A Type System for Privacy Properties (Technical Report)

Véronique Cortier  
CNRS, LORIA  
Nancy, France  
veronique.cortier@loria.fr

Joseph Lallemand  
Inria, LORIA  
Nancy, France  
joseph.lallemand@loria.fr

Niklas Grimm  
TU Wien  
Vienna, Austria  
niklas.grimm@tuwien.ac.at

Matteo Maffei  
TU Wien  
Vienna, Austria  
matteo.maffei@tuwien.ac.at

ABSTRACT
Mature push button tools have emerged for checking trace properties (e.g., secrecy or authentication) of security protocols. The case of indistinguishability-based privacy properties (e.g., ballot privacy or anonymity) is more complex and constitutes an active research topic with several recent propositions of techniques and tools.

We explore a novel approach based on type systems and provide a (sound) type system for proving equivalence of protocols, for a bounded or an unbounded number of sessions. The resulting prototype implementation has been tested on various protocols of the literature. It provides a significant speed-up (by orders of magnitude) compared to tools for a bounded number of sessions and complements in terms of expressiveness other state-of-the-art tools, such as ProVerif and Tamarin: e.g., we show that our analysis technique is the first one to handle a faithful encoding of the Helios e-voting protocol in the context of an untrusted ballot box.

1 INTRODUCTION
Formal methods proved to be indispensable tools for the analysis of advanced cryptographic protocols such as those for key distribution [43], mobile payments [30], e-voting [12, 29, 35], and e-health [38]. In the last years, mature push-button analysis tools have emerged and have been successfully applied to many protocols from the literature in the context of trace properties such as authentication or confidentiality. These tools employ a variety of analysis techniques, such as model checking (e.g., Avispa [10] and Scyther [33]), Horn clause resolution (e.g., ProVerif [19]), term rewriting (e.g., Scyther [33] and Tamarin [40]), and type systems [11, 17, 22, 36].

A current and very active topic is the adaptation of these techniques to the more involved case of trace equivalence properties. These are the natural symbolic counterpart of cryptographic indistinguishability properties, and they are at the heart of privacy properties such as ballot privacy [39], untraceability [7], or anonymity [4, 8]. They are also used to express stronger forms of confidentiality, such as strong secrecy [31], or game-based like properties [27].

Related Work. Numerous model checking-based tools have recently been proposed for the case of a bounded number of sessions, i.e., when protocols are executed a bounded number of times. These tools encompass SPEC [34], APTE [13, 24], Akiss [23], or SAT-Equiv [28]. These tools vary in the class of cryptographic primitives and the class of protocols they can consider. However, due to the complexity of the problem, they all suffer from the state explosion problem and most of them can typically analyse no more than 3-4 sessions of (relatively small) protocols, with the exception of SAT-Equiv which can more easily reach about 10 sessions. The only tools that can verify equivalence properties for an unbounded number of sessions are ProVerif [21], Maude-NPA [42], and Tamarin [16]. ProVerif checks a property that is stronger than trace equivalence, namely diff equivalence, which works well in practice provided that protocols have a similar structure. However, as for trace properties, the internal design of ProVerif renders the tool unable to distinguish between exactly one session and infinitely many: this over-approximation often yields false attacks, in particular when the security of a protocol relies on the fact that some action is only performed once. Maude-NPA also checks diff-equivalence but often does not terminate. Tamarin can handle an unbounded number of sessions and is very flexible in terms of supported protocol classes but it often requires human interactions. Finally, some recent work has started to leverage type systems to enforce relational properties for programs, exploring this approach also in the context of cryptographic protocol implementation [14]: like ProVerif, the resulting tool is unable to distinguish between exactly one session and infinitely many, and furthermore it is only semi-automated, in that it often requires non-trivial lemmas to guide the tool and a specific programming discipline.

Many recent results have been obtained in the area of relational verification of programs using Relational Hoare Logic [15, 18, 44]. The results hold on programs and cannot be directly applied to cryptographic protocols due to the more special treatment of the primitives.

Our contribution. In this paper, we consider a novel type checking-based approach. Intuitively, a type system over-approximates protocol behavior. Due to this over-approximation, it is no longer possible to decide security properties but the types typically convey sufficient information to prove security. Extending this approach to equivalence properties is a delicate task. Indeed, two protocols $P$ and $Q$ are in equivalence if (roughly) any trace of $P$ has an equivalent trace in $Q$ (and conversely). Over-approximating behavior may not preserve equivalence.

Instead, we develop a somewhat hybrid approach: we design a type system to over-approximate the set of possible traces and we collect the set of sent messages into constraints. We then propose a procedure for proving (static) equivalence of the constraints. These do not only contain sent messages but also reflect internal
checks made by the protocols, which is crucial to guarantee that whenever a message is accepted by \( P \), it is also accepted by \( Q \) (and conversely).

As a result, we provide a sound type system for proving equivalence of protocols for both a bounded and an unbounded number of sessions, or a mix of both. This is particularly convenient to analyse systems where some actions are limited (e.g., no revote, or limited access to some resource). More specifically, we show that whenever two protocols \( P \) and \( Q \) are type-checked to be equivalent, then they are in trace equivalence, for the standard notion of trace equivalence \[39\], against a full Dolev-Yao attacker. In particular, one advantage of our approach is that it proves security directly in a security model that is similar to the ones used by the other popular tools, in contrast to many other security proofs based on type systems. Our result holds for protocols with all standard primitives (symmetric and asymmetric encryption, signatures, pairs, hash), with atomic long-term keys (no fresh keys) and no private channels. Similarly to ProVerif, we need the two protocols \( P \) and \( Q \) to have a rather similar structure.

We provide a prototype implementation of our type system, that we evaluate on several protocols of the literature. In the case of a bounded number of sessions, our tool provides a significant speed-up (less than one second to analyze a dozen of sessions while other tools typically do not answer within 12 hours, with a few exceptions). To be fair, let us emphasize that these tools can decide equivalence while our tool checks sufficient conditions by the means of our type system. In the case of an unbounded number of sessions, the performance of our prototype tool is comparable to ProVerif. In contrast to ProVerif, our tool can consider a mix of bounded and unbounded number of sessions. As an application, we can prove for the first time ballot privacy of the well-known Helios e-voting protocol \[6\], without assuming a reliable channel between honest voters and the ballot box. ProVerif fails in this case as ballot privacy only holds under the assumption that honest voters vote at most once, otherwise the protocol is subject to a copy attack \[41\]. For similar reasons, also Tamarin fails to verify this protocol.

In most of our example, only a few straightforward type annotations were needed, such as indicated which keys are supposed to be secret or public. The case of the helios protocol is more involved and requires to describe the form of encrypted ballots that can be sent by a voter.

Our prototype, the protocol models, as well as a technical report are available here \[1\].

## 2 Overview of Our Approach

In this section, we introduce the key ideas underlying our approach on a simplified version of the Helios voting protocol. Helios \[6\] is a verifiable voting protocol that has been used in various elections, including the election of the rector of the University of Louvain-la-Neuve. Its behavior is depicted below:

\[
\begin{align*}
S & \rightarrow V_i : \quad r_i \\
V_i & \rightarrow S : \quad \{ (v_i)^{r_i}_{pk(k_i)} \}_{k_i} \\
S & \rightarrow V_1, \ldots, V_n : \quad v_1, \ldots, v_n
\end{align*}
\]

where \( \{ m \}_{pk(k)} \) denotes the asymmetric encryption of message \( m \) with the key \( pk(k) \) randomized with the nonce \( r \), and \( [m]_{k} \) denotes the signature of \( m \) with key \( k \). \( v_i \) is a value in the set \( \{ 0, 1 \} \), which represents the candidate \( V_i \) votes for. In the first step, the voter casts her vote, encrypted with the election’s public key \( pk(k_i) \) and then signed. Since generating a good random number is difficult for the voter’s client (typically a Javascript run in a browser), a typical trick is to input some randomness \( (r_i) \) from the server and to add it to its own randomness \( (r'_i) \). In the second step the server outputs the tally (i.e., a randomized permutation of the valid votes received in the voting phase). Note that the original Helios protocol does not assume signed ballots. Instead, voters authenticate themselves through a login mechanism. For simplicity, we abstract this authenticated channel by a signature.

A voting protocol provides vote privacy \[35\] if an attacker is not able to know which voter voted for which candidate. Intuitively, this can be modeled as the following trace equivalence property, which requires the attacker not to be able to distinguish A voting 0 and B voting 1 from A voting 1 and B voting 0. Notice that the attacker may control an unbounded number of voters:

\[
\begin{align*}
Voter(k_a, 0) & \quad \text{|} \quad Voter(k_b, 1) \quad \text{|} \quad CompromisedVoters \quad \text{|} \quad S \\
\approx & \\
Voter(k_a, 1) & \quad \text{|} \quad Voter(k_b, 0) \quad \text{|} \quad CompromisedVoters \quad \text{|} \quad S
\end{align*}
\]

Despite its simplicity, this protocol has a few interesting features that make its analysis particularly challenging. First of all, the server is supposed to discard ciphertext duplicates, otherwise a malicious eligible voter \( E \) could intercept \( A \)’s ciphertext, sign it, and send it to the server \[32\], as exemplified below:

\[
\begin{align*}
A \rightarrow S & : \quad \{ (v_a)^{r_a}_{pk(k_a)} \}_{k_a} \\
E \rightarrow S & : \quad \{ (v_a)^{r_a}_{pk(k_a)} \}_{k_a} \\
B \rightarrow S & : \quad \{ (v_b)^{r_b}_{pk(k_b)} \}_{k_b} \\
S \rightarrow A, B & : \quad v_a, v_b, v_a
\end{align*}
\]

This would make the two tallied results distinguishable, thereby breaking trace equivalence since \( v_a, v_b, v_a \not\approx v_b, v_a \).

Even more interestingly, each voter is supposed to be able to vote only once, otherwise the same attack would apply \[41\] even if the server discards ciphertext duplicates (as the randomness used by the voter in the two ballots would be different). This makes the analysis particularly challenging, and in particular out of scope of existing cryptographic protocol analyzers like ProVerif, which abstract away from the number of protocol sessions.

With our type system, we can successfully verify the aforementioned privacy property using the following types:

\[
\begin{align*}
r_a & : \tau_{[r_a]}^{LL, 1} \quad r_b : \tau_{[r_b]}^{LL, 1} \quad r'_a : \tau_{[r'_a]}^{HL, 1} \quad r'_b : \tau_{[r'_b]}^{HL, 1} \\
k_a & : \text{key}_{HL}(\{ [r_0]_{t_1}^{LL, 1} \}) + \text{HL} + \tau_{[r_a]}^{HL, 1} \\
k_b & : \text{key}_{HL}(\{ [r_0]_{t_1}^{LL, 1} \}) + \text{HL} + \tau_{[r_b]}^{HL, 1} \\
k_3 & : \text{key}_{HL}(\{ [r_0]_{t_1}^{LL, 1} \}) + \text{HL} + \tau_{[r'_a]}^{HL, 1} \quad \vee \\
k_5 & : \text{key}_{HL}(\{ [r_0]_{t_1}^{LL, 1} \}) + \text{HL} + \tau_{[r'_b]}^{HL, 1}
\end{align*}
\]
We assume standard security labels: HI stands for high confidentiality and high integrity, HL for high confidentiality and low integrity, and LL for low confidentiality and low integrity (for simplicity, we omit the low confidentiality and high integrity type, since we do not need it in our examples). The type $\tau_i^{r_i, l}$ describes randomness of security label $l$ produced by the randomness generator at position $i$ in the program, which can be invoked at most once. $\tau_i^{r_i, l}$ is similar, with the difference that the randomness generator can be invoked an unbounded number of times. These types induce a partition on random values, in which each set contains at most one element or an unbounded number of elements, respectively. This turns out to be useful, as explained below, to type-check protocols, like Helios, in which the number of times messages of a certain shape are produced is relevant for the security of the protocol.

The type of $k_a$ (resp. $k_b$) says that this key is supposed to encrypt 0 and 1 on the left- and right-hand side of the equivalence relation, further describing the type of the randomness. The type of $k_s$ inherits the two payload types, which are combined in disjunctive form. In fact, public key types implicitly convey an additional payload type, the one characterizing messages encrypted by the attacker: these are of low confidentiality and turn out to be the same on the left- and right-hand side. Key types are crucial to type-check the server code: we verify the signatures produced by $A$ and $B$ and can then use the ciphertext type derived from the type of $k_a$ and $k_b$ to infer after decryption the vote cast by $A$ and $B$, respectively. While processing the other ballots, the server discards the ciphertexts produced with randomness matching the one used by $A$ or $B$: given that these random values are used only once, we know that the remaining ciphertexts must come from the attacker and thus convey the same vote on the left- and on the right-hand side. This suffices to type-check the final output, since the two tallied results on the left- and right-hand side are the same, and thus fulfill trace equivalence.

The type system generates a set of constraints, which, if "consistent", suffice to prove that the protocol is trace equivalent. Intuitively, these constraints characterize the indistinguishability of the messages output by the process. The constraints generated for this simplified version of Helios are reported below:

$$C = \{(\text{sign(aenc(0, (x, r'_a)), pk(k_s)), k_a}) \sim \text{sign(aenc(1, (x, r'_a)), pk(k_s)), k_a)},$$
$$\text{aenc(0, (x, r'_a)), pk(k_s)} \sim \text{aenc(1, (x, r'_a)), pk(k_s))},$$
$$\text{sign(aenc(1, (y, r'_b)), pk(k_s)), k_b}) \sim \text{sign(aenc(0, (y, r'_b)), pk(k_s)), k_b)},$$
$$\text{aenc(1, (y, r'_b)), pk(k_s)} \sim \text{aenc(0, (y, r'_b)), pk(k_s))},$$
\[
[x : \text{LL}, y : \text{LL}]
\}\]

These constraints are consistent if the set of left messages of the constraints is in (static) equivalence with the set of the right messages of the constraints. This is clearly the case here, since encryption hides the content of the plaintext. Just to give an example of non-consistent constraints, consider the following ones:

$$C' = \{h(n_1) \sim h(n_2), \ h(n_1) \sim h(n_2))\}$$

where $n_1$, $n_2$ are two confidential nonces. While the first constraint alone is consistent, since $n_1$ and $n_2$ are of high confidentiality and the attacker cannot thus distinguish between $h(n_1)$ and $h(n_2)$, the two constraints all together are not consistent, since the attacker can clearly notice if the two terms output by the process are the same or not. We developed a dedicated procedure to check the consistency of such constraints.

## 3 Framework

In symbolic models, security protocols are typically modeled as processes of a process algebra, such as the applied pi-calculus [3]. We present here a calculus close to [25] inspired from the calculus underlying the ProVerif tool [20].

### 3.1 Terms

Messages are modeled as terms. We assume an infinite set of names $N$ for nonces, further partitioned into the set $\mathcal{F}N$ of free nonces (created by the attacker) and the set $\mathcal{B}N$ of bound nonces (created by the protocol parties), an infinite set of names $\mathcal{K}$ for keys, ranged over by $k$, and an infinite set of variables $\mathcal{V}$. Cryptographic primitives are modeled through a signature $\mathcal{F}$, that is a set of function symbols, given with their arity (that is, the number of arguments). Here, we will consider the following signature:

$$\mathcal{F}_c = \{\text{pk, vk, enc, aenc, sign, (\cdot, \cdot), h}\}$$

that models respectively public and verification key, symmetric and asymmetric encryption, concatenation and hash. The companion primitives (symmetric and asymmetric decryption, signature check, and projections) are represented by the following signature:

$$\mathcal{F}_d = \{\text{dec, adec, checksign, \pi_1, \pi_2}\}$$

We also consider a set $\mathcal{C}$ of (public) constants (used as agents names for instance). Given a signature $\mathcal{F}$, a set of names $N$ and a set of variables $\mathcal{V}$, the set of terms $\mathcal{T}(\mathcal{F}, \mathcal{V}, N \cup \mathcal{K})$ is the set inductively defined by applying functions to variables in $\mathcal{V}$ and names in $N$. We denote by names$(t)$ (resp. vars$(t)$) the set of names (resp. variables) occurring in $t$. A term is ground if it does not contain variables.

Here, we will consider the set $\mathcal{T}(\mathcal{F}_c \cup \mathcal{F}_d \cup C, \mathcal{V}, N \cup \mathcal{K})$ of cryptographic terms, simply called terms. Messages are terms from $\mathcal{T}(\mathcal{F}_c \cup \mathcal{C}, \mathcal{V}, N \cup \mathcal{K})$ with atomic keys, that is, a term $t \in \mathcal{T}(\mathcal{F}_c \cup \mathcal{C}, \mathcal{V}, N \cup \mathcal{K})$ is a message if any subterm of $t$ of the form $\text{pk}(t')$, $\text{vk}(t')$, $\text{enc}(t_1, t_2)$, $\text{aenc}(t_1, t_2)$, or $\text{sign}(t_1, t_2)$ is such that $t' \in \mathcal{K}$ and $t_2 = \text{pk}(t'_2)$ with $t'_2 \in \mathcal{K}$. We assume the set of variables to be split into two subsets $\mathcal{V} = \mathcal{X} \uplus \mathcal{AX}$ where $\mathcal{X}$ are variables used in processes while $\mathcal{AX}$ are variables used to store messages. An attacker term is a term from $\mathcal{T}(\mathcal{F}_c \cup \mathcal{F}_d \cup C, \mathcal{AX}, \mathcal{F}_N)$.

A substitution $\sigma = \{M_1/x_1, \ldots, M_k/x_k\}$ is a mapping from variables $x_1, \ldots, x_k \in \mathcal{V}$ to messages $M_1, \ldots, M_k$. We let dom$(\sigma) = \{x_1, \ldots, x_k\}$. We say that $\sigma$ is ground if all messages $M_1, \ldots, M_k$ are ground. We let names$(\sigma) = \bigcup_{1 \leq i \leq k} \text{names}(M_i)$. The application of a substitution $\sigma$ to a term $t$ is denoted $t[\sigma]$ and is defined as usual.

The evaluation of a term $t$, denoted $t \Downarrow$, corresponds to the application of the cryptographic primitives. For example, the decryption succeeds only if the right decryption key is used. Formally, $t \Downarrow$
Destructors used in processes:

\[ d ::= \text{dec}(\cdot, k) \mid \text{adec}(\cdot, k) \mid \text{checksign}(\cdot, \text{vk}(k)) \mid \pi_1(\cdot) \mid \pi_2(\cdot) \]

Processes:

\[ P, Q ::= \\
\quad \emptyset \\
\quad \text{new } n.P \\
\quad \text{out}(M).P \\
\quad \text{in}(x).P \\
\quad P \mid Q \\
\quad \text{let } x = d(y) \text{ in } P \text{ else } Q \\
\quad !P \\
\quad \text{where } M, N \text{ are messages.} \]

**Figure 1: Syntax for processes.**

is recursively defined as follows.

\[
\begin{align*}
\text{pk}(t) \downarrow &= \text{pk}(t) \downarrow \\
\text{vk}(t) \downarrow &= \text{vk}(t) \downarrow \\
h(t) \downarrow &= h(t) \downarrow \\
(t_1, t_2) \downarrow &= (t_1 \downarrow, t_2 \downarrow) \\
\text{enc}(t_1, t_2) \downarrow &= \text{enc}(t_1 \downarrow, t_2 \downarrow) \\
\text{sign}(t_1, t_2) \downarrow &= \text{sign}(t_1 \downarrow, t_2 \downarrow) \\
\text{aenc}(t_1, t_2) \downarrow &= \text{aenc}(t_1 \downarrow, t_2 \downarrow)
\end{align*}
\]

for some \( k \in \mathcal{K} \)

\[
\begin{align*}
\pi_1(t) \downarrow &= t_1 & \text{if } t \downarrow &= (t_1, t_2) \\
\pi_2(t) \downarrow &= t_2 & \text{if } t \downarrow &= (t_1, t_2) \\
\text{dec}(t_1, t_2) \downarrow &= t_3 & \text{if } t_1 \downarrow = \text{enc}(t_3, t_4) \text{ and } t_4 = t_2 \downarrow \\
\text{adec}(t_1, t_2) \downarrow &= t_3 & \text{if } t_1 \downarrow = \text{aenc}(t_3, \text{pk}(t_4)) \text{ and } t_4 = t_2 \downarrow \\
\text{checksign}(t_1, t_2) \downarrow &= t_3 & \text{if } t_1 \downarrow = \text{sign}(t_3, t_4) \text{ and } t_2 \downarrow = \text{vk}(t_4) \\
t \downarrow &= \bot & \text{otherwise}
\end{align*}
\]

Note that the evaluation of term \( t \) succeeds only if the underlying keys are atomic and always returns a message or \( \bot \). We write \( t \equiv_t t' \) if \( t \downarrow = t' \downarrow \).

### 3.2 Processes

Security protocols describe how messages should be exchanged between participants. We model them through a process algebra, whose syntax is displayed in Figure 1. We identify processes up to \( \alpha \)-renaming, i.e., capture avoiding substitution of bound names and variables, which are defined as usual. Furthermore, we assume that all bound names and variables in the process are distinct.

A configuration of the system is a quadruple \((E; \mathcal{P}; \phi; \sigma)\) where:

- \( \mathcal{P} \) is a multiset of processes that represents the current active processes;
- \( E \) is a set of names, which represents the private names of the processes;
- \( \phi \) is a substitution with \( \text{dom}(\phi) \subseteq \mathcal{AX} \) and for any \( x \in \text{dom}(\phi), \phi(x) \) (also denoted \( x^\phi \)) is a message that only contains variables in \( \text{dom}(\sigma) \), \( \phi \) represents the terms already output.
- \( \sigma \) is a ground substitution;

The semantics of processes is given through a transition relation \( \rightarrow_\alpha \) on the quadruples provided in Figure 2 (\( \rightarrow \) denotes a silent action). The relation \( \rightarrow_s \) is defined as the reflexive transitive closure of \( \rightarrow_\alpha \), where \( w \) is the concatenation of all actions. We also write equality up to silent actions \( \equiv_s \).

Intuitively, process \( \text{new } n.P \) creates a fresh nonce, stored in \( E \), and behaves like \( P \). Process \( \text{out}(M).P \) emits \( M \) and behaves like \( P \). Process \( \text{in}(x).P \) inputs any term computed by the attacker provided it evaluates as a message and then behaves like \( P \). Process \( P \mid Q \) corresponds to the parallel composition of \( P \) and \( Q \). Process \( \text{let } x = d(y) \text{ in } P \text{ else } Q \) behaves like \( P \) in which \( x \) is replaced by \( d(y) \) if \( d(y) \) can be successfully evaluated and behaves like \( Q \) otherwise. Process \( \text{if } M = N \text{ then } P \text{ else } Q \) behaves like \( P \) if \( M \) and \( N \) correspond to two equal messages and behaves like \( Q \) otherwise. The replicated process \( !P \) behaves as an unbounded number of copies of \( P \).

A trace of a process \( P \) is any possible sequence of transitions in the presence of an attacker that may read, forge, and send messages. Formally, the set of traces \( \text{trace}(P) \) is defined as follows.

\[
\text{trace}(P) = \{(w, \text{new } E, \phi, \sigma)|((\emptyset; \emptyset; \emptyset) \rightarrow_s \mathbf{w}; (E; \mathcal{P}; \phi; \sigma))\}
\]

**Example 3.1.** Consider the Helios protocol presented in Section 2.

For simplicity, we describe here a simplified version with only two (honest) voters \( A \) and \( B \) and a voting server \( S \). This (simplified) protocol can be modeled by the process:

\[
\text{new } r_\text{A}.Voter(k_\text{A}, v_\text{A}, r_\text{A}) \mid \text{new } r_\text{B}.Voter(k_\text{B}, v_\text{B}, r_\text{B}) \mid P_S
\]

where \( \text{Voter}(k, v, r) \) represents voter \( k \) willing to vote for \( v \) using randomness \( r \) while \( P_S \) represents the voting server. \( \text{Voter}(k, v, r) \) simply outputs a signed encrypted vote.

\[
\text{Voter}(k, v, r) = \text{out}((\text{sign}(\text{aenc}(v, r), \text{pk}(k_S)), k))
\]

The voting server receives ballots from \( A \) and \( B \) and then outputs the decrypted ballots, after some mixing.

\[
P_S = \text{in}(x_1).\text{in}(x_2).
\]

\[
\begin{align*}
\text{let } y_1 &= \text{checksign}(x_1, \text{vk}(k_\text{A})) \text{ in } \\
\text{let } y_2 &= \text{checksign}(x_2, \text{vk}(k_\text{B})) \text{ in } \\
\text{let } z_1 &= \text{adec}(y_1, k_\text{A}) \text{ in } \text{let } z'_1 = \pi_1(z_1) \text{ in } \\
\text{let } z_2 &= \text{adec}(y_2, k_\text{B}) \text{ in } \text{let } z'_2 = \pi_2(z_2) \text{ in } \\
&\quad \text{(out}(z'_1) \mid \text{out}(z'_2))
\end{align*}
\]

### 3.3 Equivalence

When processes evolve, sent messages are stored in a substitution \( \phi \) while private names are stored in \( E \). A frame is simply an expression of the form \( \text{new } E, \phi \) where \( \text{dom}(\phi) \subseteq \mathcal{AX} \). We define \( \text{dom}(\text{new } E, \phi) \) as \( \text{dom}(\phi) \). Intuitively, a frame represents the knowledge of an attacker.

Intuitively, two sequences of messages are indistinguishable to an attacker if he cannot perform any test that could distinguish them. This is typically modeled as static equivalence [3]. Here, we consider of variant of [3] where the attacker is also given the ability to observe when the evaluation of a term fails, as defined for example in [25].
We now introduce a type system to statically check trace equivalence from the scenario where the two votes are swapped.

Our typing judgements thus capture properties of pairs of terms or processes, which we will refer to as left and right term or process, respectively.

\[
\begin{align*}
(E; \{P_1 \mid P_2\} \cup \mathcal{P}; \psi; \sigma) & \vdash (E; \{P_1, P_2\} \cup \mathcal{P}; \psi; \sigma) & \text{Par} \\
(E; \emptyset) & \vdash (E; \mathcal{P}; \psi; \sigma) & \text{Zero} \\
(E; \{\text{new } n\} \cup \mathcal{P}; \psi; \sigma) & \vdash (E; \{\text{new } n\}; \mathcal{P} \cup \mathcal{P}; \psi; \sigma) & \text{New} \\
(E; \{\text{out}(t), P\} \cup \mathcal{P}; \psi; \sigma) & \vdash (E; \{\text{out}(t), \mathcal{P}\}; \psi; \phi_{ax_n}) & \text{Out} \\
(E; \{\text{in}(x), P\} \cup \mathcal{P}; \psi; \sigma) & \vdash (E; \{\text{in}(x), \mathcal{P}\}; \psi; \phi_{in(R)}) & \text{In} \\
(E; \{\text{let } x = d(M) \text{ in } P \text{ else } Q\} \cup \mathcal{P}; \psi; \sigma) & \vdash (E; \{\text{let } x = d(M) \text{ in } P \text{ else } Q\} \cup \mathcal{P}; \psi; \sigma \cup \{d(M) \downarrow \}) & \text{Let-Else} \\
(E; \{\text{if } M = N \text{ then } P \text{ else } Q\} \cup \mathcal{P}; \psi; \sigma) & \vdash (E; \{\text{if } M = N \text{ then } P \text{ else } Q\} \cup \mathcal{P}; \psi; \sigma) & \text{If-Else} \\
(E; \{P\} \cup \mathcal{P}; \psi; \sigma) & \vdash (E; \{P\} \cup \mathcal{P}; \psi; \sigma) & \text{Repl}
\end{align*}
\]

Figure 2: Semantics

**Definition 3.3 (Static Equivalence).** Two ground frames \(\text{new } E, \phi\) and \(\text{new } E', \phi'\) are statically equivalent if and only if they have the same domain, and for all attacker terms \(R, S\) with variables in \(\text{dom}(\phi) = \text{dom}(\phi')\), we have

\[
(\text{R}\phi \equiv \bot S\phi) \iff (\text{R}\phi' \equiv \bot S\phi')
\]

Then two processes \(P\) and \(Q\) are in equivalence if no matter how the adversary interacts with \(P\), a similar interaction may happen with \(Q\), with equivalent resulting frames.

**Definition 3.3 (Trace Equivalence).** Let \(P, Q\) be two processes. We write \(P \equiv \bot Q\) if for all \((s, \psi, \sigma) \in \text{trace}(P)\), there exists \((s', \psi', \sigma') \in \text{trace}(Q)\) such that \(s \equiv \bot s'\) and \(\psi\sigma = \psi'\sigma'\) are statically equivalent. We say that \(P\) and \(Q\) are trace equivalent, and we write \(P \approx \bot Q\), if \(P \equiv \bot Q\) and \(Q \equiv \bot P\).

Note that this definition already includes the attacker’s behavior, since processes may input any message forged by the attacker.

**Example 3.4.** As explained in Section 2, ballot privacy is typically modeled as an equivalence property [35] that requires that an attacker cannot distinguish when Alice is voting 0 and Bob is voting 1 from the scenario where the two votes are swapped.

Continuing Example 3.1, ballot privacy of Helios can be expressed as follows:

\[
\begin{align*}
\text{new } ra_v.Voter(k_a, 0, ra) & \mid \text{new } ra_v.Voter(k_b, 1, ra) \mid P_S \\
\approx \uplus \text{new } ra_v.Voter(k_a, 1, ra) & \mid \text{new } ra_v.Voter(k_b, 0, ra) \mid P_S
\end{align*}
\]

**4 Typing**

We now introduce a type system to statically check trace equivalence between processes. Our typing judgements thus capture properties of pairs of terms or processes, which we will refer to as left and right term or process, respectively.

**4.1 Types**

A selection of the types for messages are defined in Figure 3 and explained below. We assume three security labels (namely, HH, HL, LL), ranged over by \(I\), whose first (resp. second) component denotes the confidentiality (resp. integrity) level. Intuitively, messages of high confidentiality cannot be learned by the attacker, while messages of high integrity cannot originate from the attacker. Pair types describe the type of their components, as usual. Type \(\text{key}^l(T)\) describes keys of security level \(l\) used to encrypt (or sign) messages of type \(T\). The type \(\text{key}^l(T)_k\) (resp. \(\text{key}^l(T)_k\)) describes symmetric (resp. asymmetric) encryptions with key \(k\) of a message of type \(T\). The type \(\text{key}^l(T)_k\) describes nonces and constants of security level \(l\): the label \(a\) ranges over \([\infty, 1]\), denoting whether the nonce is bound within a replication or not (constants are always typed with \(a = 1\)). We assume a different identifier \(i\) for each constant and restriction in the process. The type \(\text{key}^l(T)_0\) is populated by a single name, i.e., it describes a constant or a non-replicated nonce and \(\text{key}^l(T)_0\) is a special type, that is instantiated to \(\text{key}^l(T)_0\) in the \(j\)th replication of the process. Type \(\text{key}^l(T)_k\) is a refinement type that restricts the set of values which can be taken by a message to values of \(\text{key}^l(T)_k\) on the left and type \(\text{key}^l(T)_k\) on the right. For a refinement type \(\text{key}^l(T)_k\) with equal types on both sides we simply write \(\text{key}^l(T)_k\). Messages of type \(T \lor T'\) are messages that can have type \(T\) or type \(T'\).

Figure 3: Types for terms (selected)
4.2 Constraints

When typing messages, we generate constraints of the form \((M \sim N)\), meaning that the attacker sees \(M\) and \(N\) in the left and right process, respectively, and these two messages are thus required to be indistinguishable.

4.3 Typing Messages

Typing judgments are parametrized over a typing environment \(\Gamma\), which is a list of mappings from names and variables to types. The typing judgement for messages is of the form the form \(\Gamma \vdash M \sim N : T \rightarrow c\) which reads as follows: under the environment \(\Gamma\), \(M\) and \(N\) are of type \(T\) and either this is a high confidentiality type (i.e., \(M\) and \(N\) are not disclosed to the attacker) or \(M\) and \(N\) are indistinguishable for the attacker assuming the set of constraints \(c\) holds true. We present an excerpt of the typing rules for messages in Figure 4 and comment on them in the following.

Confidential nonces (i.e. nonces with label \(l = HH\) or \(l = HL\)) typed with their label from the typing environment. As the attacker may not observe them, they may be different in the left and the right message and we do not add any constraints (TNONC\(E\)). Public terms are given type \(LL\) if they are the same in the left and the right message (TNONCL, TCS\(T\)F\(N\), TP\(U\)b\(K\)\(E\), TV\(K\)\(E\)). We require keys and variables to be the same in the two processes, deriving their type from the environment (TK\(E\) and TV\(A\)R). The rule for pairs operates recursively component-wise (TP\(A\)R).

For symmetric key encryptions (T\(E\)NC), we have to make sure that the payload type matches the key type (which is achieved by rule T\(E\)NC\(H\)). We add the generated ciphertext to the set of constraints, because even though the attacker cannot read the plaintext, he can perform an equality check on the ciphertext that he observed. If we type an encryption with a key that is of low confidentiality (i.e., the attacker has access to it), then we need to make sure the payload is of type \(LL\), because the attacker can simply decrypt the message and recover the plaintext (T\(E\)NC\(L\)). The rules for asymmetric encryption are the same, with the only difference that we can always choose to ignore the key type and use type \(LL\) to check the payload. This allows us to type messages produced by the attacker, which has access to the public key but does not need to respect its type. Signatures are also handled similarly, the difference here is that we need to type the payload with \(LL\) even if an honest key is used, as the signature does not hide the content. The first typing rule for hashes (T\(H\)ash\(H\)) gives them type \(LL\) and adds the term to the constraints, without looking at the arguments of the hash function: intuitively this is justified, because the hash function makes it impossible to recover the argument. The second rule (T\(H\)ash\(L\)) gives type \(LL\) only if we can also give type \(LL\) to the argument of the hash function, but does not add any constraints on its own, it is just passing on the constraints created for the arguments. This means we are typing the message as if the hash function would not have been applied and use the message without the hash, which is a strictly stronger result. Both rules have their applications: while the former has to be used whenever we hash a secret, the latter may be useful to avoid the creation of unnecessary constraints when hashing terms like constants or public nonces. Rule T\(H\)igh\(H\) states that we can give type \(HL\) to every message, which intuitively means that we can treat every message as if it were confidential. Rule T\(S\)ub allows us to type messages according to the subtyping relation, which is standard and defined in Figure 5. Rule T\(O\) allows us to give a union type to messages, if they are typable with at least one of the two types. T\(L\)R and T\(L\)R\(^{\sim}\) are the introduction rules for refinement types, while T\(L\)R' and T\(L\)R\(^{-}\) are the corresponding elimination rules. Finally, T\(L\)-\(V\)ar allows to derive a new refinement type for two variables for which we have singleton refinement types, by taking the left refinement of the left variable and the right refinement of the right variable. We will see application of this rule in the e-voting protocol, where we use it to combine A’s vote (0 on the left, 1 on the right) and B’s vote (1 on the left, 0 on the right), into a message that is the same on both sides.

4.4 Typing Processes

The typing judgement for processes is of the form \(\Gamma \vdash P \sim Q : C\) and can be interpreted as follows: If two processes \(P\) and \(Q\) can be typed in \(\Gamma\) and if the generated constraint set \(C\) is consistent, then \(P\) and \(Q\) are trace equivalent. We assume in this section that \(P\) and \(Q\) do not contain replication and that variables and nonce names are renamed to avoid any capture. We also assume processes to be given with type annotations for nonces.

When typing processes, the typing environment \(\Gamma\) is passed down and extended from the root towards the leaves of the syntax tree of the process, i.e., following the execution semantics. The generated constraints \(C\) however, are passed up from the leaves towards the root, so that at the root we get all generated constraints, modeling the attacker’s global view on the process execution.

More precisely, each possible execution path of the process - there may be multiple paths because of conditionals - creates its own set of constraints \(c\) together with the typing environment \(\Gamma\) that contains types for all names and variables appearing in \(c\). Hence a constraint set \(C\) is a set elements of the form \((c, \Gamma)\) for a set of constraints \(c\). The typing environments are required in the constraint checking procedure, as they helps us to be more precise when checking the consistency of constraints.

An excerpt of our typing rules for processes is presented in Figure 6 and explained in the following. Rule P\(Z\)ero copies the current typing environment in the constraints and checks the well-formedness of the environment (\(\Gamma \vdash \_\)), which is defined as expected. Messages output on the network are possibly learned by the attacker, so they have to be of type \(LL\) (P\(O\)ut). The generated constraints are added to each element of the constraint set for the continuation process, using the operator \(\cup\) defined as

\[ C \cup C' := \{(c \cup c', \Gamma) | (c, \Gamma) \in C\} \]

Conversely, messages input from the network are given type \(LL\) (PIn\(n\)). Rule P\(N\)ew introduces a new nonce, which may be used in the continuation processes. While typing parallel composition (PP\(A\)R), we type the individual subprocesses and take the product union of the generated constraint sets as the new constraint set. The product union of constraint sets is defined as

\[ C \cup \chi C' := \{c \cup c', \Gamma \cup \Gamma'\} | (c, \Gamma) \in C \land (c', \Gamma') \in C' \land \Gamma, \Gamma' \text{ are compatible}\]

where compatible environments are those that agree on the type of all arguments of the shared domain. This operation models the fact
that a process \( P \mid P' \) can have every trace that is a combination of any trace of \( P \) with any trace of \( P' \). The branches that are discarded due to incompatible environments correspond to impossible executions (e.g., taking the left branch in \( P \) and the right branch in \( P' \) in two conditionals with the same guard). \( \text{OR} \) is the elimination rule for union types, which requires the continuation process to be well-typed with both types.

To ensure that the destructor application fails or succeeds equally in the two processes, we allow only the same destructor to be applied to the same variable in both processes (\( \text{PLET} \)). As usual, we then type-check the then as well as the else branch and then take the union of the corresponding constraints. The typing rules for destructors are presented in Figure 7. These are mostly standard: for instance, after decryption, the type of the payload is determined by the one of the decryption key, as long as this is of high integrity (\( \text{DDecH} \)). We can as well exploit strong types for ciphertexts, typically introduced by verifying a surrounding signature (see, e.g., the types for Helios) to derive the type of the payload (\( \text{DDecT} \)). In the case of public key encryption, we have to be careful, since the public encryption key is accessible to the attacker: we thus give the payload type \( T \lor LL \) (rule \( \text{DAdecH} \)). For operations involving corrupted keys (label \( LL \)) we know that the payload is public and hence give the derived message type \( LL \).

In the special case in which we know that the concrete value of the argument of the destructor application is a nonce or constant due to a refinement type, and we know statically that any destructor application will fail, we only need to type-check the else branch.
All these special cases highlight how a careful treatment of names cannot be assumed that they are equal, as the sets are infinite, unbounded, and we check for a variant of diff-equivalence. This ensures that the value of the terms in the conditional (because the corresponding constraint set is consistent) and if a reduction step with action \( \alpha \) is applied to reduce \( P \) into \( Q \), the processes in \( P \) and \( Q \) are pairwise typably equivalent (with consistent constraints), and if a reduction step with action \( \alpha \) can be performed to reduce \( P \) into \( Q \), then \( Q \) can be reduced in one or several steps, with the same action \( \alpha \), to some multiset \( Q' \) such that the processes in \( P' \) and \( Q' \) are still typably equivalent (with consistent constraints). This is done by carefully examining all the possible typing rules used to type the processes in \( P \) and \( Q \). In addition, we show that the frames of messages output when reducing and explain how it captures the attacker’s capability to distinguish processes based on their outputs.

To define consistency, we need the following ingredients:

- \( \phi_L(c) \) and \( \phi_R(c) \) denote the frames that are composed of the left and the right terms of the constraints respectively (in the same order).
- \( \phi_L(c') \) denotes the frame that is composed of all low confidentiality nonces and keys in \( \Gamma \), as well as all public encryption keys and verification keys in \( \Gamma \). This intuitively corresponds to the initial knowledge of the attacker.
- Let \( E_L \) be the set of all nonces occurring in \( \Gamma \).
- Two ground substitutions \( \sigma, \sigma' \) are well-formed in \( \Gamma \) if they preserve the types for variables in \( \Gamma \) (i.e., \( \Gamma \vdash \sigma(x) \sim \sigma'(x) : \Gamma(x) \to v(x) \)).

**Definition 5.1 (Consistency).** A set of constraints \( c \) is consistent in an environment \( \Gamma \) if for all substitutions \( \sigma, \sigma' \) well-typed in \( \Gamma \) the frames \( \text{new } E_L (\phi_L(c') \cup \phi_R(c)\sigma) \) and \( \text{new } E_L (\phi_L(c') \cup \phi_R(c)\sigma') \) are statically equivalent. We say that \( (c, \Gamma) \) is consistent if \( c \) is consistent in \( \Gamma \) and that a constraint set \( C \) is consistent in \( \Gamma \) if each element \( (c, \Gamma) \in C \) is consistent.

We define consistency of constraints in terms of static equivalence, as this notion exactly captures all capabilities of our attacker: to distinguish two processes, he can arbitrarily apply constructors and destructors on observed messages to create new terms, on which he can then perform equality tests or check the applicability of destructors. We require that this property holds for any well-typed substitutions, to soundly cover that fact that we do not know the content of variables statically, except for the information we get by typing. In Section 6.3 we introduce an algorithm to check consistency of constraints.

## 6 MAIN RESULTS

In this section, we state our two main soundness theorems, entailing trace equivalence by typing for the bounded and unbounded case, and we explain how to automatically check consistency.

### 6.1 Soundness of the type system

Our type system soundly enforces trace equivalence: if we can typecheck \( P \) and \( Q \) then \( P \) and \( Q \) are equivalent, provided that the corresponding constraint set is consistent.

**Theorem 6.1 (Typing implies trace equivalence).** For all \( P, Q, \) and \( C \), for all \( \Gamma \) containing only keys, if \( \Gamma \vdash P \sim Q \) and \( C \) is consistent, then \( P \sim_{\Gamma} Q \).

To prove this theorem, we first show that typing is preserved by reduction, and guarantees that the same actions can be observed on both sides. More precisely, we show that if \( P \) and \( Q \) are multisets of processes which are pairwise typably equivalent (with consistent constraints), and if a reduction step with action \( \alpha \) can be performed to reduce \( P \) into \( Q' \), then \( Q \) can be reduced in one or several steps, with the same action \( \alpha \), to some multiset \( Q'' \) such that the processes in \( P' \) and \( Q' \) are still typably equivalent (with consistent constraints). This is done by carefully examining all the possible typing rules used to type the processes in \( P \) and \( Q \). In addition we show that the frames of messages output when reducing...
\[
\begin{align*}
\Gamma \vdash \varnothing & \quad \text{(PZERO)} \\
\Gamma \vdash i \sim i & \quad \text{(POUT)} \\
\Gamma \vdash \text{out}(M).P \sim \text{out}(N).Q & \quad \text{(PPar)} \\
\Gamma \vdash \text{in}(x).P \sim \text{in}(x).Q & \quad \text{(POR)} \\
\Gamma \vdash P \sim Q & \quad \text{(PLet)} \\
\Gamma \vdash \text{let } x = d(y) \text{ in } P \text{ else } P' \quad \text{(PLetLR)} \\
\Gamma \vdash \text{if } M = M' \text{ then } P \text{ else } P' & \quad \text{(PIrL)} \\
\Gamma \vdash M_1 \sim N_1 & \quad \text{(PIrLR)} \\
\Gamma \vdash M_2 \sim N_2 & \quad \text{(PIrS)} \\
\Gamma \vdash M \sim N & \quad \text{(PIrLR')} \\
\end{align*}
\]

\(P\) and \(Q\) are typably equivalent with consistent constraints; and that this entails their static equivalence.

This implies that if \(P\) and \(Q\) are typable with a consistent constraint, then for each trace of \(P\), by induction on the length of the trace, there exists a trace of \(Q\) with the same sequence of actions, and with a statically equivalent frame. That is to say \(P \equiv_t Q\). Similarly we show \(Q \equiv_t P\), and we thus have \(P \equiv_t Q\).

Since we do not have typing rules for replication, Theorem 6.1 only allows us to prove equivalence of protocols for a \textit{finite} number of sessions. An arguably surprising result, however, is that, thanks to our infinite nonce types, we can prove equivalence for an \textit{unbounded} number of sessions, as detailed in the next section.

6.2 Typing replicated processes

For more clarity, in this section, without loss of generality we consider that for each infinite nonce type \(r_{m,n}^{l} = \infty\) appearing in the processes, the set of names \(\mathcal{B}N\) contains an infinite number of fresh names \(\{m_i \mid i \in \mathbb{N}\}\) which do not appear in the processes or environments. We similarly assume that for all the variables \(x\) appearing in the processes, the set \(\mathcal{X}\) of all variables also contains fresh variables \(\{x_i \mid i \in \mathbb{N}\}\) which do not appear in the processes or environments.

Intuitively, whenever we can typecheck a process of the form \(\text{new } n : r_{n}^{l,i} \cdot \text{new } m : r_{m,n}^{l} \cdot P\), we can similarly typecheck

\[
\text{new } n : r_{n}^{l,1} \cdot (\text{new } m_1 : r_{m,n}^{l,1} \cdot P_1) \ldots (\text{new } m_k : r_{m,n}^{l,1} \cdot P_k)
\]

where in \(P_i\), the nonce \(m\) has been replaced by \(m_i\); and variables \(x\) have been renamed to \(x_i\).

Formally, we denote by \([t]^P\), the term \(t\) in which names \(n\) such that \(\Gamma(n) = r_{n}^{l,0}\) for some \(l\) are replaced by \(n_i\), and variables \(x\) are replaced by \(x_i\).

Similarly, when a term is of type \([r_{m,n}^{l,0} \succ r_{p,i}^{l,0}]\), it can be of type \([r_{m,i}^{l,0} \succ r_{p,i}^{l,0}]\) for any \(i\). The nonce type \(r_{m,n}^{l,0}\) represents infinitely many nonces (one for each session). That is, for \(n\) sessions, the type \([r_{m,n}^{l,0} \succ r_{p,n}^{l,0}]\) represents all \([r_{m,i}^{l,0} \succ r_{p,i}^{l,0}]\). Formally, given a type \(T\),
we define its expansion to \( n \) sessions, denoted \([ T ]^n\), as follows.

\[
\begin{align*}
\Gamma(k) = \text{key}^{\text{HHH}}(T) & \quad \Gamma(x) = \text{LL} & \quad (\text{DDecH}) \\
\Gamma(k) = \text{key}^{\text{LL}}(T) & \quad \Gamma(x) = \text{LL} & \quad (\text{DDecL}) \\
\Gamma(x) = (T)_k & \quad (\text{DDecT}) \\
\Gamma(k) = \text{ade}(x, k) : T & \quad \Gamma(x) = \text{LL} & \quad (\text{DAde}) \\
\Gamma(k) = \text{ade}^L(x, k) & \quad \Gamma(x) = \text{LL} & \quad (\text{DAdeL}) \\
\Gamma(x) = (T)_k & \quad (\text{DAdeT}) \\
\Gamma(x) = \text{check}(x, \text{v}(k)) : T & \quad (\text{DCheckH}) \\
\Gamma(x) = \text{check}^L(x, \text{v}(k)) : \text{LL} & \quad (\text{DCheckL}) \\
\Gamma(x) = T \times T' & \quad (\text{DSN}) \\
\Gamma(x) = T \times T' & \quad (\text{DSn}) \\
\Gamma(x) = \text{LL} & \quad (\text{DSnL}) \\
\Gamma(x) = \text{pi}(x) : T & \quad (\text{DSn}) \\
\end{align*}
\]

where \( l', l'' \in \{\text{LL}, \text{HHH}, \text{HL}\}, k \in \mathcal{K}\). Note that the size of the expanded type \([ T ]^n\) depends on \( n \).

We need to adapt typing environments accordingly. For any typing environment \( \Gamma \), we define its renaming for session \( i \) as:

\[
\begin{align*}
\Gamma_i &= \{x_i : [ T ] \mid [ \Gamma(x) = T \cup \{ k : [ T ] \mid \Gamma(k) = T \} \\
&\cup \{ m : \tau_{m}^{l, i} \mid [ \Gamma(m) = \tau_{m}^{l, i} \} \cup \{ m_i : \tau_{m}^{l, i} \mid [ \Gamma(m) = \tau_{m}^{l, i} \}
\end{align*}
\]

and then its expansion to \( n \) sessions as

\[
\begin{align*}
\Gamma_i^n &= \{x_i : [ T ] \mid [ \Gamma(x) = T \cup \{ k : [ T ] \mid [ \Gamma(k) = T \} \\
&\cup \{ m : \tau_{m}^{l, i} \mid [ \Gamma(m) = \tau_{m}^{l, i} \} \cup \{ m_i : \tau_{m}^{l, i} \mid [ \Gamma(m) = \tau_{m}^{l, i} \}
\end{align*}
\]

Note that in \([ \Gamma_i^n] \), due to the expansion, the size of the types depends on \( n \).

By construction, the environments contained in the constraints generated by typing do not contain union types. However, refinement with infinite nonce types introduce union types when expanded. In order to recover environments without union types after expanding, which, as we will explain in the next subsection, is needed for our consistency checking procedure, we define branches \([ \Gamma_i^n] \) as the set of all \( \Gamma' \), with the same domain as \([ \Gamma_i^n] \), such that for all \( x, \Gamma'(x) \) is not a union type, and either

- \([ \Gamma_i^n] = \Gamma'(x)\); or
- there exist types \( T_1, \ldots, T_k, T'_1, \ldots, T'_k \) such that

\[
\Gamma_i^n = T_1 \ldots T_k \land T'_1 \ldots T'_k
\]

Finally, when typechecking two processes containing nonces with infinite nonce types, we collect constraints that represent families of constraints.

Given a set of constraints \( C \) and an environment \( \Gamma \), we define the renaming of \( C \) for session \( i \) in \( \Gamma \) as \([ C ]_{\Gamma} = \{ \{ x_0 \mid [ \Gamma(x) = T \mid \exists i. x_i \sim x_0 \} \mid x_0 \in C \} \). This is propagated to constraint sets as follows: the renaming of \( C \) for session \( i \) is \([ C ]_{\Gamma} = \{ \{ x_0 \mid [ \Gamma(x) = T \mid \exists i. x_i \sim x_0 \} \mid x_0 \in C \} \) and its expansion to \( n \) sessions is \([ C ]_{\Gamma}^n = \{ \{ x_0 \mid [ \Gamma(x) = T \mid \exists i. x_i \sim x_0 \} \mid x_0 \in C \} \).

Our type system is sound for replicated processes provided that the collected constraint sets are consistent, when instantiated with all possible instantiations of the nonces and keys.

**Theorem 6.2.** Consider \( P, Q, P', Q', C, C' \), such that \( P, Q \) and \( P', Q' \) do not share any variable. Consider \( \Gamma \), containing only keys and nonces with types of the form \( \tau_{m}^{l, i} \).

Assume that \( P \) and \( Q \) only bind nonces with infinite nonce types, i.e., using new \( m : \tau_{m}^{l, i} \) for some label \( l \); while \( P' \) and \( Q' \) only bind nonces with finite types, i.e., using new \( m : \tau_{m}^{l, i} \).

Let us abbreviate by new \( \Pi \) the sequence of declarations of each nonce \( m \) in dom(\( \Gamma \)). If

- \( \Gamma \vdash P \sim Q \rightarrow C \),
- \( \Gamma \vdash P' \sim Q' \rightarrow C' \),
- \( C' \cup \{ x_{k} : [ \Pi] \mid k \leq n \} \) is consistent for all \( n \),

then new \( \Pi \), \( (\Pi) \mid P' \sim Q' \).
6.3 Procedure for consistency

Checking consistency of a set of constraints amounts to checking static equivalence of the corresponding frames. Our procedure follows the spirit of [5] for checking computational indistinguishability: we first open encryption, signatures and pairs as much as possible. Note that the type of a key indicates whether it is public or secret. The two resulting frames should have the same shape. Then, for unopened components, we simply need to check that they satisfy the same equalities.

From now on, we only consider constraint sets that can actually be generated when typing processes, as these are the only ones for which we need to check consistency.

Formally, the procedure check_const is described in Figure 8. It consists of four steps. First, we replace variables with refinements of finite nonce types by their left and right values. In particular a variable with a union type is not associated with a single value and thus cannot be replaced. This is why the branching operation needs to be performed when expanding environments containing refinements with types of the form \( T_n \). Second, we recursively open the constraints as much as possible. Third, we check that the resulting constraints have the same shape. Finally, as soon as two constraints \( T = T’ \) and \( N = N’ \) are such that \( N, N’ \) are unifiable, we must have \( M’ = M’’ \), and conversely. The condition is slightly more involved, especially when the constraints contain variables of refined types with infinite nonce types.

Example 6.3. Continuing Example 3.1, when typechecked with appropriate key types, the simplified model of Helios yields constraint sets containing notably the following two constraints:

\[
\{ \text{aenc}(0, r_a), \text{pk}(k_s) \} \sim \text{aenc}(1, r_a), \text{pk}(k_s), \\
\text{aenc}(1, r_b), \text{pk}(k_s) \sim \text{aenc}(0, r_b), \text{pk}(k_s) \}
\]

For simplicity, consider the set \( c \) containing only these two constraints, together with a typing environment \( \Gamma \) where \( r_a \) and \( r_b \) are respectively given types \( T_n \) and \( T_m \), and \( k_s \) is given type \( \text{key}^H(T) \) for some \( T \).

The procedure check_const(\( \{c, \Gamma\} \)) can detect that the constraint \( c \) is consistent and returns \( \text{true} \). Indeed, as \( c \) does not contain variables, \( \text{step}1(c) \) simply returns \( \text{true} \). Since \( c \) only contains messages encrypted with secret keys, \( \text{step}2(c) \) also leaves \( c \) unmodified. \( \text{step}3(c) \) then returns \( \text{true} \), since the messages appearing in \( c \) are messages asymmetrically encrypted with secret keys, which contain a secret nonce \( r_a \) and \( r_b \) directly under pairs. Finally \( \text{step}4(c) \) trivially returns \( \text{true} \), as the messages \( \text{aenc}(0, r_a), \text{pk}(k_s) \) and \( \text{aenc}(1, r_b), \text{pk}(k_s) \) cannot be unified, as well as the messages \( \text{aenc}(1, r_a), \text{pk}(k_s) \) and \( \text{aenc}(0, r_b), \text{pk}(k_s) \).

Consider now the following set \( c’ \), where encryption has not been randomized:

\[
c’ = \{ \text{aenc}(0, \text{pk}(k_s)) \sim \text{aenc}(1, \text{pk}(k_s)), \\
\text{aenc}(1, \text{pk}(k_s)) \sim \text{aenc}(0, \text{pk}(k_s)) \}
\]

The procedure check_const(\( \{c’, \Gamma\} \)) returns \( \text{false} \). Indeed, contrary to the case of \( c \), \( \text{step}2(c’ \}) \) fails, as the encrypted message do not contain a secret nonce. Actually, the corresponding frames are indeed not statically equivalent since the adversary can reconstruct the encryption of 0 and 1 with the key \( \text{pk}(k_s) \) (in his initial knowledge), and check for equality.

\[
\text{step}1(c) := [c][\text{a} \Gamma, \text{a}’ \Gamma] \text{ with }
F := \{ x \in \text{dom}(\Gamma) \mid \\
\exists m, n, l, l’, \Gamma(x) = \{ r_{m1}^1 : r_{n1}^1 \}
\}
\]

and \( \text{a} \Gamma, \text{a}’ \Gamma \) defined by
\[
\begin{align*}
\text{dom}(\text{a} \Gamma) &= \text{dom}(\text{a}’ \Gamma) = F \\
\forall x \in F, \forall m, n, l, l’, \Gamma(x) &= \{ r_{m1}^1 : r_{n1}^1 \} \Rightarrow \text{a} \Gamma(x) = [m \land \text{a}’ \Gamma(x)] = n
\end{align*}
\]

\[
\text{step}2(c) \text{ is recursively defined by, for all } M, N, M’, N’:
\begin{align*}
\text{step}21 &:= (\{ M, N \} \sim \{ M’, N’ \}) \Rightarrow \text{step}21 := (\{ M \sim M’ \} \Rightarrow \text{step}21) \\
\text{step}22 &:= (\{ \text{enc}(M, k) \sim \text{enc}(M’, k) \} \cup c, c’) := \text{step}21 ((\{ M \sim M’ \} \Rightarrow \text{step}22)) \\
\text{step}23 &:= \text{step}22 ((\{ \text{enc}(M, k) \sim \text{enc}(M’, k) \} \cup c, c’)) := \text{step}21 ((\{ M \sim M’ \} \Rightarrow \text{step}23)) \\
\text{step}24 &:= (\{ \text{sign}(M, k) \sim \text{sign}(M’, k) \} \cup c, c’) := \text{step}21 ((\{ M \sim M’ \} \Rightarrow \text{step}24)) \\
\text{step}25 &:= (\{ M \sim M’ \} \Rightarrow \text{step}25)
\end{align*}
\]

\[
\text{step}2(c) := \text{check} \text{that for all } M \sim M’ \text{ and } N \sim N’ \text{ in } c, M \text{ and } N \text{ are both }
\begin{align*}
\text{a key } k \in \mathcal{K} \text{ such that } \mathcal{E}, \Gamma(k) = \text{key}^H(T); \\
\text{nones } m, n \in N \text{ such that } \\
\exists x \in \{ 1, \infty \}, \{ n \} = \frac{1}{n \times a}, \Gamma(n) = \frac{1}{n \times a}; \\
\text{or public keys, verification keys, or constants;}
\text{or } \text{enc}(M’, k), \text{enc}(N’, k) \text{ such that } \mathcal{E}, \Gamma(k) = \text{key}^H(T); \\
\text{or either } \text{h}(M’) \text{, } \text{h}(N’) \text{ or } \text{enc}(M’, k), \text{enc}(N’, k), \text{pk}(k)), \text{ where } \mathcal{E}, \Gamma(k) = \text{key}^H(T); \text{ such that } M’ \text{ and } N’ \text{ contain directly under pairs some } n \in \mathcal{N} \text{ such that } \mathcal{E}, \Gamma(k) = \text{key}^H(T); \\
\text{or a sign}(M’, k), \text{sign}(N’, k) \text{ such that } \mathcal{E}, \Gamma(k) = \text{key}^H(T).
\end{align*}
\]

\[
\text{step}4(c) := \text{If for all } M \sim M’ \text{ and } N \sim N’ \text{ in } c \text{ such that } M, N \text{ are unifiable with a most general unifier } \mu, \text{ and such that }
\forall x \in \text{dom}(\mu), \exists l, l’, m, p, \Gamma(x) = \{ r_{m1}^1 : r_{p1}^1 \} \Rightarrow (x \mu \in \mathcal{X} \lor \exists l, x \mu = m_1)
\]

we have
\[
M \mu \theta = N’ \mu \theta
\]

where
\[
\forall x \in \text{dom}(\mu), \forall l, l’, m, p, i, \Gamma(x) = \{ r_{m1}^1 : r_{p1}^1 \} \Rightarrow \mu(x) = m_i \Rightarrow (x \mu = m_i)
\]
and \( \theta \) is the restriction of \( \mu \) to \( \{ x \in \text{dom}(\mu) \mid \Gamma(x) = \{ l, \mu(x) \in N’ \} \} \), and if the symmetric condition for the case where \( M’ \) and \( N’ \) are unifiable holds as well, then return true.

\[
\text{check}_const(C) := \text{for all } c, \Gamma \in C, \text{let } c_1 := \text{step}21(\text{step}1(c)) \text{ and check that } \text{step}31(c_1) = \text{true} \text{ and } \text{step}41(c_1) = \text{true}.
\]
For constraint sets without infinite nonce types, check_const entails consistency.

**Theorem 6.4.** Let $C$ be a set of constraints such that
\[ \forall (c, \Gamma) \in C. \forall l, l', m, p. \Gamma(x) \neq [l^{\infty}, \tau_{m}]. \]
If check_const$(C)$ = true, then $C$ is consistent.

We prove this theorem by showing that, for each of the first two steps of the procedure, if step2$_{l}$($C)$ is consistent in $\Gamma$, then $C$ is consistent in $\Gamma$. It then suffices to check the consistency of the constraint step2$_{l}$(step1$_{l}$($C)$) in $\Gamma$. Provided that step2$_{l}$ holds, we show that this constraint is saturated in the sense that any message obtained by the attacker by decomposing terms in the constraint already occurs in the constraint; and the constraint only contains messages which cannot be reconstructed by the attacker from the rest of the constraint. Using this property, we finally prove that the simple unification tests performed in step4 are sufficient to ensure static equivalence of each side of the constraint for any well-typed instantiation of the variables.

As a direct consequence of Theorems 6.1 and 6.4, we now have a procedure to prove trace equivalence of processes without replication.

For proving trace equivalence of processes with replication, we need to check consistency of an infinite family of constraint sets, as prescribed by Theorem 6.2. As mentioned earlier, not only the number of constraints is unbounded, but the size of the type of some (replicated) variables is also unbounded (i.e. of the form $\exists_{m} l^{\infty}$).

We use here two ingredients: we first show that it is sufficient to apply our procedure to two constraints only. Second, we show that our procedure applied to variables with replicated types, i.e. with some environment: $\exists_{m} l^{\infty}$, implies consistency of the corresponding constraints with types of unbounded size.

### 6.4 Two constraints suffice

Consistency of a constraint set $C$ does not guarantee consistency of $C_{l_{1} : \ldots : \ld_{n}} = C_{l_{1} : \ld_{n}}$. For example, consider
\[ C = \{(h(m) \sim h(p)), \{m : \tau_{m}, p : \tau_{p}^{H(1)} \}) \]
which can be obtained when typing
\[ \text{new } m : \tau_{m}^{H(1)}, \text{new } p : \tau_{p}^{H(1)}, \text{out(h(m))} \sim \text{out(h(p))}. \]
$C$ is consistent: since $m, p$ are secret, the attacker cannot distinguish between their hashes. However $C_{l_{1} : \ld_{n}} \subseteq C_{l_{1} : \ld_{n}}$ contains (together with some environment):
\[ \{h(m_{1}) \sim h(p), h(m_{2}) \sim h(p), \ldots, h(m_{n}) \sim h(p)\} \]
which is not, since the attacker can notice that the value on the right is always the same, while the value on the left is not.

Note however that the inconsistency of $C_{l_{1} : \ld_{n}}$ would have been discovered when checking the consistency of two copies of the constraint set only. Indeed, $C_{l_{1} : \ld_{n}} \subseteq C_{l_{1} : \ld_{n}}$ contains (together with some environment):
\[ \{h(m_{1}) \sim h(p); h(m_{2}) \sim h(p)\} \]
which is already inconsistent, for the same reason.

Actually, checking consistency (with our procedure) of two constraints $C_{l_{1} : \ld_{n}}$ and $C_{l_{1} : \ld_{n}}$ entails consistency of $C_{l_{1} : \ld_{n}}$. Note that this does not mean that consistency of $C_{l_{1} : \ld_{n}}$ and $C_{l_{1} : \ld_{n}}$ implies consistency of $C_{l_{1} : \ld_{n}}$. Instead, our procedure ensures a stronger property, for which two constraints suffice.

**Theorem 6.5.** Let $C$ and $C'$ be two constraint sets, which do not contain any common variables. For all $n \in \mathbb{N}$,
\[ \text{check}_\text{const}(C_{l_{1} : \ld_{n}} \cup C_{l_{1} : \ld_{n}}) = \text{true} \Rightarrow \text{check}_\text{const}(C_{l_{1} : \ld_{n}} \cup C_{l_{1} : \ld_{n}}) = \text{true}. \]

It is rather easy to show that if
\[ \text{check}_\text{const}(C_{l_{1} : \ld_{n}} \cup C_{l_{1} : \ld_{n}}) = \text{true}, \]
then the first three steps of the procedure check_const can be successfully applied to each element of $(C_{l_{1} : \ld_{n}} \cup C_{l_{1} : \ld_{n}})_{l_{1} : \ld_{n}}$. However the case of the fourth step is more intricate. When applying the procedure check_const to an element of the constraint set $(C_{l_{1} : \ld_{n}} \cup C_{l_{1} : \ld_{n}})_{l_{1} : \ld_{n}}$, if step4 fails, then the constraint contains an inconsistency, i.e. elements $M \sim M'$ and $N \sim N'$ for which the unification condition from step4 does not hold. Intuitively, the property holds by contraposition thanks to the fact that a similar inconsistency, up to reindexing the nonces and variables, can then already be found when considering only the first two constraint sets, i.e. in $C_{l_{1} : \ld_{n}} \cup C_{l_{1} : \ld_{n}}$. The actual proof requires a careful examination of the structure of the constraint set $(C_{l_{1} : \ld_{n}} \cup C_{l_{1} : \ld_{n}})_{l_{1} : \ld_{n}}$ to establish this reindexing.

### 6.5 Reducing the size of types

The procedure check_const applied to replicated types implies consistency of corresponding constraints with unbounded types.

**Theorem 6.6.** Let $C$ be a constraint set. Then for all $i$,
\[ \text{check}_\text{const}(C_{l_{1} : \ld_{n}}) = \text{true} \Rightarrow \forall n \geq 1. \text{check}_\text{const}(C_{l_{1} : \ld_{n}}) = \text{true}. \]

Again here, it is rather easy to show that if check_const$(C_{l_{1}})$ = true then the first three steps of the procedure check_const can successfully be applied to each element of $\cup C_{l_{1} : \ld_{n}}$. The case of step4 is more involved. The property holds thanks to the condition on the most general unifier expressed in step4. Intuitively, this condition is written in such a way that if, when applying step4 to an element of $C_{l_{1} : \ld_{n}}$, two messages can be unified, then the corresponding messages (with replicated types) in $\cup C_{l_{1} : \ld_{n}}$ can be unified with a most general unifier $\mu$ satisfying the condition. The proof uses this idea to show that if step4 succeeds on all elements of $\cup C_{l_{1} : \ld_{n}}$, then it also succeeds on the elements of $\cup C_{l_{1} : \ld_{n}}$.

### 6.6 Checking the consistency of the infinite constraint

Theorems 6.2, 6.5, and 6.6 provide a sound procedure for checking trace equivalence of processes with and without replication.

**Theorem 6.7.** Let $C$, and $C'$ be two constraint sets without any common variable.
\[ \text{check}_\text{const}(C_{1} \cup C_{2} \cup C'_{1}) = \text{true} \Rightarrow \forall n. (C'_{n} \cup C_{1} \cup C_{2})_{l_{1} : \ld_{n}} \text{ is consistent}. \]

All detailed proofs are available online [1].
7. Experimental Results

We have implemented a prototype type-checker TypeEq and applied it on various examples briefly described below.

**Symmetric key protocols.** For the sake of comparison, we consider 5 symmetric key protocols taken from the benchmark of [28], and described in [26]: Denning-Sacco, Wide Mouth Frog, Needham-Schroeder, Yahalom-Lowe, and Otway-Rees. All these protocols aim at exchanging a key $k$. We prove strong secrecy of the key, as defined in [2], i.e., $P(k_1) \approx P(k_2)$ where $k_1$ and $k_2$ are public names. Intuitively, an attacker should not be able to tell which key is used even if he knows the two possible values in advance. For some of the protocols, we truncated the last step, when it consists in using the exchanged key for encryption, since our framework currently covers only encryption with long-term (fixed) keys.

**Asymmetric key protocols.** In addition to the symmetric key protocols, we consider the well-known Needham-Schroeder-Lowe (NSL) protocol [37] and we again prove strong secrecy of the nonce sent by the receiver (Bob).

**Helios.** We model the Helios protocol for two honest voters and infinitely many dishonest ones, as informally described in Section 2. The corresponding process includes a non trivial else branch, used to express the weeding phase [32], when dishonest ballots equal to some honest one are discarded. As emphasised in Section 2, Helios is secure only if honest voters vote at most once. Therefore the protocol includes non replicated processes (for voters) as well as a replicated process (to handle dishonest ballots).

All our experiments have been run on a single Intel Xeon E5-2687Wv3 3.10GHz core, with 378GB of RAM (shared with the 19 other cores). All corresponding files can be found online at [1].

| Protocols (# sessions) | Akiss | APTE | APTE-POR | Spec | Sat-Eq | TypeEq |
|------------------------|-------|------|----------|------|--------|--------|
| Denning - Sacco        | 6     | 0.00s| 0.32s | 0.00s| 0.09s | 0.002s|
| 7                      | 3.9s  | 1.6s | 191m    | 0.3s | 0.8s   | 0.044s|
| 9                      | 29s   | 3.6s | 52m     | 1.8s | 3.4s   | 0.005s|
| 12                     | SO    | TO   | 5.5s    | 51m | 5s     | 0.063s|
| 14                     |       |      |         |     |        |        |
| Wide Mouth Frog        | 3     | 0.05s| 0.005s | 8s  | 0.006s| 0.002s|
| 6                      | 0.4s  | 0.4s | 52m     | 0.2s | 0.03s  |        |
| 7                      | 1.4s  | 1.9s | MO      | 2.3s | 0.004s |        |
| 10                     | 46s   | 5m31s| 5s      |     |        |        |
| 12                     | 17m   | TO   | 1m      |     |        |        |
| 14                     | 7m    | TO   | 4m20s   |     |        |        |
| Needham - Schroeder    | 3     | 0.1s | 0.4s   | 52s | 0.003s|        |
| 6                      | 20s   | 8m   | MO      | 4s  |        |        |
| 7                      | 2m    | TO   | 36s     | 36s |        |        |
| 10                     | SO    |      | 1m50s   |     |        |        |
| 12                     |       |      | 4m47s   |     |        |        |
| 14                     |       |      | 11m     |     |        |        |
| Yahalom - Lowe         | 3     | 0.16s| 3.6s   | 6s  | 1.4s   | 0.003s|
| 6                      | 33s   | 44s  | 132m    | 1m  |        |        |
| 7                      | 11m   | 36m  | MO      | 17m |        |        |
| 10                     | SO    |      | 63m     |     |        |        |
| 12                     |       |      | 0.009s  |     |        |        |
| 14                     |       |      | 0.05s   |     |        |        |
| Otway-Rees             | 3     | 2m   | BUG     | 1.7s| 0.004s| 0.011s|
| 6                      | 2m    | SO   | 27m     |     |        | 0.012s|
| 7                      |       |      | MO      |     |        | 0.02s |
| 10                     |       |      |         |     |        | 0.03s |
| 12                     |       |      |         |     |        | 0.1s  |
| 14                     |       |      |         |     |        | 0.007s|
| Helios                 | 3     | x    | TO      | x   | 0.002s|        |
| 6                      | 2m    | BUG  | MO      |     |        |        |
| 7                      |       |      |         |     |        |        |
| 10                     |       |      |         |     |        |        |
| 12                     |       |      |         |     |        |        |
| 14                     |       |      |         |     |        |        |

*Figure 9: Experimental results for the bounded case*
| Protocols      | ProVerif | TypeEq |
|---------------|----------|--------|
| Helios        | x        | 0.003s |
| Denning-Sacco | 0.05s    | 0.05s  |
| Needham-Schroeder-Lowe | 0.08s | 0.09s |

Figure 10: Experimental results for unbounded numbers of sessions

is that the weeding procedure makes Tamarin enter a loop where it cannot detect that, as soon as a ballot is not weed, it has been forged by the adversary.

For the sake of comparison, we run both tools (ProVerif and TypeEq) on a symmetric protocol (Denning-Sacco) and an asymmetric protocol (Needham-Schroeder-Lowe). The execution times are very similar. The corresponding results are reported in Figure 10.

8 CONCLUSION

We presented a novel type system for verifying trace equivalence in security protocols. It can be applied to various protocols, with support for else branches, standard cryptographic primitives, as well as a bounded and an unbounded number of sessions. We believe that our prototype implementation demonstrates that this approach is promising and opens the way to the development of an efficient technique for proving equivalence properties in even larger classes of protocols.

Several interesting problems remain to be studied. For example, a limitation of ProVerif is that it cannot properly handle global states. We plan to explore this case by enriching our types to express the fact that an event is “consumed”. Also, for the moment, our type system only applies to protocols $P, Q$ that have the same structure. One advantage of a type system is its modularity: it is relatively easy to add a few rules without redoing the whole proof. We plan to add rules to cover protocols with different structures (e.g. when branches are swapped). Another direction is the treatment of primitives with algebraic properties (e.g. Exclusive Or, or homomorphic encryption). It seems possible to extend the type system and discharge the difficulty to the consistency of the constraints, which seems easier to handle (since this captures the static case). Finally, our type system is sound w.r.t. equivalence in a symbolic model. An interesting question is whether it also entails computational indistinguishability. Again, we expect that an advantage of our type system is the possibility to discharge most of the difficulty to the constraints.
APPENDIX A  TYPING RULES AND DEFINITIONS

We give on Figures 11 and 12 a complete version of our typing rules for processes, as well as the formal definition of the well-formedness judgement for typing environments.

In this section, we also provide additional definitions (or more precise versions of previous definitions) regarding constraints, and especially their consistency, that the proofs require.

Definition A.1 (Constraint). A constraint is defined as a couple of messages, separated by the symbol \( \sim \):

$$u \sim v$$
We will consider sets of constraints, which we usually denote $c$. We will also consider couples $(c, \Gamma)$ composed of such a set, and a typing environment $\Gamma$. Finally we will denote sets of such tuples $C$, and call them constraint sets.

**Definition A.2 (Compatible environments).** We say that two typing environments $\Gamma, \Gamma'$ are compatible if they are equal on the intersection of their domains, i.e.

$$\forall x \in \text{dom}(\Gamma) \cap \text{dom}(\Gamma'). \Gamma(x) = \Gamma'(x)$$

**Definition A.3 (Union of environments).** Let $\Gamma, \Gamma'$ be two compatible environments. Their union $\Gamma \cup \Gamma'$ is defined by

- $\text{dom}(\Gamma \cup \Gamma') = \text{dom}(\Gamma) \cup \text{dom}(\Gamma')$
- $\forall x \in \text{dom}(\Gamma). \ (\Gamma \cup \Gamma')(x) = \Gamma(x)$
- $\forall x \in \text{dom}(\Gamma'). \ (\Gamma \cup \Gamma')(x) = \Gamma'(x)$

Note that this function is well defined since $\Gamma$ and $\Gamma'$ are assumed to be compatible.

**Definition A.4 (Operations on constraint sets).** We define two operations on constraints.

- the product union of constraint sets:
  $$C \cup X := \{(c \cup c', \Gamma \cup \Gamma') \mid (c, \Gamma) \in C \land (c', \Gamma') \in C' \land \Gamma, \Gamma' \text{ are compatible}\}$$

- the addition of a set of constraints $c'$ to all elements of a constraint set $C$:
  $$C \cup c' := C \cup \{c' \mid (c, \Gamma) \in C\}$$

**Definition A.5.** For any typing environment $\Gamma$, we denote by $\Gamma|_X$ its restriction to variables, by $\Gamma|_N$ its restriction to names and keys, and by $E_\sigma$ the set of the names it contains, i.e. $N \cap \text{dom}(\Gamma)$.

**Definition A.6 (Well-typed substitutions).** Let $\Gamma$ be a typing environment, $\theta, \theta'$ two substitutions, and $c$ a set of constraints. We say that $\theta, \theta'$ are well-typed in $\Gamma$, and that $\Gamma|_X \vdash \theta \sim \theta' : \Gamma|_X \rightarrow c$, if they are ground and

- $\text{dom}(\theta) = \text{dom}(\theta') = \text{dom}(\Gamma|_X)$,
- and

  $$\forall x \in \text{dom}(\Gamma|_X), \ \Gamma|_X \vdash \theta(x) \sim \theta'(x) : \Gamma(x) \rightarrow c_x$$

  for some $c_x$ such that $c = \bigcup_{x \in \text{dom}(\Gamma|_X)} c_x$.

**Definition A.7 (LL substitutions).** Let $\Gamma$ be an environment, $\phi, \phi'$ two substitutions and $c$ a set of constraints. We say that $\phi, \phi'$ have type LL in $\Gamma$ with constraint $c$, and write $\Gamma \vdash \phi \sim \phi' : \text{LL} \rightarrow c$ if

- $\text{dom}(\phi) = \text{dom}(\phi')$;
- for all $x \in \text{dom}(\phi)$ there exists $c_x$ such that $\Gamma \vdash \phi(x) \sim \phi'(x) : \text{LL} \rightarrow c_x$ and $c = \bigcup_{x \in \text{dom}(\phi)} c_x$.

**Definition A.8 (Frames associated to a set of constraints).** If $c$ is a set of constraints, let $\phi(c)$ and $\phi(c)$ be the frames composed of the terms respectively on the left and on the right of the $\sim$ symbol in the constraints of $c$ (in the same order).

**Definition A.9 (Instantiation of constraints).** If $c$ is a set of constraints, and $\sigma, \sigma'$ are two substitutions, let $[c]_{\sigma', \sigma}$ be the instantiation of $c$ by $\sigma$ on the left and $\sigma'$ on the right, i.e.

$$[c]_{\sigma', \sigma} = \{M\sigma \sim N\sigma' \mid M \sim N \in c\}.$$

Similarly we write for a constraint set $C$

$$[C]_{\sigma', \sigma} = \{([c]_{\sigma', \sigma}, \Gamma) \mid (c, \Gamma) \in C\}.$$
Definition A.10 (Frames associated to environments). If $\Gamma$ is a typing environment, we denote $\phi^{ffi}_{\Gamma}$ the frame containing all the keys $k$ such that $\Gamma(k) = \text{key}^{LL}(T)$ for some $T$, all the public keys $pk(k)$ and $vk(k)$ for $k \in \text{dom}(\Gamma)$, and all the nonces $n$ such that $\Gamma(n) = r_{a}^{\Gamma}$ (for $a \in \{\infty, 1\}$).

Definition A.11 (Branches of a type). If $T$ is a type, we write branches($T$) the set of all types $T'$ such that $T'$ is not a union type, and either

- $T = T'$;
- or there exist types $T_1, \ldots, T_k, T_1', \ldots, T_k'$ such that
  \[ T = T_1 \lor \ldots \lor T_k \lor T \lor T_1' \lor \ldots \lor T_k'. \]

Definition A.12 (Branches of an environment). For a typing environment $\Gamma$, we write branches($\Gamma$) the sets of all environments $\Gamma'$ such that

- $\text{dom}(\Gamma') = \text{dom}(\Gamma)$
- $\forall x \in \text{dom}(\Gamma), \Gamma'(x) \in \text{branches}(\Gamma(x))$.

Definition A.13 (Consistency). We say that $c$ is consistent in a typing environment $\Gamma$, if for any subsets $c' \subseteq c$ and $\Gamma' \subseteq \Gamma$ such that $\Gamma'_{N_{\|}} = \Gamma_{N_{\|}}$ and $\text{vars}(c') \subseteq \text{dom}(\Gamma')$, for any ground substitutions $\sigma$, $\sigma'$ well-typed in $\Gamma'$, the frames new $E_{\Gamma'.\phi'(c')\sigma}$ and new $E_{\Gamma'.\phi'(c')\sigma'}$ are statically equivalent.

We say that $(c, \Gamma)$ is consistent if $c$ is consistent in $\Gamma$.

We say that a constraint set $C$ is consistent if each element $(c, \Gamma) \in C$ is consistent.

APPENDIX B PROOFS

In this section, we provide the detailed proofs to all of our theorems.

Unless specified otherwise, the environments $\Gamma$ considered in the lemmas are implicitly assumed to be well-formed.

B.1 General results and soundness

In this subsection, we prove soundness for non replicated processes, as well as several results regarding the type system that this proof uses.

Lemma B.1 (Subtyping properties). The following properties of subtyping hold:

1. $\forall T. \text{HL} < T \implies T = \text{HL}$.
2. $\forall T. \text{LL} < T \implies T = \text{LL} \lor T = \text{HL}$.
3. $\forall T. \text{HH} < T \implies T = \text{HH} \lor T = \text{HL}$.
4. $\forall T_1, T_2, T_3. T_1 * T_2 < T_3 \implies T_3 = \text{LL} \lor T_3 = \text{HL} \lor T_3 = \text{HH} \lor (\exists T_4, T_5. T_3 = T_4 * T_5)$ i.e. $T_3$ is LL, HL, HH or a pair type.
5. $\forall T_1, T_2. T < T_1 + T_2 \implies (\exists T'_1, T'_2. T = T'_1 \land T'_2 \land T_1 < T_2 \land T_2 < T_1)$.
6. $\forall T_1, T_2. T_1 + T_2 < \text{LL} \implies T_1 < \text{LL} \land T_2 < \text{LL}$.
7. $\forall T_1, T_2. T_1 + T_2 < \text{HH} \implies T_1 < \text{HH} \land T_2 < \text{HH}$.
8. $\forall T_1, T_2. T_1 < (T_2)_{k_1} \implies (\exists T_3 < T_2. T_1 = (T_3)_{k_1})$.
9. $\forall T_1, T_2, k. T_1 < (T_2)_{k_1} \implies (\exists T_3 < T_2. T_1 = (T_3)_{k_1})$.
10. $\forall T_1, T_2, k. (T_1)_{k_1} < T_2 \implies T_2 = \text{HL} \lor (\exists T_3. T_1 < T_3 \land T_2 = (T_3)_{k_1})$.
11. $\forall T_1, T_2, k. (T_1)_{k_1} < T_2 \implies T_2 = \text{HL} \lor (\exists T_3. T_1 < T_3 \land T_2 = (T_3)_{k_1})$.
12. $\forall T, m, n, l. T < (\llbracket r_{m}^{l} : r_{n}^{l} \rrbracket, a) \implies T = (\llbracket r_{m}^{l} : r_{n}^{l} \rrbracket, a)$.
13. $\forall T, m, n, l'. T < (\llbracket r_{m}^{l} : r_{n}^{l} \rrbracket, a) \implies T = \text{HL} \lor T = (\llbracket r_{m}^{l} : r_{n}^{l} \rrbracket, a)$.
14. $\forall T, m, n, l'. T < (\llbracket r_{m}^{l} : r_{n}^{l} \rrbracket, a) \implies T = \text{HL} \lor T = (\llbracket r_{m}^{l} : r_{n}^{l} \rrbracket, a)$.
15. $\forall T, l, l'. T < (\text{key}^{l}(T')) \implies T = \text{key}^{l}(T')$.
16. $\forall T. T < \text{LL} \implies T < (\exists T'. T = \text{key}^{LL}(T')) \lor T = \text{LL}$.
17. $\forall T. T < \text{HH} \implies T < (\exists T'. T = \text{key}^{HH}(T')) \lor T = \text{HH}$.

Proof. Points 1 to 3 are immediate by induction on the subtyping proof (the proofs of the second and third points use the first one in the STRANS case).

Points 4 to 3 are immediate by induction on the subtyping proof (the proofs of the second and third points use the first one in the STRANS case).

Points 5 to 3 are proved by induction on the proof of $T < T_1 * T_2$ (the STRANS case uses the induction hypothesis).

Points 6 and 7 are proved by induction on the proof of $T < T_1 * T_2$ (resp. HH; using the previous points in the STRANS case).

Points 8 and 9 are immediate by induction on the subtyping proof.

Points 10 to 3 are proved by induction on the subtyping proof (using the first two points in the STRANS case).

Points 14 and 15 are immediate by induction on the subtyping proof.

Points 16 and 17 are proved by induction on the subtyping proof, using points 5, 8, 9 in the STRANS case.
Lemma B.2 (Terms of Type \( T \lor T' \)). For all \( \Gamma, T, T' \), for all ground terms \( t, t' \), for all \( c \), if
\[
\Gamma \vdash t \sim t' : T \lor T' \rightarrow c
\]
then
\[
\Gamma \vdash t \sim t' : T \rightarrow c \quad \text{or} \quad \Gamma \vdash t \sim t' : T' \rightarrow c
\]

Proof. We prove this property by induction on the derivation of \( \Gamma \vdash t \sim t' : T \lor T' \rightarrow c \).

The last rule of the derivation cannot be TNonce, TNoncl, TCurlFN, TPair, TKey, TPubKey, TVKey, TEnc, TEncH, TEncL, TAEnc, TSignH, TSignL, THash, THashL, THigh, TLR1, TLR\text{m}'', TLRVar, TLR', or TLRL' since the type in their conclusion cannot be \( T \lor T' \). It cannot be TVar since \( t, t' \) are ground.

In the TSub case we know that \( \Gamma \vdash t \sim t' : T'' \rightarrow c \) (with a shorter derivation) for some \( T'' = T \lor T' \); thus, by Lemma B.1, \( T'' = T \lor T' \), and the claim holds by the induction hypothesis.

Finally in the TOR case, the premise of the rule directly proves the claim. \( \Box \)

Lemma B.3 (Terms and Branch Types). For all \( \Gamma, T, c \), for all ground terms \( t, t' \), if
\[
\Gamma \vdash t \sim t' : T \rightarrow c
\]
then there exists \( T' \in \text{branches}(T) \) such that
\[
\Gamma \vdash t \sim t' : T' \rightarrow c
\]

Proof. This property is a corollary of Lemma B.2. We indeed prove it by successively applying this lemma to \( \Gamma \vdash t \sim t' : T \rightarrow c \) until \( T \) is not a union type. \( \Box \)

Lemma B.4 (Substitutions Type in a Branch). For all \( \Gamma, c \), for all ground substitutions \( \sigma, \sigma' \), if
\[
\Gamma_{N,K} \vdash \sigma \sim \sigma' : \Gamma_X \rightarrow c
\]
then there exists \( \Gamma' \in \text{branches}(\Gamma) \) such that
\[
\Gamma'_{N,K} \vdash \sigma \sim \sigma' : \Gamma'_X \rightarrow c
\]

Proof. This property follows from Lemma B.3. Indeed, by definition, \( c = \bigcup_{x \in \text{dom}(\Gamma)} c_x \) for some \( c_x \) such that for all \( x \in \text{dom}(\Gamma)(= \text{dom}(\sigma) = \text{dom}(\sigma')) \),
\[
\Gamma \vdash \sigma(x) \sim \sigma'(x) : \Gamma(x) \rightarrow c_x
\]
Hence by applying Lemma B.3 we obtain a type \( T_x \in \text{branches}(\Gamma(x)) \) such that
\[
\Gamma \vdash \sigma(x) \sim \sigma'(x) : T_x \rightarrow c_x
\]
Thus if we denote \( \Gamma'' \) by \( \forall x \in \text{dom}(\Gamma_X).\Gamma''(x) = T_x \), and \( \Gamma' = \Gamma_{N,K} \cup \Gamma'' \), we have \( \Gamma' \in \text{branches}(\Gamma) \) and \( \Gamma'_{N,K} \vdash \sigma \sim \sigma' : \Gamma'_X \rightarrow c \). \( \Box \)

Lemma B.5 (Typing Terms in Branches). For all \( \Gamma, T, c \), for all terms \( t, t' \), for all \( \Gamma' \in \text{branches}(\Gamma) \), if \( \Gamma \vdash t \sim t' : T \rightarrow c \) then \( \Gamma' \vdash t \sim t' : T \rightarrow c \).

Corollary: in that case, there exists \( T' \in \text{branches}(T) \) such that \( \Gamma' \vdash t \sim t' : T' \rightarrow c \).

Proof. We prove this property by induction on the derivation of \( \Gamma \vdash t \sim t' : T \rightarrow c \). In most cases for the last rule applied, \( \Gamma(x) \) is not directly involved in the premises, for any variable \( x \). Rather, \( \Gamma \) appears only in other typing judgements, or is used in \( \Gamma(k) \) or \( \Gamma(n) \) for some key \( k \) or nonce \( n \), and keys or nonces cannot have union types. Hence, since the typing rules for terms do not change \( \Gamma \), the claim directly follows from the induction hypothesis. For instance in the TPair case, we have \( t = (t_1, t_2), t' = (t'_1, t'_2) \), \( T = T_1 \rightarrow T_2, c = c_1 \cup c_2 \), \( \Gamma \vdash t_1 \sim t'_1 : T_1 \rightarrow c_1 \) and \( \Gamma \vdash t_2 \sim t'_2 : T_2 \rightarrow c_2 \), thus by the induction hypothesis, \( \Gamma' \vdash t_1 \sim t'_1 : T_1 \rightarrow c_1 \) and \( \Gamma' \vdash t_2 \sim t'_2 : T_2 \rightarrow c_2 \), and therefore by rule TPair, \( \Gamma' \vdash t \sim t' : T \rightarrow c \). The cases of rules TEnc, TEncH, TEncL, TAEnc, TAEncH, TAEncL, THash, THashL, TSignH, TSignL, TLR', TLRVar, TSub, TOR are similar.

The cases of rules TNonce, TNoncl, TCat, TKey, TPubKey, TVKey, THash, THigh, TLR1, and TLRm'' are immediate since these rules use neither \( \Gamma \) nor another typing judgement in their premise.

Finally, in the TVar case, \( t = t' = x \) for some variable \( x \) such that \( \Gamma(x) = T \), and \( c = \emptyset \). Rule TVar also proves that \( \Gamma' \vdash x \sim x : \Gamma'(x) \rightarrow \emptyset \). Since \( \Gamma'(x) \in \text{branches}(\Gamma'(x)) \), by applying rule TOR as many times as necessary, we have \( \Gamma' \vdash x \sim x : \Gamma'(x) \rightarrow \emptyset \), i.e. \( \Gamma' \vdash x \sim x : T \rightarrow \emptyset \), which proves the claim.

The corollary then follows, again by induction on the typing derivation. If \( T \) is not a union type, \( \text{branches}(T) = \{ T \} \) and the claim is directly the previous property. Otherwise, the last rule applied in the typing derivation can only be TVar, TSub, or TOR. The TSub case follows trivially from the induction hypothesis; since \( T \) is a union type, it is its own only subtype. In the TVar case, \( t = t' = x \) for some variable \( x \) such that \( \Gamma(x) = T \). Hence, by definition, \( \Gamma'(x) \in \text{branches}(T) \), and by rule TVar we have \( \Gamma' \vdash t \sim t' : \Gamma'(x) \rightarrow c \). Finally, in the TOR case, we have \( T = T_1 \lor T_2 \) for some \( T_1, T_2 \) such that \( \Gamma \vdash t \sim t' : T_1 \rightarrow c \). By the induction hypothesis, there exists \( T'_1 \in \text{branches}(T_1) \) such that \( \Gamma' \vdash t \sim t' : T'_1 \rightarrow c \). Since, by definition, \( \text{branches}(T_1) \subseteq \text{branches}(T_1 \lor T_2) \), this proves the claim. \( \Box \)
**Lemma B.6 (Typing destructors in branches).** For all $\Gamma, T, d, x$, for all $\Gamma' \in \text{branches}(\Gamma)$, if $\Gamma \vdash d(x) : T$ then $\Gamma' \vdash d(x) : T$.

**Proof.** This property is immediate by examining the typing rules for destructors. Indeed, $\Gamma$ and $\Gamma'$ only differ on variables, and the rules for destructors only involve $\Gamma(x)$ for $x \in X$ in conditions of the form $\Gamma(x) = T$ for some type $T$ which is not a union type.

Hence in these cases $\Gamma'(x)$ is also $T$, and the same rule can be applied to $\Gamma'$ to prove the claim. $\square$

**Lemma B.7 (Typing processes in branches).** For all $\Gamma, C, \Pi$ for all processes $P, Q$, for all $\Gamma' \in \text{branches}(\Gamma)$, if $\Gamma \vdash P \rightarrow Q \rightarrow C$ then there exists $C' \subseteq C$ such that $\Gamma' \vdash P \rightarrow Q \rightarrow C'$.

**Proof.** We prove this lemma by induction on the derivation of $\Gamma \vdash P \rightarrow Q \rightarrow C$. In all the cases for the last rule applied in this derivation, we can show that the conditions of this rule still hold in $\Gamma'$ (instead of $\Gamma$) using

- Lemma B.5 for the conditions of the form $\Gamma \vdash T : C$;
- Lemma B.6 for the conditions of the form $\Gamma \vdash d(y) : T$;
- the fact that if $\Gamma(x)$ is not a union type, then $\Gamma'(x) = \Gamma(x)$, for conditions such as $\Gamma(x) = \text{LL}$, $\Gamma(x) = \llbracket \tau^1_a ; \tau^2_a \rrbracket$ (in the PLET case);
- the induction hypothesis for the conditions of the form $\Gamma \vdash P' \rightarrow Q' \rightarrow C''$. In this case, the induction hypothesis produces a $C'' \subseteq C''$, which can then be used to show $C' \subseteq C$, since $C'$ and $C$ are usually respectively $C''$ and $C''$ with some terms added.

We detail here the cases of rules POUT, PPAR, and PON. The other cases are similar, as explained above.

If the last rule is POUT, then we have $P = \text{out}(M), P', Q = \text{out}(N), Q', C = C' \cup \forall c$ for some $P, Q', M, N, C''$, such that $\Gamma \vdash P' \rightarrow Q' \rightarrow C'$ and $\Gamma \vdash M \rightarrow N : \text{LL} \rightarrow c$. Hence by Lemma B.5, $\Gamma' \vdash M \rightarrow N : \text{LL} \rightarrow c$ and by the induction hypothesis applied to $P', Q'$, $\Gamma' \vdash P' \rightarrow Q' \rightarrow C''$ for some $C''$ such that $C'' \subseteq C''$. Therefore by rule POUT, $\Gamma' \vdash P \rightarrow Q \rightarrow C'' \cup \forall c$, and since $C'' \cup \forall c \subseteq C'' \cup \forall c (\equiv C)$, this proves the claim.

If the last rule is PPAR, then we have $P = P_1 | P_2$, $Q = Q_1 \mid Q_2$, $C = C_1 \cup C_2$ for some $P_1, P_2, Q_1, Q_2, C_1, C_2$ such that $\Gamma \vdash P_1 \rightarrow Q_1 \rightarrow C_1$ and $\Gamma \vdash P_2 \rightarrow Q_2 \rightarrow C_2$. Thus by applying the induction hypothesis twice, we have $\Gamma' \vdash P_1 \rightarrow Q_1 \rightarrow C'_1$ and $\Gamma' \vdash P_2 \rightarrow Q_2 \rightarrow C'_2$ with $C'_1 \subseteq C_1$ and $C'_2 \subseteq C_2$. Therefore by rule PPAR, $\Gamma' \vdash P_1 | P_2 \rightarrow Q_1 \lor Q_2 \rightarrow C'_1 \cup C'_2$, and since $C'_1 \cup C'_2 \subseteq C_1 \cup C_2$, this proves the claim.

If the last rule is PON, then there exist $\Gamma''$, $x, T_1, T_2, C_1$ and $C_2$ such that $\Gamma = \Gamma''$, $x : T_1 \lor T_2$, $C = C_1 \cup C_2$, $\Gamma''$, $x : T_1 \rightarrow Q \rightarrow C_1$ and $\Gamma''$, $x : T_2 \rightarrow P \rightarrow Q \rightarrow C_2$. By definition of branches, it is clear that branches($\Gamma$) $= \text{branches}(\Gamma'', x : T_1 \lor T_2)$ $= \text{branches}(\Gamma'', x : T_1 \lor T_2) \cup \text{branches}(\Gamma'', x : T_2 \lor T_2)$. Thus, since $\Gamma' \in \text{branches}(\Gamma)$, we know that $\Gamma' \in \text{branches}(\Gamma'', x : T_1)$ or $\Gamma' \in \text{branches}(\Gamma'', x : T_2)$. We write the proof for the case where $\Gamma' \in \text{branches}(\Gamma'', x : T_1)$, the other case is analogous. By applying the induction hypothesis to $\Gamma''$, $x : T_1 \rightarrow P \rightarrow Q \rightarrow C_1$, there exists $C'_1 \subseteq C_1$ such that $\Gamma' \vdash P \rightarrow Q \rightarrow C'_1$. Since $C_1 \subseteq C$, this proves the claim. $\square$

**Lemma B.8 (Environments in the constraints).** For all $\Gamma, C, \Pi$ for all processes $P, Q$, if

$$\Gamma \vdash P \rightarrow Q \rightarrow C$$

then for all $(c, \Gamma') \in C$,

$$\text{dom}(\Gamma') \subseteq \text{dom}(\Gamma) \cup \text{bvars}(P) \cup \text{bvars}(Q) \cup \text{nnames}(P) \cup \text{nnames}(Q)$$

(where $\text{bvars}(P), \text{nnames}(P)$ respectively denote the sets of bound variables and names in $P$).

**Proof.** We prove this lemma by induction on the typing derivation of $\Gamma \vdash P \rightarrow Q \rightarrow C$.

If the last rule applied in this derivation is PZERO, we have $C = \{(0, \Gamma')\}$, and the claim clearly holds.

In the PPAR case, we have $P = P_1 | P_2$, $Q = Q_1 | Q_2$, and $C = C_1 \cup C_2$ for some $P_1, P_2, Q_1, Q_2, C_1, C_2$ such that $\Gamma \vdash P_1 \rightarrow Q_1 \rightarrow C_1$ and $\Gamma \vdash P_2 \rightarrow Q_2 \rightarrow C_2$. Thus any element of $C$ is of the form $(c_1 \cup c_2, \Gamma_1 \cup \Gamma_2)$ where $(c_1, \Gamma_1) \in C_1$, $(c_2, \Gamma_2) \in C_2$, and $\Gamma_1, \Gamma_2$ are compatible. By the induction hypothesis, $\text{dom}(\Gamma_1) \subseteq \text{dom}(\Gamma) \cup \text{bvars}(P_1) \cup \text{bvars}(Q_1) \cup \text{nnames}(P_1) \cup \text{nnames}(Q_1) \subseteq \text{dom}(\Gamma) \cup \text{bvars}(P) \cup \text{bvars}(Q) \cup \text{nnames}(P) \cup \text{nnames}(Q)$, and similarly for $\Gamma_2$. Therefore, since $\text{dom}(\Gamma_1 \cup \Gamma_2) = \text{dom}(\Gamma_1) \cup \text{dom}(\Gamma_2)$ (by definition), the claim holds.

In the PBN and PLET cases, the typing judgement appearing in the condition of the rule uses $\Gamma$ extended with an additional variable, which is bound in $P$ and $Q$. We detail the PBN case, the other case is similar. We have $P = \text{in}(x).P', Q = \text{in}(x).Q'$ for some $x, P', Q'$ such that $x \notin \text{dom}(\Gamma)$ and $\Gamma, x : \text{LL} \vdash P' \rightarrow Q' \rightarrow C$. Hence by the induction hypothesis, if $(c, \Gamma') \in C, \text{dom}(\Gamma') \subseteq \text{dom}(\Gamma, x : \text{LL}) \cup \text{bvars}(P') \cup \text{bvars}(Q') \cup \text{nnames}(P') \cup \text{nnames}(Q')$. Since $\text{bvars}(P) = \{x\} \cup \text{bvars}(P')$ and $\text{bvars}(Q) = \{x\} \cup \text{bvars}(Q')$, this proves the claim.

The case of rule PNEW is similar, extending $\Gamma$ with a nonce instead of a variable.

In the POUT case, there exist $P', Q', M, N, C'$ such that $P = \text{out}(M).P', Q = \text{out}(N).Q', C = C' \cup \forall c, \Gamma \vdash M \rightarrow N : \text{LL} \rightarrow c$ and $\Gamma \vdash P' \rightarrow Q' \rightarrow C'$. If $(c', \Gamma') \in C'$, by definition of $\forall c$ there exists $c''$ such that $(c'', \Gamma') \in C'$ and $c' = c \cup c''$. By the induction hypothesis, we thus have

$$\text{dom}(\Gamma') \subseteq \text{dom}(\Gamma) \cup \text{bvars}(P') \cup \text{bvars}(Q') \cup \text{nnames}(P') \cup \text{nnames}(Q')$$

and since $\text{bvars}(P') = \text{bvars}(P), \text{nnames}(P') = \text{nnames}(P)$, and similarly for $Q$, this proves the claim.

In the PBLL case, there exist $P', P'', Q', Q'', M, N, M', N', C', C'', c, c'$ such that $P = \text{if } M = M' \text{ then } P' \text{ else } P'', Q = \text{if } N = N' \text{ then } Q' \text{ else } Q'', C = (C' \cup C'') \cup \forall c (c \cup c'), \Gamma \vdash M \rightarrow N : \text{LL} \rightarrow c, \Gamma \vdash M' \rightarrow N' : \text{LL} \rightarrow c', \Gamma \vdash P' \rightarrow Q' \rightarrow C', \Gamma \vdash P'' \rightarrow Q'' \rightarrow C'$. 

```
If $(c'',\Gamma') \in C$, by definition of $\cup \nu$ there exist $c'''$, such that $(c''',\Gamma'') \in C' \cup C''$ and $c'' = c''' \cup c \cup c'$. We write the proof for the case where $(c''',\Gamma'') \in C'$, the other case is analogous. By the induction hypothesis, we thus have
\[ \text{dom}(\Gamma') \subseteq \text{dom}(\Gamma) \cup \text{bvars}(P') \cup \text{bvars}(Q') \cup \text{nnames}(P') \cup \text{nnames}(Q') \]
and since $\text{bvars}(P') \subseteq \text{bvars}(P)$, $\text{nnames}(P') \subseteq \text{nnames}(P)$, and similarly for $Q$, this proves the claim.

The cases of rules $\text{PO}_{\text{R}}$, $\text{PL}_{\text{ER}}$, $\text{PL}_{\text{R}}$, $\text{Pf}_{\text{R}}$, $\text{Pf}_{\text{I}}$, and $\text{Pf}_{\text{LR}}$ remain. All these cases are similar, we write the proof for the $\text{Pf}_{\text{LR}}$ case. In this case, there exist $P'$, $P''$, $Q'$, $Q''$, $M$, $N$, $M'$, $N'$, $c'$, $c''$, $l$, $l'$, $m$, $n$, such that $P = \text{if } M = M' \text{ then } P'' \text{ else } P'''$, $Q = \text{if } N = N' \text{ then } Q' \text{ else } Q''$, $C = C' \cup C''$, $\Gamma + M \sim N : [l_m : l_m', r_n : r_n'] \rightarrow \emptyset$, $\Gamma + P' \sim Q' \rightarrow C'$, and $\Gamma + P'' \sim Q'' \rightarrow C''$. If $(c,\Gamma') \in C$, we thus know that $(c,\Gamma') \in C'$ or $(c,\Gamma') \in C''$. We write the proof for the case where $(c,\Gamma') \in C'$, the other case is analogous. By the induction hypothesis, we thus have
\[ \text{dom}(\Gamma') \subseteq \text{dom}(\Gamma) \cup \text{bvars}(P') \cup \text{bvars}(Q') \cup \text{nnames}(P') \cup \text{nnames}(Q') \]
and since $\text{bvars}(P') \subseteq \text{bvars}(P)$, $\text{nnames}(P') \subseteq \text{nnames}(P)$, and similarly for $Q$, this proves the claim.

**Lemma B.9 (Environments in the constraints do not contain union types).** For all $\Gamma, C$, for all processes $P, Q$, if
\[ \Gamma \vdash P \sim Q \rightarrow C \]
then for all $(c,\Gamma') \in C$,
\[ \text{branches}(\Gamma') = \{\Gamma'\} \]
i.e. for all $x \in \text{dom}(\Gamma')$, $\Gamma'(x)$ is not a union type.

**Proof.** This property is immediate by induction on the typing derivation.

**Lemma B.10 (Typing is preserved by extending the environment).** For all $\Gamma, \Gamma'$, $P, Q, C, t, t', T, c$, if $\Gamma \vdash t \rightarrow t' ; T \rightarrow c$, then $\Gamma \cup \Gamma' \vdash t \rightarrow t' ; T \rightarrow c$.

**Proof.** The first point is immediate by induction on the type derivation.

The second point is immediate by examining the typing rules for destructors.

The third point is immediate by induction on the type derivation of the processes. In the $\text{PZERO}$ case, to satisfy the condition that the environment is its own only branch, rule $\text{PO}_{\text{R}}$ needs to be applied first, in order to split all the union types in $\Gamma'$, which yields the environments branches($\Gamma \cup \Gamma'$) in the constraints.

**Lemma B.11 (Consistency for Subsets).** The following statements about constraints hold:

1. If $(c,\Gamma)$ is consistent, and $c' \subseteq c$ then $(c',\Gamma)$ is consistent.
2. Let $C$ be a consistent constraint set. Then every subset $C' \subseteq C$ is also consistent.
3. If $C \cup C'$ is consistent then $C$ also is.
4. If $C_1 \subseteq C_2$ and $C'_1 \subseteq C'_2$, then $C_1 \cup C'_1 \subseteq C_2 \cup C'_2$.
5. If $C$ is closed under $\cup$, $\cup \nu$, and $\sigma \vdash \nu$.
6. If $\sigma_1$ and $\sigma_2$ are disjoint domains, and $\Gamma' \vdash x : \sigma_1 \rightarrow c$ and $\Gamma' \vdash x : \sigma_2 \rightarrow c$, then $\Gamma' \vdash x : \sigma_1 \cup \sigma_2 \rightarrow c$.
7. If $C$ is consistent, if $\Gamma_{N,K} \vdash \sigma \rightarrow \sigma'$, then $\Gamma_X \rightarrow c$ for some $c$ and if for all $(c',\Gamma') \in C$, $\Gamma, \Gamma' \rightarrow c$, then $\Gamma, \Gamma' \rightarrow c$ is consistent.

**Proof.** Points 1 and 2 follow immediately from the definition of consistency and of static equivalence.

Point 3 follows from the point 1: for every $(c,\Gamma) \in C$, $(c \cup c',\Gamma)$ is in $C \cup C'$, which is consistent since $C \cup C'$ is; therefore $(c,\Gamma)$ also is.

Point 4 follows from the definition of $\cup \nu$, if $(c,\Gamma) \in C \cup C'$, there exists $(c_1,\Gamma_1) \in C_1$, $(c_2,\Gamma_2) \in C_2$ such that $(c,\Gamma) = (c_1 \cup c_2,\Gamma_1 \cup \Gamma_2)$ and $(c_1,\Gamma_1)$ and $(c_2,\Gamma_2)$ are compatible. Since $C_1 \subseteq C_2$, $(c_1,\Gamma_1)$ is in $C_2$. Similarly, $(c_2,\Gamma_2)$ is in $C_1$. Therefore $(c,\Gamma) \in C \cup C'$.

Points 5 and 6 follow from the definitions of $\cup \nu$, $\cup \nu$, and of consistency. Indeed, let $(c'',\Gamma'') \in [C]_{\sigma_1,\sigma_2}$. There exists $c'''$ such that $c'' = [c''']_{\sigma_1,\sigma_2}$, and $(c''',\Gamma''') \in C$. Let $c_1 \subseteq c''$ and $\Gamma_1 \subseteq \Gamma''$ such that $\Gamma_{N,K} \vdash \Gamma_{N,K} \vdash \nu_{N,K}$ and $\sigma(c_1) \subseteq \text{dom}(\Gamma_1)$. Let $\sigma, \theta'$ be well-typed in $\Gamma_1$. Since $c'' = [c''']_{\sigma_1,\sigma_2}$, there exists $c_2 \subseteq c'''$ such that $c_1 = [c_2]_{\sigma_1,\sigma_2}$. If we show that $c_\sigma \theta$ and $\sigma' \theta'$ are well-typed in $\Gamma_1 \cup \Gamma_1$, it will follow from the consistency of $\Gamma_1$ that new $\text{E}_{\Gamma_1}(\$_{LL}L \cup \$_{LL}L \sigma \theta)$ and new $\text{E}_{\Gamma_1}(\#_{LL}L \cup \#_{LL}L \sigma' \theta')$ are statically equivalent, where $\Gamma_2 = \Gamma_1 \cup \Gamma_2 \subseteq \Gamma''$. Since $\Gamma_{N,K} \vdash \Gamma_{N,K} \vdash \nu_{N,K}$ and $\Gamma \cup \Gamma'$, we have $\text{E}_{\Gamma_1}(\$_{LL}L \cup \$_{LL}L \sigma \theta)$ and $\text{E}_{\Gamma_1}(\#_{LL}L \cup \#_{LL}L \sigma' \theta')$ are statically equivalent,
It only remains to be proved that σθ and σ′θ′ are well-typed in Γ₂.

Since c is ground, σθ = σ ∪ bvars(Γ), and similarly for σ′θ′. Hence, since σ, σ′ are well-typed in Γ, and θ, θ′ are well-typed in Γ₁, their compositions also are, which concludes the proof. □

Lemma B.12 (Environments in constraints contain a branch of the typing environment). For all Γ, C, for all processes P, Q, if Γ ⊢ P ~ Q → C then for all (c, Γ ′) ∈ C, there exists Γ ″ ∈ branches(Γ) such that Γ ″ ⊆ Γ ′.

Proof. We prove this property by induction on the type derivation of Γ ⊢ P ~ Q → C. In the PZERO case, C = {⟨0, Γ⟩}, and by assumption branches(Γ) = {Γ}, hence the claim trivially holds.

In the PPAR case, we have P = P₁ | P₂, Q = Q₁ | Q₂, and C = C₁∪∪C₂ for some P₁, P₂, Q₁, Q₂, C₁, C₂ such that Γ ⊢ P₁ ~ Q₁ → C₁ and Γ ⊢ P₂ ~ Q₂ → C₂. Thus any element of C is of the form (c₁∪∪C₁, Γ₁ ∪ Γ₂) where (c₁, Γ₁) ∈ C₁, (c₂, Γ₂) ∈ C₂, and Γ₁, Γ₂ are compatible. By the induction hypothesis, both C₁ and C₂ contain a branch of Γ. The claim holds, as these are necessarily the same branch, since Γ₁ and Γ₂ are compatible.

In the POR case, we have Γ = Γ ′′, x : T₁ ∨ T₂ for some x, Γ ′′, T₁, T₂ such that Γ ′′ ⊆ Γ ′′ ⊆ Γ ′. Hence by the induction hypothesis to Γ ⊢ P ~ Q → C ′, there exists Γ ′ ∈ branches(Γ) such that Γ ′ ⊆ Γ ′′.

In the POUT case, there exist P’, Q’, M, N, C, c such that P = out(M).P’, Q = out(N).Q’, C = C′∪∪c, Γ ⊢ M ~ N : LL → c and Γ ⊢ P’ ~ Q’ → C’. If (c, Γ ′) ∈ C, by definition of Uc there exists c′ such that (c′, Γ ′) ∈ C′ and c′ = c ∪ c′. Hence by applying the induction hypothesis to Γ ⊢ P’ ~ Q’ → C ′, there exists Γ ′ ∈ branches(Γ) such that Γ ′ ⊆ Γ ′′.

In the PlL case, there exist P’, P′′, Q’, M, N, M′, N′, C′, c, c such that P = if M = M′ then P′ else P′′, Q = if N = N′ then Q′ else Q′, C = C′∪∪C″∪∪(c∪∪c′), Γ ⊢ M ~ M′ : N : LL → c, Γ ⊢ M ′ ~ N′ : LL → c, Γ ⊢ P’ ~ Q’ → C′, and Γ ⊢ P′′ ~ Q′ → C″. If (c, Γ ′) ∈ C, by definition of Uc there exists c′, such that (c′, Γ ′) ∈ C′∪∪C″ and c′ = c ′′ ∪ c ∪ c′. We write the proof for the case where (c′, Γ ′) ∈ C, the other case is analogous. By applying the induction hypothesis to Γ ⊢ P’ ~ Q’ → C′, there exists Γ ′ ∈ branches(Γ) such that Γ ′ ⊆ Γ ′′, which proves the claim.

We write the proof for the PlL case. We consider the case where (c, Γ ′) ∈ C′, the other case is analogous. By applying the induction hypothesis to Γ ⊢ P′ ~ Q′ → C′, there exists Γ ′ ∈ branches(Γ) such that Γ ′ ⊆ Γ ′′, which proves the claim.

Lemma B.13 (All branches are represented in the constraints). For all Γ, C, for all processes P, Q, if Γ ⊢ P ~ Q → C then for all Γ ′ ∈ branches(Γ), there exists (c, Γ ′′) ∈ C, such that Γ ′ ⊆ Γ ′′.

Proof. We prove this property by induction on the type derivation of Γ ⊢ P ~ Q → C. In the PZERO case, C = {⟨0, Γ⟩}, and by assumption branches(Γ) = {Γ}, hence the claim trivially holds.

In the PPAR case, we have P = P₁ | P₂, Q = Q₁ | Q₂, and C = C₁∪∪C₂ for some P₁, P₂, Q₁, Q₂, C₁, C₂ such that Γ ⊢ P₁ ~ Q₁ → C₁ and Γ ⊢ P₂ ~ Q₂ → C₂. By the induction hypothesis, there exists (c₁, Γ₁) ∈ C₁ and (c₂, Γ₂) ∈ C₂ such that Γ′ ⊆ Γ₁ and Γ′ ⊆ Γ₂. By Lemma B.8, dom(Γ₁) and dom(Γ₂) only contain dom(Γ) = dom(Γ) and variables in bvars(P₁) ∪ bvars(Q₁) ∪ bnames(P₁) ∪ bnames(Q₁) and bvars(P₂) ∪ bvars(Q₂) ∪ bnames(P₂) ∪ bnames(Q₂) respectively. Since Γ₁(x) = Γ₂(x) = Γ′(x) for all x in dom(Γ), and since the sets bvars(P₁) ∪ bvars(Q₁) ∪ bnames(P₁) ∪ bnames(Q₁) and bvars(P₂) ∪ bvars(Q₂) ∪ bnames(P₂) ∪ bnames(Q₂) are disjoint by well formedness of the processes P₁ | P₂ and Q₁ | Q₂, Γ₁ and Γ₂ are compatible. Thus (c₁∪∪C₁, Γ₁ ∪ Γ₂) ∈ C₁ ∪∪ C₂ and the claim holds since Γ′ ⊆ Γ₁ ∪ Γ₂.

In the POR case, we have Γ = Γ ′′, x : T₁ ∨ T₂ for some x, Γ ′′, T₁, T₂ such that Γ ′′ ⊆ Γ ′′ ⊆ Γ ′. Hence by the induction hypothesis to Γ ⊢ P ~ Q → C ′, there exists (c′, Γ ′) ∈ C′ such that Γ ′ ⊆ Γ ′′. By definition of Uc, (c′′, Γ ′′) ∈ C, which proves the claim.

In the PlL case, there exist P’, P′′, Q’, M, N, M′, N′, C′, c such that P = if M = M′ then P′ else P′′, Q = if N = N′ then Q′ else Q′, C = C′∪∪c, Γ ⊢ M ~ M′ : N : LL → c, Γ ⊢ M ′ ~ N′ : LL → c, Γ ⊢ P’ ~ Q’ → C′, and Γ ⊢ P′′ ~ Q′ → C″. By applying the induction hypothesis to Γ ⊢ P′ ~ Q′ → C′, there exists (c′, Γ ′) ∈ C′ such that Γ ′ ⊆ Γ ′′. By definition of Uc, (c′′, Γ ′′) ∈ C, which proves the claim.

All remaining cases are similar. We write the proof for the PlL case. In this case, there exist P’, P′′, Q’, M, N, M′, N′, C′, C″, Γ ⊢ M ~ M′ : N : LL → c, ⊢ P’ ~ Q’ → C′, and Γ ⊢ P′′ ~ Q′ → C″. By applying the induction hypothesis to Γ ⊢ P’ ~ Q’ → C′, there exists (c′, Γ ′′) ∈ C′ such that Γ ′′ ⊆ Γ ′′.
If \((c, \Gamma') \in C\), we thus know that \((c, \Gamma') \in C'\) or \((c, \Gamma') \in C''\). We write the proof for the case where \((c, \Gamma') \in C'\), the other case is analogous.

By applying the induction hypothesis to \(\Gamma \vdash P' \sim Q' \rightarrow C'\), there exists \(\Gamma'' \in \text{branches}(\Gamma)\) such that \(\Gamma'' \subseteq \Gamma', \Gamma''\), which proves the claim.

\[\square\]

**Lemma B.14 (Refinement types).** For all \(\Gamma\), for all terms \(t, t'\), for all \(n, a, l, l', c\), if \(\Gamma \vdash t \sim t' : \left[ r^L_n : r^L_t, a \right] \rightarrow c\) then \(c = 0\) and

- \(t = m, t' = n, a = \infty\) and \(\Gamma(m) = T^L_m \) and \(\Gamma(n) = T^L_n\);
- \(t = m, t' = n, a = 1\), and \(\Gamma(m) = T^L_m \) or \(\Gamma(n) = T^L_n\), and \(\Gamma(n) = T^L_n\) or \(\Gamma(l) = T^L_n\);
- \(t = m, t' = n, a = 1\), and \(\Gamma(m) = T^L_m \) or \(\Gamma(n) = T^L_n\), and \(\Gamma(n) = T^L_n\) or \(\Gamma(l) = T^L_n\);
- \(t = m, t' = n, a = 1\), and \(\Gamma(m) = T^L_m \) or \(\Gamma(n) = T^L_n\), and \(\Gamma(n) = T^L_n\) or \(\Gamma(l) = T^L_n\).

In particular if \(t, t'\) are ground then only the first case can occur.

**Proof.** The proof of this property is immediate by induction on the typing derivation for the terms. Indeed, because of the form of the type, and by well-formedness of \(\Gamma\), the only rules which can lead to \(\Gamma \vdash t \sim t' : \left[ r^L_m : r^L_t, a \right] \rightarrow c\) are \(\text{TVAR}, \text{TLR}^1\), \(\text{TLR}^\omega\), \(\text{TLRVAR}\), and \(\text{TSub}\).

In the \(\text{TVAR}\), \(\text{TLR}^1\), \(\text{TLR}^\omega\) cases the claim directly follows from the premises of the rule.

In the \(\text{TSub}\) case, \(t\) and \(t'\) are necessarily variables, and their types in \(\Gamma\) are obtained directly by applying the induction hypothesis to the premises of the rule.

Finally in the \(\text{TSub}\) case, \(\Gamma \vdash t \sim t' : T \rightarrow c\) and \(T \triangleleft \left[ r^L_m : r^L_t, a \right]\). By Lemma B.1, \(T = \left[ r^L_m : r^L_t, a \right]\) and we conclude by the induction hypothesis.

\[\square\]

**Lemma B.15 (Encryption types).** For all environment \(\Gamma\), type \(T\), key \(k \in K\), messages \(M, N\), and set of constraints \(c\):

1. If \(\Gamma \vdash M \sim N : (T)_k \rightarrow c\) then
   - either there exist \(M', N', n\) such that \(M = \text{enc}(M', k), N = \text{enc}(N', k)\), and \(\Gamma \vdash M' \sim N' : T \rightarrow c\) with a shorter derivation (than the one for \(\Gamma \vdash M \sim N : (T)_k \rightarrow c\));
   - or \(M\) and \(N\) are variables.

2. If \(\Gamma \vdash M \sim N : (T)_k \rightarrow c\) then
   - either there exist \(M', N', n\) such that \(M = \text{aenc}(M', pk(k)), N = \text{aenc}(N', pk(k))\), and \(\Gamma \vdash M' \sim N' : T \rightarrow c\) with a shorter derivation (than the one for \(\Gamma \vdash M \sim N : (T)_k \rightarrow c\));
   - or \(M\) and \(N\) are variables.

3. If \(T \triangleleft \text{LL}\) and \(\Gamma \vdash \text{enc}(M, k) \sim N : T \rightarrow c\) then \(T = \text{LL}\).

4. If \(T \triangleleft \text{LL}\) and \(\Gamma \vdash \text{aenc}(M, pk(k)) \sim N : T \rightarrow c\) then \(T = \text{LL}\).

5. If \(\Gamma \vdash \text{enc}(M, k) \sim N : \text{LL} \rightarrow c\) then there exists \(N'\) such that \(N = \text{enc}(N', k)\), and
   - either there exist \(T'\) and \(c'\) such that \(\Gamma(k) = \text{key}^\text{ll}(T')\), \(c = \{\text{enc}(M, k) \sim N\} \cup c'\), and \(\Gamma \vdash M \sim N' : T' \rightarrow c'\);
   - or \(\Gamma \vdash T' \sim c'\) such that \(\Gamma(k) = \text{key}^\text{ll}(T')\) and \(\Gamma \vdash M \sim N' : \text{LL} \rightarrow c\).

6. If \(\Gamma \vdash \text{aenc}(M, pk(k)) \sim N : \text{LL} \rightarrow c\) then there exists \(N'\) such that \(N = \text{aenc}(N', pk(k))\), and
   - either there exist \(T'\) and \(c'\) such that \(\Gamma(k) = \text{key}^\text{ll}(T')\), \(c = \{\text{aenc}(M, pk(k)) \sim N\} \cup c'\), and \(\Gamma \vdash M \sim N' : T' \rightarrow c'\);
   - or \(k \in \text{dom}(\Gamma)\) and \(\Gamma \vdash M \sim N' : \text{LL} \rightarrow c\).

7. The symmetric properties to the previous four points, i.e. when the term on the right is an encryption, also hold.

**Proof.** We prove point 1 by induction on the derivation of \(\Gamma \vdash M \sim N : (T)_k \rightarrow c\). Because of the form of the type, and by well-formedness of \(\Gamma\), the only possibilities for the last rule applied are \(\text{TVAR}, \text{TEnc}, \text{TSub}\). The claim clearly holds in the \(\text{TVAR}\) and \(\text{TEnc}\) cases. In the \(\text{TSub}\) case, we have \(\Gamma \vdash M \sim N : T' \triangleleft (T)_k \rightarrow c\), and by Lemma B.1, there exists \(T'' \triangleleft T\) such that \(T' = (T'')_k\). Therefore, by applying the induction hypothesis to \(\Gamma \vdash M \sim N : T' \rightarrow c\)

- either \(M\) and \(N\) are either two variables, and the claim holds;
- or there exist \(M', N'\) such that \(M = \text{enc}(M', k), N = \text{enc}(N', k)\), and \(\Gamma \vdash M' \sim N' : T'' \rightarrow c\), with a derivation shorter than the one for \(\Gamma \vdash M \sim N : T' \rightarrow c\). Thus by subtyping (rule \(\text{TSub}\)), \(\Gamma \vdash M' \sim N' : T \rightarrow c\) with a shorter derivation that \(\Gamma \vdash M \sim N : (T)_k \rightarrow c\), which proves the property.

Point 2 has a similar proof to point 1.

We now prove point 3 by induction on the proof of \(\Gamma \vdash \text{enc}(M, k) \sim N : T \rightarrow c\). Because of the form of the terms, the last rule applied can only be \(\text{THIGH}, \text{TOr}, \text{TEnc}, \text{TEncH}, \text{TEncL}, \text{TAEC}, \text{TAECN}, \text{TSub}\).

The \(\text{THIGH}, \text{TOr}, \text{TEnc}\) cases are actually impossible by Lemma B.1, since \(T \triangleleft \text{LL}\). In the \(\text{TSub}\) case, we have \(\Gamma \vdash \text{enc}(M, k) \sim N : T' \rightarrow c\) for some \(T'\) such that \(T' \triangleleft T\). By transitivity of \(\triangleleft\), \(T' \triangleleft \text{LL}\), and the induction hypothesis proves the claim. In all other cases, \(T = \text{LL}\) and the claim holds.

Point 4 has a similar proof to point 3.

We prove point 5 by induction on the proof of \(\Gamma \vdash \text{enc}(M, k) \sim N : \text{LL} \rightarrow c\). Because of the form of the terms and of the type (i.e. \(\text{LL}\)) the last rule applied can only be \(\text{TEncH}, \text{TEncL}, \text{TAEC}, \text{TAECN}, \text{TSub}\).
The TRL’ case is impossible, since by Lemma B.14 it would imply that $\text{enc}(M, k)$ is either a variable or a nonce.

In the TSUB case, we have $\Gamma \vdash \text{enc}(M, k) \sim N : T' \rightarrow c$ for some $T'$ such that $T' \ll \LL$. By point 3, $T' = \LL$, and the premise of the rule thus gives a shorter derivation of $\Gamma \vdash \text{enc}(M, k) \sim N : \LL \rightarrow c$. The induction hypothesis applied to this shorter derivation proves the claim.

The $\text{TAECH}$ and $\text{TENC}$ cases are impossible, since the condition of the rule would then imply $\Gamma \vdash \text{enc}(M, k) \sim N : \{T\}_k \rightarrow c'$ for some $T, k, c'$, which is not possible by point 2.

Finally, in the $\text{TENP}$ and $\text{TENC}$ cases, the premises of the rule directly proves the claim.

Point 6 has a similar proof to point 5.

The symmetric properties, as described in point 7, have analogous proofs.

**Lemma B.16 (Signature Types).** For all environment $\Gamma$, type $T$, key $k \in \mathcal{K}$, messages $M, N$, and set of constraints $c$:

1. If $T \ll \LL$ and $\Gamma \vdash \text{sign}(M, k) \sim N : T \rightarrow c$ then $T = \LL$.
2. If $\Gamma \vdash \text{sign}(M, k) \sim N : \LL \rightarrow c$ then there exists $N'$ such that $N = \text{sign}(N', k),$ and
   - either there exist $T', c'$ and $c''$ such that $\Gamma(k) = \text{key}^{\text{all}}(T'), c = \{\text{sign}(M, k) \sim N\} \cup c' \cup c''$, $\Gamma \vdash M \sim N' : T' \rightarrow c'$, and $\Gamma \vdash M \sim N' : \LL \rightarrow c''$;
   - or there exists $T'$ such that $\Gamma(k) = \text{key}^{\text{all}}(T')$ and $\Gamma \vdash M \sim N' : \LL \rightarrow c'$.
3. The symmetric properties to the previous four points, i.e. when the term on the right is a signature, also hold.

**Proof.** We prove point 1 by induction on the proof of $\Gamma \vdash \text{sign}(M, k) \sim N : T \rightarrow c$. Because of the form of the terms, the last rule applied can only be $\text{THIG}$, $\text{TOB}$, $\text{TENP}$, $\text{TENC}$, $\text{TAECH}$, $\text{TAECH}$, $\text{TSGH}$, $\text{TSGNL}$, $\text{TLR}$, $\text{TLR'}$ or $\text{TSUB}$.

The $\text{THIG}$, $\text{TLR}$, $\text{TOB}$ cases are actually impossible by Lemma B.1, since $T \ll \LL$. In the $\text{TSUB}$ case, we have $\Gamma \vdash \text{sign}(M, k) \sim N : T' \rightarrow c$ for some $T'$ such that $T' \ll T$. By transitivity of $\ll,$ $T' \ll \LL$, and the induction hypothesis proves the claim. In all other cases, $T = \LL$ and the claim holds.

We prove point 2 by induction on the proof of $\Gamma \vdash \text{sign}(M, k) \sim N : \LL \rightarrow c$. Because of the form of the terms and of the type (i.e. $\LL$) the last rule applied can only be $\text{TENC}$, $\text{TENC}$, $\text{TAECH}$, $\text{TAECH}$, $\text{TSGH}$, $\text{TSGNL}$, $\text{TLR}$ or $\text{TSUB}$.

The $\text{TLR}$ case is impossible, since by Lemma B.14 it would imply that $\text{sign}(M, k)$ is either a variable or a nonce.

In the $\text{TSUB}$ case, we have $\Gamma \vdash \text{sign}(M, k) \sim N : T' \rightarrow c$ for some $T'$ such that $T' \ll \LL$. By point 3, $T' = \LL$, and the premise of the rule thus gives a shorter derivation of $\Gamma \vdash \text{sign}(M, k) \sim N : \LL \rightarrow c$. The induction hypothesis applied to this shorter derivation proves the claim.

The $\text{TENP}$, $\text{TENC}$, $\text{TAECH}$ and $\text{TAECH}$ cases are impossible, since the condition of the rule would then imply $\Gamma \vdash \text{sign}(M, k) \sim N : \{T\}_k \rightarrow c'$ (or $\{T\}_k$) for some $T, k, c'$, which is not possible by Lemma B.15.

Finally, in the $\text{TSGH}$ and $\text{TSGNL}$ cases, the premises of the rule directly proves the claim.

The symmetric properties, as described in point 3, have analogous proofs.

**Lemma B.17 (Pair Types).** For all environment $\Gamma$, for all $M, N, T, c$:

1. For all $T_1, T_2$, if $\Gamma \vdash M \sim N : T_1 \ast T_2 \rightarrow c$ then
   - either there exist $M_1, M_2, N_1, N_2, c_1, c_2$ such that $M = \langle M_1, M_2 \rangle$, $N = \langle N_1, N_2 \rangle$, $c = c_1 \cup c_2$, and $\Gamma \vdash M_1 \sim N_1 : T_1 \rightarrow c_1$ and $\Gamma \vdash M_2 \sim N_2 : T_2 \rightarrow c_2$;
   - or $M$ and $N$ are variables.
2. For all $M_1, M_2$, if $T \ll \LL$ and $\Gamma \vdash \langle M_1, M_2 \rangle \sim N : T \rightarrow c$ then either $T = \LL$ or there exists $T_1, T_2$ such that $T = T_1 \ast T_2$.
3. For all $M_1, M_2$, if $\Gamma \vdash \langle M_1, M_2 \rangle \sim N : \LL \rightarrow c$ then there exist $N_1, N_2, c_1, c_2$ such that $c = c_1 \cup c_2$, $N = \langle N_1, N_2 \rangle$, $\Gamma \vdash M_1 \sim N_1 : \LL \rightarrow c_1$ and $\Gamma \vdash M_2 \sim N_2 : \LL \rightarrow c_2$.
4. The symmetric properties to the previous two points (i.e. when the term on the right is a pair) also hold.

**Proof.** Let us prove point 1 by induction on the typing derivation $\Gamma \vdash M \sim N : T_1 \ast T_2 \rightarrow c$. Because of the form of the type, and by well-formedness of $\Gamma$, the only possibilities for the last rule applied are $\text{TVA}, \text{TPAIR}$, and $\text{TSUB}$.

The claim clearly holds in the $\text{TVA}$ and $\text{TPAIR}$ cases.

In the $\text{TSUB}$ case, $\Gamma \vdash M \sim N : T' \rightarrow c$ for some $T' \ll T_1 \ast T_2$, and by Lemma B.1, $T' = T'_1 \ast T'_2$ for some $T'_1, T'_2$ such that $T'_1 \ll T_1$ and $T'_2 \ll T_2$. Therefore, by applying the induction hypothesis to $\Gamma \vdash M \sim N : T'_1 \ast T'_2 \rightarrow c$, $M$ and $N$ are either two variables, and the claim holds; or, for two pairs, i.e. there exist $M_1, M_2, N_1, N_2, c_1, c_2$ such that $M = \langle M_1, M_2 \rangle$, $N = \langle N_1, N_2 \rangle$, $c = c_1 \cup c_2$, and for $i \in \{1, 2\}$, $\Gamma \vdash M_i \sim N_i : T'_i \rightarrow c_i$. Hence, by subtyping, $\Gamma \vdash M_1 \sim N_1 : T_1 \rightarrow c_1$, and the claim holds.

We now prove point 2 by induction on the proof of $\Gamma \vdash \langle M_1, M_2 \rangle \sim N : T \rightarrow c$. Because of the form of the terms, the last rule applied can only be $\text{THIG}, \text{TOB}, \text{TPAIR}, \text{TENC}, \text{TENC}, \text{TAECH}, \text{TAECH}, \text{TLR}, \text{TLR}$ or $\text{TSUB}$.

The $\text{THIG}$, $\text{TLR}$, and $\text{TOB}$ cases are actually impossible by Lemma B.1, since $T \ll \LL$.

The $\text{TLR}$ and case is also impossible, since by Lemma B.14 it would imply that $\langle M_1, M_2 \rangle$ is either a variable or a nonce.
The \text{TENCH}, \text{TENC}, \text{TAEHC}, \text{TAENCL} cases are impossible, since the condition of the rule would then imply \( \Gamma \vdash (M_1, M_2) \sim N : (T)_k \rightarrow c' \) (or \( (T)_k \)) for some \( T, k, c' \), which is not possible by Lemma B.15.

In the TPAIR case, the claim clearly holds.

Finally, in the TSUB case, we have \( \Gamma \vdash (M_1, M_2) \sim N : T' \rightarrow T. By transitivity of \( <; T' \sim LL, \) and we may apply the induction hypothesis to \( \Gamma \vdash (M_1, M_2) \sim N : T' \rightarrow c. Hence either \( T' = LL \) or \( T' = T'_1 * T'_2 \) for some \( T'_1, T'_2 \). By Lemma B.1, this implies in the first case that \( T = LL \) and in the second case that \( T = LL \) or \( T = \text{a pair type} (T \neq Hl \) and \( T \neq HH \) in both cases, since we already know that \( T < LL \).

We prove point 3 as a consequence of the first two points, by induction on the derivation of \( \Gamma \vdash (M_1, M_2) \sim N : LL \rightarrow c \). The last rule in this derivation can only be TENCH, TENC, TAENCH, TAENCL, TLR', TLR' or TSBUI by the form of the types and terms, but similarly to the previous point TENCH, TENC, TAENCH, TAENCL, TLR' and TLR' are actually not possible.

Hence the last rule of the derivation is TSBUI. We have \( \Gamma \vdash (M_1, M_2) \sim N : T \rightarrow c \) for some \( T \) such that \( T < LL \). By point 2, either \( T = LL \) or there exist \( T_1, T_2 \) such that \( T = T_1 * T_2 \). If \( T = LL \), we have a shorter proof of \( \Gamma \vdash (M_1, M_2) \sim N : LL \rightarrow c \) and we conclude by the induction hypothesis. Otherwise, since \( T < LL \), by Lemma B.1, \( T_1 < LL \) and \( T_2 < LL \). Moreover by the first property, there exist \( N_1, N_2, c_1, c_2 \) such that \( N = (N_1, N_2), c = c_1 \cup c_2, \Gamma \vdash M_1 \sim N_1 : T_1 \rightarrow c_1, \) and \( \Gamma \vdash M_2 \sim N_2 : T_2 \rightarrow c_2. \)

Thus by subtyping, \( \Gamma \vdash M_1 \sim N_1 : LL \rightarrow c_1 \) and \( \Gamma \vdash M_2 \sim N_2 : LL \rightarrow c_2 \), which proves the claim.

\[ \square \]

Lemma B.18 (Type for keys, nonces and constants): For all environment \( \Gamma \), for all messages \( N, M \), for all key \( k \) in \( K \), for all nonce or constant \( n \) in \( N \cup C \), for all l, the following properties hold:

1. For all \( T \), if \( \Gamma \vdash N : key^l(T) \rightarrow c \), then \( c = 0 \) and either \( M \) are in \( K \) and \( \Gamma(M) = key^l(T) \); or \( M \) and \( N \) are variables.
2. If \( l \in \{ LL, HH \}, \) and \( \Gamma \vdash k \sim N : I \rightarrow c \), then \( N = k, c = 0 \), and there exists \( T \) such that \( \Gamma(k) = key^l(T) \).
3. If \( \Gamma \vdash pk(k) \sim N : LL \rightarrow c \), then \( k \in \text{dom}(T) \) and \( N = pk(k) \).
4. If \( \Gamma \vdash vk(k) \sim N : LL \rightarrow c \), then \( k \in \text{dom}(T) \) and \( N = vk(k) \).
5. If \( \Gamma \vdash n \sim N : HH \rightarrow n \), then \( n \in \text{BN}, c = 0 \) and either \( \Gamma(n) = h_H^{1,1} \) or \( h_H^{1,\infty} \).
6. If \( \Gamma \vdash n \sim N : LL \rightarrow c \), then \( N = n, c = 0 \), and either there exists \( a \in \{ 1, \infty \} \) such that \( \Gamma(a) = h_L^{1,1, a} \), or \( n \in F N \cup C \).
7. The symmetric properties to the previous five points (i.e. with \( k \) (resp. \( pk(k), vk(k), n \) on the right) also hold.

Proof. Point 1 is easily proved by induction on the derivation of \( \Gamma \vdash M \sim N : key^l(T) \rightarrow c. \) Indeed, by the form of the type the last rule can only be TKey, TVar, or TSBUI. In the TKey and TVar cases the claim clearly holds. In the TSBUI case, by Lemma B.1, \( key^l(T) \) is its only subtype, thus there exists a shorter derivation of \( \Gamma \vdash M \sim N : key^l(T) \rightarrow c \), an the claim holds by the induction hypothesis.

We prove point 2 by induction on the derivation of \( \Gamma \vdash k \sim N : I \rightarrow c. \) Because of the form of the terms and type, and by well-formedness of \( \Gamma \), the last rule applied can only be TENCH, TENC, TAENCH, TAENCL, TLR', TLR' or TSBUI.

The TENCH, TENC, TAENCH, TAENCL cases are impossible since they would imply that \( \Gamma \vdash k \sim N : (T')_{k'} \rightarrow c' \) (or \( (T')_{k'} \)) for some \( T', k', c' \), which is impossible by Lemma B.15.

The TLR' and TLR' cases are impossible. Indeed in these cases, we have \( \Gamma \vdash k \sim N : [T_m^{a} : T_n^{a}] \rightarrow \emptyset \) for some \( m, n \). Lemma B.14 then implies that \( m = k \) and \( n \neq N \), which is contradictory.

Finally, in the TSBUI case, we have \( \Gamma \vdash k \sim N : I \rightarrow c \) for some \( T \) such that \( T < I \). By Lemma B.1, this implies that \( T \) is either a pair type, a key type, or \( I \). Just as in the previous point, the first case is impossible and the last one is trivial. The case where \( T = key^l(T') \) (for some \( T' \)) is also impossible by point 1, since \( pk(k) \) is not in \( K \cup X \).

Finally in the TPubKey case, the claim clearly holds.

Point 4 has a similar proof to point 3.

The remaining properties have similar proofs to point 2. For point 5, i.e., if \( \Gamma \vdash n \sim t : HH \rightarrow c \), only the TNonce, TSBUI, and TLR' cases are possible. The claim clearly holds in the TNonce case.
In the TLR' case, we have \( \Gamma \vdash n \sim t : [\tau^\text{HL}_m ; \tau^\text{HL}_p] \rightarrow \emptyset \) for some \( m, p \). Lemma B.14 then implies that \( m = n \), and \( p = t \), and \( \Gamma(n) = \tau^\text{HL}_m \), and \( \Gamma(p) = \tau^\text{HL}_p \), which proves the claim.

In the TSUB case, \( \Gamma \vdash n \sim t : T \rightarrow c \) for some \( T \in \text{H} \), thus by Lemma B.1 \( T \) is either a pair type (impossible by Lemma B.17), a key type (impossible by point 1), or \( \text{H} \) (and we conclude by the induction hypothesis).

For point 6, similarly, only the TNONC, TCSTFN, TSB, TLR' cases are possible. The TSB case is proved in the same way as for the third property. The TLR' case is proved similarly to the previous point. Finally the claim clearly holds in the TNONC and TCSTFN cases.

The symmetric properties, as described in point 7, have analogous proofs.

**Lemma B.19 (Type LL implies same head symbol).** For all \( \Gamma, M, N, c \), if \( \Gamma \vdash M \sim N : LL \rightarrow c \) then either \( M \) and \( N \) have the same head symbol and use the same key if this symbol is enc, aenc or \( \text{sign}(.,.) \) or \( M, N \) both are variables.

Proof. We prove a slightly more general property: for all \( T \in \text{LL} \), if \( \Gamma \vdash M \sim N : T \rightarrow c \) then either \( M \) and \( N \) have the same head symbol and use the same key if this symbol is enc, aenc or \( \text{sign}(.,.) \) or \( M, N \) both are variables.

This is proved by induction on the typing derivation. Many of the cases for the last rule applied are immediate, since they directly state that the two terms have the same head symbol (with the same key) or are variables. This covers rules TNONC, TCSTFN, TPSUB, TKEY, TKVAR, TPAIR, TENC, TAENC, TISONC, TISONL, THASH, THASHL, TLRVAR. Among the remaining cases, some are also immediate thanks to the assumption that \( T \in \text{LL} \), as they contradict it (which we prove using Lemma B.1). This covers rules TNONC, THIGH, TORD, TLR¹, TLR⁰, TLR'. Moreover in the case of rule TSB the claim follows directly from the application of the induction hypothesis to the premise of the rule.

Only the cases of rules TENC, TENC, TAENC, TAENC, and TLR' remain. In the TENC case, \( \Gamma \vdash M \sim N : (T)_k \rightarrow c \) for some \( T, k \), and therefore by Lemma B.15 \( M \) and \( N \) are either two variables or some terms encrypted with \( k \), and in both cases the claim holds. The TENC, TAENC, TAENC cases are similar, using Lemma B.15. In the TLR' case, \( \Gamma \vdash M \sim N : [\tau^\text{HL}_m ; \tau^\text{HL}_p] \rightarrow c \) for some \( M \). Thus by Lemma B.14, either \( M \) and \( N \) are two variables or \( M = N = m \), and in any case the claim holds, which concludes this proof.

**Lemma B.20 (Application of destructors).** For all \( \Gamma, y, \) such that \( y \notin \text{dom}(\Gamma) \), for all \( d, T, T', c \), for all ground messages \( M, N \), if \( \Gamma, y : \vdash d(y) : T' \) and \( \Gamma \vdash M \sim N : T \rightarrow c \), then:

1. We have:
   \[
   (d(M)) \downarrow \vDash \iff (d(N)) \downarrow \vDash
   \]

2. And if \((d(M)) \downarrow \vDash \) then there exists \( c' \subseteq c \) such that
   \[
   \Gamma \vdash (d(M)) \downarrow \sim (d(N)) \downarrow : T' \rightarrow c'
   \]

Proof. We distinguish four cases for \( d \).

- **\( d = \text{dec}(\cdot, k) \).** We know that \( \Gamma, y : \vdash d(y) : T' \), which can be proved using either rule DDEC, rule DDECL, or rule DDECT. In the first two cases, \( T = \text{LL} \), and in the last case \( T = (T')_k \).
  - Let us prove 1) by contraposition. Assume \( (d(M)) \not\downarrow \vDash \). Hence, \( M = \text{enc}(M', k) \) for some \( M' \). Lemma B.19 in the DDEC and DDECL cases (where \( \Gamma \vdash M \sim N : \text{LL} \rightarrow c \), and Lemma B.15 in the DDECT case (where \( \Gamma \vdash M \sim N : (T')_k \rightarrow c \)), guarantee that there exists \( N' \) such that \( N = \text{enc}(N', k) \). Therefore \( (d(N)) \not\downarrow \vDash \) which proves the first direction of 1). The other direction is analogous.
  - Moreover, still assuming \( (d(M)) \not\downarrow \vDash \), and keeping the notations from the previous point, we have \( d(M) \not\vdash M' \) and \( d(N) \not\vdash N' \).

- **The destructor typing rule applied to prove \( \Gamma, y : \vdash d(y) : T' \) can be DDEC, DDECL, or DDECT.**
  - **In the DDECT case.** We have \( T = (T')_k \) and therefore \( \Gamma \vdash M \sim N : (T')_k \rightarrow c \). Lemma B.15 (point 1) then guarantees that \( \Gamma \vdash M' \sim N' : T' \rightarrow c \), which proves point 2).
  - **In the DDEC case.** We have \( T = \text{LL} \) and \( \Gamma(k) = \text{key}^\text{HL}(T') \). Thus, we have \( \Gamma \vdash \text{enc}(M', k) \sim \text{enc}(N', k) : \text{LL} \rightarrow c \), and by Lemma B.15 (point 5), we know that there exists \( c' \subseteq c \) such that \( c = c' \cup \{ M \sim N \} \) and \( \Gamma \vdash M' \sim N' : T' \rightarrow c' \), which proves point 2).
  - **In the DDECT case.** We have \( T = T' = \text{LL} \) and there exists \( T'' \) such that \( \Gamma(k) = \text{key}^\text{HL}(T'') \). Thus, we have \( \Gamma \vdash \text{enc}(M', k) \sim \text{enc}(N', k) : \text{LL} \rightarrow c \), and by Lemma B.15 (point 5), we know that \( \Gamma \vdash M' \sim N' : \text{LL} \rightarrow c \), which proves point 2).

In all cases, point 2) holds, which concludes this case.

- **\( d = \text{add}(\cdot, k) \).** We know that \( \Gamma, y : \vdash d(y) : T' \), which can be proved using either rule DADEC, rule DADECL, or rule DADECT. In the first two cases, \( T = \text{LL} \), and in the last case \( T = (T')_k \).
  - Let us prove 1) by contraposition. Assume \( (d(M)) \not\downarrow \vDash \). Hence, \( M = \text{aenc}(M', pk(k)) \) for some \( M' \). Lemma B.19 in the DADEC and DADECL cases (where \( \Gamma \vdash M \sim N : \text{LL} \rightarrow c \), and Lemma B.15 in the DADECT case (where \( \Gamma \vdash M \sim N : (T')_k \rightarrow c \)), guarantee that there exists \( N' \) such that \( N = \text{aenc}(N', pk(k)) \). Therefore \( (d(N)) \not\downarrow \vDash \) which proves the first direction of 1). The other direction is analogous.
  - Moreover, still assuming \( (d(M)) \not\downarrow \vDash \), and keeping the notations from the previous point, we have \( d(M) \not\vdash M' \) and \( d(N) \not\vdash N' \).

The destructor typing rule applied to prove \( \Gamma, y : \vdash d(y) : T' \) can be DADECT, DADECL, or DADECT.
• In the DADEcT case we have \( T = (T')_k \) and thus \( \Gamma \vdash M \sim N : (T')_k \rightarrow c \). Lemma B.15 then guarantees that \( \Gamma \vdash M' \sim N' : T' \rightarrow c \) which proves the claim.

• In the DADEc case we have \( T = \mathbb{L} \), and there exists \( T'' \) such that \( T' = T'' \lor \mathbb{L} \) and \( \Gamma(k) = \text{key}^{\text{Hk}}(T'') \). Thus, we have \( \Gamma \vdash \text{aecc}(M', \text{pk}(k)) \sim \text{aecc}(N', \text{pk}(k)) : \mathbb{L} \rightarrow c \). Hence by Lemma B.15 (point 5), we know that either there exists \( c' \) such that \( c = c' \cup \{ M \sim N \} \) and \( \Gamma \vdash M' \sim N' : T'' \rightarrow c' \), or \( \Gamma \vdash M' \sim N' : \mathbb{L} \rightarrow c \). By rule TOr, we then have \( \Gamma \vdash M' \sim N' : T'' \lor \mathbb{L} \rightarrow c' \) (resp. \( c \)), which proves point 2).

• In the DADEcL case we have \( T = T' = \mathbb{L} \) and there exists \( T'' \) such that \( \Gamma(k) = \text{key}^{\text{LL}}(T'') \). Thus, we have \( \Gamma \vdash \text{aecc}(M', \text{pk}(k)) \sim \text{aecc}(N', \text{pk}(k)) : \mathbb{L} \rightarrow c \), and by Lemma B.15 (point 5), we know that \( \Gamma \vdash M' \sim N' : \mathbb{L} \rightarrow c \), which proves point 2).

In all cases, point 2) holds, which concludes this case.

\[ d = \text{checksign}(\cdot, \text{vk}(k)) \] We know that \( \Gamma, y : T \vdash d(y) : T' \), which can be proved using either rule DCHECK or rule DCHECKL. In both cases, \( T = \mathbb{L} \).

- Let us prove 1) by contraposition. Assume \( d(M) \not\models \bot \). Hence, \( M = \text{sign}(M', k) \) for some \( M' \). Lemma B.19 (applied to \( \Gamma \vdash M \sim N : \mathbb{L} \rightarrow c \)) guarantees that there exists \( N' \) such that \( N = \text{sign}(N', k) \). Therefore \( d(N) \not\models \bot \) (which proves the first direction of 1). The other direction is analogous.

- Moreover, still assuming \( d(M) \not\models \bot \), and keeping the notations from the previous point, we have \( d(M) \models M' \) and \( d(N) \models N' \). The destructor typing rule applied to prove \( \Gamma, y : \mathbb{L} \vdash d(y) : T' \) can be DCHECK or DCHECKL.

- In the DADEcT case we have \( T = \mathbb{L} \), and \( \Gamma(k) = \text{key}^{\text{Hk}}(T') \). Thus we have \( \Gamma \vdash \text{sign}(M', k) \sim \text{sign}(N', k) : \mathbb{L} \rightarrow c \). Hence by Lemma B.16 (point 2), we know that there exist \( c', c'' \) such that \( c = c' \cup c'' \cup \{ M \sim N \} \), \( \Gamma \vdash M' \sim N' : T'' \rightarrow c' \), and \( \Gamma \vdash M' \sim N' : \mathbb{L} \rightarrow c'' \). This proves point 2).

- In the DADEcL case we have \( T = \mathbb{L} \), and there exists \( T'' \) such that \( \Gamma(k) = \text{key}^{\text{LL}}(T'') \). Hence by Lemma B.16 (point 2), we know that \( \Gamma \vdash M' \sim N' : \mathbb{L} \rightarrow c \). This proves point 2).

In all cases, point 2) holds, which concludes this case.

\[ d = \pi_1 \] We know that \( \Gamma, y : T \vdash d(y) : T' \), which can be proved using either rule DFST or DFSTL. In the first case, \( T = T_1 \star T_2 \) is a pair type, and in the second case \( T = \mathbb{L} \).

- We prove 1) by contraposition. Assume \( d(M) \not\models \bot \). Hence, \( M = \langle M_1, M_2 \rangle \) for some \( M_1, M_2 \). Thus, by applying Lemma B.17 to \( \Gamma \vdash M \sim N : T \rightarrow c \), in any case we know that there exist \( N_1, N_2 \) such that \( N = \langle N_1, N_2 \rangle \). Therefore \( d(N) \not\models \bot \) (which proves the first direction of 1). The other direction is analogous.

- Moreover, still assuming \( d(M) \not\models \bot \), and keeping the notations from the previous point, we have \( d(M) \models M_1 \) and \( d(N) \models N_1 \). In addition, we know that \( \Gamma, y : T \vdash d(y) : T' \), which can be proved using either rule DFST or DFSTL. Lemma B.17, which we applied in the previous point, also implies that there exist \( c_1, c_2 \), such that \( c = c_1 \cup c_2 \) and for \( i \in \{1, 2\} \), \( \Gamma \vdash M_i \sim N_i : T_i \rightarrow c_i \) (in the DFST case) or \( \Gamma \vdash M_i \sim N_i : \mathbb{L} \rightarrow c_i \) (in the DFSTL case).

We distinguish two cases for the rule applied to prove \( \Gamma, y : T \vdash d(y) : T' \).

- DFST: Then \( T = T_1 \star T_2 \) and \( T' = T_1 \), and \( \Gamma \vdash M_1 \sim N_1 : T_1 \rightarrow c_1(\subseteq c) \) proves 2).

- DFSTL: Then \( T = T' = \mathbb{L} \), and \( \Gamma \vdash M_1 \sim N_1 : \mathbb{L} \rightarrow c_1(\subseteq c) \) proves 2).

In both cases, point 2) holds, which concludes this case.

\[ d = \pi_2 \] This case is similar to the previous one.

\[ \square \]

**Lemma B.21 (LL type is preserved by attacker terms).** For all \( \Gamma \), for all frames \( \psi = \text{new} \ E_T, \phi \) and \( \psi' = \text{new} \ E_T, \phi' \) with \( \Gamma \vdash \phi \sim \phi' \) : \( \mathbb{L} \rightarrow c \), for all attacker term \( R \) such that \( \text{vars}(R) \subseteq \text{dom}(\phi) \), either there exists \( c' \leq c \) such that

\[ \Gamma \vdash R \phi \downarrow \sim R \phi' \downarrow \mathbb{L} \rightarrow c' \]

or

\[ R \phi \downarrow = R \phi' \downarrow = \bot \]

**Proof.** Let us recall that \( E_T \) denotes the set of names in \( \Gamma \).

We show this property by induction over the attacker term \( R \).

**Induction Hypothesis:** the statement holds for all subterms of \( R \). There are several cases for \( R \). The base cases are the cases where \( R \) is a variable, a name in \( F_N \) or a constant in \( C \).

1. \( R = x \) Since \( \text{vars}(R) \subseteq \text{dom}(\phi) \), we have \( x \in \text{dom}(\phi) = \text{dom}(\phi') \), hence \( R \phi \downarrow = \phi(x) \) and \( R \phi' \downarrow = \phi'(x) \). Since \( \Gamma \vdash \phi \sim \phi' \) : \( \mathbb{L} \rightarrow c \), we have \( \Gamma \vdash \phi(x) \sim \phi'(x) : \mathbb{L} \rightarrow c_1 \) for some \( c_1 \subseteq c \), and the claim holds.

2. \( R = a \) with \( a \in C \cup F_N \). Then \( R \phi \downarrow = R \phi' \downarrow = a \) and by rule TCSTFN, we have \( \Gamma \vdash a = a : \mathbb{L} \rightarrow \bot \). Hence the claim holds.

3. \( R = \text{pk}(k) \) We apply the induction hypothesis to \( K \) and distinguish three cases.

   a. If \( K \phi \downarrow = \bot \) then \( K \phi' \downarrow = \bot \), hence \( R \phi \downarrow = R \phi' \downarrow = \bot \).

   b. If \( K \phi \downarrow \not= \bot \) and is not a key then \( K \phi' \downarrow \not= \bot \) (by IH), and by IH we have \( \Gamma \vdash K \phi \downarrow \sim K \phi' \downarrow : \mathbb{L} \rightarrow c' \) for some \( c' \subseteq c \). Then by Lemma B.19, \( K \phi' \downarrow \) is not a key either. Hence \( R \phi \downarrow = R \phi' \downarrow = \bot \).
R = \textsc{vkh(K)} We apply the induction hypothesis to K and distinguish three cases.

(a) If $K \downarrow \perp$ then $K' \downarrow \perp$, hence $R \downarrow \perp \downarrow \perp$.

(b) If $K \downarrow \not\perp$ and is not a key then $K' \downarrow \not\perp$ (by IH), and by IH we have $\Gamma \vdash K \downarrow \sim K' \downarrow \vdash c'$ for some $c' \subseteq c$. Then, by Lemma B.19, $K' \downarrow \perp$ is not a key either. Hence $R \downarrow \perp \downarrow \perp$.

(c) If $K \downarrow \perp$ is a key, then by IH there exists $c' \subseteq c$ such that $\Gamma \vdash K \downarrow \sim K' \downarrow \vdash \lll \rightarrow c'$. Hence by Lemma B.18 $K' \downarrow \perp$, and $\Gamma(K' \downarrow \perp) = \text{key}^\lll(T)$ for some $T$. Therefore by rule T\textsc{vkh}\textsc{key}, $\Gamma \vdash R \downarrow \sim R' \downarrow : \lll \rightarrow \emptyset$ and the claim holds.

R = (R1, R2) where R1 and R2 are also attacker terms. We then apply the induction hypotheses to the same frames and R1, R2. We distinguish two cases:

(a) $R_1 \downarrow \perp \lor R_2 \downarrow \perp$. In this case we also have $R_1 \downarrow \perp \lor R_2 \downarrow \perp$. Hence $R \downarrow \perp \downarrow \perp$.

(b) $R_1 \downarrow \not\perp \land R_2 \downarrow \not\perp$. In this case, by the induction hypothesis, we also have $R_1 \downarrow \not\perp \land R_2 \downarrow \not\perp$, and we also know that there exist $c_1 \subseteq c$ and $c_2 \subseteq c$ such that $\Gamma \vdash R_1 \downarrow \sim R_1' \downarrow : \lll \rightarrow c_1$ and $\Gamma \vdash R_2 \downarrow \sim R_2' \downarrow : \lll \rightarrow c_2$.

Thus, by the rule T\textsc{pair} followed by T\textsc{sub}, $\Gamma \vdash R \downarrow \sim R' \downarrow : \lll \rightarrow c_1 \cup c_2$. Since $c_1 \cup c_2 \subseteq c$, this proves the case.

R = enc(S, K) We apply the induction hypothesis to K and distinguish three cases.

(a) If $K \downarrow \perp$ then $K' \downarrow \perp$, hence $R \downarrow \perp \downarrow \perp$.

(b) If $K \downarrow \not\perp$ and is not $\text{pk}(k)$ for some $k \in \mathcal{K}$ then $K' \downarrow \not\perp$ (by IH), and by IH there exists $c' \subseteq c$ such that $\Gamma \vdash K \downarrow \sim K' \downarrow : \lll \rightarrow c'$. Then, by Lemma B.19, $K' \downarrow \not\perp$ is not a public key either. Hence $R \downarrow \perp \downarrow \perp$.

(c) If $K \downarrow \perp$ is a key, then by IH there exists $c' \subseteq c$ such that $\Gamma \vdash K \downarrow \sim K' \downarrow : \lll \rightarrow c'$. Hence by Lemma B.18 $K' \downarrow \perp$, and $\Gamma(K' \downarrow \perp) = \text{key}^\perp(T)$ for some $T$. We then apply the IH to $S$, and either $S \downarrow \not\perp \downarrow \not\perp$ or $S \downarrow \not\perp \downarrow \perp$; in which case $R \downarrow \perp \downarrow \perp$ or $R \downarrow \perp \downarrow \perp$; or there exists $c'' \subseteq c$ such that $\Gamma \vdash S \downarrow \sim S' \downarrow : \lll \rightarrow c''$. Since $R \downarrow = \text{enc}(S \downarrow, K \downarrow)$, and similarly for $\psi'$, by rule T\textsc{enc}, we have $\Gamma \vdash R \downarrow \sim R' \downarrow : (\lll \cap K \downarrow) \rightarrow c''$, and then by rule T\textsc{enc}L, $\Gamma \vdash R \downarrow \sim R' \downarrow : \lll \rightarrow c''$.

R = aenc(S, K) We apply the induction hypothesis to K and distinguish three cases.

(a) If $K \downarrow \perp$ then $K \downarrow \perp$, hence $R \downarrow \perp \downarrow \perp$.

(b) If $K \downarrow \not\perp$ and is not $\text{pk}(k)$ for some $k \in \mathcal{K}$ then $K' \downarrow \not\perp$ (by IH), and by IH there exists $c' \subseteq c$ such that $\Gamma \vdash K \downarrow \sim K' \downarrow : \lll \rightarrow c'$. Then, by Lemma B.19, $K' \downarrow \not\perp$ is not a public key either. Hence $R \downarrow \perp \downarrow \perp$.

(c) If $K \downarrow \perp$ is a key, then by IH there exists $c' \subseteq c$ such that $\Gamma \vdash K \downarrow \sim K' \downarrow : \lll \rightarrow c'$. Hence by Lemma B.18, $K' \downarrow \not\perp$ is a key either. Hence $R \downarrow \perp \downarrow \perp$.

R = s\textsc{sign}(S, K) We apply the induction hypothesis to K and distinguish three cases.

(a) If $K \downarrow \perp$ then $K \downarrow \perp$, hence $R \downarrow \perp \downarrow \perp$.

(b) If $K \downarrow \not\perp$ and is not a key $k \in \mathcal{K}$ then $K' \downarrow \not\perp$ (by IH), and by IH we have $\Gamma \vdash K \downarrow \sim K' \downarrow : \lll \rightarrow c'$ for some $c' \subseteq c$. Then, by Lemma B.19, $K' \downarrow \not\perp$ is not a key either. Hence $R \downarrow \perp \downarrow \perp$.

(c) If $K \downarrow = k$ for some $k \in \mathcal{K}$ then by IH there exists $c' \subseteq c$ such that $\Gamma \vdash K \downarrow \sim K' \downarrow : \lll \rightarrow c'$. Hence by Lemma B.18, $K' \downarrow = k$ and $\Gamma(k) = \text{key}^\lll(T)$ for some $T$. We then apply the IH to $S$, and either $S \downarrow \not\perp \downarrow \not\perp$ or $S \downarrow \not\perp \downarrow \perp$; in which case $R \downarrow \not\perp \downarrow \perp$ or $R \downarrow \not\perp \downarrow \perp$; or there exists $c'' \subseteq c$ such that $\Gamma \vdash S \downarrow \sim S' \downarrow : \lll \rightarrow c''$. Therefore by rule T\textsc{aenc}, $\Gamma \vdash R \downarrow \sim R' \downarrow : (\lll \cap K \downarrow) \rightarrow c''$, and by rule T\textsc{aenc}L we have $\Gamma \vdash R \downarrow \sim R' \downarrow : \lll \rightarrow c''$.

R = b\textsc{sign}(S, K) We apply the induction hypothesis to S. We distinguish two cases.

(a) $S \downarrow \perp$. In this case we also have $S \downarrow \perp$. Hence $R \downarrow \perp \downarrow \perp$.

(b) $S \downarrow \not\perp$. In this case, by the induction hypothesis, we also have $S \downarrow \not\perp$. And we also know that there exists $c' \subseteq c$ such that $\Gamma \vdash S \downarrow \sim S' \downarrow : \lll \rightarrow c'$. Thus, by rule T\textsc{hash}\textsc{L}, $\Gamma \vdash R \downarrow \sim R' \downarrow : \lll \rightarrow c'$, which proves this case.

R = \textsc{pi}(S) We apply the induction hypothesis to S and distinguish three cases.

(a) $S \downarrow \perp$. Then $S \downarrow \perp$ (by IH), hence $R \downarrow \perp \downarrow \perp$.

(b) $S \downarrow \not\perp$. In this case, by the induction hypothesis, we also have $S \downarrow \not\perp$. And we know that there exists $c' \subseteq c$ such that $\Gamma \vdash S \downarrow \sim S' \downarrow : \lll \rightarrow c'$. Thus, by rule T\textsc{hash}\textsc{L}, $\Gamma \vdash R \downarrow \sim R' \downarrow : \lll \rightarrow c'$, which proves this case.

R = \textsc{dec}(S, K) We apply the induction hypothesis to K and, similarly to the case 6, we distinguish several cases.

(a) If $K \downarrow \not\perp$ or is not a key then, as in case 6, $R \downarrow \not\perp \downarrow \perp$.

(b) If $K \downarrow \perp$. Then similarly to case 6 we can show that $K \downarrow = K' \downarrow$, and $\Gamma(K' \downarrow) = \text{key}^\lll(T)$ for some $T$. We then apply the IH to S, which creates two cases. Either $S \downarrow \not\perp \downarrow \perp$ or or there exists $c' \subseteq c$ such that $\Gamma \vdash S \downarrow \sim S' \downarrow : \lll \rightarrow c'$. In the first case, the claim holds, since $R \downarrow \not\perp \downarrow \perp$. In the second case, by Lemma B.19, we know that $S \downarrow$ is an encryption by
We apply the induction hypothesis to $K$ and, similarly to the case 6, we distinguish several cases.

(a) If $Kφ \vdash \bot$ or is not a verification key then, as in case 6, $Rφ \vdash Rφ′ \vdash \bot$.

(b) If $Kφ$ is a key, then similarly to case 6 we can show that $Kφ \vdash Kφ′ \vdash \bot$, and $Γ(Kφ′) = Γ(Kφ′′)$ is keyLt(T) for some $T$. Then we apply the IH to $S$, which creates two cases. Either $Sφ \vdash Sφ′ \vdash \bot$, or there exists $c′ \subseteq c$ such that $Γ \vdash Sφ \vdash Sφ′ \vdash L \rightarrow c′$.

In the first case, the claim holds, since $Rφ \vdash Rφ′ \vdash \bot$. In the second case, by Lemma B.19, we know that $Sφ$ is an asymmetric encryption by $pk(Kφ′)$ if and only if $Sφ′$ also is an encryption by this key. Consequently, if $Sφ$ is not an encryption by $pk(Kφ′)$, then it is the same for $Sφ′$ and $Rφ \vdash Rφ′ \vdash \bot$. Otherwise, $Sφ \vdash enc(t, Kφ′)$ and $Sφ′ \vdash enc(t′, Kφ′)$ for some $t, t′$. In that case, by IH, we have $Γ \vdash enc(t, Kφ′) \sim enc(t′, Kφ′)$ for some $t$. Therefore, by Lemma B.15 (point 5), $Γ \vdash t \sim t′ \vdash L \rightarrow c′$. Hence the claim holds in this case.

(13) $R = addc(S, K)$ We apply the induction hypothesis to $K$ and, similarly to the case 6, we distinguish several cases.

(a) If $Kφ \vdash \bot$ or is not a verification key then, as in case 7, we can show that $Rφ \vdash Rφ′ \vdash \bot$.

(b) If $Kφ$ is a verification key $vk(k)$ for some $k ∈ K$, then similarly to case 7 we can show that $Kφ \vdash Kφ′ \vdash \bot$, and $k ∈ dom(Γ')$. We then apply the IH to $S$, which creates two cases. Either $Sφ \vdash Sφ′ \vdash \bot$, or there exists $c′ \subseteq c$ such that $Γ \vdash Sφ \vdash Sφ′ \vdash L \rightarrow c′$.

In the first case, the claim holds, since $Rφ \vdash Rφ′ \vdash \bot$. In the second case, by Lemma B.19, we know that $Sφ$ is a signature by $k ∈ Kφ′$. If only if $Sφ′$ also is a signature by this key. Consequently, if $Sφ$ is not signed by $k$, then neither is $Sφ′$, and $Rφ \vdash Rφ′ \vdash \bot$. Otherwise, $Sφ \vdash sign(t, k)$ and $Sφ′ \vdash sign(t′, k)$ for some $t, t′$. Thus by IH we have $Γ \vdash sign(t, k) \sim sign(t′, k)$ for $L \rightarrow c′$. Therefore, by Lemma B.16 (point 2), we know that there exists $c′ \subseteq c$ such that $Γ \vdash t \sim t′ \vdash L \rightarrow c′$. That is to say $Γ \vdash Rφ \vdash Rφ′ \vdash L \rightarrow c′$. Hence the claim holds in this case.

(14) $R = checksign(S, K)$ We apply the induction hypothesis to $K$ and, similarly to the case 7, we distinguish several cases.

(a) If $Kφ \vdash \bot$ or is not a verification key then, as in case 7, we can show that $Rφ \vdash Rφ′ \vdash \bot$.

(b) If $Kφ$ is a verification key $vk(k)$ for some $k ∈ K$, then similarly to case 7 we can show that $Kφ \vdash Kφ′ \vdash \bot$, and $k ∈ dom(Γ')$. We then apply the IH to $S$, which creates two cases. Either $Sφ \vdash Sφ′ \vdash \bot$, or there exists $c′ \subseteq c$ such that $Γ \vdash Sφ \vdash Sφ′ \vdash L \rightarrow c′$.

In the first case, the claim holds, since $Rφ \vdash Rφ′ \vdash \bot$. In the second case, by Lemma B.19, we know that $Sφ$ is a signature by $k ∈ Kφ′$ if and only if $Sφ′$ also is a signature by this key. Consequently, if $Sφ$ is not signed by $k$, then neither is $Sφ′$, and $Rφ \vdash Rφ′ \vdash \bot$. Otherwise, $Sφ \vdash sign(t, k)$ and $Sφ′ \vdash sign(t′, k)$ for some $t, t′$. Thus by IH we have $Γ \vdash sign(t, k) \sim sign(t′, k)$ for $L \rightarrow c′$. Therefore, by Lemma B.16 (point 2), we know that there exists $c′ \subseteq c$ such that $Γ \vdash t \sim t′ \vdash L \rightarrow c′$. That is to say $Γ \vdash Rφ \vdash Rφ′ \vdash L \rightarrow c′$. Hence the claim holds in this case.

LEMMA B.22 (Substitution preserves typing). For all $Γ, Γ′$, such that $Γ ⇒ Γ′ ⇒ σ$, (we do not require that $Γ$ and $Γ′$ are well-formed), for all $M, N, T, c, c_0$, for all ground substitutions $σ, σ'$, if

- $Γ, Γ'$ only contains names and, have disjoint domains,
- $Γ''$ only contains variables,
- for all $x ∈ dom(Γ''), Γ''(x)$ is not of the form $[t_m^{l_1}; t_n^{r_1}]$,
- $(Γ ∪ Γ'')_N, K, χ, σ ∼ σ' : (Γ ∪ Γ'')_X ⇒ c_0$,
- and $Γ ∪ Γ'' = M ∼ N : T ⇒ c$

then there exists $c′ \subseteq [c]_{σ, σ'}$ such that $Γ ∪ Γ'' ⇒ Mσ ∼ Nσ' : T ⇒ c'$.

In particular, if we have $Γ = Γ''_X, Γ′ = ∅$, and $Γ'' ⇒ (Γ''_N, K, χ)$, then the last three conditions trivially hold.

Proof. Note that $Γ''_N, K = Γ''_N, K = ∅, Γ''_X = Γ, Γ''_X = Γ''_N, K = Γ''''$ and $Γ''''_X = ∅$. This proof is done by induction on the typing derivation for the terms. The claim clearly holds in the TNONCE, TNONCEL, TCSFNP, TPUK, TVKEY, TKEY, THASH, THIGH, TLR3, TLR9 since their conditions do not use $Γ(x)$ (for any variable $x$) or another type judgement, and they still apply to the messages $Mσ$ and $Nσ'$.

It follows directly from the induction hypothesis in all other cases except the TVAR and TLRVAR cases, which are the base cases.

In the TVAR case, the claim also holds, since $M = N = x$ for some variable $x ∈ dom(Γ) ∪ dom(Γ')$. If $x ∈ dom(Γ')$, then $xσ = xσ' = x$, and $T = Γ''(x)$. Thus, by rule TVAR, $Γ'' \cup Γ''' ∗ xσ ∼ xσ' : Γ''(x) \rightarrow ∅$ and the claim holds. If $x ∈ dom(Γ)$, and, by hypothesis the substitutions are well-typed, there exists $c_0 ⊆ c_{σ, σ'}$ such that $(Γ ∪ Γ'')_N, K, σ \sim σ'(x) : Γ(x) ⇒ c_0$. Thus, since $(Γ ∪ Γ'')_{N, K, χ} = Γ''''$, and by applying Lemma B.10 to $Γ''''$, $Γ'' \cup Γ''' \sim σ(x) : Γ(x) ⇒ c_0$ and the claim holds.

Finally, in the TLRVAR case, there exist two variables $x, y$, and types $t_m^{l_1}; t_n^{r_1}; t_m^{l_1}; t_n^{r_1}; t_m^{l_1}$, such that $M = x, N = y, c = ∅, Γ \cup Γ'' \sim x ∼ y : [t_m^{l_1}; t_n^{r_1}] ⇒ ∅, (Γ \cup Γ'''' \sim y ∼ y : [t_m^{l_1}; t_n^{r_1}] ⇒ ∅, and $T = [t_m^{l_1}; t_n^{r_1}]$.

By Lemma B.14, this implies that $(Γ \cup Γ'' \sim xσ(x) = [t_m^{l_1}; t_n^{r_1}]$ and $(Γ \cup Γ'' \sim yσ(y) = [t_m^{l_1}; t_n^{r_1}] \sim 1^n m′′ ′′ t′′′ t′'. Therefore, by hypothesis $Γ''$ does not contain such types, and $Γ'''$ does not contain variables $x ∈ dom(Γ)$ and $y ∈ dom(Γ)$.

Moreover, by the induction hypothesis, there exist $c', c'' ⊆ [c]_{σ, σ'}$ such that $Γ \cup Γ'' \sim xσ ∼ xσ' : [t_m^{l_1}; t_n^{r_1}] ⇒ c$, and $Γ \cup Γ'' \sim yσ ∼ yσ' : [t_m^{l_1}; t_n^{r_1}] ⇒ c''$. That is to say, since $x, y ∈ dom(Γ)$, $Γ \cup Γ'' \sim xσ ∼ xσ' : [t_m^{l_1}; t_n^{r_1}] ⇒ c'$, and $Γ \cup Γ'' \sim yσ ∼ yσ' : [t_m^{l_1}; t_n^{r_1}] \sim 1^n m′′ ′′ t′′′ t′'. Hence, by Lemma B.14, and since $σ, σ'$ are ground, we have $σ(x) = m, σ'(x) = n, σ(y) = m′$, and $σ'(y) = n′$, and $Γ''''(m) = m′$ and $Γ''''(n) = n′$.

Thus, by rule TLR3, $Γ \vdash σ(x) ∼ σ'(x) : [t_m^{l_1}; t_n^{r_1}] ⇒ ∅$, which proves the claim. □
Lemma B.23 (Types LL and HH are Disjoint). For all Γ, for all ground terms M, M′, N, N′, for all sets of constraints c, c′, if Γ ⊢ M ≈ N : LL → c and Γ ⊢ M′ ≈ N′ : HH → c′ then M ≈ M′ and N ≈ N′.

Proof. First, it is easy to see by induction on the type derivation that for all ground terms M, N, for all c, if Γ ⊢ M ≈ N : LL → c then either
- M is a nonce m ∈ N such that Γ(m) = ℓ_m^{HH} for some a ∈ \{∞, 1\};
- or M is a key and Γ(M) = key^{HH}(T) for some T;
- or Γ ⊢ M ⊕ N : HH + T → c′ for some T, c′;
- or Γ ⊢ M + N : T + HH → c′ for some T, c′.

Indeed, (as Γ is well-formed) the only possible cases are TNonce, TSub, and TLR. In the TNonce case the claim clearly holds. In the TSub case we use Lemma B.1 followed by Lemma B.18. In the TLR case we apply Lemma B.14 and the claim directly follows.

Let us now show that for all M, N, N′ ground, for all c, c′, Γ ⊢ M ≈ N : LL → c and Γ ⊢ M + N : HH → c′ cannot both hold. (This corresponds, with the notations of the statement of the lemma, to proving by contradiction that M ≈ M′. The proof that N ≈ N′ is analogous.)

We show this property by induction on the size of \( t_1 \).

Since Γ ⊢ M + N : LL → c′, by the property stated in the beginning of this proof, we can distinguish four cases.
- If M is a nonce and Γ(M) = \( ℓ_m^{HH} \); then this contradicts Lemma B.18. Indeed, this lemma (point 5) implies that M ∈ BN, but also (by point 6), since Γ ⊢ M + N : LL → c, that either Γ(M) = ℓ_m^{LL} for some a ∈ \{1,∞\}, or M ∈ F N ∪ C.
- If M is a key and Γ(M) = key^{HH}(T) for some T; then by Lemma B.18, since Γ ⊢ M + N : LL → c, there exists T′ such that Γ(M) = key^{LL}(T′). This contradicts Γ(M) = key^{HH}(T).
- If Γ ⊢ M + N′ : HH + T → c′ for some T, c′; then by Lemma B.17, since M, N′ are ground, there exist M_1, M_2, N_1′, N_2′, c_1′ such that M = (M_1, M_2), N′ = (N_1′, N_2′), and Γ ⊢ M_1 + N_1′ : HH → c_1′. Moreover, since Γ ⊢ M + N : LL → c, also by Lemma B.17, there exist N_1, N_2, c_1 such that N = (N_1, N_2) and Γ ⊢ M_1 + N_1 : LL → c_1. However, by the induction hypothesis, Γ ⊢ M_1 + N_1 : HH → c_1′ and Γ ⊢ M_1 + N_1 : LL → c_1 is impossible.
- If Γ ⊢ M + N′ : T + HH → c′ for some T, c′; this case is similar to the previous one.

□

Lemma B.24 (Low Terms are Recipes on their Constraints). For all ground messages M, N, for all T ≦ : LL, for all Γ, c, if Γ ⊢ M ≈ N : T → c then there exists an attacker recipe R without destructors such that M = \( R(\phi_x(c) ∪ \phi^T) \) and N = \( R(\phi_x(c) ∪ \phi^T) \).

Proof. We prove this lemma by induction on the typing derivation of Γ ⊢ M ≈ N : T → c. We distinguish several cases for the last rule in this derivation.
- TNonce, Tenc, TAnchor, THigh, TOR, TLR, TLR∞, TLR‘, TLRVar: these cases are not possible, since the type they give to terms is never a subtype of LL by Lemma B.1.
- TVar: this case is not possible since M, N are ground.
- TSub: this case is directly proved by applying the induction hypothesis to the judgement 0 ⊢ M + N : T′ → c where T′ ≦ : T ≦ : LL, which appears in the condition of this rule, and has a shorter derivation.
- TLR: in this case, Γ ⊢ M + N : [ℓ_1^{LL} : ℓ_2^{LL}] → c′ for some nonce n, some a ∈ \{1,∞\}, some c′, and c = 0. By Lemma B.14, this implies that M = N = n, and Γ(n) = ℓ_n^{LL}. Thus, by definition, there exists x such that \( \phi^T(x) = n \) and the claim holds with R = x.
- TNoncel: in this case M = N = n for some n ∈ N such that Γ(n) = ℓ_n^{LL} for some a ∈ \{1,∞\}. Hence, by definition, there exists x such that \( \phi^T(x) = n \) and the claim holds with R = x.
- TSub: then M = N = a ∈ C ∪ F N, and the claim holds with R = a.
- Tkey: then M = N = k ∈ K and there exists T′ such that Γ(k) = key^{LL}(T′). By definition, there exists x such that \( \phi^T(x) = k \) and the claim holds with R = x.
- TPKc: then M = N = pk(k) (resp. vk(k)) for some k ∈ dom(Γ). By definition, there exists x such that \( \phi^T(x) = pk(k) \) (resp. \( \phi^T(x) \)) and the claim holds with R = x.
- TPair, THash: these cases are similar. We detail the TPair case. In that case, T = \( T_1 + T_2 \) for some \( T_1, T_2 \). By Lemma B.1, \( T_1, T_2 \) are subtypes of LL. In addition, there exist M_1, M_2, N_1, N_2, c_1, c_2 such that Γ ⊢ M_1 + N_1 : T_1 → c_1 for \( i \in \{1,2\} \). By applying the induction hypothesis to these two judgements (which have shorter proofs), we obtain R_1, R_2 such that for all i, \( M_i = R_i(\phi_x(c_i) ∪ \phi^T) \) and N_i = R_i(\phi_x(c_i) ∪ \phi^T). Therefore, the claim holds with R = \( R_1, R_2 \).
- TAnchor, TAnchorH, THash, TSigH: these four cases are similar. In each case, by the form of the typing rule, we have c = \( M ≈ N \cup c′ \) for some c′. Therefore by definition of \( \phi_x(c) \), \( \phi_x(c) \), there exists x such that \( \phi_x(c)(x) = M \) and \( \phi_x(c)(N) = N \). The claim holds with R = x.
• **TENCL, TAENCL:** these two cases are similar, we write the proof for the TENCL case. The form of this rule application is:

\[
\begin{align*}
\Pi \\
\Gamma \vdash M \sim N : (\text{LL})_k & \rightarrow c \\
\Gamma(k) = \text{key}^{\text{LL}}(T')
\end{align*}
\]

By Lemma B.15, there exist \(M', N'\) such that \(M = \text{enc}(M', k), N = \text{enc}(N', k)\), and \(\Gamma \vdash M' \sim N' : \text{LL} \rightarrow c\) with a proof shorter than \(\Pi\). Thus by applying the induction hypothesis to this judgement, there exists \(R\) such that \(M' = R(\phi_c(e) \cup \phi^{\text{LL}}_k)\) and \(N' = R(\phi_r(c) \cup \phi^{\text{LL}}_k)\). Moreover, since \(\Gamma(k) = \text{key}^{\text{LL}}(T')\), by definition, there exists \(x\) such that \(\phi^{\text{LL}}_k(x) = k\) (in the asymmetric case, the messages are encrypted with a public key \(pk(k)\), which also appears in this frame). Therefore, the claim holds with the recipe \(\text{enc}(R, x)\).

• **TSIGN:** the form of this rule application is:

\[
\begin{align*}
\Pi \\
\Gamma \vdash M' \sim N' : \text{LL} \rightarrow c \\
\Gamma(k) = \text{key}^{\text{LL}}(T')
\end{align*}
\]

with \(M = \text{sign}(M', k), N = \text{sign}(N', k)\). Thus by applying the induction hypothesis to \(\Gamma \vdash M' \sim N' : \text{LL} \rightarrow c\), there exists \(R\) such that \(M' = R(\phi_c(e) \cup \phi^{\text{LL}}_k)\) and \(N' = R(\phi_r(c) \cup \phi^{\text{LL}}_k)\). Moreover, since \(\Gamma(k) = \text{key}^{\text{LL}}(T')\), by definition, there exists \(x\) such that \(\phi^{\text{LL}}_k(x) = k\). Therefore, the claim holds with the recipe \(\text{sign}(R, x)\).

\[\square\]

**Lemma B.25 (Low frames with consistent constraints are statically equivalent).** For all ground \(\phi, \phi'\), for all \(c, \Gamma\), if

• \(\Gamma \vdash \phi \sim \phi' : \text{LL} \rightarrow c\)

• and \(c\) is consistent in \(\Gamma_{N, K}\).

then new \(E_{\Gamma}, \phi\) and new \(E_{\Gamma}, \phi'\) are statically equivalent.

**Proof.** We can first notice that since \(\phi\) and \(\phi'\) are ground, so is \(c\) (this is easy to see by examining the typing rules for terms). Let \(R, R'\) be two attackerc cases, such that \(\text{vars}(R) \cup \text{vars}(R') \subseteq \text{dom}(\phi)(= \text{dom}(\phi'))\).

For all \(x \in \text{dom}(\phi)(= \text{dom}(\phi'))\), by assumption, there exists \(c_x \subseteq c\) such that \(\Gamma \vdash \phi(x) \sim \phi'(x) : \text{LL} \rightarrow c_x\). By Lemma B.24, there exists a recipe \(R_x\) such that \(\phi(x) = R_x(\phi_c(x) \cup \phi^{\text{LL}}_k)\) and \(\phi'(x) = R_x(\phi_r(c) \cup \phi^{\text{LL}}_k)\).

Since \(c_x \subseteq c\), we also have \(\phi(x) = R_x(\phi_c(x) \cup \phi^{\text{LL}}_k)\) and \(\phi'(x) = R_x(\phi_r(c) \cup \phi^{\text{LL}}_k)\).

Let \(\overline{R}\) and \(\overline{R'}\) be the recipes obtained by replacing every occurrence of \(x\) with \(R_x\) in respectively \(R\) and \(R'\), for all variable \(x \in \text{dom}(\phi)(= \text{dom}(\phi'))\).

We then have \(\overline{R} \phi = \overline{R}(\phi_c(x) \cup \phi^{\text{LL}}_k)\) and \(\overline{R'} \phi = \overline{R}(\phi_r(c) \cup \phi^{\text{LL}}_k)\); and similarly \(\overline{R} \phi' = \overline{R}(\phi_r(c) \cup \phi^{\text{LL}}_k)\) and \(\overline{R'} \phi' = \overline{R}(\phi_r(c) \cup \phi^{\text{LL}}_k)\).

Since \(c\) is ground, and consistent in \(\Gamma_{N, K}\), by definition of consistency, the frames new \(E_{\Gamma}, \phi_c(x) \cup \phi^{\text{LL}}_k\) and new \(E_{\Gamma}, \phi_r(c) \cup \phi^{\text{LL}}_k\) are statically equivalent. Hence, by definition of static equivalence,

\[
\overline{R}(\phi_c(x) \cup \phi^{\text{LL}}_k) = \overline{R}(\phi_r(c) \cup \phi^{\text{LL}}_k) \iff \overline{R}(\phi_c(x) \cup \phi^{\text{LL}}_k) = \overline{R}(\phi_r(c) \cup \phi^{\text{LL}}_k)
\]

i.e.

\[
\overline{R} \phi = \overline{R'} \phi \iff \overline{R} \phi' = \overline{R'} \phi'
\]

Therefore, new \(E_{\Gamma}, \phi\) and new \(E_{\Gamma}, \phi'\) are statically equivalent. \[\square\]

**Lemma B.26 (Invariant).** For all \(\Gamma, \phi, \phi', \phi_Q, \sigma_p, \sigma_Q, \epsilon_c, \epsilon_c', \Gamma'\), for all multisets of processes \(P, P', Q\), where the processes in \(P, P', Q\) are noted \(\{P_1\}, \{P'_1\}, \{Q_1\}\); for all constraint sets \(\{C_1\}\); if

• \(|P| = |Q|

• \(\text{dom}(\phi_p) = \text{dom}(\phi_Q)

• \(\forall i, \text{there is a derivation } \Pi_i \text{ of } \Gamma \vdash P_i \sim Q_i \rightarrow C_i,

• \Gamma \vdash \phi_p \sim \phi_Q : \text{LL} \rightarrow c_0\)

• for all \(i \neq j\), the sets of bound variables in \(P_i\) and \(P_j\) (resp. \(Q_i\) and \(Q_j\)) are disjoint, and similarly for the names

• \(\sigma_p, \sigma_Q\) are ground, and \(\Gamma_{N, K} = \sigma_p \sim \sigma_Q : \Gamma' \rightarrow c_0\),

• \(\bigcup \{C_i, \Gamma'_{C_i} \cup \epsilon_c\} \sigma_p, \sigma_Q \cup \epsilon_c, \Gamma'_{C_\epsilon}\) is consistent,

• \((E_{\Gamma}, P, \phi_p, \sigma_p) \rightarrow (E', P', \phi'_p, \sigma'_p)\),

then there exist a word \(w\), a multiset \(Q' = \{Q'_1\}\), constraint sets \(\{C'_1\}\), a frame \(\phi'_Q\), a substitution \(\sigma'_Q\), an environment \(\Gamma'\), constraints \(\epsilon'_c\) and \(\epsilon'_c'\) such that:

• \(w = \tau\)
• \(|P\'| = |Q'|\)
• for all \(i \neq j\), the sets of bound variables in \(P'_i\) and \(P'_j\) (resp. \(Q'_i\) and \(Q'_j\)) are disjoint, and similarly for the bound names.
• \(\Gamma' \vdash \phi'_p \sim \phi'_Q : \text{LL} \rightarrow c'_\phi\)
• \(\mathcal{E}' = \mathcal{E}_\Gamma\)
• \((\mathcal{E}_\Gamma, Q, \phi_Q, \sigma_Q) \xrightarrow{\mathcal{W}} (\mathcal{E}'_\Gamma, Q', \phi'_Q, \sigma'_Q)\),
• \(\forall i, \Gamma' \vdash P'_i \sim Q'_i \rightarrow C_i'\),
• \(\sigma'_p, \sigma'_Q\) are ground and \(\Gamma'_{N, K} + \sigma'_p \sim \sigma'_Q : \Gamma'_{X} \rightarrow c'_{\sigma}\).
• \(\text{dom}(\phi'_p) = \text{dom}(\phi'_Q)\),
• \(\left(\bigcup_{C_i'} C_i'\right) \cup_{\forall C_i} \sigma_P \cup_{\forall C_i} \sigma_Q \text{ is consistent.}\)

**Proof.** First, we show that it is sufficient to prove this lemma in the case where \(\Gamma\) does not contain any union types. Indeed, assume we know the property holds in that case. Let us show that the lemma then also holds in the other case, i.e. if \(\Gamma\) contains union types. By hypothesis, \(\sigma_p, \sigma_Q\) are ground, and \(\Gamma_{N, K} + \sigma_p \sim \sigma_Q : \Gamma_{X} \rightarrow c_{\sigma}\). Hence we know by Lemma B.4 that there exists a branch \(\Gamma'' \in \text{branches}(\Gamma)\) (thus \(\Gamma''\) does not contain union types), such that \((\Gamma'')_{N, K} + \sigma_p \sim \sigma_Q : (\Gamma'')_{X} \rightarrow c_{\sigma}\).

Moreover, by Lemma B.7, \(\forall i, \Gamma'' \vdash P_i \sim Q_i \rightarrow C_i'\subseteq C_i\); and by Lemma B.5, \(\Gamma'' \vdash \phi_p \sim \phi_Q : \text{LL} \rightarrow c_{\phi}\). In addition by Lemma B.11, \(\left(\bigcup_{C_i'} C_i'\right) \cup_{\forall C_i} \sigma_P \cup_{\forall C_i} \sigma_Q\) is a subset of \(\left(\bigcup_{C_i} C_i\right) \cup_{\forall C_i} \sigma_P \cup_{\forall C_i} \sigma_Q\) and is therefore consistent. Thus, if the lemma holds when the environment does not contain union types, it can be applied to the same processes, frames, substitutions and to \(\Gamma''\), which directly concludes the proof.

Therefore, we may assume that \(\Gamma\) does not contain any union types.

Note that the assumption on the disjointness of the sets of bound variables (and names) in the processes implies, using Lemma B.8, that since \(\left(\bigcup_{C_i} C_i\right) \cup_{\forall C_i} \sigma_P \cup_{\forall C_i} \sigma_Q\) is disjoint, and similarly for the bound names.

By hypothesis, \((\mathcal{E}_\Gamma, P, \phi_P, \sigma_P)\) reduces to \((\mathcal{E}_\Gamma', P', \phi'_P, \sigma'_P)\). We know from the form of the reduction rules that exactly one process \(P_i \in P\) is reduced, while the others are unchanged. By the assumptions, there is a corresponding process \(Q_i \in Q\) and a derivation \(\Pi_i\) of \(\Gamma \vdash P_i \sim Q_i \rightarrow C_i\).

We continue the proof by a case disjunction on the last rule of \(\Pi_i\). Let us first consider the cases of the rules \(\text{PZero, PPPar, PNew, and POR}\).

- **\(\text{PZero}\):** then \(P_i = Q_i = \emptyset\). Hence, the reduction rule applied to \(P\) is Zero, and \(P' = P \setminus \{P_i\}\), \(\mathcal{E}'_P = \mathcal{E}_P, \phi'_P = \phi_P\), and \(\sigma'_P = \sigma_P\). The same reduction can be performed in \(Q\):

\[
(\mathcal{E}_Q, Q, \phi_Q, \sigma_Q) \xrightarrow{\mathcal{W}} (\mathcal{E}_Q, Q \setminus \{Q_i\}, \phi_Q, \sigma_Q)
\]

Since the other processes, the frames, environments and substitutions do not change in this reduction, all the claims clearly hold in this case (with \(c'_\phi = c_P, c'_\sigma = c_{\text{sub}}\)). In particular, the consistency of the constraints follow from the consistency hypothesis. Indeed,

\[
\left(\bigcup_{\Gamma \neq i} C_i\right) \cup_{\bigcup_{\forall C_i}} \sigma_P \cup_{\bigcup_{\forall C_i}} \sigma_Q = \left(\bigcup_{\Gamma \neq i} C_i\right) \cup_{\bigcup_{\forall C_i}} \sigma_P \cup_{\bigcup_{\forall C_i}} \sigma_Q
\]

since \(\Gamma\) is already contained in the environments appearing in each \(C_i\) (by Lemma B.12). Thus

\[
\left(\bigcup_{\Gamma \neq i} C_i\right) \cup_{\bigcup_{\forall C_i}} \sigma_P \cup_{\bigcup_{\forall C_i}} \sigma_Q = \left(\bigcup_{\Gamma \neq i} C_i\right) \cup_{\bigcup_{\forall C_i}} \sigma_P \cup_{\bigcup_{\forall C_i}} \sigma_Q
\]

- **\(\text{PPPar}\):** then \(P_i = P^1_i \upharpoonright P^2_i, Q_i = Q^1_i \upharpoonright Q^2_i\). Hence, the reduction rule applied to \(P\) is Par:

\[
(\mathcal{E}_\Gamma, P, \phi_P, \sigma_P) \xrightarrow{\mathcal{W}} (\mathcal{E}_\Gamma, P \setminus \{P_i\} \cup \{P^1_i, P^2_i\}, \phi_P, \sigma_P).
\]

We choose \(\Gamma' = \Gamma\).

In addition

\[
\begin{align*}
\Pi^1 & = \frac{\Gamma \vdash P^1_i \sim Q^1_i \rightarrow C^1_i}{\Gamma \vdash P_i \sim Q_i \rightarrow C_i} \\
\Pi^2 & = \frac{\Gamma \vdash P^2_i \sim Q^2_i \rightarrow C^2_i}{\Gamma \vdash P_i \sim Q_i \rightarrow C_i = C^1_i \cup C^2_i}
\end{align*}
\]

The same reduction rule can be applied to \(Q\):

\[
(\mathcal{E}_\Gamma, Q, \phi_Q, \sigma_Q) \xrightarrow{\mathcal{W}} (\mathcal{E}_\Gamma, Q \setminus \{Q_i\} \cup \{Q^1_i, Q^2_i\}, \phi_Q, \sigma_Q)
\]
In this case again, the claims on the substitutions and frames hold since they do not change in the reduction. Moreover, the processes in $\mathcal{P}'$ and $\mathcal{Q}'$ are still pairwise typably equivalent. Indeed, all the processes from $\mathcal{P}$ and $\mathcal{Q}$ are unchanged, except for $P_i$ and $Q_i$ which are reduced to $P'_i$ and $Q'_i$, and those are typably equivalent using $\Pi'$ and $\Pi^2$.

Finally the constraint set is still consistent, since:

$$\left[ \left( \bigcup_{i,j} C'_i \cup C'_j \right) \sigma_p, \sigma_Q, \bigcup \nu_c \right]_{\sigma_p, \sigma_Q} = \left[ \left( \bigcup_{i,j} C_i \cup C'_i \cup C'_j \right) \sigma_p, \sigma_Q, \bigcup \nu_c \right]_{\sigma_p, \sigma_Q} = \left[ \left( \bigcup_{i,j} C_i \right) \cup \bigcup \nu_c \right]_{\sigma_p, \sigma_Q}$$

- $\mathbf{PNew}$: then $P_i = \text{new } n: \tau^{L,a}_n \cdot P'_i$ and $Q_i = \text{new } n: \tau^{L,a}_n \cdot Q'_i$. $P_i$ is reduced to $P'_i$ by rule $\text{New}$:

$$(\mathcal{E}_i, \mathcal{P}, \phi_p, \sigma_P) \xrightarrow{\tau} (\mathcal{E}_i \cup \{n\}, \mathcal{P} \setminus \{P_i\} \cup \{P'_i\}, \phi_p, \sigma_P).$$

In addition

$$\Pi'_i = \frac{\Gamma, n: \tau^{L,a}_n \cdot P'_i \sim Q'_i \rightarrow C_i}{\Gamma \vdash P_i \sim Q_i \rightarrow C_i}.$$  

We choose $\Gamma' = \Gamma, n: \tau^{L,a}_n$, and we have $\mathcal{E}_{\Gamma'} = \mathcal{E}_i \cup \{n\}$.

The same reduction rule can be applied to $Q'$:

$$(\mathcal{E}_i, \mathcal{Q}, \phi_Q, \sigma_Q) \xrightarrow{\tau} (\mathcal{E}_i \cup \{n\}, \mathcal{Q} \setminus \{Q_i\} \cup \{Q'_i\}, \phi_Q, \sigma_Q).$$

The claim clearly holds: the processes are still pairwise typable (using $\Pi'_i$ in the case of $P'_i$ and $Q'_i$, and $\Pi_j$ for $j \neq i$ as these processes are unchanged by the reduction), and all the frames, substitutions, and constraints are unchanged, and since $\sigma$, $\sigma'$ are well-typed in $\Gamma'$ if and only if they are well-typed in $\Gamma$.

- $\mathbf{POut}$: this case is not possible, since we have already eliminated the case where $\Gamma$ contains union types.

In all the other cases for the last rule in $\Pi_i$, we know that the head symbol of $P_i$ is not $\mid, \emptyset$ or new.

Hence, the form of the reduction rules implies that $P_i \in \mathcal{P}$ is reduced to exactly one process $P'_i \in \mathcal{P}'$, while the other processes in $\mathcal{P}$ do not change (i.e. $P'_j = P_j$ for $j \neq i$), and $\mathcal{E}' = \mathcal{E}_i$. If we show in each case that the same reduction rule that is applied to $P_i$ can be applied to reduce $Q$ to a multiset $Q'$ by reducing process $Q_i$ into $Q'_i$, we will also have $Q'_i = Q_j$ for $j \neq i$. Therefore the claim on the cardinality of the processes multisets will hold.

Since $P_i, Q_i$ can be typed and the head symbol of $P_i$ is not new, it is clear by examining the typing rules that the head symbol of $Q_i$ is not new either. Hence, we will choose a $\Gamma'$ containing the same nonces as $\Gamma$, and we will have $\mathcal{E}_{\Gamma'} = \mathcal{E}_i$.

The proofs for these cases follow the same structure:

- The typing rule gives us information on the form of $P_i$ and $Q_i$.
- The form of $P_i$ gives us information on which reduction rule was applied to $\mathcal{P}$.
- The form of $Q_i$ is the same as $P_i$. Hence (additional conditions may need to be checked depending on the rule) $Q_i$ can be reduced to some process $Q'_i$ by applying the same reduction rule that was applied to $P_i$ (except in the $\text{POutLR}$ case).
- Thus $Q$ can be reduced too, with the same actions as $\mathcal{P}$. We then check the additional conditions on the typing of the processes, frames and substitutions, and the consistency condition.

First, let us consider the $\text{POut}$ case.

- $\mathbf{POut}$: then $P_i = \text{out}(M), P'_i$ and reduces to $P'_i$ via the $\text{Out}$ rule, and $Q_i = \text{out}(N), Q'_i$ for some $N$ and $Q'_i$. In addition

$$\Pi'_i = \frac{\Gamma \vdash P'_i \sim Q'_i \rightarrow C_i}{\Gamma \vdash P_i \sim Q_i \rightarrow C_i}.$$  

We have $\mathcal{E}' = \mathcal{E}_i$, $\sigma'_P = \sigma_P$, $\phi'_P = \phi_P \cup \{M/\alpha_{\text{ax}}\}$, and $\alpha = \text{new } \alpha_{\text{ax}}, \text{out}(\alpha_{\text{ax}})$. The same reduction rule $\text{Out}$ can be applied to reduce the process $Q_i$ into $Q'_i$, hence the claim on the reduction of $\mathcal{Q}$ holds. We choose $\Gamma' = \Gamma$. We have $\mathcal{E}_{\Gamma'} = \mathcal{E}_i$, $\sigma'_Q = \sigma_Q$, and $\phi'_Q = \phi_Q \cup \{N/\alpha_{\text{ax}}\}$. We also choose $c'_P = c_{\phi} \cup c$ and $c'_Q = c_{\sigma}$. The substitutions $\sigma_p$, $\sigma_Q$ are not extended by the reduction, and the typing environment does not change, which trivially proves the claim regarding the substitutions.

Moreover, since only $M$ and $N$ are added to the frames in the reduction, $\Pi'$ suffices to prove the claim that $\Gamma \vdash \phi'_P \sim \phi'_Q : \text{LL} \rightarrow c'_P$. Since all processes other that $P_i$ and $Q_i$ are unchanged by the reduction (and since the typing environment is also unchanged), $\Pi$ suffices to proves the claim that $\forall j, \Gamma' \vdash P'_j \sim Q'_j \rightarrow C'_j$ (with $C'_j = C_j$ for $j \neq i$).
Thus, in this case, it only remains to be proved that \[ \left( \bigcup_{j \neq i} C_j \right) \cup \mathcal{C}_i' \subseteq \sigma_Q \cup \mathcal{C}_\sigma \] is consistent. This constraint set is equal to
\[ \left( \bigcup_{j \neq i} C_j \right) \cup \mathcal{C}_i' \subseteq \sigma_Q \cup \mathcal{C}_\sigma \] i.e. to
\[ \left( \bigcup_{j \neq i} C_j \right) \cup \mathcal{C}_i' \subseteq \sigma_Q \cup \mathcal{C}_\sigma \] i.e.
\[ \left( \bigcup_{j \neq i} C_j \right) \cup \mathcal{C}_i' \subseteq \sigma_Q \cup \mathcal{C}_\sigma \] which is consistent by hypothesis. Hence the claim holds in this case.

In the remaining cases, from the form of the typing rules for processes, the head symbol of neither \( P_i \) nor \( Q_i \) is out. Thus, the reduction applied to \( P_i \) (from the assumption), as well as the one applied to \( Q_i \) (we still must show it exists and is the same as for \( P_i \), except in case \( P i LR \), where \( P_i \) can follow one branch of the conditional while \( Q_i \) follows the other), cannot be Out. Therefore no new term is output on either side, and \( \phi'_P = \phi_P \) and \( \phi'_Q = \phi_Q \). Hence the claim on the domains of the frames holds by assumption. Moreover, as we will see, in all cases \( \Gamma' \) is either \( \Gamma \), or \( \Gamma, x : T \) where \( x \) is a variable declared in (the head of) \( P_i \) and \( Q_i \), and \( T \) is not a union type. The condition that \( \mathcal{C}_\Gamma = \mathcal{C}_\Gamma \) formulated previously will thus hold.

We choose \( c'_\phi = c_\phi \). The claim that \( \Gamma' \vdash P_i \sim \phi'_P : LL \rightarrow c'_\phi \) is then actually that \( \Gamma' \vdash P_i \sim \phi'_P : LL \rightarrow c_\phi \), which is true by Lemma B.10, since by hypothesis \( \Gamma \vdash P_i \sim \phi'_P : LL \rightarrow c_\phi \).

Besides, in the cases where we choose \( \Gamma' = \Gamma \) then it is true (by hypothesis) that for \( j \neq i \), \( \Gamma' \vdash P'_j \sim Q'_j \rightarrow C_j \). In the cases where \( \Gamma \) choose \( \Gamma' = \Gamma, x : T \), where \( x \) is bound in \( P_i \) and \( Q_i \), then, since the processes are assumed to use different variable names, \( x \) does not appear in \( P_i \) or \( Q_i \) (for \( j \neq i \)). Hence, if \( j \neq i \), using the assumption that \( \Gamma \vdash P_i \sim Q_i \rightarrow C_i \), by Lemma B.10, we have \( \Gamma' \vdash P'_j \sim Q'_j \rightarrow C'_j \), where \( C'_j = \{(c, \Gamma_x \cup \{(x : T)) | (c, \Gamma_x) \in C_j\} \}

Hence, for each remaining possible last rule of \( \Pi_i \), we only have to show that:

a) The same reduction rule can be applied to \( Q_i \) as to \( P_i \), with the same action. (Except in the case of the rule \( P i LR \), as we will see, where rule If-Then may be applied on one side while rule If-Else is applied on the other side, but this has no influence on the argument, as these two rules both represent a silent action, and have a very similar form.)

b) \( (\sigma'_P, \sigma'_Q) \) are ground, and \( \Gamma' \vdash \sigma'_P \sim \sigma'_Q : \Gamma' \rightarrow c'_\phi \) for some set of constraints \( c'_\phi \). Since at most one variable \( x \) is added to the substitutions in the reduction, we only have to check that condition on this variable, i.e. \( \Gamma' \vdash \sigma'_P(x) \sim \sigma'_Q(x) : \Gamma'(x) \rightarrow c_x \) for some \( c_x \). We can then choose \( c'_x = c_x \cup c_\phi \). As we will see in the proof, we will always have \( c_x \subseteq \bigcup_{(c, \Gamma_x) \in C_j} \cup c_\sigma \).

c) the new processes obtained by reducing \( P_i \) and \( Q_i \) are typably equivalent in \( \Gamma' \), with a constraint \( C_j' \), such that
\[ \left( \bigcup_{j \neq i} C_j \right) \cup \mathcal{C}_i' \subseteq \sigma_Q \cup \mathcal{C}_\sigma \] is consistent.

The actual claim, from the statement of the lemma, is that
\[ \left( \bigcup_{j \neq i} C_j \right) \cup \mathcal{C}_i' \subseteq \sigma_Q \cup \mathcal{C}_\sigma \] is consistent, but we can show that the previous condition is sufficient.

In the case where \( \Gamma = \Gamma' \), we have \( \sigma'_P = \sigma_P, \sigma'_Q = \sigma_Q, C'_j = C_j \) for \( j \neq i \), and \( c'_x = c_x \). Thus the proposed condition is clearly sufficient (it is even necessary in this case).

In the case where \( \Gamma' = \Gamma, x : T \) for some \( T \) which is not a union type, and the substitutions \( \sigma'_P, \sigma'_Q \) are \( \sigma_P, \sigma_Q \) extended with a term associated to \( x \), the proof that the condition is sufficient is more involved. First, we show that \( \left( \bigcup_{j \neq i} C_j' \right) \cup \mathcal{C}_i' = \left( \bigcup_{j \neq i} C_j \right) \cup \mathcal{C}_i' \).

Indeed, if \( S \) denotes the set \( \left( \bigcup_{j \neq i} C_j' \right) \cup \mathcal{C}_i' \), we have
\[ S = \{(c'_j \cup \Gamma_j', \forall j, (c'_j, \Gamma_j') \in C_j' \wedge \forall j, (c'_j, \Gamma_j') \text{ are compatible}) \}
\[ = \{(c'_j \cup \Gamma_j', \forall j, (c'_j, \Gamma_j') \in C_j' \wedge \forall j \neq i, (c_j, \Gamma_j) \in C_j \) \wedge \forall j \neq i, j' \neq i, (\Gamma_j, x : T) \text{ are compatible} \wedge \forall j \neq i, (\Gamma_j, x : T) \text{ are compatible}) \} \]
since we already know that for \( j \neq i, C'_j = \{(c, \Gamma_c \cup \{x : T\})| (c, \Gamma_c) \in C_j\}\). Assuming we show that \( \Gamma, x : T \vdash P'_i \sim Q'_i \rightarrow C'_i\), by Lemma B.12, we will also have that all the \( \Gamma_j \) appearing in the elements of \( C'_i \) contain \( x : T \) (since \( T \) is not a union type). Hence:

\[
S = \{(c'_j \cup \bigcup_{j \neq i} c_j, \Gamma'_j \cup \bigcup_{j \neq i} \Gamma_j) \mid (c'_j, \Gamma'_j) \in C'_i \wedge (\forall j \neq i. (c_j, \Gamma_j) \in C_j) \wedge (\forall j \neq i, j' \neq i. \Gamma_j \text{ and } \Gamma_{j'} \text{ are compatible}) \wedge (\forall j \neq i. \Gamma'_j \text{ and } \Gamma_j \text{ are compatible}))\}
\]

It is thus sufficient to ensure the consistency of

\[
\left[ \begin{array}{l}
\bigcup_{\sigma_{\phi}} \forall \psi \sigma_{\psi} \cup \forall \sigma_{\sigma} \cup x \sigma_{\phi} \\
\end{array} \right]_{\sigma_{\phi}, \sigma_{\sigma} \sigma_{\sigma}}
\]

\( \text{i.e., since } c_{\sigma} \text{ and } c_x \text{ are ground (since the substitutions are), that}
\]

\[
\left[ \begin{array}{l}
\bigcup_{\sigma_{\phi}} \forall \psi \sigma_{\psi} \cup \forall \sigma_{\sigma} \cup x \sigma_{\phi} \\
\end{array} \right]_{\sigma_{\phi}, \sigma_{\sigma} \sigma_{\sigma}}
\]

is consistent. Using Lemma B.11, since \( \sigma'_p = \sigma_p \cup \sigma'_p(x)/x \) (and similarly for \( Q \)), it then suffices to show the consistency of

\[
\left[ \begin{array}{l}
\bigcup_{\sigma_{\phi}} \forall \psi \sigma_{\psi} \cup \forall \sigma_{\sigma} \cup x \sigma_{\phi} \\
\end{array} \right]_{\sigma_{\phi}, \sigma_{\sigma} \sigma_{\sigma}} (\sigma'_p(x)/x, \sigma'_q(x)/x)
\]

which is equal to

\[
\left[ \begin{array}{l}
\bigcup_{\sigma_{\phi}} \forall \psi \sigma_{\psi} \cup \forall \sigma_{\sigma} \cup x \sigma_{\phi} \\
\end{array} \right]_{\sigma_{\phi}, \sigma_{\sigma} \sigma_{\sigma}} (\sigma'_p(x)/x, \sigma'_q(x)/x)
\]

since \( c_x \subseteq \left[ \begin{array}{l}
\bigcup_{\sigma_{\phi}} \forall \psi \sigma_{\psi} \cup \forall \sigma_{\sigma} \cup x \sigma_{\phi} \\
\end{array} \right]_{\sigma_{\phi}, \sigma_{\sigma} \sigma_{\sigma}} \) (by point b)). Moreover, as we observed previously, the environments in all elements in \( \left[ \begin{array}{l}
\bigcup_{\sigma_{\phi}} \forall \psi \sigma_{\psi} \cup \forall \sigma_{\sigma} \cup x \sigma_{\phi} \\
\end{array} \right]_{\sigma_{\phi}, \sigma_{\sigma} \sigma_{\sigma}} \) contain \( x : T \).

Therefore by Lemma B.11, since \( \Gamma_{N,K} \vdash \sigma'_p(x) \sim \sigma'_q(x) : T \rightarrow c_x \) (as we will show, as point b)), it suffices to ensure that

\[
\left[ \begin{array}{l}
\bigcup_{\sigma_{\phi}} \forall \psi \sigma_{\psi} \cup \forall \sigma_{\sigma} \cup x \sigma_{\phi} \\
\end{array} \right]_{\sigma_{\phi}, \sigma_{\sigma} \sigma_{\sigma}} \forall \psi \psi \sigma_{\psi} \cup x \psi \psi \sigma_{\psi}
\]

is consistent, to prove the claim. This is the condition stated at the beginning of this point, since \( S = (\cup_{\Gamma_{j \neq i} C_j}) \cup x C'_i \).

We can now prove the remaining cases for the last rule of \( \Pi_i \), that is to say the cases of the rules \( \text{PIN}, \text{PLET}, \text{PLET}^\text{LR}, \text{PLiL}, \text{PLiLR}, \text{PLiS}, \text{PLiLR}^*, \text{PLiP}, \text{PLiL}, \text{PLiLR}^* \).

- **PIN:** then \( P_i = \text{in} (x) \cdot P'_i \) and reduces to \( P'_i \) via the \( \text{In} \) rule, and \( Q_i = \text{in} (x) \cdot Q'_i \) for some \( Q'_i \). In addition

\[
\Pi = \begin{array}{l}
\Gamma, x : LL \vdash P'_i \sim Q'_i \rightarrow C'_i \\
\Gamma \vdash P_i \sim Q_i \rightarrow C_i \rightarrow C'_i \\
\end{array}
\]

We have \( a = \text{in}(R) \) for some attacker recipe \( R \) such that \( \text{vars}(R) \subseteq \text{dom}(\phi_p) \), and \( R \phi_p \sigma_P \perp \perp \). We also have \( E' = E_h \), \( \sigma'_p = \sigma_p \cup \{ R \phi_p \sigma_P \downarrow / x \} \), \( \delta'_p = \phi_p \).

The same reduction rule \( \text{In} \) can be applied to reduce the process \( Q_i \) into \( Q'_i \). Indeed, \( \text{vars}(R) \subseteq \text{dom}(\phi_Q) \) since \( \text{dom}(\phi_Q) = \text{dom}(\phi_P) \) by hypothesis;

- \( R \phi_Q \sigma_Q \perp \perp \). This follows from Lemma B.21, using the fact that by Lemma B.22, \( \Gamma_{N,K} \vdash \phi_p \sigma_P \sim \phi_Q \sigma_Q : LL \rightarrow c \), for some

\[
c \subseteq \left[ \begin{array}{l}
\bigcup_{\sigma_{\phi}} \forall \psi \sigma_{\psi} \cup \forall \sigma_{\sigma} \cup x \sigma_{\phi} \\
\end{array} \right]_{\sigma_{\phi}, \sigma_{\sigma} \sigma_{\sigma}} \cup c_x.
\]

Therefore point a) holds.

We choose \( \Gamma' = \Gamma, x : LL \). We have \( \sigma'_Q = \sigma_Q \cup \{ R \phi_Q \sigma_Q \downarrow / x \} \).

Lemmas B.22 and B.21, previously evoked, guarantee that

\[
\Gamma_{N,K} \vdash R \phi_p \sigma_P \downarrow \sim R \phi_Q \sigma_Q \downarrow : LL \rightarrow c'
\]

for some \( c' \subseteq \left[ \begin{array}{l}
\bigcup_{\sigma_{\phi}} \forall \psi \sigma_{\psi} \cup \forall \sigma_{\sigma} \cup x \sigma_{\phi} \\
\end{array} \right]_{\sigma_{\phi}, \sigma_{\sigma} \sigma_{\sigma}} \). This proves point b).

Moreover, \( \Pi \) and the fact that

\[
\left[ \begin{array}{l}
(\cup_{\Gamma_{j \neq i} C_j}) \cup x C'_i \cup x \sigma_{\phi} \\
\end{array} \right]_{\sigma_{\phi}, \sigma_{\sigma} \sigma_{\sigma}} \cup x \psi \psi \sigma_{\psi} = \left[ \begin{array}{l}
(\cup_{\Gamma_{j \neq i} C_j}) \cup x \sigma_{\phi} \\
\end{array} \right]_{\sigma_{\phi}, \sigma_{\sigma} \sigma_{\sigma}} \cup x \psi \psi \sigma_{\psi}
\]

which is consistent by hypothesis, prove point c) and conclude this case.
\textbf{P LET:} then $P_i = 1 \text{ let } x = d(y) \text{ in } P_i' \text{ else } P_i''$ and $Q_i = 1 \text{ let } x = d(y) \text{ in } Q_i' \text{ else } Q_i''$ for some $P_i', P_i'', Q_i', Q_i''$. $P_i$ reduces to either $P_i'$ via the Let-In rule, or $P_i''$ via the Let-Else rule. In addition
\[
\Pi \vdash P_i \sim Q_i \Rightarrow C_i = C_i' \cup C_i''.
\]
We have $\alpha = \tau$.
Since $\Gamma_N \vdash \sigma_P(\gamma) \sim \sigma_Q(\gamma) : \Gamma(y) \Rightarrow c_y$ (for some $c_y \subseteq c_\sigma$, by hypothesis), and using $\Pi$, by Lemma B.20, we have:
\[
d(\sigma_P(\gamma)) \Downarrow \bot \iff d(\sigma_Q(\gamma)) \Downarrow \bot.
\]
Therefore, if rule Let-In is applied to $P_i$ then it can also be applied to reduce $Q_i$ into $Q_i'$, and if the rule applied to $P_i$ is Let-Else then it can also be applied to reduce $Q_i$ into $Q_i''$. This proves point a). We prove here the Let-In case. The Let-Else case is similar (although slightly easier, since no new variable is added to the substitutions).
In this case we have $\sigma_P' = \sigma_P \cup \{d(\sigma_P(\gamma)) \Downarrow \bot / x\}$ and $\sigma_Q' = \sigma_Q \cup \{d(\sigma_Q(\gamma)) \Downarrow \bot / x\}$.
By Lemma B.20, we know in this case that there exists $c \subseteq c_y$ such that $\Gamma_N \vdash d(\sigma_P(\gamma)) \Downarrow \bot \Rightarrow c$. Thus, by Lemma B.2, there exists $T' \in \text{branches}(T)$ such that $\Gamma_N \vdash d(\sigma_P(\gamma)) \Downarrow \bot \Rightarrow T' \Rightarrow c$.
We choose $T' = \Gamma, x : T'$. Since $\Gamma$ does not contain union types, $\Gamma' \in \text{branches}(\Gamma, x : T)$.
Since $c \subseteq c_y \subseteq c_\sigma$ and $\Gamma_N \vdash d(\sigma_P(\gamma)) \Downarrow \bot \Rightarrow T' \Rightarrow c$, point b) holds.
We now prove that point c) holds. Using $\Pi'$, we have $\Gamma, x : T \vdash P_i' \sim Q_i' \Rightarrow C_i'$. Hence, by Lemma B.7, there exists $C_i''' \subseteq C_i'' \subseteq C_i'$ such that $\Gamma' \vdash P_i' \sim Q_i' \Rightarrow C_i'''$.
Since $C_i''' \subseteq C_i$, we have
\[
\left(\bigcup_{X \in \mathbb{F}} C_i'''' \cup \psi \sigma_c\right)_{\sigma_P, \sigma_Q} \cup \psi_c \sigma = \left(\bigcup_{X \in \mathbb{F}} C_i'''' \cup \psi \sigma_c\right)_{\sigma_P, \sigma_Q} \cup \psi_c \sigma.
\]
This last constraint set is consistent by hypothesis. Hence, by Lemma B.11, $\left(\bigcup_{X \in \mathbb{F}} C_i'''' \cup \psi \sigma_c\right)_{\sigma_P, \sigma_Q} \cup \psi_c \sigma$ is also consistent.
This proves point c) and concludes this case.

\textbf{P LETLR:} then $P_i = 1 \text{ let } x = d(y) \text{ in } P_i' \text{ else } P_i''$ and $Q_i = 1 \text{ let } x = d(y) \text{ in } Q_i' \text{ else } Q_i''$ for some $P_i', P_i'', Q_i', Q_i''$. $P_i$ reduces to either $P_i'$ via the Let-In rule, or $P_i''$ via the Let-Else rule. In addition
\[
\Pi \vdash P_i \sim Q_i \Rightarrow C_i = C_i' \cup C_i''.
\]
We have $\alpha = \tau$.
By hypothesis, $\sigma_P, \sigma_Q$ are ground and $\Gamma_N \vdash \sigma_P \sim \sigma_Q : \Gamma_X \Rightarrow c_\sigma$. Hence, by definition of the well-typedness of substitutions, there exists $c_y \subseteq c_\sigma$ such that $\Gamma_N \vdash \sigma_P(\gamma) \sim \sigma_Q(\gamma) : \Gamma_m \vdash \Gamma_n \Rightarrow c_\sigma$. Therefore by Lemma B.14, $\sigma_P(\gamma) = \sigma_Q(\gamma) = \gamma$.
Since $m, n$ are nonces, $d(m) \Downarrow \bot = d(n)$, and we thus have $d(\sigma_P(\gamma)) \Downarrow \bot$. Therefore the reduction rule applied to $P_i$ can only be Let-Else, and $P_i$ is reduced to $P_i''$. Since we also have $d(\sigma_Q(\gamma)) \Downarrow \bot$, this rule can also be applied to reduce $Q_i$ into $Q_i''$. This proves point a).
We therefore have $\sigma_P' = \sigma_P$ and $\sigma_Q' = \sigma_Q$. We choose $\Gamma' = \Gamma$.
Since the substitutions and typing environments are unchanged by the reduction, point b) clearly holds.
Moreover, $\Pi''$, and the fact that
\[
\left(\bigcup_{X \in \mathbb{F}} C_i'''' \cup \psi \sigma_c\right)_{\sigma_P, \sigma_Q} \cup \psi_c \sigma = \left(\bigcup_{X \in \mathbb{F}} C_i'''' \cup \psi \sigma_c\right)_{\sigma_P, \sigma_Q} \cup \psi_c \sigma
\]
which is consistent by hypothesis, prove point c) and conclude this case.

\textbf{P B:} then $P_i = 1 \text{ if } M = M' \text{ then } P_i^T \text{ else } P_i^H$ and $Q_i = 1 \text{ if } N = N' \text{ then } Q_i^T \text{ else } Q_i^H$ for some $P_i^T, P_i^H, Q_i^T, Q_i^H$. $P_i$ reduces to $P_i^T$ which is either $P_i^T$ via the If-Then rule, or $P_i^H$ via the If-Else rule. In addition
\[
\Pi \vdash P_i \sim Q_i \Rightarrow C_i = C_i' \cup C_i''.
\]
We have $\alpha = \tau$, and $E' = E_\tau$. 
Since \( \Gamma \vdash M \sim N : LL \rightarrow \epsilon \), by Lemma B.22, there exists \( e'' \subseteq \{c\}_{\sigma_p, \sigma_Q} \cup \epsilon \sigma \) such that \( \Gamma_{N, \chi} \vdash M \sigma_p \sim N \sigma_Q : LL \rightarrow e'' \). Similarly, there exists \( e''' \subseteq \{c'\}_{\sigma_p, \sigma_Q} \cup \epsilon \sigma \) such that \( \Gamma_{N, \chi} \vdash M \sigma_p \sim N' \sigma_Q : LL \rightarrow e''' \).

Let \( \phi = \{ M \sigma_p / x, M' \sigma_p / y \} \) and \( \phi' = \{ N \sigma_Q / x, N' \sigma_Q / y \} \). We then have \( \Gamma_{N, \chi} \vdash \phi \sim \phi' : LL \rightarrow e'' \cup e''' \).

Let us prove that \( e \cup e'' \) is consistent in some type environment. By hypothesis, \( \sigma_p, \sigma_Q \) are ground and \( \Gamma_{N, \chi} \vdash \sigma_p \sim \sigma_Q : \Gamma_{\chi} \rightarrow \epsilon \).

Hence, by Lemma B.4, there exists \( \Gamma'' \in \text{branches}(\Gamma) \) such that \( \Gamma'' \vdash \sigma_p \sim \sigma_Q : \Gamma'' \rightarrow \epsilon \). By Lemma B.13, there exists \( (c_1, \Gamma''') \in \Gamma'' \) such that \( \Gamma''' \subseteq \Gamma'' \). Since \( C_1 = (C_1^T \cup C_1^F) \cup \psi(c \cup c') \), \( c_1 \) is of the form \( c_2 \cup c \cup c' \) for some \( c_2 \).

As we noted previously, \( \{c\}_{\sigma_p, \sigma_Q} \cup \psi \epsilon \sigma \) is consistent. Therefore, by Lemma B.11, \( \{\{c \cup c'\}_{\sigma_p, \sigma_Q} \cup \epsilon \sigma, \Gamma'''\} \) is consistent. Hence, by the same Lemma, \( \epsilon \cup e'' \) is also consistent in \( \Gamma''' \).

Thus, by Lemma B.25, \( \phi \) and \( \phi' \) are statically equivalent. Hence, in particular, \( M \sigma_p \leftrightarrow N \sigma_Q = N' \sigma_Q \).

Therefore, if rule If-Then is applied to \( P_i \) then it can also be applied to reduce \( Q_i \) into \( Q_i^T \), and if the rule applied to \( P_i \) is If-Else then it can also be applied to reduce \( Q_i \) into \( Q_i^E \). This proves point a). We prove here the If-Then case. The If-Else case is similar.

We choose \( \Gamma' = \Gamma \). We have \( \sigma_p \equiv \sigma_p \) and \( \sigma_Q \equiv \sigma_Q \).

Since the substitutions and environments do not change in this reduction, point b) trivially holds.

Moreover, by hypothesis, \( \{c \}_{\sigma_p, \sigma_Q} \cup \psi \epsilon \sigma \) is consistent. Thus by Lemma B.11, \( \{\{c \}_{\sigma_p, \sigma_Q} \cup \psi \epsilon \sigma, \sigma_p, \sigma_Q \} \subseteq \{\{c \}_{\sigma_p, \sigma_Q} \cup \psi \epsilon \sigma, \sigma_p, \sigma_Q \} \).

Thus the reduction rule applied to \( P_i \) is If-Then and \( P_i^T = P_i^T \). On the other hand, rule If-Else can be applied to reduce \( Q_i \) into \( Q_i^E = Q_i^E \). This proves point a) (these rules both correspond to silent actions).

We choose \( \Gamma' = \Gamma \). We have \( \sigma_p \equiv \sigma_p \) and \( \sigma_Q \equiv \sigma_Q \).

Since the substitutions and environments do not change in this reduction, point b) trivially holds.

Moreover, \( \Gamma' \) and the fact that \( \{c \}_{\sigma_p, \sigma_Q} \cup \psi \epsilon \sigma \) prove point c) and conclude this case.

- **PhLS**: then \( P_i = \text{if } M = M' \text{ then } P_i^L \text{ else } P_i^L \) and \( Q_i = \text{if } N = N' \text{ then } Q_i^L \text{ else } Q_i^L \) for some \( Q_i^L, Q_i^E \). \( P_i \) reduces to \( P_i^T \) which is either \( P_i^T \) via the If-Then rule, or \( P_i^T \) via the If-Else rule. In addition

\[
\Pi'' \quad \Pi'' \quad \Pi''
\]

\[
\Pi'' \quad \Pi'' \quad \Pi''
\]

\[
\Pi'' \quad \Pi'' \quad \Pi''
\]

\[
\Pi'' \quad \Pi'' \quad \Pi''
\]

\[
\Pi'' \quad \Pi'' \quad \Pi''
\]

We have \( \alpha = \tau \) in any case.
We then show that $\alpha \Gamma$. We choose $\Gamma M\sigma$ and $\sigma$. Therefore by Lemma B.23, $M\sigma$ and $\sigma$. Hence the reduction for $P_i$ is necessarily If-Else, which is also applicable to reduce $Q_i$ to $Q_i^\perp$. This proves point a).

We choose $\Gamma^\prime = \Gamma$. We have $\sigma^\prime \equiv \sigma$ and $\sigma^\prime Q = \sigma Q$.

Since the substitutions and typing environments do not change in this reduction, point b) trivially holds.

Moreover, $\Pi^\prime$ and the fact that

$$\left[(\bigcup_{x \in I} C_j)x C_j[U \forall V \phi]\right]_{\sigma P, \sigma Q} U \forall V \phi = \left[(\bigcup_{x \in I} C_j)x C_j[U \forall V \phi]\right]_{\sigma P, \sigma Q}$$

prove point c) and conclude this case.

- **PrlI:** then $P_i = \text{if } M = M^\prime \text{ then } P_i^\perp$ else $P_i^\perp$ and $Q_i = \text{if } N = N^\prime \text{ then } Q_i^\perp$ else $Q_i^\perp$ for some $Q_i^\perp$, $Q_i^\perp$. This case is similar to the PrS case: the incompatibility of the types of $M$, $N$ and $M^\prime$, $N^\prime$ ensures that the processes can only follow the else branch. $P_i$ reduces to $P_i^\perp$ which is either $P_i^\perp$ via the If-Then rule, or $P_i^\perp$ via the If-Else rule. In addition

$$\Pi^\perp \quad \Pi^I$$

$$\Pi^I_i = \frac{\Gamma \vdash P_i^\perp \sim Q_i^\perp \rightarrow C_i^\perp}{\Gamma \vdash M \sim N : T \sim T^\prime \rightarrow c} \quad \Pi^\perp \quad \Pi^I$$

$$\Gamma \vdash P_i \sim Q_i \rightarrow C_i = C_i^\perp$$

We have $a = \tau$ in any case.

By hypothesis, $\sigma P, \sigma Q$ are ground and $\Gamma M\sigma \vdash \sigma P \sim \sigma Q : \Gamma X \rightarrow \epsilon_\sigma$. Hence, by Lemma B.22, using $\Pi$, there exists $\epsilon$ such that $\Gamma M\sigma \vdash \sigma P \sim \sigma Q : T \sim T^\prime \rightarrow c$. By Lemma B.17, this implies that $\sigma P$ and $\sigma Q$ both are pairs. Similarly we can show that $\Gamma M\sigma \vdash \sigma P \sim \sigma Q : \Gamma_{[m, a]} \rightarrow \epsilon^\prime$ for some $\epsilon^\prime$. By Lemma B.14, this implies that $\sigma P = m$ and $\sigma Q \sim \sigma N$. Thus neither of these two terms are pairs.

Therefore $M\sigma P \neq M^\prime\sigma P$ and $N\sigma Q \neq N^\prime\sigma Q$. The end of the proof for this case is then the same as for the PrS case.

- **PrlS:** then $P_i = \text{if } M = t \text{ then } P_i^\perp$ else $P_i^\perp$ and $Q_i = \text{if } N = t \text{ then } Q_i^\perp$ else $Q_i^\perp$ for some $Q_i^\perp$, $Q_i^\perp$, some messages $M$, $N$, and some $t \in C \cup K \cup N$. $P_i$ reduces to $P_i^\perp$ which is either $P_i^\perp$ via the If-Then rule, or $P_i^\perp$ via the If-Else rule. In addition

$$\Pi^\perp \quad \Pi^I$$

$$\Pi_i = \frac{\Gamma \vdash M \sim N : T \sim T^\prime \rightarrow c}{\Gamma \vdash P_i \sim Q_i \rightarrow C_i = C_i^\perp}$$

We have in any case $a = \tau$.

By hypothesis, $\sigma P, \sigma Q$ are ground and $\Gamma M\sigma \vdash \sigma P \sim \sigma Q : \Gamma X \rightarrow \epsilon_\sigma$. Hence, by Lemma B.22, using $\Pi$, there exists $\epsilon' \subseteq [c]_{\sigma P, \sigma Q} \cup \epsilon_\sigma$ such that $\Gamma M\sigma \vdash \sigma P \sim \sigma Q : LL \rightarrow c'$.

We then show that $M\sigma P = t$ if and only if $N\sigma Q = t$ (note that since $t$ is ground, $t = t\sigma P = t\sigma Q$). If $M\sigma P = t$, then $\Gamma M K \vdash t \sim N\sigma Q : LL \rightarrow c'$. In all possible cases for $t$, i.e., $t \in K, t \in N$, and $t \in C$, Lemma B.18 implies that $N\sigma Q = t$. This proves the first direction of the equivalence, the other direction is similar.

Therefore, if rule If-Then is applied to $P_i$, then it can also be applied to reduce $Q_i$ into $Q_i^\perp$, and if the rule applied to $P_i$ is If-Else then it can also be applied to reduce $Q_i$ into $Q_i^\perp$. This proves point a). We prove here the If-Then case. The If-Else case is similar.

We choose $\Gamma^\prime = \Gamma$. We have $\sigma^\prime \equiv \sigma$ and $\sigma^\prime Q = \sigma Q$.

Since the substitutions and typing environments do not change in this reduction, point b) trivially holds.

Moreover, by hypothesis,

$$\left[(\bigcup_{x \in I} C_j)x C_j[U \forall V \phi]\right]_{\sigma P, \sigma Q} U \forall V \phi$$

is consistent. Since, using $C_i^\prime = C_i^\perp$ and $C_i = (C_i^\perp \cup C_i^\perp)$, we have

$$\left[(\bigcup_{x \in I} C_j)x C_j[U \forall V \phi]\right]_{\sigma P, \sigma Q} U \forall V \phi \subseteq \left[(\bigcup_{x \in I} C_j)x C_j[U \forall V \phi]\right]_{\sigma P, \sigma Q} U \forall V \phi$$

we have by Lemma B.11 that $\left[(\bigcup_{x \in I} C_j)x C_j[U \forall V \phi]\right]_{\sigma P, \sigma Q} U \forall V \phi$ is consistent. This fact proves point c) and concludes this case.
• **PfLR**: then \( P_i = \text{if } M_1 = M_2 \text{ then } P_1^\top \text{ else } P_1^\downarrow \) and \( Q_i = \text{if } N_1 = N_2 \text{ then } Q_1^\top \text{ else } Q_1^\downarrow \) for some \( Q_1^\top, Q_1^\downarrow \). \( P_i \) reduces to \( P_i' \) which is either \( P_i^\top \) via the If-Then rule, or \( P_i^\downarrow \) via the If-Else rule. In addition

\[
\begin{align*}
\Pi_i \equiv & \quad \frac{\Pi' \quad \Pi''}{\Pi} \\
& \quad \frac{\Gamma \vdash M_1 \sim N_1 : [\pi_m^L ; \pi_n^L] \rightarrow c_1}{\Pi_i}
\end{align*}
\]

We have \( \alpha = \tau \) in any case.

By hypothesis, \( \sigma_p, \sigma_Q \) are ground and \( \Gamma_N, \mathcal{K} \vdash \sigma_p \sim \sigma_Q : \Gamma \rightarrow c_\sigma \). Hence, by Lemma B.22, using \( \Pi \), there exists \( \varepsilon'' \) such that \( \Gamma_N, \mathcal{K} \vdash M_1\sigma_p \sim N_1\sigma_Q : [\pi_m^L ; \pi_n^L] \rightarrow \varepsilon'' \). Therefore by Lemma B.14, \( M_1\sigma_p = m \) and \( N_1\sigma_Q = n \). Similarly we can show that \( M_2\sigma_p = m \) and \( N_2\sigma_Q = n \).

Hence \( M'_1 = M'_2 \) and \( N'_1 = N'_2 \).

Thus the reduction rule applied to \( P_i \) is If-Then and \( P_i' = P_i^\top \). On the other hand, rule If-Then can also be applied to reduce \( Q_i \) into \( Q_i' = Q_i^\top \). This proves point a).

Note that we still need to type the other branch, even though it is not used here, as when replicating the process this test may fail if \( M_1, N_1 \) and \( M_2, N_2 \) are nonces from different sessions.

We choose \( \Gamma' = \Gamma \). We have \( \sigma_p' = \sigma_p \) and \( \sigma_Q' = \sigma_Q \).

Since the substitutions and environments do not change in this reduction, point b) trivially holds.

Moreover, \( \Pi'' \) and the fact that, with \( C_i' = C_i^\top \),

\[
\left[ (U_{XJ'} C_j) U_X C_i' \cup \cup \psi \psi \right]_{\sigma_p, \sigma_Q} \cup \psi \psi \sigma = \left[ (U_{XJ'} C_j) U_X C_i' \cup \cup \psi \psi \right]_{\sigma_p, \sigma_Q} \cup \psi \psi \sigma
\]

prove point c) and conclude this case.

• **PfLR**: then \( P_i = \text{if } M_1 = M_2 \text{ then } P_1^\top \text{ else } P_1^\downarrow \) and \( Q_i = \text{if } N_1 = N_2 \text{ then } Q_1^\top \text{ else } Q_1^\downarrow \) for some \( Q_1^\top, Q_1^\downarrow \). \( P_i \) reduces to \( P_i' \) which is either \( P_i^\top \) via the If-Then rule, or \( P_i^\downarrow \) via the If-Else rule. In addition

\[
\begin{align*}
\Pi'' \quad \Pi''' \quad \Pi'''' \quad \Pi''''' \\
\frac{\Gamma \vdash M_1 \sim N_1 : [\pi_m^L ; \pi_n^L] \rightarrow c_1}{\Pi}
\end{align*}
\]

We have \( \alpha = \tau \) in any case.

By hypothesis, \( \sigma_p, \sigma_Q \) are ground and \( \Gamma_N, \mathcal{K} \vdash \sigma_p \sim \sigma_Q : \Gamma \rightarrow c_\sigma \). Hence, by Lemma B.22, using \( \Pi \), there exists \( \varepsilon'' \) such that \( \Gamma_N, \mathcal{K} \vdash M_1\sigma_p \sim N_1\sigma_Q : [\pi_m^L ; \pi_n^L] \rightarrow \varepsilon'' \). Therefore by Lemma B.14, \( M_1\sigma_p = m \) and \( N_1\sigma_Q = n \). Similarly, using Lemma B.14, we can show that \( M_2\sigma_p = m' \) and \( N_2\sigma_Q = n' \).

Moreover, since \( \pi_m^l \neq \pi'_m \), we know that \( m \neq m' \) (by well-formedness of the processes), and similarly \( n \neq n' \).

Hence, \( M_1\sigma_p \neq M_2\sigma_p \) and \( N_1\sigma_Q \neq N_2\sigma_Q \).

Thus the reduction rule applied to \( P_i \) is If-Else and \( P_i' = P_i^\downarrow \). On the other hand, rule If-Else can also be applied to reduce \( Q_i \) into \( Q'_i = Q_i^\top \). This proves point a).

We choose \( \Gamma' = \Gamma \). We have \( \sigma_p' = \sigma_p \) and \( \sigma_Q' = \sigma_Q \).

Since the substitutions and environments do not change in this reduction, point b) trivially holds.

Moreover, \( \Pi'' \) and the fact that

\[
\left[ (U_{XJ'} C_j) U_X C_i' \cup \cup \psi \psi \right]_{\sigma_p, \sigma_Q} \cup \psi \psi \sigma = \left[ (U_{XJ'} C_j) U_X C_i' \cup \cup \psi \psi \right]_{\sigma_p, \sigma_Q} \cup \psi \psi \sigma
\]

prove point c) and conclude this case.

\[ \square \]

**Theorem B.27 (Typing implies trace inclusion).** For all processes \( P, Q \) for all \( \phi_P, \phi_Q, \sigma_p, \sigma_Q \), for all multisets of processes \( P, Q \) for all constraints \( C \), for all sequences of actions, for all \( \Gamma \) containing only keys,

\[
\Gamma \vdash P \sim Q \rightarrow C,
\]

and if \( C \) is consistent, then

\[
P \sqsubseteq_1 Q
\]
that is, if
\[
(0, \{P\}, 0, 0) \overset{s}{\rightarrow} (E_P, P, \phi_P, \sigma_P),
\]
then there exists a sequence \( s' \) of actions, a multiset \( Q \), a set of names \( E_Q \), a frame \( \phi_Q \), a substitution \( \sigma_Q \), such that

- \( s \sim s' \)
- \( (0, \{Q\}, 0, 0) \overset{s'}{\rightarrow} (E_Q, Q, \phi_Q, \sigma_Q) \)
- new \( E_P \cdot \phi_P \sigma_P \) and new \( E_Q \cdot \phi_Q \sigma_Q \) are statically equivalent.

**Proof.** We successively apply Lemma B.26 to each of the reduction steps in the reduction
\[
(0, \{P\}, 0, 0) \overset{s}{\rightarrow} (E_P, P, \phi_P, \sigma_P).
\]
The lemma can indeed be applied successively. At each reduction step of \( P \) we obtain a sequence of reduction steps for \( Q \) with the same actions, and the conclusions the lemma provides imply the conditions needed for its next application.

It is clear, for the first application, that all the hypotheses of this lemma are satisfied.

In the end, we know that there exist \( \Gamma' \), some constraint sets \( C_i \), some \( c_i, \epsilon_i \), and a reduction
\[
(0, \{Q\}, 0, 0) \overset{s'}{\rightarrow} (E_Q, Q, \phi_Q, \sigma_Q)
\]with \( s \sim s' \), such that (among other conclusions)

- \( E_P = E_T \)
- \( \phi_P \sigma_P \) are ground and \( \Gamma' \cdot \mathcal{K} \vdash \phi_P \sigma_P \rightarrow \Gamma' \cdot \chi \rightarrow \epsilon_P \)
- \( \iota \cdot \phi_P \sigma_P = \iota \cdot (\phi_Q) \)
- \( \forall i \cdot \Gamma' \cdot \phi_P \sigma_P \rightarrow \Gamma' \cdot \chi \rightarrow \epsilon_P \)
- \( \forall i \cdot \Gamma' \cdot \phi_P \sigma_P \rightarrow \Gamma' \cdot \chi \rightarrow \epsilon_P \)
- \( \phi_P \sigma_P \) are statically equivalent.

To prove the claim, it is then sufficient to show that new \( E_Q \cdot \phi_P \sigma_P \) and new \( E_Q \cdot \phi_Q \sigma_Q \) are statically equivalent.

We have \( \Gamma' \cdot \phi_P \sigma_P \rightarrow \phi_Q \sigma_Q : \chi \rightarrow \epsilon_P \).

Hence, by Lemma B.22, there exists \( c \subseteq [c_P] \cdot c_Q \) such that
\[
\Gamma' \cdot \mathcal{N} \cdot \chi \vdash \phi_P \sigma_P \rightarrow \phi_Q \sigma_Q : \chi \rightarrow \epsilon_P.
\]

We will now show that \( (c, \Gamma' \cdot \mathcal{N} \cdot \chi) \) is consistent. Since \( c \subseteq [c_P] \cdot c_Q \), by Lemma B.11, it suffices to show that \( [c_P] \cdot c_Q \) is consistent.

We have \( \Gamma' \cdot \phi_P \sigma_P \rightarrow \phi_Q \sigma_Q : \chi \rightarrow \epsilon_P \).

By Lemma B.12, there exists for all i some \( (c_i, \Gamma_i') \in C_i \) such that \( \Gamma'' \subseteq \Gamma_i' \). The disjointness condition on the bound variables implies by Lemma B.8 that for all \( i, j \), \( \Gamma_i' \) and \( \Gamma_j' \) are compatible. Thus \( \cup \mathcal{C}_i \) contains \( (c', \Gamma') \defeq (\cup \mathcal{C}_i, \cup \mathcal{C}_j, \Gamma'_i) \).

We have \( \Gamma'' \subseteq \Gamma' \).

Therefore, \( \Gamma'' \) is consistent.

It is then clear that \( \Gamma' \cdot \mathcal{N} \cdot \chi \subseteq \Gamma'' \).

Hence, by Lemma B.10, since \( \Gamma' \cdot \phi_P \sigma_P \rightarrow \phi_Q \sigma_Q : \chi \rightarrow \epsilon_P \), we have \( \Gamma'' \cdot \phi_P \sigma_P \rightarrow \phi_Q \sigma_Q : \chi \rightarrow \epsilon_P \).

Hence, we have \( \Gamma'' \cdot \phi_P \sigma_P \rightarrow \phi_Q \sigma_Q : \chi \rightarrow \epsilon_P \) consistent. Moreover, \( \phi_P \sigma_P \) and \( \phi_Q \sigma_Q \) are ground (by well-formedness of the processes).

Therefore, by Lemma B.25, the frames new \( E_T \cdot \phi_P \sigma_P \) and new \( E_T \cdot \phi_Q \sigma_Q \) are statically equivalent.

By definition of the reduction relation, and by well-formedness of the processes, since
\[
(0, \{P\}, 0, 0) \overset{s}{\rightarrow} (E_P, P, \phi_P, \sigma_P)
\]
and
\[
(0, \{Q\}, 0, 0) \overset{s'}{\rightarrow} (E_Q, Q, \phi_Q, \sigma_Q)
\]it is clear that names(\( \phi_P \sigma_P \)) \subseteq E_T and names(\( \phi_Q \sigma_Q \)) \subseteq E_T.

Thus, the only names that are relevant to the frames are \( E_T \).

Hence, new \( E_T \cdot \phi_P \sigma_P \) and new \( E_T \cdot \phi_Q \sigma_Q \) are statically equivalent.

This theorem corresponds to Theorem 6.1.
Theorem B.28 (Typing implies trace equivalence). For all \( \Gamma \) containing only keys, for all \( P \) and \( Q \), if
\[
\Gamma \vdash P \sim Q \rightarrow C
\]
and \( C \) is consistent, then
\[
P \approx_P Q.
\]

Proof. Theorem B.27 proves that under these assumptions, \( P \subseteq_P Q \). This is sufficient to prove the theorem. Indeed, it is clear from the typing rules for processes and terms that
\[
\Gamma \vdash P \sim Q \rightarrow C \iff \Gamma' \vdash Q \sim P \rightarrow C'
\]
where \( C' \) is the constraint obtained from \( C \) by swapping the left and right hand sides of all of its elements, and \( \Gamma' \) is the environment obtained from \( \Gamma \) by swapping the left and right types in all refinement types. Clearly from the definition of consistency, \( C \) is consistent if and only if \( C' \) is. Therefore, by symmetry, proving that the assumptions imply \( P \subseteq_P Q \) also proves that they imply \( Q \subseteq_P P \), and thus \( P \approx_P Q \).
\( \square \)

B.2 Typing replicated processes

In this subsection, we prove the soundness result for replicated processes.

In this subsection, as well as the following ones, without loss of generality we assume, for each infinite nonce type \( r_n^{L,\infty} \) appearing in the processes we consider, that \( N \) contains an infinite number of fresh names which we will denote by \( \{m_i \mid i \in \mathbb{N}\} \); such that the \( m_i \) do not appear in the processes or environments considered. We will denote by \( N_0 \) the set of unindexed names and by \( N_i \) the set of indexed names.

We similarly assume that for all the variables \( x \) appearing in the processes, the set \( X \) of all variables also contains variables \( \{x_i \mid i \in \mathbb{N}\} \). We denote \( X_0 \) the set of unindexed variables, and \( X_i \) the set of indexed variables.

Definition B.29 (Renaming of a process). For all process \( P \), for all \( i \in \mathbb{N} \), for all environment \( \Gamma \), we define \( [P]_i^\Gamma \), the renaming of \( P \) for session \( i \) with respect to \( \Gamma \), as the process obtained from \( P \) by:
- for each nonce \( n \) declared in \( P \) by new \( n : r_n^{L,\infty} \), and each nonce \( n \) such that \( \Gamma(n) = r_n^{L,\infty} \) for some \( l \), replacing every occurrence of \( n \) with \( n_i \), and the declaration new \( n : r_n^{L,\infty} \) with new \( n_i : r_n^{L,1} \);
- replacing every occurrence of a variable \( x \) with \( x_i \).

Lemma B.30 (Typing terms with replicated names). For all \( \Gamma', M, N, T \) and \( c \), if
\[
\Gamma \vdash M \sim N : T \rightarrow c
\]
then for all \( i, n \in \mathbb{N} \) such that \( 1 \leq i \leq n \), for all \( \Gamma' \in \text{branches}([\Gamma]_i^n) \),
\[
\Gamma' \vdash [M]_i^\Gamma \sim [N]_i^\Gamma : [T]_i^n \rightarrow [c]_i^\Gamma
\]
Proof. Let \( \Gamma, M, N, T, c \) be such as assumed in the statement of the lemma. Let \( i, n \in \mathbb{N} \) such that \( 1 \leq i \leq n \). Let \( \Gamma' \in \text{branches}([\Gamma]_i^n) \).

We prove this property by induction on the proof \( \Pi \) of
\[
\Gamma \vdash M \sim N : T \rightarrow c.
\]
There are several possible cases for the last rule applied in \( \Pi \).
- **TNonce**: then \( M = m \) and \( N = p \) for some \( m, p \in N, T = l \) for some \( l \in \{\text{HH, HL}\} \), and
\[
\Pi' = \frac{\Gamma(m) = r_m^{L,a} \Gamma(p) = r_p^{L,a}}{\Gamma \vdash m \sim p : l \rightarrow \emptyset}.
\]

It is clear from the definition of \([\Gamma]_i^n\) that \([\Gamma]_i^n([m]_i^\Gamma) = r_m^{L,1}([m]_i^n)\) and \([\Gamma]_i^n([p]_i^\Gamma) = r_p^{L,1}([p]_i^n)\). Hence, \([\Gamma']([m]_i^\Gamma) = r_m^{L,1}([m]_i^n)\) and \([\Gamma']([p]_i^\Gamma) = r_p^{L,1}([p]_i^n)\). Then, by rule TNonce, we have \( \Gamma' \vdash [M]_i^\Gamma \sim [N]_i^\Gamma : l \rightarrow \emptyset \) and the claim holds.

- **TNoncel, TCSelFN, TKey, TPubKey, TVKey, TMHash, THigh, TLRepl\rightleftharpoons**. Similarly to the TNonce case, the claim follows directly from the definition of \([\Gamma]_i^n\). \([M]_i^\Gamma\), \([N]_i^\Gamma\), \([T]_i^n\) and \([c]_i^\Gamma\) in these cases.
- **TEnch**: then \( T = LL \) and there exist \( T', k, c' \) such that
\[
\Pi' = \frac{\Gamma + M \sim N : (T')_k \rightarrow c' \Gamma(k) = \text{key}^{\text{HH}}(T')}{\Gamma + M \sim N : LL \rightarrow c = c' \cup \{M \sim N\}}.
\]

By applying the induction hypothesis to \( \Pi' \), since \([T']_k^n = ([T']_k^n)_k \), there exists a proof \( \Pi'' \) of \( \Gamma' \vdash [M]_i^\Gamma \sim [N]_i^\Gamma : ([T']_k^n)_k \rightarrow [c']_i^\Gamma \).
In addition $[\Gamma]^n_i(k) = \text{key}^{|H(]\Gamma_i^n]}$ by definition of $[\Gamma]^n_i$. Hence $\Gamma'(k) = \text{key}^{|H(]\Gamma'\ i^n]}$.

Therefore by rule TENC, we have

$$\Gamma' \vdash [M_i^n] \sim [N_i^n]: \llbracket c \rrbracket \rightarrow \{[M_i^n] \sim [N_i^n] = [c]_I^n\}.$$

- **TPAIR, TENC, TENC L, TAENC, TAENCL, TSCN H, TSCN L, THASHL, TOR:** Similarly to the TENC case, the claim is proved directly by applying the induction hypothesis to the type judgement appearing in the conditions of the last rule in these cases.

- **TVAR:** then $M = N = x$ for some $x \in X$, and

$$\Pi' = \Gamma \vdash [M \sim N : [\llbracket c \rrbracket_i = [c]_I^n].$$

We have $[M_i^n] = [N_i^n] = x_i$.

Since $\Gamma' \in \text{branches}([\Gamma]^n_i)$, we have $\Gamma'(x_i) \in \text{branches}([T]^n)$.

Hence by rule TVAR, $\Gamma' \vdash x_i \sim x_i : [T^n] \rightarrow \emptyset.$ Therefore, by rule TOR, we have

$$\Gamma' \vdash x_i \sim x_i : [T^n] \rightarrow \emptyset$$

which proves the claim.

- **TLR’** (the TLRL’ case is similar): then there exist $m, p, l$ such that $T = l$, and

$$\Pi' = \Gamma \vdash M \sim N : [\llbracket a \rrbracket_m : [\llbracket a \rrbracket_p] \rightarrow c].$$

Let us distinguish the case where $a$ is 1 from the case where $a$ is $\infty$.

If $a$ is 1: by applying the induction hypothesis to $\Pi'$, since $[\llbracket a \rrbracket_m : [\llbracket a \rrbracket_p]]_n = [\llbracket a \rrbracket_m : [\llbracket a \rrbracket_p]]_n$, we have

$$\Gamma' \vdash [M_i^n] \sim [N_i^n] : [\llbracket a \rrbracket_m : [\llbracket a \rrbracket_p] \rightarrow [c]_I^n].$$

Thus by rule TLR', we have

$$\Gamma' \vdash M_i^n \sim N_i^n : l \rightarrow [c]_I^n.$$

If $a$ is $\infty$: by applying the induction hypothesis to $\Pi'$, since $[\llbracket a \rrbracket_m : [\llbracket a \rrbracket_p]]_n = \bigvee_{1 \leq j \leq n} [\llbracket a \rrbracket_m : [\llbracket a \rrbracket_p]]_j$, we have

$$\Gamma' \vdash [M_i^n] \sim [N_i^n] : \bigvee_{1 \leq j \leq n} [\llbracket a \rrbracket_m : [\llbracket a \rrbracket_p] \rightarrow [c]_I^n].$$

Thus, by Lemma B.5, there exists $j \in [1, n]$ and a proof $\Pi''$ of

$$\Gamma' \vdash [M_i^n] \sim [N_i^n] : [\llbracket a \rrbracket_m : [\llbracket a \rrbracket_p] \rightarrow [c]_I^n].$$

Thus, by rule TLR',

$$\Gamma' \vdash M_i^n \sim N_i^n : l \rightarrow [c]_I^n,$$

which proves the claim.

- **TLRVAR:** this case is similar to the TLR case, but only the case where $a$ is 1 is possible.

- **TSUB:** then there exists $T' <: T$ such that

$$\Pi' = \Gamma \vdash M \sim N : T' \rightarrow c \quad \Gamma' \vdash T' <: T.$$

By applying the induction hypothesis to $\Pi'$, we have

$$\Gamma' \vdash M_i^n \sim N_i^n : [T']_n \rightarrow [c]_I^n.$$

Since it is clear by induction on the subtyping rules that $T' <: T$ implies that $[T']_n <: [T]_n$, rule TSub can be applied and proves the claim.

- **TLR**<sup>∞</sup>: then $M = m, N = p, c = \emptyset$, and $T = [\llbracket a \rrbracket_m : [\llbracket a \rrbracket_p]_r]$ for some $m, p \in N, c = \emptyset$, and

$$\Pi = \Gamma(m) = [\llbracket a \rrbracket_m] \quad \Gamma(p) = [\llbracket a \rrbracket_p]_r.$$

Hence by rule TSub, we have

$$\Gamma \vdash m \sim p : [\llbracket a \rrbracket_m : [\llbracket a \rrbracket_p]_r] \rightarrow \emptyset.$$
We have by definition \( [M]_i^v = m_i \) and \( [N]_i^v = p_i \), and \( [\Gamma]_i^v(m_i) = [\Gamma]_i^v(p_i) = r_{m_i}^v, r_{p_i}^v \). Thus \( \Gamma' = r_{m_i}^v, \Gamma'' = r_{p_i}^v \). Hence by rule TLR, we have \( \Gamma' = \{M\} \vdash [N]_i^v \\vdash [r_{m_i}^v, r_{p_i}^v] \rightarrow \emptyset \).

In addition, \( \bigwedge_{1 \leq j \leq n} \Gamma' \vdash \{r_{m_j}^v, r_{p_j}^v\} \rightarrow \emptyset \). Therefore, by applying rule TO\( \neg \), we have

\[
\Gamma' \vdash [M]_i^v \sim [N]_i^v \vdash \Bigl[ [r_{m_j}^v, r_{p_j}^v] \Bigr]_i^v \rightarrow \emptyset
\]

which proves the claim.

\[
\square
\]

**Lemma B.31 (Typing destructors with replicated names).** For all \( \Gamma, d, x, T \), if

\[
\Gamma \vdash d(x) : T
\]

then for all \( i, n \in \mathbb{N} \) such that \( 1 \leq i \leq n \),

\[
[\Gamma]_i^n \vdash d(x_i) : [T]^n
\]

**Proof.** Immediate by examining the typing rules for destructors.

\[
\square
\]

**Lemma B.32 (Branches and Expansion).**

- For all \( T \),

\[
\bigcup_{T' \in \text{branches}(T)} \text{branches}([T'^n]) = \text{branches}([T^n])
\]

- For all \( \Gamma \) for all \( i, n \in \mathbb{N} \),

\[
\bigcup_{i \in \text{branches}(\Gamma)} \text{branches}([\Gamma'^n]) = \text{branches}([\Gamma]^n)
\]

**Proof.** The first point is proved by induction on \( T \).

If \( T = T' \lor T'' \) for some \( T', T'' \), then

\[
\text{branches}([T^n]) = \text{branches}([T'^n]) \cup \text{branches}([T''^n]) = (\bigcup_{T' \in \text{branches}(T')} \text{branches}([T'^n])) \cup (\bigcup_{T'' \in \text{branches}(T'')} \text{branches}([T''^n]))
\]

by the induction hypothesis. Since \( \text{branches}(T) = \text{branches}(T') \cup \text{branches}(T'') \), this proves the claim.

Otherwise, \( \text{branches}(T) = \{T\} \) and the claim trivially holds.

The second point directly follows from the first point, using the definition of \( [\Gamma]^n \).

\[
\square
\]

**Lemma B.33 (Typing processes in all branches).** For all \( P, Q, \Gamma, \{C_T\}_{T' \in \text{branches}(\Gamma)} \), if

\[
\forall \Gamma' \in \text{branches}(\Gamma). \quad \Gamma' \vdash P \sim Q \rightarrow C_{\Gamma'}
\]

then

\[
\Gamma \vdash P \sim Q \rightarrow \bigcup_{\Gamma' \in \text{branches}(\Gamma)} C_{\Gamma'}.
\]

Consequently if for some \( C, C_{\Gamma'} \subseteq C \) for all \( \Gamma' \), then there exists \( C' \subseteq C \) such that

\[
\Gamma \vdash P \sim Q \rightarrow C'.
\]

**Proof.** The first point is easily proved by successive applications of rule POR. The second point is a direct consequence of the first point.

\[
\square
\]

**Lemma B.34 (Expansion and Union).**

- For all \( C, C' \), such that \( \forall (\cdot, \cdot, \Gamma) \in C \cup C' \), \( \text{branches}(\Gamma) = \{\Gamma\}, i.e. such that \( \Gamma \) does not contain union types, and \( \text{names}(c) \subseteq \text{dom}(\Gamma) \cup \mathcal{F}N \), and \( \Gamma \) only nonce types with names from \( N_0 \) (i.e. unindexed names), we have

\[
[CU_c C']_i^n = [C]_i^n [U_c]_i^n [C']_i^n
\]

- For all \( C, c, \Gamma \), such that \( \text{names}(c) \subseteq \text{dom}(\Gamma) \) and \( \forall (\cdot, \cdot, \Gamma') \in C \), \( \Gamma_{N, c} \subseteq \Gamma' \), we have

\[
[C \cup_U c]_i^n = [C]_i^n [U_c]_i^n [c]_i^n
\]
Theorem B.35 (Typing processes with expanded types). For all $\Gamma, P, Q$ and $C$, if

$$ \Gamma \vdash P \sim Q \rightarrow C $$

then for all $i, n \in \mathbb{N}$ such that $1 \leq i \leq n$, there exists $C' \subseteq \{ C \}^n_i$ such that

$$ \{ \Gamma \}^n_i \vdash \{ P \}^n_i \sim \{ Q \}^n_i \rightarrow \{ C \}^n_i $$

Proof. We prove this theorem by induction on the derivation $\Pi$ of $\Gamma \vdash P \sim Q \rightarrow C$. We distinguish several cases for the last rule applied in this derivation.

- **PZero:** Then $P = Q = \{ P \}^n_i = \{ Q \}^n_i = \emptyset$, and $C = \{ (\emptyset, \Gamma) \}$. Hence

  $$ \{ C \}^n_i = \{ (\emptyset, \Gamma) \} \lor (\Gamma' \in \text{branches}(\{ \Gamma \}^n_i)) $$

  Thus, by applying rule PO or as many times as necessary to split $\{ \Gamma \}^n_i$ into all of its branches, followed by rule PZero, we have $\{ \Gamma \}^n_i \vdash \{ P \}^n_i \sim \{ Q \}^n_i \rightarrow \{ C \}^n_i$. 

The first point follows from the definition of $\{ \cdot \}^n_i$ and $\cup_\times$. Indeed, if $C, C'$ are as assumed in the claim, we have:

$$ [C \cup_\times C']^{n_i} = \{(c, \Gamma') \mid (c, \Gamma) \in [C]^{n_i} \land (c', \Gamma') \in [C']^{n_i} \land \Gamma' \in \text{branches}(\{ \Gamma \}^n_i)\} $$

- **1**

  $$ = \{(c, \Gamma) \cup_\times (c', \Gamma') \mid (c, \Gamma) \in [C]^{n_i} \land (c', \Gamma') \in [C']^{n_i} \land \Gamma' \in \text{branches}(\{ \Gamma \}^n_i)\} $$

  $$ = \{(c, \Gamma) \cup_\times (c', \Gamma') \mid (c, \Gamma) \in [C]^{n_i} \land (c', \Gamma') \in [C']^{n_i} \land \Gamma' \in \text{branches}(\{ \Gamma \}^n_i)\} $$

  $$ = \{(c, \Gamma) \cup_\times (c', \Gamma') \mid (c, \Gamma) \in [C]^{n_i} \land (c', \Gamma') \in [C']^{n_i} \land \Gamma' \in \text{branches}(\{ \Gamma \}^n_i)\} $$

  Hence, $\{ C \}^{n_i} \cup_\times \{ C' \}^{n_i}$ is indeed equal to $\{ C \}^{n_i} \cup_\times \{ C' \}^{n_i}$.

This last step is proved by directly showing both inclusions.

On the other hand, we have:

$$ [C]^{n_i} \cup_\times \{ C' \}^{n_i} = \{(c, \Gamma) \in [C]^{n_i} \land (c', \Gamma') \in [C']^{n_i} \land \Gamma' \in \text{branches}(\{ \Gamma \}^n_i)\} $$

- **1**

  $$ = \{(c, \Gamma) \cup_\times (c', \Gamma') \mid (c, \Gamma) \in [C]^{n_i} \land (c', \Gamma') \in [C']^{n_i} \land \Gamma' \in \text{branches}(\{ \Gamma \}^n_i)\} $$

  Hence, $\{ C \}^{n_i} \cup_\times \{ C' \}^{n_i}$ is indeed equal to $\{ C \}^{n_i} \cup_\times \{ C' \}^{n_i}$.

This last step comes from the fact that if $(c_1, \Gamma_1) \in C$ and $(c_2, \Gamma_2) \in C'$, then by assumption $\Gamma_1$ and $\Gamma_2$ do not contain union types. This implies that if $\Gamma \in \text{branches}(\{ C \}^{n_i})$ and $\Gamma' \in \text{branches}(\{ C' \}^{n_i})$ are compatible, then $\Gamma_1$ and $\Gamma_2$ are compatible. Indeed, let $x \in \text{dom}(\Gamma_1) \cap \text{dom}(\Gamma_2)$. Hence $x_1 \in \text{dom}(\Gamma_1) \cap \text{dom}(\Gamma_2)$, and since they are compatible, $\Gamma_1(x_1) = \Gamma_2(x_1)$. That is to say that there exists $\Gamma \in \text{branches}(\{ C_1 \}^{n_i}) \cap \text{branches}(\{ C_2 \}^{n_i})$.

If $\Gamma_1(x) = \{ t_1 \}^{m_1}_p \cup_\tau \{ t_2 \}^{m_2}_p$ (for some $m, p, l, i'$), then $\Gamma_2(x) = \{ t_3 \}^{m_3}_p \cup_\tau \{ t_4 \}^{m_4}_p$, and thus there exists $j \in [1, n]$ such that $T = \{ t_j \}^{m_j}_p \cup_\tau \{ t_{j'} \}^{m_{j'}}_p$. Hence, $\{ t_1 \}^{m_1}_p \cup_\tau \{ t_2 \}^{m_2}_p$ is a branch of $\{ t_3 \}^{m_3}_p \cup_\tau \{ t_4 \}^{m_4}_p$. Because of the definition of $\{ \cdot \}^{n_i}$, and since $\Gamma_1(x)$ is not a union type (by assumption), this implies that $\Gamma_2(x) = \{ t_3 \}^{m_3}_p \cup_\tau \{ t_4 \}^{m_4}_p$, and therefore $\Gamma_1(x) = \Gamma_2(x)$.

If $\Gamma_1(x)$ is not of the form $\{ t_1 \}^{m_1}_p \cup_\tau \{ t_2 \}^{m_2}_p$ (for some $m, p, l, i'$), then neither is $\Gamma_2(x)$ (by contraposition, following the same reasoning as in the previous case). $\Gamma_1(x)$ and $\Gamma_2(x)$ are not of the form $\tau \lor \tau'$ either, by assumption. Therefore, neither $\{ (\emptyset, \Gamma) \}^{n_i}$ nor $\{ (\emptyset, \Gamma) \}^{n_i}$ are union types (from the definition of $\{ \cdot \}^{n_i}$). This implies that $\tau = \{ (\emptyset, \Gamma) \}^{n_i} = \{ (\emptyset, \Gamma) \}^{n_i}$, which implies $\Gamma_1(x) = \Gamma_2(x)$.

In both cases $\Gamma_1(x) = \Gamma_2(x)$, and $\Gamma_1, \Gamma_2$ are therefore compatible.

Hence $\{ C \}^{n_i} \cup_\times \{ C' \}^{n_i} = \{ C \cup_\times \cup_\times C' \}^{n_i}$, which proves the claim.

The second point directly follows from the definition of $\{ \cdot \}^{n_i}$ and $\cup_\times$. Indeed, for all $C, c, \Gamma$ satisfying the assumptions, we have:

$$ [C \cup_\times \{ C' \}^{n_i}] = \{(c', \Gamma') \mid (c, \Gamma) \in [C]^{n_i} \land (c', \Gamma') \in [C']^{n_i} \land \Gamma' \in \text{branches}(\{ \Gamma \}^n_i)\} $$

- **1**

  $$ = \{(c, \Gamma) \cup_\times (c', \Gamma') \mid (c, \Gamma) \in [C]^{n_i} \land (c', \Gamma') \in [C']^{n_i} \land \Gamma' \in \text{branches}(\{ \Gamma \}^n_i)\} $$

  Hence, $\{ C \}^{n_i} \cup_\times \{ C' \}^{n_i}$ is indeed equal to $\{ C \}^{n_i} \cup_\times \{ C' \}^{n_i}$.
• POut: then $P = \text{out}(M).P', Q = \text{out}(N).Q'$ for some messages $M$, $N$ and some processes $P', Q'$, and

$$\Pi' \quad \Pi''$$

$$\Pi = \frac{\Gamma \vdash P' \sim Q' \rightarrow C'}{\Gamma \vdash P \sim Q \rightarrow C = C' \cup \{(c')\}}.$$  

By applying the induction hypothesis to $\Pi'$, there exists $C'' \subseteq [C]' \vec{P}$ and a proof $\Pi''$ of $[\Gamma]_{\vec{P}}^{n} \vdash [P']_{\vec{P}}^{n} \sim [Q']_{\vec{P}}^{n} \rightarrow C''$. Hence, by Lemma B.7, for all $\Gamma' \in \text{branches}([\Gamma]_{\vec{P}}^{n})$, there exist $\Gamma C \subseteq C''$ and a proof $\Pi'_{\Gamma}$ of $\Gamma' \vdash [P']_{\vec{P}}^{n} \sim [Q']_{\vec{P}}^{n} \rightarrow \Gamma C$. Moreover, by Lemma B.30, for all $\Gamma' \in \text{branches}([\Gamma]_{\vec{P}}^{n})$, there exists a proof $\Pi'_{\Gamma'}$, $\Gamma' = [M]_{\vec{P}}^{n} \vdash [N]_{\vec{P}}^{n} \rightarrow \Gamma' C$.  

In addition, $[P]_{\vec{P}}^{n} = \text{out}(M).P'_{\vec{P}}^{n} = \text{out}([M]_{\vec{P}}^{n}).P'_{\vec{P}}^{n}$. Similarly, $[Q]_{\vec{P}}^{n} = \text{out}([N]_{\vec{P}}^{n}).Q'_{\vec{P}}^{n}$. Therefore, using $\Pi'_{\Gamma}$, $\Pi'_{\Gamma'}$, and rule POut, we have for all $\Gamma' \in \text{branches}([\Gamma]_{\vec{P}}^{n})$ that $\Gamma' \vdash [P]_{\vec{P}}^{n} \sim [Q]_{\vec{P}}^{n} \rightarrow \Gamma C \cup \{c\} \subseteq C'' \cup \{c_{1}\}$. Thus by Lemma B.33, there exists $C_{1} \subseteq C'' \cup \{c\}$ such that $[\Gamma]_{\vec{P}}^{n} \vdash [P]_{\vec{P}}^{n} \sim [Q]_{\vec{P}}^{n} \rightarrow C_{1}$.  

Finally, $[C]_{\vec{P}}^{n} = [C'] \cup \{c\} = [C'] \cup \{c_{1}\}$ (by Lemma B.34, whose conditions are satisfied, by Lemma B.12). Hence $C'' \cup \{c_{1}\} \subseteq [C]_{\vec{P}}^{n}$, which proves the claim.

• PIn: then $P = \text{in}(x).P', Q = \text{in}(x).Q'$ for some variable $x$ and some processes $P', Q'$, and

$$\Pi' \quad \Pi''$$

$$\Pi = \frac{\Gamma, x : \text{LL} \vdash P' \sim Q' \rightarrow C}{\Gamma \vdash P \sim Q \rightarrow C}.$$  

Since $[\Gamma, x : \text{LL}]^{n} = [\Gamma]^{n}, x_{1} : \text{LL} \vdash [P']_{\vec{P}}^{n} \sim [Q']_{\vec{P}}^{n}$, by applying the induction hypothesis to $\Pi'$, there exists $C' \subseteq [C]' \vec{P}$ and a proof $\Pi''$ of $[\Gamma, x_{1} : \text{LL}]^{n} \vdash [P']_{\vec{P}}^{n,x_{1}} \sim [Q']_{\vec{P}}^{n,x_{1}} \rightarrow C'$. In addition, $[P]_{\vec{P}}^{n} = [\text{in}(x).P']_{\vec{P}}^{n} = [\text{in}(x_{1}).P']_{\vec{P}}^{n} = [\text{in}(x_{1}).[P']_{\vec{P}}^{n,x_{1}}]$; similarly, $[Q]_{\vec{P}}^{n} = [\text{in}(x_{1}).[Q']_{\vec{P}}^{n,x_{1}}]$. Therefore, using $\Pi''$ and rule PIn, we have $[\Gamma]_{\vec{P}}^{n} \vdash [P]_{\vec{P}}^{n} \sim [Q]_{\vec{P}}^{n} \rightarrow C' \subseteq [C]_{\vec{P}}^{n}$.

• PNew: then $P = \text{new } m : \tau_{m}^{a} . P', Q = \text{new } m : \tau_{m}^{a} . Q'$ for some processes $P', Q'$, and

$$\Pi' \quad \Pi''$$

$$\Pi = \frac{\Gamma, m : \tau_{m}^{a} \vdash P' \sim Q' \rightarrow C}{\Gamma \vdash P \sim Q \rightarrow C}.$$  

- If $a = 1$:  

Since $[\Gamma, m : \tau_{m}^{1}]_{i}^{n} = [\Gamma]_{\vec{P}}^{n}, m : \tau_{m}^{1}$, by applying the induction hypothesis to $\Pi'$, there exists $C' \subseteq [C]' \vec{P}$ and a proof $\Pi''$ of $[\Gamma]_{\vec{P}}^{n, m : \tau_{m}^{1}} \vdash [P']_{\vec{P}}^{n, m : \tau_{m}^{1}} \rightarrow C'$.  

In addition $[P]_{\vec{P}}^{n} = [\text{new } m . P']_{\vec{P}}^{n} = [\text{new } m : \tau_{m}^{1} . [P']_{\vec{P}}^{n, m : \tau_{m}^{1}}]$; and similarly for $Q$. Therefore, using $\Pi''$ and rule PNew, we have $[\Gamma]_{\vec{P}}^{n} \vdash [P]_{\vec{P}}^{n} \sim [Q]_{\vec{P}}^{n} \rightarrow C' \subseteq [C]_{\vec{P}}^{n}$.

- If $a = \infty$:  

Since $[\Gamma, m : \tau_{m}^{\infty}]_{i}^{n} = [\Gamma]_{\vec{P}}^{n}, m_{i} : \tau_{m_{i}}^{1}$, by applying the induction hypothesis to $\Pi'$, there exists $C' \subseteq [C]' \vec{P}$ and a proof $\Pi''$ of $[\Gamma]_{\vec{P}}^{n, m_{i} : \tau_{m_{i}}^{1}} \vdash [P']_{\vec{P}}^{n, m_{i} : \tau_{m_{i}}^{1}} \rightarrow C'$.  

In addition $[P]_{\vec{P}}^{n} = [\text{new } m . P']_{\vec{P}}^{n} = [\text{new } m_{i} : \tau_{m_{i}}^{1} . [P']_{\vec{P}}^{n, m_{i} : \tau_{m_{i}}^{1}}]$; and similarly for $Q$. Therefore, using $\Pi''$ and rule PNew, we have $[\Gamma]_{\vec{P}}^{n} \vdash [P]_{\vec{P}}^{n} \sim [Q]_{\vec{P}}^{n} \rightarrow C' \subseteq [C]_{\vec{P}}^{n}$.

• PPar: then $P = \text{P} \mid P'', Q = \text{Q} \mid Q''$ for some processes $P', P'', Q', Q''$, and

$$\Pi' \quad \Pi''$$

$$\Pi = \frac{\Gamma \vdash P' \sim Q' \rightarrow C'}{\Gamma \vdash P \sim Q \rightarrow C = C' \cup \{c''\}}.$$
By applying the induction hypothesis to $\Pi'$, there exists $C'' \subseteq [C']_T$ and a proof $\Pi''$ of $[\Gamma]_T \vdash [P']_T \sim [Q']_T \rightarrow C'''$. Similarly, by applying the induction hypothesis to $\Pi''$, there exists $C''' \subseteq [C'']_T$ and a proof $\Pi'''$ of $[\Gamma]_T \vdash [P''']_T \sim [Q''']_T \rightarrow C'''$. In addition, $[P]_T = [P'']_T = [P''']_T$. Similarly, $[Q]_T = [Q'']_T = [Q''']_T$. Finally, $[C]_T = [C'']_T \cup [C''']_T$, by Lemma B.34 (using Lemma B.9 to ensure the condition that the environments do not contain union types).

Therefore, using $\Pi''''$, $\Pi'''$ and rule PPAR, we have $[\Gamma]_T \vdash [P]_T \sim [Q]_T \rightarrow C'''' \subseteq [C]_T$.

- **POB**: then $\Gamma = \Gamma', x : T \lor T'$ for some $\Gamma'$, some $x \in X$ and some types $T, T'$, and

$$
\Pi_T = \begin{cases}
\Gamma', x : T \vdash P \sim Q \rightarrow C' \\
\Gamma' \vdash P \sim Q \rightarrow C'''
\end{cases}.
$$

By applying the induction hypothesis to $\Pi_T$, there exist $C_1 \subseteq [C']_T$ and a proof $\Pi_1$ of $[\Gamma']_T \vdash [P]_{T,i} \sim [Q]_{T,i} \rightarrow C_1$. Similarly, with $\Pi_T$, there exist $C_2 \subseteq [C''']_T$ and a proof $\Pi_2$ of $[\Gamma']_T \vdash [P']_{T,i} \sim [Q']_{T,i} \rightarrow C_2$.

In addition $[P]_T = [P']_T \vdash [P']_T \sim [P]_T$, and similarly for $Q$.

Thus by rule POB, we have

$$
[\Gamma']_T \vdash x : T \lor T' \vdash x : T \vdash \Pi_T '. \vdash \Pi_T '''. \vdash [\Gamma']_T \vdash [\Pi']_T \vdash [\Pi''']_T \vdash [C']_T \vdash [C''']_T.
$$

Since $[\Gamma']_T = [\Gamma']_T, x : T \lor T' = [\Gamma']_T \vdash x : T \vdash [Q']_T \vdash [Q''']_T$, this proves the claim in this case.

- **PLET**: then $P = \text{let } x = d(y) \text{ in } P' \text{ else } P''$, $Q = \text{let } x = d(y) \text{ in } Q' \text{ else } Q''$ for some variable $x$ and some processes $P', Q', P'', Q''$.

$$
\Pi_d = \begin{cases}
\Gamma, x : T \vdash P \sim Q \rightarrow C' \\
\Gamma \vdash P' \sim Q' \rightarrow C'' \\
\Gamma \vdash P'' \sim Q'' \rightarrow C'''
\end{cases}.
$$

Since $[\Gamma, x : T]_T = [\Gamma]_T, x : T$, by applying the induction hypothesis to $\Pi'$, there exist $C'' \subseteq [C']_T$ and a proof $\Pi''$ of $[\Gamma']_T, x : T \vdash [P']_T \sim [Q']_T \rightarrow C'$. Similarly, there exist $C''' \subseteq [C''']_T$ and a proof $\Pi'''$ of $[\Gamma']_T \vdash [P''']_T \sim [Q''']_T \rightarrow C'''$.

By Lemma B.31 applied to $\Pi_d$, we also have

$$
[\Gamma]_T \vdash d(y) : T.
$$

In addition, $[P]_T = \text{let } x = d(y) \text{ in } P' \text{ else } P''$, $[Q]_T = \text{let } x = d(y) \text{ in } Q' \text{ else } Q''$.

Similarly, $[Q']_T = \text{let } x = d(y) \text{ in } [Q']_T \vdash x : T \vdash [Q'']_T \vdash x : T$. Therefore, using $\Pi''$ and rule PLET, we have $[\Gamma]_T \vdash [P]_T \sim [Q]_T \rightarrow C'''' \subseteq [C]_T$.

- **PLET/LR**: then $P = \text{let } x = d(y) \text{ in } P' \text{ else } P''$, $Q = \text{let } x = d(y) \text{ in } Q' \text{ else } Q''$ for some variable $x \in X$ and some processes $P', Q', P'', Q''$.

$$
\Pi = \begin{cases}
\Gamma(y) = \begin{bmatrix}
\lambda_m^a \vdash T_p^a \vdash C
\end{bmatrix} \\
\Gamma \vdash P' \sim Q' \rightarrow C
\end{cases}.
$$

for some $m, p$.

By applying the induction hypothesis to $\Pi'$, there exists $C' \subseteq [C]_T$ and a proof $\Pi''$ of $[\Gamma]_T \vdash [P']_T \sim [Q']_T \rightarrow C'$. We have $[P]_T = \text{let } x = d(y) \text{ in } P' \text{ else } P''$, $[Q]_T = \text{let } x = d(y) \text{ in } [P']_T \text{ else } [P'']_T$. Similarly, $[Q']_T = \text{let } x = d(y) \text{ in } [Q']_T \text{ else } [Q'']_T$.

We distinguish two cases, depending on whether the types in the refinement $[\tau_m^a \vdash \tau_p^a]$ are finite non-types or infinite non-types, i.e. whether $a$ is 1 or 0.

1. **If $a = 1$**: Then by definition of $[\Gamma]_T$, we have $[\Gamma]_T(y) = [\tau_m^1 \vdash \tau_p^1]$.

Thus by definition, there exists $j \in \{1, n\}$, such that $\Gamma_j(y) = [\tau_m^j \vdash \tau_p^1]$. Using $\Pi''$ and Lemma B.21, there exist $C_j \subseteq [C]_T$ and a derivation $\Pi''_j$ of $\Gamma_j \vdash [P''']_T \sim [Q''']_T \rightarrow C_j$. Therefore, by rule PLET/LR, we have $[\Gamma_j]_T \vdash [P''']_T \sim [Q''']_T \rightarrow C''''$. Therefore, by rule PLET/LR, we have $[\Gamma]_T \vdash [P]_T \sim [Q]_T \rightarrow C''''$.
By applying the induction hypothesis to $\Pi'$, there exists $C''' \subseteq \{ C' \}_{t, i}$ and a proof $\Pi''$ of $[ \Gamma ]^p_i \vdash [ P' ]^p_j \sim [ Q' ]^p_j \rightarrow C'''$. Similarly, there exists $C'''' \subseteq \{ C' \}_{t, i}$ and a proof $\Pi'''$ of $[ \Gamma ]^p_i \vdash [ P'' ]^p_j \sim [ Q'' ]^p_j \rightarrow C''''$.

Hence, by Lemma B.7, for all $\Gamma' \in \text{branches}(\Gamma^p_i)$, there exist $C_{\Gamma'} \subseteq C'''$ and a proof $\Pi_{1, \Gamma'}$ of $\Gamma' \vdash [ P' ]^p_j \sim [ Q' ]^p_j \rightarrow C_{\Gamma'}$; as well as $C_{\Gamma'} \subseteq C''''$ and a proof $\Pi_{2, \Gamma'}$ of $\Gamma' \vdash [ P'' ]^p_j \sim [ Q'' ]^p_j \rightarrow C_{\Gamma'}$.

Moreover, by Lemma B.30, for all $\Gamma' \in \text{branches}(\Gamma^p_i)$, there exists a proof $\Pi_{1, \Gamma'}$ of $\Gamma' \vdash [ M ]^p_j \sim [ N ]^p_j : LL \rightarrow [ c ]^p_j$. Similarly, there exists a proof $\Pi_{2, \Gamma'}$ of $\Gamma' \vdash [ M ]^p_j \sim [ N ]^p_j : LL \rightarrow [ c ]^p_j$.

In addition, $[ P ]^p_j = [ \Gamma = M = t $ then $P' $ else $P'' ]^p_j = [ \Gamma = M = t $ then $[ P' ]^p_j $ else $[ P'' ]^p_j $ Similarly, $[ Q ]^p_j = [ \Gamma = N = t $ then $Q' $ else $Q'' ]^p_j $.

Therefore, using $\Pi_{1, \Gamma'}, \Pi_{2, \Gamma'}, \Pi_{1, \Gamma'}, \Pi_{2, \Gamma'}$, and rule PlfP, we have

$$\Gamma' \vdash [ P ]^p_j \sim [ Q ]^p_j \rightarrow (C_{\Gamma'} \cup C_{\Gamma'}) \cup (c^p_i \cup c^p_j) \subseteq (C''' \cup C''') \cup (c^p_i \cup c^p_j)$$

Thus by Lemma B.33, there exists $C_1 \subseteq (C''' \cup C''') \cup (c^p_i \cup c^p_j)$ such that

$$[ \Gamma ]^p_i \vdash [ P ]^p_j \sim [ Q ]^p_j \rightarrow C_1.$$
\[\text{PfrLR: then } P = \text{if } M_1 = M_2 \text{ then } P_1 \text{ else } P_2, Q = \text{if } N_1 = N_2 \text{ then } Q_1 \text{ else } Q_2 \text{ for some messages } M_1, M_2, N_1, N_2, \text{ and some processes } P_1, Q_1, P_2, Q_2, \text{ and there exist } m, p, m', p' \text{ such that}\]

\[\begin{array}{c}
\Pi_1 \\
\hline
\Gamma \vdash M_1 \sim N_1 : [m_1^1 : \tau_p^{a_1}] \rightarrow c
\end{array}\]

\[\begin{array}{c}
\Pi_2 \\
\hline
\Gamma \vdash M_2 \sim N_2 : [m_2^1 : \tau_p^{a_1}] \rightarrow c'
\end{array}\]

\[\Pi = \frac{\Gamma \vdash \Pi \rightarrow Q' \rightarrow C}{\Gamma \vdash P_1 \sim Q_1 \rightarrow C}\]

\[b = (m_1^1 \equiv m_2^1) \quad b' = (\tau_p^{a_1} \equiv \tau_p^{a_1})\]

By applying the induction hypothesis to \(\Pi'\), there exist \(C' \subseteq [C]^\eta\) \text{ and a proof } \Pi'' \text{ of } [\Gamma]^\eta \vdash [Q']^\eta \rightarrow [Q']^\eta \rightarrow C'.

\[\begin{array}{c}
\Pi' \\
\hline
\Gamma \vdash P_1 \sim Q_1 \rightarrow C
\end{array}\]

\[\begin{array}{c}
\Pi_1 \\
\hline
\Gamma \vdash M \sim N : \text{LL} \rightarrow c
\end{array}\]

\[\Pi_2 \\
\hline
\Gamma \vdash M' \sim N' : \text{HH} \rightarrow c'
\end{array}\]

By applying the induction hypothesis to \(\Pi'\), there exists \(C' \subseteq [C]^\eta\) \text{ and a proof } \Pi'' \text{ of } [\Gamma]^\eta \vdash [P''] \rightarrow [Q''] \rightarrow C'.

\[\begin{array}{c}
\Pi' \\
\hline
\Gamma \vdash P'' \sim Q'' \rightarrow C
\end{array}\]

\[\begin{array}{c}
\Pi_1 \\
\hline
\Gamma \vdash M \sim N : \text{TT} \rightarrow c
\end{array}\]

\[\Pi_2 \\
\hline
\Gamma \vdash M' \sim N' : [T]^\eta \rightarrow c'
\end{array}\]

\[\Pi = \frac{\Gamma \vdash P'' \sim Q'' \rightarrow C}{\Gamma \vdash P \sim Q \rightarrow C}\]

Thus by Lemma B.33, there exists \(C_1 \subseteq [C]^\eta\) such that

\[\begin{array}{c}
[\Gamma]^\eta \vdash [P']^\eta \sim [Q']^\eta \rightarrow C_1,
\end{array}\]

which proves the claim.

\[\text{PfrS: then } P = \text{if } M = M' \text{ then } P' \text{ else } P'' \text{, } Q = \text{if } N = N' \text{ then } Q' \text{ else } Q'' \text{ for some messages } M, N, M', N', \text{ and some processes } P', Q', P'', Q'', \text{ and}\]

\[\begin{array}{c}
\Pi' \\
\hline
\Gamma \vdash P'' \sim Q'' \rightarrow C
\end{array}\]

\[\begin{array}{c}
\Pi_1 \\
\hline
\Gamma \vdash M \sim N : \text{LL} \rightarrow c
\end{array}\]

\[\Pi_2 \\
\hline
\Gamma \vdash M' \sim N' : \text{HH} \rightarrow c'
\end{array}\]

By applying the induction hypothesis to \(\Pi'\), there exists \(C' \subseteq [C]^\eta\) \text{ and a proof } \Pi'' \text{ of } [\Gamma]^\eta \vdash [P''] \rightarrow [Q''] \rightarrow C'.

\[\begin{array}{c}
\Pi' \\
\hline
\Gamma \vdash P'' \sim Q'' \rightarrow C
\end{array}\]

\[\begin{array}{c}
\Pi_1 \\
\hline
\Gamma \vdash M \sim N : T \rightarrow T' \rightarrow c
\end{array}\]

\[\Pi_2 \\
\hline
\Gamma \vdash M' \sim N' : [T]^\eta \rightarrow [T']^\eta \rightarrow \text{HH} \rightarrow c'
\end{array}\]

Thus by Lemma B.33, there exists \(C_1 \subseteq [C]^\eta\) such that

\[\begin{array}{c}
[\Gamma]^\eta \vdash [P']^\eta \sim [Q']^\eta \rightarrow C_1,
\end{array}\]

which proves the claim.

\[\text{PfrL: then } P = \text{if } M = M' \text{ then } P' \text{ else } P'' \text{, } Q = \text{if } N = N' \text{ then } Q' \text{ else } Q'' \text{ for some messages } M, N, M', N', \text{ and some processes } P', Q', P'', Q'', \text{ and there exist types } T, T', \text{ and names } m, p, \text{ such that}\]

\[\Pi' \\
\hline
\Gamma \vdash P'' \sim Q'' \rightarrow C
\end{array}\]

\[\begin{array}{c}
\Pi_1 \\
\hline
\Gamma \vdash M \sim N : \text{TT} \rightarrow c
\end{array}\]

\[\Pi_2 \\
\hline
\Gamma \vdash M' \sim N' : [T]^\eta \rightarrow [T']^\eta \rightarrow \text{HH} \rightarrow c'
\end{array}\]

By applying the induction hypothesis to \(\Pi'\), there exist \(C' \subseteq [C]^\eta\) \text{ and a proof } \Pi'' \text{ of } [\Gamma]^\eta \vdash [P''] \rightarrow [Q''] \rightarrow C'.

\[\begin{array}{c}
\Pi' \\
\hline
\Gamma \vdash P'' \sim Q'' \rightarrow C
\end{array}\]

\[\begin{array}{c}
\Pi_1 \\
\hline
\Gamma \vdash M \sim N : \text{TT} \rightarrow c
\end{array}\]

\[\Pi_2 \\
\hline
\Gamma \vdash M' \sim N' : [T]^\eta \rightarrow [T']^\eta \rightarrow \text{HH} \rightarrow c'
\end{array}\]

Let \(\Gamma' \in \text{branches}(\Gamma\eta)\). By applying Lemma B.7 to \(\Pi''\), there exists \(C_\Gamma \subseteq C'\), such that there exists a proof \(\Pi_\Gamma\) \text{ of } \Gamma \vdash \Gamma' \rightarrow [P''] \rightarrow [Q''] \rightarrow C'.

\[\begin{array}{c}
\Pi' \\
\hline
\Gamma \vdash P'' \sim Q'' \rightarrow C
\end{array}\]

\[\begin{array}{c}
\Pi_1 \\
\hline
\Gamma \vdash M \sim N : \text{TT} \rightarrow c
\end{array}\]

\[\Pi_2 \\
\hline
\Gamma \vdash M' \sim N' : [T]^\eta \rightarrow [T']^\eta \rightarrow \text{HH} \rightarrow c'
\end{array}\]

Moreover, by Lemma B.30 applied to \(\Pi_1\), there exists a proof \(\Pi'_{\Gamma,1}\) \text{ of } \Gamma' \vdash [M] \rightarrow [N] \rightarrow [T] \rightarrow \text{HH} \rightarrow c'. Similarly, there exists a proof \(\Pi'_{\Gamma,2}\) \text{ of } \Gamma' \vdash [M'] \rightarrow [N'] \rightarrow [T'] \rightarrow \text{HH} \rightarrow c'.

Therefore, using \(\Pi_\Gamma, \Pi'_{\Gamma,1}, \Pi'_{\Gamma,2}, \text{ and rule PfrL}, \text{ we have for all } \Gamma' \in \text{branches}(\Gamma\eta)\)

\[\Gamma' \vdash [P'] \rightarrow [Q'] \rightarrow C_\Gamma\subseteq C' \subseteq [C]^\eta\]

Thus by Lemma B.33, there exists \(C_1 \subseteq [C]^\eta\) such that

\[\begin{array}{c}
[\Gamma]^\eta \vdash [P'] \sim [Q'] \rightarrow C_1,
\end{array}\]

which proves the claim.
In addition, if $M = M'$ then $P' \iff P''$ \iff $M = M'$ \iff $P = P'$. Similarly, if $N = N'$ \iff $Q = Q'$ \iff $N = N'$ \iff $Q = Q'$. We distinguish two cases.

- **If $\Delta = \emptyset$:** Then $[r_{m,1}^{l,1} : r_{p,1}^{l,1}]^n = [r_{m,1}^{l,1} : r_{p,1}^{l,1}]^n$, and using $\Pi^{\top}, \Pi^{\bot}, \Pi^{\top \bot}$, and rule $\Pi I$, we have for all $\Gamma' \in \text{branches}([\Gamma]^n)$

$$
\Gamma' \vdash [P^n_I \iff Q^n_I] \Rightarrow C_I \subseteq C' \subseteq [C]^n.
$$

Thus by Lemma B.33, there exists $C_1 \subseteq [C]^n$ such that

$$
[\Gamma]^n_I + [P^n_I \iff Q^n_I] \Rightarrow C_1.
$$

which proves the claim in this case.

- **If $\Delta = \Delta'$:** Moreover, by applying Lemma B.5 to $\Pi^{\top \bot}$, there exists a type $T' \in \text{branches}([r_{m,1}^{l,1} : r_{p,1}^{l,1}])$, such that there exists a proof $\Pi^{\top \bot}$ of $\Gamma' + [M^n_i \iff N^n_i] : T' \Rightarrow \Gamma'$. Therefore, by definition of branches, there exists $j$ such that $T'' = [r_{m,1}^{l,1} : r_{p,1}^{l,1}]$.

Hence, using $\Pi^{\top}, \Pi^{\bot}$, $\Pi^{\top \bot}$, by applying rule $\Pi I$, we have for all $\Gamma' \in \text{branches}([\Gamma]^n)$ that

$$
\Gamma' + [P^n_I \iff Q^n_I] \Rightarrow C_I \subseteq C' \subseteq [C]^n.
$$

Thus by Lemma B.33, there exists $C_1 \subseteq [C]^n$ such that

$$
[\Gamma]^n_I + [P^n_I \iff Q^n_I] \Rightarrow C_1
$$

which proves the claim in this case.

**PflLR:** then $P = \cases{M_1 = M_2 \text{ or } P_I \text{ else } P \cases{N_1 = N_2 \text{ or } Q_I \text{ else } Q}}$ for some messages $M_1, N_1, M_2, N_2$, and some processes $P_I, Q_I, P, Q$, and there exist $m, p, l, l'$ such that

$$
\Pi_1 = \Gamma + M_1 \sim N_1 : [\Gamma^{l,1}_m^{l,1} : r_{p,1}^{l,1}] \Rightarrow \Gamma \Pi_2 = \Gamma + M_2 \sim N_2 : [\Gamma^{l,1}_m^{l,1} : r_{p,1}^{l,1}] \Rightarrow \Gamma \Pi_3 = \Gamma + P \sim Q \Rightarrow C
$$

By applying the induction hypothesis to $\Pi^{\top}$, there exist $C' \subseteq [C_1]^n \subseteq [C]^n$ and a proof $\Pi'$ of $[\Gamma]^n_I + [P_I^n \iff Q_I^n] \Rightarrow C'$. Similarly with $\Pi_1$, there exist $C'' \subseteq [C_2]^n \subseteq [C]^n$ and a proof $\Pi''$ of $[\Gamma]^n_I + [P_I^n \iff Q_I^n] \Rightarrow C''$. Hence, by Lemma B.7, for all $\Gamma' \in \text{branches}([\Gamma]^n_I)$, there exist $C_1 \subseteq C' \subseteq [C]^n$, $C_2 \subseteq C'' \subseteq [C]^n$, and proofs $\Pi^{\top}_1, \Pi^{\top}_2$, and $\Pi^{\top \bot}$ of $\Gamma' + [P_I^n \iff Q_I^n] \Rightarrow C_1$ and $\Gamma' + [P_I^n \iff Q_I^n] \Rightarrow C_2$. Moreover, by Lemma B.30 applied to $\Pi_1$, for all $\Gamma' \in \text{branches}([\Gamma]^n_I)$, there exists a proof $\Pi'_1$ of $\Gamma' + [M_1^n_I \iff N_1^n_I] : \forall_{1 \leq j \leq n} [\Gamma^{l,1}_m^{l,1} : r_{p,1}^{l,1}] \Rightarrow \Gamma'$. Similarly, there exists a proof $\Pi'_2$ of $\Gamma' + [M_2^n_I \iff N_2^n_I] : \forall_{1 \leq j \leq n} [\Gamma^{l,1}_m^{l,1} : r_{p,1}^{l,1}] \Rightarrow \Gamma'$. Let $\Gamma' \in \text{branches}([\Gamma]^n_I)$. By Lemma B.5, there exists $T' \in \text{branches}([\Gamma^{l,1}_m^{l,1} : r_{p,1}^{l,1}])$ such that there exists a proof $\Pi'_3$ of $\Gamma' + [M_2^n_I \iff N_2^n_I] : T' \Rightarrow \Gamma'$. Similarly, $\Pi'_4$ of $\Gamma' + [M_1^n_I \iff N_1^n_I] : T' \Rightarrow \Gamma'$. By definition of branches, there exists $j'$ such that $T = [\Gamma^{l,1}_m^{l,1} : r_{p,1}^{l,1}]$ and $T' = [\Gamma^{l,1}_m^{l,1} : r_{p,1}^{l,1}]$. In addition, $[P^n_I \iff Q^n_I] = \cases{M_1 = M_2 \text{ or } P_I \text{ else } P \cases{N_1 = N_2 \text{ or } Q_I \text{ else } Q}}$. Similarly, $[Q^n_I \iff P^n_I] = \cases{M_1 = M_2 \text{ or } P_I \text{ else } P \cases{N_1 = N_2 \text{ or } Q_I \text{ else } Q}}$. Therefore, using $\Pi^{\top}_1, \Pi^{\top}_2, \Pi^{\top}_3, \Pi^{\top}_4$ and rule PflLR, either $j = j'$ and we have

$$
\Gamma' + [P^n_I \iff Q^n_I] \Rightarrow C_1, \Gamma' \subseteq [C]^n
$$

or $j \neq j'$ and we have

$$
\Gamma' + [P^n_I \iff Q^n_I] \Rightarrow C_2, \Gamma' \subseteq [C]^n.
$$

This holds for any $\Gamma' \in \text{branches}([\Gamma]^n_I)$. Thus by Lemma B.33, there exists $C' \subseteq [C]^n$ such that

$$
[\Gamma]^n_I + [P^n_I \iff Q^n_I] \Rightarrow C'
$$

which proves the claim in this case.
\[
\Pi = \frac{\Gamma \vdash M_1 \sim N_1 : [t_{m_1}^a; t_p^r.a] \rightarrow \emptyset}{\Pi_1}
\]

\[
\Pi_1
\]

\[
\Gamma \vdash M_2 \sim N_2 : [t_{m_2}^{a'}; t_{p'}^{r'}.a'] \rightarrow \emptyset
\]

\[
\Pi_2
\]

\[
\Gamma \vdash P \sim Q \rightarrow C
\]

By applying the induction hypothesis to \(\Pi'\), there exist \(C' \subseteq [C]_n^p\) and a proof \(\Pi''_r\) of \([\Gamma]_j^p + [P]_j^p \sim [Q]_j^p \rightarrow C'\). Hence, by Lemma B.7, for all \(\Gamma' \in \text{branches}([\Gamma]_n^p)\), there exist \(C_{\Gamma'} \subseteq C'([\Gamma]_n^p)\), and a proofs \(\Pi'_{r,\Gamma'}\) of \(\Gamma' + [M]_j^p \sim [N]_j^p \rightarrow C_{\Gamma'}\). Moreover, by Lemma B.30 applied to \(\Pi_1\), for all \(\Gamma' \in \text{branches}([\Gamma]_n^p)\), there exists a proof \(\Pi''_{1,\Gamma'}\) of \(\Gamma' + [M_1]_j^p \sim [N_1]_j^p\) : \([t_{m_1}^a; t_{p}^r.a] \rightarrow_c [C]_j^p\). Similarly, there exists a proof \(\Pi''_{2,\Gamma'}\) of \(\Gamma' + [M_2]_j^p \sim [N_2]_j^p\) : \([t_{m_2}^{a'}; t_{p'}^{r'}.a'] \rightarrow [C']_j^p\).

We distinguish several cases, depending on \(a\) and \(a'\).

- If \(a\) and \(a'\) are both 1, then this rule is a particular case of rule \(\text{PlfLR}\), and the result is proved in a similar way.
- If \(a\) is 1 and \(a'\) is \(\infty\), then \(\Pi''\) is a particular case of rule \(\text{PlfLR}\), and the proof is similar to the symmetric one.
- If \(a\) is \(\infty\) and \(a'\) is 1, then \(\Pi''\) is a particular case of rule \(\text{PlfLR}\), and the proof is similar to the case where \(a\) is 1 and \(a'\) is \(\infty\).

\[\square\]

**Theorem B.36 (Typing n Sessions).** For all \(\Gamma, P, Q, C\), such that

\[\Gamma \vdash P \sim Q \rightarrow C\]

then for all \(n \in \mathbb{N}\), there exists \(C' \subseteq [C]_n^p\) such that

\[\left[ {\Gamma} \right]_n^p + [P]_n^p \mid \ldots \mid [Q]_n^p \rightarrow C'\]

where \(\left[ {\Gamma} \right]_n^p\) is defined as \(\cup_{1 \leq i \leq n} [\Gamma]_i^p\).

**Proof.** Let us assume \(\Gamma, P, Q, C\) such that

\[\Gamma \vdash P \sim Q \rightarrow C\]

Let \(n \in \mathbb{N}\).

Note that the union \(\cup_{1 \leq i \leq n} [\Gamma]_i^p\) is well-defined, as for \(i \neq j\), \(\text{dom}([\Gamma]_i^p) \cap \text{dom}([\Gamma]_j^p) \subseteq \mathcal{K} \cup \mathcal{N}\), and the types associated to keys and nonces are the same in each \([\Gamma]_i^p\).

The property follows from Theorem B.35. Indeed, this theorem guarantees that for all \(i \in [1, n]\), there exists \(C_i \subseteq [C]_i^p\) such that

\[\left[ {\Gamma} \right]_i^p + [P]_i^p \rightarrow [Q]_i^p \rightarrow C_i\]

By construction, all variables in \(\text{dom}([\Gamma]_i^p)\) are indexed with \(i\), and as we mentioned earlier, for all \(i, j\), \([\Gamma]_i^p\) and \([\Gamma]_j^p\) have the same values on their common domain.

Hence we have \(\left[ {\Gamma} \right]_n^p = [\bigcup_{1 \leq i \leq n} ([\Gamma]_i^p)]\). Therefore, by Lemma B.10, we have for all \(i \in [1, n]\)

\[\left[ {\Gamma} \right]_i^p + [P]_i^p \rightarrow [Q]_i^p \rightarrow C_i\]
where
\[ C'_\Gamma = \{ (c, \Gamma' \cup \Gamma'')(c, \Gamma') \in C_i \land \Gamma'' \in \text{branches}(\bigcup_{j \neq i} (\Gamma_j)'_i) \} \]

Thus, by applying rule PPAR \( n \) \(-1\) times, we have
\[ [\Gamma]^n \vdash [P]^n \upharpoonright_1 \ldots \vdash [P]^n_{i} \upharpoonright_1 \vdash \ldots \vdash [Q]^n_{i} \upharpoonright_1 \ldots \vdash [Q]^n_{n} \rightarrow \bigcup_{i \leq n} C'_i. \]

It only remains to be proved that \( \bigcup_{i \leq n} C'_i \subseteq \bigcup_{i \leq n} C_i \). For all \( i \in [1, \ldots, n] \) we have \( C_i \subseteq [C]^n_i \), by Lemma B.11 we know that \( \bigcup_{i \leq n} C_i \subseteq \bigcup_{i \leq n} [C]^n_i \).

Hence it suffices to show that \( \bigcup_{i \leq n} C'_i \subseteq \bigcup_{i \leq n} C_i \).

Let \( (c, \Gamma') \in \bigcup_{i \leq n} C'_i \). By definition there exist \( (c_1, \Gamma_1) \in C'_1, \ldots, (c_n, \Gamma_n) \in C'_n \) such that \( c = \bigcup_{1 \leq i \leq n} c_i \), \( \Gamma' = \bigcup_{1 \leq i \leq n} \Gamma_i \), and for all \( i \neq j, \Gamma_i \) and \( \Gamma_j \) are compatible.

For all \( i, (c_i, \Gamma_i) \in C'_i \). Thus by definition of \( C'_i \) there exist \( \Gamma'_i \) and \( \Gamma''_i \) such that \( (c_i, \Gamma'_i) \in C_i, \Gamma'_i \in \text{branches}(\bigcup_{j \neq i} (\Gamma_j)'_i) \), and \( \Gamma_i = \Gamma'_i \cup \Gamma''_i \).

Moreover, \( [(\bigcup_{i \leq n} C')_i] = [(\bigcup_{i \leq n} C'_i)] \). Indeed, they have the same domain, i.e., \( \{ x \in \text{dom}(\Gamma) \land 1 \leq i \leq n \} \), and are compatible since for all \( i \neq j, \Gamma'_i \) and \( \Gamma'_j \) are compatible.

Thus \( \Gamma' = (\bigcup_{i \leq n} \Gamma'_i) \), and since the \( \Gamma'_i \) are all pairwise compatible, for all \( i, (c_i, \Gamma_i) \in C_i \), we have \( (c, \Gamma') \in \bigcup_{i \leq n} C_i \).

This proves that \( \bigcup_{i \leq n} C'_i \subseteq \bigcup_{i \leq n} C_i \), which concludes the proof. \( \square \)

This next theorem corresponds to Theorem 6.2:

**Theorem B.37.** Consider \( P, Q, P', Q', C, C' \), such that \( P, Q \) and \( P', Q' \) do not share any variable. Consider \( \Gamma \), containing only keys and nonces with types of the form \( \mathbb{T} \).

Assume that \( P \) and \( Q \) only bind nonces with infinite nonce types, i.e. using new \( m : \mathbb{T}^\infty \) for some label \( l \); while \( P' \) and \( Q' \) only bind nonces with finite types, i.e. using new \( m : \mathbb{T}^1 \).

Let us abbreviate by \( \pi \) the sequence of declarations of each nonce \( m \in \text{dom}(\Gamma) \).

- \( \Gamma \vdash P \rightarrow Q \rightarrow C \).
- \( \Gamma \vdash P' \rightarrow Q' \rightarrow C' \).

Then \( C' \cup \bigcup_{i \leq n} (\Gamma_i')^n \) is consistent for all \( n \),

\[ \text{new } \pi \cdot ((P) | P') \equiv_t \text{new } \pi \cdot ((Q) | Q'). \]

**Proof.** Note that since \( \Gamma \) only contains keys and nonces with finite types, for all \( i, [P]^n_i \vdash [P]^n_i \) is just \( P \) where all variables and some names have been \( \alpha \)-renamed, and similarly for \( Q \). Since \( P', Q' \) only contain nonces with finite types, \( [P']^n_i \) and \( [Q']^n_i \) are \( P', Q' \) where all variables have been \( \alpha \)-renamed.

By Theorem B.36, we know that for all \( i, n \),
\[ [\Gamma]^n \vdash [P]^n \upharpoonright_1 \ldots \vdash [P]^n_{i} \upharpoonright_1 \vdash \ldots \vdash [Q]^n_{i} \upharpoonright_1 \ldots \vdash [Q]^n_{n} \rightarrow C' \]

where \( [\Gamma]^n = \bigcup_{1 \leq i \leq n} [\Gamma]^n_i \), and \( C'' \subseteq \bigcup_{1 \leq i \leq n} C^n_i \).

By Theorem B.35, there also exists \( C''' \subseteq [C]_{1}^{\infty} \), such that
\[ [\Gamma]^n \vdash [P']^n_{i} \vdash [Q']^n_{i} \rightarrow C'''. \]

Therefore, by Lemma B.10, we have
\[ [\Gamma]^n \vdash [P']^n \vdash [Q']^n \rightarrow C'''' \]

where \( C'''' \) is \( C'' \) where all the environments have been extended with \( \bigcup_{1 \leq i \leq n} (\Gamma_i)_i \) (note that this environment still only contains nonces and keys).

Therefore, by rules PPAR and PNew,
\[ \Gamma' \vdash \text{new } \pi \cdot (\{ [P]_{1}^{1} \upharpoonright_1 \ldots \vdash [P]_{i}^{1} \upharpoonright_1 \vdash \ldots \vdash [Q]_{i}^{1} \upharpoonright_1 \vdash \ldots \vdash [Q]_{n}^{1} \} ) \rightarrow C'' \]

where \( \Gamma' \) is the restriction of \( [\Gamma]^n \) to keys.

If \( [C]^{\infty}_{1} \cup \bigcup_{i \leq n} [C]_{i}^{1} \) is consistent, similarly to the reasoning in the proof of Theorem B.36, \( \bigcup_{i \leq n} C'''' \) also is.

Then, by Theorem B.28,
\[ \text{new } \pi \cdot (\{ [P]_{1}^{1} \upharpoonright_1 \ldots \vdash [P]_{i}^{1} \upharpoonright_1 \vdash \ldots \vdash [Q]_{i}^{1} \upharpoonright_1 \vdash \ldots \vdash [Q]_{n}^{1} \} ) \rightarrow \text{new } \pi \cdot (\{ [P']_{1}^{1} \upharpoonright_1 \ldots \vdash [P']_{i}^{1} \upharpoonright_1 \vdash \ldots \vdash [Q']_{i}^{1} \upharpoonright_1 \vdash \ldots \vdash [Q']_{n}^{1} \} ) \]

which implies (since \( [P']_{1}^{1} \) is just a renaming of the variables in \( P' \)) that
\[ \text{new } \pi \cdot (\{ [P]_{1}^{1} \upharpoonright_1 \ldots \vdash [P]_{i}^{1} \upharpoonright_1 \vdash \ldots \vdash [Q]_{i}^{1} \upharpoonright_1 \vdash \ldots \vdash [Q]_{n}^{1} \} ) \rightarrow Q' \]
Since \([ P ]_P^P\) and \([ Q ]_Q^Q\) are just \(\alpha\)-renamings of \(P, Q\), this implies that for all \(n\),
\[
\text{new } \pi. \ (P_1 \mid \ldots \mid P_n) \ | \ P' \approx_t \text{new } \pi. \ (Q_1 \mid \ldots \mid Q_n) \ | \ Q'
\]
where \(P_1 = \cdots = P_n = P\), and \(Q_1 = \cdots = Q_n = Q\). Therefore
\[
\text{new } \pi. \ (\langle \langle P \rangle \ | \ P' \rangle) \approx_t \text{new } \pi. \ (\langle \langle Q \rangle \ | \ Q' \rangle).
\]
\(\square\)

### B.3 Checking consistency

In this subsection, we first recall the \texttt{check\_const} procedure from Section 6.3, in more detail, and prove its correctness in the non-replicated case.

For a constraint \(c\) and an environment \(\Gamma\), let
\[
\text{step1}_\Gamma(c) := ([c]_{\sigma_F, \sigma'_F, \Gamma'}),
\]
where
\[
F = \{ x \in \text{dom}(\Gamma) \mid \exists m, n, l, l'. \ (x(x) = \llbracket \tau_m^{l_1} ; \tau_n^{l'_1} \rrbracket) \},
\]
\(\sigma_F, \sigma'_F\) are the substitutions defined by
- \(\text{dom}(\sigma_F) = \text{dom}(\sigma'_F) = F\)
- \(\forall x \in F. \ \forall m, n, l, l'. \ (x(x) = \llbracket \tau_m^{l_1} ; \tau_n^{l'_1} \rrbracket) \Rightarrow \sigma_F(x) = m \land \sigma'_F(x) = n\),

and \(\Gamma'\) is the environment obtained by extending the restriction of \(\Gamma\) to \(\text{dom}(\Gamma)\) with \(\Gamma'(n) = \tau_n^{l_1}\) for all nonce \(n\) such that \(\tau_n^{l_1}\) occurs in \(\Gamma\). This is well defined, since by assumption on the well-formedness of the processes and by definition of the processes, a name \(n\) is always associated with the same label.

Let \(\longrightarrow_\Gamma\) be the reduction relation defined on couples of sets of constraints by (all variables are universally quantified)
\[
\begin{align*}
(\langle \langle M, N \rangle \sim (M', N') \rangle) & \cup c, c' & \longrightarrow_\Gamma & (\langle M \sim M', N \sim N' \rangle) \cup c, c' \\
(\{\text{enc}(M, k) \sim \text{enc}(M', k)\}) & \cup c, c' & \longrightarrow_\Gamma & (\langle M \sim M' \rangle) \cup c, c' & \text{if } \Gamma(k) = \text{key}^{ll_1}(T) \text{ for some } T \\
(\{\text{aenc}(M, pk(k)) \sim \text{aenc}(M', pk(k))\}) & \cup c, c' & \longrightarrow_\Gamma & (\langle M \sim M' \rangle) \cup c, c' & \text{if } \Gamma(k) = \text{key}^{ll_1}(T) \text{ for some } T \\
(\{\text{sign}(M, k) \sim \text{sign}(M', k)\}) & \cup c, c' & \longrightarrow_\Gamma & (\langle M \sim M' \rangle) \cup c, c' & \text{if } \Gamma(k) = \text{key}^{ll_1}(T) \text{ for some } T \\
(\{\text{sign}(M, k) \sim \text{sign}(M', k)\}) & \cup c, c' & \longrightarrow_\Gamma & (\langle M \sim M' \rangle) \cup c, \{\text{sign}(M, k) \sim \text{sign}(M', k)\} \cup c' & \text{if } \Gamma(k) = \text{key}^{ll_1}(T) \text{ for some } T
\end{align*}
\]

Let then \(\text{step2}_\Gamma(c) = c_1 \cup c_2\) where \((c_1, c_2)\) is the normal form of \((\emptyset, \emptyset)\) for \(\longrightarrow_\Gamma\). This definition is equivalent to the one described in Section 6.3, but more practical for the proofs.

We define the condition \(\text{step3}_\Gamma(c)\) as: check that \(c\) only contains elements of the form \(M \sim N\) where \(M\) and \(N\) are both
- a key \(k \in K\) such that \(\exists T. \Gamma(k) = \text{key}^{ll_1}(T)\);
- nonces \(m, n \in N\) such that \(\Gamma(n) = \tau_n^{ll_1} \land \Gamma(m) = \tau_n^{ll_1} a\);
- or public keys, verification keys, or constants;
- or enc(\(M', k\), enc(\(N', k\)) such that \(\exists T. \Gamma(k) = \text{key}^{ll_1}(T)\);
- or either hashes \(h(M')\), \(h(N')\) or encryptions \(\text{enc}(M', pk(k))\), \(\text{aenc}(N', pk(k))\) with a honest key \(k\), i.e. such that \(\exists T. \Gamma(k) = \text{key}^{ll_1}(T)\);
- such that \(M'\) and \(N'\) contain directly under pairs a nonce \(n\) such that \(\Gamma(n) = \tau_n^{ll_1} a\) or a secret key \(k\) such that \(\exists T. \Gamma(k) = \text{key}^{ll_1}(T)\);
- or signatures \(\text{sign}(M', k), \text{sign}(N', k)\) with honest keys, such that \(\exists T. \Gamma(k) = \text{key}^{ll_1}(T)\).
step3Γ(c) returns true if this check succeeds and false otherwise.

We then proceed to step4. We define condition step4Γ(c) as follows. We consider all \( M \sim M' \in c \) and \( N \sim N' \in c \), such that \( M, N \) are unifiable with a most general unifier \( \mu \), and such that

\[
\forall x \in \text{dom}(\mu). \forall l', m, n. (\Gamma(x) = \llbracket r_m^{L,\omega}; r_p^{L,\omega} \rrbracket) \Rightarrow (x \mu \in X \lor \exists i. x \mu = m_i).
\]

We then define the substitution \( \theta \), over all variables \( x \in \text{dom}(\mu) \) such that \( \Gamma(x) = \llbracket r_m^{L,\omega}; r_p^{L,\omega} \rrbracket \) by

\[
\forall x \in \text{dom}(\mu). \forall l', m, p, i. (\Gamma(x) = \llbracket r_m^{L,\omega}; r_p^{L,\omega} \rrbracket \land \mu(x) = m_i) \Rightarrow \theta(x) = p_l
\]

and \( \theta(x) = x \) otherwise.

Let then \( \sigma \) be the restriction of \( \mu \) to \( \{x \in \text{dom}(\mu) | \Gamma(x) = \llbracket L, \omega \rrbracket \land \mu(x) \in N \} \).

We then check that \( M' \sigma \theta = N' \sigma \theta \).

Similarly, we check that the symmetric condition, when \( M' \) and \( N' \) are unifiable, holds for all \( M \sim M' \in c \) and \( N \sim N' \in c \).

If all these checks succeed, \( \text{step4Γ(c)} \) returns true.

Finally, \( \text{check_const(}C) \) is computed by considering all \( (c, \Gamma) \in C \). We let \( (\Gamma, \overline{\Gamma}) = \text{step1Γ(c)} \), and \( \overline{c} = \text{step2Γ(}\overline{\Gamma}) \). We then check that \( \text{step3Γ(}\overline{c} \rangle \) is true and \( \text{step4Γ(}\overline{c} \rangle \) is true. If this check succeeds for all \( (c, \Gamma) \in C \), we say that \( \text{check Const(}C) \) is true.

Note that we only consider constraints obtained by typing, and therefore such that there exists \( c_\phi \) such that \( \Gamma \vdash \phi_\Gamma(c) \sim \phi_\Gamma(c) : LL \rightarrow c_\phi \).

Indeed, it is clear by induction on the typing rules for terms that

\[
\forall \Gamma, M, N, T, c. \quad \Gamma \vdash M \sim N : T \rightarrow c \quad \Rightarrow \quad (\forall u \sim v \in c \cdot \exists c'. \quad \Gamma \vdash u \sim v : LL \rightarrow c').
\]

From this result, and using Lemmas \( B.3 \) and \( B.12 \), it follows clearly by induction on the typing rules for processes that

\[
\forall \Gamma, P, Q, c. \quad \Gamma \vdash P \sim Q : C \quad \Rightarrow \quad (\forall \Gamma, c \in C \cdot \forall u \sim v \in c \cdot \exists c'. \quad \Gamma \vdash u \sim v : LL \rightarrow c').
\]

Let us now prove that the procedure is correct for constraints without infinite nonce types, i.e. constraint sets \( C \) such that

\[
\forall(c, \Gamma) \in C. \forall l', m, n, \Gamma(x) \neq \llbracket r_m^{L,\omega}; r_n^{L,\omega} \rrbracket.
\]

We fix such a constraint set \( C \) (obtained by typing).

Let \( (c, \Gamma) \in C \). Let \( (\overline{\Gamma}, \overline{\Gamma}) = \text{step1Γ(c)} \), and \( \overline{c} = \text{step2Γ(}\overline{\Gamma}) \). Let us assume that \( \text{step2Γ(}\overline{c} \rangle \) is true and \( \text{step4Γ(}\overline{c} \rangle \) is true.

**Lemma B.38.** If \( \overline{\Gamma} \) is consistent in \( \Gamma \), then \( c \) is consistent in \( \Gamma \).

**Proof.** Let \( c' \) be a set of constraints and \( \Gamma' \) be a typing environment such that \( c' \subseteq c \). \( \Gamma' \subseteq \Gamma \), \( \Gamma'_{N,K} = \Gamma_{N,K} \) and \( \text{vars}(c') \subseteq \text{dom}(\Gamma') \).

Let \( \sigma, \sigma' \) be two substitutions such that \( \Gamma'_{N,K} \vdash \sigma \sim \sigma' : \Gamma' \rightarrow c_\sigma \) for some set of constraints \( c_\sigma \).

To prove the claim, we need to show that the frames new \( E_{\Gamma'}(\phi_{\overline{\Gamma}}^{\overline{\Gamma}_{LL}} \cup \phi_{\Gamma}(\llbracket c' \rrbracket_{\sigma,\sigma'})) \) and new \( E_{\Gamma'}(\phi_{\overline{\Gamma}}^{\overline{\Gamma}_{LL}} \cup \phi_{\Gamma}(\llbracket c' \rrbracket_{\sigma,\sigma'})) \) are stastically equivalent. Let \( D \) denote \( \text{dom}(\Gamma') \).

For all \( x \in F \cap D \), by definition of \( F \), there exist \( m, n, l', l'' \) such that \( \Gamma(x) = \llbracket r_m^{L,1}; r_n^{L,1} \rrbracket \). Thus, by well-typedness of \( \sigma, \sigma' \), there exists \( c_x \) such that \( \Gamma \vdash \sigma(x) \sim \sigma'(x) : \llbracket r_m^{L}; r_n^{L} \rrbracket \rightarrow c_x \). Hence, by Lemma B.14, since \( \sigma, \sigma' \) are ground, we have \( \sigma(x) = m \) and \( \sigma'(x) = n \). Therefore, \( \sigma|_{D \cap F} = \sigma|_{D \cap F} \) and \( \sigma'|_{D \cap F} = \sigma'|_{D \cap F} \).

Let \( c' \) be the set \( \llbracket c' \rrbracket_{\sigma|_{D \cap F}, \sigma'|_{D \cap F}} \). We also have \( \llbracket c' \rrbracket_{\sigma|_{D \cap F}, \sigma'|_{D \cap F}} \subseteq \llbracket c' \rrbracket_{\sigma|_{D \cap F}, \sigma'|_{D \cap F}} \), which is equal to \( \llbracket c' \rrbracket_{\sigma|_{D \cap F}, \sigma'|_{D \cap F}} \), since \( \sigma|_{D \cap F} \subseteq \overline{\Gamma} \).

Let \( \Gamma'' = \Gamma|_{\text{dom}(\Gamma')} \cap F \). We have \( \Gamma'' \subseteq \overline{\Gamma} \).

Moreover, since \( \Gamma'_{N,K} \vdash \sigma \sim \sigma' : \Gamma' \rightarrow c_\sigma \), it is clear from the definition of well-typedness for substitutions that we also have \( \Gamma''_{N,K} \vdash \sigma|_{D \cap F} \sim \sigma'|_{D \cap F} : \Gamma''_{D \cap F} \rightarrow c_\sigma \) for some \( c_\sigma \). Finally, \( \sigma'(c') \subseteq \text{dom}(\Gamma'') \) by definition of instantiation, thus \( \text{vars}(c') \subseteq \text{dom}(\Gamma'') \).

We have established that \( c' \subseteq \overline{\Gamma}, \Gamma'' \subseteq \overline{\Gamma}, \text{vars}(c') \subseteq \text{dom}(\Gamma'') \), and \( \Gamma''_{N,K} \vdash \sigma|_{D \cap F} \sim \sigma'|_{D \cap F} : \Gamma''_{D \cap F} \rightarrow c_\sigma \). Therefore, by definition of the consistency of \( \overline{\Gamma} \) in \( \Gamma \), the frames new \( E_{\Gamma'}(\phi_{\overline{\Gamma}}^{\overline{\Gamma}_{LL}} \cup \phi_{\Gamma}(\llbracket c' \rrbracket_{\sigma,\sigma'})) \) and new \( E_{\Gamma'}(\phi_{\overline{\Gamma}}^{\overline{\Gamma}_{LL}} \cup \phi_{\Gamma}(\llbracket c' \rrbracket_{\sigma,\sigma'})) \) are stastically equivalent.

Since \( \phi_{\overline{\Gamma}}^{\overline{\Gamma}_{LL}} \subseteq \phi_{\overline{\Gamma}}^{\overline{\Gamma}_{LL}} \), that is to say that new \( E_{\Gamma'}(\phi_{\overline{\Gamma}}^{\overline{\Gamma}_{LL}} \cup \phi_{\Gamma}(\llbracket c' \rrbracket_{\sigma,\sigma'})) \) and new \( E_{\Gamma'}(\phi_{\overline{\Gamma}}^{\overline{\Gamma}_{LL}} \cup \phi_{\Gamma}(\llbracket c' \rrbracket_{\sigma,\sigma'})) \) are stastically equivalent. This proves the consistency of \( c \) in \( \Gamma \).
Lemma B.39. If \((c_1, c_2) \rightarrow_{\Gamma} (c'_1, c'_2)\) then for all \(x \in \text{dom}(\phi_\Gamma (c'_1 \cup c'_2))\) there exists a recipe \(R\) such that

- \(\text{vars}(R) \subseteq \text{dom}(\phi_\Gamma (c'_1 \cup c'_2))\)
- \(\phi_\Gamma (c'_1 \cup c'_2)(x) = R(\phi_\Gamma (c_1 \cup c_2)) \downarrow\)
- \(\phi_\Gamma (c'_1 \cup c'_2)(x) = R(\phi_\Gamma (c_1 \cup c_2)) \downarrow\).

Conversely, if \((c_1, c_2) \rightarrow_{\Gamma} (c'_1, c'_2)\) then for all \(x \in \text{dom}(\phi_\Gamma (c_1 \cup c_2))\) there exists a recipe \(R\) without destructors, i.e., in which \(\text{dec}, \text{adec}, \text{checksing}, \pi_1, \pi_2\) do not appear, such that

- \(\text{vars}(R) \subseteq \text{dom}(\phi_\Gamma (c'_1 \cup c'_2))\)
- \(\phi_\Gamma (c_1 \cup c_2)(x) = R(\phi_\Gamma (c_1 \cup c_2)) \downarrow\)
- \(\phi_\Gamma (c_1 \cup c_2)(x) = R(\phi_\Gamma (c_1 \cup c_2)) \downarrow\).

Proof. For both directions, it suffices to prove that the claim holds for all \(c_1, c_2, c'_1, c'_2\) such that \((c_1, c_2) \rightarrow_{\Gamma} (c'_1, c'_2)\). Indeed, in that case we prove the result for \(\longrightarrow_{\Gamma}\) by composing all the recipes. The proof for one reduction step is clear by examining the cases for the reduction \(\longrightarrow_{\Gamma}\).

\[\square\]

Lemma B.40. If \(\tilde{c}\) is consistent in \(\Gamma\), then \(\tilde{c}\) is consistent in \(\Gamma\).

Proof. This follows directly from Lemma B.39.

Let \(c'\) be a set of constraints and \(\Gamma'\) be a typing environment such that \(c' \subseteq \tilde{c}, \Gamma' \subseteq \Gamma, \Gamma'_{N,K} = \Gamma_{N,K} \text{ and vars}(c') \subseteq \text{dom}(\Gamma')\). Let \(\sigma, \sigma'\) be two substitutions such that \(\Gamma'_{N,K} \sigma \sim \sigma' : \Gamma' \rightarrow \tilde{c}\) (for some set of constraints \(c_\sigma\)).

To prove the claim, we need to show that the frames new \(\mathcal{E}_{\Gamma'}(\phi_\Gamma^{T}_{\Gamma'_{N,K}} \cup \phi_\Gamma (\tilde{c}'))\) and new \(\mathcal{E}_{\Gamma'}(\phi_\Gamma^{T}_{\Gamma'_{N,K}} \cup \phi_\Gamma (\tilde{c}))\) are statically equivalent.

Since \(\tilde{c}\) is consistent in \(\Gamma\), we know that the frames new \(\mathcal{E}_{\Gamma}(\phi_\Gamma^{T} \cup \phi_\Gamma (\tilde{c}'))\) and new \(\mathcal{E}_{\Gamma}(\phi_\Gamma^{T} \cup \phi_\Gamma (\tilde{c}))\) are statically equivalent.

By Lemma B.39, the frames new \(\mathcal{E}_{\Gamma}(\phi_\Gamma^{T} \cup \phi_\Gamma (\tilde{c}'))\) and new \(\mathcal{E}_{\Gamma}(\phi_\Gamma^{T} \cup \phi_\Gamma (\tilde{c}))\) can be written as a recipe on the frames new \(\mathcal{E}_{\Gamma}(\phi_\Gamma^{T} \cup \phi_\Gamma (\tilde{c}'))\) and new \(\mathcal{E}_{\Gamma}(\phi_\Gamma^{T} \cup \phi_\Gamma (\tilde{c}))\).

Therefore, they are also statically equivalent, which proves the claim.

\[\square\]

Lemma B.41. There exists \(c_\phi\) such that \(\Gamma \vdash \phi_\Gamma (\tilde{c}) \sim \phi_\Gamma (\tilde{c}) : \Gamma \rightarrow c_\phi\).

Proof. As explained previously, there exists \(c'_\phi\) such that \(\Gamma \vdash \phi_\Gamma (c) \sim \phi_\Gamma (c) : \Gamma \rightarrow c'_\phi\).

Moreover, we have by definition \(\Gamma = \Gamma_F \cup \Gamma_T\), where \(F\) is defined as in step1 and \(\Gamma_F\) is the restriction of \(\Gamma\) to \(F\), and for some \(\Gamma_T' \subseteq \Gamma_T\).

In addition \(\Gamma_{F,K} \vdash \sigma_F \sim \sigma'_F : \Gamma_{F,X} \rightarrow c'\) for some \(c'\). By definition of \(F\), and since the refinement types in \(\Gamma\) only contain ground terms by assumption, we also know that \(\Gamma_T\) does not contain refinement types. Hence, by Lemma B.22, and Lemma B.10, there exists \(c_\phi\) such that \(\Gamma \vdash \phi_\Gamma (c) \sim \phi_\Gamma (c) : \Gamma \rightarrow c_\phi\).

Since \(\tilde{c} = [\Gamma_{F,K}, \sigma_F, \sigma'_F]\), this proves that \(\Gamma \vdash \phi_\Gamma (\tilde{c}) \sim \phi_\Gamma (\tilde{c}) : \Gamma \rightarrow c_\phi\). Besides, it is clear from the definition of \(\phi_\Gamma^{T}\) and rules TCstFN, TNonCEl, TKey, TPutKey, TVKey that \(\Gamma \vdash \phi_\Gamma^{T} \sim \phi_\Gamma^{T} : LL \rightarrow \emptyset\).

These two results prove the lemma.

\[\square\]

Lemma B.42. There exists \(c_\phi\) such that \(\Gamma \vdash \phi_\Gamma^{T} \cup \phi_\Gamma (\tilde{c}) \sim \phi_\Gamma^{T} \cup \phi_\Gamma (\tilde{c}) : LL \rightarrow c_\phi\).

Proof. By Lemma B.41, there exists \(c'_\phi\) such that \(\Gamma \vdash \phi_\Gamma^{T} \cup \phi_\Gamma (\tilde{c}) \sim \phi_\Gamma^{T} \cup \phi_\Gamma (\tilde{c}) : LL \rightarrow c'_\phi\).

Moreover, by definition, exist \(c_1, c_2\) such that \((\tilde{c}, \emptyset) \rightarrow_{\Gamma} (c_1, c_2)\) and \(\tilde{c} = c_1 \cup c_2\). Hence, we know by Lemma B.39 that for all \(x \in \text{dom}(\phi_\Gamma (\tilde{c}))\) there exists a recipe \(R\) such that \(\text{names}(R) = \emptyset, \phi_\Gamma (\tilde{c})(x) = R(\phi_\Gamma (c_1 \cup c_2)) \downarrow\) and \(\phi_\Gamma (\tilde{c})(x) = R(\phi_\Gamma (c_1 \cup c_2)) \downarrow\).

Thus, by Lemma B.21, there exists \(c_\phi\) such that \(\Gamma \vdash \phi_\Gamma (\tilde{c}) \sim \phi_\Gamma (\tilde{c}) : LL \rightarrow c_\phi\).

Besides, it is clear from the definition of \(\phi_\Gamma^{T}\) and rules TCstFN, TNonCEl, TKey, TPutKey, TVKey that \(\Gamma \vdash \phi_\Gamma^{T} \sim \phi_\Gamma^{T} : LL \rightarrow \emptyset\).

These two results prove the lemma.

\[\square\]

We now assume that \(\tilde{c}\) satisfies the condition step3(\(\tilde{c}\)).

Note that we write \(\longrightarrow_{\Gamma}\) for \(\longrightarrow_{\Gamma}\) as these relations are equal.

Lemma B.43. If \((\Gamma, \emptyset) \rightarrow_{\Gamma} (c_1, c_2)\) and \(\text{sign}(M, k) \sim \text{sign}(N, k') \in c_2\) then there exists a recipe \(R\) without destructors, i.e., in which \(\text{dec}, \text{adec}, \text{checksing}, \pi_1, \pi_2\) do not appear, such that

- \(\text{vars}(R) \subseteq \text{dom}(\phi_\Gamma (c_1 \cup c_2))\)
- \(M = R(\phi_\Gamma (c_1 \cup c_2))\)
- \(N = R(\phi_\Gamma (c_1 \cup c_2))\).

Proof. For both directions, it suffices to prove that the claim holds for all \(c_1, c_2, c'_1, c'_2\) such that \((c_1, c_2) \rightarrow_{\Gamma} (c'_1, c'_2)\). Indeed, in that case we prove the result for \(\longrightarrow_{\Gamma}\) by composing all the recipes. The proof for one reduction step is clear by examining the cases for the reduction \(\longrightarrow_{\Gamma}\).

\[\square\]
Proof. We prove this property by induction on the length of the reduction. It trivially holds if no reduction step is performed since in that case $c_2 = \emptyset$. Otherwise there exist $c'_1$, $c'_2$ such that $(\tau, \emptyset) \rightarrow^*(c'_1, c'_2) \rightarrow^* (c_1, c_2)$.

If $(c'_1, c'_2) \rightarrow^* (c_1, c_2)$ is any case except the honest signature case, we have $c_2 = c'_2$. Thus if $\text{sign}(M,k) \sim \text{sign}(N,k') \in c_2$, then by the induction hypothesis there exists $R'$ without destructors such that

- $\text{vars}(R') \subseteq \text{dom}(\phi_{LL}^R \cup \phi_\ell(c'_1 \cup c'_2))$
- $M = R'(\phi_{LL}^R \cup \phi_\ell(c'_1 \cup c'_2))$
- $N = R'(\phi_{LL}^R \cup \phi_\ell(c'_1 \cup c'_2))$.

We then prove the claim by applying (the second part of) Lemma B.39 and composing the recipes.

If $(c'_1, c'_2) \rightarrow^* (c_1, c_2)$ corresponds to the honest signature case, we have $c'_1 = c'_1 \cup \{\text{sign}(M', k'') \sim \text{sign}(N', k''\})$, $c_1 = c'_1 \cup \{M' \sim N'\}$, and $c_2 = c'_2 \cup \{\text{sign}(M', k'') \sim \text{sign}(N', k''\})$ for some $c'_1$, $M'$, $N'$, $k''$, $T$ such that $\Gamma(k') = \text{key}_{\text{wh}}(T)$. If $(\text{sign}(M', k''), \text{sign}(N', k'')) \neq (\text{sign}(M, k), \text{sign}(N, k'))$, then the same proof as in the previous case shows the claim. Otherwise, $M \sim N \in c_1$, and therefore the claim trivially holds.

Lemma B.44. If $\text{sign}(M, k) \sim \text{sign}(N, k') \in c$ then there exists a recipe $R$ without destructors, i.e. in which dec, ade, checksign, $\pi_1$, $\pi_2$ do not appear, such that

- $\text{names}(R) = \emptyset$,
- $\text{vars}(R) \subseteq \text{dom}(\phi_{LL}^R \cup \phi_\ell(c))$,
- $M = R(\phi_{LL}^R \cup \phi_\ell(c))$,
- $N = R(\phi_{LL}^R \cup \phi_\ell(c))$.

Proof. This property directly follows from Lemma B.43, applied to $(\tau, \emptyset) \rightarrow^*(c_1, c_2)$ such that $c = c_1 \cup c_2$. Indeed, since $(c_1, c_2)$ is a normal form for $\rightarrow^*$, if $\text{sign}(M, k) \sim \text{sign}(N, k') \in c = c_1 \cup c_2$, then $\text{sign}(M, k) \sim \text{sign}(N, k') \in c_2$, as if it was an element of $c_1$ then another reduction step would be possible.

Lemma B.45. For all recipe $R$ such that $\text{vars}(R) \subseteq \text{dom}(\phi_{LL}^R \cup \phi_\ell(c))$, and $R(\phi_{LL}^R \cup \phi_\ell(c)) \downarrow \neq \bot$ or $R(\phi_{LL}^R \cup \phi_\ell(c)) \downarrow \neq \bot$, there exists a recipe $R'$ without destructors, i.e. in which dec, ade, checksign, $\pi_1$, $\pi_2$ do not appear, such that

- $\text{vars}(R') \subseteq \text{dom}(\phi_{LL}^{R'} \cup \phi_\ell(c))$,
- $R(\phi_{LL}^{R'} \cup \phi_\ell(c)) \downarrow = R'(\phi_{LL}^{R'} \cup \phi_\ell(c))$,
- $R(\phi_{LL}^{R'} \cup \phi_\ell(c)) \downarrow = R'(\phi_{LL}^{R'} \cup \phi_\ell(c))$.

Proof. Let us first note that

$$R(\phi_{LL}^R \cup \phi_\ell(c)) \downarrow \neq \bot \text{ or } R(\phi_{LL}^R \cup \phi_\ell(c)) \downarrow \neq \bot$$

is equivalent to

$$R(\phi_{LL}^{R'} \cup \phi_\ell(c)) \downarrow \neq \bot \text{ and } R(\phi_{LL}^{R'} \cup \phi_\ell(c)) \downarrow \neq \bot.$$ 

This follows from Lemmas B.42 and B.21.

We prove the property by induction on $R$.

- If $R = n \in \mathbb{N}$ or $R = x \in A\mathbb{X}$ or $R = a \in C \cup \mathfrak{F} \mathbb{N}$ then the claim holds with $R' = R$.
- If the head symbol of $R$ is a constructor, i.e. if there exist $R_1$, $R_2$, such that $R = p\text{k}(R_1)$ or $R = v\text{k}(R_1)$ or $R = \text{enc}(R_1, R_2)$ or $R = a\text{enc}(R_1, R_2)$ or $R = \text{sign}(R_1, R_2)$ or $R = \langle R_1, R_2 \rangle$ or $R = h(R_1)$, we may apply the induction hypothesis to $R_1$ and $R_2$ when it is present. All these cases are similar, we write the proof generically for $R = f(R_1, R_2)$. By the induction hypothesis, there exist $R'_1, R'_2$ such that
  - $\text{vars}(R'_1) \cup \text{vars}(R'_2) \subseteq \text{dom}(\phi_{LL}^{R'_1} \cup \phi_\ell(c'))$,
  - for all $i \in \{1, 2\}$, $R_i(\phi_{LL}^{R'_i} \cup \phi_\ell(c')) \downarrow = R'_i(\phi_{LL}^{R'_i} \cup \phi_\ell(c'))$,
  - for all $i \in \{1, 2\}$, $R_i(\phi_{LL}^{R'_i} \cup \phi_\ell(c')) \downarrow = R'_i(\phi_{LL}^{R'_i} \cup \phi_\ell(c'))$.

Let $R''$ be $f(R'_1, R'_2)$. The first two points imply that $R''$ satisfies the conditions on variables. Since $R(\phi_{LL}^{R} \cup \phi_\ell(c)) \downarrow = f(R_1(\phi_{LL}^{R_1} \cup \phi_\ell(c))) \downarrow$, $R_2(\phi_{LL}^{R_2} \cup \phi_\ell(c)) \downarrow$, the third point implies that $R(\phi_{LL}^{R} \cup \phi_\ell(c)) \downarrow = R'(\phi_{LL}^{R' \cup \phi_\ell(c)})$. Similarly, $R(\phi_{LL}^{R} \cup \phi_\ell(c)) \downarrow = R'(\phi_{LL}^{R' \cup \phi_\ell(c)})$, and the claim holds.

- If $R = \text{dec}(S, K)$ for some recipes $S, K$, then since $R(\phi_{LL}^{R} \cup \phi_\ell(c)) \downarrow \neq \bot$, we have $K(\phi_{LL}^{R} \cup \phi_\ell(c)) \downarrow = k$ for some $k \in K$, and $S(\phi_{LL}^{R} \cup \phi_\ell(c)) \downarrow = \text{enc}(M, k)$, where $M = R(\phi_{LL}^{R} \cup \phi_\ell(c))$.

Similarly, there exists $k' \in \mathcal{K}$ such that $K(\phi_{LL}^{R} \cup \phi_\ell(c)) \downarrow = k'$ and $S(\phi_{LL}^{R} \cup \phi_\ell(c)) \downarrow = \text{enc}(N, k')$, where $N = R(\phi_{LL}^{R} \cup \phi_\ell(c))$. 

In addition, by Lemma B.42, there exists $c'$ such that $\Gamma \vdash \phi_{\Gamma} \cup \phi_r(\tilde{c}) \sim \phi_{\Gamma} \cup \phi_r(c) : L \rightarrow \tilde{c}'$. Thus by Lemma B.21, there exists $c''$ such that $\Gamma \vdash K(\phi_{\Gamma} \cup \phi_r(\tilde{c})) \downarrow K(\phi_{\Gamma} \cup \phi_r(c)) : L \rightarrow c''$, which is to say $\Gamma \vdash k \rightarrow k' : L \rightarrow c''$. Hence by Lemma B.18, $k = k'$ and $\Gamma(k) = k_{\text{key}}(T)$ for some type $T$.

Since $S(\phi_{\Gamma} \cup \phi_r(c)) \downarrow = \text{enc}(M, k) \neq \perp$, by the induction hypothesis, there exists $S'$ such that $\text{vars}(S') \subseteq \text{vars}(S)$, $S'(\phi_{\Gamma} \cup \phi_r(c)) = S(\phi_{\Gamma} \cup \phi_r(c)) \downarrow = \text{enc}(M, k)$, and $S'(\phi_{\Gamma} \cup \phi_r(c)) = S(\phi_{\Gamma} \cup \phi_r(c)) \downarrow = \text{enc}(N, k)$.

It is then clear that either $S' = \chi$ for some variable $x \in \mathcal{AX}$, or $S' = \text{enc}(S'', K')$ for some $S''$, $K'$. The first case is impossible, since we have already shown that $\Gamma(k) = k_{\text{key}}(T)$, and since by step3\(\tilde{c}\), $\tilde{c}$ only contains messages encrypted with keys $k''$ such that $\Gamma(k'') = k_{\text{key}}(T')$ for some $T'$.

Hence there exist $S''$, $K'$ such that $S' = \text{enc}(S'', K')$. Since $S'(\phi_{\Gamma} \cup \phi_r(c)) = \text{enc}(M, k)$, we have $S''(\phi_{\Gamma} \cup \phi_r(c)) = M$. Hence $R(\phi_{\Gamma} \cup \phi_r(c)) \downarrow = M = S''(\phi_{\Gamma} \cup \phi_r(c))$, and similarly for $\phi_r(c)$. Moreover, $S''$ being a subterm of $S'$ it also satisfies the conditions on the domains, and thus the property holds with $R' = S''$.

• If $R = \text{check sign}(S, K)$ for some recipes $S, K$: this case is similar to the symmetric case.

• If $R = \text{check sign}(S, K)$ for some recipes $S, K$ then since $R(\phi_{\Gamma} \cup \phi_r(c)) \downarrow \neq \perp$, we have $K(\phi_{\Gamma} \cup \phi_r(c)) \downarrow = \text{vk}(k)$ for some $k \in K$, and $S(\phi_{\Gamma} \cup \phi_r(c)) \downarrow = \text{sign}(M, k)$, where $M = R(\phi_{\Gamma} \cup \phi_r(c)) \downarrow$.

Similarly, there exists $k' \in \mathcal{K}$ such that $K(\phi_{\Gamma} \cup \phi_r(c)) \downarrow = \text{vk}(k')$ and $S(\phi_{\Gamma} \cup \phi_r(c)) \downarrow = \text{sign}(N, k')$, where $N = R(\phi_{\Gamma} \cup \phi_r(c)) \downarrow$.

Since $S(\phi_{\Gamma} \cup \phi_r(c)) \downarrow = \text{sign}(M, k) \neq \perp$, by the induction hypothesis, there exists $S'$ such that $\text{vars}(S') \subseteq \text{vars}(S)$, $S'(\phi_{\Gamma} \cup \phi_r(c)) = S(\phi_{\Gamma} \cup \phi_r(c)) \downarrow = \text{sign}(M, k)$, and $S'(\phi_{\Gamma} \cup \phi_r(c)) = S(\phi_{\Gamma} \cup \phi_r(c)) \downarrow = \text{sign}(N, k)$.

Since $S'(\phi_{\Gamma} \cup \phi_r(c)) = \text{sign}(M, k)$, it is clear from the definition of $\downarrow$ that either $S' = x$ for some $x \in \mathcal{AX}$, or $S' = \text{sign}(S'', K')$ for some $S''$, $K'$.

In the first case, we therefore have $\text{sign}(M, k) = \text{sign}(N, k') \in \tilde{c}$, and Lemma B.44 directly proves the claim.

In the second case, there exist $S''$, $K'$ such that $S' = \text{sign}(S'', K')$. Since $S'(\phi_{\Gamma} \cup \phi_r(c)) = \text{sign}(M, k)$, we have $S''(\phi_{\Gamma} \cup \phi_r(c)) = M$.

Hence $R(\phi_{\Gamma} \cup \phi_r(c)) \downarrow = M = S''(\phi_{\Gamma} \cup \phi_r(c))$, and similarly for $\phi_r(c)$. Moreover, $S''$ being a subterm of $S'$ it also satisfies the conditions on the domains, and thus the property holds with $R' = S''$.

• If $R = \pi_1(S)$ for some recipe $S$ then since $R(\phi_{\Gamma} \cup \phi_r(c)) \downarrow \neq \perp$, we have $S(\phi_{\Gamma} \cup \phi_r(c)) \downarrow = (M_1, M_2)$, where $M_1 = R(\phi_{\Gamma} \cup \phi_r(c)) \downarrow$ and $M_2$ is a message.

Similarly, $S(\phi_{\Gamma} \cup \phi_r(c)) \downarrow = (N_1, N_2)$, where $N_1 = R(\phi_{\Gamma} \cup \phi_r(c)) \downarrow$, and $N_2$ is a message.

Since $S'(\phi_{\Gamma} \cup \phi_r(c)) \downarrow = (M_1, M_2) \neq \perp$, by the induction hypothesis, there exists $S'$ such that $\text{vars}(S') \subseteq \text{vars}(S)$, $S'(\phi_{\Gamma} \cup \phi_r(c)) = S(\phi_{\Gamma} \cup \phi_r(c)) \downarrow = (M, k)$, and $S'(\phi_{\Gamma} \cup \phi_r(c)) = S(\phi_{\Gamma} \cup \phi_r(c)) \downarrow = (N, k)$.

Since $S'(\phi_{\Gamma} \cup \phi_r(c)) = (M_1, M_2)$, it is clear from the definition of $\downarrow$ that either $S' = x$ for some $x \in \mathcal{AX}$, or $S' = (S_1, S_2)$ for some $S_1$, $S_2$.

The first case is impossible, since by step3\(\tilde{c}\), $\tilde{c}$ does not contain pairs.

In the second case, there exist $S_1, S_2$ such that $S' = (S_1, S_2)$. Since $S'(\phi_{\Gamma} \cup \phi_r(c)) = (M_1, M_2)$, we have $S_1(\phi_{\Gamma} \cup \phi_r(c)) = M_1$. Hence $R(\phi_{\Gamma} \cup \phi_r(c)) \downarrow = M_1 = S_1(\phi_{\Gamma} \cup \phi_r(c))$, and similarly for $\phi_r(c)$. Moreover, $S_1$ being a subterm of $S'$ it also satisfies the conditions on the domains, and thus the property holds with $R' = S_1$.

• If $R = \pi_2(S)$ for some $S$: this case is similar to the $\pi_1$ case.

□

**Lemma B.46.** For all term $t$ and substitution $\sigma$ containing only messages, if $t \downarrow \neq \perp$, then $(t(\sigma)) \downarrow = (t \downarrow)\sigma$.

**Proof.** This property is easily proved by induction on $t$. In the base case where $t$ is a variable $x$, by definition of $\downarrow$, since $\sigma(x)$ is a message, $\sigma(x) \downarrow = \sigma(x)$ and the claim holds. In the other base cases where $t$ is a name, key or constant the claim trivially holds. We prove the case where $t$ starts with a constructor other than enc, aenc, sign generically for $t = f(t_1, t_2)$. We then have $t_1 \sigma \downarrow \neq \perp$ and $t_2 \sigma \downarrow \neq \perp$, and $t \sigma \downarrow = (f(t_1 \downarrow \sigma, t_2 \downarrow \sigma)$, i.e. to $f(t_1, t_2) \downarrow \sigma$. The case where $f$ is enc, aenc or sign is similar, but we add in addition that $t_2 \downarrow$ is a key.

Finally if $t$ starts with a destructor, $t = d(t_1, t_2)$, we know that $t_1 \downarrow$ starts with the corresponding constructor $f: t_1 \downarrow = f(t_3, t_4)$. In the case of encryptions and signatures we know in addition that $t_4 \downarrow$ and $t_2 \downarrow$ are the same key (resp. public key/verification key). We then have $t_1 \downarrow = t_3 \downarrow$ and $t \sigma \downarrow = t_3 \sigma \downarrow$ (or $t_4 \downarrow$ in the case of the second projection $\pi_2)$. Hence by the induction hypothesis, $t \sigma \downarrow = t_3 \downarrow \sigma = t \downarrow \sigma$ and the claim holds.

□
LEMMA B.47. For all \( \sigma, \sigma' \), for all recipe \( R \) such that \( \text{vars}(R) \subseteq \text{dom}(\phi^R_{\text{LL}} \cup \phi_r(\tilde{c})) \), if \( R(\phi^R_{\text{LL}} \cup \phi_r([\tilde{c}]_{\sigma, \sigma'})) \neq \bot \) then \( R(\phi^R_{\text{LL}} \cup \phi_r(\tilde{c})) \neq \bot \); and similarly if \( R(\phi^R_{\text{LL}} \cup \phi_r([\tilde{c}]_{\sigma, \sigma'})) \neq \bot \) then \( R(\phi^R_{\text{LL}} \cup \phi_r(\tilde{c})) \neq \bot \). 

Proof. We prove the property for \( \phi_r(\tilde{c}) \), as the proof for \( \phi_r(\tilde{c}) \) is similar. 

We prove this by induction on \( R \).

- If \( R = \text{enc}, \text{aecn}, \text{sign}, \text{pk}, \text{vk} \): all these cases are similar, we detail the proof for the asymmetric encryption case.

We have \( R = \text{aecn}(S, K) \) for some recipes \( S, K \). Since \( R(\phi^R_{\text{LL}} \cup \phi_r([\tilde{c}]_{\sigma, \sigma'})) \neq \bot \), we have \( S(\phi^R_{\text{LL}} \cup \phi_r([\tilde{c}]_{\sigma, \sigma'})) \neq \bot \), and \( K(\phi^R_{\text{LL}} \cup \phi_r([\tilde{c}]_{\sigma, \sigma'})) \neq \bot \).

By the induction hypothesis, we thus have \( K(\phi^R_{\text{LL}} \cup \phi_r([\tilde{c}]_{\sigma, \sigma'})) \neq \bot \). Hence, by Lemma B.45, there exists a recipe \( K' \) without destructors, such that \( K'(\phi^R_{\text{LL}} \cup \phi_r([\tilde{c}]_{\sigma, \sigma'})) \neq \bot \) and \( \text{sign}(\phi^R_{\text{LL}} \cup \phi_r([\tilde{c}]_{\sigma, \sigma'})) \neq \bot \).

Since \( K'(\phi^R_{\text{LL}} \cup \phi_r([\tilde{c}]_{\sigma, \sigma'})) \neq \bot \), there exists a variable \( x \) such that

- either \( K' = \text{dec} \), and then since \( \phi_r(\tilde{c}) \) does not contain variables by \text{step2}\( \tilde{c} \), we have \( (\phi^R_{\text{LL}} \cup \phi_r([\tilde{c}]_{\sigma, \sigma'}))(x) = \text{pk}(k) \).

By Lemma B.42, there exists \( c_k \) such that \( \text{check}(\phi_r([\tilde{c}]_{\sigma, \sigma'}))(x) = \text{pk}(k) \), we know by Lemma B.18 that \( \phi_r(\tilde{c})(x) = \text{pk}(k) \).

- or \( K' = \text{dec} \), and then since \( \phi_r(\tilde{c}) \) does not contain variables by \text{step2}\( \tilde{c} \), we have \( (\phi^R_{\text{LL}} \cup \phi_r([\tilde{c}]_{\sigma, \sigma'}))(x) = k \).

In any case \( K'(\phi^R_{\text{LL}} \cup \phi_r([\tilde{c}]_{\sigma, \sigma'})) \neq \bot \) and \( \text{sign}(\phi^R_{\text{LL}} \cup \phi_r([\tilde{c}]_{\sigma, \sigma'})) \neq \bot \), and therefore

\[
\text{K}(\phi^R_{\text{LL}} \cup \phi_r([\tilde{c}]_{\sigma, \sigma'})) \neq \bot.
\]
Therefore, $R(\phi_{LL}^T \cup \phi_\epsilon(\hat{\epsilon})) \downarrow = (\pi_1(S))(\phi_{LL}^T \cup \phi_\epsilon(\hat{\epsilon})) \not\downarrow \perp$.

\[ \text{Lemmas B.48. For all } \alpha, \alpha', \text{ for all recipes } R \text{ such that } \text{vars}(R) \subseteq \text{dom}(\phi_{LL}^T \cup \phi_\epsilon(\hat{\epsilon})), R(\phi_{LL}^T \cup \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha}) \not\downarrow \perp \text{ and } R(\phi_{LL}^T \cup \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha'}) \not\downarrow \perp, \text{ there exists a recipe } R' \text{ without destructors, i.e. in which dec, adeq, checksign, } \pi_1, \pi_2, \text{ do not appear, such that} \]

\begin{itemize}
  \item $\text{vars}(R') \subseteq \text{dom}(\phi_{LL}^T \cup \phi_\epsilon(\hat{\epsilon}))$, \n  \item $R(\phi_{LL}^T \cup \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha}) \downarrow = R'(\phi_{LL}^T \cup \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha})$, \n  \item $R(\phi_{LL}^T \cup \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha'}) \downarrow = R'(\phi_{LL}^T \cup \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha'})$. \n\end{itemize}

\[ \text{Proof. By Lemma B.47, } R(\phi_{LL}^T \cup \phi_\epsilon(\hat{\epsilon})) \not\downarrow \perp \text{ and } R(\phi_{LL}^T \cup \phi_\epsilon(\hat{\epsilon})) \not\downarrow \perp. \]

Hence we may apply Lemma B.45, and there exists a recipe $R'$ without destructors, such that

\begin{itemize}
  \item $\text{vars}(R') \subseteq \text{dom}(\phi_{LL}^T \cup \phi_\epsilon(\hat{\epsilon}))$, \n  \item $R(\phi_{LL}^T \cup \phi_\epsilon(\hat{\epsilon})) \downarrow = R'(\phi_{LL}^T \cup \phi_\epsilon(\hat{\epsilon}))$, \n  \item $R(\phi_{LL}^T \cup \phi_\epsilon(\hat{\epsilon})) \downarrow = R'(\phi_{LL}^T \cup \phi_\epsilon(\hat{\epsilon}))$. \n\end{itemize}

By Lemma B.46, we have

\[ (R(\phi_{LL}^T \cup \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha})) \downarrow = (R(\phi_{LL}^T \cup \phi_\epsilon(\hat{\epsilon}))\sigma) \downarrow = (R(\phi_{LL}^T \cup \phi_\epsilon(\hat{\epsilon})) \downarrow \sigma \]

Hence

\[ (R(\phi_{LL}^T \cup \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha})) \downarrow = R'(\phi_{LL}^T \cup \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha}) = R'(\phi_{LL}^T \cup \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha}). \]

Similarly we can show that

\[ (R(\phi_{LL}^T \cup \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha})) \downarrow = R'(\phi_{LL}^T \cup \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha}). \]

This proves the claim.

\[ \text{Lemmas B.49. For all } \alpha, \alpha', \text{ for all recipe } R \text{ such that } \text{vars}(R) \subseteq \text{dom}(\phi_{LL}^T \cup \phi_\epsilon(\hat{\epsilon})), \text{ for all } x \in \text{dom}(\phi_{LL}^T \cup \phi_\epsilon(\hat{\epsilon})), \text{ if } R(\phi_{LL}^T \cup \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha'}) = (\phi_{LL}^T \cup \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha'})(x) \text{ then } R \text{ is a variable } y \in \text{dom}(\phi_{LL}^T \cup \phi_\epsilon(\hat{\epsilon})), \text{ or } R \in C \cup \mathcal{F}N. \]

Similarly, if $R(\phi_{LL}^T \cup \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha'}) = (\phi_{LL}^T \cup \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha'})(x)$ then $R$ is a variable $y \in \text{dom}(\phi_{LL}^T \cup \phi_\epsilon(\hat{\epsilon}))$ or $R \in C \cup \mathcal{F}N.

\[ \text{Proof. We only detail the proof for } \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha}, \text{ as the proof for } \phi_\epsilon(\hat{\epsilon}) \text{ is similar.} \]

We distinguish several cases for $R$.

\begin{itemize}
  \item If $R = a \in C \cup \mathcal{F}N$, the claim clearly holds. \n  \item If $R = x \in \mathcal{ACK}$, then the claim trivially holds. \n  \item If $R = \text{enc}(S, K)$ or $\text{sign}(S, K)$ for some recipes $S, K$: these two cases are similar, we only detail the encryption case. $(\phi_{LL}^T \cup \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha})(x)$ is then an encrypted message, which, because of the form of $\hat{\epsilon}$, implies that $K(\phi_{LL}^T \cup \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha}) = k$ for some $k \in K$ such that $\Gamma(k) = \text{key}^{\mathcal{H}T}(T)$ for some $T$. This is only possible if there exists a variable $z$ such that $\Gamma(z) = z$ and $(\phi_{LL}^T \cup \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha})(z) = k$, which, by step3(\hat{\epsilon}) and the definition of $\phi_{LL}^T$, implies that $\Gamma(k) = \text{key}^{\mathcal{H}T}(T')$ for some $T'$, which is contradictory. \n  \item If $R = \text{aenc}(S, K)$ or $h(S)$ for some recipes $S, K$: these two cases are similar, we only detail the encryption case. $(\phi_{LL}^T \cup \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha})(x)$ is then an asymmetrically encrypted message, which, because of the form of $\hat{\epsilon}$, implies that $S(\phi_{LL}^T \cup \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha})$ contains directly under pairs a nonce $n$ such that $\Gamma(n) = r_n^{\mathcal{H}T}$, or a key $k \in K$ such that $\Gamma(k) = \text{key}^{\mathcal{H}T}(T)$ for some $T$. This is only possible if there exists a recipe $S'$ such that $S(\phi_{LL}^T \cup \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha}) = n$ (resp. $k$). \n  Since $R$ can only contain names from $\mathcal{F}N$, this implies that there exists a variable $z$ such that $(\phi_{LL}^T \cup \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha})(z) = n$ (resp. $k$), which, by step3(\hat{\epsilon}), and the definition of $\phi_{LL}^T$, implies that $\Gamma(n) = r_n^{\mathcal{H}T}$ (resp. $\Gamma(k) = \text{key}^{\mathcal{H}T}(T')$ for some $T'$), which is contradictory. \n  \item Finally, the head symbol of $R$ cannot be $\langle \cdot, \cdot \rangle$, $\text{dec}$, $\text{adeq}$, $\text{checks} \text{ign}, \pi_1, \pi_2$, because of the form of $\hat{\epsilon}$ (step3(\hat{\epsilon})). \n\end{itemize}

\[ \square \]

We now assume that step4(\hat{\epsilon}) holds. Note that since $\Gamma$, and hence $\Gamma$, do not contain refinements or nonces with infinite nonce types, the step4(\hat{\epsilon}) condition is simpler than in the general case. Indeed the condition on the most general unifier $\mu$ is always trivially satisfied, and the substitution $\rho$ is the identity.

\[ \text{Lemmas B.50. For all } \alpha, \alpha' \text{ such that } \exists \alpha'' \subseteq \Gamma, \exists \sigma. \Gamma' \cup \alpha' \sim \alpha : \Gamma' \rightarrow c_\sigma, \text{ the frames } \text{new } \mathcal{E}_\Gamma \phi_{LL}^T \cup \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha} \text{ and } \text{new } \mathcal{E}_\Gamma \phi_{LL}^T \cup \phi_\epsilon([\hat{\epsilon}])_{\sigma, \alpha'} \text{ are statically equivalent.} \]
Proof. Let \( R, S \) be two recipes such that \( \text{vars}(R) \cup \text{vars}(S) \subseteq \text{dom}(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) = \text{dom}(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \).

Let us show that
\[
(R(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma}))) ⊨ (S(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma}))) \quad \iff \quad (R(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma}))) ⊨ (S(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})))
\]

We only detail the proof for the \( \Rightarrow \), as the other direction is similar. We then assume that \( (R(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma}))) ⊨ (S(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma}))) \).

Let us first note that
\[
(R(\phi^T_{LL} \cup \phi_T(\bar{c}))) \not\equiv \quad \iff \quad (S(\phi^T_{LL} \cup \phi_T(\bar{c}))) \not\equiv.
\]

This follows from Lemmas B.42 and B.21.

Since \( \Gamma'_{X',\kappa} \vdash \sigma \sim \sigma' : \Gamma'_{X} \rightarrow c_{\sigma}, \) as by definition we only type messages, \( \sigma \) and \( \sigma' \) only contain messages. Hence, by Lemmas B.47 and B.46,
\[
R(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \not\equiv \quad \iff \quad \phi_T(\bar{c}) \not\equiv.
\]

and
\[
R(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \not\equiv \quad \iff \quad \phi_T(\bar{c}) \not\equiv.
\]

Hence, by chaining all these equivalences, we have
\[
R(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \not\equiv \quad \iff \quad R(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \not\equiv.
\]

Similarly, we can show that
\[
S(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \not\equiv \quad \iff \quad S(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \not\equiv.
\]

Therefore, if \( R(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \not\equiv \), i.e. \( S(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \not\equiv \), then \( R(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \not\equiv \) and \( S(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \not\equiv \), and the claim holds.

Let us now assume that \( R(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \not\equiv \), then \( R(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \not\equiv \) and \( S(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \not\equiv \).

By Lemma B.48, there exist \( R', S' \) without destructors such that
- \( \text{vars}(R') \subseteq \text{dom}(\phi^T_{LL} \cup \phi_T(\bar{c})) \) and \( \text{vars}(S') \subseteq \text{dom}(\phi^T_{LL} \cup \phi_T(\bar{c})) \),
- \( R(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \not\equiv \) \( R'(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \),
- \( R(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \not\equiv \) \( R'(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \),
- \( S(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \not\equiv \) \( S'(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \),
- \( S(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \not\equiv \) \( S'(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \).

Since \( (R(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma}))) \not\equiv \), we have \( R'(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) = S'(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \).

We show that \( R'(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) = S'(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \), by induction on the recipes \( R', S' \). Since \( R'(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) = S'(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \), we can distinguish four cases for \( R' \) and \( S' \):
- If they have the same head symbol, either this symbol is a nonce or constant and the claim is trivial, or it is a variable, and we handle this case later, or it is a destructor or constructor. We write the proof for this last case generically for \( R' = f(R'') \) and \( S' = f(S'') \).
- We have necessarily \( R''(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) = S''(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \). It then follows by applying the induction hypothesis to \( R'' \) and \( S'' \) that \( R''(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) = S''(\phi^T_{LL} \cup \phi_T([\bar{c}]_{\sigma',\sigma})) \). The claim follows by applying \( f \) on both sides of this equalities.
- If \( R' \) is a variable and not \( S' \), then by Lemma B.49, \( S' \in C \cup \mathcal{T} \). Let us denote \( R' = x \). By Lemma B.42, there exists \( c_x \) such that \( \Gamma + (\phi^T_{LL} \cup \phi_T(\bar{c})) \vdash R' \rightarrow c_x \). Since \( (\phi^T_{LL} \cup \phi_T(\bar{c}))(x) = R'(\phi^T_{LL} \cup \phi_T(\bar{c})) \in C \cup \mathcal{T} \), by Lemma B.18, we have \( \phi^T_{LL} \cup \phi_T(\bar{c})(x) = R'(\phi^T_{LL} \cup \phi_T(\bar{c})), \) i.e. \( R'(\phi^T_{LL} \cup \phi_T(\bar{c})) = S'(\phi^T_{LL} \cup \phi_T(\bar{c})), \) and \( R'(\phi^T_{LL} \cup \phi_T(\bar{c})) \).
- If \( S' \) is a variable and not \( R' \), this case is similar to the previous one.
- If \( R', S' \) are two variables \( x \) and \( y \), we have \( \phi^T_{LL} \cup \phi_T(\bar{c})(x) = \phi^T_{LL} \cup \phi_T(\bar{c})(y) \).
- We can then prove \( (\phi^T_{LL} \cup \phi_T(\bar{c}))(x) = \phi^T_{LL} \cup \phi_T(\bar{c})(y) \).
- Indeed:
  - if \( x, y \in \text{dom}(\phi^T_{LL}) \), this follows from the definition of \( \phi^T_{LL} \).
  - if \( x \in \text{dom}(\phi^T_{LL}) \) and \( y \in \text{dom}(\phi_T(\bar{c})) \), then by definition of \( \phi^T_{LL} \), \( R'(\phi^T_{LL} \cup \phi_T(\bar{c})) = \phi^T_{LL}(x) \) is a nonce, key, public key, or verification key. Hence \( \phi_T(\bar{c})(y) \) is also a nonce, key, public key or verification key. By step 3, \( \phi_T(\bar{c})(y) = \phi_T(\bar{c})(y) \).
  - This implies that \( \phi_T(\bar{c})(y) \) is also a nonce, key, public key or verification key. By Lemma B.42, there exists \( c_y \) such that \( \Gamma + \phi_T(\bar{c}) \vdash \phi_T(\bar{c}) \rightarrow c_y \), and hence by Lemma B.18, \( \phi_T(\bar{c})(y) = \phi_T(\bar{c})(y) \). That is to say
  \( R'(\phi^T_{LL} \cup \phi_T(\bar{c})) = (\phi^T_{LL} \cup \phi_T(\bar{c}))(x) = \phi^T_{LL} \cup \phi_T(\bar{c})(y) \).
In this subsection, we prove the results regarding the procedure when checking consistency in the replicated case. Note that this implies use it to denote the replacement of nonces appearing in the refinements. In the case of typing environments it denotes the replacement of $\text{dom}$. This theorem corresponds to Theorem 6.5.

Since the frames $P/r.sc/o.sc/o.sc/f.sc.$ $\llbracket \Gamma \rrbracket$ be such that $\text{vars}(\alpha) \cap \llbracket \Gamma \rrbracket = \emptyset$. By definition of $\text{dom}(\Gamma)$, any occurrence of $\sigma = \llbracket \Gamma \rrbracket \cup \llbracket \sigma \rrbracket$ is a nonce.

This proves the property.

LEMMA B.51. $c$ is consistent in $\Gamma$.

Proof. By Lemmas B.38 and B.40, it suffices to show that $\tilde{c}$ is consistent in $\Gamma$. Let $c' \subseteq \bar{c}$, $\Gamma' \subseteq \Gamma$ be such that $\llbracket \Gamma' \rrbracket \subseteq \llbracket \Gamma \rrbracket$. Let $\sigma, \sigma'$ be such that $\Gamma', N, \mathcal{K} \vdash \sigma \sim \sigma' : \Gamma' \to c_{\sigma'}$ for some $c_{\sigma'}$. By Lemma B.50, the frames new $\tilde{\epsilon}^{c}_{\Gamma} \llbracket \phi_{L} \cup \phi_{r}(\llbracket \tilde{c} \rrbracket_{\sigma, \sigma'})$ and new $\tilde{\epsilon}^{c}_{\Gamma} \llbracket \phi_{L} \cup \phi_{r}(\llbracket \tilde{c} \rrbracket_{\sigma, \sigma'})$ are statically equivalent.

Since the frames new $\tilde{\epsilon}^{c}_{\Gamma} \llbracket \phi_{L} \cup \phi_{r}(\llbracket \tilde{c} \rrbracket_{\sigma, \sigma'})$ and new $\tilde{\epsilon}^{c}_{\Gamma} \llbracket \phi_{L} \cup \phi_{r}(\llbracket \tilde{c} \rrbracket_{\sigma, \sigma'})$ are subsets of these frames, they also are statically equivalent. Therefore $\tilde{c}$ is consistent in $\Gamma$.

This next theorem corresponds to Theorem 6.4.

THEOREM B.52 (Soundness of the procedure). Let $C$ be a constraint set without infinite nonce types, i.e.,

$$\forall (c, \Gamma) \in C. \forall l, l', m, n. \Gamma(x) \neq \llbracket \cdot \rrbracket_{m, n} \cup \llbracket \cdot \rrbracket_{l', n}.$$ 

If check\_const$(C)$ succeeds, then $C$ is consistent.

Proof. The previous lemmas directly imply that for all $(c, \Gamma) \in C$, $c$ is consistent in $\Gamma$. This proves the theorem. □

B.4 Consistency for replicated processes

In this subsection, we prove the results regarding the procedure when checking consistency in the replicated case.

In this subsection, we only consider constraints obtained by typing processes (with the same key types). Notably, by the well-formedness assumptions on the processes, this means that a nonce $n$ is always associated with the same nonce type.

This theorem corresponds to Theorem 6.5.

THEOREM B.53. Let $C$ and $C'$ be two constraint sets such that

$$\forall (c, \Gamma) \in C. \forall (c', \Gamma'). \in C'. \quad \text{dom}(\Gamma_{X}) \cap \text{dom}(\Gamma'_{X}) = \emptyset.$$ 

For all $n \in \mathbb{N}$, if check\_const$(\llbracket C \rrbracket_{1}^{n} \cup \llbracket C \rrbracket_{2}^{n} \cup \llbracket C' \rrbracket_{1}^{n} = true$, then check\_const$(\llbracket \cup_{1 \leq i \leq n} (C \llbracket i \rrbracket_{1}^{n} \cup C' \llbracket i \rrbracket_{1}^{n}) = true$.

Proof. Let $n \in \mathbb{N}$. Let $C$ be such that $\forall (c, \Gamma) \in C. \text{dom}(\Gamma) \cap \text{dom}(\Gamma') = \emptyset$. Let $(c, \Gamma) \in (\cup_{1 \leq i \leq n} (C \llbracket i \rrbracket_{1}^{n} \cup C' \llbracket i \rrbracket_{1}^{n})$. By definition of $\cup_{1 \leq i \leq n}$, there exists $(c', \Gamma') \in \llbracket C' \rrbracket_{1}^{n}$, and for all $i \in [1, n]$, there exists $(c_{i}, \Gamma_{i}) \in \llbracket C \rrbracket_{1}^{n}$, such that

- $c = (\cup_{1 \leq i \leq n} c_{i}) \cup c';$
- $\Gamma = (\cup_{1 \leq i \leq n} \Gamma_{i}) \cup \Gamma'$.

Let $i \in [1, n]$. Since $(c_{i}, \Gamma_{i}) \in \llbracket C \rrbracket_{1}^{n}$, by definition of $\llbracket C \rrbracket_{1}^{n}$ there exists $(c'_{i}, \Gamma'_{i}) \in C$ such that

- $c_{i} = \llbracket c'_{i} \rrbracket_{1}^{n};$
- $\Gamma_{i} \in \text{branches}(\llbracket \Gamma'_{i} \rrbracket_{1}^{n}).$

Note that this implies $\text{dom}(\Gamma_{X})$ only contains variables indexed by $i$, and, from the assumption that $\text{vars}(c_{i}) \subseteq \text{dom}(\Gamma'_{X})$, that $\text{vars}(c_{i}) \subseteq \text{dom}(\Gamma_{X}).$

For all $i \in [1, n]$, let $\delta_{i}^{1}$ denote the function on terms which consists in exchanging all occurrences of the indices $i$ and 1, i.e. replacing any occurrence of $m_{1}$ (for all nonce $m$ with an infinite nonce type) with $m_{i}$, any occurrence of $m_{1}$ with $m_{i}$, any occurrence of $x_{1}$ with $x_{i}$ (for all variable $x$), and any occurrence of $x_{1}$ with $x_{i}$ (also for all variable $x$).

We also (abusing notations) apply this function to constraints, types, typing environments and constraint sets. In the case of types we use it to denote the replacement of nonces appearing in the refinements. In the case of typing environments it denotes the replacement of
nonces appearing in the types, and of nonces and variables in the domain of the environment, i.e. $(\delta^1_1(\Gamma)(x_1) = \delta^1_1(\Gamma(x_1)))$. In the case of constraint sets it denotes the application of the function to each constraint and environment in the constraint set.

Similarly, we denote $\delta^2_i$ the function exchanging indices $i$ and $2$.

For all $h \in [1, n]$ and all $i \neq j \in [1, n]$, such that $i \neq 2$ and $j \neq 1$, let

$$(c_{h,i}^{i,j}, \Gamma_{h,i}^{i,j}) = (c_{h}, \Gamma_{h}) \delta^1_i \delta^2_j.$$  

Similarly, for all $h \in [1, n]$ and all $i \in [1, n]$, let

$$(c_{h,i}^{i,2}, \Gamma_{h,i}^{i,2}) = (c_{h}, \Gamma_{h}) \delta^2_i.$$  

Finally, for all $i \in [1, n]$, let $\Gamma^{i,i}$ be the typing environment such that $\text{dom}(\Gamma^{i,i}) = \text{dom}(\Gamma')$ and $\forall x \in \text{dom}(\Gamma^{i,i}), \Gamma^{i,i}(x) = \Gamma'(x) \delta^1_i$.

Since $(c_{i}, \Gamma_{i}) \in [C]_n$, we can show that $(c_{i}^{i,j}, \Gamma_{i}^{i,j}) \in [C]_n$. Indeed, recall that there exists $(c_{i}^{i,j}, \Gamma_{i}^{i,j}) \in C$ such that $c_{i} = [c_{i}^{i,j}]_{\Gamma_{i}}$ and $\Gamma_{i} \in \text{branches}(\Gamma_{i}^{i,j})$. $c_{i}$ only contains variables and names indexed by $i$, hence it is clear that $c_{i}^{i,j} = [c_{i}^{i,j}]_{\Gamma_{i}^{i,j}}$. Moreover, since $\Gamma_{i} \in \text{branches}(\Gamma_{i}^{i,j})$, it is clear that $\Gamma_{i}^{i,j} \in \text{branches}(\Gamma_{i}^{i,j})$.

By definition, indexed nonces or variables appear in $[\Gamma_{i}^{i,j}]_n$ only in its domain, and as parts of union types of the form $[m_1; \ldots; m_n]$. This union type is left unchanged by $\delta^1_i \delta^2_j$; since $i \neq j$, $i \neq 2$, and $j \neq 1$, $\delta^1_i \delta^2_j$ is indeed only performing a permutation of the indices. Hence, $[\Gamma_{i}^{i,j}]_n \delta^1_i \delta^2_j = [\Gamma_{i}^{i,j}]_n$. Thus $\Gamma_{i}^{i,j} \in \text{branches}(\Gamma_{i}^{i,j})$.

Therefore, $(c_{i}^{i,j}, \Gamma_{i}^{i,j}) \in [C]_n$.

Note that $\text{dom}(\Gamma_{i}^{i,j})$ only contains variables indexed by 1; and that, since $\text{vars}(c_{i}) \subseteq \text{dom}(\Gamma_{i})$, we have $\text{vars}(c_{i}^{i,j}) \subseteq \text{dom}(\Gamma_{i}^{i,j})$.

Similarly, if $j \neq i$, $i \neq 2$ and $j \neq 1$, $(c_{i}^{2,j}, \Gamma_{i}^{2,j}) \in [C]_n$. Note that $\text{dom}(\Gamma_{i}^{2,j})$ only contains variables indexed by 2; and that $\text{vars}(c_{i}^{2,j}) \subseteq \text{dom}(\Gamma_{i}^{2,j})$.

Similarly, we also have $(c', \Gamma') \in [C']_n$.

Note that for all $(c'', \Gamma'') \in [C]_n$, and all $(c''', \Gamma''') \in [C']_n$, and since by assumption: $\forall (c, \Gamma) \in C. \forall (c', \Gamma') \in C$. $\text{dom}(\Gamma) \cap \text{dom}(\Gamma') = \emptyset$,

we know that $\Gamma''$ and $\Gamma'''$ are compatible. In particular this applies to all the $\Gamma_{i}^{i,j}$ and $\Gamma'$ (as well as $\Gamma_{i}^{i,j}$ and $\Gamma''$).

Moreover, for all $(c'', \Gamma'') \in [C]_n$, and all $(c''', \Gamma''') \in [C']_n$, since $\text{dom}(\Gamma''') \subseteq \{x_1 \mid x \in X\}$, and $\text{dom}(\Gamma'') \subseteq \{x_2 \mid x \in X\}$, $\Gamma''$ and $\Gamma'''$ are also compatible. This in particular applies to $\Gamma_{i}^{i,j}$ for all $i \neq j \in [1, n]$ and $\Gamma'$ (as well as $\Gamma_{i}^{i,j}$ and $\Gamma''$).

If $C$ is empty, then so is $\text{U}_{1 \leq i \leq n} [C]_n$ and the claim clearly holds. Let us now assume that $C$ is not empty. Hence for all $i \in [1, n]$, $[C]_n$ is not empty.

The procedure for $c, \Gamma$ is as follows:

1. We compute $\text{step}_{1}(c)$. Following the notations used in the procedure, we have

$$F = \{x \in \text{dom}(\Gamma) \mid \exists m_{1,1}. \Gamma(x) = [\Gamma_{m_{1,1}}^{1,1}]\},$$

and we write $(\overline{c}, \overline{\Gamma}) \overset{\text{def}}{=} \text{step}_{1}(c)$.

For all $i \in [1, n]$, let $\overline{c_{i}^{i,j}}, \overline{\Gamma_{i}^{i,j}} \overset{\text{def}}{=} \text{step}_{1}(c_{i})$. Let also $(\overline{c}, \overline{\Gamma}) \overset{\text{def}}{=} \text{step}_{1}(c')$. We have $\overline{c} = (\cup_{1 \leq i \leq n} c_{i}), \overline{\Gamma} = (\cup_{1 \leq i \leq n} \Gamma_{i}) \cup \Gamma'$.

For all $h, i, j \in [1, n]$, such that either $i \neq j$ and $i \neq 2$ and $j \neq 1$, or $i = j$, let also $\overline{c_{h,i}^{i,j}}, \overline{\Gamma_{h,i}^{i,j}} \overset{\text{def}}{=} \text{step}_{1}(c_{h})$. Similarly, for all $i \in [1, n]$, let also $(\overline{c_{h,i}^{i,j}}, \overline{\Gamma_{h,i}^{i,j}}) \overset{\text{def}}{=} \text{step}_{1}(c_{h})$. Since, for $i \neq j$, $(\overline{c_{h,i}^{i,j}}, \overline{\Gamma_{h,i}^{i,j}}) = (c_{h}, \Gamma_{h}) \delta^1_i \delta^2_j$, it can easily be shown (by induction on the terms) that $(\overline{c_{h,i}^{i,j}}, \overline{\Gamma_{h,i}^{i,j}}) = (\overline{c_{h}}, \overline{\Gamma_{h}}) \delta^1_i \delta^2_j$. Similarly, $(\overline{c_{h,i}^{i,2}}, \overline{\Gamma_{h,i}^{i,2}}) = (\overline{c_{h}}, \overline{\Gamma_{h}}) \delta^2_i$.

Finally, we similarly also have $(\overline{c'}, \overline{\Gamma'}) = (\overline{c'}, \overline{\Gamma'}) \delta^1_i \delta^2_j$.

2. We compute $\overline{c} \overset{\text{def}}{=} \text{step}_{2}(\overline{\Gamma})$.

Note that, by the assumption that $C$ and $C'$ are obtained by typing processes, all the environments $\Gamma, \Gamma_{i}, \Gamma', \Gamma_{h,i}^{i,j}, \Gamma_{h,i}^{i,2}$ contain the same keys, associated with the same labels. The same is thus true for $\overline{\Gamma}, \overline{\Gamma}_{i}, \overline{\Gamma}', \overline{\Gamma}_{h,i}^{i,j}, \overline{\Gamma}_{h,i}^{i,2}$. Hence, $\text{step}_{2}(\overline{\Gamma})$, $\text{step}_{2}(\overline{\Gamma}_{i})$, $\text{step}_{2}(\overline{\Gamma})$, $\text{step}_{2}(\overline{\Gamma}_{h,i}^{i,j})$, $\text{step}_{2}(\overline{\Gamma}_{h,i}^{i,2})$, all denote the same function.

For all $i \in [1, n]$, let $\overline{c_{i}} \overset{\text{def}}{=} \text{step}_{2}(\overline{\Gamma}_{i})$. Similarly, let $\overline{c'} = \text{step}_{2}(\overline{\Gamma'})$. It is clear that $\overline{c'} = (\cup_{1 \leq i \leq n} \overline{c_{i}}) \cup \overline{c'}$.

Similarly, for all $h, i, j \in [1, n]$, such that either $i \neq j$ and $i \neq 2$ and $j \neq 1$, or $i = j$, let $\overline{c_{h,i}^{i,j}} \overset{\text{def}}{=} \text{step}_{2}(\overline{\Gamma_{h,i}^{i,j}})$. Let also $\overline{c_{h}} \overset{\text{def}}{=} \text{step}_{2}(\overline{\Gamma_{h}})$. It can easily be seen that for all $h \in [1, n], i \neq j \in [1, n]$ such that $i \neq 2$ and $j \neq 1$, $\overline{c_{h,i}^{i,j}} = \overline{c_{h}} \delta^1_i \delta^2_j$, we have $\overline{c_{h,i}^{i,j}} = \overline{c_{h}} \delta^1_i \delta^2_j$.

Similarly, $\overline{c_{h,i}^{i,j}} = \overline{c_{h}} \delta^1_i \delta^2_j$. Finally, we similarly also have $\overline{c'} = \overline{c'} \delta^1_i \delta^2_j$. 

(3) We check that step3(ς) holds, i.e. that each \( M \sim N \in \bar{e} \) has the correct form (with respect to the definition of step3).

If \( M \sim N \in \bar{e} \), either \( M \sim N \in \bar{e} \), or there exists \( i \in [1, n] \) such that \( M \sim N \in \bar{e}_i \).

- In the first case, \( M \sim N \in \bar{e}_i \). By assumption, \( | \Gamma_i \setminus \alpha \times \Gamma_i \setminus \alpha \) is not empty. Hence there exist \( (e'', \Gamma'') \in \bar{e}_i \). Thus, \( (e'' \cup \alpha \cup \Gamma'' \cup \Gamma'') \in \bar{C}_i \). (as noted previously, \( \Gamma'', \Gamma'' \) are compatible).

- In the second case, \( M \sim N \in \bar{e}_i \) for some \( i \in [1, n] \).

Let \( M' = M_i^{\delta_1} \) and \( N' = N_i^{\delta_1} \). Since \( \bar{e}_i^{\delta_1} = \bar{e}_j^{\delta_1} \), we have \( M' \sim N' \in \bar{e}_i^{\delta_1} \).

By assumption, \( | \Gamma_i^{\delta_1} \setminus \alpha \times \Gamma_i^{\delta_1} \setminus \alpha \) is not empty, hence there exists \( (e''', \Gamma''') \in \bar{e}_i^{\delta_1} \). Thus, \( (e''' \cup \alpha \cup \Gamma''' \cup \Gamma''') \in \bar{C}_i^{\delta_1} \) (as noted previously, \( \Gamma'', \Gamma''' \) are compatible).

Hence, by assumption, check_const((\( (e'' \cup \alpha \cup \Gamma'' \cup \Gamma'') \)) succeeds.

If \( e'' = \text{step2}_M(\text{fs}(\text{step1}_M(e''))) \), then \( e'' \cup \alpha \cup \Gamma'' \cup \Gamma' \) holds. Therefore, \( \text{step3}_M(e'' \cup \alpha \cup \Gamma'') \) holds.

In particular, \( M \sim N \in \bar{e} \) has the correct form.

Therefore, step3(ς) holds.

(4) Finally, we check that step4(ς) holds. Let \( M_1 \sim N_1 \in \bar{e}_1 \) and \( M_2 \sim N_2 \in \bar{e}_2 \). Let us use the property in the case where \( M_1 \) and \( M_2 \) are unifiable with a most general unifier \( \mu \). The case where \( N_1 \) and \( N_2 \) are unifiable is similar.

Let then \( \alpha \) be the restriction of \( \mu \) to \( \{ x \in \text{vars}(M_1) \cup \text{vars}(M_2) \} \) \( \bar{T}(x) = \text{LL} \land \mu(x) \in \bar{N} \).

We have proved that \( N_1 \alpha = N_2 \alpha \).

Since we already have \( \bar{e} = (\cup_{1 \leq i \leq 2} \bar{e}_i) \cup \bar{e}' \), we know that:

- either there exists \( i \in [1, n] \) such that \( M_1 \sim N_1 \in \bar{e}_i \);
- or \( M_1 \sim N_1 \in \bar{e}' \);

and

- either there exists \( j \in [1, n] \) such that \( M_2 \sim N_2 \in \bar{e}_j \);
- or \( M_2 \sim N_2 \in \bar{e}' \).

Let us first prove the case where there exist \( i, j \in [1, n] \) such that \( M_1 \sim N_1 \in \bar{e}_i \) and \( M_2 \sim N_2 \in \bar{e}_j \). We distinguish two cases.

- If \( i \neq j \). The property to prove is symmetric between \( M_1 \sim N_1 \in \bar{e}_1 \) and \( M_2 \sim N_2 \in \bar{e}_2 \). Hence without loss of generality we may assume that \( i \neq 2 \) and \( j \neq 2 \). Indeed, if we assume that the property can be proved in that case, then in the case where \( i = 2 \) or \( j = 1 \), we may exchange the two constraints. The property holds for \( M_2 \sim N_2 \in \bar{e}_j \) and \( M_1 \sim N_1 \in \bar{e}_i \); as \( i \neq j \), and \( i = 2 \) or \( j = 1 \), we know that \( i \neq 1 \) and \( j \neq 2 \). Then by symmetry it also holds for \( M_1 \sim N_1 \in \bar{e}_i \) and \( M_2 \sim N_2 \in \bar{e}_j \).

Let us hence assume that \( i \neq 2 \) and \( j \neq 2 \).

Let then \( M_1' = M_i^{\delta_1} \), \( N_1' = N_i^{\delta_1} \), \( M_2' = M_i^{\delta_1} \), \( N_2' = N_i^{\delta_1} \).

Since \( \bar{e}_i^{\delta_1} = \bar{e}_j^{\delta_1} \), we have \( M_1' \sim N_1' \in \bar{e}_i^{\delta_1} \).

Similarly, \( M_2' \sim N_2' \in \bar{e}_j^{\delta_1} \).

Since \( M_1 \) and \( M_2 \) are unifiable, then so are \( M_1' \) and \( M_2' \), with a most general unifier \( \mu' \) which satisfies \( \mu(x) = t \Leftrightarrow \mu'(x\delta_1^{\delta_2}) = t\delta_1^{\delta_2} \).

Let then \( \alpha' \) be the restriction of \( \mu' \) to \( \{ x \in \text{vars}(M_1') \cup \text{vars}(M_2') \} \) \( \bar{T}(x) = \text{LL} \land \mu'(x) \in \bar{N} \).

Similarly \( \alpha' \) is such that \( \forall x \in \text{dom}(\alpha') \forall n. \alpha'(n) = n \Leftrightarrow \alpha'(x\delta_1^{\delta_2}) = n\delta_1^{\delta_2} \).

By assumption, \( \text{check\_const}(\{((e'' \cup \alpha \cup \Gamma'' \cup \Gamma'') \}) \) succeeds.

Thus, \( \text{step4}(\{((e'' \cup \alpha \cup \Gamma'' \cup \Gamma'') \}) \) succeeds. If \( e'' = \text{step2}(\text{fs}(\text{step1}(e''))) \), then \( e'' \cup \alpha \cup \Gamma'' \cup \Gamma' \) holds. Therefore, \( \text{step4}(e'' \cup \alpha \cup \Gamma'') \) holds.

Thus \( N_1' \alpha = N_2' \alpha \).

Since \( N_1' \alpha = N_2' \alpha \), \( e'' = e''' \).

Therefore the claim holds in this case.

- If \( i = j \) then let \( M_1' = M_i^{\delta_1} \), \( N_1' = N_i^{\delta_1} \), \( M_2' = M_i^{\delta_1} \), \( N_2' = N_i^{\delta_1} \).

Since \( \bar{e}_i^{\delta_1} = \bar{e}_i^{\delta_1} \), we have \( M_1' \sim N_1' \in \bar{e}_i^{\delta_1} \).

Similarly, \( M_2' \sim N_2' \in \bar{e}_i^{\delta_1} \).

Since \( M_1 \) and \( M_2 \) are unifiable, then so are \( M_1' \) and \( M_2' \), with a most general unifier \( \mu' \) which satisfies \( \mu(x) = t \Leftrightarrow \mu'(x\delta_1^{\delta_2}) = t\delta_1^{\delta_2} \).

Let then \( \alpha' \) be the restriction of \( \mu' \) to \( \{ x \in \text{vars}(M_1') \cup \text{vars}(M_2') \} \) \( \bar{T}(x) = \text{LL} \land \mu'(x) \in \bar{N} \).

Similarly \( \alpha' \) is such that \( \forall x \in \text{dom}(\alpha') \forall n. \alpha'(n) = n \Leftrightarrow \alpha'(x\delta_1^{\delta_2}) = n\delta_1^{\delta_2} \).

By assumption, \( \text{check\_const}(\{((e'' \cup \alpha \cup \Gamma'' \cup \Gamma'') \}) \) succeeds.

Thus, \( \text{step4}(\{((e'' \cup \alpha \cup \Gamma'' \cup \Gamma'') \}) \) succeeds. If \( e'' = \text{step2}(\text{fs}(\text{step1}(e''))) \), then \( e'' \cup \alpha \cup \Gamma'' \cup \Gamma' \) holds. Therefore, \( \text{step4}(e'' \cup \alpha \cup \Gamma'') \) holds.

Thus \( N_1' \alpha \delta_1^{\delta_2} = N_2' \alpha \delta_1^{\delta_2} \), i.e. \( N_1 \alpha = N_2 \alpha \).

Therefore the claim holds in this case.
Thus, step4(Γ(1), Γ′, Γ′′, c) holds, and since \( \{\alpha_1^i \sim \alpha_2^{i'} \cup \alpha'' \cup c' \} \) holds, and since \( \{\alpha_3^j \sim \alpha_4^{j'} \cup \alpha' \} \cup \{\alpha_1^i \sim \alpha_2^{i'} \cup \alpha'' \cup c' \} \) holds, we know that \( \alpha' = \alpha''_a' \).

Thus \( \delta_4^{i'} \alpha'' \delta_4^{i'} \), i.e., since \( \delta_4^{i'} \alpha'' \delta_4^{i'} \), \( N_1 \alpha = N_2 \alpha \). Therefore the claim holds in this case.

Let us now prove the case where there exists \( i \in [1, n] \) such that \( M_1 \sim N_1 \in c \), and \( M_2 \sim N_2 \in c' \). The symmetric case, where \( M_1 \sim N_1 \in c' \) and there exists \( j \in [1, n] \) such that \( M_2 \sim N_2 \in c \), is similar.

Let then \( M_1 = M_2 \delta_4^{i'} \), \( N_1 = N_2 \delta_4^{i'} \). Since \( \Gamma_i \cup c' = c_1 \delta_4^{i'} \), we have \( \Gamma_i = N_1 \delta_4^{i'} \). Similarly, \( M_2 = N_2 \delta_4^{i'} \). Since \( M_1 \) and \( M_2 \) are unifiable, then so are \( M_1 ' \) and \( M_2 ' \), with a most general unifier \( \mu' \) which satisfies \( \mu(x) = t \Leftrightarrow \mu'(x) \in \delta_4^{i'} \).

Let then \( \alpha' \) be the restriction of \( \mu' \) to \( x \in \text{vars}(M_1) \cup \text{vars}(M_2) \). \( \Gamma_i(1) \cup \Gamma_i(2) \) is a nonempty set.

Since \( \alpha' \) is such that \( \forall x \in \text{dom}(\alpha'), \forall n \in \alpha'(x) = n \Leftrightarrow \alpha'(x) = n \delta_4^{i'} \), i.e., \( \delta_4^{i'} \alpha'' \delta_4^{i'} = \alpha$. Therefore, by assumption, \( \text{check}_{\alpha'} \{\Gamma_i(1) \cup \Gamma_i(2) \} \) succeeds.

By assumption, \( \{ \Gamma_i(1) \} \) is not empty, hence there exists \( \{ \Gamma_i(1) \} \). Moreover, as noted previously, \( \{ \Gamma_i(1) \} \). Thus, \( \{ \Gamma_i(1) \} \cup c' \). Hence, by assumption, \( \text{check}_{\alpha'} \{ \Gamma_i(1) \cup c' \} \) succeeds.

Let \( \alpha' \), \( \alpha'' \) be the substitutions defined by \( \text{check}_{\alpha'} \{ \Gamma_i(1) \cup c' \} \) succeeds.

Therefore, for every \( (c, \Gamma) \in (U_{1 \leq i \leq n} \{ \Gamma_i(1) \}) \cup \{ \Gamma_i(1) \} \), \( \text{check}_{\alpha'} \{ \Gamma_i(1) \} \) succeeds, which proves the claim. \( \square \)

This next lemma is as follows:

**Lemma B.54.** For all \( (c, \Gamma) \) such that \( \text{vars}(c) \subseteq \text{dom}(\Gamma) \) which only contains variables indexed by 1 or 2, and all names in \( c \) have finite non-zero types, if \( \text{check}_{\alpha'} \{ \Gamma_i(1) \} \) succeeds, then for all \( \Gamma'' \in \text{branches}(\Gamma) \), where \( \Gamma' = \Gamma | \bigcup_{1 \leq i \leq n} \{ \Gamma_i(1) \} \) and \( \Gamma'' \in \text{branches}(\Gamma) \), \( \text{check}_{\alpha'} \{ \Gamma'' \} \) succeeds.

**Proof.** Let \( a \in N \).

Let \( \alpha' \) be as assumed in the statement of the lemma.

Let \( a \) be as assumed in the statement of the lemma.

Let us assume that \( \text{check}_{\alpha'} \{ \Gamma_i(1) \} \) succeeds. Let \( \Gamma' = \Gamma | \bigcup_{1 \leq i \leq n} \{ \Gamma_i(1) \} \) and \( \Gamma'' \in \text{branches}(\Gamma') \).

The procedure \( \text{check}_{\alpha'} \{ \Gamma_i(1) \} \) is as follows:

(1) We compute \( (c, \Gamma'') = \text{step1}_{\alpha'}(c) \). Following the notations in the procedure, we denote \( F = \{ x \in \text{dom}(\Gamma'') \mid \exists m, p, l, l'. \Gamma''(x) = [m_{1}, l_{1} ; p_{1} l_{1}'] \} \). Let \( F' = \{ x \in \text{dom}(\Gamma) \mid \exists m, p, l, l'. \Gamma(x) = [m_{1}, l_{1} ; p_{1} l_{1}'] \} \). Let \( F'' = \{ x \in \text{dom}(\Gamma) \mid \exists m, p, l, l'. \Gamma'(x) = [m_{1}, l_{1} ; p_{1} l_{1}'] \} \). It is easily seen from the definition of \( \Gamma' \) that \( F = F' \cup F'' \).

By definition of \( \text{step1}_{\alpha'}(c) \), \( \Gamma'' \) contains \( \Gamma'' | \text{dom}(\Gamma'') \).

Let \( (c, \Gamma') = \text{step1}_{\alpha'}(c) \).

It is clear from the definition of \( \Gamma' \) and \( \Gamma'' \) that for all \( x \in F'' \), there exists \( i \in [1, n] \) and \( m, p, l, l' \) such that \( \Gamma(x) = [m_{1}, l_{1} ; p_{1} l_{1}'] \) and \( \Gamma''(x) = [m_{1}, l_{1} ; p_{1} l_{1}'] \). Let \( \sigma_{\ell} \) and \( \sigma_{r} \) be the substitutions defined by \( \text{dom}(\sigma_{\ell}) = \text{dom}(\sigma_{r}) = F'' \).

and

\( \forall x \in F''. \forall m, p \in N, \forall l, l'. \Gamma''(x) = [m_{1}, l_{1} ; p_{1} l_{1}'] \Rightarrow (\sigma_{\ell}(x) = m_{1} \And \sigma_{r}(x) = p_{1}) \).

It is clear from the definition of \( \Gamma' \) and \( \Gamma'' \) that \( \text{check}_{\alpha'} \{ \Gamma'' \} \) succeeds.
(2) We compute \( \hat{c} \overset{\text{def}}{=} \text{step2}(\Gamma) \).

Similarly, \( \hat{c'} \overset{\text{def}}{=} \text{step2}(\Gamma') \).

It can easily be seen by induction on the reduction \( (\Gamma, \emptyset) \rightarrow (c_1, c_2) \) (using the fact that \( \rightharpoonup \equiv \rightarrow_1 \)) that \( \hat{c} = [\hat{c}]_{\sigma_{\ell}, \sigma_r} \).

(3) We check that \( \text{step3}(\hat{c}) \) holds.

Let \( u \sim v \in \hat{c} \). Since \( \hat{c} = [\hat{c'}]_{\sigma_{\ell}, \sigma_r} \), there exists \( u' \sim v' \in \hat{c'} \) such that \( u = u' \sigma_{\ell} \) and \( v = v' \sigma_r \).

Since \( \text{check\_const}((\{c, \Gamma\}) = \text{true} \), we know that \( u' \) and \( v' \) have the required form. Note that by definition of \( \Gamma' \), the keys which are low in \( \Gamma' \), i.e. the keys \( k \in \mathcal{K} \) such that there exist \( T \) such that \( \Gamma'(k) = \text{key}_{LL}(T) \), are exactly the keys which are low in \( \Gamma \).

It clearly follows, by examining all cases for \( u' \) and \( v' \), that \( u' \sigma_{\ell} \) and \( v' \sigma_r \), i.e. \( u \) and \( v \), also have the required form.

Therefore, \( \text{step3}(\hat{c}) \) holds.

(4) Finally, we check the condition \( \text{step4}(\hat{c}) \).

Let \( \Gamma_1 \sim N_1 \sim c \) and \( \Gamma_2 \sim N_2 \sim c \). Since \( \hat{c} = [\hat{c'}]_{\sigma_{\ell}, \sigma_r} \), there exist \( \Gamma'_1 \sim N'_1 \sim c' \) and \( \Gamma'_2 \sim N'_2 \sim c' \) such that \( \Gamma_1 = \Gamma'_1 \sigma_{\ell} \), \( N_1 = N'_1 \sigma_r \), \( \Gamma_2 = \Gamma'_2 \sigma_{\ell} \), and \( N_2 = N'_2 \sigma_r \).

Let us prove the first direction of the equivalence, i.e. the case where \( \Gamma_1, \Gamma_2 \) are unifiable. The proof for the case where \( N_1, N_2 \) are unifiable is similar.

If \( \Gamma_1, \Gamma_2 \) are unifiable, let \( \mu \) be their most general unifier. We have \( \Gamma_1 \mu = \Gamma_2 \mu \), i.e. \( (\Gamma_1' \sigma_{\ell}) \mu = (\Gamma_2' \sigma_{\ell}) \mu \).

Let \( \tau \) denote the substitution \( \sigma_{\ell} \mu \). Since \( \Gamma'_1 \tau = \Gamma_2' \tau \), \( \Gamma'_1 \) and \( \Gamma'_2 \) are unifiable. Let \( \mu' \) be their most general unifier. There exists \( \theta \) such that \( \tau = \mu' \theta \).

Let also \( \alpha \) be the restriction of \( \mu \) to \( \{ x \in \text{vars}(\Gamma_1) \cup \text{vars}(\Gamma_2) \mid \Gamma'(x) = \text{LL} \land \mu(x) = N_1 \} \).

Note that \( \Gamma'(x) = \text{LL} \iff \Gamma(x) = \text{LL} \).

We have to prove that \( N_1 \alpha = N_2 \alpha \).

Let \( x \in \text{vars}(\Gamma'_1) \cup \text{vars}(\Gamma'_2) \) such that \( \Gamma'(x) = \text{LL} \land \mu'(x) \in N \). We have \( (x \mu') \theta = (x \sigma_{\ell} \mu) = (x \sigma_{\ell} \mu) \mu = m \mu = m_i \).

Thus, \( x \mu' \) can only be either a variable \( y \) such that \( y \theta = m_i \), or the nonce \( m_i \).

Therefore, \( \mu' \) satisfies the conditions on the most general unifier expressed in \( \text{step4}(\hat{c}) \).

Let \( x \in \text{vars}(\Gamma_1) \cup \text{vars}(\Gamma_2) \) such that \( \Gamma'(x) = \text{LL} \land \mu(x) \in N \). We have \( (x \mu') \theta = (x \sigma_{\ell} \mu) = (x \sigma_{\ell} \mu) \mu = m \mu = \mu(x) \in N \).

Thus, \( x \mu' \) can only be either a variable \( y \) (such that \( y \theta = \mu(x) \)), or the nonce \( \mu(x) \).

Conversely, let \( x \in \text{vars}(\Gamma'_1) \cup \text{vars}(\Gamma'_2) \) such that \( \Gamma(x) = \text{LL} \land \mu'(x) \in N \). We have \( x \mu = (x \sigma_{\ell} \mu) = (x \mu') \theta = \mu'(x) \).

Let then \( \theta' \) be the substitution with domain \( \{ x \in \text{vars}(\Gamma'_1) \cup \text{vars}(\Gamma'_2) \mid \exists m, p, l', l \exists i \in [1, n], \Gamma(x) = \text{LL} \land \mu'(x) = m_i \} \) such that \( \forall x \in \text{dom}(\theta') \), \( \theta'(x) = p_i \) if \( \mu'(x) = m_i \) and \( \Gamma(x) = \text{LL} \land \mu'(x) = m_i \).

Let also \( \alpha' \) be the restriction of \( \mu' \) to \( \{ x \in \text{vars}(\Gamma'_1) \cup \text{vars}(\Gamma'_2) \mid \Gamma(x) = \text{LL} \land \mu'(x) \in N \} \).

Since \( \text{check\_const}((\{c, \Gamma\}) = \text{true} \), we know that \( \text{step4}(\hat{c}) \) holds. Since \( \Gamma'_1 \sim N'_1 \sim c' \) and \( \Gamma'_2 \sim N'_2 \sim c' \), this implies that \( N'_1 \alpha' \theta = N'_2 \alpha' \theta' \).

As we have just shown, for all \( x \in \text{dom}(\theta') \), \( x \sigma_{\ell} \mu = m_i \) and \( x \sigma_r = p_i \), and \( \mu'(x) \) is either \( m_i \) or a variable. By definition of \( \text{dom}(\theta') \), only the case where \( \mu'(x) = m_i \) is actually possible, we have \( \theta'(x) = p_i \).

Thus, \( \forall x \in \text{dom}(\theta') \), \( \sigma_{\ell} \mu = \theta'(x) \).

It is then clear from the definitions of the domains of \( \theta' \) and \( \sigma_{\ell} \) that there exists \( \tau' \) such that \( \sigma_r = \theta' \).

Thus, since we have shown that \( N'_1 \alpha' \theta = N'_2 \alpha' \theta' \), we have \( (N'_1 \alpha' \theta') \tau' = (N'_2 \alpha' \theta') \tau' \), that is to say \( N'_1 \alpha' \sigma_{\ell} = N'_2 \alpha' \sigma_r \), i.e. since \( \alpha' \) and \( \sigma_r \) have disjoint domains, and are both ground, \( N_1 \alpha' = N_2 \alpha' \).

Moreover, we have shown that for all \( x \in \text{vars}(\Gamma'_1) \cup \text{vars}(\Gamma'_2) \) such that \( \Gamma(x) = \text{LL} \land \mu'(x) \in N \), \( \mu(x) = \mu'(x) \). That is to say that for all \( x \in \text{dom}(\alpha') \), \( \mu(x) = \alpha'(x) \).

In addition, it is clear from the definition of \( \sigma_{\ell} \) that \( \{ x \in \text{vars}(\Gamma'_1) \cup \text{vars}(\Gamma'_2) \mid \Gamma(x) = \text{LL} \} = \{ x \in \text{vars}(\Gamma'_1) \cup \text{vars}(\Gamma'_2) \mid \Gamma'(x) = \text{LL} \} \).
Hence
\[
\text{dom}(\alpha) = \{ x \in \text{vars}(M_1) \cup \text{vars}(M_2) \mid \overrightarrow{T}(x) = \text{LL} \land \mu(x) \in \mathcal{N} \} = \{ x \in \text{vars}(M'_1) \cup \text{vars}(M'_2) \mid \overrightarrow{T}(x) = \text{LL} \land \mu(x) \in \mathcal{N} \} \supseteq \{ x \in \text{vars}(M'_1) \cup \text{vars}(M'_2) \mid \overrightarrow{T}(x) = \text{LL} \land \mu'(x) \in \mathcal{N} \} = \{ x \in \text{vars}(M'_1) \cup \text{vars}(M'_2) \mid \overrightarrow{T}(x) = \text{LL} \land \mu'(x) \in \mathcal{N} \} = \text{dom}(\alpha') \).
\]

Therefore, \( \forall x \in \text{dom}(\alpha') \), \( x \in \text{dom}(\alpha) \land \alpha'(x) = \alpha(x) \). Thus there exists \( \alpha'' \) such that \( \alpha = \alpha' \alpha'' \).

Since we already have \( N_1 \alpha' = N_2 \alpha' \), this implies that \( N_1 \alpha = N_2 \alpha \), which concludes the proof that \( \text{step} 4 \) (\( \check{\alpha} \)) holds. Hence, check\_const([{(c, \Gamma') \})] = true.

\[ \square \]

We can now prove the following theorem:

**Theorem B.55.** Let \( C \) and \( C' \) be two constraint sets without any common variable.

\[
\text{check\_const}([ C ]_1 \cup \{ C \} \cup \{ C' \} \cup \{ \Gamma \} \} = true \Rightarrow \forall n. \ [ C' ]_1^n \cup \{ \cup_{1 \leq i \leq n}[ C ]_i^n \} \text{ is consistent.}
\]

**Proof.** Assume \( \text{check\_const}([ C ]_1 \cup \{ C \} \cup \{ C' \} \cup \{ \Gamma \} \} = true \). Let \( n > 0 \). Let us show that \( [ C' ]_1^n \cup \{ \cup_{1 \leq i \leq n}[ C ]_i^n \} \text{ is consistent.} \)

By Theorem B.52, it suffices to show that \( \text{check\_const}([ C' ]_1^n \cup \{ \cup_{1 \leq i \leq n}[ C ]_i^n \} = true. \)

By Theorem B.53, it suffices to show that \( \text{check\_const}([ C' ]_1^n \cup \{ \cup_{1 \leq i \leq n}[ C ]_i^n \} = true. \)

By assumption, we know that \( \text{check\_const}([ C ]_1 \cup \{ C \} \cup \{ C' \} \cup \{ \Gamma \} = true. \)

That is to say, for each \((c_1, \Gamma_1) \in C, (c_2, \Gamma_2) \in C, (c_3, \Gamma_3) \in C', \) if \( c' = \{ c_1 \} \cup \{ c_2 \} \cup \{ c_3 \} \) and \( \Gamma' = \{ \Gamma_1 \} \cup \{ \Gamma_2 \} \cup \{ \Gamma_3 \} \), check\_const(([c', \Gamma']) = true.

Thus, by Lemma B.54, for all \((c_1, \Gamma_1) \in C, (c_2, \Gamma_2) \in C, (c_3, \Gamma_3) \in C', \) if \( c' = \{ c_1 \} \cup \{ c_2 \} \cup \{ c_3 \} \) and \( \Gamma' = \{ \Gamma_1 \} \cup \{ \Gamma_2 \} \cup \{ \Gamma_3 \} \), check\_const(([c', \Gamma']) = true.

That is to say, check\_const([ C ]_1^n \cup \{ C \} \cup \{ C' \} \cup \{ \Gamma \} = true, which concludes the proof.

\[ \square \]