}
\text{Hyp} \\
\Rightarrow \\
(\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash \text{lock } s_{j_1}, \dots, s_{j_m}; Q' \\
\text{bn}(\text{lock } s_{j_1}, \dots, s_{j_m}; Q') \cap \text{fn}(M) = \emptyset \\
x \notin \text{bv}(\text{lock } s_{j_1}, \dots, s_{j_m}; Q') \\
\pi_{\text{lock}, \text{bn}(), \text{bv}()} \\
\Rightarrow \\
\bigwedge \quad (\mathcal{E}', \mathcal{T}, \iota, \lambda \cup \{j_1, \dots, j_m\}) \vdash Q' \\
\{\mathcal{T} \mid \begin{array}{l} \mathcal{S} \leq \mathcal{T} \\ \mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda] \end{array}\} \\
\text{bn}(Q') \cap \text{fn}(M) = \emptyset \\
x \notin \text{bv}(Q') \\
\text{I.H.} \\
\Rightarrow \\
\bigwedge \quad (\mathcal{E}, \mathcal{T}, \iota, \lambda \cup \{j_1, \dots, j_m\}) \vdash Q'\{M/x\} \\
\{\mathcal{T} \mid \begin{array}{l} \mathcal{S} \leq \mathcal{T} \\ \mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda] \end{array}\} \\
\tau_{\text{lock}} \\
\Rightarrow \\
(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \text{lock } s_{j_1}, \dots, s_{j_m}; Q'\{M/x\} (= Q\{M/x\})
\end{array}$$

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

$$\begin{array}{l}
\text{Hyp} \\
\Rightarrow \\
(\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash \text{unlock } s_{j_1}, \dots, s_{j_m}; Q' \\
\text{bn}(\text{unlock } s_{j_1}, \dots, s_{j_m}; Q') \cap \text{fn}(M) = \emptyset \\
x \notin \text{bv}(\text{unlock } s_{j_1}, \dots, s_{j_m}; Q') \\
\pi_{\text{lock}, \text{bn}(), \text{bv}()} \\
\Rightarrow \\
\bigwedge \quad (\mathcal{E}', \mathcal{T}, \iota, \lambda \setminus \{j_1, \dots, j_m\}) \vdash Q' \\
\{\mathcal{T} \mid \begin{array}{l} \mathcal{S} \leq \mathcal{T} \\ \mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda] \end{array}\} \\
\text{bn}(Q') \cap \text{fn}(M) = \emptyset \\
x \notin \text{bv}(Q') \\
\text{I.H.} \\
\Rightarrow \\
\bigwedge \quad (\mathcal{E}, \mathcal{T}, \iota, \lambda \setminus \{j_1, \dots, j_m\}) \vdash Q'\{M/x\} \\
\{\mathcal{T} \mid \begin{array}{l} \mathcal{S} \leq \mathcal{T} \\ \mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda] \end{array}\} \\
\tau_{\text{lock}} \\
\Rightarrow \\
(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \text{unlock } s_{j_1}, \dots, s_{j_m}; Q'\{M/x\} (= Q\{M/x\})
\end{array}$$

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

$$\begin{array}{l}
\text{Hyp} \\
\Rightarrow \\
(\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash s_{j_1}, \dots, s_{j_m} := N_1, \dots, N_m; Q' \\
\text{bn}(s_{j_1}, \dots, s_{j_m} := N_1, \dots, N_m; Q') \cap \text{fn}(M) = \emptyset \\
x \notin \text{bv}(s_{j_1}, \dots, s_{j_m} := N_1, \dots, N_m; Q') \\
\tau_{\text{write}, \text{bn}(), \text{bv}()} \\
\Rightarrow \\
\bigwedge \quad \begin{array}{l} \mathcal{T} \leq \mathcal{T}[j_k \mapsto \mathcal{E}'(N_k) \mid 1 \leq k \leq m] \wedge \\ (\mathcal{E}', \mathcal{T}[j_k \mapsto \mathcal{E}'(N_k) \mid 1 \leq k \leq m], \iota, \lambda) \vdash Q' \end{array} \\
\{\mathcal{T} \mid \begin{array}{l} \mathcal{S} \leq \mathcal{T} \\ \mathcal{T} = \mathcal{T}[j \mapsto \mathcal{S}(j) \mid j \in \lambda] \end{array}\} \\
\text{bn}(Q') \cap \text{fn}(M) = \emptyset \\
x \notin \text{bv}(Q') \\
\text{I.H.} \\
\Rightarrow \\
\bigwedge \quad \begin{array}{l} \mathcal{T} \leq \mathcal{T}[j_k \mapsto \mathcal{E}'(N_k) \mid 1 \leq k \leq m] \wedge \\ (\mathcal{E}, \mathcal{T}[j_k \mapsto \mathcal{E}'(N_k) \mid 1 \leq k \leq m], \\ \iota, \lambda) \vdash Q'\{M/x\} \end{array} \\
\{\mathcal{T} \mid \begin{array}{l} \mathcal{S} \leq \mathcal{T} \\ \mathcal{T} = \mathcal{T}[j \mapsto \mathcal{S}(j) \mid j \in \lambda] \end{array}\} \\
\text{Lem. 5.1} \\
\Rightarrow \\
\bigwedge \quad \begin{array}{l} \mathcal{T} \leq \mathcal{T}[j_k \mapsto \mathcal{E}(N_k\{M/x\}) \mid 1 \leq k \leq m] \wedge \\ (\mathcal{E}, \mathcal{T}[j_k \mapsto \mathcal{E}(N_k\{M/x\}) \mid \\ 1 \leq k \leq m], \iota, \lambda) \vdash Q'\{M/x\} \end{array} \\
\{\mathcal{T} \mid \begin{array}{l} \mathcal{S} \leq \mathcal{T} \\ \mathcal{T} = \mathcal{T}[j \mapsto \mathcal{S}(j) \mid j \in \lambda] \end{array}\} \\
\tau_{\text{write}} \\
\Rightarrow \\
(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash s_{j_1}, \dots, s_{j_m} := N_1\{M/x\}, \dots, N_m\{M/x\}; \\
Q'\{M/x\} \quad (\overset{s_{j_1} \neq x, \dots, s_{j_m} \neq x}{=} Q\{M/x\})
\end{array}$$

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

$$\begin{array}{l}
\begin{array}{l}
\text{Hyp} \\
\Rightarrow
\end{array}
\quad (\mathcal{E}', \mathcal{S}, \iota, \lambda) \vdash \text{read } s_{j_1}, \dots, s_{j_m} \text{ as } y_1, \dots, y_m; Q' \\
\quad \text{bn}(\text{read } s_{j_1}, \dots, s_{j_m} \text{ as } y_1, \dots, y_m; Q') \cap \text{fn}(M) = \emptyset \\
\quad x \notin \text{bv}(\text{read } s_{j_1}, \dots, s_{j_m} \text{ as } y_1, \dots, y_m; Q') \\
\\
\begin{array}{l}
\tau_{\text{read}, \text{bn}(), \text{bv}()} \\
\Rightarrow
\end{array}
\quad \begin{array}{l}
\{\mathcal{T} \mid \frac{\mathcal{S} \leq \mathcal{T}}{\mathcal{T} = \mathcal{T}[j \mapsto \mathcal{S}(j) \mid j \in \lambda]}\} \wedge \\
(\mathcal{E}' \cup \{y_k \mapsto \mathcal{T}(j_k) \mid 1 \leq k \leq m\}, \mathcal{T}, \\
(\mathcal{T}(j_1) :: \dots \mathcal{T}(j_m) :: \iota), \lambda) \vdash Q' \\
\text{bn}(Q') \cap \text{fn}(M) = \emptyset \\
x \notin \text{bv}(Q')
\end{array} \\
\\
\begin{array}{l}
\text{I.H.} \\
\Rightarrow
\end{array}
\quad \begin{array}{l}
\{\mathcal{T} \mid \frac{\mathcal{S} \leq \mathcal{T}}{\mathcal{T} = \mathcal{T}[j \mapsto \mathcal{S}(j) \mid j \in \lambda]}\} \wedge \\
(\mathcal{E} \cup \{y_k \mapsto \mathcal{T}(j_k) \mid 1 \leq k \leq m\}, \mathcal{T}, \\
(\mathcal{T}(j_1) :: \dots \mathcal{T}(j_m) :: \iota), \lambda) \vdash Q'\{M/x\}
\end{array} \\
\\
\begin{array}{l}
\tau_{\text{read}} \\
\Rightarrow
\end{array}
\quad (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \text{read } s_{j_1}, \dots, s_{j_m} \text{ as } y_1, \dots, y_m; \\
\quad Q'\{M/x\} \quad (x \notin \{s_{j_1}, y_1, \dots, s_{j_m}, y_m\}) \quad Q\{M/x\} \quad \square
\end{array}$$

**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),  $\iota$ , and  $\lambda$  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  $\tau_{\text{nil}}$ ,  $(\mathcal{E}, \mathcal{T}, \iota, \lambda) \vdash Q$ .

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

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

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

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

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

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

$$\begin{array}{l}
\begin{array}{l}
\text{Hyp.} \\
\Rightarrow \\
\tau_{\text{lock}}^k
\end{array}
\quad (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \text{lock } s_{j_1}, \dots, s_{j_m}; Q' \\
\forall \mathcal{U} (\mathcal{S} \leq \mathcal{U} \wedge \mathcal{U} = \mathcal{U}[k \mapsto \mathcal{S}(k) \mid k \in \lambda]) \Rightarrow \\
(\mathcal{E}, \mathcal{U}, \iota, \lambda \cup \{j_1, \dots, j_m\}) \vdash Q' \\
\\
\begin{array}{l}
\text{transitivity of } \leq \wedge \\
\mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda] \\
\Rightarrow
\end{array}
\quad \forall \mathcal{U} (\mathcal{T} \leq \mathcal{U} \wedge \mathcal{U} = \mathcal{U}[k \mapsto \mathcal{S}(k) \mid k \in \lambda]) \Rightarrow \\
(\mathcal{E}, \mathcal{U}, \iota, \lambda \cup \{j_1, \dots, j_m\}) \vdash Q' \\
\\
\begin{array}{l}
\mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda] \\
\Rightarrow \\
\tau_{\text{lock}}^k
\end{array}
\quad \forall \mathcal{U} (\mathcal{T} \leq \mathcal{U} \wedge \mathcal{U} = \mathcal{U}[k \mapsto \mathcal{T}(k) \mid k \in \lambda]) \Rightarrow \\
(\mathcal{E}, \mathcal{U}, \iota, \lambda \cup \{j_1, \dots, j_m\}) \vdash Q' \\
(\mathcal{E}, \mathcal{T}, \iota, \lambda) \vdash \text{lock } s_{j_1}, \dots, s_{j_m}; Q' (= Q)
\end{array}$$

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

$$\begin{array}{l}
\begin{array}{l}
\text{Hyp.} \\
\Rightarrow \\
\tau_{\text{unlock}}^k
\end{array}
\quad (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \text{unlock } s_{j_1}, \dots, s_{j_m}; Q' \\
\forall \mathcal{U} (\mathcal{S} \leq \mathcal{U} \wedge \mathcal{U} = \mathcal{U}[k \mapsto \mathcal{S}(k) \mid k \in \lambda]) \Rightarrow \\
(\mathcal{E}, \mathcal{U}, \iota, \lambda \setminus \{j_1, \dots, j_m\}) \vdash Q' \\
\\
\begin{array}{l}
\text{transitivity of } \leq \wedge \\
\mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda] \\
\Rightarrow
\end{array}
\quad \forall \mathcal{U} (\mathcal{T} \leq \mathcal{U} \wedge \mathcal{U} = \mathcal{U}[k \mapsto \mathcal{S}(k) \mid k \in \lambda]) \Rightarrow \\
(\mathcal{E}, \mathcal{U}, \iota, \lambda \setminus \{j_1, \dots, j_m\}) \vdash Q' \\
\\
\begin{array}{l}
\mathcal{T} = \mathcal{T}[k \mapsto \mathcal{S}(k) \mid k \in \lambda] \\
\Rightarrow \\
\tau_{\text{unlock}}^k
\end{array}
\quad \forall \mathcal{U} (\mathcal{T} \leq \mathcal{U} \wedge \mathcal{U} = \mathcal{U}[k \mapsto \mathcal{T}(k) \mid k \in \lambda]) \Rightarrow \\
(\mathcal{E}, \mathcal{U}, \iota, \lambda \setminus \{j_1, \dots, j_m\}) \vdash Q' \\
(\mathcal{E}, \mathcal{T}, \iota, \lambda) \vdash \text{unlock } s_{j_1}, \dots, s_{j_m}; Q' (= Q)
\end{array}$$

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

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

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

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

## 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),  $\iota$  a sequence of patterns, and  $\lambda$  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)  $(\text{bn}(B) \cup \text{bv}(B)) \cap \text{dom}(\mathcal{E}) = \emptyset$ ; and
- (iv)  $\forall a \in \text{fn}(B), \text{attacker}(\overline{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0$ ; and
- (v)  $\forall x \in \text{fv}(B), \text{attacker}(\overline{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0$ ; and
- (vi) for all  $i \in \{1, \dots, n\}$ ,  $i \in \lambda$  if and only if  $B$  is in the scope of a lock  $\dots s_i \dots$  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

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

**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,  $\mathcal{E}(\mathcal{S}_0) \leq \mathcal{S}$ .

$$(iii) \begin{array}{l} \xRightarrow{\text{Def.}} \text{bn}(B_1) \cup \text{bv}(B_1) \subseteq \text{bn}(B) \cup \text{bv}(B) \\ \text{bn}(B_2) \cup \text{bv}(B_2) \subseteq \text{bn}(B) \cup \text{bv}(B) \\ \xRightarrow{\text{Hyp.}} (\text{bn}(B_1) \cup \text{bv}(B_1)) \cap \text{dom}(\mathcal{E}) \subseteq (\text{bn}(B) \cup \text{bv}(B)) \cap \text{dom}(\mathcal{E}) = \emptyset \\ (\text{bn}(B_2) \cup \text{bv}(B_2)) \cap \text{dom}(\mathcal{E}) \subseteq (\text{bn}(B) \cup \text{bv}(B)) \cap \text{dom}(\mathcal{E}) = \emptyset \end{array}$$

$$(iv) \begin{array}{l} \xRightarrow{\text{Def.}} \text{fn}(B_1) \subseteq \text{fn}(B) \xRightarrow{\text{Hyp.}} \forall a \in \text{fn}(B_1). \text{attacker}(\bar{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0 \\ \xRightarrow{\text{Def.}} \text{fn}(B_2) \subseteq \text{fn}(B) \xRightarrow{\text{Hyp.}} \forall a \in \text{fn}(B_2). \text{attacker}(\bar{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0 \end{array}$$

$$(v) \begin{array}{l} \xRightarrow{\text{Def.}} \text{fv}(B_1) \subseteq \text{fv}(B) \xRightarrow{\text{Hyp.}} \forall x \in \text{fv}(B_1). \text{attacker}(\bar{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0 \\ \xRightarrow{\text{Def.}} \text{fv}(B_2) \subseteq \text{fv}(B) \xRightarrow{\text{Hyp.}} \forall x \in \text{fv}(B_2). \text{attacker}(\bar{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0 \end{array}$$

(vi)  $B$  is in the scope of a lock  $\dots s_i \dots$  if and only if  $B_1$  and  $B_2$  are too, so  $B_1, \lambda$  and  $B_2, \lambda$  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

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

and, according to our typing system

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

**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,  $\mathcal{E}(\mathcal{S}_0) \leq \mathcal{S}$ .

$$(iii) \begin{array}{l} \xRightarrow{\text{Def.}} \text{bn}(B') \cup \text{bv}(B') = \text{bn}(B) \cup \text{bv}(B) \\ \xRightarrow{\text{Hyp.}} (\text{bn}(B') \cup \text{bv}(B')) \cap \text{dom}(\mathcal{E}) = (\text{bn}(B) \cup \text{bv}(B)) \cap \text{dom}(\mathcal{E}) = \emptyset \end{array}$$

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

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

(vi)  $B$  is in the scope of a lock  $\dots s_i \dots$  if and only if  $B'$  is too, so  $B', \lambda$  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,  $\mathcal{E}(\mathcal{S}_0) \leq \mathcal{S}$ .

$$(iii) \begin{array}{l} \xRightarrow{\text{Def.}} \text{bn}(B_1) \cup \text{bv}(B_1) \subseteq \text{bn}(B) \cup \text{bv}(B) \\ \text{bn}(B_2) \cup \text{bv}(B_2) \subseteq \text{bn}(B) \cup \text{bv}(B) \\ \xRightarrow{\text{Hyp.}} (\text{bn}(B_1) \cup \text{bv}(B_1)) \cap \text{dom}(\mathcal{E}) \subseteq (\text{bn}(B) \cup \text{bv}(B)) \cap \text{dom}(\mathcal{E}) = \emptyset \\ (\text{bn}(B_2) \cup \text{bv}(B_2)) \cap \text{dom}(\mathcal{E}) \subseteq (\text{bn}(B) \cup \text{bv}(B)) \cap \text{dom}(\mathcal{E}) = \emptyset \end{array}$$

$$(iv) \begin{array}{l} \xRightarrow{\text{Def.}} \text{fn}(B_1) \subseteq \text{fn}(B) \xRightarrow{\text{Hyp.}} \forall a \in \text{fn}(B_1). \text{attacker}(\bar{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0 \\ \xRightarrow{\text{Def.}} \text{fn}(B_2) \subseteq \text{fn}(B) \xRightarrow{\text{Hyp.}} \forall a \in \text{fn}(B_2). \text{attacker}(\bar{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0 \end{array}$$

$$(v) \begin{array}{l} \xRightarrow{\text{Def.}} \text{fv}(B_1) \subseteq \text{fv}(B) \xRightarrow{\text{Hyp.}} \forall x \in \text{fv}(B_1). \text{attacker}(\bar{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0 \\ \xRightarrow{\text{Def.}} \text{fv}(B_2) \subseteq \text{fv}(B) \xRightarrow{\text{Hyp.}} \forall x \in \text{fv}(B_2). \text{attacker}(\bar{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0 \end{array}$$

(vi)  $B$  is in the scope of a lock  $\dots s_i \dots$  if and only if  $B_1$  and  $B_2$  are too, so  $B_1, \lambda$  and  $B_2, \lambda$  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

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

and, according to our typing system

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

**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,  $\mathcal{E}(\mathcal{S}_0) \leq \mathcal{S}$ .

$$(iii) \begin{array}{l} \xRightarrow{\text{Def.}} \text{bn}(B_2) \cup \text{bv}(B_2) \subseteq \text{bn}(B) \cup \text{bv}(B) \\ \xRightarrow{\text{Hyp.}} (\text{bn}(B_2) \cup \text{bv}(B_2)) \cap \text{dom}(\mathcal{E}) \subseteq (\text{bn}(B) \cup \text{bv}(B)) \cap \text{dom}(\mathcal{E}) = \emptyset \end{array}$$

$$(iv) \xRightarrow{\text{Def.}} \text{fn}(B_2) \subseteq \text{fn}(B) \xRightarrow{\text{Hyp.}} \forall a \in \text{fn}(B_2). \text{attacker}(\bar{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0$$

$$(v) \xRightarrow{\text{Def.}} \text{fv}(B_2) \subseteq \text{fv}(B) \xRightarrow{\text{Hyp.}} \forall x \in \text{fv}(B_2). \text{attacker}(\bar{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0$$

(vi)  $B$  is in the scope of a lock  $\dots s_i \dots$  if and only if  $B_2$  is too, so  $B_2, \lambda$  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

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

**Case**  $B = \text{new } a; B'$ . By hypothesis on  $A$ ,  $\text{bn}(A) \cap \text{bn}(P'_0) = \emptyset$ , thus  $a \notin \text{bn}(P'_0)$ . Let  $\mathcal{E}' = \mathcal{E} \cup \{a \mapsto \text{attn}[]\}$ .

(i) By hypothesis,  $\mathcal{E}_0 \subseteq \mathcal{E} \subseteq \mathcal{E} \cup \{a \mapsto \text{attn}[]\} = \mathcal{E}'$ . Moreover, by hypothesis  $\text{bn}(B) \cap \text{dom}(\mathcal{E}) = \emptyset$ , thus  $a \notin \text{dom}(\mathcal{E})$ , 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)  $\begin{array}{l} \xRightarrow{\text{Def.}} \text{bn}(B') \cup \text{bv}(B') \subseteq \text{bn}(B) \cup \text{bv}(B) \\ \xRightarrow{\text{Hyp.}} (\text{bn}(B') \cup \text{bv}(B')) \cap \text{dom}(\mathcal{E}) \subseteq (\text{bn}(B) \cup \text{bv}(B)) \cap \text{dom}(\mathcal{E}) = \emptyset \end{array}$

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

**Case**  $b \neq a$ .  $\begin{array}{l} \xRightarrow{\text{Def.}} b \in \text{fn}(B) \\ \xRightarrow{\text{Hyp.}} \text{attacker}(\overline{\mathcal{S}}, \mathcal{E}(b)) \in \mathcal{F}_0 \\ \xRightarrow{\mathcal{E}'(b)=\mathcal{E}(b)} \text{attacker}(\overline{\mathcal{S}}, \mathcal{E}'(b)) \in \mathcal{F}_0 \end{array}$

**Case**  $b = a$ .  $\begin{array}{l} \xRightarrow{\text{Def.}} \text{attacker}(\overline{\mathcal{E}(\mathcal{S}_0)}, \text{attn}[]) \in \mathcal{C}_0 \\ \xRightarrow{\text{Def.}} \text{attacker}(\overline{\mathcal{E}(\mathcal{S}_0)}, \text{attn}[]) \in \mathcal{F}_0 \\ \xRightarrow{\mathcal{E}(\mathcal{S}_0) \leq \mathcal{S}} \text{attacker}(\overline{\mathcal{S}}, \text{attn}[]) \in \mathcal{F}_0 \\ \xRightarrow{\mathcal{E}'(b)=\text{attn}[]} \text{attacker}(\overline{\mathcal{S}}, \mathcal{E}'(b)) \in \mathcal{F}_0 \end{array}$

(v) Let  $x \in \text{fv}(B')$ .  $\begin{array}{l} \xRightarrow{\text{Def.}} x \in \text{fv}(B) \\ \xRightarrow{\text{Hyp.}} \text{attacker}(\overline{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0 \\ \xRightarrow{\mathcal{E}'(x)=\mathcal{E}(x)} \text{attacker}(\overline{\mathcal{S}}, \mathcal{E}'(x)) \in \mathcal{F}_0 \end{array}$

(vi)  $B$  is in the scope of a lock  $\dots s_i \dots$  if and only if  $B'$  is too, so  $B', \lambda$  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

$$\frac{a \notin \text{bn}(P'_0) \Rightarrow (\mathcal{E} \cup \{a \mapsto \text{attn}[]\}, \mathcal{S}, \iota, \lambda) \vdash B'}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \text{new } a; B'} \tau_{\text{new } A}$$

**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  $g(\mathcal{E}(M_1), \dots, \mathcal{E}(M_k)) \rightarrow M$ . 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,  $\mathcal{E}(\mathcal{S}_0) \leq \mathcal{S}$ .

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

$$\begin{aligned}
(iv) \quad & \stackrel{\text{Def.}}{\Rightarrow} \quad \text{fn}(B_1) \subseteq \text{fn}(B) \quad \stackrel{\text{Hyp.}}{\Rightarrow} \quad \forall a \in \text{fn}(B_1). \text{attacker}(\bar{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0 \\
& \stackrel{\text{Def.}}{\Rightarrow} \quad \text{fn}(B_2) \subseteq \text{fn}(B) \quad \stackrel{\text{Hyp.}}{\Rightarrow} \quad \forall a \in \text{fn}(B_2). \text{attacker}(\bar{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0
\end{aligned}$$

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

$$\begin{aligned}
\text{Case } y \in \text{fv}(B). \quad & \stackrel{\text{Hyp.}}{\Rightarrow} \quad \text{attacker}(\bar{\mathcal{S}}, \mathcal{E}(y)) \in \mathcal{F}_0 \\
& \quad \mathcal{E}(y) \stackrel{=}{=} \mathcal{E}'(y) \quad \Rightarrow \quad \text{attacker}(\bar{\mathcal{S}}, \mathcal{E}'(y)) \in \mathcal{F}_0 \\
& \quad \stackrel{\mathcal{S} \leq \mathcal{T}}{\Rightarrow} \quad \text{attacker}(\bar{\mathcal{T}}, \mathcal{E}'(y)) \in \mathcal{F}_0
\end{aligned}$$

**Case**  $y = x$ . In that case, by construction

$\mathcal{C} = \text{att}(xs, N_1) \wedge \dots \wedge \text{attacker}(xs, N_k) \Rightarrow \text{attacker}(xs, N) \in \mathcal{C}_0$  for some  $N_1, \dots, N_k$  and  $\mathcal{E}(M_1) = N_1\sigma, \dots, \mathcal{E}(M_k) = N_k\sigma, M = N\sigma$  for some  $\sigma$ . Now,

$$\begin{aligned}
& \stackrel{\text{Hyp.}}{\Rightarrow} \quad \forall i \in \{1, \dots, k\} \forall u \in \text{fn}(M) \cup \text{fv}(M_i) \text{attacker}(\bar{\mathcal{S}}, \mathcal{E}(u)) \\
& \stackrel{\text{Lem. 4}}{\Rightarrow} \quad \forall i \in \{1, \dots, k\} \text{attacker}(\bar{\mathcal{S}}, \mathcal{E}(M_i)) \in \mathcal{F}_0 \\
& \stackrel{\mathcal{S} \leq \mathcal{T}}{\Rightarrow} \quad \forall i \in \{1, \dots, k\} \text{attacker}(\bar{\mathcal{T}}, \mathcal{E}(M_i)) \in \mathcal{F}_0 \\
& \sigma \wedge \mathcal{C} \in \mathcal{C}_0 \quad \text{attacker}(\bar{\mathcal{T}}, M) \in \mathcal{F}_0 \\
& \mathcal{E}'(y) \stackrel{=}{=} M \quad \text{attacker}(\bar{\mathcal{T}}, \mathcal{E}'(y)) \in \mathcal{F}_0
\end{aligned}$$

$$\stackrel{\text{Def.}}{\Rightarrow} \quad \text{fv}(B_2) \subseteq \text{fv}(B) \quad \stackrel{\text{Hyp.}}{\Rightarrow} \quad \forall x \in \text{fv}(B_2). \text{attacker}(\bar{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0$$

(vi)  $B$  is in the scope of a lock  $\dots s_i \dots$  if and only if  $B_1$  and  $B_2$  are too, so  $B_1, \lambda$  and  $B_2, \lambda$  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

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

and, according to our typing system

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

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

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

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

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

$$(iv) \quad \stackrel{\text{Def.}}{\Rightarrow} \quad \text{fn}(B') \subseteq \text{fn}(B) \quad \stackrel{\text{Hyp.}}{\Rightarrow} \quad \forall a \in \text{fn}(B'). \text{attacker}(\bar{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0$$

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

(vi)  $B$  is in the scope of a lock  $\dots s_i \dots$  if and only if  $B'$  is too, so  $B', \lambda$  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  $a \in \text{fn}(M) \cup \text{fn}(N)$ ,  $\text{attacker}(\bar{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0$ , because  $\text{fn}(M) \cup \text{fn}(N) \subseteq \text{fn}(B)$ ; and
- for all  $x \in \text{fv}(M) \cup \text{fv}(N)$ ,  $\text{attacker}(\bar{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0$ , because  $\text{fv}(M) \cup \text{fv}(N) \subseteq \text{fv}(B)$ .

Thus according to lemma 4 it is the case that

$$\text{attacker}(\bar{\mathcal{S}}, \mathcal{E}(M)) \in \mathcal{F}_0 \quad \text{and} \quad \text{attacker}(\bar{\mathcal{S}}, \mathcal{E}(N)) \in \mathcal{F}_0$$

And because

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

we have by resolution that

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

Thus, according to our typing system

$$\frac{\text{message}(\bar{\mathcal{S}}, \mathcal{E}(M), \mathcal{E}(N)) \in \mathcal{F}_0 \quad (\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash B'}{(\mathcal{E}, \mathcal{S}, \iota, \lambda) \vdash \text{out}(M, N); B' (= B)} \tau_{out}$$

**Case**  $B = \text{in}(M, x); B'$ . 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  $\text{message}(\bar{\mathcal{T}}, \mathcal{E}(M), N) \in \mathcal{F}_0$ ,  $\mathcal{E}' = \mathcal{E} \cup \{x \mapsto N\}$ , 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.

$$(ii) \quad \begin{array}{l} \stackrel{\text{Hyp.}}{\Rightarrow} \quad \mathcal{E}(\mathcal{S}_0) \leq \mathcal{S} \\ \text{Transitivity of } \leq \quad \Rightarrow \quad \mathcal{E}(\mathcal{S}_0) \leq \mathcal{T} \end{array}$$

$$(iii) \quad \begin{array}{l} \stackrel{\text{Def.}}{\Rightarrow} \quad \text{bn}(B') \cup \text{bv}(B') \subset \text{bn}(B) \cup \text{bv}(B) \\ \stackrel{\text{Hyp.}}{\Rightarrow} \quad (\text{bn}(B') \cup \text{bv}(B')) \cap \text{dom}(\mathcal{E}) \subset (\text{bn}(B) \cup \text{bv}(B)) \cap \text{dom}(\mathcal{E}) = \emptyset \end{array}$$

$$(iv) \quad \text{Let } a \in \text{fn}(B'). \quad \begin{array}{l} \stackrel{\text{Def.}}{\Rightarrow} \quad a \in \text{fn}(B) \\ \stackrel{\text{Hyp.}}{\Rightarrow} \quad \text{attacker}(\bar{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0 \\ \mathcal{E}(a) \stackrel{= \mathcal{E}'(a)}{\Rightarrow} \quad \text{attacker}(\bar{\mathcal{S}}, \mathcal{E}'(a)) \in \mathcal{F}_0 \\ \mathcal{S} \leq \mathcal{T} \quad \Rightarrow \quad \text{attacker}(\bar{\mathcal{T}}, \mathcal{E}'(a)) \in \mathcal{F}_0 \end{array}$$



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

$$\begin{array}{l}
\text{Case } y \in \text{fv}(B). \quad \begin{array}{l} \text{Hyp.} \\ \mathcal{E}(y) = \mathcal{E}'(y) \\ \mathcal{S} \leq \mathcal{T} \end{array} \quad \begin{array}{l} \text{attacker}(\overline{\mathcal{S}}, \mathcal{E}(y)) \in \mathcal{F}_0 \\ \text{attacker}(\overline{\mathcal{S}}, \mathcal{E}'(y)) \in \mathcal{F}_0 \\ \text{attacker}(\overline{\mathcal{T}}, \mathcal{E}'(y)) \in \mathcal{F}_0 \end{array} \\
\\
\text{Case } y = x. \quad \begin{array}{l} \text{Hyp.} \\ \text{Lem. 4} \\ \mathcal{S} \leq \mathcal{T} \end{array} \quad \begin{array}{l} \forall u \in \text{fn}(M) \cup \text{fv}(M) \text{ attacker}(\overline{\mathcal{S}}, \mathcal{E}(u)) \\ \text{attacker}(\overline{\mathcal{S}}, \mathcal{E}(M)) \in \mathcal{F}_0 \\ \text{attacker}(\overline{\mathcal{T}}, \mathcal{E}(M)) \in \mathcal{F}_0 \end{array} \\
\\
\text{message}(xs, xc, xm) \wedge \\
\text{attacker}(xs, xc) \Rightarrow \text{attacker}(xs, xm) \in \mathcal{C}_0 \quad \text{attacker}(\overline{\mathcal{T}}, N) \in \mathcal{F}_0 \\
\Rightarrow \\
\mathcal{E}'(y) = N \quad \text{attacker}(\overline{\mathcal{T}}, \mathcal{E}'(y)) \in \mathcal{F}_0
\end{array}$$

(vi)  $B$  is in the scope of a lock  $\dots s_i \dots$  if and only if  $B'$  is too, so  $B', \lambda$  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

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

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

**Case**  $B = \text{lock } s_{j_1}, \dots, s_{j_m}; B'$ . 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  $\lambda' = \lambda \cup \{j_1, \dots, j_m\}$ .

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

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

$$\begin{array}{l}
\text{(iii)} \quad \begin{array}{l} \text{Def.} \\ \text{Hyp.} \end{array} \quad \begin{array}{l} \text{bn}(B') \cup \text{bv}(B') = \text{bn}(B) \cup \text{bv}(B) \\ (\text{bn}(B') \cup \text{bv}(B')) \cap \text{dom}(\mathcal{E}) = (\text{bn}(B) \cup \text{bv}(B)) \cap \text{dom}(\mathcal{E}) = \emptyset \end{array}
\end{array}$$

$$\begin{array}{l}
\text{(iv)} \text{ Let } a \in \text{fn}(B'). \quad \begin{array}{l} \text{Def.} \\ \text{Hyp.} \\ \mathcal{S} \leq \mathcal{T} \end{array} \quad \begin{array}{l} a \in \text{fn}(B) \\ \text{attacker}(\overline{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0 \\ \text{attacker}(\overline{\mathcal{T}}, \mathcal{E}(a)) \in \mathcal{F}_0 \end{array}
\end{array}$$

$$\begin{array}{l}
\text{(v)} \text{ Let } x \in \text{fv}(B'). \quad \begin{array}{l} \text{Def.} \\ \text{Hyp.} \\ \mathcal{S} \leq \mathcal{T} \end{array} \quad \begin{array}{l} x \in \text{fv}(B) \\ \text{attacker}(\overline{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0 \\ \text{attacker}(\overline{\mathcal{T}}, \mathcal{E}(x)) \in \mathcal{F}_0 \end{array}
\end{array}$$

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

Thus,  $B'$ ,  $\mathcal{E}$ ,  $\mathcal{T}$ ,  $\iota$ , and  $\lambda'$  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

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

**Case**  $B = \text{unlock } s_{j_1}, \dots, s_{j_m}; B'$ . 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  $\lambda' = \lambda \setminus \{j_1, \dots, j_m\}$ .

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

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

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

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

$$\begin{aligned} \text{(v)} \quad \text{Let } x \in \text{fv}(B'). \quad & \xrightarrow{\text{Def.}} x \in \text{fv}(B) \\ & \xrightarrow{\text{Hyp.}} \text{attacker}(\overline{\mathcal{S}}, \mathcal{E}(x)) \in \mathcal{F}_0 \\ & \xrightarrow{\mathcal{S} \leq \mathcal{T}} \text{attacker}(\overline{\mathcal{T}}, \mathcal{E}(x)) \in \mathcal{F}_0 \end{aligned}$$

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

Thus,  $B'$ ,  $\mathcal{E}$ ,  $\mathcal{T}$ ,  $\iota$ , and  $\lambda'$  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

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

**Case**  $B = \text{read } s_{j_1}, \dots, s_{j_m} \text{ as } x_1, \dots, x_k; B'$ . 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  $\iota' = \mathcal{T}(j_1) :: \dots :: \mathcal{T}(j_m) :: \iota$ .

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

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

$$(iii) \quad \begin{array}{l} \xRightarrow{\text{Def.}} \text{bn}(B') \cup \text{bv}(B') \subset \text{bn}(B) \cup \text{bv}(B) \\ \xRightarrow{\text{Hyp.}} (\text{bn}(B') \cup \text{bv}(B')) \cap \text{dom}(\mathcal{E}) \subset (\text{bn}(B) \cup \text{bv}(B)) \cap \text{dom}(\mathcal{E}) = \emptyset \end{array}$$

$$(iv) \quad \text{Let } a \in \text{fn}(B'). \quad \begin{array}{l} \xRightarrow{\text{Def.}} a \in \text{fn}(B) \\ \xRightarrow{\text{Hyp.}} \text{attacker}(\overline{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0 \\ \xRightarrow{\mathcal{E}(a) = \mathcal{E}'(a)} \text{attacker}(\overline{\mathcal{S}}, \mathcal{E}'(a)) \in \mathcal{F}_0 \\ \xRightarrow{\mathcal{S} \leq \mathcal{T}} \text{attacker}(\overline{\mathcal{T}}, \mathcal{E}'(a)) \in \mathcal{F}_0 \end{array}$$

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

$$\text{Case } y \in \text{fv}(B). \quad \begin{array}{l} \xRightarrow{\text{Hyp.}} \text{attacker}(\overline{\mathcal{S}}, \mathcal{E}(y)) \in \mathcal{F}_0 \\ \xRightarrow{\mathcal{E}(y) = \mathcal{E}'(y)} \text{attacker}(\overline{\mathcal{S}}, \mathcal{E}'(y)) \in \mathcal{F}_0 \\ \xRightarrow{\mathcal{S} \leq \mathcal{T}} \text{attacker}(\overline{\mathcal{T}}, \mathcal{E}'(y)) \in \mathcal{F}_0 \end{array}$$

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

$$\begin{aligned} \mathcal{C}_i &= \text{message}((xs_1, \dots, xs_n), xc, xm) \wedge \\ &\quad \text{attacker}((xs_1, \dots, xs_n), \mathcal{E}_0(s_i)) \Rightarrow \\ &\quad \text{attacker}((xs_1, \dots, xs_n), xs_i) \in \mathcal{C}_0 \end{aligned}$$

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

(vi)  $B$  is in the scope of a lock  $\dots s_i \dots$  if and only if  $B'$  is too, so  $B', \lambda$  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

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

**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  $cells(A) \subseteq \text{fn}(P)$ . Thus by construction of  $\mathcal{E}_0$ ,  $cells(A) \subseteq \text{dom}(\mathcal{E}_0)$ , and by construction of  $\mathcal{C}_0$  for all  $i \in \{1, \dots, n\}$

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

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

Moreover,

$$\begin{array}{l} \xRightarrow{\text{Def.}} \text{attacker}(\overline{\mathcal{E}(\mathcal{S}_0)}, \text{attch}[]) \in \mathcal{C}_0 \\ \xRightarrow{\text{Def.}} \text{attacker}(\overline{\mathcal{E}(\mathcal{S}_0)}, \text{attch}[]) \in \mathcal{F}_0 \\ \xRightarrow{\mathcal{E}(\mathcal{S}_0) \leq \mathcal{S}} \text{attacker}(\overline{\mathcal{S}}, \text{attch}[]) \in \mathcal{F}_0 \\ \xRightarrow{S \leq T} \text{attacker}(\overline{\mathcal{T}}, \text{attch}[]) \in \mathcal{F}_0 \end{array}$$

Combining all this we can infer the following

$$\begin{array}{l} \xRightarrow{c_{i_1}, c_{i_2} \in \mathcal{C}_0 \text{ and } S \leq T} \forall K, L \text{ message}(\overline{\mathcal{T}}, K, L) \in \mathcal{F}_0 \Rightarrow \text{message}(\overline{\mathcal{T}'}, K, L) \in \mathcal{F}_0 \\ \forall K \text{ attacker}(\overline{\mathcal{T}}, K) \in \mathcal{F}_0 \Rightarrow \text{attacker}(\overline{\mathcal{T}'}, K) \in \mathcal{F}_0 \\ \text{attacker}(\overline{\mathcal{T}}, \text{attch}[]) \in \mathcal{F}_0 \\ \xRightarrow{\text{Def.}} \mathcal{T} \leq \mathcal{T}' \end{array}$$

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

$$(ii) \quad \begin{array}{l} \xRightarrow{\text{Hyp.}} \mathcal{E}(\mathcal{S}_0) \leq \mathcal{S} \\ \xRightarrow{\text{Transitivity of } \leq} \mathcal{E}(\mathcal{S}_0) \leq \mathcal{T} \\ \xRightarrow{\text{Transitivity of } \leq} \mathcal{E}(\mathcal{S}_0) \leq \mathcal{T}' \end{array}$$

$$(iii) \quad \begin{array}{l} \xRightarrow{\text{Def.}} \text{bn}(B') \cup \text{bv}(B') = \text{bn}(B) \cup \text{bv}(B) \\ \xRightarrow{\text{Hyp.}} (\text{bn}(B') \cup \text{bv}(B')) \cap \text{dom}(\mathcal{E}) = (\text{bn}(B) \cup \text{bv}(B)) \cap \text{dom}(\mathcal{E}) = \emptyset \end{array}$$

$$(iv) \text{ Let } a \in \text{fn}(B'). \quad \begin{array}{l} \xRightarrow{\text{Def.}} a \in \text{fn}(B) \\ \xRightarrow{\text{Hyp.}} \text{attacker}(\overline{\mathcal{S}}, \mathcal{E}(a)) \in \mathcal{F}_0 \\ \xRightarrow{S \leq T} \text{attacker}(\overline{\mathcal{T}}, \mathcal{E}(a)) \in \mathcal{F}_0 \\ \xRightarrow{T \leq T'} \text{attacker}(\overline{\mathcal{T}'}, \mathcal{E}(a)) \in \mathcal{F}_0 \end{array}$$

$$(v) \text{ Let } y \in \text{fv}(B'). \quad \begin{array}{l} \xRightarrow{\text{Def.}} y \in \text{fv}(B) \\ \xRightarrow{\text{Hyp.}} \text{attacker}(\overline{\mathcal{S}}, \mathcal{E}(y)) \in \mathcal{F}_0 \\ \xRightarrow{S \leq T} \text{attacker}(\overline{\mathcal{T}}, \mathcal{E}(y)) \in \mathcal{F}_0 \\ \xRightarrow{T \leq T'} \text{attacker}(\overline{\mathcal{T}'}, \mathcal{E}(y)) \in \mathcal{F}_0 \end{array}$$

(vi)  $B$  is in the scope of a `lock`  $\dots s_i \dots$  if and only if  $B'$  is too, so  $B', \lambda$  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

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

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,  $\text{dom}(\mathcal{E}_0) = \text{fn}(P) \cup \text{fn}(A) \cup \text{cell}P$  and  $(\text{bn}(A) \cup \text{bv}(A)) \cap (\text{fn}(P) \cup \text{fn}(A) \cup \text{cell}P) = \emptyset$ , thus  $(\text{bn}(A) \cup \text{bv}(A)) \cap \text{dom}(\mathcal{E}_0) = \emptyset$ .

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

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

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

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

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

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

### 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  $\sigma, \rho, H, \iota, \phi$ , and  $\lambda$ . We first prove by induction on the size of  $Q$ , that if

(i)  $\rho$  binds all the free names and variables of  $Q, H, \iota$  and  $\phi$ ;

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

(iii)  $\sigma$  is a closed substitution;

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

(v)  $\mathcal{C}_0 \supseteq \llbracket Q \rrbracket \rho H \iota \bar{\phi} \lambda$ ;

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

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

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  $\tau_{nil}$  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,  $\text{fv}(Q_1) \cup \text{fv}(Q_2) = \text{fv}(Q)$  and  $\text{fn}(Q_1) \cup \text{fn}(Q_2) = \text{fn}(Q)$ . Thus if  $\rho$  binds the free names and variables of  $Q$ , it also binds the free names and variables of  $Q_1$  and  $Q_2$ .

$$\begin{aligned} \text{(ii)} \quad & \xRightarrow{\text{Def.}} \quad \text{bn}(Q_1) \cup \text{bv}(Q_1) \subseteq \text{bn}(Q) \cup \text{bv}(Q) \\ & \quad \text{bn}(Q_2) \cup \text{bv}(Q_2) \subseteq \text{bn}(Q) \cup \text{bv}(Q) \\ & \xRightarrow{\text{Hyp.}} \quad (\text{bn}(Q_1) \cup \text{bv}(Q_1)) \cap \text{dom}(\rho) \subseteq (\text{bn}(Q) \cup \text{bv}(Q)) \cap \text{dom}(\rho) = \emptyset \\ & \quad (\text{bn}(Q_2) \cup \text{bv}(Q_2)) \cap \text{dom}(\rho) \subseteq (\text{bn}(Q) \cup \text{bv}(Q)) \cap \text{dom}(\rho) = \emptyset \end{aligned}$$

(iii) By hypothesis  $\sigma$  is a closed substitution.

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

$$\begin{aligned} \text{(v)} \quad \mathcal{C}_0 & \xrightarrow{\text{Hyp.}} \llbracket Q_1 \mid Q_2 \rrbracket \rho H \iota \bar{\phi} \lambda \\ & \stackrel{\text{Def.}}{=} \llbracket Q_1 \rrbracket \rho H \iota \bar{\phi} \lambda \cup \llbracket Q_2 \rrbracket \rho H \iota \bar{\phi} \lambda \end{aligned}$$

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

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

We can thus apply our induction hypothesis to infer that

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

But then according to our type system

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

**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,  $\text{fv}(Q') = \text{fv}(Q)$  and  $\text{fn}(Q') = \text{fn}(Q)$ . Thus if  $\rho$  binds the free names and variables of  $Q$ , it also binds the free names and variables of  $Q'$ .

(ii)  $\xrightarrow{\text{Def.}}$   $\text{bn}(Q') \cup \text{bv}(Q') = \text{bn}(Q) \cup \text{bv}(Q)$   
 $\xrightarrow{\text{Hyp.}}$   $(\text{bn}(Q') \cup \text{bv}(Q')) \cap \text{dom}(\rho) = (\text{bn}(Q) \cup \text{bv}(Q)) \cap \text{dom}(\rho) = \emptyset$

(iii) By hypothesis  $\sigma$  is a closed substitution.

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

(v)  $\mathcal{C}_0 \xrightarrow{\text{Hyp.}} \llbracket !Q' \rrbracket \rho H \iota \bar{\phi} \lambda$   
 $\xrightarrow{\text{Def.}} \llbracket Q' \rrbracket \rho H \iota \bar{\phi} \lambda$

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

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

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

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

**Case  $Q = \text{if } M = N \text{ then } Q_1 \text{ else } Q_2$  with  $M\rho\sigma = N\rho\sigma$ .** Let  $\theta = \text{mgu}(\rho(M), \rho(N))$ . Since  $M\rho\sigma = N\rho\sigma$ , by definition of a most general unifier, there exists  $\sigma'$  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)  $\xrightarrow{\text{Def.}}$   $\text{fn}(Q_1) \cup \text{fv}(Q_1) \cup \text{fn}(\iota') \cup \text{fv}(\iota') \cup \text{fn}(H') \cup \text{fv}(H') \cup \text{fn}(\phi') \cup \text{fv}(\phi') \subseteq$   
 $\text{fn}(Q) \cup \text{fv}(Q) \cup \text{fn}(\iota\theta) \cup \text{fv}(\iota\theta) \cup \text{fn}(H\theta) \cup \text{fv}(H\theta) \cup \text{fn}(\phi\theta) \cup \text{fv}(\phi\theta)$   
 $\xrightarrow{\text{Def.}}$   $\text{fn}(Q_1) \cup \text{fv}(Q_1) \cup \text{fn}(\iota') \cup \text{fv}(\iota') \cup \text{fn}(H') \cup \text{fv}(H') \cup \text{fn}(\phi') \cup \text{fv}(\phi') \subseteq$   
 $\text{fn}(Q) \cup \text{fv}(Q) \cup \text{fn}(\iota) \cup \text{fv}(\iota) \cup \text{fn}(H) \cup \text{fv}(H) \cup \text{fn}(\phi) \cup \text{fv}(\phi) \cup$   
 $\text{fn}(\{M, N\}) \cup \text{fv}(\{M, N\})$

$\xrightarrow{\text{Def.}}$   $\text{fn}(Q_1) \cup \text{fv}(Q_1) \cup \text{fn}(\iota') \cup \text{fv}(\iota') \cup \text{fn}(H') \cup \text{fv}(H') \cup \text{fn}(\phi') \cup \text{fv}(\phi') \subseteq$   
 $\text{fn}(Q) \cup \text{fv}(Q) \cup \text{fn}(\iota) \cup \text{fv}(\iota) \cup \text{fn}(H) \cup \text{fv}(H) \cup \text{fn}(\phi) \cup \text{fv}(\phi)$

$\xrightarrow{\text{Hyp.}}$   $\text{fn}(Q_1) \cup \text{fv}(Q_1) \subseteq \text{dom}(\rho)$

$\text{dom}(\rho) \stackrel{=}{=} \text{dom}(\rho')$   $\xrightarrow{\text{Hyp.}}$   $\text{fn}(Q_1) \cup \text{fv}(Q_1) \subseteq \text{dom}(\rho')$

$\Rightarrow$   $\rho'$  binds the free names and variables of  $Q_1, \iota', H', \phi'$

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

$$\begin{array}{l}
(ii) \quad \stackrel{\text{Def.}}{\Rightarrow} \quad \text{bn}(Q_1) \cup \text{bv}(Q_1) \subseteq \text{bn}(Q) \cup \text{bv}(Q) \\
\stackrel{\text{Def.}}{\Rightarrow} \quad (\text{bn}(Q_1) \cup \text{bv}(Q_1)) \cap \text{dom}(\rho) \subseteq (\text{bn}(Q) \cup \text{bv}(Q)) \cap \text{dom}(\rho) = \emptyset \\
\stackrel{\text{dom}(\rho) = \text{dom}(\rho')}{\Rightarrow} \quad (\text{bn}(Q_1) \cup \text{bv}(Q_1)) \cap \text{dom}(\rho') = \emptyset
\end{array}$$

$$\begin{array}{l}
\stackrel{\text{Def.}}{\Rightarrow} \quad \text{bn}(Q_2) \cup \text{bv}(Q_2) \subseteq \text{bn}(Q) \cup \text{bv}(Q) \\
\stackrel{\text{Def.}}{\Rightarrow} \quad (\text{bn}(Q_2) \cup \text{bv}(Q_2)) \cap \text{dom}(\rho) \subseteq (\text{bn}(Q) \cup \text{bv}(Q)) \cap \text{dom}(\rho) = \emptyset
\end{array}$$

$$\begin{array}{l}
(iii) \quad \stackrel{\text{Hyp.}}{\Rightarrow} \quad \sigma \text{ is closed} \\
\stackrel{\sigma = \theta\sigma'}{\Rightarrow} \quad \sigma' \text{ is closed}
\end{array}$$

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

$$\begin{array}{l}
(v) \quad \mathcal{C}_0 \stackrel{\text{Hyp.}}{\supseteq} \quad \llbracket \text{if } M = N \text{ then } Q_1 \text{ else } Q_2 \rrbracket \rho H \iota \bar{\phi} \lambda \\
\stackrel{\text{Def.}}{=} \quad \llbracket Q_1 \rrbracket (\rho\theta)(H\theta)(\iota\theta)(\bar{\phi}\theta)\lambda \cup \llbracket Q_2 \rrbracket \rho H \iota \bar{\phi} \lambda \\
\stackrel{\text{Def.}}{=} \quad \llbracket Q_1 \rrbracket \rho' H' \iota' \bar{\phi}' \lambda' \cup \llbracket Q_2 \rrbracket \rho H \iota \bar{\phi} \lambda
\end{array}$$

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

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

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

$$\begin{array}{l}
(vii) \quad \begin{array}{l} \stackrel{\text{Hyp.}}{\Rightarrow} \quad \text{attacker}(\bar{\phi}\sigma, \text{attch}[]) \\ \stackrel{\sigma = \theta\sigma'}{\Rightarrow} \quad \text{attacker}(\bar{\phi}\theta\sigma', \text{attch}[]) \\ \stackrel{\phi' = \phi\theta}{\Rightarrow} \quad \text{attacker}(\bar{\phi}'\sigma', \text{attch}[]) \end{array} \quad \Bigg| \quad \stackrel{\text{Hyp.}}{\Rightarrow} \quad \text{attacker}(\bar{\phi}\sigma, \text{attch}[])
\end{array}$$

We can now apply our induction hypothesis to infer that

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

and thus according to our typing system

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



**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)  $\begin{array}{l} \xRightarrow{\text{Def.}} \text{fn}(Q_2) \cup \text{fv}(Q_2) \subseteq \text{fn}(Q) \cup \text{fv}(Q) \\ \xRightarrow{\text{Hyp.}} \text{fn}(Q_2) \cup \text{fv}(Q_2) \subseteq \text{dom}(\rho) \\ \Rightarrow \rho \text{ binds the free names and variables of } Q_2, \iota, H, \phi \end{array}$
- (ii)  $\begin{array}{l} \xRightarrow{\text{Def.}} \text{bn}(Q_2) \cup \text{bv}(Q_2) \subseteq \text{bn}(Q) \cup \text{bv}(Q) \\ \xRightarrow{\text{Def.}} (\text{bn}(Q_2) \cup \text{bv}(Q_2)) \cap \text{dom}(\rho) \subseteq (\text{bn}(Q) \cup \text{bv}(Q)) \cap \text{dom}(\rho) = \emptyset \end{array}$

(iii) By hypothesis,  $\sigma$  is closed.

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

- (v)  $\mathcal{C}_0 \begin{array}{l} \supseteq^{\text{Hyp.}} \llbracket \text{if } M = N \text{ then } Q_1 \text{ else } Q_2 \rrbracket \rho H \iota \bar{\phi} \lambda \\ \supseteq^{\text{Def.}} \llbracket Q_2 \rrbracket \rho H \iota \bar{\phi} \lambda \end{array}$

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

- (vii)  $\xRightarrow{\text{Hyp.}} \text{attacker}(\bar{\phi}\sigma, \text{attch}[])$

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

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

**Case**  $Q = \text{let } x = g(M_1, \dots, M_k) \text{ in } Q_1 \text{ else } Q_2$ . Let  $M$  be a pattern such that  $g(\rho\sigma(M_1), \dots, \rho\sigma(M_k)) \rightarrow M$  using  $g(p_1, \dots, p_k) \rightarrow p \in \text{def}(g)$  with  $\text{fv}(\{p_1, \dots, p_k, p\}) = \{x_1, \dots, x_\ell\}$ . Let  $\sigma' = \{x_i \mapsto y_i \mid 1 \leq i \leq \ell\}$  with  $y_1, \dots, y_\ell$  fresh, and  $\theta = \text{mgu}(g(\rho(M_1), \dots, \rho(M_k)), g(p_1\sigma', \dots, p_k\sigma'))$ . By definition of a most general unifier, there exists  $\sigma''$  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).

$$\begin{aligned}
(i) \quad & \stackrel{\text{Def.}}{\Rightarrow} \text{fn}(Q_1) \cup \text{fv}(Q_1) \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{fn}(Q) \cup \text{fv}(Q) \cup \{x\} \cup \text{fn}(\iota\theta) \cup \text{fv}(\iota\theta) \cup \text{fn}(H\theta) \cup \text{fv}(H\theta) \cup \text{fn}(\phi\theta) \cup \text{fv}(\phi\theta) \\
& \stackrel{\text{Def.}}{\Rightarrow} \text{fn}(Q_1) \cup \text{fv}(Q_1) \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{fn}(Q) \cup \text{fv}(Q) \cup \{x\} \cup \text{fn}(\iota) \cup \text{fv}(\iota) \cup \text{fn}(H) \cup \text{fv}(H) \cup \\
& \text{fn}(\phi) \cup \text{fv}(\phi) \cup \text{fn}(\{M_1, \dots, M_k\}) \cup \text{fv}(\{M_1, \dots, M_k\}) \cup \{y_i \mid 1 \leq i \leq \ell\} \\
& \stackrel{\text{Def.}}{\Rightarrow} \text{fn}(Q_1) \cup \text{fv}(Q_1) \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{fn}(Q) \cup \text{fv}(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) \\
& \stackrel{\text{Hyp.}}{\Rightarrow} \text{fn}(Q_1) \cup \text{fv}(Q_1) \subseteq \text{dom}(\rho) \cup \{x\} \cup \{y_i \mid 1 \leq i \leq \ell\} \\
\text{dom}(\rho') = \text{dom}(\rho) \cup \{x\} \cup \{y_i \mid 1 \leq i \leq \ell\} \quad & \Rightarrow \text{fn}(Q_1) \cup \text{fv}(Q_1) \subseteq \text{dom}(\rho') \\
& \Rightarrow \rho' \text{ binds the free names and variables of } Q_1, \iota', H', \phi' \\
& \stackrel{\text{Def.}}{\Rightarrow} \text{fn}(Q_2) \cup \text{fv}(Q_2) \subseteq \text{fn}(Q) \cup \text{fv}(Q) \\
& \stackrel{\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}$$

$$\begin{aligned}
(ii) \quad & \stackrel{\text{Def.}}{\Rightarrow} \text{bn}(Q_1) \cup \text{bv}(Q_1) \subseteq \text{bn}(Q) \cup \text{bv}(Q) \\
& \stackrel{\text{Def.}}{\Rightarrow} (\text{bn}(Q_1) \cup \text{bv}(Q_1)) \cap \text{dom}(\rho) \subseteq (\text{bn}(Q) \cup \text{bv}(Q)) \cap \text{dom}(\rho) = \emptyset \\
& \begin{array}{l} x \notin \text{bv}(Q_1) \\ y_1, \dots, y_\ell \text{ fresh} \end{array} \Rightarrow (\text{bn}(Q_1) \cup \text{bv}(Q_1)) \cap \text{dom}(\rho') = \emptyset \\
& \stackrel{\text{Def.}}{\Rightarrow} \text{bn}(Q_2) \cup \text{bv}(Q_2) \subseteq \text{bn}(Q) \cup \text{bv}(Q) \\
& \stackrel{\text{Def.}}{\Rightarrow} (\text{bn}(Q_2) \cup \text{bv}(Q_2)) \cap \text{dom}(\rho) \subseteq (\text{bn}(Q) \cup \text{bv}(Q)) \cap \text{dom}(\rho) = \emptyset
\end{aligned}$$

$$\begin{aligned}
(iii) \quad & \stackrel{\text{Hyp.}}{\Rightarrow} \sigma \text{ is closed} \\
& \stackrel{\sigma = \theta\sigma''}{\Rightarrow} \sigma'' \text{ is closed}
\end{aligned}$$

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

$$\begin{aligned}
(v) \quad \mathcal{C}_0 \quad & \stackrel{\text{Hyp.}}{\supseteq} \llbracket \text{let } x = g(M_1, \dots, M_k) \text{ in } Q_1 \text{ else } Q_2 \rrbracket \rho H \iota \bar{\phi} \lambda \\
& \stackrel{\text{Def.}}{=} \llbracket Q_1 \rrbracket \rho' H' \iota' \bar{\phi}' \lambda' \cup \llbracket Q_2 \rrbracket \rho H \iota \bar{\phi} \lambda
\end{aligned}$$

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

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

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

$$\begin{array}{l|l}
(vii) \quad \stackrel{\text{Hyp.}}{\Rightarrow} \text{attacker}(\bar{\phi}\sigma, \text{attch}[]) & \stackrel{\text{Hyp.}}{\Rightarrow} \text{attacker}(\bar{\phi}\sigma, \text{attch}[]) \\
\stackrel{\sigma = \theta\sigma''}{\Rightarrow} \text{attacker}(\bar{\phi}\theta\sigma'', \text{attch}[]) & \\
\stackrel{\phi' = \phi\theta}{\Rightarrow} \text{attacker}(\bar{\phi}'\sigma'', \text{attch}[]) & 
\end{array}$$

We can now apply our induction hypothesis to infer that

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

and thus according to our typing system

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

**Case**  $Q = \text{new } a; Q'$ . 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)

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

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

(iii) By hypothesis,  $\sigma$  is a closed substitution.

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

$$\begin{array}{l}
(v) \quad \mathcal{C}_0 \\
\begin{array}{l}
\text{Hyp.} \\
\text{Def.} \\
\rho' = \rho \cup \{a \mapsto a[\iota]\}
\end{array} \\
\begin{array}{l}
\llbracket \text{new } a; Q' \rrbracket \rho H \iota \phi \lambda \\
\llbracket Q' \rrbracket (\rho \cup \{a \mapsto a[\iota]\}) H \iota \phi \lambda \\
\llbracket Q' \rrbracket \rho' H \iota \phi \lambda
\end{array}
\end{array}$$

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

(vii) By hypothesis  $\text{attacker}(\bar{\phi}\sigma, \text{attach}[]) \in \mathcal{F}_0$ .

We can thus apply our induction hypothesis to infer that

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

But then according to our typing system

$$\frac{a \in \text{bn}(P'_0) \quad (\rho\sigma \cup \{a \mapsto a[\iota]\}, \phi\sigma, \iota\sigma, \lambda) \vdash Q'}{(\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash \text{new } a; Q' (= Q)} \tau_{\text{new}P}$$

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

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

$$(ii) \quad \begin{array}{l} \xRightarrow{\text{Def.}} \text{bn}(Q') \cup \text{bv}(Q') \subseteq \text{bn}(Q) \cup \text{bv}(Q) \\ \xRightarrow{\text{Hyp.}} (\text{bn}(Q') \cup \text{bv}(Q')) \cap \text{dom}(\rho) \subseteq (\text{bn}(Q) \cup \text{bv}(Q)) \cap \text{dom}(\rho) = \emptyset \end{array}$$

(iii) By hypothesis,  $\sigma$  is a closed substitution.

(iv) By definition  $Q$  is under a lock  $\dots s_i \dots$  in  $P_0$  if and only if  $Q'$  is under a lock  $\dots s_i \dots$  in  $P_0$ . Thus  $Q'$  and  $\lambda$  satisfy (iii) because by hypothesis  $Q'$  and  $\lambda$  satisfy it.

$$(v) \quad \mathcal{C}_0 \quad \begin{array}{l} \xRightarrow{\text{Hyp.}} \llbracket \text{out}(K, L); Q' \rrbracket \rho H \iota \bar{\phi} \lambda \\ \xRightarrow{\text{Def.}} \llbracket Q' \rrbracket \rho H \iota \bar{\phi} \lambda \end{array}$$

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

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

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

$$\llbracket Q \rrbracket \rho H \iota \bar{\phi} \lambda = \{H \Rightarrow \text{message}(\bar{\phi}, \rho(K), \rho(L))\} \cup \llbracket Q' \rrbracket \rho H \iota \bar{\phi} \lambda.$$

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

$$\text{message}(\phi\sigma, \rho(K)\sigma, \rho(L)\sigma) \stackrel{\text{fn}(Q) \subseteq \text{dom}(\rho)}{=} \text{message}(\phi\sigma, (\rho\sigma)(K), (\rho\sigma)(L))$$

Finally, according to our typing system

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

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

$$\begin{aligned}
(i) \quad & \stackrel{\text{Def.}}{\Rightarrow} \text{fv}(Q') \cup \text{fn}(Q') \cup \text{fv}(t') \cup \text{fn}(t') \cup \text{fv}(H') \cup \text{fn}(H') \cup \text{fv}(\phi') \cup \text{fn}(\phi') \subseteq \\
& \text{fv}(Q) \cup \{x\} \cup \text{fn}(Q) \cup \text{fv}(t) \cup \text{fn}(t) \cup \text{fv}(H) \cup \text{fn}(H) \cup \\
& \text{fv}(\phi) \cup \text{fn}(\phi) \cup \{vs_j \mid j \notin \lambda\} \\
& \stackrel{\text{Hyp.}}{\Rightarrow} \text{fv}(Q') \cup \text{fn}(Q') \cup \text{fv}(t') \cup \text{fn}(t') \cup \text{fv}(H') \cup \text{fn}(H') \cup \text{fv}(\phi') \cup \text{fn}(\phi') \subseteq \\
& \text{dom}(\rho) \cup \{x\} \cup \{vs_j \mid j \notin \lambda\} \\
\text{dom}(\rho') \supseteq \text{dom}(\rho) \cup \{x\} \cup \{vs_j \mid j \notin \lambda\} & \stackrel{\Rightarrow}{=} \text{fv}(Q') \cup \text{fn}(Q') \cup \text{fv}(t') \cup \text{fn}(t') \cup \text{fv}(H') \cup \text{fn}(H') \cup \\
& \text{fv}(\phi') \cup \text{fn}(\phi') \subseteq \text{dom}(\rho') \\
& \Rightarrow \rho' \text{ binds the free names and variables of } Q', t', H', \phi'
\end{aligned}$$

$$\begin{aligned}
(ii) \quad & \stackrel{\text{Def.}}{\Rightarrow} \text{bn}(Q') \cup \text{bv}(Q') \subseteq \text{bn}(Q) \cup \text{bv}(Q) \\
& \stackrel{\text{Hyp.}}{\Rightarrow} (\text{bn}(Q') \cup \text{bv}(Q')) \cap \text{dom}(\rho) \subseteq (\text{bn}(Q) \cup \text{bv}(Q)) \cap \text{dom}(\rho) = \emptyset \\
& \begin{array}{l} x \notin \text{bv}(Q') \\ vs_1, \dots, vs_n \text{ fresh} \end{array} \stackrel{\Rightarrow}{=} (\text{bn}(Q') \cup \text{bv}(Q')) \cap \text{dom}(\rho') = \emptyset
\end{aligned}$$

$$\begin{aligned}
(iii) \quad & \stackrel{\text{Hyp.}}{\Rightarrow} L, \phi(1), \dots, \psi(n) \text{ are ground} \\
& \stackrel{\text{Def.}}{\Rightarrow} \{x \mapsto L\} \cup \{vs_j \mapsto \psi(j) \mid j \notin \lambda\} \text{ is closed} \\
& \stackrel{\sigma \text{ closed}}{\Rightarrow} \sigma \cup \{x \mapsto L\} \cup \{vs_j \mapsto \psi(j) \mid \lambda\} \text{ is closed} \\
& \sigma' = \sigma \cup \{x \mapsto L\} \cup \{vs_j \mapsto \psi(j) \mid j \notin \lambda\} \stackrel{\Rightarrow}{=} \sigma' \text{ is closed}
\end{aligned}$$

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

$$\begin{aligned}
(v) \quad \mathcal{C}_0 & \stackrel{\text{Hyp.}}{\supseteq} \llbracket \text{in}(K, x); Q' \rrbracket \rho H \iota \bar{\phi} \lambda \\
& \stackrel{\text{Hyp.}}{\equiv} \llbracket Q' \rrbracket (\rho \cup \{x \mapsto x\} \cup \{vs_j \mapsto vs_j \mid j \notin \lambda\}) (H \wedge \\
& \quad \text{message}(\bar{\phi}', \rho(K), x)) (x :: \iota) \bar{\phi}' \lambda \\
& \stackrel{\text{Def.}}{=} \llbracket Q' \rrbracket \rho' H' \iota' \bar{\phi}' \lambda'
\end{aligned}$$

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

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

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

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

$$\begin{aligned}
& \stackrel{\text{Hyp.}}{\Rightarrow} \text{message}(\bar{\psi}, \rho(K) \sigma, L) \text{ is derivable from } \mathcal{C}_0 \\
x \notin \text{fn}(\rho(K)) & \stackrel{vs_1, \dots, vs_n \text{ fresh}}{\Rightarrow} \text{message}(\bar{\psi}, \rho(K) \sigma', L) \text{ is derivable from } \mathcal{C}_0 \\
\sigma'(x) = L & \stackrel{\Rightarrow}{=} \text{message}(\bar{\psi}, \rho(K) \sigma', x \sigma') \text{ is derivable from } \mathcal{C}_0 \\
\psi = \phi' \sigma' & \stackrel{\Rightarrow}{=} \text{message}(\bar{\phi}' \sigma', \rho(K) \sigma', x \sigma') \text{ is derivable from } \mathcal{C}_0 \\
\text{message}(\xi', K', L') = \text{message}(\bar{\phi}', \rho(K), x) & \stackrel{\Rightarrow}{=} \text{message}(\xi' \sigma', K' \sigma', L' \sigma') \text{ is derivable from } \mathcal{C}_0
\end{aligned}$$

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

We can thus apply our induction hypothesis to infer that

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

Thus according to our typing system

$$\frac{\forall \psi \forall L (\phi\sigma \leq \psi \wedge \psi = \psi[j \mapsto \phi\sigma(j) \mid j \in \lambda] \wedge \text{message}(\bar{\psi}, (\rho\sigma)(K), L) \in \mathcal{F}_0 \Rightarrow (\rho\sigma \cup \{x \mapsto L\}, \psi, L :: \iota\sigma, \lambda) \vdash Q')}{(\rho\sigma, \phi\sigma, \iota\sigma, \lambda) \vdash \text{in}(K, x); Q' (= Q)} \text{Tin}$$

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

- (i)  $\stackrel{\text{Def.}}{\Rightarrow} \text{fn}(Q') \cup \text{fv}(Q') \cup \text{fn}(\iota') \cup \text{fv}(\iota') \cup \text{fn}(H') \cup \text{fv}(H') \cup \text{fn}(\phi') \cup \text{fv}(\phi') \subseteq \text{fn}(Q) \cup \text{fv}(Q) \cup \text{fn}(\iota) \cup \text{fv}(\iota) \cup \text{fn}(H) \cup \text{fv}(H) \cup \text{fn}(\phi) \cup \text{fv}(\phi) \cup \{vs_j \mid j \notin \lambda\}$
- $\stackrel{\text{Hyp.}}{\Rightarrow} \text{fn}(Q') \cup \text{fv}(Q') \cup \text{fn}(\iota') \cup \text{fv}(\iota') \cup \text{fn}(H') \cup \text{fv}(H') \cup \text{fn}(\phi') \cup \text{fv}(\phi') \subseteq \text{dom}(\rho) \cup \{vs_j \mid j \notin \lambda\}$
- $\rho' = \text{dom}(\rho) \cup \{vs_j \mapsto vs_j \mid j \notin \lambda\} \stackrel{\text{Hyp.}}{\Rightarrow} \text{fn}(Q') \cup \text{fv}(Q') \subseteq \text{dom}(\rho')$
- (ii)  $\stackrel{\text{Def.}}{\Rightarrow} \text{bn}(Q') \cup \text{bv}(Q') = \text{bn}(Q) \cup \text{bv}(Q)$
- $\stackrel{\text{Hyp.}}{\Rightarrow} (\text{bn}(Q') \cup \text{bv}(Q')) \cap \text{dom}(\rho) = (\text{bn}(Q) \cup \text{bv}(Q)) \cap \text{dom}(\rho) = \emptyset$
- $vs_1, \dots, vs_n \notin \text{bv}(Q') \stackrel{\text{Hyp.}}{\Rightarrow} (\text{bn}(Q') \cup \text{bv}(Q')) \cap \text{dom}(\rho') = \emptyset$
- (iii)  $\stackrel{\text{Hyp.}}{\Rightarrow} \psi(1), \dots, \psi(n)$  are ground
- $\stackrel{\text{Def.}}{\Rightarrow} \{vs_j \mapsto \psi(j) \mid j \notin \lambda\}$  is closed
- $\stackrel{\sigma \text{ closed}}{\Rightarrow} \sigma \cup \{vs_j \mapsto \psi(j) \mid j \notin \lambda\}$  is closed
- $\stackrel{\sigma' = \sigma \cup \{vs_j \mapsto \psi(j) \mid j \notin \lambda\}}{\Rightarrow} \sigma'$  is closed
- (iv) By definition  $Q'$  is under a lock  $\dots s \dots$  if and only if either  $s \in \{s_{j_1}, \dots, s_{j_m}\}$  or  $Q$  is under a lock  $\dots s \dots$ , so condition (iii) is satisfied by construction of  $\lambda'$ .

(v)  $\mathcal{C}_0 \stackrel{\text{Hyp.}}{\supseteq} \llbracket \text{lock } s_{j_1}, \dots, s_{j_m}; Q' \rrbracket \rho H \iota \bar{\phi} \lambda$

$\stackrel{\text{Def.}}{=} \llbracket Q' \rrbracket (\rho \cup \{vs_j \mapsto vs_j \mid j \notin \lambda\}) H \iota$

$(\bar{\phi}[j \mapsto vs_j \mid j \notin \lambda = \text{false}])(\lambda \cup \{j_1, \dots, j_m\})$

$\stackrel{\text{Def.}}{=} \llbracket Q' \rrbracket \rho' H' \iota' \bar{\phi}' \lambda'$

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

$$\begin{array}{l} \xRightarrow{\text{Hyp.}} \\ vs_1, \dots, vs_n \notin \text{fv}(\xi) \cup \text{fv}(K) \cup \text{fv}(L) \end{array} \quad \begin{array}{l} \text{message}(\xi\sigma, K\sigma, L\sigma) \text{ is derivable from } \mathcal{C}_0 \\ \text{message}(\xi\sigma', K\sigma', L\sigma') \text{ is derivable from } \mathcal{C}_0 \end{array}$$

$$\begin{array}{l} \text{(vii)} \quad \xRightarrow{\text{Hyp.}} \quad \text{attacker}(\bar{\phi}\sigma, \text{attch}[]) \in \mathcal{F}_0 \\ \quad \quad \quad \xRightarrow{\phi\sigma \leq \psi} \quad \text{attacker}(\bar{\psi}, \text{attch}[]) \in \mathcal{F}_0 \\ \quad \quad \quad \xRightarrow{\psi = \phi'\sigma'} \quad \text{attacker}(\bar{\phi}'\sigma', \text{attch}[]) \in \mathcal{F}_0 \end{array}$$

We can thus apply our induction hypothesis to infer that

$$\begin{array}{l} \xRightarrow{\text{I.H}} \quad (\rho'\sigma', \phi'\sigma', \iota'\sigma', \lambda') \vdash Q' \\ \xRightarrow{\text{Def.}} \quad (\rho\sigma \cup \{vs_j \mapsto \psi(j) \mid j \notin \lambda\}, \psi, \iota\sigma, \lambda \cup \{j_1, \dots, j_m\}) \vdash Q' \\ \xRightarrow{\text{Lem. 5}} \quad (\rho\sigma, \psi, \iota\sigma, \lambda \cup \{j_1, \dots, j_m\}) \vdash Q'\{vs_j \mapsto \psi(j) \mid j \notin \lambda\} \\ vs_1, \dots, vs_n \notin \text{fv}(Q') \xRightarrow{} \quad (\rho\sigma, \psi, \iota\sigma, \lambda \cup \{j_1, \dots, j_m\}) \vdash Q' \end{array}$$

But then according to our typing system

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

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

$$\begin{array}{l} \text{(i)} \quad \xRightarrow{\text{Def.}} \quad \text{fn}(Q') \cup \text{fv}(Q') \cup \text{fn}(\iota') \cup \text{fv}(\iota') \cup \text{fn}(H') \cup \text{fv}(H') \cup \\ \quad \quad \quad \text{fn}(\phi') \cup \text{fv}(\phi') \subseteq \text{fn}(Q) \cup \text{fv}(Q) \cup \text{fn}(\iota) \cup \text{fv}(\iota) \cup \text{fn}(H) \cup \text{fv}(H) \cup \\ \quad \quad \quad \text{fn}(\phi) \cup \text{fv}(\phi) \cup \{vs_j \mid j \notin \lambda\} \\ \quad \quad \quad \xRightarrow{\text{Hyp.}} \quad \text{fn}(Q') \cup \text{fv}(Q') \cup \text{fn}(\iota') \cup \text{fv}(\iota') \cup \text{fn}(H') \cup \text{fv}(H') \cup \\ \quad \quad \quad \text{fn}(\phi') \cup \text{fv}(\phi') \subseteq \text{dom}(\rho) \cup \{vs_j \mid j \notin \lambda\} \\ \rho' = \text{dom}(\rho) \cup \{vs_j \mapsto vs_j \mid j \notin \lambda\} \xRightarrow{} \quad \text{fn}(Q') \cup \text{fv}(Q') \subseteq \text{dom}(\rho') \\ \text{(ii)} \quad \xRightarrow{\text{Def.}} \quad \text{bn}(Q') \cup \text{bv}(Q') = \text{bn}(Q) \cup \text{bv}(Q) \\ \quad \quad \quad \xRightarrow{\text{Hyp.}} \quad (\text{bn}(Q') \cup \text{bv}(Q')) \cap \text{dom}(\rho) = (\text{bn}(Q) \cup \text{bv}(Q)) \cap \text{dom}(\rho) = \emptyset \\ vs_1, \dots, vs_n \notin \text{bv}(Q') \xRightarrow{} \quad (\text{bn}(Q') \cup \text{bv}(Q')) \cap \text{dom}(\rho') = \emptyset \\ \text{(iii)} \quad \xRightarrow{\text{Hyp.}} \quad \psi(1), \dots, \psi(n) \text{ are ground} \\ \quad \quad \quad \xRightarrow{\text{Def.}} \quad \{vs_j \mapsto \psi(j) \mid j \notin \lambda\} \text{ is closed} \\ \quad \quad \quad \xRightarrow{\sigma \text{ closed}} \quad \sigma \cup \{vs_j \mapsto \psi(j) \mid j \notin \lambda\} \text{ is closed} \\ \sigma' = \sigma \cup \{vs_j \mapsto \psi(j) \mid k \notin \lambda\} \xRightarrow{} \quad \sigma' \text{ is closed} \end{array}$$

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

$$\begin{aligned}
(v) \quad \mathcal{C}_0 &\stackrel{\text{Hyp.}}{\supseteq} \llbracket \text{unlock } s_{j_1}, \dots, s_{j_m}; Q' \rrbracket \rho H \iota \bar{\phi} \lambda \\
&\stackrel{\text{Def.}}{=} \llbracket Q' \rrbracket (\rho \cup \{vs_j \mapsto vs_j \mid j \notin \lambda\}) H \iota \\
&\quad (\bar{\phi}[j \mapsto vs_j \mid j \notin \lambda]) (\lambda \setminus \{j_1, \dots, j_m\}) \\
&\stackrel{\text{Def.}}{=} \llbracket Q' \rrbracket \rho' H' \iota' \bar{\phi}' \lambda'
\end{aligned}$$

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

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

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

We can thus apply our induction hypothesis to infer that

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

But then according to our typing system

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

**Case Q** =  $s_{j_1}, \dots, s_{j_m} := K_1, \dots, K_m; Q'$  Let  $\psi$  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

$$\begin{aligned}
&\llbracket s_{j_1}, \dots, s_{j_m} := K_1, \dots, K_m; Q' \rrbracket \rho H \phi \iota \lambda = \\
&\quad \llbracket Q' \rrbracket (\rho \cup \{vs_j \mapsto vs_j \mid j \notin \lambda\}) H \iota \phi'' \lambda \\
&\quad \cup \{H \wedge \text{message}(\bar{\phi}', wc, wm) \rightarrow \text{message}\{\bar{\phi}'', wc, wm\}\} \quad (= \mathcal{C}_1) \\
&\quad \cup \{H \wedge \text{attacker}(\bar{\phi}', wm) \rightarrow \text{message}\{\bar{\phi}'', wm\}\} \quad (= \mathcal{C}_2)
\end{aligned}$$

for some  $wc, wm$  fresh.

- $\stackrel{\text{Hyp}}{\Rightarrow} \text{attacker}(\bar{\phi} \sigma, \text{attch}[]) \in \mathcal{F}_0$   
 $\stackrel{\phi \sigma \leq \psi}{\Rightarrow} \text{attacker}(\bar{\psi}, \text{attch}[]) \in \mathcal{F}_0$

- Let  $\text{message}(\bar{\psi}, K', L') \in \mathcal{F}_0$ .

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



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

$$\begin{array}{l}
\begin{array}{l}
\text{Hyp.} \\
\Rightarrow \\
vs_1, \dots, vs_n \text{ fresh} \\
\Rightarrow \\
\psi = \phi' \sigma' \\
\Rightarrow \\
\mathcal{C}_2 \subseteq \mathcal{C}_0 \\
\Rightarrow \\
\psi[j_k \mapsto (\rho\sigma)(M_K) \mid 1 \leq k \leq m] = \phi'' \sigma'
\end{array}
\end{array}
\begin{array}{l}
H\sigma \text{ derivable from } \mathcal{C}_0 \wedge \text{attacker}(\bar{\psi}, K') \in \mathcal{F}_0 \\
H\sigma' \text{ derivable from } \mathcal{C}_0 \wedge \text{attacker}(\bar{\psi}, K') \in \mathcal{F}_0 \\
H\sigma' \text{ derivable from } \mathcal{C}_0 \wedge \text{attacker}(\bar{\phi}'\sigma', K') \in \mathcal{F}_0 \\
\text{attacker}(\bar{\phi}''\sigma', K') \in \mathcal{F}_0 \\
\text{attacker}(\bar{\psi}[j_k \mapsto (\rho\sigma)(M_K) \mid 1 \leq k \leq m], K') \in \mathcal{F}_0
\end{array}$$

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

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

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

$$\begin{array}{l}
(ii) \quad \begin{array}{l}
\text{Def.} \\
\Rightarrow \\
\text{bn}(Q') \cup \text{bv}(Q') = \text{bn}(Q) \cup \text{bv}(Q) \\
\text{Hyp.} \\
\Rightarrow \\
vs_1, \dots, vs_m \text{ fresh} \\
\Rightarrow \\
(\text{bn}(Q') \cup \text{bv}(Q')) \cap \text{dom}(\rho) = (\text{bn}(Q) \cup \text{bv}(Q)) \cap \text{dom}(\rho) = \emptyset \\
(\text{bn}(Q') \cup \text{bv}(Q')) \cap \text{dom}(\rho') = \emptyset
\end{array}
\end{array}$$

$$\begin{array}{l}
(iii) \quad \begin{array}{l}
\text{Hyp.} \\
\Rightarrow \\
\psi(1), \dots, \psi(n) \text{ are ground} \\
\text{Def.} \\
\Rightarrow \\
\sigma \text{ closed} \\
\Rightarrow \\
\sigma' = \sigma \cup \{vs_j \mapsto \psi(j) \mid j \notin \lambda\} \\
\Rightarrow \\
\sigma' \text{ is closed}
\end{array}
\end{array}$$

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

$$\begin{array}{l}
(v) \quad \begin{array}{l}
\mathcal{C}_0 \quad \text{Hyp.} \\
\supseteq \\
\llbracket s_{j_1}, \dots, s_{j_m} := K_1, \dots, K_m; Q' \rrbracket \rho H \iota \bar{\phi} \lambda \\
\text{Def.} \\
\supseteq \\
\llbracket Q' \rrbracket \rho' H' \iota' \bar{\phi}'' \lambda'
\end{array}
\end{array}$$

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

$$\begin{array}{l}
(vii) \quad \begin{array}{l}
\text{Hyp.} \\
\Rightarrow \\
\phi\sigma \leq \psi \\
\Rightarrow \\
\psi \leq \phi' \\
\Rightarrow \\
\psi' = \phi'' \sigma' \\
\Rightarrow \\
\text{attacker}(\bar{\phi}\sigma, \text{attch}[]) \in \mathcal{F}_0 \\
\text{attacker}(\bar{\phi}'\sigma, \text{attch}[]) \in \mathcal{F}_0 \\
\text{attacker}(\bar{\phi}''\sigma, \text{attch}[]) \in \mathcal{F}_0 \\
\text{attacker}(\bar{\phi}''\sigma', \text{attch}[]) \in \mathcal{F}_0
\end{array}
\end{array}$$

We can thus apply our inductive hypothesis to infer that

$$\begin{array}{c}
\begin{array}{c}
\text{L.H.} \\
\Rightarrow \\
\rho'\sigma' = \rho\sigma \cup \{vs_j \mapsto vs_j \mid j \notin \lambda\} \\
\phi''\sigma' = \psi[j_k \mapsto (\rho\sigma)(K_k) \mid 1 \leq k \leq m] \\
\Rightarrow
\end{array}
&
(\rho'\sigma', \phi''\sigma', \iota'\sigma', \lambda') \vdash Q' \\
&
(\rho\sigma \cup \{vs_j \mapsto vs_j \mid j \notin \lambda\}, \\
&
\psi[j_k \mapsto (\rho\sigma)(K_k) \mid 1 \leq k \leq m], \iota'\sigma', \lambda') \vdash Q' \\
\\
\text{Lem. 5} \\
\Rightarrow \\
vs_1, \dots, vs_m \notin \text{fv}(Q') \subseteq \text{dom}(\rho) \\
\iota'\sigma' = \iota\sigma \quad \lambda' = \lambda \\
\Rightarrow \\
&
(\rho\sigma, \psi[j_k \mapsto (\rho\sigma)(K_k) \mid 1 \leq k \leq m], \iota'\sigma', \lambda') \vdash \\
&
Q'\{vs_j \mapsto vs_j \mid j \notin \lambda\} \\
&
(\rho\sigma, \psi[j_k \mapsto (\rho\sigma)(K_k) \mid 1 \leq k \leq m], \iota'\sigma', \lambda') \vdash Q' \\
&
(\rho\sigma, \psi[j_k \mapsto (\rho\sigma)(K_k) \mid 1 \leq k \leq m], \iota\sigma, \lambda) \vdash Q'
\end{array}$$

But then according to our typing system

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

**Case Q = read**  $s_{j_1}, \dots, s_{j_m}$  as  $x_1, \dots, x_m; Q'$ . Let  $\psi$  be a state such that  $\phi\sigma \leq \psi$  and  $\psi = \psi[j \mapsto \phi\sigma(j) \mid j \in \lambda]$ . Let  $\rho' = \rho \cup \{vc \mapsto vc, vm \mapsto vm\} \cup \{x_k \mapsto \phi'(j_k) \mid 1 \leq k \leq m\} \cup \{vs_j \mapsto vs_j \mid j \notin \lambda\}$ ,  $\sigma' = \sigma \cup \{vs_j \mapsto \psi(j) \mid j \notin \lambda\} \cup \{vc \mapsto \text{attach}[], vm \mapsto \text{attach}[]\}$ ,  $\iota' = x_1 :: \dots :: x_m :: \iota$ ,  $\phi' = \phi[j \mapsto vs_j \mid j \notin \lambda]$ ,  $H' = H \wedge \text{message}(\overline{\phi'}, vc, vm)$ , and  $\lambda' = \lambda$ , for  $vc, vm, vs_1, \dots, vs_n$  fresh. We show that  $(Q', \sigma', \rho', H', \iota', \phi', \lambda')$  satisfy conditions (i)-(vii).

$$\begin{array}{c}
(i) \quad \begin{array}{c}
\text{Def.} \\
\Rightarrow \\
\text{fv}(Q') \cup \text{fn}(Q') \cup \text{fv}(\iota') \cup \text{fn}(\iota') \cup \text{fv}(H') \cup \text{fn}(H') \cup \\
\text{fv}(\phi') \cup \text{fn}(\phi') \subseteq \text{fv}(Q) \cup \{x_1, \dots, x_m\} \cup \text{fn}(Q) \cup \text{fv}(\iota) \cup \\
\text{fn}(\iota) \cup \text{fv}(H) \cup \text{fn}(H) \cup \{vc, vm\} \cup \text{fv}(\phi) \cup \text{fn}(\phi)\{vs_j \mid j \notin \lambda\} \\
\text{Hyp.} \\
\Rightarrow \\
\text{fv}(Q') \cup \text{fn}(Q') \cup \text{fv}(\iota') \cup \text{fn}(\iota') \cup \text{fv}(H') \cup \\
\text{fn}(H') \cup \text{fv}(\phi') \cup \text{fn}(\phi') \subseteq \text{dom}(\rho) \cup \{x_1, \dots, x_m\} \cup \{vc, vm\} \\
\text{dom}(\rho') = \text{dom}(\rho) \cup \{x_1, \dots, x_m\} \cup \{vs_j \mid j \notin \lambda\} \\
\Rightarrow \\
\text{fv}(Q') \cup \text{fn}(Q') \cup \text{fv}(\iota') \cup \text{fn}(\iota') \cup \text{fv}(H') \cup \text{fn}(H') \cup \\
\text{fv}(\phi') \cup \text{fn}(\phi') \subseteq \text{dom}(\rho')
\end{array}
\end{array}$$

$$\begin{array}{c}
(ii) \quad \begin{array}{c}
\text{Def.} \\
\Rightarrow \\
\text{bn}(Q') \cup \text{bv}(Q') \subseteq \text{bn}(Q) \cup \text{bv}(Q) \\
\text{Hyp.} \\
\Rightarrow \\
(\text{bn}(Q') \cup \text{bv}(Q')) \cap \text{dom}(\rho) \subseteq (\text{bn}(Q) \cup \text{bv}(Q)) \cap \text{dom}(\rho) = \emptyset \\
x_1, \dots, x_m \notin \text{bv}(Q') \\
vc, vm, vs_1, \dots, vs_n \text{ fresh} \\
\Rightarrow \\
(\text{bn}(Q') \cup \text{bv}(Q')) \cap \text{dom}(\rho') = \emptyset
\end{array}
\end{array}$$

$$\begin{array}{c}
(iii) \quad \begin{array}{c}
\text{Hyp.} \\
\Rightarrow \\
L, \psi(1), \dots, \psi(n), \text{attach}[] \text{ are ground} \\
\text{Def.} \\
\Rightarrow \\
\{vs_j \mapsto \psi(j) \mid j \notin \lambda\} \cup \{vc \mapsto \text{attach}[], vm \mapsto \text{attach}[]\} \text{ is closed} \\
\sigma^{\text{closed}} \\
\Rightarrow \\
\sigma \cup \{vs_j \mapsto \psi(j) \mid j \notin \lambda\} \cup \{vc \mapsto \text{attach}[], vm \mapsto \text{attach}[]\} \text{ is closed} \\
\sigma' = \sigma \cup \{vs_j \mapsto \psi(j) \mid j \notin \lambda\} \cup \\
\{vc \mapsto \text{attach}[], vm \mapsto \text{attach}[]\} \\
\Rightarrow \\
\sigma' \text{ is closed}
\end{array}
\end{array}$$

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

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

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

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

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

If  $\text{message}(\xi', K', L') = \text{message}(\bar{\phi}', vc, vm)$ .

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

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

We can thus apply our induction hypothesis to infer that

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

Thus according to our typing system

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

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

(i) Since by hypotheses  $\text{fv}(P'_0) = \emptyset$  and  $\text{fn}(P'_0) \subseteq \text{dom}(\mathcal{E}_0)$  by construction,  $\rho$  binds the free names and variables of  $P_0$ ,  $\iota$ ,  $H$  and  $\phi$ .

- (ii) By construction,  $\text{dom}(\mathcal{E}_0) = \text{fn}(P'_0) \cup \text{cells}(P'_0) \cup \{\text{attach}\}$ , and by hypothesis  $\text{bn}(P'_0) \cap \text{fn}(P'_0) = \emptyset$ . Thus  $(\text{bn}(P_0) \cup \text{bv}(P_0)) \cap \text{dom}(\mathcal{E}_0) = \emptyset$ .
- (iii) By definition  $\sigma$  is a closed substitution.
- (iv)  $P_0$  is not under any lock in  $P_0$ , thus  $\emptyset$  satisfies condition (iii).
- (v) By definition  $\mathcal{C}_0 \supseteq \llbracket P_0 \rrbracket \rho H \iota \bar{\phi} \lambda$ .
- (vi) by definition  $H\sigma = \text{true}$ , and thus  $H\sigma$  can trivially be derived from  $\mathcal{C}_0$ .
- (vii) By construction,  $\text{attacker}(\overline{\mathcal{E}_0(\mathcal{S}_0)}, \text{attach}[]) \in \mathcal{C}_0$ . So in particular, we have that  $\text{attacker}(\overline{\phi\sigma}, \text{attach}[]) \in \mathcal{F}_0$ .

Thus,  $P_0$ ,  $\rho$ ,  $\sigma$ ,  $H$ ,  $\iota$ ,  $\phi$  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  $(\mathcal{E}, \mathcal{S}, \mathcal{Q}) \rightarrow (\mathcal{F}, \mathcal{T}, \mathcal{R})$  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  $\text{cells}(\mathcal{Q}) \subseteq \{s_1, \dots, s_n\}$ . If  $(\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\iota), \lambda) \vdash Q$  for all  $(Q, \iota, \lambda) \in \mathcal{Q}$ , then  $(\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(j), \nu) \vdash R$  for all  $(R, j, \nu) \in \mathcal{R}$ .*

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

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

$$\begin{array}{l} \xRightarrow{\text{Hyp.}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(j), \nu) \vdash R \\ \xRightarrow{\mathcal{F}=\mathcal{E} \ \mathcal{T}=\mathcal{S}} (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(j), \nu) \vdash R \end{array}$$

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

$$\begin{array}{l} \text{If } (R, j, \nu) \in \mathcal{Q}'. \quad \begin{array}{l} \xRightarrow{\text{Hyp.}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(j), \nu) \vdash R \\ \xRightarrow{\mathcal{F}=\mathcal{E} \ \mathcal{T}=\mathcal{S}} (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(j), \nu) \vdash R \end{array} \\ \\ \text{If } (Q, j, \nu) = (R, \iota, \lambda). \quad \begin{array}{l} \xRightarrow{\text{Hyp.}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\iota), \lambda) \vdash !Q \\ \xRightarrow{\tau_{\text{repl}}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\iota), \lambda) \vdash Q \\ \xRightarrow{\mathcal{F}=\mathcal{E} \ \mathcal{T}=\mathcal{S} \ \ j=\iota \ \ \nu=\lambda \ \ R=Q} (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(j), \nu) \vdash R \end{array} \end{array}$$

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

$$\begin{array}{l} \text{If } (R, j, \nu) \in \mathcal{Q}'. \quad \begin{array}{l} \xRightarrow{\text{Hyp.}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(j), \nu) \vdash R \\ \xRightarrow{\mathcal{F}=\mathcal{E} \ \mathcal{T}=\mathcal{S}} (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(j), \nu) \vdash R \end{array} \end{array}$$

$$\begin{array}{lcl}
\text{If } (R, j, \nu) = (Q_1, i, \lambda). & \begin{array}{l} \text{Hyp.} \\ \xRightarrow{\tau_{par}} \\ \mathcal{F}=\mathcal{E} \quad T=\mathcal{S} \quad j=i \quad \nu=\lambda \quad R=Q_1 \end{array} & \begin{array}{l} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(i), \lambda) \vdash Q_1 \mid Q_2 \\ (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(i), \lambda) \vdash Q_1 \\ (\mathcal{F}, \mathcal{F}(T), \mathcal{F}(j), \nu) \vdash R \end{array} \\
\text{If } (R, j, \nu) = (Q_2, i, \lambda). & \begin{array}{l} \text{Hyp.} \\ \xRightarrow{\tau_{par}} \\ \mathcal{F}=\mathcal{E} \quad T=\mathcal{S} \quad j=i \quad \nu=\lambda \quad R=Q_2 \end{array} & \begin{array}{l} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(i), \lambda) \vdash Q_1 \mid Q_2 \\ (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(i), \lambda) \vdash Q_2 \\ (\mathcal{F}, \mathcal{F}(T), \mathcal{F}(j), \nu) \vdash R \end{array}
\end{array}$$

**Case R = Red New 1.** In this case,  $\mathcal{F} = \mathcal{E} \cup \{a' \mapsto a[\mathcal{E}(i)]\}$  where  $a'$  is fresh,  $a \in \text{bn}(P'_0)$ ,  $T = \mathcal{S}$ ,  $\mathcal{Q} = \mathcal{Q}' \cup \{(\text{new } a; Q, i, \lambda)\}$ , and  $\mathcal{R} = \mathcal{Q}' \cup \{(Q\{a'/a\}, i, \lambda)\}$ . Let  $(R, j, \nu) \in \mathcal{R}$ , then either  $(R, j, \nu) \in \mathcal{Q}'$  or  $(R, j, \nu) = (Q\{a'/a\}, i, \lambda)$ .

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

$$\begin{array}{lcl}
& \text{Hyp.} & (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(j), \nu) \vdash R \\
& \xRightarrow{a' \text{ fresh}} & (\mathcal{E}, \mathcal{F}(\mathcal{S}), \mathcal{F}(j), \nu) \vdash R \\
& \xRightarrow{a \notin \text{fn}(R)} & (\mathcal{E}, \mathcal{F}(\mathcal{S}), \mathcal{F}(j), \nu) \vdash R\{a'/a\} \\
& \xRightarrow{\text{Lem. 5.2}} & (\mathcal{F}, \mathcal{F}(\mathcal{S}), \mathcal{F}(j), \nu) \vdash R
\end{array}$$

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

$$\begin{array}{lcl}
& \text{Hyp.} & (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(i), \lambda) \vdash \text{new } a; Q' \\
& \xRightarrow{\tau_{newP}} & (\mathcal{E} \cup \{a \mapsto a[\mathcal{E}(i)]\}, \mathcal{E}(\mathcal{S}), \mathcal{E}(i), \lambda) \vdash Q' \\
& \xRightarrow{a' \text{ fresh}} & (\mathcal{E} \cup \{a \mapsto a[\mathcal{E}(i)]\}, \mathcal{F}(\mathcal{S}), \mathcal{F}(i), \lambda) \vdash Q' \\
& \xRightarrow{a' \text{ fresh, } \alpha\text{-renaming}} & (\mathcal{E} \cup \{a' \mapsto a[\mathcal{E}(i)]\}, \mathcal{F}(\mathcal{S}), \mathcal{F}(i), \lambda) \vdash Q'\{a'/a\} \\
& \xRightarrow{\mathcal{F}=\mathcal{E} \cup \{a' \mapsto a[\mathcal{E}(i)]\}} & (\mathcal{F}, \mathcal{F}(\mathcal{S}), \mathcal{F}(i), \lambda) \vdash Q'\{a'/a\} \\
& \xRightarrow{T=\mathcal{S} \quad j=i \quad \nu=\lambda \quad R=Q'\{a'/a\}} & (\mathcal{F}, \mathcal{F}(T), \mathcal{F}(j), \nu) \vdash Q'\{a'/a\}
\end{array}$$

**Case R = Red New 2.** In this case,  $\mathcal{F} = \mathcal{E} \cup \{a' \mapsto \text{attn}[]\}$  where  $a'$  is fresh,  $a \notin \text{bn}(P'_0)$ ,  $T = \mathcal{S}$ ,  $\mathcal{Q} = \mathcal{Q}' \cup \{(\text{new } a; Q, i, \lambda)\}$ , and  $\mathcal{R} = \mathcal{Q}' \cup \{(Q\{a'/a\}, i, \lambda)\}$ . Let  $(R, j, \nu) \in \mathcal{R}$ , then either  $(R, j, \nu) \in \mathcal{Q}'$  or  $(R, j, \nu) = (Q\{a'/a\}, i, \lambda)$ .

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

$$\begin{array}{lcl}
& \text{Hyp.} & (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(j), \nu) \vdash R \\
& \xRightarrow{a' \text{ fresh}} & (\mathcal{E}, \mathcal{F}(\mathcal{S}), \mathcal{F}(j), \nu) \vdash R \\
& \xRightarrow{a \notin \text{fn}(R)} & (\mathcal{E}, \mathcal{F}(\mathcal{S}), \mathcal{F}(j), \nu) \vdash R\{a'/a\} \\
& \xRightarrow{\text{Lem. 5.2}} & (\mathcal{F}, \mathcal{F}(\mathcal{S}), \mathcal{F}(j), \nu) \vdash R
\end{array}$$

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

$$\begin{array}{lcl}
& \text{Hyp.} & (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(i), \lambda) \vdash \text{new } a; Q' \\
& \xRightarrow{\tau_{newP}} & (\mathcal{E} \cup \{a \mapsto \text{attn}[]\}, \mathcal{E}(\mathcal{S}), \mathcal{E}(i), \lambda) \vdash Q' \\
& \xRightarrow{a' \text{ fresh}} & (\mathcal{E} \cup \{a \mapsto \text{attn}[]\}, \mathcal{F}(\mathcal{S}), \mathcal{F}(i), \lambda) \vdash Q' \\
& \xRightarrow{a' \text{ fresh, } \alpha\text{-renaming}} & (\mathcal{E} \cup \{a' \mapsto \text{attn}[]\}, \mathcal{F}(\mathcal{S}), \mathcal{F}(i), \lambda) \vdash Q'\{a'/a\} \\
& \xRightarrow{\mathcal{F}=\mathcal{E} \cup \{a' \mapsto \text{attn}[]\}} & (\mathcal{F}, \mathcal{F}(\mathcal{S}), \mathcal{F}(i), \lambda) \vdash Q'\{a'/a\} \\
& \xRightarrow{T=\mathcal{S} \quad j=i \quad \nu=\lambda \quad R=Q'\{a'/a\}} & (\mathcal{F}, \mathcal{F}(T), \mathcal{F}(j), \nu) \vdash Q'\{a'/a\}
\end{array}$$

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

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

**If  $(R, j, \nu) \in \mathcal{Q}'$ .** First note that  $g(M_1, \dots, M_n) \rightarrow M$ , implies that  $g(\mathcal{E}(M_1), \dots, \mathcal{E}(M_k)) \rightarrow \mathcal{E}(M)$  because the  $M_i$ s only contain variables and constructors.

$$\begin{array}{l} \xRightarrow{\text{Hyp.}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(j), \nu) \vdash R \\ \xRightarrow{\mathcal{F}=\mathcal{E} \quad \mathcal{T}=\mathcal{S}} (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(j), \nu) \vdash R \end{array}$$

**If  $(R, j, \nu) = (Q_1\{M/x\}, \iota, \lambda)$ .** Note first that  $g(\mathcal{E}(M_1), \dots, \mathcal{E}(M_n)) \rightarrow \mathcal{E}(M)$  because the  $M_i$ s only contain variables and constructors.

$$\begin{array}{l} \xRightarrow{\text{Hyp.}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\iota), \lambda) \vdash \text{let } x = g(M_1, \dots, M_n) \text{ in } Q_1 \text{ else } Q_2 \\ \xRightarrow{\tau_{\text{let}}} (\mathcal{E} \cup \{x \mapsto \mathcal{E}(M)\}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\iota), \lambda) \vdash Q_1 \\ \xRightarrow{\text{Lem. 5.2}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\iota), \lambda) \vdash Q_1\{M/x\} \\ \xRightarrow{\mathcal{F}=\mathcal{E} \quad \mathcal{T}=\mathcal{S} \quad j=\iota \quad \nu=\lambda \quad R=Q_1\{M/x\}} (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(j), \nu) \vdash R \end{array}$$

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

$$\begin{array}{l} \text{If } (R, j, \nu) \in \mathcal{Q}'. \quad \xRightarrow{\text{Hyp.}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(j), \nu) \vdash R \\ \quad \quad \quad \xRightarrow{\mathcal{F}=\mathcal{E} \quad \mathcal{T}=\mathcal{S}} (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(j), \nu) \vdash R \end{array}$$

$$\text{If } (R, j, \nu) = (Q_2, \iota, \lambda). \quad \xRightarrow{\text{Hyp.}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\iota), \lambda) \vdash \text{let } x = g(M_1, \dots, M_n) \text{ in } Q_1 \text{ else } Q_2$$

$$\xRightarrow{\tau_{\text{let}}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\iota), \lambda) \vdash Q_2 \\ \xRightarrow{\mathcal{F}=\mathcal{E} \quad \mathcal{T}=\mathcal{S} \quad j=\iota \quad \nu=\lambda \quad R=Q_2} (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(j), \nu) \vdash R$$

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

$$\begin{array}{l} \text{If } (R, j, \nu) \in \mathcal{Q}'. \quad \xRightarrow{\text{Hyp.}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(j), \nu) \vdash R \\ \quad \quad \quad \xRightarrow{\mathcal{F}=\mathcal{E} \quad \mathcal{T}=\mathcal{S}} (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(j), \nu) \vdash R \end{array}$$

$$\text{If } (R, j, \nu) = (Q_1, \iota, \lambda). \quad \xRightarrow{\text{Hyp.}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\iota), \lambda) \vdash \text{if } M = M \text{ then } Q_1 \text{ else } Q_2 \\ \xRightarrow{\tau_{\text{if}}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\iota), \lambda) \vdash Q_1$$

$$\xRightarrow{\mathcal{F}=\mathcal{E} \quad \mathcal{T}=\mathcal{S} \quad j=\iota \quad \nu=\lambda \quad R=Q_1} (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(j), \nu) \vdash R$$

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

$$\begin{array}{l} \text{If } (R, j, \nu) \in \mathcal{Q}'. \quad \xRightarrow{\text{Hyp.}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(j), \nu) \vdash R \\ \quad \quad \quad \xRightarrow{\mathcal{F}=\mathcal{E} \quad \mathcal{T}=\mathcal{S}} (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(j), \nu) \vdash R \end{array}$$

$$\text{If } (R, j, \nu) = (Q_2, \iota, \lambda). \quad \xRightarrow{\text{Hyp.}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\iota), \lambda) \vdash \text{if } M = N \text{ then } Q_1 \text{ else } Q_2 \\ \xRightarrow{\tau_{\text{if}}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\iota), \lambda) \vdash Q_2$$

$$\xRightarrow{\mathcal{F}=\mathcal{E} \quad \mathcal{T}=\mathcal{S} \quad j=\iota \quad \nu=\lambda \quad R=Q_2} (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(j), \nu) \vdash R$$

**Case R = Red I/O.** In this case,  $\mathcal{F} = \mathcal{E}$ ,  $\mathcal{T} = \mathcal{S}$ ,  $\mathcal{Q} = \mathcal{Q}' \cup \{(\text{out}(M, N); Q_1, \iota_1, \lambda_1), (\text{in}(M, x); Q_2, \iota_2, \lambda_2)\}$ , and

$\mathcal{R} = \mathcal{Q}' \cup \{(Q_1, \iota_1, \lambda_1), (Q_2\{N/x\}, (N :: \iota_2), \lambda_2)\}$ .

Let  $(R, j, \nu) \in \mathcal{R}$ , then either  $(R, j, \nu) \in \mathcal{Q}'$  or

$(R, j, \nu) \in \{(Q_1, \iota_1, \lambda_1), (Q_2\{N/x\}, (N :: \iota_2), \lambda_2)\}$ .

If  $(R, j, \nu) \in \mathcal{Q}'$ .  $\xrightarrow{\text{Hyp.}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(j), \nu) \vdash R$   
 $\xrightarrow{\mathcal{F}=\mathcal{E} \Rightarrow \mathcal{T}=\mathcal{S}} (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(j), \nu) \vdash R$

If  $(R, j, \nu) = (Q_1, \iota_1, \lambda_1)$ .  $\xrightarrow{\text{Hyp.}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\iota_1), \lambda_1) \vdash \text{out}(M, N); Q_1$   
 $\xrightarrow{\tau_{\text{out}}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\iota_1), \lambda_1) \vdash Q_1$   
 $\xrightarrow{\mathcal{F}=\mathcal{E} \quad \mathcal{T}=\mathcal{S} \quad j=\iota_1 \quad \nu=\lambda_1 \quad R=Q_1} (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(j), \nu) \vdash R$

If  $(R, j, \nu) = (Q_2\{N/x\}, (N :: \iota_2), \lambda_2)$ . First note that, by hypothesis we have that  $(\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\iota_1), \lambda_1) \vdash \text{out}(M, N); Q_1$ , but then according to the typing rule  $\tau_{\text{out}}$ ,  $\text{message}(\mathcal{E}(\mathcal{S}), \mathcal{E}(M), \mathcal{E}(N)) \in \mathcal{F}_0$ . Moreover

$\xrightarrow{\text{Hyp.}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\iota_2), \lambda_2) \vdash \text{in}(M, x); Q_2$   
 $\xrightarrow{\tau_{\text{in}}} (\mathcal{E} \cup \{x \mapsto \mathcal{E}(N)\}, \mathcal{E}(\mathcal{S}), (\mathcal{E}(N) :: \mathcal{E}(\iota_2)), \lambda_2) \vdash Q_2$   
 $\xrightarrow{\text{Lem. 5.2}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), (\mathcal{E}(N) :: \mathcal{E}(\iota_2)), \lambda_2) \vdash P_2\{N/x\}$   
 $\xrightarrow{\mathcal{E}(N) :: \mathcal{E}(\iota_2) = \mathcal{E}(N :: \iota_2)} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(N :: \iota_2), \lambda_2) \vdash Q_2\{N/x\}$   
 $\xrightarrow{\mathcal{F}=\mathcal{E} \quad \mathcal{T}=\mathcal{S} \quad j=N :: \iota_2 \quad \nu=\lambda_2 \quad R=Q_2\{N/x\}} (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(j), \nu) \vdash R$

**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,

$\mathcal{F} = \mathcal{E}$ ,  $\mathcal{T} = \mathcal{S}$ ,  $\mathcal{Q} = \mathcal{Q}' \cup \{(\text{lock } s_{j_1}, \dots, s_{j_m}; Q, \iota, \lambda)\}$ , and  $\mathcal{R} = \mathcal{Q}' \cup \{(Q, \iota, \lambda \cup \{j_1, \dots, j_m\})\}$ . Let  $(R, j, \nu) \in \mathcal{R}$ , then either  $(R, j, \nu) \in \mathcal{Q}'$  or  $(R, j, \nu) = (Q, \iota, \lambda \cup \{j_1, \dots, j_m\})$ .

If  $(R, j, \nu) \in \mathcal{Q}'$ .  $\xrightarrow{\text{Hyp.}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(j), \nu) \vdash R$   
 $\xrightarrow{\mathcal{F}=\mathcal{E} \Rightarrow \mathcal{T}=\mathcal{S}} (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(j), \nu) \vdash R$

If  $(R, j, \nu) = (Q, \iota, \lambda \cup \{j_1, \dots, j_m\})$ .  
 $\xrightarrow{\text{Hyp.}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\iota), \lambda) \vdash \text{lock } s_{j_1}, \dots, s_{j_m}; Q$   
 $\xrightarrow{\tau_{\text{lock}}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\iota), \lambda \cup \{s_{j_1}, \dots, s_{j_m}\}) \vdash Q$   
 $\xrightarrow{\mathcal{F}=\mathcal{E} \quad \mathcal{T}=\mathcal{S} \quad j=\iota \quad \nu=\lambda \cup \{s_{j_1}, \dots, s_{j_m}\} \quad R=Q} (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(j), \nu) \vdash R$

**Case R = Red Unlock.** In this case,  $\mathcal{F} = \mathcal{E}$ ,  $\mathcal{T} = \mathcal{S}$ ,  $\mathcal{Q} = \mathcal{Q}' \cup \{(\text{unlock } s_{j_1}, \dots, s_{j_m}; Q, \iota, \lambda)\}$ , and  $\mathcal{R} = \mathcal{Q}' \cup \{(Q, \iota, \lambda \setminus \{s_{j_1}, \dots, s_{j_m}\})\}$ . Let  $(R, j, \nu) \in \mathcal{R}$ , then either  $(R, j, \nu) \in \mathcal{Q}'$  or  $(R, j, \nu) = (Q, \iota, \lambda \setminus \{s_{j_1}, \dots, s_{j_m}\})$ .

If  $(R, j, \nu) \in \mathcal{Q}'$ .  $\xrightarrow{\text{Hyp.}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(j), \nu) \vdash R$   
 $\xrightarrow{\mathcal{F}=\mathcal{E} \Rightarrow \mathcal{T}=\mathcal{S}} (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(j), \nu) \vdash R$

If  $(R, j, \nu) = (Q, \iota, \lambda \setminus \{s_{j_1}, \dots, s_{j_m}\})$ .  
 $\xrightarrow{\text{Hyp.}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\iota), \lambda) \vdash \text{unlock } s_{j_1}, \dots, s_{j_m}; Q$   
 $\xrightarrow{\tau_{\text{unlock}}} (\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\iota), \lambda \setminus \{s_{j_1}, \dots, s_{j_m}\}) \vdash Q$   
 $\xrightarrow{\mathcal{F}=\mathcal{E} \quad \mathcal{T}=\mathcal{S} \quad j=\iota \quad \nu=\lambda \setminus \{s_{j_1}, \dots, s_{j_m}\} \quad R=Q} (\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(j), \nu) \vdash R$

**Case R = Red Read.** In this case,  $\mathcal{F} = \mathcal{E}$ ,  $\mathcal{T} = \mathcal{S}$ ,  $\mathcal{Q} = \mathcal{Q}' \cup \{(\text{read } s_{j_1}, \dots, s_{j_m} \text{ as } x_1, \dots, x_m; Q, \iota, \lambda)\}$ , and

$\mathcal{R} = \mathcal{Q}' \cup \{(Q\{\mathcal{S}(j_1)/x_1, \dots, \mathcal{S}(j_m)/x_m\}, (\mathcal{S}(j_1) :: \dots :: \mathcal{S}(j_m) :: \iota), \lambda)\}$ .

Let  $(R, j, \nu) \in \mathcal{R}$ , then either  $(R, j, \nu) \in \mathcal{Q}'$  or

$(R, j, \nu) = (Q\{\mathcal{S}(j_1)/x_1, \dots, \mathcal{S}(j_m)/x_m\}, (\mathcal{S}(j_1) :: \dots :: \mathcal{S}(j_m) :: \iota), \lambda)$ .

If  $(R, j, \nu) \in \mathcal{Q}'$ .  $\xrightarrow{\text{Hyp.}}$   $(\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(j), \nu) \vdash R$   
 $\xrightarrow{\mathcal{F}=\mathcal{E} \ \mathcal{T}=\mathcal{S}}$   $(\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(j), \nu) \vdash R$

If  $(R, j, \nu) = (Q\{\mathcal{S}(s_{j_1})/x_1, \dots, \mathcal{S}(s_{j_m})/x_m\}, (\mathcal{S}(j_1) :: \dots :: \mathcal{S}(j_m) :: \iota), \lambda)$ .

$\xrightarrow{\text{Hyp.}}$   $(\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\iota), \lambda) \vdash \text{read } s_{j_1}, \dots, s_{j_m} \text{ as } x_1, \dots, x_m; Q$

$\xrightarrow{\tau_{read}}$   $(\mathcal{E} \cup \{x_k \mapsto \mathcal{E}(\mathcal{S}(j_k)) \mid 1 \leq k \leq m\},$

$\mathcal{E}(\mathcal{S}), (\mathcal{E}(\mathcal{S}(j_1)) :: \dots :: \mathcal{E}(\mathcal{S}(j_m)) :: \mathcal{E}(\iota), \lambda) \vdash Q$

$\xrightarrow{\text{Lem. 5.2}}$   $(\mathcal{E}, \mathcal{E}(\mathcal{S}), (\mathcal{E}(\mathcal{S}(j_1)) :: \dots :: \mathcal{E}(\mathcal{S}(j_m)) :: \mathcal{E}(\iota), \lambda) \vdash$   
 $Q\{\mathcal{S}(j_k)/x_k \mid 1 \leq k \leq m\}$

$\mathcal{E}(\mathcal{S}(j_1)) :: \dots :: \mathcal{E}(\mathcal{S}(j_m)) :: \mathcal{E}(\iota) = (\mathcal{E}, \mathcal{E}(\mathcal{S}), (\mathcal{E}(\mathcal{S}(j_1)) :: \dots :: \mathcal{S}(j_m) :: \iota), \lambda) \vdash$

$\mathcal{E}(\mathcal{S}(j_1)) :: \dots :: \mathcal{S}(j_m) :: \iota$   
 $\Rightarrow Q\{\mathcal{S}(j_k)/x_k \mid 1 \leq k \leq m\}$

$\xrightarrow{\mathcal{F}=\mathcal{E} \ \mathcal{T}=\mathcal{S} \ j=\mathcal{S}(j_1) :: \dots :: \mathcal{S}(j_m) :: \iota \wedge \nu=\lambda \wedge R=Q\{\mathcal{S}(s_k)/x_k\}}$   $(\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(j), \nu) \vdash R$   
 $\Rightarrow$

**Case R = Red Write.** In this case,  $\mathcal{F} = \mathcal{E}$ ,  $\mathcal{T} = \mathcal{S}[j_k \mapsto M_k \mid 1 \leq k \leq m]$ ,  $Q = \mathcal{Q}' \cup \{(s_{j_1}, \dots, s_{j_m} := M_1, \dots, M_m; Q, \iota, \lambda)\}$ , and  $\mathcal{R} = \mathcal{Q}' \cup \{(Q, \iota, \lambda)\}$ . Let  $(R, j, \nu) \in \mathcal{R}$ , then either  $(R, j, \nu) \in \mathcal{Q}'$  or  $(R, j, \nu) = (Q, \iota, \lambda)$ . Let us first note that by hypothesis having  $(s_{j_1}, \dots, s_{j_m} := M_1, \dots, M_m; Q, \iota, \lambda) \in \mathcal{Q}$  implies  $(\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\iota), \lambda) \vdash s_{j_1}, \dots, s_{j_m} := M_1, \dots, M_m; Q$ . And thus because  $\mathcal{E}(\mathcal{S}) \leq \mathcal{E}(\mathcal{S})$  according to the typing rule  $\tau_{write}$  it is the case that  $\mathcal{E}(\mathcal{S}) \leq \mathcal{E}(\mathcal{T})$ .

If  $(R, j, \nu) \in \mathcal{Q}'$ .  $\xrightarrow{\text{Hyp.}}$   $(\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(j), \nu) \vdash R$

$\xrightarrow{\text{Lem. 6}}$   $(\mathcal{E}, \mathcal{E}(\mathcal{T}), \mathcal{E}(j), \nu) \vdash R$

$\xrightarrow{\mathcal{F}=\mathcal{E}}$   $(\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(j), \nu) \vdash R$

If  $(R, j, \nu) = (Q, \iota, \lambda)$ .  $\xrightarrow{\text{Hyp.}}$   $(\mathcal{E}, \mathcal{E}(\mathcal{S}), \mathcal{E}(\iota), \lambda) \vdash s_{j_1}, \dots, s_{j_m} := M_1, \dots, M_m; Q$

$\xrightarrow{\tau_{write}}$   $(\mathcal{E}, \mathcal{E}(\mathcal{T}), \mathcal{E}(\iota), \lambda) \vdash Q$

$\xrightarrow{\mathcal{F}=\mathcal{E} \ j=\iota \ \nu=\lambda \ R=Q}$   $(\mathcal{F}, \mathcal{F}(\mathcal{T}), \mathcal{F}(j), \nu) \vdash R$

□